Enhancing Morphological Analysis and Example Sentence Extraction for Japanese Language Learning

Tolmachev Arseny

2022

Contents

Abstract

Language learning is increasingly important in modern globalized world. The ways of teaching languages and making the teaching materials are also changing. Natural Language Processing as a discipline provide tools for automated handling of human languages by a computer. The tools and methods from natural language processing are useful for automating the needs of language learning. This thesis presents a foundation for Japanese word segmentation and part-of-speech tagging and an example of educational application: automated examples of high-quality example sentences.

Japanese language has no spaces and even the task of segmenting the sentence into words becomes non-trivial. This task is crucial for the Japanese Natural Processing pipeline, is often performed together with part-of-speech tagging and is called Morphological Analysis. High accuracy of analysis, especially of part-of-speech tagging of grammar makes the analyzer very useful for educational applications. Still, for the morphological analysis to be practical, it should not just have high accuracy, but also be fast as well. It also helps if the analyzer is compact.

Learning vocabulary is a crucial part of language learning and require students to perform a large amount of homework. Flashcard approaches are often used to increase the efficiency of vocabulary learning. Still, the flashcards often do not contain the context, not teaching the student how to use the world correctly.

We describe a Juman++ V2 improvement of morphological analyzer which achieves very high analysis accuracy (> 99.5 segmentation F1 score on newspaper domain), while being as fast as classical analyzers. Our description consists of algorithmic improvements and implementation details which allow Juman++ to efficiently utilize modern out-of-order CPUs. Juman++ supports training with partially annotated data and provides active learning tooling for selecting sentences from a non-annotated corpus which are difficult to analyze.

We also describe an experiment in implementing a fully neural model for morphological analysis which does not model the dictionary data explicitly. Instead, it learns the dictionary data implicitly, together with segmentation and part-of-speech tagging decisions from large-scale training data, analyzed by a bootstrapping classical analyzer. The fully neural model achieves comparable or superior accuracy to

the bootstrapping analyzer, while having significantly smaller model size (20 times less than the bootstrap one).

Finally, we describe an automatic high-quality example extraction system. The system consists of a distributed search engine, which selects a large number of example sentence candidates with rich syntactic structure around the target word and a filtering step which selects a small set of sentences, which are independently good and non-similar to each other. We performed a blind experiment with Japanese language learners and a teacher of Japanese language who have preferred our proposed system to two baselines.

Acknowledgments

Writing this thesis would be impossible without the enormous support I have received. First, I am grateful for the financial support from the MEXT Scholarship by the Government of Japan and the Ministry of Education, Culture, Sports, Science, and Technology. Without the scholarship, I would never be able to come to Japan, enroll at Kyoto University, and perform this research. Next, I would like to thank academic advisor Prof. Sadao Kurohashi for the support and advice during the long course of eight years of the Master's program, Ph.D. program, and after the Ph.D. period of finishing the thesis and the remaining publications. I would also like to thank Kyoto University Design School, its founder, and coordinator Tooru Ishida, for allowing me to participate in the project. Participation in the Kyoto University Design School had given me many experiences and allowed me to expand the scope of the materials covered in this thesis. I would also like to thank Prof. Kusumi Takashi and Prof. Kawahara Tatsuya who had accepted roles as the examinators and provided very useful feedback.

This thesis would not be possible without the help and advice of Morita Hajime. He is the author of the original Juman++ morphological analyzer and his advice supported most of the work contained in this thesis. Prof. Daisuke Kawahara had also a lot of input into the work related to morphological analysis. Yugo Murawaki gave plentiful useful advice and edits for the work on fully-neural morphological analysis. Most of the textual content of the thesis became actually readable text instead of remaining primordial English soup because of the input of these people.

I would like to thank the Artificial Intelligence Laboratory of Fujitsu Laboratories, and its heads Okamoto Seiji and Anai Hirokazu, Maruhashi Koji, and other members of the XAI team for the support and permission to work on this thesis while working in Fujitsu Laboratories. I hope that I did not use too much company time on this thesis. I would also like to thank Takaoka Kazuma and Uchida Yoshitaka of Works Applications Enterprise, Tokushima NLP Laboratory for giving me time to finish the work on this thesis.

Time in Kyoto University was fun and productive thanks to Kurohashi laboratory members and staff: Kishimoto Yudai, Sakaguchi Tomohiro, Kiyomaru Hirokazu, Raj Dabre, John Richardson, Yin Joh (Leslie) Huang, Kurita Shuhei, Shibata Tomohide,

Tariq Alkhadi and Eugeni Puzikov (listed in order without any specific meaning). I would like to thank Yuko Ashihara and Naho Yoshitoshi for their administrative support during being the student in the Kurohashi laboratory.

The evaluation experiment for example extraction was made possible by volunteer evaluators, thank you very much for your opinion on example sentences.

I would like to thank all people I met while studying the Japanese language and during my first trip to Japan in 2008 which without any doubt was a founding stone for the decision to perform this type of research in the first place.

Finally, I would like to thank my family and friends for their unconditional and unending love and support. I love you too!

1 Introduction

1.1 Thesis Overview

With the current globalized world, language learning is very important. Most people know more than one language and compulsory education in the majority of countries includes a foreign language. In Europe, almost all students learn at least one foreign language[1]. In many countries, foreign language learning is included in compulsory primary education. Language learning is supported by teachers and teaching materials, both interactive and non-interactive. Interactive teaching materials build on various foundations and natural language processing (NLP) techniques frequently appear as a recent development.

Language learning is a multifaceted and complex activity. Vocabulary learning is probably a cornerstone of language learning, and it can not be fully performed purely in the classroom setting. The students must practice on their own in addition to the classroom to acquire large vocabularies. Based on these characteristics, vocabulary learning is a perfect target for learning optimization techniques like flashcards. One main theme of this thesis is the improvements for the flashcards with the help of example sentences and the application of NLP technology to automate high-quality example extraction.

We would like to focus on improving the learning of the Japanese language in particular. One of defining traits of the Japanese language is the lack of natural word segmentation — spaces. This makes NLP technology for Japanese different from languages with spaces. The raw text first must be segmented into words. Such segmentation can be defined manually by human annotators or defined automatically by optimizing some, often information-theoretic, metric. In the latter case, the segmentation usually diverges with human-defined one but can achieve better accuracy in downstream applications. Language learning is, on the other hand, word-based and it is crucial that the words are defined by humans.

Automated word segmentation is often accompanied by part-of-speech (POS) tagging in Japanese and the whole joint process is called Morphological Analysis.

[1] https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Foreign_language_learning_statistics

外国人参政権が認められていない
↓
外国|人|参政|権|が|認め|られて|い|ない

Figure 1.1: An example of Japanese sentence segmentation into morphemes

The part of speech information is very useful for educational applications. Because the morphological analysis is the first step of the Japanese NLP pipeline, its accuracy greatly affects the accuracy, and by extension the overall usability of the whole system, so it is crucial to make that accuracy as high as possible. On the other hand, applications can require a large number of text data, so the processing speed of the morphological analysis is also very important. Another main theme of this thesis is to propose approaches to make the Japanese morphological analysis accurate, fast, and having compact models.

The structure of the thesis is the following. This chapter provides an introduction to all topics covered by the thesis. While the motivation of this thesis lies in Japanese language learning, the Morphological Analysis lies earlier in the NLP pipeline. Section 1.2 gives the introduction to the problem of Japanese Morphological Analysis with the detailed discussion following in Part 1.5. Section 1.3 from language learning, define the motivation and problem of high-quality example extraction and outline the proposed solution, discussed in detail in Part 3.5. Section 1.4 defines the contributions of this thesis with the supported publications. Section 1.5 discusses the considerations of our work from the design standpoint.

1.2 Japanese Morphological Analysis

The Japanese language has a continous script: there are no delimiting spaces. However, the applications usually expect the text to be segmented into tokens. An example of segmentation is shown in Figure 1.1. In most cases, there exist several different ways to segment a given sentence. Still, most of these ways produce nonsense segmentation from the language perspective.

There are two general types of approaches defining what these tokens are: supervised and unsupervised. In the supervised approaches, the tokens are usually defined to be morphemes, defined by segmentation standards and related human-annotated corpora. In unsupervised approaches, there is no such apriori knowledge for algorithms. The segmentation is decided mechanically by the algorithm, often to minimize some information-theoretic measure. Our focus in this work lies in supervised segmentation.

1.2.1 Morphemes and Morphological Analysis Approach Types

Generally speaking, the definition of morpheme in Japanese morphological analysis differs from the definition in European languages, and the term *word* is ill-defined for Japanese (Murawaki 2019). In this thesis, we use the terms *morpheme* and *word* interchangeably as a unit defined by a segmentation standard.

For Japanese, there are several popular standards like IPADic, Jumandic, and UniDic. Because the way to segment text into morphemes is defined by humans, it matches with the human expectation of what a word is. When an underlying application will be interacting with users on a word level, tokens being human-compatible is an essential requirement for segmentation.

Most Japanese analyzers use segmentation dictionaries that define corpus segmentation standards. They usually have rich part-of-speech information attached and are human-curated. One focus of segmentation dictionaries is to be consistent: it should be possible to segment a sentence using the dictionary entries only in a single correct way. Such dictionaries are often maintained together with annotated corpora. On the other hand, Chinese-focused systems do not put much focus on dictionaries. The dictionary is created by taking every unique word from the corpus (Kruengkrai, Uchimoto, Jun'ichi Kazama, Y. Wang, Torisawa, and Isahara 2009; Zheng, Hanyang Chen, and T. Xu 2013a), in contrast to Japanese where the corpora are usually accompanied by a dictionary. Still, almost all approaches use rich feature templates or additional resources such as pre-trained character n-gram or word embeddings, which increase the model size.

Unsupervised approaches (Mochihashi, Yamada, and Ueda 2009; Uchiumi, Tsukahara, and Mochihashi 2015; Zhikov, Takamura, and Okumura 2010), on the other hand, decide segmentation without external definitions. They do not require annotated corpora to work, but the resulting segmentation does not match with the human expectation of a word. Nevertheless, unsupervised segmentation achieved better accuracy on underlying tasks where user interaction is on a sentence or text-level, like machine translation (Kudo and Richardson 2018).

There exist two main lines of approaches to supervised Japanese morphological analysis: pointwise and search-based. Pointwise approaches make a segmentation decision for each character, usually based on the information from its surroundings. Search-based approaches look for a maximum scored interpretation in some structure over the input sentence.

1.2.2 Specifics of Educational Applications

Japanese segmentation standards and corpora are developed with the focus on monolingual NLP pipelines and have different sets of annotations. POS tag hierarchies are also different. For educational applications, those annotations can make the annotation standard easier or harder to use.

Namely, IPADic does not handle the canonicalization of orthographic variations which are very common in Japanese. Unidic and Jumandic, on the other hand, include canonicalization as the dictionary metadata. For example, "あんまり" and "あまり" are orthographic variants as a single word. Orthographic variants can be written in different scripts (でかい and デカい) and can have different kanji (檜 and 桧). Verbs can also have non-standard *okurigana* (着ける and 着る). The combination of different reasons for orthographic variants results in a large number of possibilities.

In the design of the POS tag hierarchy, Jumandic and Unidic follow different philosophies. Unidic encodes ambiguity that is not resolvable at the morpheme level in POS tags, leaving it unresolved in the corpus as well. Jumandic resolves this ambiguity, both in POS tags and in the corpus, making ambiguity resolution the task of the morphological analyzer. We argue that while the accuracy of morphological analyzers is not perfect, it is enough for the practical usage of automatically inferred POS tags.

Based on this comparison, we believe Jumandic segmentation and POS standard to be the most acceptable for the example extraction. There are some problems, though. Jumandic does not segment copula from the stem of na-adjectives, effectively treating na-adjectives conjugatable similarly to other parts of speech. No other resource follows this decision, so when using multiple resources one has to perform the above-described segmentation in preprocessing. Jumandic-based corpora, also, do not provide *correct* pronunciation annotations, making analyzers trained on that corpora provide inconsistent pronunciation estimation. Still, for the basic analysis, we prefer it to IPADic and Unidic standards.

1.2.3 Pointwise Approaches

Pointwise approaches make a segmentation decision independently for each position. They can be seen as a sequence tagging task. Such approaches are more popular for the Chinese.

KyTea (Neubig, Nakata, and Mori 2011) is an example of this approach in Japanese. It makes a binary decision for each character: whether to insert a boundary before it or not. It also can be seen as a sequence tagging with {B, I} tagset – whether a

character starts a new token or not. POS tagging is done after inferring segmentation. The decisions are made by feature-based approaches, using characters, character n-grams, character type information, and dictionary information as features. KyTea can use word features obtained from a dictionary. It checks whether the character sequence before and after the current character forms a word from the dictionary. It also checks whether the current word is inside a word. Neural networks were shown to be useful for the Japanese in this paradigm as well (Kitagawa and Komachi 2018). They use character embeddings, character type embeddings, character n-gram embeddings, and tricks to incorporate dictionary information into the model. However, they only consider the segmentation task and do not do any tagging.

Tolmachev, D. Kawahara, and Kurohashi (2019) have used a bootstrapping approach to create a purely neural-network pointwise analyzer that uses only character unigrams as input data. The model seems to learn the dictionary information implicitly from a large-scale automatically annotated corpus. They have achieved slightly better accuracy for both segmentation and POS tagging than the underlying analyzer used for tagging a large corpus while having a small model size. In such an approach it is, however, difficult to make the analyzer recognize new words. The word should be added to the underlying analyzer, and then the whole huge corpus needs to be retagged before re-learning the neural network model.

Many studies on Chinese adopt the pointwise approach. Often, the segmentation task is reformulated as sequence tagging (Xue 2003) with {B, I, E, S} tagset. Peng, Feng, and McCallum (2004) showed that CRFs help further in this task. This tactic was followed by many subsequent feature-based approaches (Tseng, P. Chang, Andrew, Jurafsky, and Christopher D Manning 2005; L. Zhang, H. Wang, X. Sun, and Mansur 2013; H. Zhao, C.-N. Huang, Mu Li, and Lu 2006), using character n-gram, character type and word features.

Neural networks were applied to this paradigm as well. Zheng, Hanyang Chen, and T. Xu (2013b) used a feed-forward network on character and categorical features that were shown to be useful for computing a segmentation score from a fixed window. Qi, Das, Collobert, and Weston (2014) used a similar architecture. They predicted not only segmentation but POS tags and performed named entity recognition as well. The character representation was pretrained on a language modeling task. Shao, Hardmeier, Tiedemann, and Nivre (2017) used a bidirectional recurrent network with GRU cells followed by a CRF layer for joint segmentation and POS tagging. They used pretrained character n-gram embeddings together with sub-character level information extracted by CNNs as features. Using a dictionary with NN is also popular (J. Liu, F. Wu, C. Wu, Y. Huang, and Xie 2018; Q. Zhang, X. Liu, and J. Fu 2018).

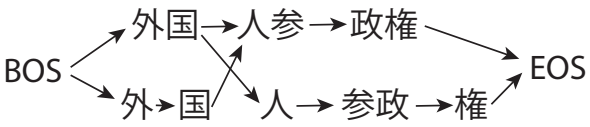


Figure 1.2: Lattice for input sentence 外国人参政権

1.2.4 Search-based Approaches

Search-based approaches induce a structure over a sentence and perform a search over it. A most frequently used structure is a lattice (Jiang, Mi, and Q. Liu 2008; Christopher D. Manning and Schütze 1999, p. 327), see also Figure 1.2. **Lattice** is a directed acyclic graph that contains all possible segmentation tokens as nodes. Token boundaries become the lattice edges, connecting nodes starting and ending at the boundary. The search then finds the highest scoring path through the lattice.

Another branch of search-based approaches splits decisions into transitions (starting a new token and appending a character to the token) and searches for the highest scoring chain of transitions. This also can be seen as dynamically constructing a lattice while performing the search in it at the same time.

Lattice-based approaches are dominant for the Japanese language. Most of the time, the lattice is based on words that are present in a segmentation dictionary and a rule-based component for handling out-of-dictionary words. An example of a lattice is shown in Figure 1.2. Usually, there are no machine-learning components in lattice creation, but the scoring can be machine-learning based. We believe that the availability of high-quality consistent morphological analysis dictionaries is the reason for that. Still, the work of Kaji and Kitsuregawa (2013) is a counterexample of a lattice-based approach for Japanese, where the lattice is created using a machine-learning approach.

Traditional lattice-based approaches for Japanese use mostly POS tags or other latent information accessible from the dictionary to score paths through the lattice. JUMAN (Kurohashi 1994) is one of the first analyzers, which uses a hidden Markov model with manually-tuned weights for scoring. Lattice path scores are computed using connection weights for each pair of part-of-speech tags. ChaSen (Matsumoto, Takaoka, and Asahara 2008) is an improvement over JUMAN with adding automated weight tuning for the hidden Markov model (Asahara and Matsumoto 2000).

The most known and used morphological analyzer for Japanese is MeCab. Its ideas were reimplemented in a large number of analyzers like Kuromoji[2] and Sudachi (Takaoka, Hisamoto, N. Kawahara, Sakamoto, Uchida, and Matsumoto 2018). It builds a lattice and uses conditional random fields for learning the scoring func-

[2] https://www.atilika.org/

tion. MeCab is very fast: it can analyze almost 50k sentences per second while having good accuracy. The speed is realized by precomputing feature weights into a 2D array for bigram features and token weights for unigram and unknown node features. Unfortunately, precomputing the 2D array limits the kind and maximum variety of feature templates it is possible to use. For example, MeCab cannot use *fully* lexicalized features, and its bigram feature templates usually capture relations between POS tags. On the other hand, *partial* lexicalization, usually of auxiliary words, is often used in bigram MeCab features. When the bigram features start to contain a large number of diverse feature instances, the 2D array, and so the model itself, becomes very large. For example, the UniDic model for modern Japanese v2.3.0[3] (Den, Nakamura, Ogiso, and Ogura 2008) takes 5.5GB because it uses many feature templates, which makes such an approach rather unwieldy.

Search-based models relying on POS-based features have problems when analyzing sequence of tokens with the same POS tag. One famous example is "外国人参政権", which has the correct segmentation "外国|人|参政|権", but the often produced segmentation is "外国|人参|政権". In a model with non-lexicalized bigram features, this is equivalent to making a decision whether P(外国|NN)P(人参|NN)P(政権|NN) is lower than P(外国|NN)P(人|NN)P(参政|NN)P(権|NN), where NN is the POS tag for common nouns. We believe that in general, if both sides contain words of similar frequency, it is undecidable. However, even lexical bigram information can help to solve most of such situations.

Another workaround to this problem is to register the whole sequence 外国人参政権 to the dictionary as a single super-token and handle such super-tokens in post-processing. JUMAN, ChaSen, and Sudachi support this approach. It can also be viewed as partial lexicalization. Still, this requires a lot of careful curating and super-token selection while not every possible situation is handleable in this way. Such super-tokens must also be not context-dependent. Context-dependent situations (e.g. "米原発" which can be segmented as "米原|発" or "米|原発") require surrounding information to produce the correct segmentation and can not be hard-coded.

The neural network-based model was integrated with the lattice search. Juman++ (Morita, D. Kawahara, and Kurohashi 2015) uses dictionary-based lattice construction with the combination of two models for path scoring: the feature-based linear model using soft-confidence weighted learning (J. Wang, P. Zhao, and Hoi 2016) and a recurrent neural network (Mikolov 2012). It significantly reduced the number of both segmentation and POS tagging errors. However, it was very slow, being able to analyze only about 15 sentences per second, hence the original version was

[3] https://unidic.ninjal.ac.jp/

impractical. The following improvement (Tolmachev, D. Kawahara, and Kurohashi 2018, 2020), also described in chapter 2 greatly increased analysis speed by doing aggressive beam trimming and performing heavyweight NN evaluation only after lightweight scoring by the linear model.

Direct lattice-based approaches are not very popular in Chinese, but some are lattice-based in spirit. A line of work by Yue Zhang and Clark (2008, 2010) builds the lattice dynamically from partial words, searching paths with a perceptron-based scorer and customized beam search. The dictionary is built dynamically from the training data as frequent word-tag pairs which help the system to prune unlikely POS tags for word candidates.

Some studies use lattice-based approaches only for POS tagging. For example, Z. Wang, Zong, and Xue (2013) uses a character-based segmenter to get n-best results, compresses them to a lattice, and then applies a lattice-based POS tagger to get the tagging results.

One more variation on lattice-based approaches for Chinese is the work by Cai and H. Zhao (2016). In this work, a segmentation dictionary is used to construct a subnetwork, which combines character representations into word representations used for computing sentence-wise segmentation scores. This can be seen as explicitly learning dictionary information by a model. The resulting segmentation is still created from the start to the end by growing words one by one while performing a beam search. The follow up (Cai, H. Zhao, Z. Zhang, Xin, Y. Wu, and F. Huang 2017) simplifies that model and shows that greedy search can be enough for estimating segmentation when using neural networks. Still, this line of work does not consider POS tagging.

Another branch of search-based approaches uses **transitions**. They treat input data (most frequently – characters) as input queue and store a current, possibly incomplete, token in a buffer. Models then usually infer whether they should create a new token from a character in the input queue or append an input character to the already existing token. Neural models are often used in this paradigm. Jianqiang Ma and Hinrichs (2015) defines two actions: Append and Separate, which move the first character of the queue to the buffer either appending it to the current token or creating a new token, respectively. They use a simple neural model for embedding representations. Meishan Zhang, Yue Zhang, and G. Fu (2016) use a more complex LSTM-based model with a sequence of a word, character unigram, bigram, and action representation features to score the next action. Yang, Yue Zhang, and Dong (2017) use character window representation, compressed by an MLP from character embeddings near the target character, partial word representation, and word embeddings. Ji Ma, Ganchev, and Weiss (2018) use a stacked bi-LSTM model

where the outputs of a backward direction are fed into the forward direction. The inputs are pretrained character unigram and bigram embeddings. This work only considers segmentation. Meishan Zhang, N. Yu, and G. Fu (2018) also, use a bi-LSTM model, but it infers segmentation and POS tags as a subtype of a separate action. It also used word-context embeddings (H. Zhou, Z. Yu, Yue Zhang, S. Huang, DAI, and J. Chen 2017) that were shown to be helpful for transition-based methods. Almost all of them use both word and character n-gram embeddings. This paradigm was extended to do parsing jointly with morphological analysis (Hatori, Matsuzaki, Miyao, and Jun'ichi Tsujii 2012; Kurita, D. Kawahara, and Kurohashi 2017).

1.2.5 Semi-supervised Approaches

Semi-supervised approaches to segmentation and POS tagging fall into several categories. The first one uses raw or automatically-annotated data to precompute feature representations and then uses these feature representations for supervised learning. For example, W. Sun and J. Xu (2011) and Y. Wang, Jun'ichi Kazama, Tsuruoka, W. Chen, Yujie Zhang, and Torisawa (2011) use data from automatically segmented texts as features. They precompute the features beforehand and train an analyzer afterward. In addition to that, L. Zhang, H. Wang, X. Sun, and Mansur (2013) uses a variation of smoothing for handling automatic annotation errors. A lot of neural-based methods pre-train word and character n-gram embeddings. Yang, Yue Zhang, and Dong (2017) pre-train a part of the model on different data sources, including automatically segmented text, but the model itself is trained only on the gold data.

Another approach is to use heterogeneous data (annotated in incompatible annotation standards). In addition to corpus statistics from a raw corpus, H. Zhao and Kit (2008) exploit heterogeneous annotations. Z. Li, Chao, Min Zhang, and W. Chen (2015) use corpora with different annotation standards. They combine tags into "bundles" (e.g. [NN, n]) and infer them at the same time while paying attention to ambiguity. Hongshen Chen, Yue Zhang, and Q. Liu (2016) train a classifier that can annotate several standards jointly.

Finally, it is possible to use raw or automatically-annotated data directly. A study (Suzuki and Isozaki 2008) is an example of a feature-based algorithm that uses raw data. Tri-training (Z.-H. Zhou and Ming Li 2005) is a generic way to use raw data. They propose to train on automatically analyzed examples where two of three diverse analyzers agree. Søgaard (2010) show that tri-training helps English POS-tagging with SVM and MaxEnt-based approaches. H. Zhou, Z. Yu, Yue Zhang, S. Huang, DAI, and J. Chen (2017) use self-training and tri-training for Chinese word segmentation. They, however, also pre-train other features like word-context

character embeddings, character unigrams, and bigrams.

1.3 Example Extraction for Language Learning

1.3.1 Vocabulary Learning

Learning vocabulary is a crucial step in learning foreign languages and mastering the vocabulary of a foreign language requires substantial time and effort. Moreover, we do not use plain words for communicating. Words are always surrounded by other words and grammar, forming **word usages**. When learning vocabulary, learning these word usages is as important as learning words themselves.

The word-learning process may be grouped into four parts. A learner must accomplish the following for most of the words in the vocabulary to gain proficiency in a language:

- understand the word itself,
- understand its usage in context,
- remember the word,
- remember its usage.

There are tools at our disposal for each of the abovementioned actions. We use dictionaries to understand the words from definitions or translations. We must actively read many books to fully understand the usage of words in different situations with their subtle nuances. When learning a new language, we must constantly repeat new words, otherwise, most of us simply forget them.

Example sentences, which demonstrate the usage of a word in context serve as tools to facilitate each of the above actions. For example, dictionaries contain example sentences as additional information to clarify the meaning and context of a word.

There is one more very useful tool for language learning — flashcards and spaced repetition systems (SRS) — which is discussed in the next section. This tool is very successfully used for language learning because it helps learners remember the words and meanings. For example, the most popular SRS — Anki — has at least 1 million active installations on Android alone, and its most popular usage is for learning foreign languages. Two main problems form the drawbacks of the SRS:

1. It is very tedious to manually create suitable flashcards.
2. Flashcards often lack *context*.

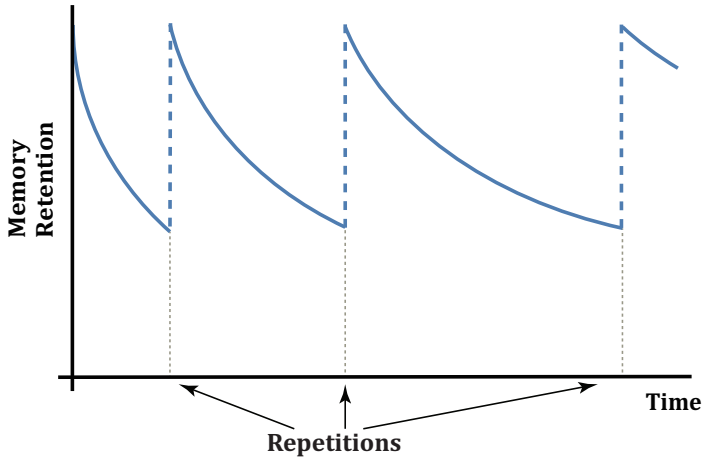


Figure 1.3: Forgetting curve. A probability to remember a fact exponentially decreases with time. The decrease speed becomes slower on each repetition.

Learning lexical information using an SRS is typically done using only words. This does not give the learner information about how the word is used. Manually adding examples to flashcards is not only tedious but also challenging in that suitable example sentences are difficult to find. Moreover, even if examples are added, it is almost impossible to manually add many examples. However, only a large number of suitable example sentences can give the learner sufficient exposure to the different usages of a word. Thus, it is preferable to automatically add examples to cards.

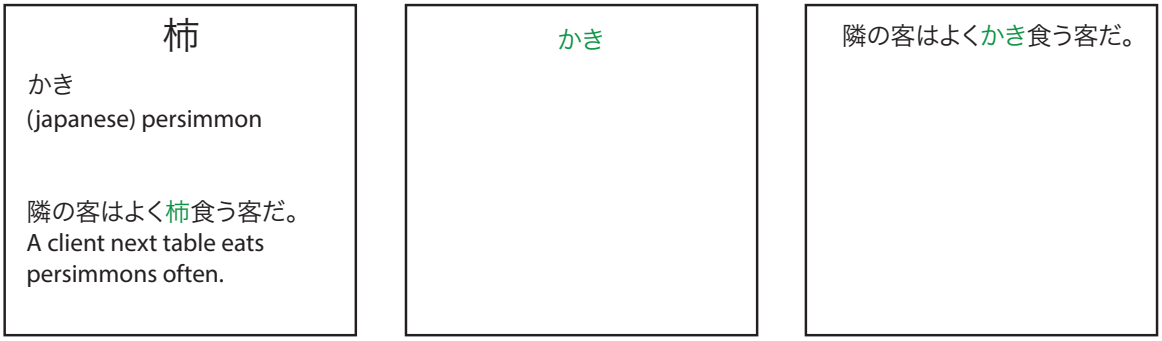
The main objective of the research defined in this thesis is to explore and evaluate a way to automatically acquire a large number of high-quality example sentences for the Japanese language. Future work entails using these example sentences in an SRS for the Japanese language, so the goal is to obtain sentences suitable for use on flashcards. Before discussing the traits of example sentences that are appropriate for flashcards, we must briefly describe flashcards and the SRS.

1.3.2 Spaced Repetition Systems and Flashcards

Almost anyone who has tried to learn a foreign language is familiar with the feeling of forgetting recently learned words. The process of forgetting has been researched in the context of psychology with Ebbinghaus formulating principles of human memory (Wozniak 1999). One of them is that the probability of **retention** of a fact is proportional to e^{-t}, where t is the time passed since the last repetition.

Repeating the fact temporarily resets retention to the maximum value, after which the retention probability starts decaying again. If the retention was successful, the decay rate slows down. Figure 1.3 depicts this process.

The **spaced repetition** method exploits the structure of the forgetting curve. The



A flashcard for word 柿 (answer card)

Question for the reading card without examples. User is expected to remember the writing.

Question for the reading card using examples. User is expected to remember the writing.

Figure 1.4: A flashcard for word "柿"

basic idea is that after the successful retention of a fact, its next repetition is scheduled just before its *high probability of forgetting*. If the learner reviews the item before that time, it will be remembered for a while. Effectively, intervals or the **spacing** between repetitions increases exponentially each time the learner successfully remembers the item, enabling the learner to simultaneously remember a large amount of data.

A computer program implementing spaced repetition is called an SRS. Most SRSs use flashcards. A **flashcard** is a piece of knowledge formatted in a question–answer way, as shown in Figure 1.4. The left image is an answer flashcard for the word "柿". For the Japanese language, the flashcard typically contains reading, writing, meaning, and possibly example sentences that explain the usage patterns of the word. The middle figure is a question flashcard which uses the word reading and the learner is expected to remember the content of the answer card and self-rate how difficult it was to remember the content of the answer card. Other types of cards, e.g. writing-question or meaning-question, are often used as well with the answer card staying the same.

The main problem with these systems is the large amount of work required to create new flashcards. To learn words efficiently, the cards should be created by the learner himself, for example when encountering unknown words in the text. Existing software requires the user to fill in all of the information about the a word from a dictionary before proceeding to learn it. Pre-made decks created by different users are available in some software, but they do not create "emotional attachment" (for example, a user can remember the circumstances under which he encountered a certain word for the first time, and this helps him remember it) to words, and this makes it more difficult to learn these words. It is easier for a user to remember the words that he added to his learning system himself, and the system should support this.

Another Japanese-specific problem is a large number of homonyms. For example, when trying to define the written word "かき" (as shown on the Figure 1.4, middle) it is impossible to give a single correct answer because the question is underspecified. It is thus difficult to learn such words. The software mentioned above does not address this situation and delegates the responsibility of creating suitable flashcards to the user, rendering these cases even more difficult and time-consuming.

An effective system should automate almost every aspect of adding new words for learning, transferring all of the burdens of inputting the information from the user to the system. Completing the meaning and reading information from the dictionary is relatively easy but does not form a comprehensive solution. Words are not just used by themselves; they are always used in context. By incorporating example sentences in flashcards as questions and reference material as shown in Figure 1.4, it is possible to solve the polysemy and homonymy problem of plain flashcards. Example sentences give the learner *context*, which itself is a hint for selecting a correct word (including the writing in the case of Japanese). At the same time, reading a large number of example sentences helps the user remember usage patterns, thereby improving the efficiency of word learning.

Unfortunately, there are no example databases with high quantities of example sentences for a wide range of words, including rare and obscure ones. Examples from dictionaries are often sentence fragments and are not very suitable as well. This problem is addressed by automatically extracting example sentences suitable for flashcards.

1.3.3 Example Extraction

Example sentences provide additional information about a word. However, words are used in different contexts, with different grammar, and can have multiple meanings. Thus, *a set* of example sentences describing a *target* word is more appropriate than *a single* example sentence. In this thesis, target refers to the particular word that the sentence aims to explain.

A sentence could include a target word, but still not provide a good example of the word usage. The sentence could be too difficult for a learner, too long so while reading it, the learner would forget an actual usage of the target word or just plain non-grammatical. A good example sentence should give hints on the meaning of the target word by the context. For example, when "ばりばり" is a target and a learner does not know it, a sentence "首がばりばりになる", will give learner hints that "ばりばり" can be used with "首", but will not give hints on its meaning. Adding context, for instance, "働き過ぎると首がばりばりになる", is significantly better because it connects the meaning with overworking and being tired. Still,

the last sentence would be a bad example for a word "働く" as a target, because onomatopoeia are generally very difficult to learn, and ばりばり is going to shadow 働く with its difficulty.

The goal of the work described in this thesis is to extract high-quality example sentences. We assume that a sentence has a high *intrinsic value* if it satisfies the following requirements.

- Sentences should appropriately explain the usage of the target word.
- Sentences should be understandable by the learner.
- Sentences should be reasonably short. If the sentence is too long, the concrete usage of the target word may be shadowed by other words.
- Sentences should be grammatically correct.
- Sentences should be closely related to the *target* word; The usage should be *representative*.
- Sentences should be complete, not fragments.
- Sentences should not require deep additional knowledge or additional context for understanding. In contrast to learners, native speakers usually have cultural and language knowledge, allowing them to reconstruct contexts from experience.

In addition to the above, the sentences in the set should be *diverse*, covering distinct meanings, including different grammatical constructions and words. We call sentences *high-quality* compared with other sentences for the same target word if

1. each sentence has a high intrinsic value,
2. sentences are diverse.

In addition, these sentence sets should be created for as many target words as possible, including rare ones, and cover the rarer usages of targets. Thus, an example extraction system should consider the following three aspects of sentences: inherit value, diversity, and *coverage*.

Example Sentences for Reference and Repetition

Another important aspect of example sentences is that those used for the flashcard and repetition should be slightly different from those usually used for the *reference* (i.e., standard usage in dictionaries). A list of example sentences for each sense of the

target word would help the user understand the word itself and its usage. A learner can distinguish the usage of a word in different contexts from different examples.

In contrast, an SRS displays flashcards without any relationship between them. This means that the sentences in a series of flashcards are also unrelated. Hence, these sentences should be easy to read and understand for learners, thereby increasing the efficiency of the learning.

We focus on the extraction of the example sentences for usage in an SRS. However, reference usage is still important and should be addressed by other developed tools.

1.3.4 Related Work

Several SRSs are available such as Anki[4], open-source software which is probably the most popular SRS available; Mnemosyne[5], another open-source solution that incorporates research on long-term human memory; and SuperMemo[6], made by the developer of the original SuperMemo algorithm, which is closed-source and not free. Anki has a plugin that simplifies the process of creating flashcards for the Japanese language. However, none of the above systems automatically provide example sentences. In addition, all of the systems provide only static flashcards without tools to change the content for each repetition. Showing different example sentences on question cards on each repetition should improve the efficiency of learning vocabulary.

Dictionaries often include example sentences in their articles about words. However, these examples are often very short. 広辞苑 has the following two examples for the second sense of the word 振るう: “采配を振るう” and “拳を振るう”. They are extremely short and are more like collocations than examples. They do not provide any useful context such as the situations for which this word can be used.

There exist dictionaries that consist mostly of example sentences such as Progressive (Tohno, Itabashi, and Althaus 2001) and Wisdom (Inoue and Akano 2007). Example sentences in Progressive are usually full sentences while the example sentences in Wisdom mostly have a fragment-like structure containing only the information needed to understand the word in context. Both of these dictionaries also contain English translations because they explain word usage mostly using the example sentences and this is extremely useful for the language learners. Still, because the sentences were manually assembled by human editors, their number is limited. Another point to consider is that dictionary content is under copyright protection and cannot be easily used for automated flashcard generation.

