Juman++: A Morphological Analysis Toolkit for Scriptio Continua

Arseny Tolmachev
Graduate School
of Informatics
Kyoto University
Yoshida-honmachi, Sakyo-ku
Kyoto, 606-8501, Japan
arseny@kotonoha.ws

Daisuke Kawahara
Graduate School
of Informatics
Kyoto University
Yoshida-honmachi, Sakyo-ku
Kyoto, 606-8501, Japan
dk@i.kyoto-u.ac.jp

Sadao Kurohashi
Graduate School
of Informatics
Kyoto University
Yoshida-honmachi, Sakyo-ku
Kyoto, 606-8501, Japan
kuro@i.kyoto-u.ac.jp

Abstract

We present a three-part toolkit for developing morphological analyzers for languages without natural word boundaries. The first part is a lattice-based morphological analysis library that uses a combination of linear and recurrent neural net language models for analysis. The other parts are a tool for exposing problems in the trained model and a partial annotation tool. Our morphological analyzer for Japanese achieves new SOTA on Jumandic-based corpora while being 250 times faster than the previous one. We also perform a small experiment and quantitive analysis of using our toolkit.

1 Introduction

Processing *scriptio continua* natural languages, or languages without natural word boundaries, like space in English, frequently requires performing tokenization into morphemes in the natural language processing pipeline. Usually, tokenization is done together with part of speech (POS) tagging, pronunciation estimation or other subtasks. This process is usually referred to as morphological analysis.

A morphological analyzer is useful from a practical point of view only if it is fast as well as highly accurate in its analysis. Because morphological analysis is very important to languages without word boundaries (like Japanese), there already exist many approaches and tools for performing it. Morphological analyzers usually use two kinds of resources: a dictionary which defines possible morphemes and an annotated corpus which is used to train an analysis model for connecting morphemes together.

Modern morphological analyzers achieve high accuracy (a segmentation F1 score of $>$ .99 for Japanese) on established domains like newspaper. However, when using them on out-of-domain or open domain data (like web texts) the accuracy decreases, and it is difficult to improve that accuracy without creating costly annotations by trained experts.

The Juman++ Japanese morphological analyzer (Morita et al. 2015, referred to also as V1), which uses a combination of a linear model and a neural network-based language model (RNNLM) to compute a semantic plausibility of a segmentation. Juman++ has achieved state-of-the-art analysis accuracy on Jumandic (the JUMAN dictionary and segmentation standard (Kurohashi and Kawahara, 2012)) based corpora, and drastically reduced the number of intolerable analysis errors. Unfortunately, its execution speed was extremely slow and this has limited the practical usage of Juman++.

We have developed a morphological analysis toolkit consisting of three components: a morphological analyzer and two support tools which help with the development of analysis models. The analyzer is a complete rewrite of core ideas of Juman++, released as Juman++ V2[1] (Tolmachev and Kurohashi, 2018). Our reimplementation is **more than 250 times faster** than V1, reaching the speed of traditional analyzers, at the same time achieving better accuracy than V1.

We have also developed a tool which uses *raw, unannotated data* and gives an insight into problematic dictionary and grammar points, together with finding sentences which contain situations not present in training data by exploiting differences in the analysis that arise from using different beam search configurations. It is bundled with V2. Also, we have developed a partial annotation tool which makes it easy to review problematic sentences and create partial annotation data for improving the analysis[2]. To the best of our knowl-

[1] https://github.com/ku-nlp/jumanpp
[2] https://github.com/eiennohito/nlp-tools-demo

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (System Demonstrations), pages 54–59
Brussels, Belgium, October 31–November 4, 2018. ©2018 Association for Computational Linguistics

edge, a similar set of tools has never been developed before. Juman++ V2 and the development tools are language and segmentation standard independent and are released under the permissive Apache 2 open source license.

2 Related Work

KyTea (Neubig et al., 2011) is a similar tool that can perform morphological analysis for languages with the continuous script. It can also be trained using partial annotation data and output point-wise confidence scores for the analysis result which were used for creating partially annotated data in an active learning scenario. Still, by using a point-wise approach and estimating auxiliary tags (like POS) after computing segmentation, KyTea trades off accuracy for simplicity. Juman++ is faster, has better accuracy, does tag estimation jointly with segmentation, uses an online learning approach and can use longer contexts in forms of RNNLM and trigram features.

3 Juman++ Morphological Analyzer V2

Juman++ V2 is implemented with modern C++, with the intention to be used not only as a program, but also as an embeddable library, usable in a multi-threaded environment. Additionally, V2 is not hardwired to a particular dictionary and can use partially annotated data for training.