[4] http://ankisrs.net/
[5] http://mnemosyne-proj.org/
[6] http://www.supermemo.com/

Freely available example sentence databases also exist. The Tatoeba Project[7] is a wiki-style database of example sentences maintained by human volunteers. It consists of example sentences for many languages. Thirteen languages have more than 100,000 sentences registered and 39 have more than 10,000. Of note, as of January 22, 2016, there were approximately 570,000 English sentences and 180,000 Japanese sentences. The sentences are interrelated, facilitating translations in different languages in a many-to-many fashion. Still, most of these sentences focus on relatively easy words and many of the sentences are very similar to each other.

Automated extraction of example sentences from a corpora has also been proposed. GDEX (Kilgarriff, Husák, McAdam, Rundell, and Rychlý 2008) describes semi-automated example extraction for the preparation of the electronic version of a Macmillan English Dictionary. The authors select example sentences for English learners and define a suitable example sentence as:

- typical, showing frequent and dispersed patterns of usage,
- informative, helping to educate the definition,
- readable, meaning intelligible to learners, avoiding difficult words, anaphora and other structures that make it difficult to understand a sentence without access to the wider context.

The authors used sentence length, word frequency, information about the presence of pronouns and some other heuristics to judge the quality of sentences. Subsequently, the final example sentences for the dictionary were manually selected by editors. The authors reported problems with garbage and list-like constructions in the raw data that are not useful for example sentences. Still, their approach decreased the time required to construct the dictionary.

There are numerous works that approach the problem of selecting example sentences mostly as a word sense disambiguation (WSD) problem (de Melo and Weikum 2009; Kathuria and Shirai 2012; Shinnou and Sasaki 2008). Specifically, de Melo and Weikum (2009) proposed the use of parallel corpora to extract disambiguated sentences from an aligned subtitle database. However, they only examined Spanish-English language pairs and the total number of sentences in their work was small (117,000 sentence pairs). Aligned corpora usually are small or belong to a specific domain, whereas example sentences should be from a variety of domains and cover rare words.

Shinnou and Sasaki (2008) target example extraction for Japanese language. They cluster initial sentences into a specified number of clusters. Then, by showing a pair

[7] http://tatoeba.org/eng/

of central sentences to a human operator, they decide whether the clusters should be merged. The authors only consider nouns and measure the similarity between sentences using word overlap and a thesaurus. Their approach is semi-supervised and requires user interaction for system decisions. Furthermore, it is not created for language learners.

Kathuria and Shirai (2012) explore the use of disambiguated example sentences in a reading assistant system for Japanese learners. They create a system that assists reading by showing disambiguated example sentences that have the same sense as the word in the text. The senses are defined by the EDICT dictionary (Breen 2004). The authors perform WSD based on the similarity between sentences where the similarity consists of collocation and a sema-syntactic part. The second aspect is based on dependency parse information combined with Goi-bunrui-hyou (Natural Institute for Japanese Language and Linguistics (ed.) 2004) prefix matches. Only an aligned corpus is used to extract the example sentences, limiting the number of potential example sentences.

1.4 Contributions

Our contributions are separated into two main parts. The first part contains improvements in Japanese Morphological Analysis, and the second part contains an approach for automated high-quality example sentence extraction.

1.4.1 Morphological Analysis

Our first contribution for the morphological analysis is the improvement of Juman++ morphological analyzer based on the classical lattice search approach. The reimplementation: Juman++ V2[8] has more than 250 times faster analysis speed than the base version V1 while improving on the analysis accuracy, both in segmentation and part of speech tagging. We also provide a partial annotation tool for the Juman++ V2 which can help to quickly improve accuracy for new domains. The improvements for Juman++ are published as (Tolmachev, D. Kawahara, and Kurohashi 2018) and (Tolmachev, D. Kawahara, and Kurohashi 2020).

Our second contribution is a very straightforward fully-neural morphological analyzer which uses *only character unigrams* as its input (Tolmachev, D. Kawahara, and Kurohashi 2019)[9]. Such an analyzer, when trained only on human-annotated *gold* data has low accuracy. However, when trained on a large amount of automatically tagged *silver* data, the analyzer rivals and even outperforms, albeit slightly, the

[8]The analyzer is available at https://github.com/ku-nlp/jumanpp
[9]The source code is avaliable at https://github.com/eiennohito/rakkyo

bootstrapping analyzer. We conclude that there is no need for rich input representation. Neural networks learn the information to combine characters into words by themselves when given enough data.

Ignoring explicit dictionary information and rich input representations makes it possible to make analyzers that are highly accurate and very compact at the same time. We also perform ablation experiments which show that the encoder component of such an analyzer is more important than character embeddings.

1.4.2 Example Extraction

The second large contribution is an automated example sentence extraction system for Japanese language learners. The system description is published as (Tolmachev and Kurohashi 2017a) and (Tolmachev, Kurohashi, and D. Kawahara 2022). The system consists of two parts: search and extraction. The current implementation of the system is for Japanese, but generally, it is unsupervised and requires only a dependency parser and mono-lingual text to bootstrap.

The first component of the system, search, is a high-performance distributed dependency and grammar-aware search engine. This component description was published as (Tolmachev, Morita, and Kurohashi 2016). Its design and implementation make it possible to process a vast corpus relatively quickly, thereby handling rare words and the coverage problem. Automatically handling grammar and dependency information facilitates the quick selection of sentences with patterns that are useful for understanding a target word. For example, verbs are more easily understood if dependent objects or subjects are included in the sentence. The search system allows the specification of such restrictions in the query and is described in more detail in Chapter 4.2.

The second part, extraction, takes a relatively large list of search results and selects only a small subset of sentences that are most useful for a learner. The proposed method uses the determinantal point process (DPP), a mathematical method for modeling diverse datasets. It uses two sets of features: similarity and quality. The similarity features are vectors and are used to measure the relationships between two sentences. They consist of three parts: lexical, syntactic, and semantic similarity. The quality features are per-item and are used to represent the intrinsic sentence value. They consist of semantic and syntactic centrality, difficulty, and low-level "goodness". The extraction part is described in Chapter 4 and published as (Tolmachev and Kurohashi 2017a).

The example sentences extracted by the system were evaluated by language learners. Specifically, the examples were selected for the purpose of flashcard usage (Tolmachev and Kurohashi 2017b). Learners were asked to compare the sentences

produced by the system to two baseline selection algorithms. The experimental setup and results are described in 4.5.

1.5 Design Considerations

Engineering and system design is built on trade-offs. Usually, we have to decide between several properties of the resulting solution, like cost, time to implement, and overall quality. Low-level blocks of the overall system place strong restrictions on the weight of such trade-offs. Generally, improving low-level building blocks help to diminish trade-offs. In the case of morphological analysis, we can choose between accuracy, affecting the overall robustness and reliability of the system. On the other hand, the low analysis speed decreases the system response time and can make it impossible to process a large number of data without using an extreme amount of computational resources. Our improvements in the Juman++ soften the trade-off weights and make it possible to construct overall better systems that utilize NLP for Japanese. Experiments on fully-neural morphological analysis make the dictionary size smaller. It can allow, for example, embedding the analyzer into the client-side of a web application, making new types of applications possible.

In regard to language learning, our ultimate goal is to make language students learn not just words, but their correct usage at the same time as well. We do this by providing high-quality example sentences which cover diverse word usage, in lexical, syntactic, and semantic aspects. Exposing students to the word usage contextual information should naturally train them to recognize natural word usage from unnatural one. Usually, this happens by being exposed to the language and culture, for example by reading. Moreover, the automatic nature of example extraction can be further tailored for specific user needs.

2 Juman++ Morphological Analyzer

2.1 Introduction

Languages with a continuous script, like Japanese and Chinese, do not have natural word boundaries in most cases. Natural language processing for such languages requires segmenting text into words. Segmentation is commonly done jointly with part of speech (**POS**) tagging and the whole process is usually referred to as Morphological Analysis (**MA**).

Modern morphological analyzers achieve high accuracy (a tokenwise segmentation F1 score of $> .99$ for Japanese) on established domains like newspaper texts. 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.

To increase the overall analysis accuracy, Morita, D. Kawahara, and Kurohashi (2015) have proposed using a combination of a feature-rich linear model with a recurrent neural network-based language model (**RNNLM**) for morphological analysis and implemented it as the initial version of Juman++. The combined model considers semantic plausibility of segmentation, and because of this has drastically reduced the number of intolerable analysis errors, achieving the state-of-the-art analysis accuracy on Jumandic (the JUMAN dictionary and segmentation standard, Kurohashi and D. Kawahara (2012)) based corpora. In this thesis, we refer to it as **Juman++ V1** or simply **V1**. Unfortunately, its execution speed was extremely slow, and that limits the practical usage of Juman++ V1.

We have developed a morphological analysis toolkit (Tolmachev, D. Kawahara, and Kurohashi 2018) 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++ V1, released as Juman++ V2[1] (Tolmachev and Kurohashi 2018). In this thesis we refer to it as **Juman++ V2**, **V2**, or simply **Juman++**. 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.

[1]The analyzer is available at https://github.com/ku-nlp/jumanpp

Juman++ follows the dictionary-based morphological analyzer design. During the analysis, it looks up word candidates using a dictionary and unknown word handlers, and then builds a lattice by connecting adjacent word candidates. The path through the lattice with the highest score becomes the analysis result.

The main goal of Juman++ is to achieve a reasonable analysis speed while being flexible and powerful at the same time. MeCab (Kudo 2018; Kudo, Yamamoto, and Matsumoto 2004) achieves its high analysis speed by precomputing lattice node connection features into a 2D table, referring only to the precomputed table at the analysis time. This makes it impossible for MeCab to use *fully* lexicalized bigram feature templates with *a large number of instances*. This design decision also makes it impossible for MeCab to use surface-based connection features. We must remark that partial lexicalization of bigram features is supported by MeCab and is frequently used, mostly for auxiliary words.

Juman++, on the other hand, supports such features for the sake of analysis accuracy. We achieve a reasonable analysis speed by designing the architecture of the analyzer, in a way that the modern CPUs could efficiently execute the compiled code. In order to achieve a high analysis speed, each stage of the analysis was designed for efficiency. For example, the dictionary structure and feature representation are designed without string-based operations in computations. For in-memory lattice representation, we focus on improving CPU cache efficiency. Feature-based path scoring is done by generating a model-specific C++ code which is organized to mask memory latency by asynchronously prefetching the model weights. We also perform the aggressive beam trimming and deduplicate the paths through the RNN which would have the same representation.

This chapter presents an insight into Juman++ internal architecture, design, and rationale for making those design decisions. We discuss the internal data structures and inner workings of Juman++ V2. When compared to MeCab, which performs a single 2D array access and summation with a combined unigram score (2 operations), Juman++ does significantly more computations: it evaluates about 60 features for each lattice node while performing the beam search while being only 5 times slower. We believe that the information on how to perform a large number of computations efficiently in machine learning-based software will be useful for the further development of natural language processing tools.

The chapter is structured as follows. In section 2.2 we discuss the problem of morphological analysis and present approaches. Section 2.3 discusses the internals of Juman++ and explains our decisions. In Juman++ we heavily exploit the capabilities of modern CPUs, some of which are described in Appendix 2.9. Section 2.4 discusses the training of Juman++ models. Section 2.5 evaluates and compares Juman++ to

different morphological analyzers in three categories: model size, analysis speed, and analysis accuracy. Finally, section 2.7 gives the discussion on the remaining problems of morphological analysis and Juman++.

2.2 Morphological Analysis Overview

2.3 Juman++ Internals

The logical subcomponents that contributed to the Juman++ speedup are the following:

1. linear model score computation,
2. beam search,
3. RNN model.

While the last two subcomponents are rather small and self-contained, the first one is complex and forms the core of Juman++ design decisions: dictionary structure, feature computation hashing, code generation, and low-level implementation details like prefetching and struct-of-arrays layout (described in the Appendix 2.9.2) for the lattice. In this section we walk through the components in the order an input sentence goes through the analysis, starting with a general overview.

We provide the estimates of individual contributions to the overall speedup. Unfortunately, the rigorous experiment and comparison can not be performed because the components are highly intertwined, and swapping one component for another means doing a large-scale rewrite. Additionally, the development of Juman++ V2 did not happen in the order of the paper and it is difficult to use the revision history as a means for comparison.

2.3.1 Juman++ Analysis Overview

For an input sentence, the detailed description of Juman++ analysis steps is the following:

1. Lookup dictionary-based lattice node candidates. This step uses the dictionary to emit triples of (token start, token end, entry pointer) for every dictionary candidate that can exist in the input string.
2. Create unknown word node candidates. This stage also creates triples like the first stage.

3. Build the lattice. This stage reads token information from the dictionary, computes token-specific features and part of the score which depends only on unigram features.

4. Find top k highest scoring paths through the lattice using beam search. This stage computes the feature-based score for nodes.

5. Perform RNN reranking of the top k paths.

6. Output the analysis result.

The full morphological analysis score s for a sentence is a sum of per-node scores and is defined as

$$s \doteq \sum_{\text{nodes in lattice path}} s_l + \alpha(s_r + \beta), \tag{2.1}$$

where s_l is a *linear model* score for a lattice node, s_r is an *RNN model* score — non-normalized log-probabilities, α and β are scale and bias RNN hyperparamters. For computational efficiency, we use self-normalizing RNNLM which makes it possible to leave out the computation of the probability normalizer over the whole vocabulary. Still, the RNN scores are in a different range from that of the linear model. The RNN bias and scale parameters make the scores compatible with the linear model score. We also compute the RNN score only for paths with the top k linear model scores because the computational cost of applying RNN to the whole lattice is prohibitively high. The RNN itself is an optional component, its absence is equivalent to setting the scale $\alpha = 0$.

The linear score itself is defined as

$$s_l \doteq \sum_i \phi_i w_i, \tag{2.2}$$

where ϕ is an **n-gram feature** count vector and w is linear model weight vector. N-gram features are composed of **tokenwise features** using the templates defined in the spec. The presence of high-order n-gram features necessitates the usage of **beam search** to find the top-scoring path through the lattice. We must note that the current implementation of Juman++ does not use the RNN model during the beam search.

The training of Juman++ includes optimizing values of w and searching for RNN-related hyperparameters. Juman++ can use both fully and partially annotated data for training of w.

Most of the computations at the analysis time happen during the path scoring. We adopt the struct-of-array layout of lattice-related data structures to improve the CPU cache efficiency of the entire process.

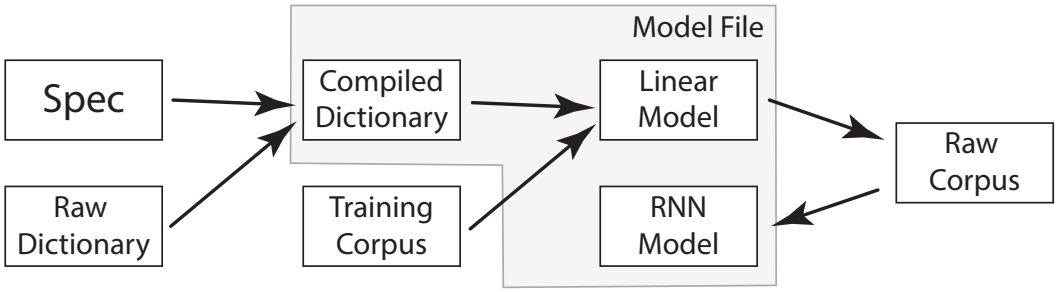


Figure 2.1: Juman++ model file construction process

2.3.2 Juman++ Model

Juman++ by itself is only a toolkit for building morphological analyzers. The rules on how to perform the analysis are contained in a model, which is built from the following components:

1. Analysis specification (**Spec**),
2. Analysis dictionary,
3. Training corpus,
4. RNN model (optional).

Spec is a configuration for Juman++ instance, similar to feature templates used in other analyzers. It configures dictionary structure, feature templates, unknown word handling, and training loss. The documentation for the language and the configuration capabilities are available at Juman++ repository[2]. The dictionary contains the basic vocabulary the MA is going to recognize. For the analysis to work fast we need to *compile* the dictionary to the format which will enable fast lookup of dictionary information. We also have to train linear and RNN models to have the analysis result correct. The combination of these components forms a Juman++ analysis model. The model components and their interaction are shown in Figure 2.1.

2.3.3 Spec and Dictionary

In this subsection we give a brief overview of the dictionary specification, then we describe the dictionary compilation process logic, followed by the important implementation details.

[2] https://github.com/ku-nlp/jumanpp/blob/master/docs/spec.md

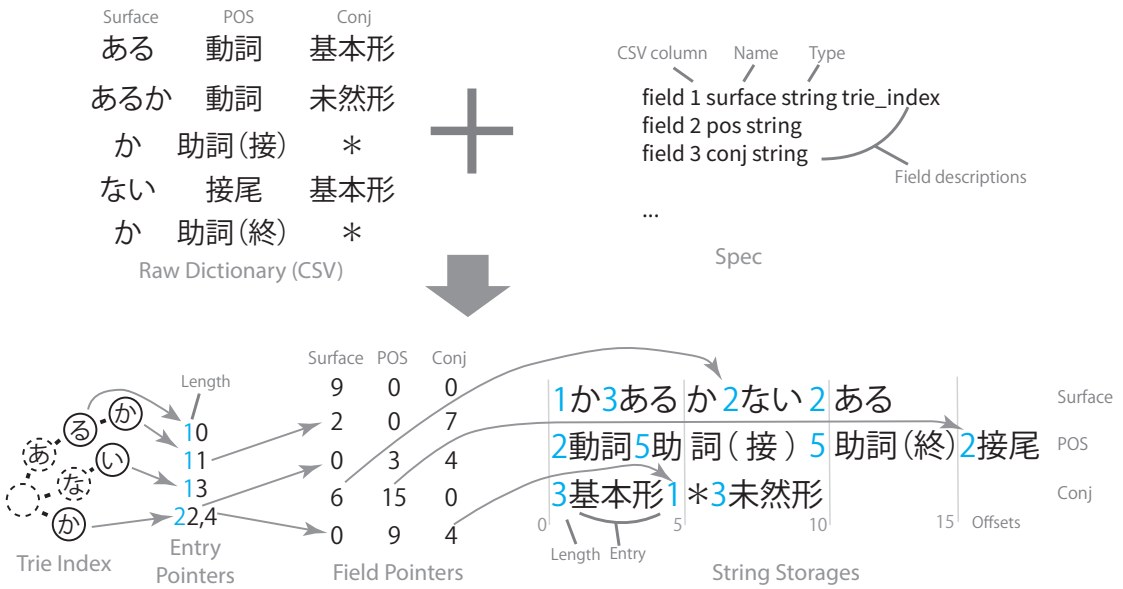


Figure 2.2: Dictionary compilation. The actual information is black, explanations are gray. String and list lengths are blue. From the raw dictionary and spec field descriptions, we create compiled dictionary components: trie index, entry pointers, field pointers, and string storages. Trie index leaf nodes (non-leaf nodes are dashed here) point to a list of entry pointers (index keys can be duplicates). Each entry pointer refers to a row in the field pointer table, corresponding to a row in the raw dictionary. Field pointers refer to deduplicated column entries of the raw dictionary, packed in column-specific string storage.

Dictionary Specification

The first section of a spec defines which fields of a raw dictionary would be used for analysis and available for output. Based on that definition, we compile a raw dictionary into a representation that enables effective dictionary access at analysis time. An example of dictionary compilation is shown in Figure 2.2. Dictionary compilation has the following goals:

1. During the lattice construction phase we need to look up the dictionary entries corresponding to surface string fragments. This process should be fast and not dependent on the dictionary size. Often, a trie-based approach is adopted for this objective.

2. Handling strings as a sequence of characters or bytes is slow. On the other hand, feature computation for the lattice path scoring should be as fast as it can get. We would like to move as much computation as possible from the analysis step to the dictionary construction step. Our compiled dictionary design should allow us to treat all string entries in the dictionary as integers for the ease of further processing.

3. Reducing the compiled dictionary size would also be beneficial.

4. On-disk dictionary representation should be memory-mappable and should not require time-consuming processing logic at startup.

Raw dictionary is a RFC 4180 (Shafranovich 2005) CSV file. Each row corresponds to a single dictionary entry and columns correspond to the dictionary fields. Juman++ can use only a subset of columns from the raw dictionary. Namely, each of spec **field descriptions** selects a single column from the dictionary.

Dictionary fields are typed. The handling of field data and its compiled representation depends on the field type. Juman++ supports four field types:

- Integers. 32-bit signed integer. While it is possible to use integers in a spec directly, this field type is used internally for storing values computed at dictionary compilation time.

- Strings. Strings are treated as categorical features: the strings themselves are replaced by integer pointers to the deduplicated values Most dictionary fields are usually strings.

- String lists. Dictionary entries can contain non-tabular information and this field type is useful for storing such information. The handling of individual strings is the same as the previous type, we also deduplicate the lists themselves.

- String key-value pair lists. Similar to the string list, but each entry holds two values.

Dictionary Compilation

For the compiled dictionary storage we adopt *column database* paradigm. We deduplicate string field contents and pack them, prefixed by their length, into **string storage**. Because the contents of such storage could be uniquely identified by their position inside the storage, we replace dictionary entries by 32-bit integer **field pointers** into respective string storages. This process is shown in Figure 2.2. For clarity, the figure treats the size of everything as 1. The actual implementation takes into account variable-length UTF-8 string encoding and the integers are encoded using the variable-length LEB128 format (described later).

A spec must contain exactly one string field which would be used as a lookup key into the dictionary itself. Juman++ will build a double array trie index using the values of that field. There could be multiple entries with the same key. We handle that by pointing the trie index to the length-prefixed list of entry pointers.

Table 2.1: An LEB128 encoding example. The processing flows from the top to the bottom.

Decimal	105			624485
Binary	1100101		100110000111101100101	
Split into 7-bit groups	1100101	0100110	0001110	1100101
Prepend 1 to non-last groups	01100101	00100110	10001110	11100101
In hexadecimal	0x65	0x26	0x8E	0xE5
On disk	0x65		0xE5	0x8E 0x26

By default, we use different string storages for different fields. However, Juman++ allows sharing string storages as well. For example, surface forms, readings, and dictionary forms usually contain the same entries. Sharing them greatly helps to reduce the size of the compiled dictionary.

Byte-level Storage and Alignment

Before this point, we were talking about the logical representation of the complied dictionary. However, the disk representation works with bytes and bits. Juman++ uses several techniques from search engines to store the dictionary on disk. Namely, we adopt variable length LEB128[3] (little endian base-128) integer encoding (an example is shown in Table 2.1) for storing all integers (both pointers and lengths) in the compiled dictionary. With it, small-valued integers take a smaller number of bytes to store regardless of their original size. Reading small numbers in this representation is less computationally expensive as well. Furthermore, integers that are less than 127 are stored in a single byte having the same representation as the lower 8 bits of the original 32-bit integer.

We exploit these facts about LEB128, which ultimately decreases compiled dictionary size and increases analysis speed. For example, we sort string storage contents by their frequency in the CSV file, in decreasing order. Because of this, more frequent strings will get closer to the beginning of the containing storage, meaning that they will have smaller pointer values, so they will effectively take less space in the field pointer table and could be decoded into the regular integers faster.

Juman++ also allows customizing **alignment** of string storage. By default, length-prefixed stored strings can start at any position. With the alignment of z, those starts can only be multiple of 2^z. Default alignment means $z = 0$. For aligned fields, the lower z bits of field pointers will always be zero and can be ignored. Thus, we bit-shift the pointers to the right by z and store only upper $32 - z$ bits as on-disk field pointer representation. An example of applying alignment to a string storage

[3] https://en.wikipedia.org/wiki/LEB128

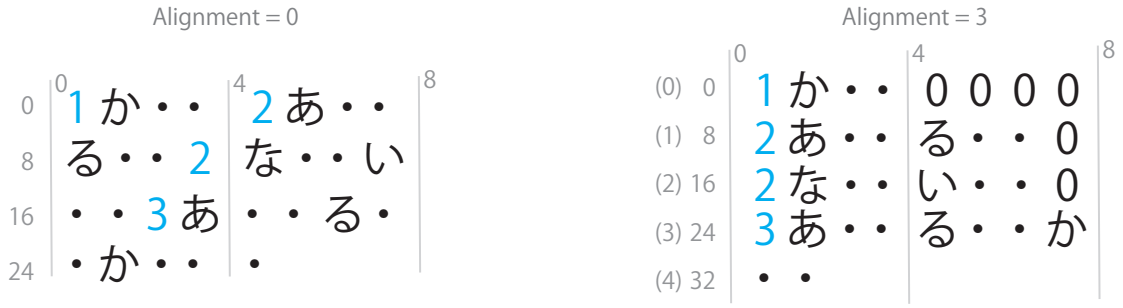


Figure 2.3: String storage alignment example. Dots are additional bytes used by UTF-8 encoding for the respective symbols. Storage with alignment=3 can be indexed using the numbers in brackets.

is shown in Figure 2.3. Aligning string storage makes all respective field pointers smaller, reaping the benefits of LEB128. Moreover, if a field contains only a limited and small set of values (e.g. POS tags), then choosing big enough alignment will guarantee that pointers for that field are always going to be stored in a single byte.

Alignment is a trade-off between using storage for actual strings or field pointers. At the same time, making the field pointer table smaller decreases the amount of memory to read during the analysis, increasing the probability of cache hits, resulting in ~ 15% improved analysis speed compared to zero alignment in our Jumandic experiments. Juman++ includes a development tool that estimates changes in string storage size versus field pointer table size when changing the alignment for a chosen string storage to help users with making the correct decision.

When storing sequences of integers to the disk when their ordering does not matter (or they are already sorted), we store them in **delta encoding**. Instead of storing a sequence $a_1, a_2, a_3, ..., \forall j > i : a_j \geq a_i$ we store the differences between the successive elements: $a_1, a_2 - a_1, a_3 - a_2, ...$ Differences $a_{i+1} - a_i$ have smaller values than values a_i themselves, once more reaping the benefits of LEB128 encoding. We apply the delta encoding on top of LEB128 for entry pointer lists (which are already sorted), string list-typed fields, and key parts of key-value list-typed fields (which we sort).

Representation Limitations

Our compiled dictionary representation uses signed 32-bit integers as pointers and because of that suffers from some limitations. Namely, in the current design, individual sizes of entry pointers, field pointers, or string storage tables cannot exceed $2^{31} - 1$ bytes.

However, this does not mean that the dictionary size cannot be more than $2^{31} - 1$ bytes. For string fields, Juman++ cannot handle more than $2^{31} - 1$ bytes of *unique*

```
field 1 surface string trie_index
field 2 pos string

feature next_char = codepoint 1
feature aux_word = match [pos] with "AUX" then [surface, pos] else [pos]

ngram [pos]                       # No1
ngram [pos, next_char][pos]       # No2
ngram [aux_word][pos, next_char] # No3
ngram [pos][surface, pos][pos]    # No4
```

Figure 2.4: An example of Juman++ spec with feature definitions

string values for each field. In the case of Jumandic, shared string storage for surface, the base form, and reading fields contain 1.7M entries, taking 40 MB of space, less than 2% of the total limit. Because the size scales with unique entries, in practice the limit is not significant.

The field pointer table scales proportionally to the number of entries in the dictionary and can become a limit for truly enormous dictionaries. For reference, Jumandic contains 1.8M entries and its field pointer table takes only 32 MB, giving approximately 18 bytes per entry. It is difficult to give exact numbers on the maximum possible number of dictionary entries, but we believe that the current implementation can handle dictionaries with up to 100M entries. It is possible to align the field pointer table as well to step over the current restrictions, though field pointer table alignment is not implemented currently.

2.3.4 Features

Feature computation is a cornerstone for the analysis speed of Juman++. We use hashing for combining the values from the data into the final feature values. By arranging the computation order, we achieve a large reduction of duplicated calculations. Finally, we describe our reasons and gains from using code generation for feature computation and scoring.

Feature Layers

In Juman++ we ideologically distinguish features into three layers. The features of the top layers are computed using the values of the previous layer.

- Primitive, e.g. field values or other token-specific information. `pos` and `next_char` from Figure 2.4 are primitive features. They are represented by an unsigned 32-bit value.

- Tokenwise, namely a combination of one or more primitive features inside a single token. `[pos, next_char]`, `[pos]` and `[aux_word]` are tokenwise features. They are represented by an unsigned 64-bit value.

- N-gram, which is a combination of tokenwise features over different tokens.

Primitive features are defined for the current token and come from two classes. The first one comes directly from the dictionary: they are values of integer and string-typed dictionary fields. For example, a part-of-speech or a surface string representation would be such a primitive feature. Primitive values of the second type either come from the analyzer input (e.g. characters near the current word) or are computed from the dictionary values (e.g. length of a surface field). The full list of supported primitive features is available in spec format description documentation.

Juman++ has also support for **partial lexicalization**. See the *aux_word* feature definition in Figure 2.4. *Match* primitive feature is similar to an *if* statement. The feature itself is expanded in one of two lists, depending on the condition. The condition can match a list of fields to a list of values and is computed at *dictionary compile* time to save the computations at analysis time. Conditions for multiple *match* features are packed as distinct bits to an automatically generated integer-typed dictionary field.

Tokenwise features combine primitive features within a single token. We distinguish them because n-gram features can contain several identical combinations of primitive features. For example, Figure 2.4, n-grams 1, 2, 3 share the `[pos, next_char]` feature combination. We compute a 64-bit integer representation for each such combination. The value tf_t of a tokenwise feature t is computed as

$$tf_t = hash(t, n, seed_{tok}, pf_1, pf_2, ..., pf_n),$$

where n is the total count of primitive feature components of this tokenwise feature, $seed_{tok}$ is a hash seed value for tokenwise features and pf_i are respective primitive feature values. The hash function is discussed later.

N-gram features combine tokenwise features between several n-gram components into a 64-bit integer. The formula is similar to the one for tokenwise features.

$$nf_j = hash(j, n, seed_{ngram}, tf_0, tf_{-1}, ..., tf_{-n+1}),$$

where j is an index of n-gram feature, n is the order of the n-gram, $seed_{ngram}$ is seed value, tf_k is a value of tokenwise feature in the current path relative to the current position. tf_0 would be the value for the current token, tf_{-1} would point to the previous token, and so on. This enables us to get rid of identical computations.

procedure JppHash(state, input)
  $v \leftarrow$ state $\oplus$ input
  $v \leftarrow v \cdot$ 0x6eed0e9da4d94a4f
  t $\leftarrow$ RotLeft(v, 32)
  $v \leftarrow v \oplus$ t
  return v
end procedure

Figure 2.5: Juman++ hashing algorithm. It operates on 64-bit unsigned integers. $\oplus$ denotes a **XOR** operation.

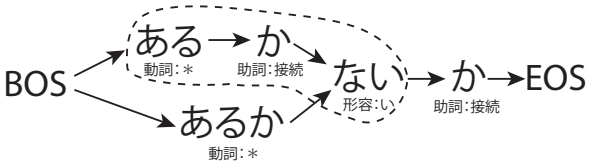


Figure 2.6: Nodes used for the computation of a trigram features of the upper path at the node ない

Hashing

Because Juman++ uses hashing for computing features, we have created a hash function for the task. Usually, the hash functions are designed for hashing strings, operating on sequences of bytes, but it is not the case for Juman++. Instead, it operates on 64-bit integer values, producing a 64-bit result. The basic design of the function is that it takes its current state and a parameter and produces the next state. Hashing multiple values is done by applying the hashing function in sequence:

$$\mathrm{hash}(seed, a, b, c, ...) = \textsc{JppHash}(\textsc{JppHash}(\textsc{JppHash}(\textsc{JppHash}(seed, a), b), c), ...).$$

We adopt a lightweight hash function based on PCG (O'Neill 2014) algorithm for generating pseudorandom numbers, which itself is based on linear congruential generators. The function is shown on Figure 2.5. The total computations consist of two **XOR** operations, one 64-bit multiplication and the right shift. The bit left rotate and the second **XOR** operations improve the randomness of lowest bits, which are used for accessing feature weights. Please refer to the PCG paper for the details and reasoning for this step.

Computation Order

As can be seen, tokenwise feature components of n-grams are consumed from the end to the beginning. This allows us to cut the total number of computations by performing identical parts of computations only once and achieve better cache locality. Figure 2.6 shows an example of a lattice. The node ない has two inbound

paths, let us denote their last trigrams as ある-か-ない and BOS-あるか-ない.

While the overall scores for these paths should be different, they share some features. Namely, unigram features for ない are identical and that part of the score should be identical as well. Additionally, the parts of bigram and trigram hash states are identical too. Let's consider trigram features. $\mathtt{hash}(\mathrm{j}, 3, \mathtt{seed}_{\mathtt{ngram}}, \mathtt{tf}_{\text{ない}}, \mathtt{tf}_{\text{か}}, \mathtt{tf}_{\text{ある}})$ and $\mathtt{hash}(\mathrm{j}, 3, \mathtt{seed}_{\mathtt{ngram}}, \mathtt{tf}_{\text{ない}}, \mathtt{tf}_{\text{あるか}}, \mathtt{tf}_{\mathrm{BOS}})$ share prefixes until $\mathtt{tf}_{\text{ない}}$ and thus will result in identical computations when computed naively. In contrast to that, we break the computation of n-gram features into steps, memorizing intermediate hash function state and sharing it between n-gram features with identical prefixes.

The order in which we perform feature and score computation at first glance is the following:

1. Compute primitive features.
2. Compute tokenwise features.
3. Compute 1-gram features and respective parts of 2,3-grams.
4. Compute score components which are provided by 1-gram features
5. Finish computing 2-gram features, respective score components and respective pars of 3-gram features
6. Finish computing 3-gram features and scores

We compute scores for all nodes starting at a particular character boundary at the same time. Because of the struct-of-arrays layout for lattice structures, feature values for nodes of a boundary are stored in a single array. When performing computations in the specified order, we minimize the number of jumps between boundaries and have very high data locality in computations.

Code Generation

Juman++ has two versions of feature computations: *dynamic*, focusing on the clarity and correctness of the implementation and *static*, focusing on the execution speed. The dynamic implementation models the feature computation process using the object-oriented paradigm. Each of the feature kinds (primitive, tokenwise and ngrams) is modeled as an interface with corresponding implementations using the C++ inheritance and polymorphism. Because the polymorphism uses the virtual function dispatch, this results in a long sequence of indirect function calls, with each function being very short. A long sequence of indirect calls from a single call site, even non-changing between iterations, strains the branch predictor and can become

a sequence of branch mispredictions resulting in a large overhead over the actual feature computation costs.

To achieve high performance, we provide static feature computation: a way to generate C++ code[4] which implements feature value and score computations inline, as defined by a spec. The code generation pursues the following main goals:

1. Enable the compiler to perform the feature-specific optimizations.
2. Exploit asynchronous and out-of-order nature of modern CPUs to mask the cost of random feature accesses.
3. Remove the overhead caused by indirect function calls.

While emitting the C++ code for feature computation and scoring automatically solves the third goal, the ways to achieve the first two and effects from them are discussed in more depth.

Enabling compiler optimizations

Our computation of tokenwise and n-gram features contain constants as the initial arguments to the hashing process. In the case of static feature computation, it opens the possibility for the compiler to perform the constant folding optimization, reducing the number of computations we need to perform at the runtime. An early V2 prototype performed the feature computations in the opposite order: constants after the values from the lattice nodes. By ordering the constants first we observed a significant decrease in analysis speed because of constant folding optimizations. While the constant folding was probably the most easily observable effect of compiler optimizations, inlining the logic also allows the compiler to perform other optimizations, for example, reordering code to improve register utilization.

Interleave Feature Computations and Memory Access

Because we use hashing for computing features, the model weight access pattern becomes random. Random access means that every data access is a cache miss with high probability. Fortunately, modern CPUs provide ways to **prefetch** data from memory to cache. User-controlled prefetching works by issuing special instructions, which are a no-op on the architectural level (so the computation result is the same), but communicate a hint to the CPU that the pointed memory is going to be accessed soon. Code-generation allows us to effectively exploit prefetching to effectively remove cache misses, or reduce their effect, during the score computation. We

[4] An example of generated code for Jumandic can be seen at https://git.io/fjvpy

```
for node n at boundary do
    s_{n-1} ← 0
    for feature f in active ngram features do
        compute value of f(n)
        s_{n-1} ← s_{n-1} + w_{f(n-1)}
        prefetch w_{f(n)}
    end for
end for
```

Figure 2.7: Skeleton of code-generated kernels

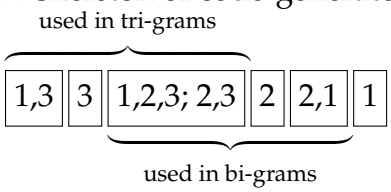


Figure 2.8: Ordering of stored tokenwise features

also take into account the out-of-order execution capabilities of modern CPUs by splitting the overall logic into multiple dependency chains.