Juman++ is a lattice-based analyzer. For an input sentence, it looks up all possible morphemes from a dictionary, makes a lattice from them, and assigns a score s to each path going through the lattice. The path with the highest score is considered to be the analysis result. The score s consists of two components: a feature-based linear model score s^l and an RNNLM score s^{RNN}, which are combined as

$$s = s^l + \alpha(s^{\text{RNN}} + \beta),$$

where α and β are scale and bias hyperparameters respectively. The RNNLM score is a log-probability score from the language model.

The linear score is defined as

$$s^l = \sum_{t \in p} \mathbf{f}(t, p)\mathbf{w},$$

where $\mathbf{f}(t, p)$ is an indicator vector which contains features, extracted for a node t in a path p, and $\mathbf{w}$ is a model weight vector. Features use dictionary information of the node t and at most two previous nodes and their surface surroundings. Weights are learned using the Soft Confidence Weighted algorithm (Wang et al., 2016).

In practice, it is intractable to compute scores for all paths through the lattice. Instead, we use beam search. Additionally, because the RNNLM is computationally-heavy, we compute the RNN score only for the paths which remain in the beam of the special end-of-string token. The overall design is to cut off improbable analysis results with a simple model and then re-rank by RNNLM.

3.1 Search Space Trimming

One of important optimizations of V2 is the reduction of search space by changing the beam search operation mode. Because the Juman++ linear model uses up to 3-gram features, the beam searching procedure has to deal with combinatorial explosion caused by higher order n-gram features. V1 uses *local beams* of width j, meaning that *each lattice node* keeps j incoming paths with top scores. The rest of the paths are discarded. The problem is that paths are discarded after evaluating their scores and the number of evaluations is still large. Most of the sentences have several boundaries which have 20-30 both left and right nodes. Because almost all these paths are useless, we do not want to consider them in the analysis at all.

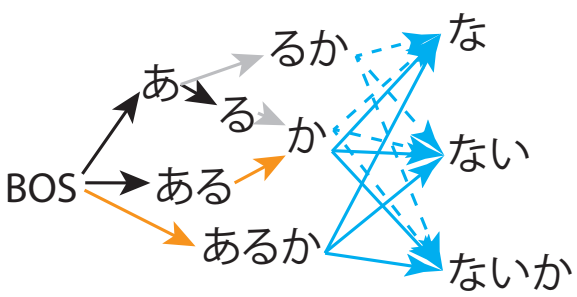
The first improvement is to use only paths ending on left nodes with top k scores instead of using all left paths. Connections for the remaining paths are not considered. We refer to this process as *left global beam*. The application of the left global beam is shown in Figure 1a.

The second improvement is the *right global beam*, displayed in Figure 1b. As the first substep, we use top $l(\leq k)$ paths ending at left nodes to compute scores of right nodes. After that, we evaluate the connections from the remaining $k - l$ left paths to the top-scored m right nodes. The rest of connections are not considered.

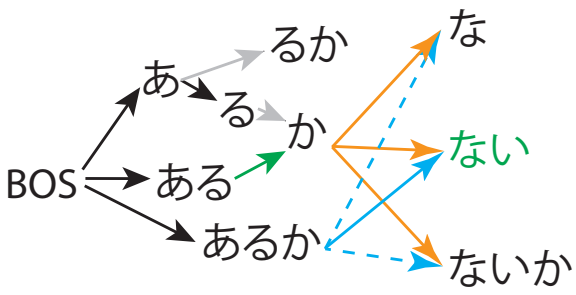
We call search using local beams as *full beam*. In comparison to that, a combination of left and right global beams is referred to as *trimmed beam* because it uses significantly smaller search space.

3.2 Partially Annotated Data

Juman++ V2 supports both training with partially annotated data and soft-constrainted analysis using partially annotated data as constraints.



(a) Left global beam. $k = 2$. Top left paths are displayed in orange. Remaining left paths are displayed in solid gray. Non-considered paths are displayed as dashed blue arrows. Considered paths are solid blue.



(b) Right global beam. $l = m = 1$. A top left path and a top right node are displayed in green. Orange paths via the boundary were used for scoring right nodes. Solid blue path via the boundary connects the remaining $k - l$ top left paths the top right nodes. Dashed blue paths are not considered.

Figure 1: Trimming the search space using the global beams

There exist three types of annotations: boundary (break here), non-boundary (no break here) and *word*. A word annotation means that a sequence of characters must not contain boundaries inside it; additionally, the word can have tags attached to it, meaning that the analysis result also must have these tags.

For training, we need to provide sets of correct and incorrect features for the training algorithm. A feature computed from the correct path would be correct, and a feature containing any lattice node that is not contained in the correct path would be incorrect.

When using the fully annotated data, there exist only a single correct path. However, for partial annotated data, there could exist multiple allowed paths through the lattice. Because of this, when training using the partially annotated data, we use several highly-scored candidates which do not violate annotation requirements instead of the only correct path as in the case of fully annotated data.

3.3 Performance Comparison

We compare both speed and accuracy of five different Japanese morphological analyzers: JUMAN[3], MeCab[4], KyTea, Juman++ (V1) and Juman++ (V2). For both versions of Juman++ we report results both using and not using RNNLM (noRNN).

Analysis speed We used a desktop computer with Intel i7-6850K CPU, 64GB of RAM and Ubuntu 16.04 Linux for analysis speed comparison. The models were trained from scratch using the same Juman++ dictionary, the Kyoto University[5] (KU) and KWDLC[6] corpora for all morphological analyzers except JUMAN, which is not trainable. For KyTea we also report the throughput using the Unidic-based models, which are available for download from the KyTea website. A Jumandic-based model for KyTea was learned using default parameters. V1 uses the full beam of width $j = 5$. V2 uses trimmed beam with parameters $j = 5$, $k = 6$, $l = 1$, $m = 5$. All analyzers were using only a single thread.

Analyzer	Speed (sents/s)	Ratio
JUMAN	8,802	1.00
MeCab	52,410	0.17
KyTea (Jumandic)	4,892	1.79
KyTea (Unidic)	1,995	4.41
V1 noRNN	27	328.82
V1 RNN	16	535.72
V2 noRNN	7,422	1.18
V2 RNN	4,803	1.83

Table 1: Morphological analysis speed comparison

Table 1 shows the analysis speed of the considered morphological analyzers and speed ratio as compared to JUMAN. The speed was measured by analyzing 50k sentences from a web corpus. Each analyzer was launched five times and the median time was used for computing the analysis speed. V2 noRNN is only 20% slower than JUMAN while having a considerably complex model. V2

[3] http://nlp.ist.i.kyoto-u.ac.jp/EN/index.php?JUMAN

[4] http://taku910.github.io/mecab/

[5] http://nlp.ist.i.kyoto-u.ac.jp/EN/index.php?Kyoto University Text Corpus

[6] http://nlp.ist.i.kyoto-u.ac.jp/EN/index.php?KWDLC

Analyzer	KU Seg	KU +Pos	KWDLC Seg	KWDLC +Pos
JUMAN	98.41	97.20	98.10	97.01
KyTea	99.12	98.16	98.00	96.75
MeCab	99.14	98.58	98.28	97.61
V1 noRNN	98.94	98.42	97.66	96.95
V1 RNN	99.38	98.95	98.41	97.87
V2 noRNN	99.44	98.98	98.44	97.79
V2 RNN	**99.51**	**99.05**	**98.67**	**98.02**

Table 2: F1 scores of morphological analyzers on Jumandic-based corpora. Seg is segmentation; +Pos is correctly guessing the POS-tags after segmentation.

RNN has 1.8 times the execution speed of JUMAN and is more than 250 times faster than V1.

Accuracy Table 2 shows F1 scores for both the KU and KWDLC corpora. A concatenation of training sections of both corpora was used to train a combined model; the reported scores are for the test sections. MeCab and V2 have hyperparameters optimized using Spearmint (Snoek et al., 2012).

V2 RNN achieves a higher F1 score than the previous SOTA of V1 RNN. Even the scores of V2 noRNN are higher in some cases than those of V1 RNN. Note that the scores of V1 noRNN are one of the lowest, and thus we hypothesize that the number of training iterations of the V1 linear model was not sufficient. However, it was difficult to increase it because of very slow analysis speed.

With V2, we could find an optimal number of iterations for learning the linear model with the best accuracy. The other reason for the improved accuracy for V2 is that it uses surface character and character type features.