We generate code using the skeleton shown in Figure 2.7. The main idea is to create a temporal delay between the computation of the actual feature value $f(n)$ and its usage for accumulating the weight contribution to the linear model score s_n. We accumulate the weight for the *previous* lattice node inside the boundary $n - 1$, while computing the features for the current node n. Our implementation stores n-gram feature values in two buffers: one for the previous lattice node and one for the current one. Reordering code and utilizing prefetching gave about 2× speedup over the prototype without these features.

We also localize the tokenwise feature access when computing n-gram features. Namely, the tokenwise features are stored with the order shown in Figure 2.8. The numbers denote the rank of n-gram features that use the tokenwise features. Because the computation of bi-gram and tri-gram features happen separately, this ordering results in reduced CPU cache pressure.

In the case of unigrams, we additionally combine and interleave the first four steps of feature and score computation (primitive, tokenwise, n-gram feature components and scoring, see 2.3.4). This results in a larger temporal delay between the n-gram feature weight prefetch operation and the actual weight access and reduces the time an access operation needs to wait for the weight in the CPU reorder buffer. The generated code can be seen in the example at `https://git.io/fjvpy` from the line 944. The generated code for the first feature with annotations is shown in Figure 2.9.

Finally, interleaving opens a door for reducing the number of tokenwise features we need to store. In a spec there usually exist tokenwise features that are used

```
// pattern feature #8 (with unigram), usage=3
/* usage is a bitmap, this feature used by 1-gram (1) and 2-gram (2) */
constexpr jumanpp::core::features::impl::CopyPrimFeatureImpl pfobj_surface_{0};
/* constexpr forces the object to be constructed at the compile time. It is either
   completely optimized away or stored in the constant section of the binary. */
::jumanpp::u64 pf_surface_0 = pfobj_surface_.access(ctx, nodeInfo, entry);
/* primitive feature access */
auto fe_pat_hash_8 = ::jumanpp::util::hashing::FastHash1{}
  .mix(8ULL).mix(1ULL).mix(34359656763621376ULL);
/* Prefix of tokenwise feature: index, number of components and seed value */
fe_pat_hash_8 = fe_pat_hash_8.mix(pf_surface_0);
/* Mixing in primitive feature to the hash state */
::jumanpp::u64 fe_pat_8 = fe_pat_hash_8.result();
/* Finalizing tokenwise feature */
constexpr ::jumanpp::core::features::impl::UnigramFeature fng_uni_0_{0, 0, 8};
/* Instantiating the 1-gram (surface) feature.
   Number in the name of the n-gram feature corresponds to the order in the spec. */
auto value_fng_uni_0_ = fng_uni_0_.maskedValueFor(fe_pat_8, mask);
/* Instantiating n-gram feature */
float score_part_0 = weights.at(buf2.at(0)); // perceptron op
/* Computing linear model score component. The weight should be prefetched. */
weights.prefetch<::jumanpp::util::PrefetchHint::PREFETCH_HINT_T0>(value_fng_uni_0_);
/* Prefetching operation */
buf1.at(0) = value_fng_uni_0_;
/* Storing the computed n-gram feature value to a temporary buffer. */
patterns.at(8) = fe_pat_8;
/* storing the tokenwise feature value because it is used by 2-grams */
```

Figure 2.9: Annotated code of interleaved computation. Annotations are in `/* comment blocks */`.

only by unigram features. In the Jumandic-based model, about 30% of tokenwise features are like that. Such tokenwise features will not be used by *other nodes* in n-gram features, so we can skip storing them for the current node (when using code-generated scoring and feature computation logic). Their values never leave CPU registers, so this reduces overall CPU data cache pressure and analysis memory usage.

Our computations perform the score computations in the floating point without quantizing the weights like MeCab. Because weight access is explicitly asynchronous in Juman++, the registers[5] which are used for weights would be stored for a long time in the CPU reorder buffer. At the same time, feature hashing has rather high register pressure and high throughput because of high cache utilization, we would like those registers to be quickly reused. Modern CPU architectures often have two sets of physical registers: for integers and for floating points. Using the

[5]We are talking about microarchitectural physical registers instead of architectural logical registers in this paragraph. For example, Intel Skylake microarchitecture has 180 integer physical registers and 168 floating point registers, but only 16 logical integer scalar registers and 16 vector registers (they can be both integer and floating point). See https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client) for details.

floating point arithmetic for weights allows the CPU to efficiently re-utilize integer registers during the hashing while keeping the weights in the floating point registers.

2.3.5 Unknown Word Handling

Dictionary-based morphological analyzers cannot contain every possible word in their dictionary. Instead, they devise means to handle unknown words. Usually, unknown words are handled with rules, e.g. grouping characters of the same type and using those entries as regular lattice nodes. The lattice search algorithm would then treat the unknown nodes in the same manner as the dictionary nodes for scoring purposes.

Juman++ follows the rule-based approach. The exact rules are defined in the spec using rule constructors. We implement five types of constructors:

1. A single character of a specified type. This type of rule is used for symbols that do not correspond to any word. It is a good idea to have a catch-all rule of this type in the model because otherwise, Juman++ could fail to build a lattice for some input strings.

2. A character sequence sharing the same character type. This is the most used type of rule for defining "regular" unknown words.

3. A number. This constructor handles numbers, written with Kanji in literal (一万五千三百五十二) and numeric (一五三五二) styles and digits (15324). Kanji numbers in numeric styles can be separated into groups of three digits with the centered dot (一五・三五二) and digit numbers can be separated with commas (15,324).

4. An onomatopoeia pattern (Sasano, Kurohashi, and Okumura 2013). We define the patterns as ABAB, ABCABC, and ABCDABCD where A, B, C, D are characters of the same and specified character class.

5. A normalized dictionary entry. Following Sasano, Kurohashi, and Okumura (2013), we also add nodes which can be produced from the dictionary by inserting repeated characters (痛い→痛いいい), prolongation (はい→は〜い) or small "tsu" (かたい→かったい).

Primitive Feature Handling

To be able to score unknown nodes in the same manner as dictionary nodes, unknown word handlers need to provide the same primitive features as there exist for dictionary-based lattice nodes. The first four constructors use dictionary patterns:

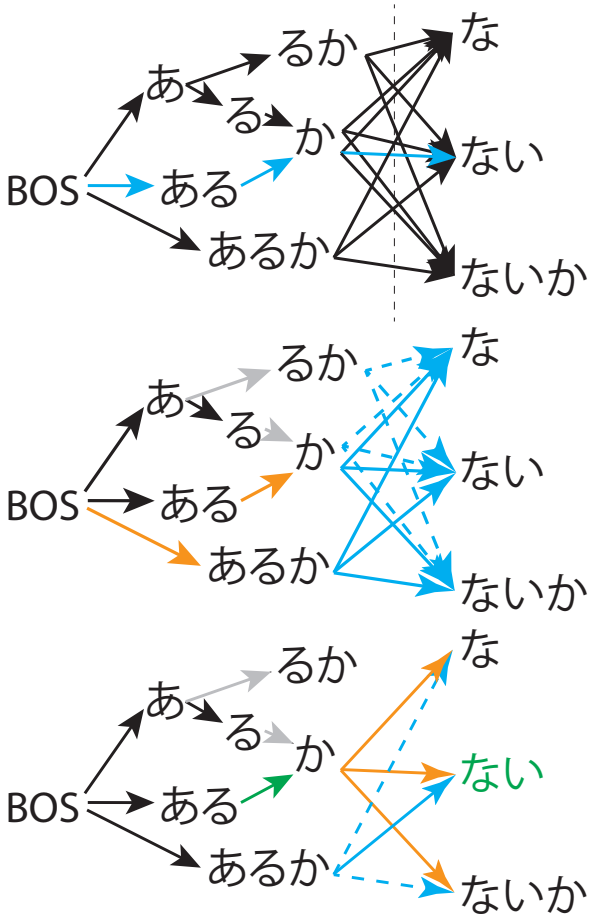


Figure 2.10: A step of path search at a character boundary. Top: full beam, middle: left trim ($b_l = 2$), bottom: right trim ($m_l = 1$, $m_r = 1$). Trimmed connections are dashed.

special dictionary entries that are not indexed for surface lookup. The primitive features for unknown nodes, created by these handlers, are initialized from dictionary patterns and then a subset of primitive features, configured in the Spec, is replaced by a hash value of a surface string corresponding to the unknown lattice node. In the Jumandic we replace surface, the base form, and reading with the actual surface string of the unknown handler. In the case of normalization, we use the dictionary entry which acts as the normalization result as the pattern, keeping the rest of handling the same.

Handlers also can communicate with the scoring procedure by setting values of *placeholder* primitive features. For example, the character sequence handler would set 1 to the placeholder in the case if the current unknown word contains a dictionary word as a prefix.

2.3.6 Beam Search and Trimming

Searching for the top-scored path in the lattice is done on a character boundary basis. For each boundary, there are *left* lattice nodes, which contain words ending at the current boundary and *right* nodes, which contain words starting from the current boundary. Left nodes also contain paths ending at those nodes. A search step grows those paths, adding connections that cross the boundary between left and right nodes. The process is illustrated on Figure 2.10.

Because Juman++ uses trigram features, the usually employed Viterbi search will have $O(n^3)$ computational complexity on each segmentation boundary, which will completely overturn all benefits we gain from optimization. We employ the beam search instead. V1 uses node-local beams of width b_f, meaning that it keeps b_f paths with the top scores are kept for each lattice node. This configuration will be referred to as *full beam*.

Full beam considers every combination of *left* (ending on the boundary) and *right* (starts from the boundary) nodes on each character boundary. For each left node, it also considers the paths in the respective beam. This setup conservatively assumes that it is impossible to compare the scores of paths which end at different nodes, and has a high computational cost. Still, in our experiments, most of the sentences contain several boundaries where there exist around 20-30 of both left and right nodes. In the full beam setup the computational complexity scales as the product of a number of left and right nodes at a character boundary. Moreover, the majority of the combinations are useless and can never be the correct analysis result.

To increase the analysis speed we have implemented more aggressive beam trimming. The first improvement is straightforward: instead of using all paths from all the left nodes, we form a global beam of width b_l for the boundary. The path scoring process can use the formed global beam instead of all the local beams. In this setting, the path scores become comparable for the identical sequences of surface characters.

Using the global beam for the left side of the boundary greatly helps to reduce the number of computations during the scoring. The number of right nodes can still be large. We further decrease the number of computations by ranking the right nodes and performing the full computations only to the perspective nodes. Namely, we rank the right nodes by evaluating their scores in the connection to top m_l paths from the left global beam. Then we evaluate the scores for the remaining paths, but we use only top-ranked m_r right nodes.

The full algorithm for this setting (which we refer to as *trimmed beam*) is:

1. Form the *global beam* of width b_l from the local beams of left nodes (trimmed beam: left)

2. Rank the right nodes using the top $m_l (< b_l)$ path candidates from the global beam (trimmed beam: rank)

3. Compute the remaining path candidates over the boundary using the remaining $b_l - m_l$ paths of the global beam and top m_r best right nodes (trimmed beam: right)

4. Form the local beams of width b_f for the right nodes using the results of steps 2 and 3

In our experiments, the trimmed beam configuration showed the same accuracy while having a significantly faster analysis speed. Moreover, it was sufficient to rank the right nodes using only $m_r = 1$ left paths. The accuracy results are discussed in detail in subsection 2.5.4. We also discuss the setting of analyzing out-of-corpus data, which had a larger number of situations when the trimmed beam result was different from the full beam.

2.3.7 RNN Language Model

An RNN language model helps the analyzer to be more accurate by adding additional lexicalization learned from a large-scale corpus. Juman++ uses a recurrent neural network-based language model (Mikolov 2012) to estimate semantic plausibility of different segmentation possibilities. Still, a complete lattice search with RNN is computationally expensive and is one of the reasons for the slow analysis speed of V1. Instead, V2 uses the RNN model only to rerank analysis candidates which remain in the beam of the EOS node after beam search with the linear model.

Juman++ can evaluate RNN not only on the surface string but on any combination of dictionary string fields. We will denote such a combination as **RNN key**. For the Jumandic RNN key, we use a combination of word base form (without conjugations) with a rough part-of-speech tag.

Remember that Juman++ lattice nodes are unique for a combination of primitive features participating in n-gram computations. The RNN model is evaluated only on a subset of those primitive features, which mean that paths traversing through the distinct lattice nodes may have identical RNN key. Naively evaluating all paths lead to unnecessary computations, which decreases overall analysis speed. Because of this, we deduplicate possible RNN paths before performing the actual computations. This greatly reduces the number of computations for the RNN language model.

For out-of-dictionary keys of the RNN model, we use a length-based unknown word penalty instead of the raw RNN scores. Namely, the score is defined as

$$s_r = u_b + l \cdot u_l, \tag{2.3}$$

where u_b is a constant part of an unknown RNN score hyperparameter, l is a count of Unicode codepoints in the surface of the node with an out-of-dictionary RNN key and u_l is a length part of an unknown RNN score hyperparameter.

Because the RNN model is independent of the analysis dictionary, the RNN representation lookup differs from the segmentation dictionary lookup. We build two separate indices for the RNN. The first one contains the concatenated field pointers in LEB128 encoding as the key and embedding id as the value. It is used to handle normal lattice nodes which are completely built from the dictionary. The second one uses the normal strings for the fields which unknown word handlers replace by surface and field pointers for the remaining fields. It is used to handle lattice nodes, which are contained in the language model but not in the segmentation dictionary.

2.4 Training

The training of the linear and RNN models for Juman++ is done independently. The parameters for mixing them are optimized after training both, using a hyperparameter search. We also use the hyperparameter search to optimize the training procedure for the linear model.

Juman++ uses a slightly peculiar data scheduling regime for the training. The full training process is divided into epochs, each of which iterates several times over the training data. The training data is also shuffled before the start of each epoch. The first iteration of each epoch uses the full beam for the analysis and the following iterations use the trimmed beam search. This scheduling improves the analysis stability for out-of-domain data.

Additionally, to reduce the model pollution with features corresponding to bad nodes, the first epoch uses only fully-annotated examples. The partially annotated examples are added to the training only from the second epoch when the model learns to rank perspective nodes higher than bad ones.

2.4.1 Linear Model

Juman++ can train the linear model using both fully-annotated and partially-annotated data. Data in the fully-annotated format define a sentence as a sequence of lattice nodes. Each node must contain all dictionary fields used in n-gram features.

Partially annotated data contain sentences as sequences of characters with additional annotations. Annotations can be of three types: segment, no-segment, and word. Segment annotation forces a boundary between two Unicode codepoints. No-segment annotation is an opposite – it forces *an absence* of a boundary between

two codepoints. Word annotation has two segment annotations at the beginning and the end of the word, no-segment annotations between them, and optional tags that correspond to required values of dictionary fields for matching nodes.

Juman++ uses the Soft Confident Weighted (J. Wang, P. Zhao, and Hoi 2016) online learning algorithm to optimize model weights. For the fully-annotated examples the application is straightforward: we use the combination of all n-gram features in the path as sentence features. We update the model for the sentences where the correct analysis, as specified by the training example, falls off the beam.

We have also tried two early update strategies known to be useful for structured learning: maximum violation and beam falloff when the model is updated only for the subset of parameters starting from the beginning of the sentence and ending at the point of maximum violation (difference between top beam score and gold score) or where the correct analysis falls off the beam. In our experiments, however, the partial update strategies showed worse accuracy than the full update strategy.

In the case of partially annotated examples, we only update features contained in partial violations: codepoint boundaries where the top-scored lattice node does not adhere to the partial annotation rules. As the correct node, we try to find several node candidates that satisfy the partial annotation rules, deduplicating features from them so that the features shared by multiple lattice nodes won't be overweighted.

2.4.2 RNN training

In the current implementation of Juman++, the RNN model is trained completely independently of the linear model. We use the faster-rnnlm software[6] for the RNN model training. It is first trained on a large scale automatically analyzed data and then fine-tuned on the human-segmented data. The resulting model is then used for the hyperparameter tuning.

2.4.3 Hyperparameters Tuning

For the training of Jumandic-based Juman++ models we optimize the following hyperparameters:

- Soft Confident Weighted learning hyperparameters
- Number of training epochs and in-epoch iterations
- RNN mixing parameters
- RNN unknown word hyperparameters

[6] https://github.com/yandex/faster-rnnlm

Table 2.2: Dictionary and model size comparison

Analyzer	Dictionary size (MB)	Model size (MB)
Raw Dictionary	256	-
MeCab	*311	7.7
KyTea	-	200
V1	445	135
V2	158	16

We use the Gaussian process-based Spearmint (Snoek, Larochelle, and Adams 2012) package for the hyperparameter tuning. The search is performed on the average of F1 scores for segmentation and POS-tagging over the 10-fold cross-validation on the training data. We have optimized hyperparameters in two groups: linear model training-related parameters and RNN-related parameters.

2.5 Experiments

The overall performance of a morphological analyzer consists of different factors. We compare the following performance components:

- Dictionary or model size,
- Analysis speed,
- Analysis accuracy.

For comparison, we use JUMAN, MeCab, and KyTea as our baselines.

In all the experiments we use the Jumandic segmentation dictionary with Kyoto University (KU) and Kyoto University Web Document Leads (KWDLC) corpora. JUMAN uses the Jumandic without expanded conjugation forms as it is, for other analyzers, we expand all possible conjugation forms with conjugation rules.

2.5.1 Dictionary Size

The compiled dictionary and model sizes for the different morphological analyzers are shown in the table 2.2. All dictionaries except KyTea's were built using the same conjugation-expanded Jumandic. For KyTea we used only deduplicated entries consisting of surface, POS, and sub-POS fields. V1 and V2 models do not include the RNN model size. We can see that the dictionary size of MeCab is larger than the raw dictionary file; however, that is the size of a trie index. Please note that its index size is a consequence of MeCab being very popular and backward compatible with its models. It is extremely easy to swap its current implementation of a double

Table 2.3: Juman++ Jumandic model size breakdown

Parts	Size, bytes	Model %	Dict %
Double Array Trie Index	33.12 M	8.11%	27.49%
Entry Pointers	7.92 M	1.94%	6.57%
Field Pointers	30.81 M	7.55%	25.57%
String storages			
surface, baseform, reading	38.35 M	9.40%	31.83%
pos	205	-	-
subpos	702	-	-
conjform	2704	-	-
conjtype	860	-	-
canonic	2.87 M	0.70%	2.39%
features	5.17 M	1.27%	4.29%
features (list pointers)	2.23 M	0.55%	1.85%
Linear model	16.00 M	3.92%	N/A
RNN	271.71 M	66.56%	N/A

array trie with the API-compatible implementation darts-clone[7] which will reduce the index size in half. Juman++ V2 uses this implementation. V1 dictionary size is significantly larger than of other analyzers because it was using data structures that are optimized for shared memory interprocess communication for the on-disk storage of the compiled dictionary. KyTea stores the string feature representations and its models become relatively large because of that. For the KyTea model trained on the concatenation of KU and KWDLC corpora, the model size is even larger than the V2 model without an RNN.

We also analyze the size of a V2 Jumandic model in more detail. Table 2.3 contains a breakdown of a V2 Jumandic model[8]. The table shows that the dictionary information itself is represented very compactly. About half of the dictionary space is taken by the double array index. Using a more compact representation of the trie index can make the models even smaller, still, the total model size is dominated by the RNN, thus the resulting size reduction of the whole model will not be significant.

The Jumandic model shares the string storages for surface form, reading, and a base conjugation form which makes the string storage with the most diverse entries. On the other hand, the content of POS-related fields is not diverse, as can be seen by the sizes of their string storages. The on-disk size of field pointers depends on the size of the respective string storage, and together with the field alignment, it is possible to achieve the size of one byte per POS-related field.

[7] https://github.com/s-yata/darts-clone

[8] This information is accessible as a result of `jumanpp --model-info` command for an arbitrary spec.

Table 2.4: Morphological analysis speed comparison

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

2.5.2 Analysis Speed

We used a computer with Intel i7-6850K CPU, 64 GB of RAM, and Ubuntu 16.04 Linux for the analysis speed comparison. For the speed benchmarking we have downclocked the CPU frequency to 3 GHz (from the 3.6 GHz base frequency), disabled the Turbo Boost technology, and used the performance CPU scaling governor which runs the CPU at the fixed frequency. The models were trained from scratch using the same Juman++ dictionary, the Kyoto University[9] (KU) and KWDLC[10] corpora for all morphological analyzers except JUMAN, which is not trainable. For KyTea we also report the throughput of Unidic-based models which also perform the reading estimation, and are available for download from the KyTea website. A Jumandic-based model for KyTea does not use the reading estimation feature of KyTea and it was learned using the 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.

Table 2.4 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. We use the median time of five launches with identical parameters for computing the analysis speed. V2 noRNN is only 20% slower than JUMAN while having a considerably complex model. V2 RNN has 1.8 times the execution speed of JUMAN and is more than 250 times faster than V1.

2.5.3 Analysis Accuracy

Figure 2.11 shows F1 scores with 95% confidence intervals for both the KU and KWDLC corpora. Data in the tabular format is in Appendix 2.8. A concatenation of training sections of both corpora was used to train a combined model; the reported

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

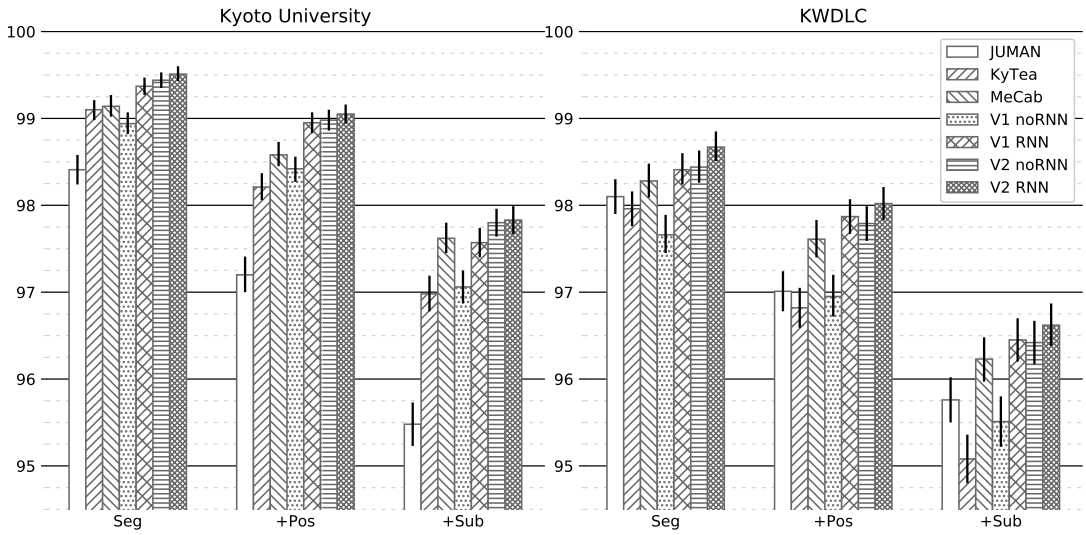


Figure 2.11: F1 scores and 95% confidence intervals for accuracy of morphological analyzers on Jumandic-based corpora. Seg is segmentation; +Pos is correctly guessing the POS-tags after segmentation.

scores are for the test sections. MeCab and V2 have hyperparameters optimized using Spearmint (Snoek, Larochelle, and Adams 2012). The confidence intervals are computed using bootstrap resampling with 10,000 iterations. Note that if sides of two confidence intervals overlap by half, it means the statistical significance of $p = 0.036$, and non-overlapping intervals mean statistical significance at least of $p = 0.006$ (Cumming and Finch 2005).

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 a 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.

2.5.4 Effects of Beam Trimming

We evaluate the effects of beam trimming on analysis accuracy by making an analysis experiment with different trimmed beam settings. We train multiple Juman++ models using different trimmed beam settings and compare their accuracy to each other and the full beam setting. For the experiment, we use 10-fold cross-validation on the train section instead of the usual test-train split.

Figure 2.12 shows the average of the POS F1 score of different trimmed beam

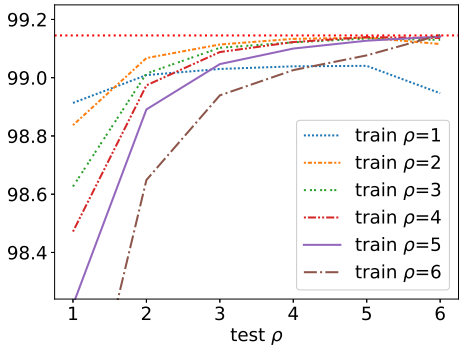


Figure 2.12: Cross-validation test F1 score average on KU corpus for POS tags when using different timmed beam parameters $b_l = m_r = \rho$, $m_l = 1$. Dotted line at the top is F1 score in the full beam setting.

settings. Generally, when using the larger or equal beam size in test time than in the training time, the accuracy is not distinguishable from the full beam setting. We should note that training the model in the setting $\rho = 1$ (equivalent to Viterbi search on bigrams) fails to utilize the high-order features efficiently and achieves lower accuracy.

We also noticed that there exist sentences when the full and trimmed search configurations do not agree on the top-scoring path when analyzing random web sentences. 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% of sentences had different analysis results, a relatively small number. Of those sentences, the full beam analysis was correct only in around 50% of the cases. The rest of the sentences had both 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.

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 2.12 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 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, Opper, and Sompolinsky 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.

2.6 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.

Figure 2.13: Annotation tool: diff view. UI explanations are in blue. Variant 1 is selected.

2.6.1 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 2.13. 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 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 2.14, 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 pro-

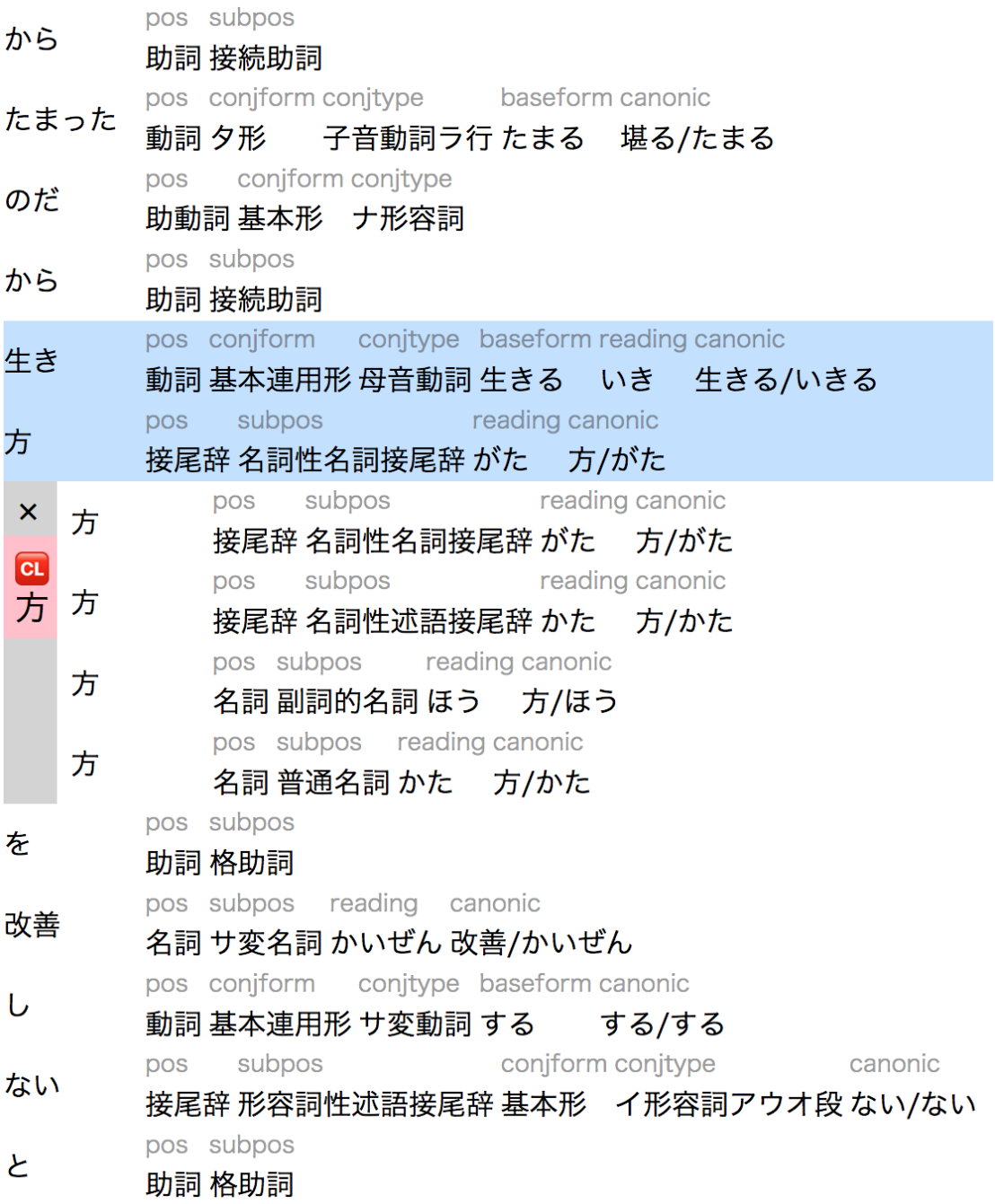


Figure 2.14: Annotation tool: interactive analysis. Constraint nodes background is blue. Other analysis variants for the last constraint node are displayed after the gray indent. The "CL" button removes a constraint.

vide 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.

2.6.2 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 subsection 2.5.4. 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.

2.7 Discussion

While Juman++ achieves very high accuracy, there still exist some errors. We would like to discuss the most frequently occurring types.

Segmentation errors are the most visible ones for the users of MA. Juman++ often incorrectly segments katakana words. For example, ギネス|ビール, ストップ|ウオッチor シベリアン|ハスキー are written in the corpus as single words. In these cases, the whole word is composed of the subcomponents and it is difficult to say that Juman++ segmentation is critically incorrect. However, there are also cases of over-segmentation like ブル|ドーザー and ショベル|カー when the combined word is not composed of its parts and the segmentation does not make sense.

Another frequent case is an ambiguity in segmentation caused by a different part of speech. In Jumandic segmentation standard usages of na-adjectives when modifying verbs are not segmented (優位に|動く), but are segmented when に is a case marker (優位|に|立つ). This problem also manifests when the word is katakana. For example, Juman++ incorrectly treats リベラル as an adjective in the sentence 民主党リベラル|に|よる高福祉. There exist very tricky cases like 準々決勝で|伊達|に|敗れ in a sentence about a sporting event. It is possible to decide the 100% correct segmentation only after checking the fact that a person named 伊達 participated in

the event.

There exist compound words which can be segmented in different ways: 不快|感or 不|快感; 工学|部or 工|学部. While a non-corpus segmentation is registered as a segmentation error, this is more a problem of the segmentation standard than of an analyzer.

Lastly, there is a large number of over-segmented named entities like 新|民連 or デルタ|航空, especially in texts of the web domain. Sequences of characters of the same type are not as problematic as the entities of the second kind, containing characters of multiple classes (e.g. kanji and katakana or katakana and romaji). They could not be solved in the current paradigm of unknown word handling well. Nevertheless, we would like to argue that the named entities frequently consist of several morphemes. Here we have a conflict of how easy it is to use the results of the morphological analysis in other applications (handling sequences of tokens is more difficult than handling single tokens) versus the closeness to the uniformity of the tokenization. Unfortunately, the Jumandic does not make a clear decision at this point so conflicting situations arise in the segmentation.

Non-local Dependencies

One group of remaining accuracy problems is when the correct decision requires non-local information. The most frequent, albeit important such problem is the ambiguity between で being either a conjugation of copula だ or the instrumental case marker. For example, in the sentence "この湯豆腐と、あえ物、ごま豆腐、おひたしなどの精進料理で、体のしんまでポッカポカ。" で is the case marker, but the decision requires to take the overall sentence into the account. Moreover, punctuation does not help to make this decision.

Analyzers that make the local decisions are poorly fit to solve this problem. One possible approach would be to retain the ambiguity in the morphological analysis result and try to resolve this problem at the syntactic parsing level. It is, still, difficult to decide what kind of ambiguity to keep and what to pass. There was an attempt to leave this ambiguity on the morphologic analysis level and try to resolve it as a syntactic parsing problem (D. Kawahara, Hayashibe, Morita, and Kurohashi 2017), but there was not much success. Accessing the global information about the sentence is key to solving this problem, so neural approaches could be helpful, but it was not thoughtfully studied yet.

Linear Weights Precomputation

While the current version of Juman++ does not precompute parts of linear model weights corresponding to unigram and bigram features to speed up computations, similar to MeCab, it is possible. Still, not every feature can be precomputed, mostly because Juman++ allows us to use surface-based features which depend on the input string at the analysis time. Taking that into consideration, the design should give the user the ability to control the size of the precomputed weight table. Ultimately, this leads to significantly increased complexity of scoring and statically generated code. Also, the statically generated scoring procedure is going to depend on the decisions made at the precomputation time, compared to the current state when it depends only on the spec. Because of these considerations, the current version of Juman++ does not contain the functionality to precompute linear model weights.

It is, however, incorrect to think that Juman++ does not perform any type of precomputations. Converting the dictionary into a form that is easy to process is a form of precomputation. We additionally resolve conditional statements at the dictionary compile time.

Using SIMD and Vector Instructions

While Juman++ exploits out-of-order capabilities of modern CPUs, we do not exploit the SIMD parallelism in the linear model. RNN implementation uses Eigen library (Guennebaud, Jacob, et al. 2010) and is vectorized.

We did some experiments with vectorizing the feature and score computation; however, the scalar version was faster when using SSE/128-bit vectors and on the same level of performance for AVX2/256-bit vectors. We did not experiment with AVX-512 because of the lack of easily accessible AVX-512 hardware at the active development time of Juman++. One of the problems with them is the high latency of gather-type instructions needed for random access of model weights and the lack of gather-prefetch instructions. It would be interesting to explore this direction in the future.

Portability

The techniques proposed here can be divided into two parts: architecture-dependent and architecture-independent. Everything related to the dictionary structure is architecture-independent. Other parts make assumptions about the underlying hardware, but most modern and widely used hardware satisfies our assumptions.

Linear congruent generators, which our hash function is based on, have better statistical properties when they have larger state space, and our implementation uses

64-bit integers because the most widely used architectures are 64-bit and provide a large number of 64-bit integer registers with fast operations on such numbers. The algorithms should work with the 32-bit intermediate hash state, but we did not test it.

The rest of the improvements assume that the executing CPU will have out-of-order execution. Having a complex memory subsystem that supports prefetching and multiple asynchronous and simultaneous memory accesses usually stems from the out-of-order nature. Fortunately, nearly every modern CPU aimed at performance, in both server, desktop, or mobile form-factors, has out-of-order execution. As far as we know, only extremely low-power or simple microcontrollers and CPUs with special needs (e.g. ARM Cortex R series which have fixed latency for all instructions) are in-order. We believe that Juman++, in general, would be a bad match for tiny embedded systems not only because of applied optimization methods but also because such CPUs usually are not packaged with a large amount of RAM. Software prefetching, however, sometimes is supported on in-order CPUs as well, and can be used to improve the execution speed in such situations.

Juman++ and its techniques target the common characteristics of out-of-order CPUs, like asynchronous memory access, without being optimized for a specific microarchitecture. There is nothing prohibiting optimizations of Juman++ to work on ARMv8+/PowerPC CPUs in addition to x86_64 without difficult porting. Finally, we must remark that approaches to computer architecture like VLIW which put the decisions over instruction-level parallelism on the programmer and compiler could also be exploited similarly.