4 Beam Search Diffs as Active Learning

We compared the accuracy of the trimmed beam search to the full configuration in the in-domain setting. For this experiment, we trained the model using 1 pass of full beam search followed by 4 passes of trimmed beam search for the optimal number of iterations for different trimmed beam parameters. Figure 2 shows the F1 score average on 10-fold cross-validation over the KU corpus. It can be seen that the accuracy of the models does not fall even if using very small global beam sizes. Nevertheless, we noticed that there exist sentences

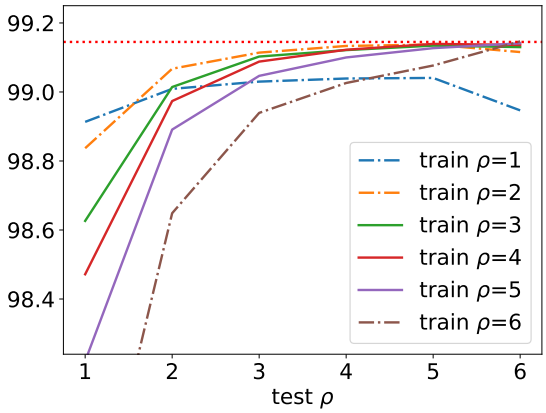


Figure 2: Cross-validation test F1 score average on the KU corpus for POS tags when using different global beam parameters $k = m = \rho$, $l = 1$. Red dotted line at the top is F1 score of a model without global beams.

when the full and trimmed search configurations do not agree on the top-scoring path when analyzing random web sentences and produce diffs.

To get a better picture, we analyzed a large number of sentences from a raw Japanese corpus, crawled from the web. On average, 0.38% $(1969/510595)$ of sentences were diffs, a relatively small number. In contrary to our expectations, the full beam analysis was correct only in around 50% of the cases. The rest of sentences had both of analysis variants incorrect either because the dictionary did not support the language phenomena (20%) or lack of coverage by the training data (10%); the trimmed beam analysis was the correct one (10%); and other situations that were difficult to decide or impossible to analyze correctly like typos.

Based on these insights, we believe that the diffs can be treated as a result of selection by the Query-by-Committee active learning algorithm with two committee members (Settles and Craven, 2008; Seung et al., 1992). We think that diffs actually reveal problems not only with a training corpus (the goal of active learning) but also with an analysis dictionary as well. Note that we are using exactly the same model in both beam modes and the only difference is beam configuration. So, a diff can form only if trimmed beam ranking was not learned correctly either because the model capacity was not enough, features were not strong enough, or the situation was not present in the training data.

When the beam size at training is very small,

the model does not learn to rank for the trimmed case well enough, lowering the accuracy on larger beam sizes. Increasing the beam size above three makes the trimmed beam score indistinguishable with the full beam. On the other hand, the model loses accuracy on smaller beam sizes at test time, because the trimmed ranking fails in those situations. Thus we believe that the model capacity and the feature set should be enough to capture the ranking and the diffs are caused mostly by the lack of training data. The fact that the diffs include situations when it is *impossible* to produce the correct analysis at all, namely lack of dictionary words and typos, confirms our belief.

On the other hand, we also believe that the corrections of places pointed by diffs are not going to significantly improve benchmark scores exactly because of the same reason. The benchmark corpus will usually be relatively in-domain and not contain dictionary or grammar problems, because they would be fixed when creating the corpus. So we hope that this method would be useful to improve *real world* morphological analysis accuracy and for domain adaptation.

5 Partial Annotation Tool

Because the diffs must be reviewed by human annotators to be useful as training data, we have developed a partial annotation tool based on a simple idea: to allow annotators to select a correct analysis from two candidates. The output of the tool can be used as partially annotated training data.

5.1 Tool Description

The tool is a web application, implemented in Scala. As an input, it uses sentences with some parts being diffs, which are produced by a tool bundled with the Juman++ V2 distribution. For each sentence, annotators are asked to select a correct variant, correct an analysis if both are not correct, or report if it is impossible to select a correct analysis or there is a problem with the sentence itself.

A sentence diff view is shown in Figure 3. The annotation targets are diffs: we want annotators to choose a correct analysis from the possible variants, which are displayed side-by-side. A sentence can contain more than one annotation target in general and each variant can contain more than one morpheme. Non-diff parts form contexts and are displayed in grey. The tool shows diffs in an

Figure 3: Annotation tool: diff view. UI explanations are in blue. Variant 1 is selected.

unspecified order. In the shown example, the left variant is selected as the correct one. The area in the middle of the target section contains a button which activates an interactive analysis mode and buttons for assigning special tags in the case when the mistake in the target part is due to a typo or a phenomenon that we do not want to support in the automatic analysis, such as netspeak.

In the case when both analyses are incorrect, an annotator can use the interactive analysis mode, shown in Figure 4, which performs constrained morphological analysis. Constraint nodes are shown in blue. A full beam analysis becomes initial constraints.

The interactive analysis mode uses lattice information from the Juman++ to provide morpheme candidates for constraints. It is possible to show either all morphemes containing a focused character or morphemes spanning exactly selected characters. The corrected analysis replaces the closest diff variant which was not selected by any annotator yet, or creates a new variant if replacing is not possible. Finally, if there are no variants, annotators can select a character span and report that it is impossible to choose a correct morpheme in this sentence for that span.