Low Accuracy on Sentence Level

While there exists an impression that the morphological analysis has extremely high accuracy (e.g. 99.5% F1 score for segmentation of newspaper texts), there exist some caveats. The evaluation is done on a token level and for Japanese, there exists a large number of *easy* tokens, mostly related to the grammar. Even JUMAN, as an example of an analyzer that uses an HMM model for analysis with mostly bi-gram features, gets the segmentation score above 98% F1. We should note that greedy matching prefix dictionary strings do not perform well in the segmentation task and achieve only a 79% F1 score. For suffixes (analyzing the end to the beginning) it is even worse with 68%.

On the other hand, when it comes to the sentence level the accuracy is not that high. For the KU corpus, it is 93% (127/1783 contain segmentation errors) and for the KUDLC it is 88% (265/2195). Moreover, for more complicated and noisy domains like user-generated texts in social media, it is even lower.

Being Robust to Input Errors

Most of the current morphological analyzers treat the input in a way so it does not contain errors. Unfortunately, real word texts contain typos and other linguistic phenomena. Some of them, like IME misconversions, are mostly Japanese-specific.

Lattice-based analyzers can't handle such situations well. For example, when the misconversion contains actual dictionary words, the analyzer would use those, producing nonsense analysis. Still, humans are pretty robust against such errors and can infer the original meaning of the sentence pretty easily. There exist prior work in this direction (Saito, Sadamitsu, Asano, and Matsuo 2014), but it is one of the open problems for the morphological analysis.

Approaches With and Without Dictionary

The current state of morphological analysis accuracy is happened because of the high quality of segmentation dictionaries for Japanese. Also, from the user's point of view, it is easy to tune the analyzer for their task or domain by adding new words to the dictionary. Because the MeCab-based analyzers use primarily POS-based features for scoring, adding new words could be done without retraining the whole model. This, however, makes their models relatively large for large dictionaries.

Juman++ , in theory, allows adding new words to the analysis dictionary. If we add new content to the end of string storages we will not change values of present field pointers, so the feature hash values will stay the same. The field pointer table itself can be completely rebuilt without retraining the linear and RNN models. This functionality, however, is not implemented presently because of time constraints.

Tolmachev, D. Kawahara, and Kurohashi (2019) bootstrapped a neural network-based morphological analyzer based on a large-scale corpus analyzed by a traditional dictionary-based analyzer. Their approach did not model the dictionary explicitly, predicting segmentation and POS from the encoded unigram representations. Their approach, nevertheless, matched and in some cases surpassed the bootstrapping analyzer, having a very compact model. This approach seems to learn a *word model*, preferring to output some tokens which make sense to humans, but have different segmentation in the corpus, e.g. 食材 which is segmented into two characters in the corpus. The downside to their approach is that adding new words to the model requires reanalyzing a huge corpus and then retraining the model, making the process very cumbersome.

Table 2.5: F1 scores and 95% confidence intervals for accuracy of morphological analyzers on Jumandic-based corpora. Seg is segmentation; +Pos is correctly guessing the POS-tags after segmentation.

	Kyoto University			KWDLC		
Analyzer	Seg	+Pos	+Sub	Seg	+Pos	+Sub
JUMAN	$98.41^{98.58}_{98.24}$	$97.20^{97.40}_{96.99}$	$95.48^{95.73}_{95.23}$	$98.10^{98.30}_{97.90}$	$97.01^{97.24}_{96.78}$	$95.76^{96.02}_{95.50}$
KyTea	$99.10^{99.22}_{98.99}$	$98.21^{98.36}_{98.05}$	$96.98^{97.18}_{96.77}$	$97.96^{98.16}_{97.76}$	$96.82^{97.05}_{96.59}$	$95.08^{95.36}_{94.80}$
MeCab	$99.14^{99.26}_{99.01}$	$98.58^{98.71}_{98.43}$	$97.62^{97.79}_{97.44}$	$98.28^{98.47}_{98.08}$	$97.61^{97.82}_{97.39}$	$96.23^{96.49}_{95.98}$
V1 noRNN	$98.94^{99.06}_{98.81}$	$98.42^{98.57}_{98.28}$	$97.06^{97.25}_{96.87}$	$97.66^{97.87}_{97.43}$	$96.95^{97.18}_{96.70}$	$95.51^{95.80}_{95.22}$
V1 RNN	$99.37^{99.47}_{99.27}$	$98.95^{99.07}_{98.83}$	$97.57^{97.74}_{97.40}$	$98.41^{98.58}_{98.22}$	$97.87^{98.07}_{97.67}$	$96.45^{96.70}_{96.20}$
V2 noRNN	$99.44^{99.53}_{99.35}$	$98.98^{99.10}_{98.86}$	$97.80^{97.96}_{97.64}$	$98.44^{98.62}_{98.25}$	$97.79^{97.99}_{97.59}$	$96.42^{96.67}_{96.17}$
V2 RNN	$99.51^{99.59}_{99.42}$	$99.05^{99.16}_{98.94}$	$97.83^{97.99}_{97.67}$	$98.67^{98.83}_{98.49}$	$98.02^{98.21}_{97.83}$	$96.62^{96.86}_{96.37}$

2.8 Accuracy Comparison Tabular Data

See Table 2.5. Each table entry consists of three numbers. The left number is the F1 score on the test data. Top right and bottom right numbers are upper and lower confidence interval bounds, respectively.

2.9 Some Notes on Computer Architecture

Modern computers are complex. While the speed of CPUs was increasing very fast, memory latency did not increase that much. Because of this, modern CPUs have multi-level cache hierarchy with different access speeds. The CPUs also execute multiple instructions at the same time. When utilizing ineffective hardware utilization does not change the correctness of a program, it can drastically change the execution speed. We use some tricks that explicitly target the details of modern CPUs in the implementation of Juman++. This section provides some simplified details on modern computer architecture which are required to explain our decisions for Juman++. Juman++ for the lattice adopts struct-of-arrays data layout, which is described in this section.

2.9.1 CPUs and Memory

Modern computers have complex memory hierarchy and different memory access patterns can have different execution speeds. Figure 2.6[11] shows approximate access

[11] https://gist.github.com/jboner/2841832

Table 2.6: Approximate latencies of the modern hardware

Category	Latency	Comment
L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	14× L1 cache
Main memory reference	100 ns	20× L2 cache, 200× L1 cache
Read 1 MB sequentially from RAM	250,000 ns	
Read 1 MB sequentially from SSD	1,000,000 ns	4× memory
Disk seek	10,000,000 ns	
Read 1 MB sequentially from disk	20,000,000 ns	

latencies of different parts of modern hardware. While the absolute numbers are not precise, the orders of magnitude are correct.

It is easy to see that cache accesses are much faster than accesses to main memory, nevertheless the external storage. However, cache sizes are very limited. Modern Intel CPUs have 32 KB of L1 data cache per execution core, organized by 4096 512-bit cache lines. L2 cache is 256KB per core and is shared for code and data. L3 cache is usually shared by all the CPUs and the number is around 1.5MB per core for server CPUs.

When a CPU executes an instruction accessing the memory, the data is first read from the L1 cache, if it does not contain the requested data then the request is forwarded to the L2, and then similarly to the L3. Only if the data is not contained in the L3, the access is forwarded to the main memory. The computation continues only after the data is fetched through all the cache levels. Data is evicted from caches based on a least frequently used heuristics. This means that accessing the same or close data will hit the cache, but random accesses will miss.

Also, it is possible to *prefetch* the data. By issuing a special instruction, we give the CPU a hint that a piece of data will be accessed soon, so it should get it from the memory and put it into the cache. Prefetching can be used to hide the latency of the main memory. In addition to *manual* prefetching, CPUs also detect frequently-used access patterns and automatically prefetch the data into the cache. Automatic prefetching is more effective because it does not require to dispatch additional instructions, but it can not detect random access patterns.

Modern high-performance CPUs are also *out-of-order*. They do not execute instructions sequentially (in-order), and they can execute multiple instructions each cycle. Instead, they convert instructions into a directed acyclical graph-like structure and traverse it in parallel. Inputs to an instruction become its dependencies in the graph. Converted instructions are placed into a reorder buffer. When all the inputs to the instruction in the buffer are ready, it is executed. For example, Intel Haswell

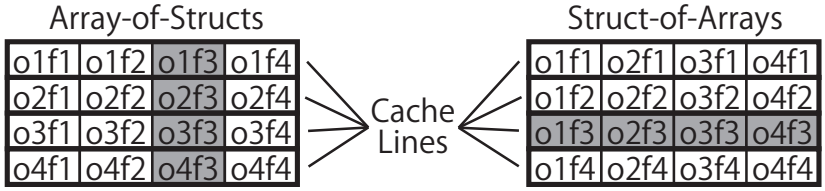


Figure 2.15: Illustration of stuct-of-arrays and arrray-of-structs layouts of a 4-field object in a 4 element array. Gray is an accessed field. When using array-of-structs layout, accessing *f3* of all objects uses all four cache lines, however in stuct-of-arrays layout accesses are localized in a single cache line.

microarchitecture has a reorder buffer of size 192[12], meaning that there could be 192 instructions at the given time "in flight" and it can execute up to 4 instructions each cycle.

2.9.2 Memory Layouts and Cache Efficiency

Almost every programming language provides a means to pack the data into *structures* as a way to produce abstractions. They work by joining the data, possibly of a different kind, together. Struct data is usually laid sequentially after each other by language compilers and runtimes as well. When working with multiple similar data, it is often put into arrays: homogenous sequences of the data. This data layout is called *array-of-structs* (AoS).

Structs are an important building block of abstractions and are irreplaceable for building complex software. Still, structs can be large and when performing the array processing, programs tend to operate only on a subset of each struct. On the other hand, CPU caching works in cache line granularity. A CPU fetches data from the memory in the relatively large packets of fixed size (on the x86 architecture it is 512 bits — 64 bytes). If an application uses large structs (e.g. larger than a cache line) and touches only a single 32-bit sized integer field from each of them, this means that the *effective* cache utilization is very low (only 6.25% is used, the rest is wasted by the non-touched data) and there is a large number of main memory accesses, which are much slower than cache accesses.

Of course, this is an example of a pathological case and in the real world, the cache efficiency is not as bad. Still, high-performance code often adopts *struct-of-arrays* (SoA) data layout instead of usual array-of-structs. With this layout, the fields are packed together in an array for each field of the structure. Each struct becomes decomposed over several arrays. The difference between AoS and SoA layouts for a

[12] https://en.wikichip.org/wiki/intel/microarchitectures/haswell_(client)#Execution_engine

struct of 4 fields in a 4 element array is illustrated in Figure 2.15. With AoS layout the access patterns which touches only a subset of the fields at a time utilize the CPU caches much effectively. Moreover, hardware prefetchers of modern CPUs efficiently support SoA access patterns.

3 Fully-Neural Morphological Analysis

3.1 Introduction

Although some NLP applications, like neural machine translation, started to use unsupervised segmentation methods (Kudo and Richardson 2018), resulting segmentation often has decisions which are not natural to humans. Supervised segmentation based on a human-defined standard is essential for applications which are designed for interaction on a word-level granularity, for example, full-text search. Segmentation is commonly done jointly with part of speech (**POS**) tagging and usually referred to as Morphological Analysis.

Modern Japanese Morphological Analyzers (**MA**) are very accurate, having a >99 segmentation tokenwise F1 score on news domain and a >98.5 F1 on web domain (Tolmachev, D. Kawahara, and Kurohashi 2018). They often use segmentation dictionaries which define possible words. Also, their models are generally large and unwieldy, spanning hundreds of megabytes in case of traditional symbolic feature-based approaches. Neural models with word or n-gram embeddings are even larger, easily reaching gigabytes. This makes it difficult to deploy MA in space-constrained environments such as mobile applications and browsers.

It has been shown that simple or straightforward models can match or outperform complex models when using a large number of training data. For example, a straightforward backoff technique rivals a complicated smoothing technique for language models (Brants, Popat, P. Xu, Och, and Dean 2007). Pretraining a bidirectional language model on a large dataset helps to solve a variety of NLP tasks (Devlin, M.-W. Chang, Lee, and Toutanova 2019). Our approach is inspired by this line of work.

We propose a very straightforward fully-neural morphological analyzer which uses *only character unigrams* as its input[1]. Such an analyzer, when trained only on human-annotated *gold* data has low accuracy. However, when trained on a large amount of automatically tagged *silver* data, the analyzer rivals and even outperforms, albeit slightly, the bootstrapping analyzer. We conclude that there is no need for rich input representation. Neural networks learn the information to combine characters

[1]The source code is avaliable at https://github.com/eiennohito/rakkyo

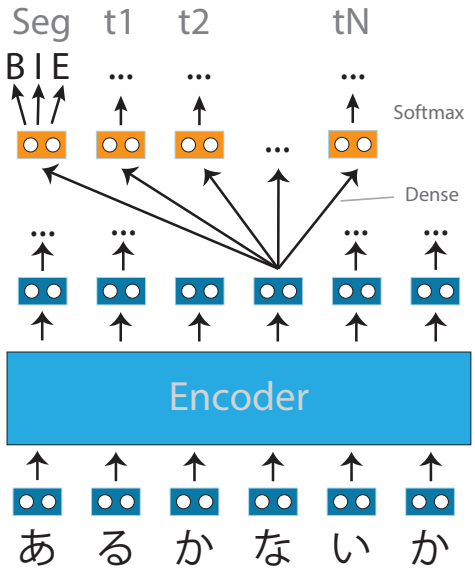


Figure 3.1: Proposed model. We encode character unigram embeddings into shared representations for each character. The shared representation is projected into a tag-specific representations from which we independently infer segmentation and per-character tags.

into words by themselves when given enough data.

Ignoring explicit dictionary information and rich input representations makes it possible to make analyzers that are highly accurate and very compact at the same time. We also perform ablation experiments which show that the encoder component of such an analyzer is more important than character embeddings.

3.2 Proposed Approach

In order for MA to be practical, it should be not only accurate, but also fast and have relatively compact models. The speed of search-based approaches is dependent on how computationally heavy a weighting function is. Heavyweight models, like neural networks, require a large number of computations, and we think that it will be very difficult to create a practical search-based fully NN morphological analyzer with analysis speed comparable to traditional analyzers.

We do not want to use any explicit information about how to combine characters to form a word, like dictionaries, which takes space and is not trivial to incorporate into a character-based model. We also want our model to be fast, at least comparable with the speed of traditional analyzers. To this end, we follow a pointwise approach and force the neural network to learn the dictionary information from a corpus.

We use a straightforward architecture shown in Figure 3.1. We embed each character, and then apply an encoder, which produces an encoded representation for

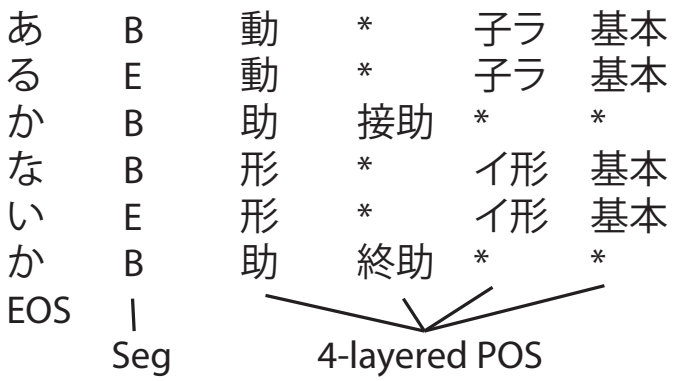


Figure 3.2: An example of full sentence annotation

each character. Encoded character representations are independently transformed into tag representations. For each tag, the encoded representation is projected with a fully-connected layer with SeLU non-linearity (Klambauer, Unterthiner, Mayr, and Hochreiter 2017). Finally, we multiply the tag representation by tag-specific embeddings and apply softmax non-linearity to get normalized tag probabilities.

3.2.1 Encoder Architectures

We use two architectures for the encoder: a stacked bidirectional recurrent architecture with LSTM cells (Hochreiter and Schmidhuber (1997), **bi-LSTM**) and a Transofrmer-inspired mutihead self-attention network (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin (2017), **SAN**). We concatenate both directions of bi-LSTM outputs before passing them to the next layer without residual connections. We also apply layer normalization (J. L. Ba, Kiros, and Hinton 2016) to the concatenated outputs. We do not use dropout in encoders when using silver data for training.

3.2.2 Data Encoding

Our model infers a tag for every input character. While this decision is natural for segmentation, POS tags are not usually tagged in this way.

For segmentation, we adopt {B, I, E} scheme. For POS tagging we broadcast tags to every character which is contained in a token. We use corpora with the JUMAN-based segmentation standard (**Jumandic**), which has 4-layered POS tags: rough POS, fine POS, conjugation type and conjugation form. We treat each tag layer independently in our model, as shown in Figure 3.2.

We also consider a **partial annotation scheme**, where some tags are unknown. An example of partial sentence annotation is shown in Figure 3.3. Unknown tags are displayed by "?" symbols. We create partially annotated silver data by marking as

```
あ      B       動      *       ?       ?
る      ?       動      *       ?       ?
か      ?       ?       ?       ?       ?
な      B       ?       ?       イ形    基本
い      E       ?       ?       イ形    基本
か      B       助      終助    *       *
EOS
```

Figure 3.3: An example of partial sentence annotation

unknown all tags which are ambiguous in a top-k analysis result. When computing the training loss, we treat unknown tags as padding: corresponding values are masked out of loss computation.

Loss Following Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin (2017), we smooth softmax labels. They use the technique described by Szegedy, Vanhoucke, Ioffe, Shlens, and Wojna (2016), which uniformly distributes some small factor ϵ like 0.1 to incorrect labels. However, we do not induce a uniform smoothing. Instead, we want to prevent the model from being overconfident in its decisions without inducing uniformity. We slightly modify the cross-entropy loss as follows.

Remember that softmax probabilities are computed from non-normalized log-probabilities l_i as $q_i = e^{l_i}/Z$, where $Z = \sum_j e^{l_j}$. The cross-entropy loss will be

$$L = -\sum_i p_i \log q_i,$$

where p_i are gold probabilities. In our case the vector p is one-hot, meaning that $p_c = 1$ and other values are zero. This gives a sparse cross-entropy

$$L = -\log q_c = \log Z - l_c,$$

which is often implemented in deep learning frameworks. It has a minimum when $\log Z$ is equal to l_c, but it makes the model overconfident. Instead, we want to stop when $q_c = 1 - \epsilon$, or in other words $e^{l_c}/Z = 1 - \epsilon$. This gives us our modified loss:

$$L = \max(\log Z - l_c + \log(1 - \epsilon), 0).$$

It can be efficiently implemented using the sparse cross-entropy operation. In our experiments we use $\epsilon = 0.2$.

Our final loss is a weighted sum of individual tag softmax losses. We use a weight

	Train		Test	
Corpus	Sents	Tokens	Sents	Tokens
KU	37k	930k	1783	46k
Leads	14k	217k	2195	36k

Table 3.1: Benchmark corpora sizes

coefficient of 10 for segmentation and 2 for the first POS tag layer.

3.3 Experiments

We conduct experiments on Japanese morphological analysis. For training we use two data sources. The first is usual human-annotated *gold* training data. The second is *silver* data from the results of automatic analysis. We use Juman++ V2 – the current state-of-the-art analyzer for the JUMAN segmentation standard as the bootstrap analyzer.

We use two gold corpora. The first is the Kyoto University Text Corpus (Kurohashi and Nagao (2003), referred to as **KU**), containing newspaper data. The second is the Kyoto University Web Document Leads Corpus (Hangyo, D. Kawahara, and Kurohashi (2012), referred to as **Leads**) which consists of web documents. Corpus statistics are shown in Table 3.1. We denote models which use gold training data by **G**.

We take raw data to generate our silver annotated data from a crawled web corpus of 9.8B *unique* sentences. We sample 3B sentences randomly from it and analyze them using the Juman++ baseline model. From it we sample 500M sentences, which become our training silver data, prioritizing sentences which contain at least one not very frequent word. We prepare both top-scored (denoted as **T**) and non-ambigous in beam (denoted as **B**) variants of the silver data. Our silver data is in-domain for Leads and out-of-domain for KU.

3.3.1 Baselines

We use four baselines: **JUMAN**, **MeCab**, **KyTea** and **Juman++** (V2). For MeCab, KyTea and Juman++ we train a model using the same dictionary and merged training sections of KU and Leads, which is evaluated on each corpus independently.

3.3.2 Neural Models

The hyper-parameters of the bi-LSTM-based model are displayed in Table 3.2. We use all unique characters present in our huge web corpus (18,581) as input. We

Parameter	bi-LSTM	SAN
Char embedding size	128	128
Tag embedding size	32	32
# Layers	4	6
Hidden Size	128×2	32
# Heads	-	4
Projection Inner Dim	-	512
# Emedding Parameters	2.38M	2.38M
# Total Parameters	3.88M	3.59M

Table 3.2: Hyperparameters for neural models

select sizes of both neural models restricting the total number of parameters to be less than 4M. For optimization we use the Adam optimizer (Kingma and J. Ba 2014) with hyperparameters and learning rate scheduling described by Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin (2017). We train all models on Nvidia GPUs. On a single GeForce 1080Ti the bi-LSTM model can consume about 4,500 sentences per second and the SAN-based model about 6,500 sentences per second for training. We denote bi-LSTM-based models by **L** and SAN-based models by **S** in experimental results.

3.3.3 Treatment of Gold Data

Existing methods are already highly accurate on this task, and it is difficult to perform hyperparameter and architecture selection reliably with a small development set. Because of that, we split our data in an unusual way. Generally, we use the silver data (B or T) as a train set, the human-annotated original training data (G) as a dev set and the original test set as a test set. Our hyperparameter selection decisions were based entirely on this setting. We do not perform additional hyperparameter search for a combination of silver and gold data for training.

The exception is cases when we use *only gold data* for training. For that, we cheat and optimize our hyperparameters, including dropout, which we use only for this setting, on test scores. Nontheless, the best scores on this setting are significantly lower than the worst baseline.

Experimental Results Results of our experiments are shown in Table 3.3. For each analyzer, we show six values. **Seg** is a tokenwise F1 measure on segmentation. **+P1** requires the 1st layer of POS tags (coarse-grained POS tags) also to match gold data. For the sake of simplicity, we use only POS tags co-located with "B" Seg tags for the evaluation. **+P2** is analogous for the 2nd layer of POS tags. For all results in this table, we train NN-based models for a single epoch, which means *the training*

	KU, News			Leads, Web		
Analyzer	Seg	+P1	+P2	Seg	+P1	+P2
			Baselines			
JUMAN	98.41	97.18	95.45	98.09	96.96	95.71
MeCab	99.10	98.56	97.59	98.25	97.60	96.22
KyTea	99.13	98.25	97.01	97.98	96.85	95.11
Juman++	**99.52**	**99.10**	97.86	98.61	98.07	96.70
			bi-LSTM			
L:G	97.46	96.56	94.78	96.33	95.43	93.46
L:B	99.22	98.82	97.50	98.57	98.01	96.61
L:T	99.33	98.90	97.59	98.68	98.16	96.71
L:BG	99.43	99.05	**98.06**	98.59	98.04	96.76
L:TG	99.43	99.05	98.01	**98.71**	**98.19**	96.80
			Self-attention			
S:G	98.28	97.67	95.66	97.23	96.36	93.91
S:B	99.19	98.75	97.34	98.56	97.99	96.59
S:T	99.23	98.78	97.36	98.66	98.15	96.75
S:BG	99.30	98.90	97.83	98.60	98.03	96.70
S:TG	99.37	98.97	97.93	**98.70**	98.15	**96.83**
			Pre-training scenario			
S:B→G(a)	99.24	98.85	97.75	98.58	98.03	96.64
S:B→G(b)	99.15	98.65	97.55	98.39	97.78	96.36
S:B→G(c)	99.27	98.82	97.75	98.50	97.91	96.43
S:B→G(d)	99.26	98.82	97.76	98.52	97.94	96.48

Table 3.3: Test F1 score comparison on benchmark corpora. Legend: bi-[L]STM, [S]AN, [G]old data, [T]op-only and [B]eam-non-ambigous silver data.

procedure sees each silver sentence only once. We use one gold example for ten silver examples for mixed-data settings, looping over the gold data until the silver data is extinguished.

Training neural models only on gold data quickly results in overfitting which can be seen in L:G and S:G results. These scores are significantly lower than that of our worst baseline: JUMAN.

Models trained on only non-ambiguous silver data (*:B) are comparable to the best baseline on Leads (in-domain), although they cannot reach the accuracy of Juman++ on KU. Using top-only silver data (*:T) further improves accuracy. Both of our models in this setting slightly outperform previous Leads SOTA and have more or less the same accuracy. On KU, the LSTM-based model seems to be slightly better than the SAN-based one. In the context of semi-supervised learning, tri-training emphasizes using data when there exists a disagreement between the analyzers. Instead, we throw away difficult cases for beam-based data, denoising it in a sense, but NN seem to handle that kind of noise relatively well.

	Size, MB		
Analyzer	Dictionary	Model	Total
JUMAN	288	1	289
MeCab	312	8	320
KyTea:G	-	569	569
KyTea:TG	-	3218	3218
Juman++	157	288	434
bi-LSTM	1	14	15
SAN	1	13	14

Table 3.4: MA model sizes for Jumandic

Adding the gold data to the silver data (*:BG, *:TG) allows both models to improve their accuracy further. Results on Leads are comparable for both L:TG and S:TG and higher than the previous SOTA, giving segmentation error reduction of 8% in comparison to Juman++. On KU, the LSTM-based models seem to perform better without a significant difference on the TG and BG settings, while still underperforming the Juman++ baseline except +P2 case, where both models are stronger than Juman++.

3.3.4 Pre-training Scenario

We also check the fine-tuning approach when we first learn the representations on a large corpus and then refine the model on a gold corpus. S:B→G(a-d) are four such runs of a SAN-based model with different hyperparameters. All four runs are initialized with the same S:B model and trained on the gold data only. We found it difficult to find good hyperparameters for fine-tuning. The models were prone to overfit very fast. Mixing gold and silver data resulted in stable training without hyperparameter search.

3.3.5 Model Sizes

We compare the model sizes of analyzers in Table 3.4. In case of dictionary-based analyzers the dictionary takes most of the space. We count sizes of compiled models for all analyzers. KyTea, as another example of pointwise MA, uses string-based features and treats its features uniformly, hence dictionary size is not applicable to it. A **KyTea:TG** variant that uses additional 2M silver sentences takes almost 6x the space of the original model, reaching 3GB. When using neural networks, on the other hand, it is possible to control model sizes more easily. Moreover, our proposed models take significantly less space while having comparable accuracy.

Analyzer	KU	Leads
KyTea-D:G	98.45	97.04
KyTea-D:T	98.51	98.10
KyTea-D:TG	99.18	98.31
KyTea:G	99.13	97.98
KyTea:TG	99.33	98.42

Table 3.5: KyTea test Seg F1 comparison. -D models do not use the dictionary. T models use silver data (2M sentences, created like in the main experiment)

Dictionary-based analyzers store other information, like readings and lemma forms, in addition to token surface forms and POS, but removing that information would not make model sizes comparable with NN-based ones. For NN-based analyzers, we count a dictionary as 1 MB because they need a character-to-id mapping to work. However, the list of characters contains non-frequently used characters, some of which could be treated as UNKs without any accuracy loss. We also treat weights as 4-byte floating points, and so it would be possible to further decrease the NN model size, for example by using less precise storage formats.

3.4 Discussion

3.4.1 Dictionaries

Dictionary information is usually added to character-based models either using a binary feature vector (e.g. a dictionary contains a trigram to the left of the decision point) or word embeddings. We believe that a dictionary can be replaced with a large training corpus which includes most of the entries from that dictionary. A neural model with only the unigram character input can solve word segmentation and POS tagging only if it builds some knowledge about the dictionary internally. Our main experimental results (Table 3.3.3) show that it seems to be the case and there is no need to model the dictionary explicitly.

Table 3.5 shows an effect of using dictionaries and silver data on KyTea, an instance of symbolic feature-based analyzer. Models tagged with **T** use additional 2M silver training data analyzed by Juman++. KyTea has better accuracy in settings when it uses the dictionary. The dictionary even helps in the setting with additional silver data. Unfortunately, the model size increases as well, limiting the amount of silver data we can use, and the accuracy cannot rival neural approaches.

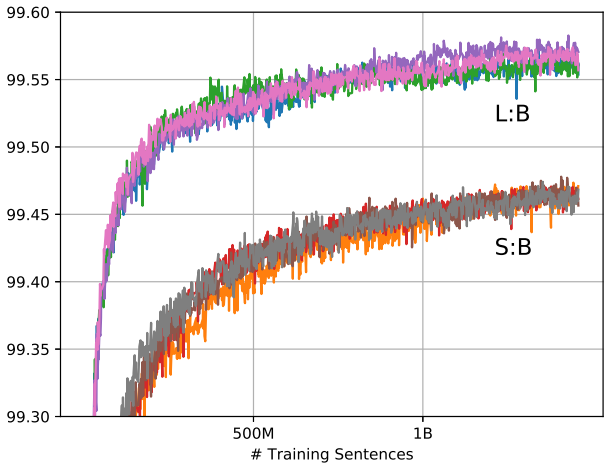


Figure 3.4: Dev Seg F1 curves for L:B and S:B

3.4.2 How much data do we need?

For our main experiments, we train all models for a single epoch on our silver dataset. Figure 3.4 shows KU train (our dev set) Seg F1 curves for L:B and S:B for three epochs. We ran each experiment four times with different random seeds. The learning curves become less sloppy when reaching 500M sentences but do not become flat there. The training does not seem to completely converge even after 3 epochs. We still use one full epoch (500M) for our main experiments. The curves are pretty noisy, but it seems that the model is robust with respect to initialization.

3.4.3 SAN Ablation Experiments

The proposed MA achieves high accuracy while having very compact models. The inputs do not contain any information on how to combine characters into words and we assume that the model learns it from the data. To get the model size even smaller, we check which model parts contribute more to the resulting analysis accuracy, meaning that they contain the dictionary knowledge.

We perform ablation experiments on the SAN model by varying its hyperparameters and checking how it affects the accuracy of the resulting analyzer. The LSTM model could not converge in this setting. We used 2.5M of silver training data for these experiments.

Figure 3.5 shows the segmentation F1 score when varying input embedding, shared representation and SAN hidden dimension sizes. JUMAN score, as a lowest acceptable baseline, is shown in red. The embedding size seems to have a lower impact on accuracy than the shared representation and the SAN hidden dimension size. Namely, the (128-16) model with the embedding size of 16 has higher accuracy

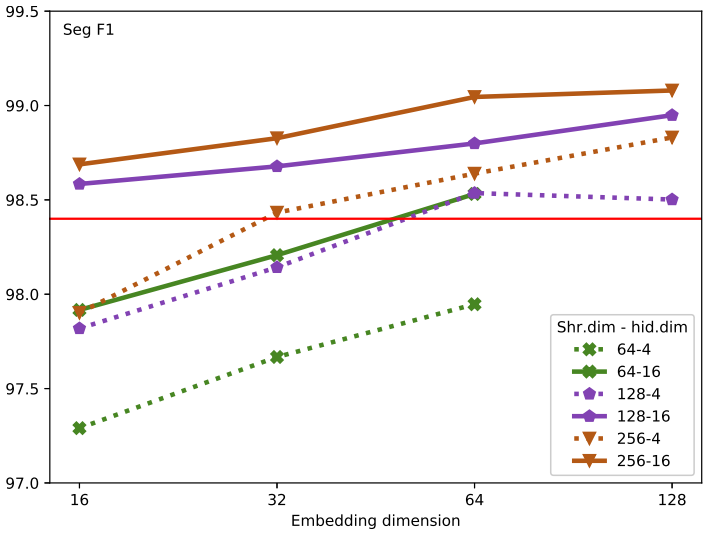


Figure 3.5: Effect of embedding size on Seg F1

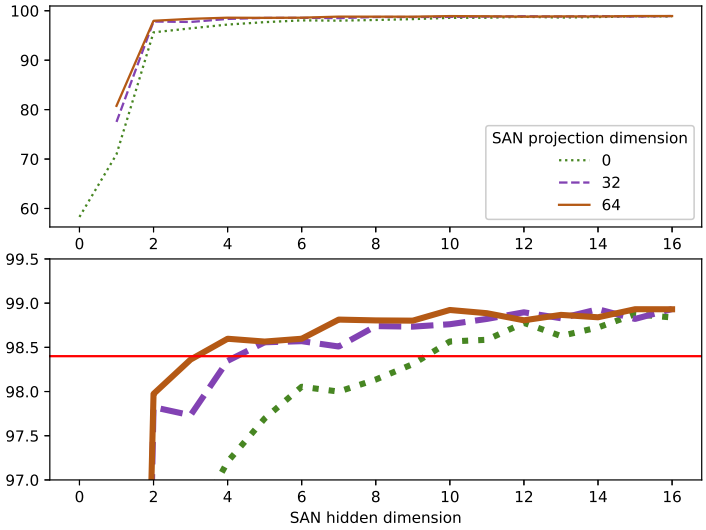


Figure 3.6: Effect of SAN hidden dimension on Seg F1. Bottom is a scaled-up part of the whole graph.

than the (128-4) model with the embedding size of 128. Accordingly, we believe that the encoder contributes much stronger to learning the dictionary than character embeddings.

One more interesting observation is that the models are still better than JUMAN, while having much less parameters than our base model. We explore more extreme settings of the SAN hidden state, shown in Figure 3.6. We fix embedding and shared representation dimensions to 128 and vary the SAN hidden and projection dimensions. The lower subgraph is a scale-up version of top graph. The point at SAN hidden size equal to 0 means that we directly use unigram embeddings to predict segmentation without any encoder.

The SAN projection size is consistent with accuracy, especially on smaller SAN hidden sizes. An interesting observation here is that the SAN model seems to work

System Segmentation	Correct Segmentation	Transliteration	Meaning
こう|ゆう|曲って	こうゆう|曲|って	ko:yu: kyoku tte	this song is
なんて|すん|ごい	なんて|すんごい	nante sungoi	how awesome
んな|わけな|いって	んな|わけ|ない|って	nna wake nai tte	no way!
あっ|ちゃんと|遊び|たい	あっ|ちゃん|と|遊び|たい	*see main text*	

Table 3.6: n-grams with inconsistent POS tags which are also Juman++ errors

even with hidden dimension of 2. When the hidden dimension size reaches 4, the extremely small model accuracy is higher than the JUMAN baseline. This shows that it is possible to create an extremely small MA with acceptable accuracy.

Label Uncertainty and Error Analysis Because our neural models infer all tags independently, they can be inconsistent, for example, a word can have different POS tags on different characters. We looked into frequent 3-grams where the central word has inconsistent tags (POS tags are not the same for all characters, or they do not form a correct 4-layered tag). Most of these trigrams occur in ambiguous situations.

We have picked several examples which are actually errors in Juman++ segmentation as well. They are shown in Table 3.6. In Japanese, words often have several orthographic forms. The most common variant is usage of hiragana (phonetic script) instead of kanji (ideographic characters). Verbs can have different possible endings, e.g. 曲がる and 曲る (magaru – to turn or bend) are two orthographic variants of a single verb. There are also colloquial variants; namely the verb 言う is usually read as いう (iu – to say), but can also be written as ゆう because the pronunciation is close. These phenomena are relatively common in web and user-generated texts, but corpus and segmentation dictionary coverage of them is not very good.

The first two examples contain alternative colloquial spellings of words こういう (ko:iu – such) and すごい (sugoi – awesome). In the first example the system incorrectly recognizes 曲|って (kyoku tte) as 曲って (magatte) – a conjugation of 曲る. The fourth example (a chanto asobitai/ac-chan to asobitai - ah! [I] want to play properly/[I] want to play with ac-chan <person name>) is actually ambiguous and can have two meanings. The second one is more probable though. The fact that frequent words with uncertain POS tags are Juman++ errors as well implies that insufficient gold data causes the uncertainty.

We also compare differences between Juman++ and our models to get an insight on general problems with proposed methods. Neural models make many errors in hiragana words. For example, both neural models make errors in the sentence 弱者|が|とうた|さ|れて (jyakusya ga to:ta sarete - weaklings lose to natural selection). LSTM makes a segmentation mistake (と|うたさ) and SAN does a POS

tagging mistake, while Juman++ produces the correct answer. It knows that とうた is a special type of noun that is often followed by されて from POS tags. Hiragana-based spellings of most content words are somewhat rare in Japanese, and NN models do not have enough training data for these spellings. It could be possible to improve the situation by using data augmentation techniques. Another frequent problem is segmentation and tagging of proper nouns. We believe that this problem could be solved by data augmentation, but we leave this as future work.

3.5 Conclusion and Future Work

We presented a novel way to train small neural models for Japanese Morphological analysis by directly feeding the network a large number of silver training data. Our method achieves new SOTA on web domain when combining the silver data with gold one. This is an empirical evidence that there is no need for feature engineering for neural morphological analysis at all. A neural network can learn implicit dictionary information itself and it does not need to be large. We also show that training by mixing the data together works better than fine-tuning and is more stable.

Our work can be extended in the future in different ways. We will consider how to make the model to recognize new words, which is an important feature for a practical analyzer. Using tri-training also seems to be a natural extension for this work. It is easy to provide diverse models, required for tri-training, by using different types of encoder and varying network parameters. Furthermore, our tagging approach should be universal and work with other tasks like named entity recognition. A method to incorporate tags with a large number of possible values (like readings and lemmas) without introducing embeddings for them, hence keeping the models small, could also be a useful extension.

4 High-Quality Example Extraction

4.1 Outline

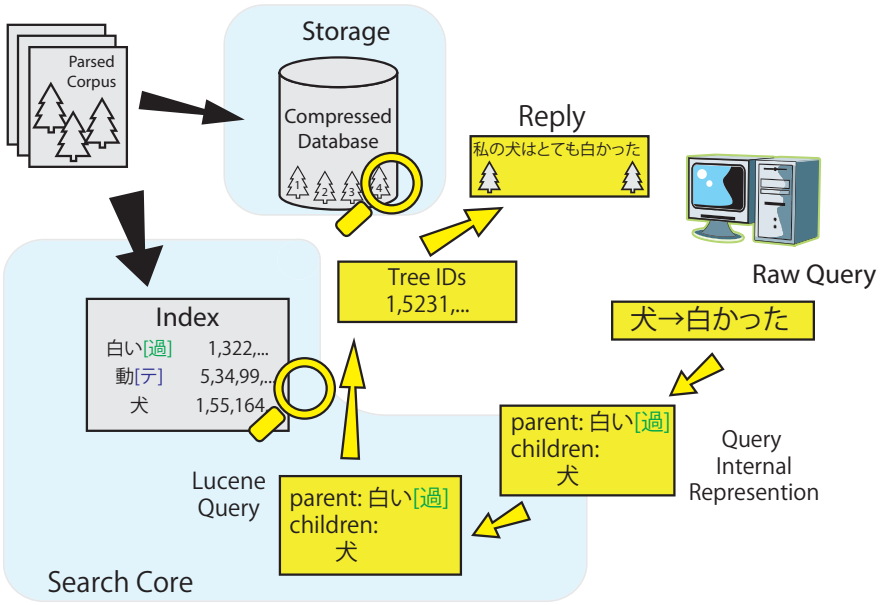


Figure 4.1: Search system outline. We build search index and storage from parsed corpus. Searching converts the query into internal representation and uses that to traverse the index. Search returns parsed trees.

This chapter explains the example extraction system for Japanese language learning. Recalling the criteria defined in section 1.3.3, we call sentences *high-quality* if

1. each sentence has a high intrinsic value,
2. sentences are diverse.

We propose an example extraction system architecture consisting of two main components.

The first component is a dependency and grammar-aware search engine. It indexes a large corpus of dependency-parsed sentences and can perform a search with partially POS-erased subtrees as queries. We describe the search engine in chapter 4.2. The example search system workflow is shown in Figure 4.1. We use a target

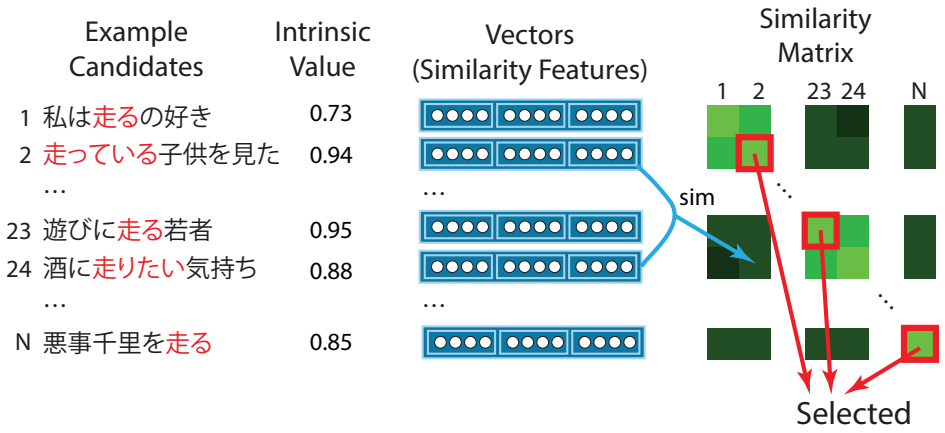


Figure 4.2: Example sentence extraction outline. The objective is to select "best" and non-similar example sentences from the input list. The target word is marked red.

word with part of speech tags of usual dependencies for the target word as a query. Fetching sentences with rich syntactic structure near the target word allows us to select better sentences than a simple word-based system would allow us to. The list is created for a *target* word using the search system described in the 4.2 in a way to filter out really bad sentences and select useful *patterns* of target word usage based on POS information of the target. Details of pre-selection are described in the Section 4.2.4. The search part produces about 10,000 example sentences candidates which are passed into the second stage.

The output of the pre-selection stage is further processed to extract a small number of sentences that are going to be used for the actual learning process. Those example sentences should satisfy requirements described in subsection 1.3.3. The selection of example sentences has a complex objective function: sentences should be globally non-similar, covering distinct and rare senses. However, at the same time sentences should have high intrinsic value (how good an individual sentence is without considering other sentences), for example, a sentence difficulty should be acceptable for a learner. The outline of example extraction is shown in Figure 4.2.

For the extraction of examples from the candidate sentence list we adopt an approach based on Determinantal Point Process (DPP) (Kulesza and Taskar 2012). DPP is a probabilistic method for modeling diverse datasets. The proposed feature representation of the DPP method consists of two parts: diversity and quality. This decomposition seems very suitable for the example extraction problem. DPP as a mathematical framework is discussed in Section 4.3.

The feature representation for a sentence is computed from similarity and quality parts. The similarity part defines how two sentences are close to each other. The quality part reflects the intrinsic value of a sentence. Detailed discussion on the

feature representation is provided in Section 4.4.

4.2 Dependency and Grammar Aware Search Engine

4.2.1 Introduction

General search systems like Google or Microsoft Bing are designed for searching documents relevant to a specific query. Such documents are usually long pieces of text, for example web pages. Language learners and teachers use such systems for acquiring example usages or contexts for words they learn. However, general search systems are not well-suited for this task. Firstly, users doing such searches are not looking for documents, they are looking for sentences. Additionally, conventional search engines ignore grammatical information when indexing text, although this data is extremely useful for finding example usages of words.

Searching sentences for educational and linguistic usage often requires more features than just querying on terms as general systems do. Sentence level search should support queries not only on the lexical level, but on lexical dependencies, part of speech (POS) tags, conjugation forms, and **grammatic words** like case markers (が,を) or auxiliary verbs (いる after テ form) as well. Usually, users want to find usages of a word in some context.

Understanding onomatopoeia is difficult for many Japanese language learners. Example sentences like "肌がピリピリする" are mostly for learners because they give no idea about what ピリピリ is. Having sentences like "肌がピリピリ痛く感じる" are substantially better in this case. It can be said that a system should give users the ability to choose *"a form"* of context for the word. Additionally, a system should be able to work with a huge amount of data, because context patterns are usually sparse. At the same time, to be useful the system should give replies quickly.

We have implemented a distributed high-performance sentence level search system for the Japanese language that can process queries with not only lexical information but grammatic words and lexical dependency information as well. The system achieves less than 300 ms query times for 90% of queries when 700M sentences are indexed.

The proposed system architecture consists of two main parts: compressed database and search core.

Search core is based on Apache Lucene[1] with additional components for dependency and POS tag support for indexing and querying. Furthermore, to support fast queries on huge corpora, the system is implemented in a distributed master-

[1] https://lucene.apache.org/core/

slave-like manner. Distribution was done using the actor programming model and Akka library[2].

Related work

Sentence search tools are related the most to corpus management and exploration tools. However, there are not many tools that use structural information. Jakubíček, Kilgarriff, McCarthy, and Rychlý (2010) implements a syntactic corpus search system. This system focused on searching using constituency instead of dependency syntactic structures. Also, the system was not a search engine and query times on sentences structure were in orders of tens of minutes which renders working with huge corpora impossible.

For the Japanese language, dependencies are used in search as well (Shinzato, Shibata, D. Kawahara, and Kurohashi 2011; Takeuchi and Junichi Tsujii 2005). TSUBAKI search system (Shinzato, Shibata, D. Kawahara, and Kurohashi 2011) uses dependency trees for indexing and querying and it is distributed. However, it is a document-level search system and does not allow doing queries using POS information. A system proposed by Takeuchi and Tsujii (Takeuchi and Junichi Tsujii 2005) uses dependency information, however does not allow to use of grammatical and dependency information. It has more focus on handling paraphrases. Recent versions of Chaki (茶器) corpus management tool[3] support queries using dependencies, lexical, and part of speech information, however, it is not a search engine – it was not designed for a scale of several hundred million sentences.

Query Language & Examples

The Japanese language has no natural word boundaries symbols and the definition of a morpheme varies from one analyzer to another. In this sense, writing queries using sub-bunsetsu units is going to be difficult and error-prone, therefore another approach was taken. Search system query language is designed by adding special symbols to plain Japanese. The list of special symbols is [-,→*~]. Minus (-) has its usual meaning as in a typical search system: absence of a term in search results.

A query is divided into parts by commas. Each part is completely independent of others. Usually, the whole query is a disjunction of its parts. General search systems like Google use whitespace instead of commas.

Dependency between two terms is specified by an arrow symbol: "A→B", meaning B as a parent of A. Arrows can be chained "A→B→C", however in this case, both

[2] http://akka.io/
[3] https://osdn.jp/projects/chaki/

A and B would be treated as sibling children of C. Sentence head can be specified as "A→EOS", where EOS is literal.

A star after a word means that you want to ignore its conjugation information. For example, "食べる*" would match any conjugation form of 食べる and even 食べない, however query "食べる" would match *only current (dictionary) form* of the term. Only the last conjugation is going to be ignored: "食べたい*" will match 食べたくない, but won't match 食べない. Ignoring conjugations inside words is not supported yet. For non-conjugatable POS this symbol is a no-op.

A tilde (~) before a content word means to replace it with its own POS tag. This means "~食べて" is going to match a テ *form of any verb*. For grammatic words, this replacement is not supported presently.

Here is some example queries the system supports. A query "~物が→ぴりぴり→~動く*,-する*" finds usages of onomatopoeia ぴりぴり in context, but usages with する are difficult to understand and are therefore ignored. Search results include sentences like "寒さより肌がピリピリと痛く感じます。", "頬がピリピリ凍る、京都の冬です。", and "全身の神経がピリピリ緊張する。" which are good examples for the ピリピリ.

In the query "~書いた→ため,-ため→EOS" the objective is to find out usages of ため after past forms of verbs so that it is not a head of the sentence. Search results are sentences like 陰謀を暴いたために脅迫された.

For a query "いい加減→~やらない", the objective was to find unusual imperative-like usages of verbs with いい加減 as a modifier. Results contained sentences like "その妄想いい加減やめない？" or "良い子はいい加減止めない？" which are exactly the usages we seek.

These types of queries are impossible to search with a regular search engine.

4.2.2 System overview

The proposed system consists of two main architectural parts, as shown in Figure 4.3: a compressed tree database and a search core. The tree database stores dependency parse trees in compressed form. They are used to build replies to search queries. The search core is built on Apache Lucene with a custom tokenizer and querying. Instead of content word morphemes, tokens are created from parse trees in a way to make it possible to issue queries on conjugation forms of POS with grammatic words attached.

Current implementation uses KNP[4] dependency parser trees as input. KNP groups morphemes to bunsetsu units which are useful units for human interpretation. The search core uses KNP bunsetsu as a core unit for further processing.

[4] http://nlp.ist.i.kyoto-u.ac.jp/EN/?KNP

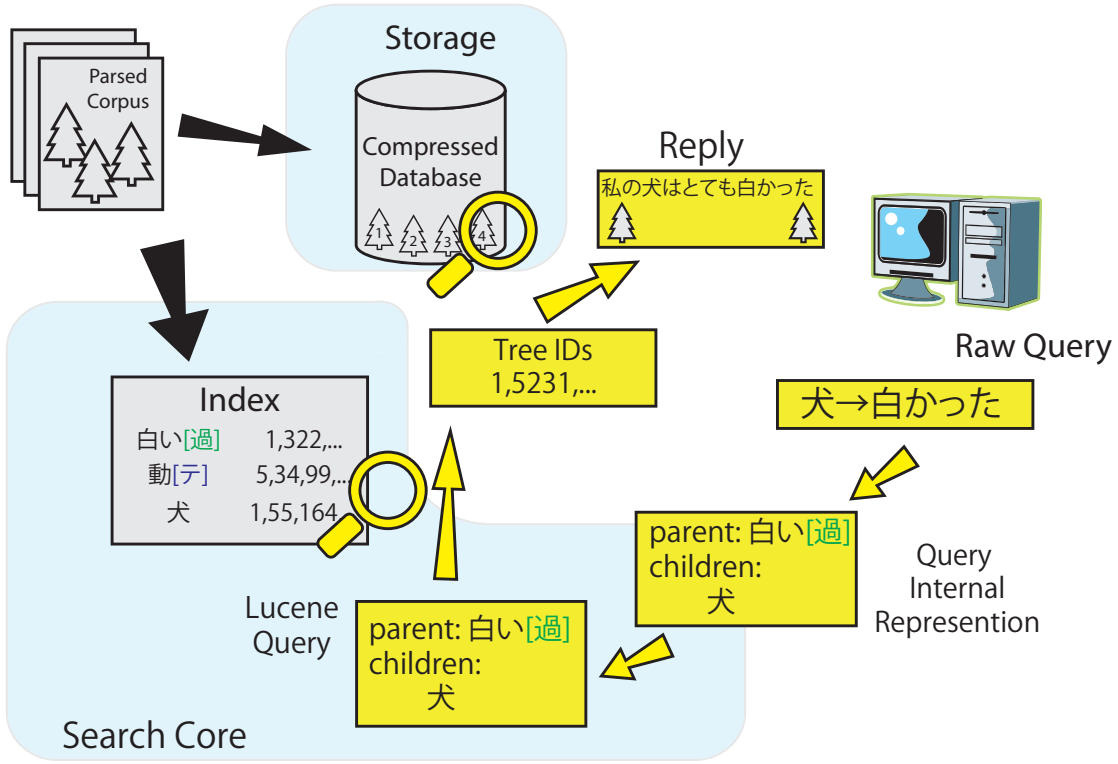


Figure 4.3: System structure and querying workflow

Compressed Database

The system operates on dependency trees as input. Those trees should be extractable to create replies, and because the output of dependency parsers is rather verbose, the data should be stored on a disk in a compressed form. However, usual compression tools do not support random access to compressed files. Building a specialized compressed database overcomes this limitation.

The compressed database consists of an index stored as a B-tree using MapDB[5] and data files. The database index is a mapping from a tree id to a tree compressed pointer and tree size pair. Data files consist of 64-kilobyte blocks. Each block is archived independently of others and saved to disk. There are no inter-block dependencies, meaning that blocks can be read in arbitrary order. To extract a tree from a data file, the system needs to read a block from a disk, decompress it, and get a tree from that block. The information about the block and the position inside the block can be stored in a single compressed pointer.

Compressed pointer is a trick taken from the bioinformatics BAM/BGZF[6] storage format used for storing DNA sequences in compressed files. The compressed pointer consists of two parts: the beginning of a compressed block in a data file and an offset

[5] http://www.mapdb.org/
[6] http://samtools.github.io/hts-specs/SAMv1.pdf

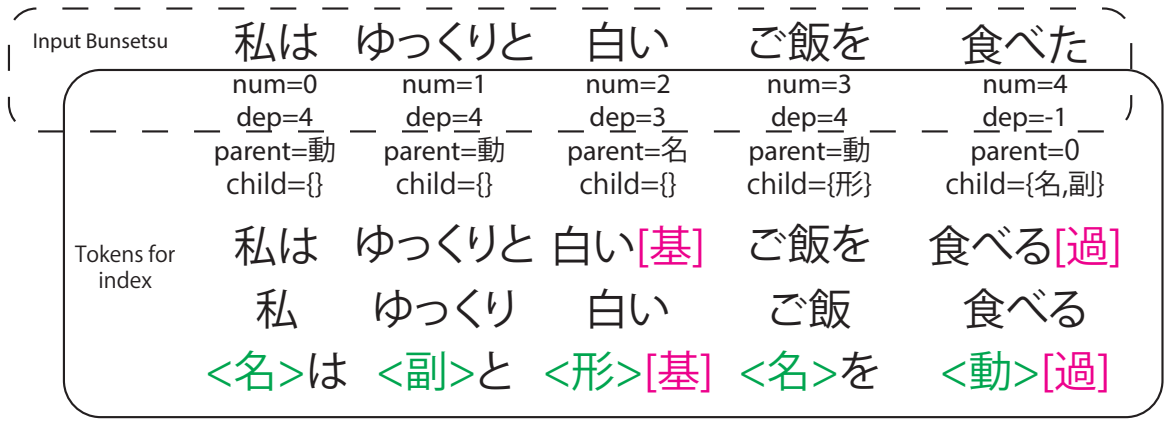


Figure 4.4: Bunsetsu to token conversion for indexing sentence "私はゆっくりと白いご飯を食べた". Tokens contain lexical information (black), POS tags (green) and conjugation forms (magenta). Dependency information is common for a set of tokens spawned from a single bunsetsu. This information consists of bunsetsu number, dependency number, POS of parent and set of children POS.

of needed data inside the decompressed block. Block sizes are fixed to 64k and the 64-bit pointer can be formed by making the lower 16 bits to store the uncompressed offset and the remaining 48 bits to store the block start address in the compressed file.

Search: Indexing

Speed of search engines comes from a special structure – **inverse index** – that is created from original documents. The index is created from tokens which are produced by analyzing input. For the proposed system tokens are created from bunsetsu of input trees. Each bunsetsu is numbered and has a **dependency number** (number of a parent). Both of these numbers with POS information of bunsetsu children and parent are transferred to tokens. Tokens are created for each occurrence of the bunsetsu in the tree. When stored to the index they are inverted and become **token postings** – information about documents that contain a certain token.

Search systems work by fully matching query tokens with indexed ones and processing posting information linked to the tokens. The main objective behind the token design was to combine lexical and grammatic information in a single place, meaning that it could be both stored in the index and at the same time easily constructed from a search query. Adding dependency information to these tokens enables the implementation of a system to meet all the requirements stated in the introduction.

Tokens are generated from a bunsetsu in two steps. The first step generates a seed token from the bunsetsu. Token content is a concatenation of bunsetsu

morphemes. Morphemes with conjugatable POS are represented by a lemma form with the conjugation tag. For example, the verb "帰った" would be represented as "帰る[過]". Non-conjugatable POS morphemes are represented by themselves.

The next step generates rewritten tokens from the seed tokens until no more new tokens can be created using the rule-based rewriting process. Rewriting is done by replacing content word lexical information with part of speech information or removing some parts of tokens. For example, case markers of nouns are removed for some rules.

This representation allows to easily match the same forms of different words while getting the benefits of the reverse index in terms of performance. A list of created tokens for raw sentence "私はゆっくりと白いご飯を食べた" is shown on Fig. 4.4.

Querying for a single POS tag is not very useful – it is going to match any document for most POS tags. In addition to that storing such tokens in the index will consume much of the index space. However, searching in a case when POS is a child or parent of something is useful. Storing the information about parent and children POS for each bunsetsu allows answering POS dependency queries efficiently. By limiting such information only to nouns, verbs, adverbs, and adjectives it is possible to store the POS dependencies **packed** – using only one additional byte per posting.

Search: Querying

Input search queries undergo two transformations. The first transformation is to analyze the input query with a morphological analyzer and build an internal query representation. This representation is transferred to actual working nodes, where internal query representation is finally transformed to low-level Lucene queries.

Analysis of input queries is performed in two steps. The first step removes special symbols from the query replacing some of them with commas. This is done so that special symbols do not interfere with the morphological analyzer. This step is followed by morphological analysis using JUMAN. Results of the morphological analysis are merged with the information from special symbols forming **internal query representation**. Internal representation consists of one or more **parts**, separated in a raw query by commas. Each part consists of a **parent** with zero or more **children**.

There are five Lucene queries used. Two are built-in core queries and three are ad-hoc for using the postings information. Core Lucene queries are **term** and **boolean** queries. The boolean query is used to bind multiple query parts together. The term queries are used for parts without any children because there is no need to use dependencies.

Ad-hoc Lucene queries use information in postings to select documents. The first

one – **packed term query** – is also an extension of the core term query. It is used when a part has a POS-only parent or child. Its specific work is to compare both child and parent-packed POS dependency from postings with a reference created from the query part.

For query parts when at least one of the children is not just a POS tag, the second one – **dependency query** – is used. The main idea behind it is to find trees that contain the conjunction of parent and all the children from the query part. Also, for each found tree, the child dependency numbers should be equal to the parent bunsetsu number. POS-only children are handled in the same way as in the packed term query.

The last one is used for supporting queries of type "A→EOS" which checks dependency numbers of postings.

By default, Lucene uses tf.idf similarity for result ranking. Idf measure does not make any sense in the case of pos-like tokens, so only tf is used in the system. Also, the length is normalized in a way so that moderately long (4-5 bunsetsu) sentences gain the highest score.

Sentence Scoring

Search systems usually output results as a list sorted by a **score** – a number that specifies how relevant the document is for a query. For a query that has multiple subqueries, the total score is usually going to be a sum of scores for the individual subquery.

Elementary queries (not containing subqueries) calculate the score directly from the documents. For example, a frequently used tf.idf model calculates a document score for a term query as a product of term frequency (number of times a query term was found in the document) and inverse document frequency (total number of documents divided by the number of documents which contain the query term). Inverse document frequency is used to give more weight to less frequent terms, which are thought to be more important for the query.

Inverse document frequency is mostly irrelevant for the case if documents are sentences because they are short and most of the time each term inclusion is equally important. Moreover, the concept of importance because of the frequency does not make any sense for tokens that contain POS tags with grammatic words. It is very strange to say that usage of で case with nouns is more important than the を case because it has a higher idf. Because of this, the search system does not use idf for result ranking.

Additionally, Lucene by default normalizes documents by dividing document score by square root of document length. For the case of sentence search, it gives a

much higher score to very short sentences which are usually fragments and are not useful as example sentences.

Instead of the default normalization function, the search system gives the most weight to a "sweet spot": sentences of moderate length — 4-5 bunsetsu. The weighting function decreases slowly to the right still giving high weight to longer ones, but drops to almost zero for shorter sentences, effectively prohibiting them from search results.

Distribution

Handling a large amount of data is impossible without building the system in a distributed manner. Searching the index is an operation limited mostly by IO bandwidth and memory bandwidth, and getting the stored data from the database is an operation limited by IO latency. Distributing the system increases available resources for every listed operation and makes it possible to work with huge datasets interactively.

Fortunately, search engines are relatively easy to make distributed. ElasticSearch and Apache Solr are two distributed search engines built on Apache Lucene. Both of them use a simple master-slave model for distribution. Unfortunately, they are not designed for trees: neither for storing them nor for doing queries with trees.

Distribution using master-slave search engine architecture was implemented using Akka middleware. Master node divides documents for indexing for each slave node to make the number of indexed sentences approximately equal on each node. For the search query, the master fans out the input to all nodes. Then it merges results from slaves, keeping only ones with top scores.

One of the problems related to distribution is that some nodes were greatly slower than others. Setting timeouts for reply help with this problem, however, it is difficult to tune the actual value of that timeout. The problem was solved to collect reply time statistics from slave nodes and tune the full reply timeout so that at least 70% of slaves were able to reply. A random query from the list is automatically sent to the system every 5 seconds so that reply times statistics do not become outdated.

4.2.3 System state

The system is currently accessible online[7]. It is deployed on 55 slave nodes and 1 master node of our cluster system. The corpus used for search is a part of a web corpus, of 700M sentences (about 1TB of compressed results of syntactic parse).

[7] http://lotus.kuee.kyoto-u.ac.jp/depfinder/search

Each node has approximately 20GB of compressed trees stored in the compressed database. The index size is 3GB per node on average.

We have measured search response times in this setting. In the experiment, we used a list of 1600 queries that contain frequent verbs, nouns, and adjectives with dependency queries like "~私を→分かる" with different frequent POS and words in parent and child place. Queries from that list were run once per 500ms and the response time was measured. For each search, trees for the top 100 results were extracted from the database and sent as a response. 152179 response times were collected. 99% were less than 424 ms, 90% were less than 285 ms; median and average response times were 26 ms and 179 ms respectively. We believe that the current search speed is reasonably good for general usage. We should note, however, that the system is presently deployed on a shared cluster that runs programs by other users at the same time, implying that dedicated installation is going to have a more stable performance.

The system uses the results of an automatic morphological analyzer and parser, so there are errors in the analysis. Sentences with different words do appear in search results because of it. With the improvement of the analyzers, this problem is going to disappear.

The source code of the system is going to be distributed under an open-source license. It is also possible to match arbitrary tree structure without modifying indexing and tokenization, only on the query level, however, this has not been implemented yet.

4.2.4 Selecting Example Sentence Candidates

Depending on the part of speech, example sentence candidates are searched in a way so the target word usage would be more or less useful for a learner.

For example, if the target is a verb sentences are selected so they would include some arguments for a verb and possibly a syntactic parent. Arguments are useful both for understanding by learners and computing features for the extraction part. If a verb will have arguments, then the predicate-argument analysis result data is going to be populated as well giving the ability to compute semantic features.

For selecting candidates we use queries that match a target word with up to 3 children or parents. The exact types of parents of children depend on the POS of the target word. The number 3 was chosen to have balance with different arguments and to keep the syntactic vicinity of the target word diverse between the example sentence candidates.

One specific feature of the search system is the ability to limit the number of matching subqueries in a compound query. Usually, a disjunction (OR) query makes

its score as a sum of scores of all its subqueries. If the number of processed sentences is large, there will be a large number of sentences that fulfill most of the subqueries for frequent words. By matching all the subqueries an OR query effectively becomes an AND query, however that is not very useful for example sentences. It decreases the total diversity of search results making every result to be more or less the same. To deal with it we implement a special type of compound query that uses only top N scores of its subqueries.

For a word and its part of speech, the system generates a search query that represents frequent patterns of word usage. For this section the following notation is used:

- *word* denotes the *target word*
- ⟨noun⟩, ⟨verb⟩, ⟨adj⟩, ⟨adv⟩ denote any word that have the indicated part of speech
- → denotes a dependency between two terms with right being a parent and left being a child.
- Items in a list form a disjunction, or an "OR" query.
- EOS token means end of sentence. Dependency to it like word → EOS means that the word should be the last word in a sentence.

Patterns for individual parts of speech are described below. Each pattern contains an additional search term that decreases the score of a sentence if it contains multiple inclusions of a target word.

Verbs

- ⟨noun⟩-*ga* → word
- ⟨noun⟩-*wo* → word
- ⟨noun⟩-*ni* → word
- ⟨adv⟩ → word
- word → ⟨noun⟩
- word → EOS

Patterns for verbs capture that the verb should have some arguments, be the last one or in the middle of the sentence. The limit on matching subqueries is 3. There is a

slight bias on the subquery that has ⟨noun⟩-*wo* argument, because object arguments frequently mutate the senses of verbs and are especially important for understanding the example sentences.

Nouns

- word-*ga*
- word-*wo*
- word-*ni*
- word-*de*
- word-*ha* *
- one of:
  - ⟨noun⟩-*no* → word *
  - ⟨noun⟩ → word
  - word-*no* → ⟨noun⟩ *
  - word → ⟨noun⟩
- word → (⟨verb⟩ OR ⟨adj⟩)
- (⟨verb⟩ OR ⟨adj⟩) → word

The limit on subqueries is 3. The *ha* case marker gives less information about the relation of the word than other cases, so it is given a slight penalty. Dependencies on the nouns with *no* case marker are given a slightly higher weight because *no* could be important for the meaning of nouns.

Adjectives

- ⟨noun⟩-ga → word
- ⟨noun⟩-ha → word
- word → ⟨verb⟩
- word → ⟨noun⟩
- word → EOS

There is no limit on subqueries. Dependency of a target adjective is given a slightly higher weight if it is a child of a noun, because modifying nouns is a main role of an adjective. Dependency on EOS is given a slightly lower weight.

Adverbs

- (⟨noun⟩-ga AND word) → ⟨verb⟩
- (⟨noun⟩-ga AND word) → ⟨adj⟩
- (⟨noun⟩-ga AND word) → ⟨noun⟩

Adverbs usually modify verbs and interesting usages of them are linked both with the subject of the sentence and the predicate. Patterns try to capture this relationship.

4.2.5 Conclusion

We have developed a distributed large-scale sentence search engine that should be useful for linguists, researchers, teachers, and students studying Japanese. It supports queries not only on lexical information but also on POS, grammatic, and dependency information as well. An installation that uses 700M sentences from the web is available on the Internet. 90% of simple dependency queries get a response in 300 milliseconds.

A design of tokens that contained lexical, POS, and grammatic information at a single place allows using general search technology. Dependency trees were stored in a specialized compressed database.

4.3 Determinantal Point Process Framework

The objective function of the example extraction task is rather complex. We need to find a balance between the diversity of the whole set and the inclusion of good example sentences in the final result. Authors of the DPP have shown that the method performs well on a series of similar tasks. Reported tasks were text summarization, modeling non-overlapping human poses in images and video and building timelines of important news stores. Each task requires extracting non-similar high-quality items from an initial set. Indeed, task formulation for each one is very close to the expected output of the example extraction system.

A second reason for using DPP as a selection algorithm is its high execution speed. If similarity features are vectors of length D and similarity between two items is a dot product of these two items, then a greedy selection algorithm works with a complexity of $O(ND^2k + D^3)$ to select k items for situations when a feature dimension D is smaller than the number of items N and k is smaller than D, which is usually the case. This is linear in the number of items N and scalable. The greedy algorithm is approximate and does not fully solve the problem of finding a subset

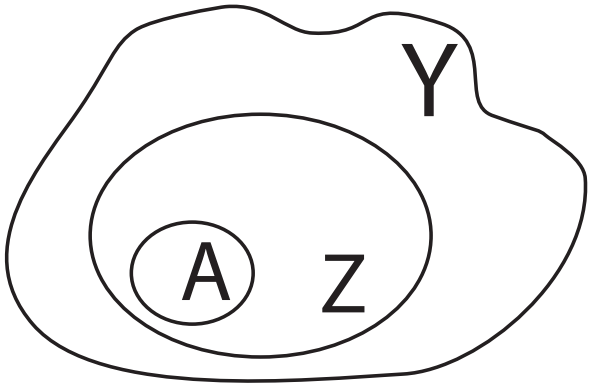


Figure 4.5: Sets participating in the Equation (4.1)

with the highest possible probability (maximum assignment problem) of selecting from the DPP. The maximum assignment problem itself is shown to be NP-hard. Detailed explanations and proofs for theorems should be looked up in the original work (Kulesza and Taskar 2012).

Subsection 4.3.3 provides an improvement on the greedy selection algorithm in the terms of execution speed using the structure of the matrix over one provided in the DPP paper.

Subsection 4.3.4 illustrates how DPP works by applying a greedy selection algorithm to a toy problem of selecting non-similar points from a plane. We show the importance of centrality-like features for the task of selecting items that are non-similar, but central at the same time.

4.3.1 Main Ideas

Let $\mathcal{Y}$ be finite **ground set** that contains N **items**. Without loss of generality, let elements of $\mathcal{Y}$ be integers from 1 to N. **Point process** assigns a probability measure for each subset of $\mathcal{Y}$ and there exist 2^N such subsets. $\mathcal{P}$ is called a **determinantal point process** if, when Z is a random subset drawn according to $\mathcal{P}$, for every $A \subseteq \mathcal{Y}$, a marginal probability

$$\mathcal{P}(A \subseteq Z) = \sum_{Z:\begin{cases} Z \subseteq \mathcal{Y} \\ A \subseteq Z \end{cases}} \mathcal{P}(Z) = \det(K_A) \tag{4.1}$$

is defined for some real, symmetric $N \times N$ matrix K that is indexed by elements of $\mathcal{Y}$. K_A denotes restriction of matrix K to the elements of A, $K_A = [K_{i,j}] : i, j \in A$. Matrix K is called a **marginal kernel** of DPP, because it contains all information to compute marginal probabilities of any subset A included in random sample Z.

Limits of summation in the Equation (4.1) are not straightforward. Figure 4.5

displays Venn diagram-like image of sets participating in the equation. Summation goes over all sets Z, however, a set Z should be a superset of A and a subset of $\mathcal{Y}$ at the same time.

For modeling real data, it is difficult to come up with a good marginal kernel beforehand, thus L-**ensembles** are introduced. They define specific probabilities of subset Y being drawn from DPP as

$$\mathcal{P}_L(Z = Y) \propto \det(L_Y), \tag{4.2}$$

where L is a some positive semi-definite (SPD) matrix or a kernel. To get actual probabilities, one needs to compute a normalizer $\sum_{Y \subseteq \mathcal{Y}} \det(L_Y)$. One important theorem for L-ensembles is that for any $A \subseteq \mathcal{Y}$ and any $Y \subseteq \mathcal{Y}$ such that $A \subseteq Y$

$$\mathcal{P}(A \subseteq Y) = \sum_{A \subseteq Y \subseteq \mathcal{Y}} \mathcal{P}_L(Y) = \det(L + I_{\bar{A}}),$$

where $\mathcal{P}(A \subseteq Y)$ is a marginal probability of all A's elements to appear in a random Y sampled from DPP and $I_{\bar{A}}$ is a diagonal matrix with ones in the diagonal positions corresponding to elements of $\bar{A} = \mathcal{Y} \backslash A$. Using this theorem, the connection between L and K kernels is defined as

$$K = L(L + I)^{-1} = I - (L + I)^{-1}. \tag{4.3}$$

Let's consider Equation (4.2) once more. A probability of item to be selected is proportional to an on-diagonal element

$$\mathcal{P}_L(Y = \{i\}) \propto |L_{ii}| = L_{ii}.$$

For two items, the formula becomes

$$\mathcal{P}_L(Y = \{i, j\}) \propto \begin{vmatrix} L_{ii} & L_{ij} \\ L_{ij} & L_{jj} \end{vmatrix} = L_{ii}L_{jj} - L_{ij}^2.$$

The more off-diagonal element L_{ij} is, the less is going be the probability to select these two items together. This means, if L is a similarity matrix, then DPP assigns higher probability to more diverse or globally non-similar subsets of a ground set.

Equation (4.3) could be formulated using eigendecomposition. Because L is a SPD matrix, it can be composed into

$$L = \sum_{i=1}^{N} \lambda_i v_i v_i^T \tag{4.4}$$

where λ_i is a i-th eigenvalue and v_i is a respective eigenvector. Because of the SPD definition, $\lambda_i \geq 0$. In this representation, marginal kernel K could also be calculated from eigendecomposition (4.4) by scaling eigenvalues

$$K = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda_i + 1} v_i v_i^T. \tag{4.5}$$

Converting L kernel to K takes $O(N^3)$ operations in any case, be it matrix inversion or eigendecomposition. Eigendecomposition can be used for sampling from DPP and allows to decrease computation complexity of computing marginal probabilities if elements of L are computed as a dot product of two vectors.

4.3.2 Dual Representation

For the tasks when L is a similarity-like matrix, DPP assigns higher probability to the subsets that are more *diverse*, or contain globally non-similar items. Specifically, if the elements of L are calculated like

$$L_{i,j} = q_i \phi_i^T \phi_j q_j,$$

it is possible to improve the complexity of DPP algorithms. In this representation, there are two types of features: **quality** scalar features q_i and **diversity** (or similarity) vector features ϕ_i of dimension D. In the matrix notation, the L kernel would be

$$L = B^T B,$$

where B is a feature matrix built by stacking individual item feature vectors $q_i \phi_i$. It is known that eigenvalues λ_i for a matrix $B^T B$ are equal to BB^T. Thus, it is possible to compute them using a much smaller matrix $C = BB^T$, which has dimensions $D \times D$ compared to $N \times N$ of a L kernel. This is called a **dual representation** of a DPP.