5.2 Annotation Experiment

We performed a small scale annotation experiment using the developed annotation tool. For the experiment, we trained a model on the concatenation of the training and test data from both benchmark corpora and the copy of data augmented by removing “ga”, “ha” and “wo” case markers, which are often omitted in the spoken language so that the model would be more robust to case marker omission. Using this model we have collected sentences which contained diffs, as described in Sec-

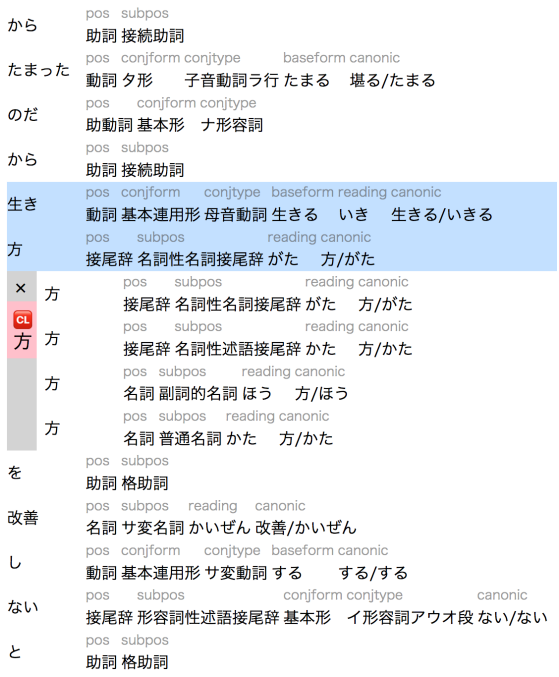


Figure 4: Annotation tool: interactive analysis. Constraint nodes background is blue. Other analysis variants for the last constraint node are displayed after the gray indent. The "CL" button removes a constraint.

tion 4. We asked two annotators to work for three hours each.

During the allocated time, the two annotators could review 111 and 112 sentences each. Both the annotators started annotating rather slowly, while gradually increasing their annotation speed. An inter-annotator agreement was 0.819. The annotators have selected at least one of analysis candidates in most (82%) cases; the rest were special tags.

We also checked sentences where the annotators did not agree on the correct analysis, but both the annotations were analysis results (9 in total). Each of them was difficult to decide even for us. We believe that the relatively large number of sentences which caused annotators to spend a long time on annotation is caused by the fact that the diff extractor selects difficult cases.

6 Conclusion and Future Work

We present Juman++ V2: a morphological analysis and tokenization toolkit for languages with the continuous script. Our current implementation focuses on Japanese and the Jumandic-based segmentation standard, but the core library is language independent. Juman++ V2 achieves a new state-of-the-art accuracy for both the Kyoto University and KWDLC corpora while drastically reducing the analysis time compared to Juman++ V1. Juman++ V2 can be used as a library and can use both fully and partially annotated data for training.

We also release a morphological analysis developer tool which reveals problematic places of a dictionary and segmentation standard using only unannotated data by comparing the analysis results in the two different beam search configurations. Also, we release the partial annotation tool for easily viewing and fixing these problems.

We plan to create a Unidic version of Juman++ V2 and use it to annotate readings of the Jumandic-based corpora, which are currently rather arbitrary, enabling the Jumandic-based models to estimate correct readings as well. We also plan to continue collecting partially annotated data and release a partially annotated web corpus.

References

Sadao Kurohashi and Daisuke Kawahara. 2012. Japanese morphological analysis system juman 7.0 users manual. *Kyoto University*.

Hajime Morita, Daisuke Kawahara, and Sadao Kurohashi. 2015. Morphological analysis for unsegmented languages using recurrent neural network language model. In *EMNLP*, pages 2292–2297.

Graham Neubig, Yosuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust, adaptable japanese morphological analysis. In *ACL*, pages 529–533.

Burr Settles and Mark Craven. 2008. An analysis of active learning strategies for sequence labeling tasks. In *EMNLP*, pages 1070–1079, Honolulu, Hawaii. Association for Computational Linguistics.

H. S. Seung, M. Opper, and H. Sompolinsky. 1992. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 287–294, New York, NY, USA. ACM.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *NIPS*, pages 2951–2959.

Arseny Tolmachev and Sadao Kurohashi. 2018. Juman++ v2: A practical and modern morphological analyzer. In *ANLP*, pages 917–920, Okayama, Japan.

Jialei Wang, Peilin Zhao, and Steven CH Hoi. 2016. Soft confidence-weighted learning. *ACM TIST*, 8(1):15.