Dual representation of DPP is also SPD and its eigendecomposition is

$$C = \sum_{n=1}^{D} \lambda_n \hat{v}_n \hat{v}_n^T.$$

It is possible to show (Kulesza and Taskar 2012), p.37 that it is related to L kernel as

$$L = \sum_{n=1}^{D} \lambda_n \left(\frac{1}{\sqrt{\lambda_n}} B^T \hat{v}_n \right) \left(\frac{1}{\sqrt{\lambda_n}} B^T \hat{v}_n \right)^T.$$

Because of this, elements of marginal kernel K could be computed as

$$K_{i,j} = \sum_{n=1}^{D} \frac{\lambda_n}{\lambda_n + 1} \left(\frac{1}{\sqrt{\lambda_n}} B_i^T \hat{v}_n \right) \left(\frac{1}{\sqrt{\lambda_n}} B_j^T \hat{v}_n \right). \tag{4.6}$$

Note, that this computation does not depend on the number of all items N and its complexity is $O(D^2)$. To infer a marginal probability of a set of size k, it is necessary to compute $\frac{k(k+1)}{2}$ elements of K for computing a determinant. This process requires only $O(D^2k^2 + k^3)$ time.

The number of items N is usually fixed by the number of input items and decreasing it can greatly reduce the total goodness of the output. However dimensionality of feature space D could be changed. One useful way of handling large vectors is random projections. DPP paper gives theoretical guarantees on the compression of feature vector dimensions with random projections. The application of random projections is described in Section 4.4.4.

4.3.3 Selecting Items Using DPP

Kuleza et al. (Kulesza and Taskar 2012) show a simple greedy algorithm for selecting diverse subsets using a DPP. However, it is possible to improve on its performance using several facts about the structure of matrices which participate in the algorithm.

Recall that the marginal probability of subset A contained in a sample from DPP Y is defined in the Equation (4.1) as $P(A \subseteq Y) = \det(K_A)$. When doing a greedy selection naively, the probability $P(A \cup i | A \subseteq Y)$ is computed for each item i which are not present in A, that means computing a determinant for selecting one item. Determinant computation using a matrix decomposition is an $O(n^3)$ operation. When selecting items up to subset size k, the total order of operations is $O(Nk^3)$, because the computation for each greedy step will be shadowed by the last step. However, the running time for each step can be reduced to $O(Nk^2)$. The kernel for the subset A is computed as K_A, or elements of marginal kernel K indexed by elements of A. The conditional marginal probability is computed as

$$P(A \cup i | A) = \frac{P(A \cup i)}{P(A)}.$$

For the fixed A, P(A) is constant and when finding i that maximizes the probability $P(A \cup i | A)$, one can do that by finding

$$i = \arg\max_{i \in \mathcal{Y} \setminus A} P(A \cup i).$$

This requires computing a determinant $\det(K_{A\cup i})$ for each $i \in \mathcal{Y}\backslash A$. However, they are of special structure. For the j-th selection step ($j < k$) of the algorithm, the matrix $K_{A\cup i}$ is

$$K_{A\cup i} = \left(\begin{array}{ccc|c} K_{A_1,A_1} & \cdots & K_{A_1,A_{j-1}} & K_{A_1,i} \\ K_{A_1,A_2} & \cdots & K_{A_2,A_{j-1}} & K_{A_2,i} \\ \vdots & \ddots & \vdots & \vdots \\ K_{A_1,A_{j-1}} & \cdots & K_{A_{j-1},A_{j-1}} & K_{A_{i-1},i} \\ \hline K_{A_1,i} & \cdots & K_{A_{i-1},i} & K_{A_{i,i}} \end{array}\right) = \begin{pmatrix} K_A & V \\ V^T & d \end{pmatrix}.$$

When computing determinant of it, in case if $\det K_A$ is invertible, one can write $\det K_{A\cup i}$ in the following manner

$$\det \begin{pmatrix} K_A & V \\ V^T & d \end{pmatrix} = \det K_A \det(d - V^T K_A^{-1} V).$$

Determinant $\det(d - V^T K_A^{-1} V)$ is the value of $(d - V^T K_A^{-1} V)$ because the matrix size is 1×1.

The part $K_A^{-1}V$ could be computed by decomposing $K_A = GDG^T$ and solving a system of linear equations $GDG^T X = V$. This decomposition is usually called LDL^T, but the letter L is already used by L-kernel. G is a lower triangular matrix with ones on the main diagonal and D is a diagonal matrix. Decomposition has the $O(n^3)$ computational complexity but the solution of a linear system is only $O(n^2)$ when the matrix is triangular. Also, this decomposition can be computed once for the selection step and its diagonal matrix D also can be used for computing determinants by taking a product of its elements.

In step j there is $O(j^3)$ operations for the decomposition and $O(Nj^2)$ operations corresponding to computations of determinants for finding marginal probabilities of subsets $A \cup i$ for each item i. If there are k selection steps, the total complexity of improved greedy selection becomes $O(k^3 + Nk^2)$ which is faster than $O(Nk^3)$ for the brute force solution.

For a dual representation, a full matrix K is not needed. Marginal probabilities for selecting a single item lie on the main diagonal of K. For each additional selection step, only a single additional row of K is required. Computing a main diagonal or a row of K means computing $O(N)$ elements using the Equation (4.6), yielding $O(ND^2)$ complexity. Using a dual representation it is possible to get $O(D^3 + ND^2k + Nk^2 + k^3)$ complexity. Members of the sum are for eigendecomposition, calculating a diagonal or a row of K matrix, calculating marginal probabilities and GDG^T decomposition for each selection step respectfully. Additionally, we need to compute $C = B^T B$ as

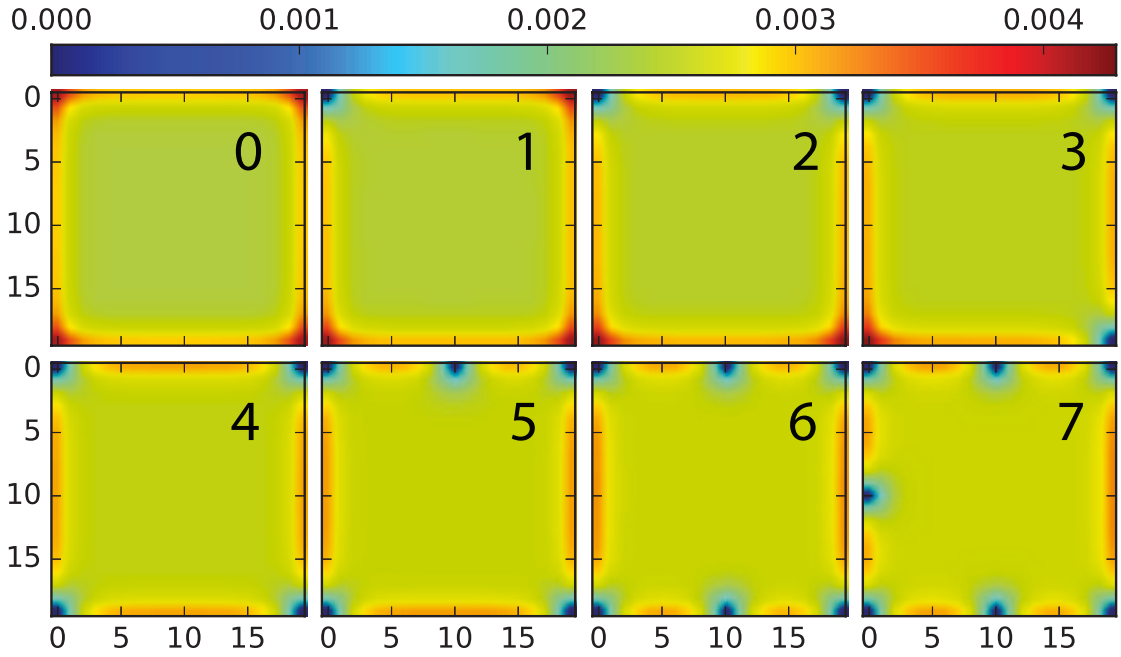


Figure 4.6: Item Selection Marginal Probabilities Heatmap: diversity features only. On each step an item with highest probability is selected. Places with zero probability are from selected items.

well, but the complexity of this operation is $O(ND^2)$ and hidden by computing the elements of K. This means that the computation time is usually a much smaller number for a dual representation if feature dimension size D is small enough than $O(N^3 + Nk^2 + k^3)$ for using full L kernel to construct marginal kernel K. Furthermore, D is usually much bigger than k and last two members can be ignored yielding $O(D^3 + ND^2k)$ complexity for a greedy selection using a dual representation of DPP.

4.3.4 Example: Greedy Selection from Artificial Data

To illustrate usage of DPP as a weighting scheme for a greedy selection algorithm we are going to use an artificial task of selecting points from a plane. We use a grid of 20×20 evenly spaced points for this task. Similarity of two points p_i and p_j is computed from their distance as

$$\mathrm{sim}(p_i, p_j) = \frac{d_m - \mathrm{dist}(p_i, p_j)}{d_m},$$

where $d_m = \max \mathrm{dist}(p_i, p_j)$ for any pair of p_i and p_j. Distance between two points is a Euclidean distance between them. The distance is used as elements of L-kernel:

$$L_{i,j} = \mathrm{sim}(p_i, p_j).$$

Normalized item selection marginal probability heatmap is shown on Figure 4.6.

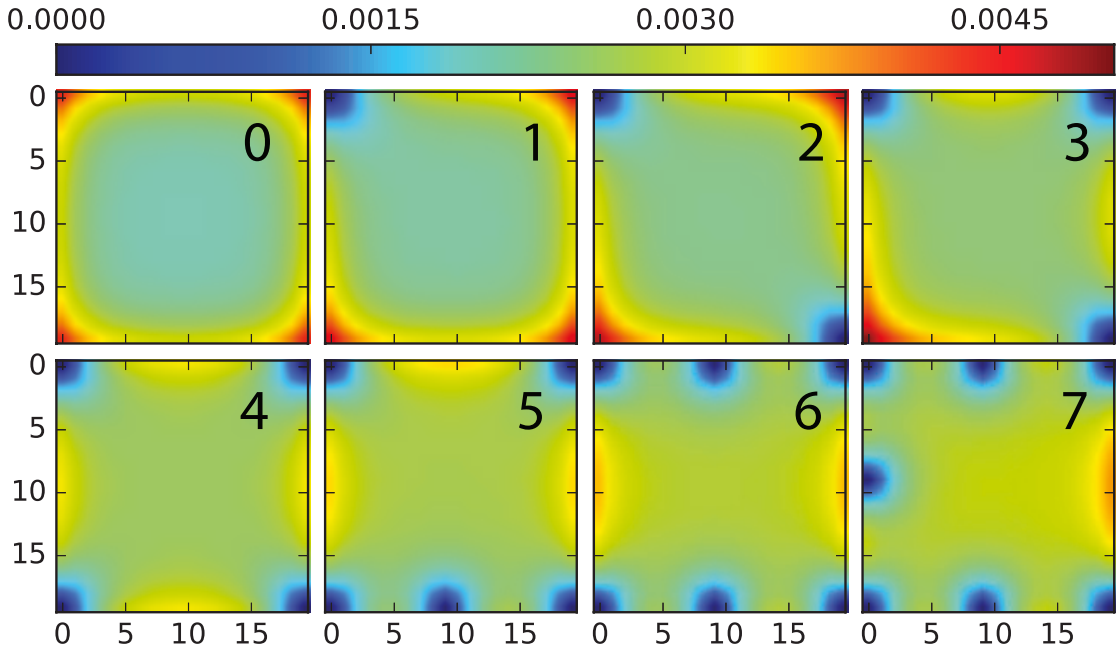


Figure 4.7: Selection Marginal Probabilities Heatmap: diversity features, $r = 0.9$

The top-left image is for initial probability estimation. Other images are created after some points are selected. It is possible to observe that DPP assigns higher probabilities to items on borders and especially in corners.

The reason for that is simple. Items in corners have the highest distance from each other, which means that similarity for them is going to be the lowest. The probability of selecting a set of items according to DPP is proportional to the determinant of indexed L kernel. The determinant value is going to be higher if the off-diagonal elements are smaller, which is exactly the case for points in the corners of the rectangle.

Another observation is that regions of lower probability near selected items are relatively small. Authors of the DPP paper add a parameter r that forces every item to be more similar for their tasks. Intuitively, each point here *is* a point and should not be very different from other points. Using this idea, the L-kernel items are going to be computed as

$$L_{i,j} = r + (1 - r)\,\mathrm{sim}(p_i, p_j)$$

for the artificial toy task.

For the value of $r = 0.9$ selection heatmaps are shown on the Figure 4.7. In this setting, it is possible to see that compared to the Figure 4.6, changes in probabilities after each selection became much rapid.

Still, it is possible to say that the greedy algorithm on diversity features only is going to select non-similar items, however, are not going to be *representative*. Furthermore, selected items are going to be outliers, which is not a useful property

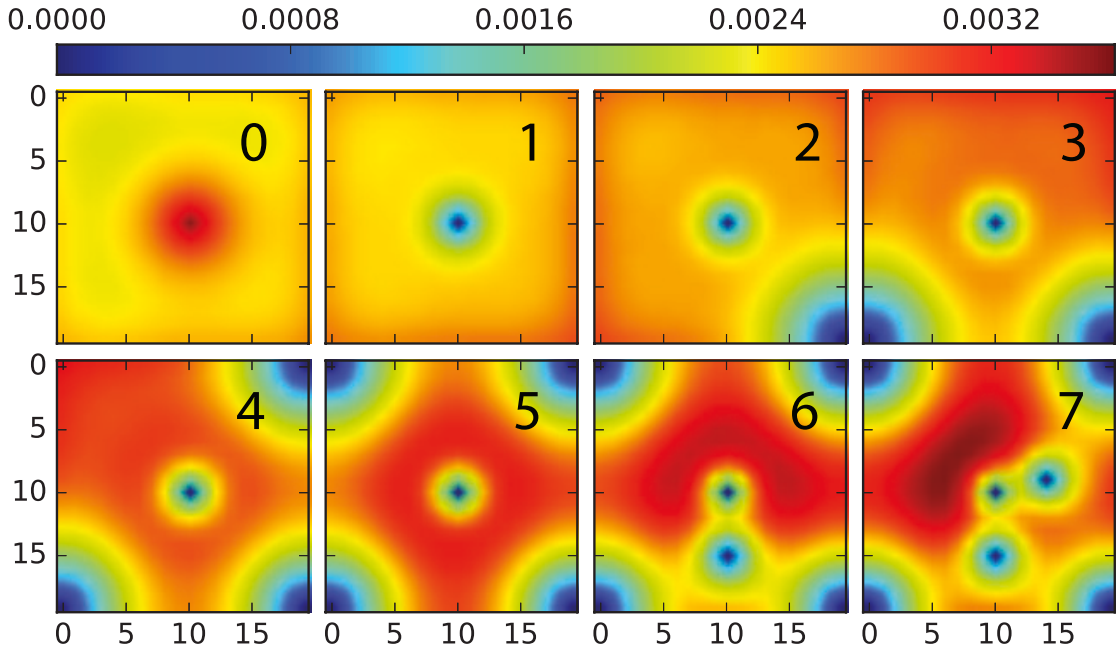


Figure 4.8: Selection Marginal Probabilities Heatmap: diversity and quality features, $r = 0.9$

for selecting example sentences. For the task of summarization, authors of the DPP paper use features that serve a measure of centrality as *quality* features.

Let us see how quality features can change probability distribution. For each item let's introduce a scalar value q_i that is going to be close to 1 if the item is closer to the center of the plane and decreases with the distance from the center of the plane. One possibility is to have

$$q_i = e^{-\operatorname{dist}(p_i, p_0)},$$

where p_0 is the center of the plane. Values of L-kernel are going to be

$$L_{i,j} = r + (1 - r)\operatorname{sim}(p_i, p_j) q_i q_j.$$

The Figure 4.8 shows point selection marginal probability heatmap for the case of using both types of features and the parameter $r = 0.9$. This time the selection starts from the center of the plane which is more useful for selecting items that are non-similar and representative at the same time. For the cases when DPP is used to select items which are central and non-similar at the same time, usage of a centrality-like measure in quality features is essential.

4.4 Features

For the task of example sentence extraction, we adopt the feature representation recommended by authors of the DPP paper. L kernel is computed from vector similarity features ϕ_i and scalar quality features q_i for items i and j as

$$L_{i,j} = q_i \phi_i^T \phi_j q_j.$$

We construct a similarity feature vector as a weighted stacking of three individual feature parts

$$\phi_i = f([w_1 s_i^{lex}; w_2 s_i^{synt}; w_3 s_i^{sema}; r])$$

and a parameter r which makes all sentences similar to each other, following the text summarization task in (Kulesza and Taskar 2012). We set $r = 0.7$ in our experiments, following the recommendations.

Three similarity feature parts are **lexical**, **syntactic** and **semantic** similarity. Feature weights w_i allow us to prioritize similarity feature components. Lexical and syntactic similarity features are created as count-based vectors and have a large dimensionality. The transformation f here is a compression into a 600-dimensional vector using Gaussian random projections as recommended by (Kulesza and Taskar 2012) to make the dimensionality of ϕ_i, D, small.

Quality features represent a *intrinsic value* of individual sentences as examples of word usage. They consist of five components:

- Semantic centrality – similarity to cluster centroids for semantic features. Intuition for them is to select example sentences that have *representative* meanings.
- Syntactic centrality – similarity to cluster centroids for syntactic features.
- Semantic "cardinality" – difference of actual semantic cluster size to an average cluster size. This feature could be used to bias selection in favor of frequent senses or rare ones.
- Relative Difficulty – a coefficient that approximates how sentences are difficult for a learner of a fixed level
- Goodness – coefficient that gives a lower score to sentences that contains bad parts.

Each quality feature is a scalar, they are multiplied together to get a final value for a sentence.

Most of the features are computed not for just a sentence, but for a combination of a sentence and a target. That is done to measure how good the sentence not in a

general sense, but exactly for the target word. Additionally, to support processing on huge datasets some effort was put to make all computations distributable.

4.4.1 Similarity: Lexical

Lexical similarity is an instance of a count-based vector space model from information extraction. Its goal is to measure a simple word overlap between sentences. We use a modified *tf* weighting scheme (without the usual idf factor), content words are given a weight of 1.0; non-content words are given a weight of 0.1. We use the canonicalization feature of JUMAN (代表表記 dictionary feature) for terms to handle Japanese orthographic variants.

We also use the hashing trick for computing term ids. The example extraction system is distributed and sentences from a single list can be processed on different computational nodes. Because of this, the usual solution of using a mapping from a word to an id requires having the mapping to be synchronized across all nodes. Fortunately, hashing trick (Weinberger, Dasgupta, Langford, Smola, and Attenberg 2009) can be used to directly assign ids from word hashes. This steps around the problem of mapping distribution.

4.4.2 Similarity: Syntactic

Syntactic similarity for two sentences should be higher if they have similar syntactic structure near the target word, meaning that it was used in a similar syntactic way. In other words, dependency structure, POS tags, and grammatic words should be similar near the target word.

We use tf-like weighting scheme for syntactic vectors as well. Terms for syntactic vectors, however, are lexically-erased small subtrees containing the target word.

The motivation for syntactic similarity comes from the problem of example databases that provide a search interface. Let us consider the search for word “走る” from the tatoeba.org free example sentence database. On the first page of the search there are 3 sentences: “彼は走るのが速い”, “彼女は走るのが遅い”, “リンは走るのが速い” having more or less different words, but the same syntactic structure. It is more useful to select sentences with different syntactic structures for language learners to show more word usage patterns.

The idea for the syntactic similarity method is based on the efficient calculation of graph similarity using graphlets. Graphlets are parts of a graph, and it is shown by Shervashidze (Shervashidze, Vishwanathan, Petri, Mehlhorn, and Borgwardt 2009) that they can be used for the fast approximate computation of graph similarity. One of the problems raised in that paper was finding out whether two graphs are the

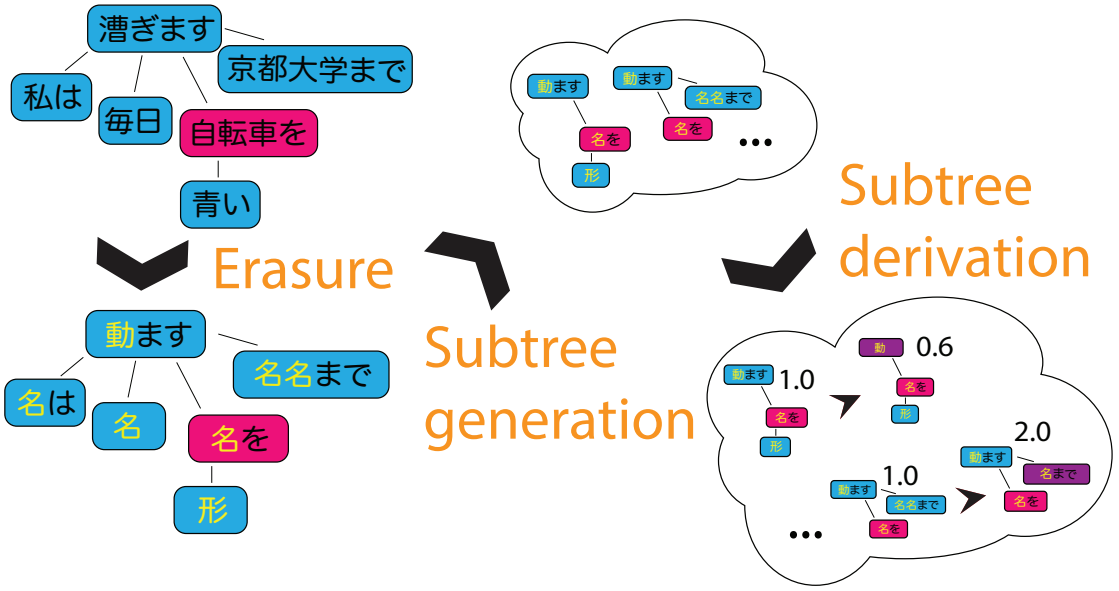


Figure 4.9: Outline of syntactic similarity bag-of-subtrees feature collection creation. Bunsetsu containing the target word is colored.

same – graph isomorphism problem. Case for dependency trees is simpler than the graph problem because unordered tree isomorphism can be solved more efficiently. Dependency parse trees are treated as unordered because in the Japanese language dependencies on a single level could be reordered in arbitrary ways without changing the overall meaning of the sentence.

The outline for the syntactic similarity calculation is shown on the Figure 4.9. The main idea is to generate bunsetsu subtrees up to a certain size, by growing them from the target word and using those subtrees as features in the vector space. Overall, the syntactic similarity model can be thought of as a bag-of-subtrees model. Subtrees are treated as unordered because bunsetsu in Japanese can be moved on the same dependency level. Starting from the original parse tree with the target word (shown in red on Figure 4.9), the process of calculation consists of 4 main steps.

1. Parse tree is **erased** by stripping lexical information for open parts of speech and replacing them with part of speech tags. Grammatic words are left as they were. Parts of speech are shown in yellow on the figure.

2. A set of bunsetsu subtrees up to the size of 3 is generated from the erased tree. Generation starts from the bunsetsu containing the target word and continues until no new subtrees can be created.

3. Extending feature space by generating **derived** subtrees using a simple ruleset and adding those subtrees into the resulting subtree set, but with different weights. Bunsetsu that were rewritten in the derivation process are shown in the Figure in purple.

4. Generating vector elements by using the weight of subtree features as vector values and subtree **hashes** directly as vector indices. Hashing eases the problem of working with unordered trees.

Feature calculation steps are discussed in more detail below.

Erased trees

The original parse tree is erased to ignore lexical information from the sentence and focus solely on the syntactic structure causing grammatically and structurally similar sentences to have a higher similarity score. Using the target word as a starting place for the tree growing makes a further focus on concrete syntactic usage of the target word, mostly ignoring parts of the sentence that are far from the target word.

Generating subtrees

Subtrees that are going to be elements of the syntactic similarity vector are generated by growing them from the bunsetsu containing the target the word until the tree contains up to n bunsetsu nodes. In experiments, n was chosen to be 3. The number of all possible subtrees is exponential in n, thus choosing it too big is going to yield a very large number of subtrees.

Growing a subtree from the target node ensures that the feature will capture only the vicinity of the target word and ignore the information that is too far from the target node. This idea of *focusing* makes the method different from other methods of measuring tree similarity, like tree edit distance or tree kernels. General methods measure the overall similarity of two trees, however, the proposed method measures only the similarity near the target node. Arguably, it makes the method more appropriate for measuring syntactical similarity in the context of example sentences with a pin-pointed word.

Subtree derivation

Generated set of bunsetsu is going to be sparse. Some exact language constructs have mostly the same grammatical meaning and should be generalized to improve coverage of the algorithm. For example, a bunsetsu “京都大学” when erased is going to be represented as “⟨名詞⟩⟨名詞⟩”. However, from a syntactic point of view, it is almost the same as a sentence having only a “大学” word, especially if it is not an example target. Similarly, a phrase “そうですね” is mostly the same as “そうだよ”. This problem is combated with **deriving** new subtrees by making a copy of a subtree and **rewriting** the offending bunsetsu to a more general variant. Derived subtree is then put into the resulting subtree set.

Table 4.1: Bunsetsu rewrite rules for deriving subtrees. The target word is colored.

Name	Example	Factor	Derived *mass*
Remove final particles	そうだね→そうだ	3.0	0.8
Remove successive nouns	京都大学→大学	2.0	0.7
Remove successive verbs	乗り切る→乗る	1.5	0.6
Remove *non-target* verbal suffixes	白く咲きそうな→白く咲く	3.0	0.8
Remove *other* "suffixes"	飲みそうな→飲む	1.0	0.5
Remove auxiliary verbs	できません→できます	1.0	0.5
Remove case markers	学校に→学校	0.7	0.4

A problem with this approach is that adding new subtrees changes "similarity mass" (this term has the same intuition as in the relation between probability and probability mass) distribution of the entire syntactical vector. For example, if one sentence had 5 different original subtrees with "⟨名詞⟩⟨名詞⟩" bunsetsu and 5 derived subtrees with "⟨名詞⟩" bunsetsu, and another sentence had only "⟨名詞⟩" bunsetsu pattern, the total similarity would be only 0.5 if those subtrees were completely same except for compound noun. By setting different weights for newly created subtrees it is possible to work around this issue and decide the place where the similarity mass is going to be put.

Table 4.1 shows rules that are used for bunsetsu *rewriting* in the tree derivation process. Examples are shown in their lexical form, but should be thought *only* as operations, resulting in syntactic and grammatical changes in the parse tree. Factor column specifies a multiplier for a weight of current subtree `orig`, so rewritten subtree weight is going to be $\texttt{rewr} = \texttt{orig} \cdot \texttt{factor}$. Subtree set has both original and derived versions of the subtrees and they both are going to be used as features. Other sentences, that have only derived subtrees matching their subtrees, are going to have only the proportion of similarity mass that is specified in the *derived mass* column because of vector normalization. If two sentences have non-derived subtrees matching, it means that they had similar structure originally and should have both parts of similarity mass: original and derived.

Factors for rules were chosen with the general direction that factor should be greater than one whether rewriting does not change *general syntactic meaning* of the sentence. Otherwise, a factor of less than 1 was chosen.

The first rule erases final particles like ね or よ that have no syntactic meaning in the sentence. The second rule rewrites successive nouns to a single noun. Syntactically, compound nouns are mostly the same as single nouns, so the rule moves similarity mass from original subtrees to the rewritten ones, enabling higher scores when matching with other subtrees that are simple from the beginning. The third rule does the same for the compound verbs, but with more conservative scoring, because there are grammatical combinations of 連用形 form of verbs with auxiliary verbs like

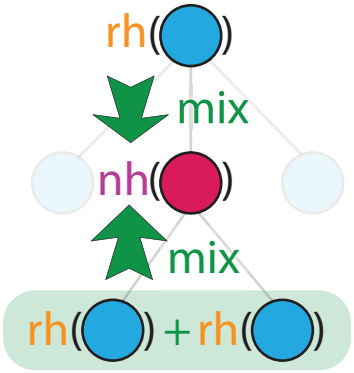


Figure 4.10: Hash function sketch for a subtree. A node containing the target word is colored.

逃げ出す or 逃げ切る that are not purely lexical. The fourth rule conjugation suffixes from *non-target* words. Non-target conjugations are not as crucial for the example sentence as target ones, erasing them does not change the syntactic meaning of the sentence a lot.

The remaining rules remove grammatical suffixes from words, essentially stripping them from the grammatical meaning, increasing recall of the similarity function instead. Because of that, they put most of the similarity mass to non-rewritten subtrees.

Hashing subtrees

To convert a set of subtrees to a vector form, each subtree should be assigned a number. Converting subtree to a number can be done by building a string representation from some fixed tree traversal and then assigning a number to that string, for example by storing strings with their associated numbers in a hash map. However, creating a canonical string representation for an unordered tree is non-trivial. Furthermore, distributed construction of vectors, while having a shared mapping from a string representation to a vector requires keeping a synchronized mapping on each node and is impractical.

Calculating directly a hash for subtrees is a solution that does not have both mentioned drawbacks. Hash value calculation from data usually consists of operations of *mixing* individual values together. MurmurHash3[8] is a family of general mixing functions that can be used for creating specialized hash functions for the data. The design of a hash function for subtree is closely related to the tree traversal. Idea for it is displayed on Figure 4.10.

A recursive function `rh(node)` is called for a subtree node. It mixes three components using a MurmurHash3 algorithm: a hash value of its content `nh(node)`,

[8] https://code.google.com/p/smhasher/wiki/MurmurHash3

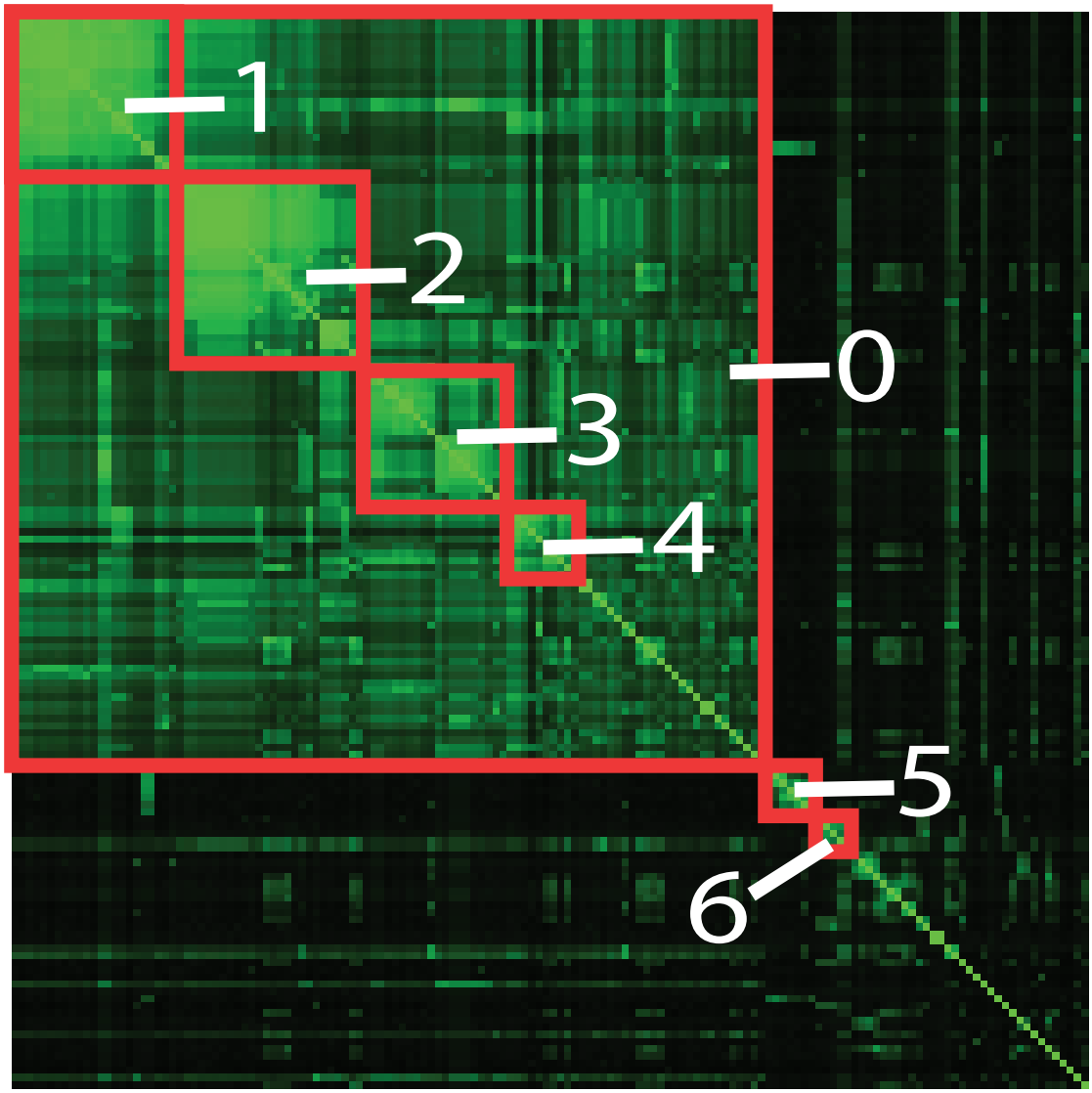


Figure 4.11: Agglomerative clustering on a syntax similarity for 150 random sentences with the word 黒い

a result of a recursive call to own parent, and a sum of values of recursive calls to own children. Because hash values for children are mixed with a commutative function (plus), the resulting hash function is invariant for the order of children nodes. Additionally, the directions of mixing are mixed into respective hash values as well, separating children from parents.

Agglomerative clustering on syntactic similarity

Figure 4.11 displays agglomerative clustering result on syntactic similarity for similarity matrix for the word "黒い". Lighter colors represent the higher similarity between two sentences. Clusters are empathized by red squares. Cluster 0 contains sentences that have word 黒い being a dependent of a noun, like in sentence 黒い

犬. Inner clusters have parent of 黒い to have different role in a sentence:

1. noun being a subject (黒い犬が吠える)
2. object (黒い犬を飼う)
3. dependent of another noun with の (黒い犬のえさ)
4. sentences when noun has at least two dependents: 黒い and some another noun like in sentence 寝ている黒い猫
5. 黒い being a dependent to a verb, like in sentence 黒く染まる
6. 黒い being dependent to another adjective like in 黒くて甘い

It can be said that a set of syntactic similarity features capture a *structural usage pattern* of a word in an example sentence. By clustering on the syntactic similarity, it is possible to distinguish patterns that are frequently used with the target word or expression from bad sentences that are present in the web data.

4.4.3 Similarity: Semantic

The semantic similarity score should be higher if the target word is used in the same or a close sense. Other words in the sentence should not directly change the measure.

It is possible to have **embeddings** (vector representations) for words. These embeddings can be trained from raw text and have interesting properties. For example, word2vec method (Mikolov, Yih, and Zweig 2013) give embeddings that have linguistic regularities. For example, if d(king) is an embedding for the word "king", then it is possible show things like

$$d(\text{queen}) - d(\text{woman}) + d(\text{man}) \approx d(\text{king}).$$

Prototype Projections

Unfortunately, such methods usually have only a single vector representation for a word, meaning it is impossible to distinguish different senses of a word. However, there are method called **prototype projections** (Tsubaki, Duh, Shimbo, and Matsumoto 2013).

Consider a binary relation $R(a_1, a_2)$ between two words, for example, a relation between a predicate and object with the loss of generality. If we fix a verb in the a_1 slot, then frequently occurring words that can take place in the a_2 slot are going to form a set of **prototype words**. Those prototype words somehow define different

senses of the target word. For example, a verb would be "掛ける", then the set of prototype words would contain 声, 電話, 迷惑, 副, 鍵. In Japanese を marks an object for a predicate, and those words usually fill the を case slot in sentences with 掛ける. If we have a parsed corpus, then the set of prototype words $\{w_1, w_2, ..., w_m\}$ can easily be computed.

By using embeddings for the prototype words $d(w_i)$ it is possible to construct a **prototype matrix** C. Matrix is built by simply stacking embedding vectors for the prototype words on top of each other

$$C = [d(w_1), d(w_2), \cdots, d(w_m)]^T.$$

A singular value decomposition (SVD) of matrix $C = U\Sigma V^T$ is going to contain "the most important" directions of the prototype words in the right singular vectors V_i^T corresponding to the largest singular values Σ_i. By dropping the singular vectors corresponding to the smallest singular values, it is possible to get a **prototype space**. It is represented by a matrix $\Sigma_{0:k}V_{0:k}^T$ for k highest singular values. It is possible to create a projection matrix to this subspace in the original space by creating a matrix

$$P_{R,a_1} = (\Sigma_{0:k}V_{0:k}^T)^T(\Sigma_{0:k}V_{0:k}^T).$$

An embedding $d(w_0)$ of a word w_0 projected into a prototype subspace spanned by the word the slot a_1 over the relation R is going to be denoted as

$$w_0|_{a_1}^R = P_{R,a_1}d(w_0).$$

By applying prototype projection to the both ends of the relation, and combining the together by summing them it is possible to get a vector representation for the whole relation instance $d(R(a_1, a_2)) = a_1|_{a_2}^R + a_2|_{a_1}^R$. This representation is going to have high cosine similarity to relation instances of the similar meaning and low for usages in a different sense. The Figure 4.12 displays a construction of a vector representation for a phrase "迷惑を掛ける". It consists of a relation over を case and two prototype projections. The first uses the predicates of "迷惑" over the を case as prototype words and another uses the arguments of "掛ける" over the same case.

Authors have shown that this method is very good for detecting paraphrases like "run a company" versus "operate a company". However, if it can detect semantic information even in distinct words, it should be possible to use it for coarse unsupervised word sense disambiguation as well.

Word embeddings for representations were trained from the web corpus with 0.7B sentences. Lists of frequently used arguments were computed from Japanese

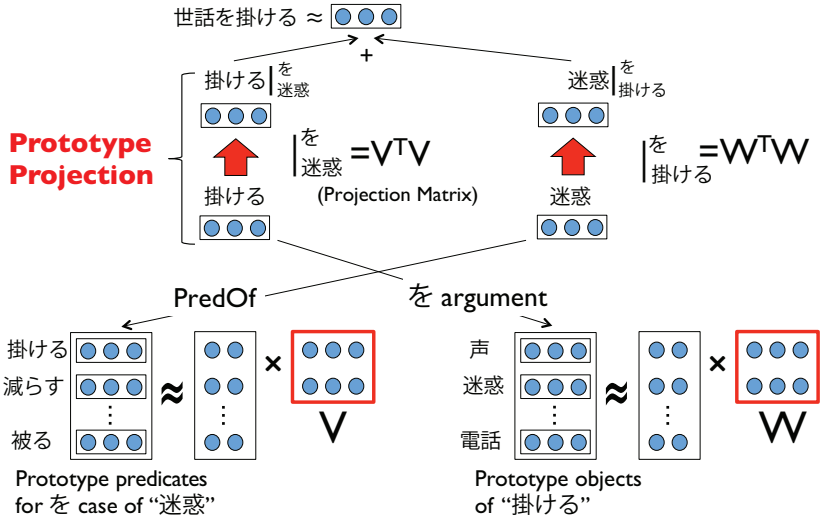


Figure 4.12: Protype Projection for a phrase "迷惑を掛ける"

Table 4.2: Similarity for Prototype Projection ○|を 掛ける + 掛ける|を ○, 0 : k, $d_{l\%} = 0.2$

	世話	迷惑	負担	電話	コート	眼鏡	鍵	声
世話	1.000	0.877	0.774	0.734	0.595	0.712	0.686	0.753
迷惑	0.877	1.000	0.834	0.716	0.559	0.716	0.667	0.783
負担	0.774	0.834	1.000	0.696	0.578	0.722	0.707	0.764
電話	0.734	0.716	0.696	1.000	0.728	0.825	0.820	0.796
コート	0.595	0.559	0.578	0.728	1.000	0.872	0.795	0.694
眼鏡	0.712	0.716	0.722	0.825	0.872	1.000	0.883	0.809
鍵	0.686	0.667	0.707	0.820	0.795	0.883	1.000	0.775
声	0.753	0.783	0.764	0.796	0.694	0.809	0.775	1.000

case frames (D. Kawahara and Kurohashi 2006) by simple aggregation taking top 200 arguments on each side.

Improving Similarity

Subspaces are created by performing an SVD on a matrix composed of stacked word vectors. Words in the matrix are frequently occurring with the target over the relation. The subspace is created using parts of singular values Σ and right singular vectors V^T for the SVD. The original setting of creating a prototype projection is to drop some percentage of right singular vectors corresponding to lowest $d_{l\%} = 0.2$ part of all singular values. This means that k in the expression $\Sigma_{0:k} V^T_{0:k}$ is going to be computed so it will include $1 - d_{\%}$ percent of all singular values.

Table 4.2 shows を case semantic similarity for the for the かける as a predicate and words 電話, 世話, 迷惑, 負担, コート and 鍵 as arguments. The method seems to work for the definition of working that similarity for the closer meaning should be higher than for distant ones. There seems to be a certain cluster for the words 世

Table 4.3: Similarity of Prototype Projection $\bigcirc|^{\text{を}}_{\text{掛ける}}$ + 掛ける$|^{\text{を}}_{\bigcirc}$, 1 : k, $d_{l\%} = 0.5$

	世話	迷惑	負担	電話	コート	眼鏡	鍵	声
世話	1.000	0.746	0.661	0.415	-0.105	0.052	0.260	0.534
迷惑	0.746	1.000	0.522	0.283	-0.348	-0.085	0.024	0.465
負担	0.661	0.522	1.000	0.279	0.028	0.013	0.257	0.487
電話	0.415	0.283	0.279	1.000	0.217	0.452	0.438	0.481
コート	-0.105	-0.348	0.028	0.217	1.000	0.539	0.306	-0.076
眼鏡	0.052	-0.085	0.013	0.452	0.539	1.000	0.500	0.183
鍵	0.260	0.024	0.257	0.438	0.306	0.500	1.000	0.299
声	0.534	0.465	0.487	0.481	-0.076	0.183	0.299	1.000

話, 迷惑, 負担 and their relation for the application to humans. Their similarity with other selected words is lower. On the other hand, 電話 has the highest similarity with 鍵, but 声 is still higher than other words.

For the task of phrase similarity, this model was good, however for the task of assigning lower similarity for semantically distinct senses and higher similarity to semantically close words it does not produce a similar result.

SVD is related to eigendecomposition, which means that top vectors are, speaking informally, *main* components of every vector in the matrix. Assuming that the singular vector related to the highest singular value holds the information of the "word itself" and other vectors hold semantic information of its usage in different contexts, it would make sense to discard *top* singular vector as well. Table 4.3 shows similarity of prototype projections for the same expression when expression for computing prototype subspace is $\Sigma_{1:k}V^{T}_{1:k}$, dropping a top singular vector and bottom singular values related to $d_{l\%} = 0.5$ singular values.

Using prototype projections in this setting is more suitable for detecting different senses, however, parts of projection themselves are interesting as well. Actually, in the setting of $d_{l\%} = 0.2$ projections かける into subspaces spanned by each word かける$|_{\circ}$ yielded almost similar vectors, most having similarity of > 0.9. Summing them to projection $\circ|_{\text{かける}}$ created a situation that almost every similarity is high. For setting $d_{l\%} = 0.5$, 1 : k, projection かける$|_{\circ}$ had lower similarities, however they were not usable for similarity calculations. In contrast to that, projection $\circ|_{\text{かける}}$ yielded much more "clean" matrix in sense that it clearly separates *semantically close* senses. It is presented in Table 4.4.

Computing Semantic Representations

Prototype projection as a method allows computing a representation for a pair of words and a certain relation like predicate-wo-argument. However, it is only a part of a sentence and there could be different arguments possibly modifying the meaning

Table 4.4: Prototype Projection similarity of $\bigcirc|^{\text{を}}_{\text{掛ける}}$, 1 : k, $d_{l\%} = 0.5$

	世話	迷惑	負担	電話	コート	眼鏡	鍵	声
世話	1.000	0.813	0.473	0.070	-0.470	-0.324	-0.231	0.235
迷惑	0.813	1.000	0.532	0.044	-0.550	-0.408	-0.352	0.315
負担	0.473	0.532	1.000	-0.089	-0.466	-0.367	-0.261	0.184
電話	0.070	0.044	-0.089	1.000	0.132	0.250	0.181	0.031
コート	-0.470	-0.550	-0.466	0.132	1.000	0.729	0.239	-0.397
眼鏡	-0.324	-0.408	-0.367	0.250	0.729	1.000	0.373	-0.317
鍵	-0.231	-0.352	-0.261	0.181	0.239	0.373	1.000	-0.293
声	0.235	0.315	0.184	0.031	-0.397	-0.317	-0.293	1.000

of the word. Output semantic representation a target word in a sentence was combined from such elementary representation. If $v_s^x(t)$ is a elementary semantic vector representation for a target word t over a relation x, then a full representation $v_s(t)$ is going to be a normalized sum of elementary representation

$$v_s(t) = \sum_i w_i v_s^i(t),$$

where w_i is a weight of an elementary representation. Parameters of a prototype projection were 1 : k and $d_{l\%} = 0.5$. Depending on a part of speech, there are 4 variations in how exactly the representation was created.

Semantic representations for verbs and adjectives were tested as a word sense disambiguation task on the Japanese SemEval (Okumura, Shirai, Komiya, and Yokono 2010) data. Using semantic representations with a simple nearest neighbor classifier achieved significantly higher accuracy than the most frequent sense baseline. For more details please refer to Appendix.

Verbs For verbs, the main information for creating semantic vectors is the predicate-argument structure of a sentence. The verb is going to be a predicate and its arguments can be used to compute semantic vector representation. Data for collocations can be aggregated from the case frames. The following list of cases was used for a summation: を, が, に, と, から, まで, 修飾 and で. For all cases except で, weigts w_i were set to 1 and elementary representation was computed as $t|_{\bigcirc} + \bigcirc|_t$. For the で case only projection of a target word $t|_{\bigcirc}$ was used and the weight was set to 0.5.

Adjectives Adjectives are treated the same as verbs. Moreover, they usually use only が case and the meaning mostly depends on the argument of が case slot.

Adverbs Adverbs usually modify a verb and their meaning can depend on the verb's subject or object, especially if a verb is almost auxiliary like する. Adverbs themselves frequently occupy 修飾 slot of a parent verb as well. Because of this, if an adverb is a child of a verb, representation-wise it can be treated as the verb itself, albeit with the reduced number of cases used. Only 修飾 and が cases are used for computing the semantic representation. The adverb's *parent* is used as a target.

If an adverb is a predicate by itself, the algorithm for verbs is used without modifications.

Nouns Nouns can create a relationship that changes meaning using a dependency with another noun and a の particle. For example, a phrase "私の犬" is certainly about a dog, but a phrase "警察の犬" can have a different meaning. Thus, this data should be used as well. Original case frames operate only on predicate-argument structure, so this data was collected from a web corpus. Frequent words over this relation were collected to make case frame-like aggregated lists.

If a noun is a predicate of the sentence, then, as in the verb case, predicate-argument analysis information is used in addition to a relation over の if it exists. Otherwise, in addition to a relation over の, a relation between a noun and its parent is used to compute a semantic similarity vector.

4.4.4 Random Projections

Using counting approaches for creating word vectors results in high dimensionality. Usually, the dimensionality is equal to dictionary size. However such sizes are unwieldy for DPP dual representation.

Random projections is a family of methods that allow to decrease the number of dimensions of a data, while keeping some of the original data properties.

Recall that DPP features can be represented as a matrix B. Rows of this matrix are vectors for each data item. Using lexical and syntactic similarity features, which were described in previous subsections, is going to produce vectors of large dimensionality M, in hundreds of thousands. However, DPP works efficiently if the number of feature dimensions is relatively low.

Gaussian Random Projection is defined by a matrix P of dimensionality $M \times D$, with the elements of P independently drawn from a normal distribution $\mathcal{N}(0, \frac{1}{M})$, having a zero mean and a variance of $\frac{1}{M}$. A classical result on Random Projections (Johnson and Lindenstrauss 1984) shows that even if $D \propto \log(M)$, with a high probability distance between original points are approximately equal to the distances between the projected ones. DPP paper explains the applicability of those results to DPP case in great detail.

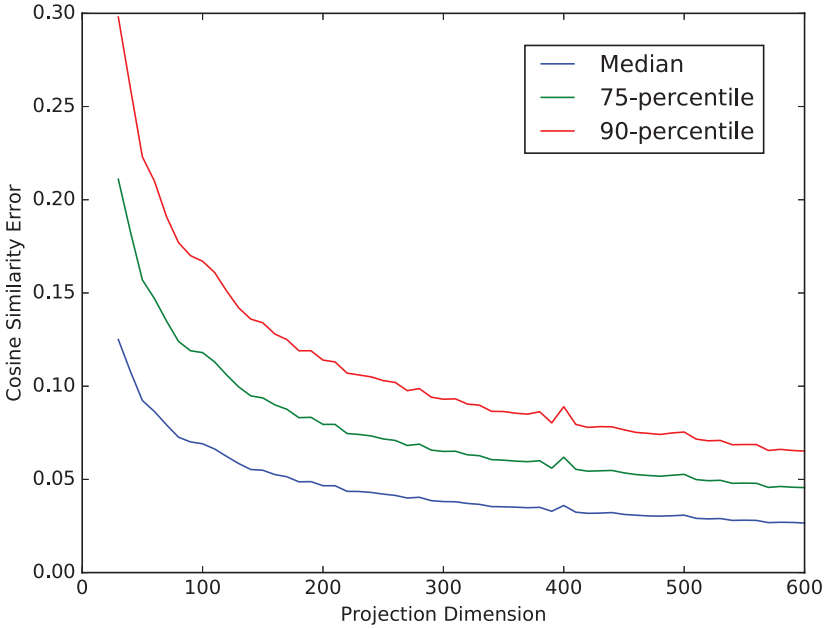


Figure 4.13: Cosine similarity error of a random projection

For the example extraction, random projection is used to compress lexical, syntactic, and semantic parts of similarity vectors into a single similarity vector. The size of that vector is chosen experimentally to have $D = 250$. Figure 4.13 displays cosine similarity errors of projected vectors compared to non-projected ones. For vectors a and b, and a projection matrix P the cosine similarity error is $|a^T b - (Pa)^T Pb|$.

For this experiment, 1000 sentences were used. Original dimensions of vectors were 100k, 400k and 300 for lexical, syntactic, and semantic similarities respectively. The semantic similarity vector is small and dense, however, after a prototype projection, its effective dimension is smaller than 300. To ease further operations, a concatenated vector of all similarity features was projected into a smaller dimension. For the selected value D of 250, 90% of similarity errors are less than 0.1. Moreover, those errors usually occur when vectors are nearly perpendicular, meaning that their dot product is close to 0. This was not the case for vectors close to each other. Still, when D becomes very small, the approximation error becomes very large.

4.4.5 Quality: Centrality

One of the main objectives for example sentence is for them to being representative. If the sentence is semantically representative, that means the sentence uses the target word in a frequently used sense. If the sentence is syntactically representative, that means the sentence used the target word in a frequent grammatical pattern. For example, a noun can be frequently used in a certain case with a verb.

This parameter could be treated as a sort of "centrality" of individual usage

Table 4.5: Word difficulties based on word frequency rank

Max Word Rank	Word Difficulty
500	0
1000	1
2000	2
5000	3
10000	4
20000	5
50000	6
Rest	7

inside a sentences sample. This centrality could be calculated as a distance to the nearest centroid after an application of a clustering procedure over a fixed similarity measure. For the semantic centrality, the distance measure naturally could be created from a semantic similarity measure. The idea is the same for syntactical centrality.

For the clustering K-Means++ algorithm was used because it does not require a similarity matrix and is scalable to large datasets. A number of clusters k was set to 30 for semantic centrality. For syntactic centrality, the number of clusters was set to 10 because syntactic diversity is usually lower than semantic one.

4.4.6 Quality: Difficulty

Example sentences should be understandable for learners. Most of the time, the target word should not be "shrouded" in other words. To do that, the difficulty of the sentence is adapted as a quality feature. It is done in two steps. First, the difficulty of the sentence is estimated as a single number. Then the difficulty is transformed to a quality coefficient, a number from 0 to 1. This difficulty quality coefficient is multiplied by other quality features.

A sentence difficulty consists of several factors: grammar difficulty, lexical difficulty, presence of anaphora and probably many others. However, a simple lexical feature is probably should be a good first approximation to this task. Lexical sentence difficulty d_s of the sentences is approximated from the word difficulties using the formula

$$d_s = \left(\sum_{w_i \in s} d_{w_i}^4 \right)^{\frac{1}{4}}.$$

Here d_w is a word difficulty, and w_i are all content words in a sentence. The idea was to make a softmax-like function, that will still accumulate difficulty if a sentence contains a lot of medium words. Power of 4 seemed to be the best option.

A word difficulty is estimated using word frequencies in a corpus and JLPT word

Table 4.6: Frequency-based difficulty for words in JLPT lists. Bold entries were moved to lexical difficulty corresponding to the JLPT level

	Difficulty							
JLPT Level	0	1	2	3	4	5	6	7
N5	182	92	**172**	**84**	**37**	**14**	**5**	**3**
N4	116	68	266	**126**	**50**	**20**	**5**	**1**
N3	167	84	692	492	**187**	**100**	**18**	**3**
N2	49	30	218	388	413	**460**	**207**	**19**
N1	138	73	406	595	648	896	**451**	**42**

lists. First, words are ranked by their frequencies in the corpus. After that, words are assigned a difficulty number based on their rank. Assignments are displayed in the Table 4.5. After the first approximation of the difficulty, the ranked list is merged with JLPT (Japanese Language Proficiency Test) word lists. Some words that have relatively low frequency, yet still are present in relatively low JLPT levels and should not be treated as very difficult ones.

Merging with the lists is done by moving the words which have greater frequency difficulty than JLPT difficulty. Word JLPT difficulty was chosen to correspond with the frequency difficulty. JLPT level N5 was assigned difficulty 1 and JLPT level N1 to the difficulty 5. Frequency-based difficulties for words are shown in the Table 4.6. Bold entries correspond to words that were moved into position so they would have the difficulty specified by JLPT lists.

Here are some examples of words that had frequency difficulty 5 or 6, but much lower JLPT difficulty. For JLPT N5 ($d_w = 1$) moved words were something like おまわりさん, 花瓶, 作文 or 上着 which often occur in beginner level textbooks, but do not have high frequency in the actual language. For N4 ($d_w = 2$), words like 乗り物, 郊外, 押し入れ or 水泳 were moved.

After the computation of the sentence difficulty, the value is transformed into a quality coefficient. It is transformed using a sigmoid-like piecewise linear function $q_d(d_s)$ that is shown on the Figure 4.14. The idea is to have a slower initial quality decrease when the sentences are not difficult and a rapid decrease when the difficulty becomes larger. The default configuration of this difficulty conversion function stops giving high qualities after the sentence difficulty becomes more than 4, which is about JLPT N3 – intermediate Japanese proficiency level.

It is possible to modify the difficulty function as $q = q_d(d_s + \text{bias}_d)$, where the bias term would make sentences easier or harder, making it possible to select sentences for learners of other proficiency levels.

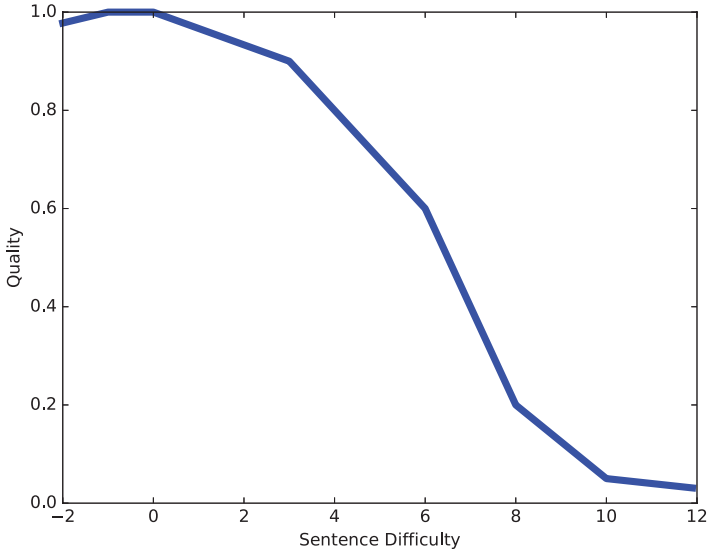


Figure 4.14: Sentence difficulty to quality coefficient conversion

4.4.7 Quality: Goodness

The final part of quality features performs soft filtering on the sentences. Web corpus contains a large number of sentences that are sentence fragments, missing some parts. For example, they could begin with a case marker like “に誰も入ってこない。”. Other sentences could not be suitable for example sentences by simple cosmetic criteria. For example, sentences that contain punctuation that hints that the sentence is unfinished are usually not helpful as an example.

Each example sentence is rated by a series of rules each of that outputs a number from 0 to 1. The product of those numbers is going to be the final “goodness” quality measure.

There are three main types of rules. The first type tries to find out sentences that are definitely will not be good examples, like ones that begin or end with a case marker. It assigns a low score, near 0.2 or 0.3 to such sentences making the probability of them appearing in results extremely low.

The second one ranks things that are normal in small quantities but are bad in large. For example, lots of punctuation in a sentence usually makes it a bad example. Another example is that sentences that contain random numbers and alphabet are usually something like part numbers of some items or identifiers and are useless as examples. However, a single number in a sentence is perfectly fine.

The third type is something in between. For example, words that were automatically acquired from Wikipedia are not useful in example sentences.

4.4.8 Related Work

It is possible to use another language to perform the word sense disambiguation (de Melo and Weikum 2009) for example sentence extraction. One more important feature of that work is a concern about *diversity* of example sentences. They generate a set of 1,2,3-grams for each example sentence and use them for scoring example sentences. After one sentence is selected, all scores for n-grams that are contained in the selected sentence are set to zero, decreasing scores of similar sentences. It is possible to say, that de Melo work considers mostly lexical diversity and centrality, and does not consider sentence difficulty. Sentence difficulty turns out to be a major factor in the evaluation by language learners.

4.5 Evaluation

Evaluating the suitability of example sentences for learning a foreign language is difficult. Evaluating the sentences one by one does not determine the diversity of the extracted sentence list.

The automatic evaluation of example sentences is possible if the problem is formulated such that the only criterion is that example sentences should be present for every sense of a word. However, this evaluation does not determine whether the example sentences are useful for learners.

We perform an evaluation experiment with learners and a teacher with two distinct main goals:

1. To assess the performance of the example extraction system;
2. To validate the assumptions on the meaning of the "quality" of example sentences.

The first goal is achieved by having participants vote on lists of example sentences and select their preferred lists.

For the second goal, the evaluation was performed in the form of an interview. Participants were asked why they have or have not chosen specific lists of example sentences after the initial preference selection.

4.5.1 Baselines

We used three methods in the evaluation: the proposed one and two baselines. The proposed method is labeled **DPP** in the evaluation results.

The first baseline was a method by de Melo (de Melo and Weikum 2009), explained in the Section 4.4.8. However, because our setting uses only monolingual corpora,

only lexical centrality and diversity parts were used from this method. As an additional point, the method did not use raw sentences as input, it was using results of a search engine instead. The method is referred to as **DeMelo**.

The second baseline was a simple uniform random sampling without replacement. The input data, as in **DeMelo**, was a list of example sentence candidates from a search system, not raw examples. Random sampling should have high diversity in many aspects, however, its results could not be consistent. This method is referred to as **Rand**.

4.5.2 Data Preparation

For the experiment we have selected following 14 words:

- verbs: 飛ぶ, 積む, 走る, 取る, 掛ける;
- adjectives: 青い, 汚い, 鋭い;
- adverbs: 全然, バリバリ, サッパリ;
- nouns: 頭, 卵, 足.

Each word has more than one sense and diverse usage patterns. Most of the words are relatively easy and should be familiar to language learners.

For each of the words, we used the top 10k search results from the search engine as example sentence candidates. Each of the words had more than 10k containing sentences. After that, 12 sentences were extracted by each method from each list. That yields a total of $14 \times 12 \times 3$ sentences which were presented to participants of the experiment.

For our system, we selected the original dimensions of vectors to be 100k, 400k and 300 for lexical, syntactic, and semantic similarities respectively. The semantic similarity vector is small and dense, and after a prototype projection, its effective dimension is smaller than 300. We selected the resulting dimension of random projection D to be 250, so that 95% of similarity errors are less than 0.1:

$$|sim(a, b) - sim(P(a), P(b))| < 0.1.$$

Here a and b are two full sentence representations, consisting of all three components, and P is a random projection. We used weights 0.6, 1.0, 1.0 for lexical, syntactic and semantic similariy vector parts respectively.

4.5.3 Experiment Results

The first part of the evaluation experiment used Japanese language learners as participants. For each word, participants were given three lists of example sentences produced by three methods. The lists were placed side by side in a random order to force participants to read sentence lists in a different order every time. Participants were asked to select a list that was more useful from their point of view for putting sentences on the flashcards. After a participant would select a personally preferable list, anonymized names for the methods were displayed and the participant was asked to explain the reasons behind their choice.

Before the experiment, participants were shown the experiment guidelines, consisting of three main points:

- a brief introduction, explaining about flashcards and usage of example sentences in flashcards;
- explanation of the experiment;
- data handling policy.

The experiment explanation itself was exactly the following text:

- You are going to see automatically-collected example sentences.
- Sentences are going to be created for 14 words: 5 verbs, 3 adjectives, 3 adverbs, and 3 nouns.
- Sentences will be grouped into 3 lists.
- You should select a list which you would prefer to use for creating flashcards for a word.

No explicit criteria for selecting the best list of example sentences was given. Instead of that, participants were asked to explain the selection in an interview-like manner to satisfy the second goal of the experiment.

In total, evaluation and interviews were performed with 23 learners. The first stage of evaluation had 11 participants (1-11) and the additional evaluation had 12 participants (12-23). The evaluation took about 1.5 hours per learner on average. In the first evaluation, there were two participants with relatively low levels. They mostly preferred sentences extracted by DPP because sentence difficulty was used as a quality feature. Participant #1 still had problems with understanding sentences because of the low Japanese proficiency, however, participant #9 could understand most of them. Interview results for the evaluation by learners are discussed in

Table 4.7: Learners' votes on the best example lists. Bold numbers are the majority for a person. FC means that the learner has experience of using flashcards. The level is approximate JLPT-style Japanese language proficiency from N5 (lowest) to N1 (highest). + and − near a level means that the participant is higher or lower than the specified level, however, the specified level is the closest one.

#	FC	Level	Rand	DeMelo	DPP
1	*	N4+	3	1	**10**
2	*	N2+	5	3	**6**
3		N2	4	**6**	4
4		N2	5	2	**7**
5		N1	**7**	4	3
6	*	N2−	3	4	**7**
7		N1−	**8**	0	6
8		N1−	4	**7**	3
9	*	N3−	0	1	**13**
10	*	N1−	2	3	**9**
11	*	N1−	3	2	**9**
12	*	N2	0	1	**12**
13		N2	4	**5**	**5**
14	*	N2	3	**6**	5
15		N2	6	1	**7**
16	*	N3	3	2	**9**
17		N1	3	**6**	5
18		N2	**8**	0	6
19		N1	**7**	3	4
20	*	N2	2	2	**10**
21		N3	2	2	**10**
22	*	N3	4	0	**10**
23		N2	0	2	**12**
Total			86	63	173
Percentage			26.7%	19.6%	53.7%

Table 4.8: Learner vote ratios for DPP and the corresponding 95% confidence intervals, computed with bootstrap resampling

Level	Votes, %	Lower Bound	Upper Bound
All	53.7	48.8	58.6
N3	75.0	64.3	85.7
N2	53.2	46.1	60.4
N1	39.8	30.6	49.0

the following subsection. In the second evaluation, we focused on learners with intermediate (N3-N2) levels, for which we had insufficient coverage in the first evaluation.

Vote counts for users and aggregated counts are shown in Table 4.7. DPP gets about half of all votes, which is a good result for the proposed method. It also gets a majority for every participant who had the experience of using flashcards or spaced repetition systems. This gives hope that example sentences are going to be useful inside the flashcards.

Table 4.8 shows 95% confidence intervals for learners' votes over the example sentence lists. Confidence intervals (CI) were computed by bootstrap resampling: we resampled individual learner votes from a categorical distribution using the collected data as a vote distribution for a learner. Then we aggregated the results in the same way as in the main experiment. We used 100,000 samples in the bootstrap resampling and show lower and upper bounds as the computed percentage at 2.5 and 97.5 levels.

Based on the bootstrap resampling, N3 learners' vote for DPP is larger with a statistical significance than the majority of votes (50%) with the lower CI bound of 64.3. Voting for all learners and N2 learners is larger with a statistical significance than a random choice (33%). On the other hand, the voting of N1 learners for DPP is not larger with a statistical significance than a random choice.

4.5.4 Evaluation by a Native Teacher of Japanese

The second part experiment was performed by showing the same example sentence lists to a native Japanese language teacher. In addition to selecting the best list, a teacher was asked to rank from 1 to 5 how appropriate the list was for students of approximately N3 and N2 JLPT levels. Identically to the learners' case, no explicit criteria were given. Unfortunately, because of time limitations, only one teacher has participated in the second part of the evaluation.

For the initial selection, the teacher commented that the best list was selected as if examples were for learners of N3 level. The votes on the initial selection were 0, 4,

Table 4.9: Sense coverage of the extracted example sentences. Gold is the number of senses in the monolingual dictionary. Rand, DeMelo and DPP is the number of senses in the extracted example sentences.

Word	Gold	Rand	DeMelo	DPP
飛ぶ	13	4	6	5
積む	6	4	3	3
走る	12	4	4	3
取る	75	11	11	10
掛ける	48	9	10	5
青い	3	2	2	3
汚い	6	3	3	3
鋭い	5	4	3	3
全然	3	2	2	1
バリバリ	3	2	2	2
さっぱり	5	2	3	1
頭	10	3	3	3
卵	4	2	2	4
足	6	1	2	2

10 for Rand, DeMelo, and DPP respectively. Average ranks for lists were 3.36, 3.79, 4.64 and 3.86, 4.21, 4.36 for N3 and N2 learner level respectively.

Results of the evaluation by the teacher also assign the DPP system as the best for N3 learners both by vote numbers and by average rank. For N2 learners a score for DPP was lower, at the same time the score for DeMelo has raised. The score for Rand was the lowest.

Criteria for selection were the following. Non-target words in a sentence should not be too difficult. A sentence should not depend on the outer context like if it was inside the conversation or about current affairs. The sentences should be short and the usages of the target words should be common. These criteria are strongly aligned with the objectives DPP uses for sentence extraction, which seems to be the reason for its high appraisal by the teacher.

If examples would be selected for N2-like learners, a sentence should include more different structures and usages. However, if usages are too non-usual, in contrary they are more difficult to use, albeit interesting. However, some high-level students had a different point of view.

4.5.5 Evaluating Semantic Diversity

To check the semantic diversity we have performed word sense disambiguation manually, with senses defined by Super Daijirin Japanese Dictionary. Table 4.9

Table 4.10: Comparison of Learners' Feedback for Methods

Methods		
DPP	**DeMelo**	**Rand**
Positive Feedback		
∘ Short ∘ Contain different usages ∘ Simple and easy to understand ∘ No useless words ∘ Possible to guess word meaning from context ∘ Look "good" ∘ Closer to daily life	∘ Contain different usages ∘ Contain different grammar	∘ Contain different usages ∘ Interesting usages ∘ Long and lots of context
Negative Feedback		
∘ Sentences are short and bland ∘ Compound verbs (e.g. 飛び込む) seem not the same as plain verbs (飛ぶ) ∘ Similar sentences	∘ Many sentence fragments ∘ Used difficult grammatical constructions ∘ Sentences look too informal ∘ More difficult to read fast ∘ Similar sentences ∘ Lack of punctuation	∘ Too much katakana-words ∘ Very long sentences ∘ Contain difficult words ∘ Have words that are not useful for examples ∘ More difficult to read fast ∘ Emoticons

shows the number of senses in the extracted sentences and the total number of senses in a word, as defined by the dictionary. Generally, all methods have comparable performance. However, DPP has significantly lower semantic diversity with 掛ける, and could not produce more than one sense for 全然 and さっぱり. On the other hand, it has better semantic diversity for 青い and 卵. To conclude, DPP keeps the diversity comparable to other methods, while producing overall easier to read sentences and cleaner sentences.

4.6 Discussion

All evaluations were performed in an interview manner. Participants were asked to explain their choices about lists and criteria they were using. We show frequently discussed positive and negative feedback in Table 4.10.

Generally, list diversity was regarded as one of the main criteria for the selection. Semantic and lexical diversity was the mainly referred part. However, grammatic diversity was named as well. By grammatic diversity participants usually meant, for example, usage of a verb in different grammatic forms. Other themes that frequently came into the criteria for the selection were sentence difficulty and *how interesting*

were the sentences. Each of the points is discussed in greater detail below.

4.6.1 Similarity and Diversity

Diversity was the main hypothesis behind this work and it was validated by the answers of the participants. Most of them have stated that the non-similarity of sentences in a list was one of the main criteria for the selection.

All three used methods were specialized to produce non-similar sentences. DeMelo explicitly tries to select sentences with frequent words and penalize such words in the next selections. Random selection is going to select different words with the high probability if the ground set contained the diverse sense in approximately equal proportions.

For the DPP features were explicitly crafted to deal with semantic and syntactic similarity in addition to lexical similarity. Based on the results, there were cases where DPP was better in terms of diversity and the cases when it was worse.

One example of good performance in this regard was the word “卵”. In addition to the usual meaning of an egg in a sentence like “それには多くの卵を割る必要があります”, the DPP have also displayed several sentences for the usage like “医師の卵に期待が集まっている” with the meaning of “future profession”. Other methods did not produce example sentences with this sense.

Sentences for the word “頭” show a similar, but slightly mixed result. DPP has selected 6 sentences that have the regular meaning of the word as “head” like “彼女は僕の頭に手をかける”. However, the other 6 had the meaning of the beginning of a time period like in the sentence “今年の頭に撮った写真です”. For nouns, most semantic similarity comes from the relation over の particle and the rest from the relation with the noun’s parent if there is one.

The worst-rated DPP selection was for the word “取る”. It had several almost same sentences, for example, “自分の行動に責任を取れ” and “自分の行動に責任をとる”. General sense diversity was not good as well. There could be several reasons for this.

The first reason is that writings of 取る in example sentences were different, as using kanji – “取れ” in the first sentence against hiragana-only “とる” in the second sentence. Hiragana only writing can have multiple ambiguous kanji writings – at least (取る, 執る, 捕る, 採る, 摂る, 撮る, 盗る). A tool could correctly disambiguate them and select the correct one – 取る, but it could not do it in this case and simply produced a list of all candidates. Because of the results of predicate-argument structure analysis, which are used for building semantic similarity vectors, contained all the possible variants and the semantic representation vectors become non-similar for these two similar sentences.

In the case of 掛ける, while there is a lack of sense diversity, which is mostly defined by arguments in を case, DPP selects sentences with differing arguments in other slots, e.g. 遊んでいる時に声をかけてみました。 (time case present with 時 as argument) and 何かあったら声をかけてね。 (no additional cases).

The second reason probably lies in model parameters. There is no training data at the moment to tune model parameters to produce the best example sentences, the greedy selection algorithm with DPP ranking requires tuning in terms of similarity features, quality features, and making every item more or less similar. Applications of DPP described in the Kulesza et al (Kulesza and Taskar 2012) used training of the quality features and tuning of similarity parameter r, although in this application there was no training data to do so.

The first reason is that writings of 取る in example sentences were different, as using kanji – "取れ" in the first sentence against hiragana-only "とる" in the second sentence. Hiragana only writing can have multiple ambiguous kanji writings – at least (取る, 執る, 捕る, 採る, 摂る, 撮る, 盗る). A tool could correctly disambiguate them and select the correct one – 取る, but it could not do it in this case and simply produced a list of all candidates. Because of the results of predicate-argument structure analysis, which are used for building semantic similarity vectors, contained all the possible variants and the semantic representation vectors become non-similar for these two similar sentences.

The second reason probably lies in model parameters. There is no training data at the moment to tune model parameters to produce the best example sentences, but as it was shown in Subsection 4.3.4, greedy selection with DPP ranking algorithm requires tuning in terms of similarity features, quality features, and making every item more or less similar. Applications of DPP described in the Kulesza and Taskar (2012) used training of the quality features and tuning of similarity parameter r, although in this application there was no training data to do so.

The last probable reason lies in the DPP method itself. It is possible not only to select from it greedily, as we do in this thesis but also to sample from DPP as a distribution, where more diverse subsets are assigned a higher probability. If a size of an individual sample is n, then it is shown (Kulesza and Taskar 2012) that an *expected size* of a sample $E[n]$ from a DPP is

$$E[n] = \sum_{i=1}^{N} \frac{\lambda_i}{\lambda_i + 1},$$

where λ_i are the eigenvalues of C or L matrices. Expected sample sizes for the data used in our experiment are shown in Table 4.11. Most of the expected sample sizes are very large, much larger than the 12 sentences that were extracted. Large

Table 4.11: Expected DPP sample sizes

Word	飛ぶ	積む	走る	取る	掛ける	青い	汚い
E[n]	58.27	67.67	74.53	69.78	75.24	74.32	77.60
Word	鋭い	全然	バリバリ	サッパリ	頭	卵	足
E[n]	62.31	110.51	32.80	101.41	110.28	77.59	113.40

expected sample sizes could mean that one needs to select a large number of items to get a good diverse subset. It seems that similarity features are the reason for high expected sample sizes. Lexical features could give large variation to the shape of vectors and form many distinct eigenvectors. Saying that, tuning parameters of similarity feature mixing may help this problem as well.

As a side note, examples for ばりばり, which had the lowest expected sample size, were good and it was a clear win of the DPP algorithm. The votes were 2, 1, 8 for Random, DeMelo, and DPP respectively.

4.6.2 Difficulty

Sentence difficulty is also one of the main criteria learners have used for the list selection. The initial assumption for the creation of the system is that example sentences should be easy to understand and as short as possible. One reason for this is because if example sentences are shown as a question on each flashcard, a learner has to read many sentences and overly long sentences create too much cognitive load.

Some learners agree with the initial vision on example sentence difficulty: "an example sentence should not contain words harder than its target". However, there was another point of view as well. If learners thought that the sentences were *for the reference*, like those shown with the definition of a word in a dictionary, then they selected the sentences which were readable, but not *too easy* compared to others. Examples by DPP were *too easy* for those learners. Mostly the people who did not have experience of using flashcards selected sentences in this manner.

There was another small group of learners who wanted to see really difficult sentences. Probably, the difficulty of example sentences should be customizable and one-size-fits-all type of solution is not going to work.

One more "low hanging fruit" that was not done for the sentence difficulty is using kanji for estimation. Learners of lower levels from countries that do not use Kanji simply could not read sentences containing kanji unknown to them. This point should be included in the improved version of the difficulty estimation.

Hypotheses

Optimal for a learner relative difficulty of an example sentence for a target word is a function having at least the following three parameters:

- example usage mode,
- level of a learner,
- learner's familiarity with a target word.

The difficulty of kanji, sentence non-target words, and grammar are included in the function as well, however, we would like to discuss the parameters mentioned in the bullet list in more detail.

Example usage mode is whether example sentences are used for reference or a review. **Reference** usage occurs in a dictionary-like setting where a list of example sentences is presented for a learner to compare between word usage in different senses or situations. In this setting, a learner has other sentences to serve as a sort of "anchor" to focus on a target word. Because of this, and a need to provide a way to compare sentences from each other, example sentences could have higher difficulty when they make up a *reference list*. In contrast to that, if an example sentence is shown as a *flashcard question*, there is no such "anchor" to compare a single sentence to others. **Review** sentences should be slightly easier to understand because they have fewer hints for a learner in general.

The level of a learner is another major factor in whether an example sentence is going to be useful or not. Learners of higher levels are going to understand more difficult sentences and easy ones become boring to them. However beginners and intermediate level learners can find it difficult (and sometimes even impossible without an additional explanation) to understand sentences that advanced learners find interesting.

The last point is the learner's familiarity with the example sentence target word. It strongly relates to the difficulty of a non-target word in a sentence. If a learner is not familiar with the target word, then other words serve mostly as an explanation for target's *meaning* and the sentence itself should be easier. If a learner is generally familiar with the word, that context given by an example sentence helps the learner to learn and remember *usage situations* of the target. Sentences in this period of familiarity could be a bit harder than the average. However when a learner is completely familiar with the target word, then even usage situations can be inferred by a simple collocation. Collocation is going to help with disambiguating word senses and nothing else.

It seems that we should talk not about good example sentences *in general*, but good example sentences *for a learner at some point in a learning process*. Static example lists are not going to solve this problem efficiently, but an educational tool like SRS can. It has access not only to the learner's general knowledge level but to the learning process data for individual words as well. Using it, an example extraction system can provide the best examples learner needs at that point in time.

4.6.3 Sentence Content

Another criterion that was used by learners for selecting sentences was if the sentences were *interesting*. There were 3 main types of such sentences:

- Sentence has a story.
  "画像が汚いのは、携帯カメラで撮ったからです、今度綺麗な写真でも撮っておきましょう" vs "画像が汚かったりしたら買う気しませんからね"

- Sentence displaying a vivid image.
  "旧ソ連の宇宙飛行士ガガーリンの有人宇宙飛行「地球は青かった」"

- Sentence is funny or unusual for a participant.
  "「中天」とは死者が修行を積むような場所"

At the beginning of the evaluation, there were cases where a single *interesting* sentence gave a reason for a participant to select a list of sentences even if it contained sentences of generally lower quality like complete fragments. After that case participants were additionally instructed to try judge lists on a whole and not focus only on a single sentence.

Still, such content usually occur only in more or less lengthy sentences containing many different words. DPP was heavily biased against such sentences.

At the same time, lengthy sentence content does not usually interact with the target words directly. Thus such cases should be treated specially.

At the present, it is very hard to automatically judge if a sentence is going to be funny, unusual or generally interesting for a learner. However, it should be possible to automatically get sentences that has some kind of story. A simple story is two events with a cause-effect relationship. By exploiting information about events and event relations developed by Shibata (Shibata, Kohama, and Kurohashi 2014) it should be possible to collect sentences with frequent events. These story-like sentences could be more helpful for remembering the connotations of a word and its usage.

4.6.4 Errors in Pipeline

Recall that example sentence candidates were preselected by a search engine that tries to find good patterns of target word usage, while keeping a sentence length not very long. This kind of selection has helped to sidestep most problems related to errors in automatic syntactic analysis (because the sentences were generally short) and predicate-argument structure analysis (because the sentences were selected to have patterns useful for that analysis, refer to Section 4.2.4 for details). However, there were several classes of problems with external tooling that has decreased performance of example extraction system in general.

One of those problem classes was discussed in the Section 4.6.1. Ambiguity in a canonic representation makes makes it difficult to compute good semantic similarity vectors for such words. Most of the other problems manifested in invalid search results providing bad example sentences to the pre-selection lists.

These problems were mostly caused by the mistakes of the morphological analyzer. Sometimes, because of that a completely different word could appear in search instead of a target. For example a sentence 相手が居なけりゃ自分でやりゃあええ。 was found when searching for the word "青い". It seems that morphological analyzers have performed incorrect segmentation (やりゃ｜あええ) instead of (やりゃあ｜ええ) and because of that 青い[9] have "appeared" in the sentence. Incorrect words in sentences because of segmentation errors are a bad problem because such sentences do not contain the target word and are completely useless as examples.

Another problem related to analysis arises from character sequences that were unexpected for the analyzer like typos or dialects. For example, a sentence "そんなことを言うとりました" contains a dialect usage and it is a low priority to have robust dialect support for the analyzers. Analyzer thinks that "とりました" is a verb 取る and causes a sentence with an incorrect word to appear in the search result.

4.6.5 Support of Other Languages

An example extraction system is developed for Japanese, however underlying methods have very few Japanese-specific parts. The system itself is unsupervised and has only a tokenizer, morphologic analyzer, and dependency parser as software dependencies. All other data can be created from a raw corpus analyzed by these three tools.

For example, English could be supported by using a dependency parser that performs labeled dependency analysis like a Stanford Core NLP parser (D. Chen and C. Manning 2014). Parsing a corpus will produce information sufficient to build

[9] あええ can be a phonetic change from あおい in male speech

a database of relations with prototypes, needed to create semantic representation. Creating word embeddings does not require even that information.

5 Conclusion

This thesis discusses the improvements for Japanese language learning with natural language processing technology. Japanese language, because of the nature of its script, requires programs to segment the input text into words before the further process. Chapters 2 and 3 of the thesis discuss the improvments in Japanese morphological analysis. Chapter 4 discusses an approach for automated high-quality example sentence extraction for Japanese language.

5.1 Morphological Analysis

We give an overview of history and types of approaches for morphological analysis. We reviewed the two main styles of morphological analysis, comparing the pluses and minuses of pointwise and search-based approaches.

5.1.1 Improvements of Juman++ Morphological Analyzer

We achieved a 250x improvement in analysis speed compared to the original version of the Juman++ morphological analyzer. Because of our improvements, it became a practical morphological analysis of Japanese with extremely high analysis accuracy. Improvements that made it possible can be classified into algorithmic improvements and microarchitectural optimizations. The main algorithmic improvements are the following:

- Organizing binary dictionary as a column-based database and using the dictionary organization to compute hash-based features without accessing string-based features, described in Subsection 2.3.2.

- Modifying the beam search procedure to greatly decrease the number of considered paths spanning character boundaries by heuristically weighting lattice nodes, described in Subsection 2.3.6.

- Moving dictionary-related computations to the dictionary build step

In addition to algorithmic improvements, Juman++ contains multiple enhancements, which also improved its analysis speed. The most notable improvements are:

- Generating static linear model evaluation code from model definition, described in Subsection 2.3.4.
- Prefetching linear model weights by delaying computations of the linear model score, described in Subsection 2.3.4.
- Performing necessary computations of the linear model score only once, described in Subsection 2.3.4.
- Vectorizing and batching the computations of the recurrent neural language model, described in Subsection 2.3.7.

Both types of improvements also enabled Juman++ to improve the analysis accuracy as well by enriching the model representative power and making it possible to train the linear model better. Such a high-accuracy practical morphological analyzer in our opinion is a very useful tool for most applications of natural language processing.

5.1.2 Fully-Neural Low-Footprint Morphological Analyzer

The high accuracy of search-based morphological analyzers was realized because of human-curated high-quality morphological dictionaries. It was, however, difficult to combine the dictionary information with pointwise morphological analyzers. We proposed a fully-neural model for morphological analysis and a training regime that can incorporate rich dictionary information into the pointwise model. The training procedure uses a bootstrap analyzer to produce large quantities of automatically-labeled training data. The resulting analyzer can have the same analysis accuracy as the bootstrap analyzer while having a much smaller model (20x smaller in our experiments). Our additional experiments showed that it is possible to shrink the model even further, while sacrificing analysis accuracy, opening doors to high-accuracy sub-megabyte models for morphological analysis. Such models can be easily deployed in mobile or serverless applications which are sensitive to the model size. Sub-megabyte models can even be usable from a browser, opening ways to new applications.

5.1.3 Future Work

One unexplored direction for further improving the analysis speed is the MeCab-like precomputing of feature weights. While it would not be feasible to fully precompute all features, it would be interesting to perform partial precomputation of features. The implementation, however, would be significantly more complex than it is now, so that would be a trade-off for computation speed versus complexity and maintainability.

Another important direction would be to develop a morphological analyzer, robust to errors in the input text. Current analyzers expect that the input text is correct, which is true for professionally edited newspaper texts but can contain errors, typos, or IME misconversions. Such an analyzer would be very useful for analyzing user-generated text from social media.

One unresolved problem of Jumandic-based analyzers is the low quality of reading estimation of analysis results. This is also an important direction for the improvements.

Finally, it would be interesting to find a way to improve the generalization of neural networks with limited training data. We have shown that the data-rich approach works, but efficient usage of rich dictionary information while training only on human-annotated data remains an open research question.

5.2 Example Extraction

We proposed an automated example sentence extraction system for language learning using automated tools like spaced repetition systems. The proposed example extraction system focuses on extracting diverse and good sentences and consists of a syntactically aware distributed search system and determinantal point process-based extraction part. We evaluated the system on the Japanese language with learners of different levels and a native teacher of Japanese as a second language. In both settings, the participants preferred our approach to baselines. The proposed method was especially preferred by intermediate (N3) learners, followed by upper-intermediate (N2), with the results being statistically significant in both cases. Advanced level learners (N1) found sentences produced by our method to be short and bland, but still slightly preferred our method to other methods. Our approach can be used to create a learning experience fully personalized for students' needs and knowledge.

5.2.1 Future Work

There are several areas for improvement in the proposed approach. One area is providing interesting example sentences. Learner evaluators have expressed a preference for interesting example sentences. Having an automated approach for selecting interesting sentences can provide a way to make language learning more engaging activity, and increase the motivation of learners.

Another important direction would be to improve the method of computing the similarity vector representation for example sentences. While the current approach was shown to work, an approach based on contextualized language models can greatly improve the quality of vector representations, and, hopefully, the extracted example sentences.

Finally, the experiments have shown that sentence difficulty is very important for learners. Inferring sentence difficulty from lexical data was shown to be useful. However, an approach that takes into consideration words, grammar, and context should be more precise and produce reliable results.

Bibliography

References

Asahara, Masayuki and Yuji Matsumoto (2000). *Extended Hidden Markov Model for Japanese Morphological Analyzer*. Tech. rep. 54. The Special Interest Group Notes of IPSJ, pp. 1–8.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (July 2016). "Layer Normalization". en. In: *arXiv:1607.06450 [cs]*. URL: https://arxiv.org/abs/1607.06450 (visited on 12/05/2018).

Brants, Thorsten, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean (June 2007). "Large Language Models in Machine Translation". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 858–867. URL: http://www.aclweb.org/anthology/D/D07/D07-1090 (visited on 12/04/2018).

Breen, Jim (2004). "Proceedings of the Workshop on Multilingual Linguistic Resources". In: chap. JMdict: a Japanese-Multilingual Dictionary, pp. 65–72. URL: http://aclweb.org/anthology/W04-2209.

Cai, Deng and Hai Zhao (2016). "Neural Word Segmentation Learning for Chinese". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 409–420. DOI: 10.18653/v1/P16-1039. URL: http://aclweb.org/anthology/P16-1039.

Cai, Deng, Hai Zhao, Zhisong Zhang, Yuan Xin, Yongjian Wu, and Feiyue Huang (July 2017). "Fast and Accurate Neural Word Segmentation for Chinese". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 608–615. URL: http://aclweb.org/anthology/P17-2096 (visited on 12/01/2018).

Chen, Danqi and Christopher Manning (Oct. 2014). "A Fast and Accurate Dependency Parser using Neural Networks". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Associ-

ation for Computational Linguistics, pp. 740–750. DOI: 10.3115/v1/D14-1082. URL: https://www.aclweb.org/anthology/D14-1082.

Chen, Hongshen, Yue Zhang, and Qun Liu (2016). "Neural Network for Heterogeneous Annotations". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 731–741. DOI: 10.18653/v1/D16-1070. URL: http://aclweb.org/anthology/D16-1070 (visited on 12/02/2018).

Cumming, Geoff and Sue Finch (Feb. 2005). "Inference by Eye: Confidence Intervals and How to Read Pictures of Data". In: *The American psychologist* 60, pp. 170–80. DOI: 10.1037/0003-066X.60.2.170.

de Melo, Gerard and Gerhard Weikum (2009). "Extracting sense-disambiguated example sentences from parallel corpora". In: *Proceedings of the 1st Workshop on Definition Extraction*. WDE '09. Stroudsburg, PA, USA: Association for Computational Linguistics, pp. 40–46. ISBN: 978-954-452-013-7. URL: http://dl.acm.org/citation.cfm?id=1859765.1859772 (visited on 10/21/2013).

Den, Yasuharu, Junpei Nakamura, Toshinobu Ogiso, and Hideki Ogura (May 2008). "A Proper Approach to Japanese Morphological Analysis: Dictionary, Model, and Evaluation". In: *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*. Marrakech, Morocco: European Language Resources Association (ELRA). URL: http://www.lrec-conf.org/proceedings/lrec2008/pdf/258_paper.pdf.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (June 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

Guennebaud, Gaël, Benoît Jacob, et al. (2010). *Eigen v3*. URL: http://eigen.tuxfamily.org.

Hangyo, Masatsugu, Daisuke Kawahara, and Sadao Kurohashi (2012). "Building a Diverse Document Leads Corpus Annotated with Semantic Relations". In: *Proceedings of the 26th Pacific Asia Conference on Language, Information, and Computation*. Bali, Indonesia: Faculty of Computer Science, Universitas Indonesia, pp. 535–544. URL: http://aclweb.org/anthology/Y12-1058.

Hatori, Jun, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii (July 2012). "Incremental Joint Approach to Word Segmentation, POS Tagging, and Dependency Parsing in Chinese". In: *Proceedings of the 50th Annual Meeting of the Association for*

Computational Linguistics (Volume 1: Long Papers). Jeju Island, Korea: Association for Computational Linguistics, pp. 1045–1053. URL: http://www.aclweb.org/anthology/P12-1110 (visited on 12/02/2018).

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1997). "Long Short-Term Memory". In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735 (visited on 12/11/2018).

Inoue, Nagayuki and Ichirou Akano (2007). *The Wisdom English-Japanese Dictionary (in Japanese)*. Senseido.

Jakubíček, Miloš, Adam Kilgarriff, Diana McCarthy, and Pavel Rychlý (2010). "Fast Syntactic Searching in Very Large Corpora for Many Languages". In: *Proceedings of the 24th PACLIC*. Workshop on Advanced Corpus Solutions. Tohoku University, Sendai, Japan, pp. 741–747.

Jiang, Wenbin, Haitao Mi, and Qun Liu (Aug. 2008). "Word Lattice Reranking for Chinese Word Segmentation and Part-of-Speech Tagging". In: *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*. Manchester, UK: Coling 2008 Organizing Committee, pp. 385–392. URL: https://www.aclweb.org/anthology/C08-1049.

Johnson, William B and Joram Lindenstrauss (1984). "Extensions of Lipschitz mappings into a Hilbert space". In: *Contemporary mathematics* 26.189-206, p. 1.

Kaji, Nobuhiro and Masaru Kitsuregawa (Oct. 2013). "Efficient Word Lattice Generation for Joint Word Segmentation and POS Tagging in Japanese". In: *Proceedings of the Sixth International Joint Conference on Natural Language Processing*. Nagoya, Japan: Asian Federation of Natural Language Processing, pp. 153–161. URL: http://www.aclweb.org/anthology/I13-1018 (visited on 12/02/2018).

Kathuria, Pulkit and Kiyoaki Shirai (Jan. 2012). "Word Sense Disambiguation Based on Example Sentences in Dictionary and Automatically Acquired from Parallel Corpus". In: *Advances in Natural Language Processing*. Lecture Notes in Computer Science 7614, pp. 210–221. (Visited on 10/21/2013).

Kawahara, Daisuke, Yuta Hayashibe, Hajime Morita, and Sadao Kurohashi (Sept. 2017). "Automatically Acquired Lexical Knowledge Improves Japanese Joint Morphological and Dependency Analysis". In: *Proceedings of the 15th International Conference on Parsing Technologies*. Pisa, Italy: Association for Computational Linguistics, pp. 1–10. URL: https://www.aclweb.org/anthology/W17-6301.

Kawahara, Daisuke and Sadao Kurohashi (May 2006). "Case Frame Compilation from the Web using High-Performance Computing". In: *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC'06)*. URL: http://aclweb.org/anthology/L06-1203.

Kilgarriff, Adam, Miloš Husák, Katy McAdam, Michael Rundell, and Pavel Rychlý (July 2008). "GDEX: Automatically Finding Good Dictionary Examples in a Corpus". In: *Proceedings of the 13th EURALEX International Congress*. Ed. by Janet DeCesaris Elisenda Bernal. Barcelona, Spain: Institut Universitari de Linguistica Aplicada, Universitat Pompeu Fabra, pp. 425–432. isbn: 978-84-96742-67-3.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs.LG]*. arXiv: 1412.6980. url: https://arxiv.org/abs/1412.6980.

Kitagawa, Yoshiaki and Mamoru Komachi (Dec. 2018). "Long Short-Term Memory for Japanese Word Segmentation". In: *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*. Hong Kong: Association for Computational Linguistics. url: https://www.aclweb.org/anthology/Y18-1033.

Klambauer, Günter, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter (2017). "Self-Normalizing Neural Networks". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 971–980. url: http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf.

Kruengkrai, Canasai, Kiyotaka Uchimoto, Jun'ichi Kazama, Yiou Wang, Kentaro Torisawa, and Hitoshi Isahara (Aug. 2009). "An Error-Driven Word-Character Hybrid Model for Joint Chinese Word Segmentation and POS Tagging". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, pp. 513–521. url: https://www.aclweb.org/anthology/P09-1058.

Kudo, Taku (2018). *Theory and Implementation of Morphological Analysis (In Japanese)*. Jissen NLP Series. Kindaikagakusha.

Kudo, Taku and John Richardson (Nov. 2018). "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, pp. 66–71. doi: 10.18653/v1/D18-2012. url: https://www.aclweb.org/anthology/D18-2012.

Kudo, Taku, Kaoru Yamamoto, and Yuji Matsumoto (July 2004). "Applying Conditional Random Fields to Japanese Morphological Analysis". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona,

Spain: Association for Computational Linguistics, pp. 230–237. URL: https://www.aclweb.org/anthology/W04-3230.

Kulesza, Alex and Ben Taskar (2012). "Determinantal Point Processes for Machine Learning". In: *Foundations and Trends® in Machine Learning* 5.2–3, pp. 123–286. ISSN: 1935-8237. DOI: 10.1561/2200000044. URL: http://dx.doi.org/10.1561/2200000044.

Kurita, Shuhei, Daisuke Kawahara, and Sadao Kurohashi (2017). "Neural Joint Model for Transition-based Chinese Syntactic Analysis". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 1204–1214. DOI: 10.18653/v1/P17-1111. URL: http://aclweb.org/anthology/P17-1111 (visited on 12/01/2018).

Kurohashi, Sadao (1994). "Improvements of Japanese morphological analyzer JUMAN". In: *Proceedings of The International Workshop on Sharable Natural Language, 1994*.

Kurohashi, Sadao and Daisuke Kawahara (2012). *Japanese Morphological Analysis System JUMAN 7.0 Users Manual*. Kyoto University.

Kurohashi, Sadao and Makoto Nagao (2003). "Building A Japanese Parsed Corpus". en. In: *Treebanks: Building and Using Parsed Corpora*. Text, Speech and Language Technology. Dordrecht: Springer Netherlands, pp. 249–260. ISBN: 978-94-010-0201-1. DOI: 10.1007/978-94-010-0201-1_14. URL: https://doi.org/10.1007/978-94-010-0201-1_14 (visited on 12/10/2018).

Li, Zhenghua, Jiayuan Chao, Min Zhang, and Wenliang Chen (July 2015). "Coupled Sequence Labeling on Heterogeneous Annotations: POS Tagging as a Case Study". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1783–1792. URL: http://www.aclweb.org/anthology/P15-1172 (visited on 12/04/2018).

Liu, Junxin, Fangzhao Wu, Chuhan Wu, Yongfeng Huang, and Xing Xie (2018). "Neural Chinese Word Segmentation with Dictionary Knowledge". en. In: *Natural Language Processing and Chinese Computing*. Ed. by Min Zhang, Vincent Ng, Dongyan Zhao, Sujian Li, and Hongying Zan. Lecture Notes in Computer Science. Springer International Publishing, pp. 80–91. ISBN: 978-3-319-99501-4.

Ma, Ji, Kuzman Ganchev, and David Weiss (2018). "State-of-the-art Chinese Word Segmentation with Bi-LSTMs". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Com-

putational Linguistics, pp. 4902–4908. URL: http://aclweb.org/anthology/D18-1529 (visited on 12/01/2018).

Ma, Jianqiang and Erhard Hinrichs (July 2015). "Accurate Linear-Time Chinese Word Segmentation via Embedding Matching". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1733–1743. URL: http://www.aclweb.org/anthology/P15-1167 (visited on 12/04/2018).

Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press. 680 pp. ISBN: 0262133601.

Matsumoto, Yuji, Kazuma Takaoka, and Masayuki Asahara (2008). *Morphological Analysis System ChaSen Users Manual. Version 2.4.3.* Nara Institute of Science and Technology.

Mikolov, Tomas (2012). "Statistical Language Models Based on Neural Networks". Ph. D. Thesis. Brno: Brno University of Technology. URL: http://www.fit.vutbr.cz/~imikolov/rnnlm/thesis.pdf (visited on 12/03/2018).

Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig (2013). "Linguistic regularities in continuous space word representations". In: *Proceedings of NAACL-HLT*, pp. 746–751.

Mochihashi, Daichi, Takeshi Yamada, and Naonori Ueda (Aug. 2009). "Bayesian Unsupervised Word Segmentation with Nested Pitman-Yor Language Modeling". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, pp. 100–108. URL: https://www.aclweb.org/anthology/P09-1012.

Morita, Hajime, Daisuke Kawahara, and Sadao Kurohashi (2015). "Morphological Analysis for Unsegmented Languages using Recurrent Neural Network Language Model". en. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 2292–2297. DOI: 10.18653/v1/D15-1276. URL: http://aclweb.org/anthology/D15-1276 (visited on 12/03/2018).

Murawaki, Yugo (2019). "On the Definition of Japanese Word". In: *CoRR* abs/1906.09719. arXiv: 1906.09719. URL: http://arxiv.org/abs/1906.09719.

National Institute for Japanese Language and Linguistics (ed.) (2004). *Bunrui Goi Hyo*. Dainippon Tosho.

Neubig, Graham, Yosuke Nakata, and Shinsuke Mori (2011). "Pointwise Prediction for Robust, Adaptable Japanese Morphological Analysis". In: *Proceedings*

of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. Portland, Oregon, USA: Association for Computational Linguistics, pp. 529–533. URL: http://aclweb.org/anthology/P11-2093 (visited on 12/01/2018).

O'Neill, Melissa E. (Sept. 2014). *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*. Tech. rep. HMC-CS-2014-0905. Claremont, CA: Harvey Mudd College.

Okumura, Manabu, Kiyoaki Shirai, Kanako Komiya, and Hikaru Yokono (July 2010). "SemEval-2010 Task: Japanese WSD". In: *Proceedings of the 5th International Workshop on Semantic Evaluation*. Uppsala, Sweden: Association for Computational Linguistics, pp. 69–74. URL: https://www.aclweb.org/anthology/S10-1012.

Peng, Fuchun, Fangfang Feng, and Andrew McCallum (2004). "Chinese Segmentation and New Word Detection Using Conditional Random Fields". In: *Proceedings of the 20th International Conference on Computational Linguistics*. COLING '04. Stroudsburg, PA, USA: Association for Computational Linguistics. DOI: 10.3115/1220355.1220436. URL: https://doi.org/10.3115/1220355.1220436 (visited on 12/04/2018).

Qi, Yanjun, Sujatha G. Das, Ronan Collobert, and Jason Weston (2014). "Deep Learning for Character-Based Information Extraction". en. In: *Advances in Information Retrieval*. Ed. by Maarten de Rijke, Tom Kenter, Arjen P. de Vries, ChengXiang Zhai, Franciska de Jong, Kira Radinsky, and Katja Hofmann. Lecture Notes in Computer Science. Springer International Publishing, pp. 668–674. ISBN: 978-3-319-06028-6. DOI: https://doi.org/10.1007/978-3-319-06028-6_74.

Saito, Itsumi, Kugatsu Sadamitsu, Hisako Asano, and Yoshihiro Matsuo (Aug. 2014). "Morphological Analysis for Japanese Noisy Text based on Character-level and Word-level Normalization". In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin, Ireland: Dublin City University and Association for Computational Linguistics, pp. 1773–1782. URL: https://www.aclweb.org/anthology/C14-1167.

Sasano, Ryohei, Sadao Kurohashi, and Manabu Okumura (Oct. 2013). "A Simple Approach to Unknown Word Processing in Japanese Morphological Analysis". In: *Proceedings of the Sixth International Joint Conference on Natural Language Processing*. Nagoya, Japan: Asian Federation of Natural Language Processing, pp. 162–170. URL: https://www.aclweb.org/anthology/I13-1019.

Settles, Burr and Mark Craven (Oct. 2008). "An Analysis of Active Learning Strategies for Sequence Labeling Tasks". In: *EMNLP*. Honolulu, Hawaii: Association

for Computational Linguistics, pp. 1070–1079. URL: http://www.aclweb.org/anthology/D08-1112.

Seung, H. S., M. Opper, and H. Sompolinsky (1992). "Query by Committee". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: ACM, pp. 287–294. ISBN: 0-89791-497-X. DOI: 10.1145/130385.130417. URL: http://doi.acm.org/10.1145/130385.130417.

Shafranovich, Y. (Oct. 2005). *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. RFC 4180. Internet Requests for Comments. URL: http://www.rfc-editor.org/rfc/rfc4180.txt.

Shao, Yan, Christian Hardmeier, Jörg Tiedemann, and Joakim Nivre (2017). "Character-based Joint Segmentation and POS Tagging for Chinese using Bidirectional RNN-CRF". In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Taipei, Taiwan: Asian Federation of Natural Language Processing, pp. 173–183. URL: http://aclweb.org/anthology/I17-1018 (visited on 12/01/2018).

Shervashidze, Nino, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt (Apr. 2009). "Efficient graphlet kernels for large graph comparison". In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Vol. 5. PMLR, pp. 488–495.

Shibata, Tomohide, Shotaro Kohama, and Sadao Kurohashi (May 2014). "A Large Scale Database of Strongly-related Events in Japanese". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), pp. 3283–3288. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/1107_Paper.pdf.

Shinnou, Hiroyuki and Minoru Sasaki (2008). "Division of Example Sentences Based on the Meaning of a Target Word Using Semi-Supervised Clustering". In: *LREC 2008*. URL: http://aclweb.org/anthology/L08-1339.

Shinzato, Keiji, Tomohide Shibata, Daisuke Kawahara, and Sadao Kurohashi (2011). "TSUBAKI: An Open Search Engine Infrastructure for Developing Information Access Methodology". In: *Journal of Information Processing* 52.12, pp. 216–227.

Snoek, Jasper, Hugo Larochelle, and Ryan P Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms". In: *NIPS*, pp. 2951–2959.

Søgaard, Anders (July 2010). "Simple Semi-Supervised Training of Part-Of-Speech Taggers". In: *Proceedings of the ACL 2010 Conference Short Papers*. Uppsala, Sweden: Association for Computational Linguistics, pp. 205–208. URL: http://www.aclweb.org/anthology/P10-2038 (visited on 12/02/2018).

Sun, Weiwei and Jia Xu (July 2011). "Enhancing Chinese Word Segmentation Using Unlabeled Data". In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, pp. 970–979. URL: http://www.aclweb.org/anthology/D11-1090 (visited on 12/02/2018).

Suzuki, Jun and Hideki Isozaki (June 2008). "Semi-Supervised Sequential Labeling and Segmentation Using Giga-Word Scale Unlabeled Data". In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, pp. 665–673. URL: http://www.aclweb.org/anthology/P/P08/P08-1076 (visited on 12/02/2018).

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna (June 2016). "Rethinking the Inception Architecture for Computer Vision". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.html.

Takaoka, Kazuma, Sorami Hisamoto, Noriko Kawahara, Miho Sakamoto, Yoshitaka Uchida, and Yuji Matsumoto (May 2018). "Sudachi: a Japanese Tokenizer for Business". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA). URL: https://www.aclweb.org/anthology/L18-1355.

Takeuchi, Junpei and Junichi Tsujii (2005). "A Search System for Japanese Using Dependencies And Paraphrases (in Japanese)". In: *Proceedings of the Eleventh Annual Meeting of the Association for Natural Language Processing*, pp. 568–571.

Tohno, Fumi, Yoshie Itabashi, and Mary E. Althaus (2001). *Shogokukan Progressive English - Japanese Dictionary, 4rd Edition*. Shogakukan.

Tseng, Huihsin, Pichuan Chang, Galen Andrew, Daniel Jurafsky, and Christopher D Manning (2005). "A Conditional Random Field Word Segmenter for Sighan Bakeoff 2005". In: *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*. Association for Computational Linguistics. URL: http://aclweb.org/anthology/I/I05/I05-3027 (visited on 12/04/2018).

Tsubaki, Masashi, Kevin Duh, Masashi Shimbo, and Yuji Matsumoto (Oct. 2013). "Modeling and Learning Semantic Co-Compositionality through Prototype Projections and Neural Networks". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 130–140. URL: http://www.aclweb.org/anthology/D13-1014.

Uchiumi, Kei, Hiroshi Tsukahara, and Daichi Mochihashi (2015). "Inducing Word and Part-of-Speech with Pitman-Yor Hidden Semi-Markov Models". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pp. 1774–1782. URL: https://www.aclweb.org/anthology/P15-1171/.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Wang, Jialei, Peilin Zhao, and Steven CH Hoi (2016). "Soft Confidence-Weighted Learning". In: *ACM TIST* 8.1, p. 15.

Wang, Yiou, Jun'ichi Kazama, Yoshimasa Tsuruoka, Wenliang Chen, Yujie Zhang, and Kentaro Torisawa (Nov. 2011). "Improving Chinese Word Segmentation and POS Tagging with Semi-supervised Methods Using Large Auto-Analyzed Data". In: *Proceedings of 5th International Joint Conference on Natural Language Processing*. Chiang Mai, Thailand: Asian Federation of Natural Language Processing, pp. 309–317. URL: http://www.aclweb.org/anthology/I11-1035 (visited on 12/02/2018).

Wang, Zhiguo, Chengqing Zong, and Nianwen Xue (Aug. 2013). "A Lattice-based Framework for Joint Chinese Word Segmentation, POS Tagging and Parsing". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 623–627. URL: http://www.aclweb.org/anthology/P13-2110.

Weinberger, Kilian, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg (2009). "Feature hashing for large scale multitask learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, pp. 1113–1120.

Wozniak, Robert H (1999). *Classics in Psychology, 1855-1914: Historical Essays*. Thoemmes press.

Xue, Nianwen (2003). "Chinese Word Segmentation as Character Tagging". In: *International Journal of Computational Linguistics & Chinese Language Processing, Volume 8, Number 1, February 2003: Special Issue on Word Formation and Chinese Language Processing*, pp. 29–48. URL: http://aclweb.org/anthology/O03-4002.

Yang, Jie, Yue Zhang, and Fei Dong (July 2017). "Neural Word Segmentation with Rich Pretraining". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 839–849. URL: http://aclweb.org/anthology/P17-1078 (visited on 12/01/2018).

Zhang, Longkai, Houfeng Wang, Xu Sun, and Mairgup Mansur (Oct. 2013). "Exploring Representations from Unlabeled Data with Co-training for Chinese Word Segmentation". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 311–321. URL: http://www.aclweb.org/anthology/D13-1031.

Zhang, Meishan, Nan Yu, and Guohong Fu (Sept. 2018). "A Simple and Effective Neural Model for Joint Word Segmentation and POS Tagging". en. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.9, pp. 1528–1538. ISSN: 2329-9290, 2329-9304. DOI: 10.1109/TASLP.2018.2830117. URL: https://ieeexplore.ieee.org/document/8351918/ (visited on 12/01/2018).

Zhang, Meishan, Yue Zhang, and Guohong Fu (2016). "Transition-Based Neural Word Segmentation". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 421–431. DOI: 10.18653/v1/P16-1040. URL: http://aclweb.org/anthology/P16-1040 (visited on 12/01/2018).

Zhang, Qi, Xiaoyu Liu, and Jinlan Fu (2018). "Neural Networks Incorporating Dictionaries for Chinese Word Segmentation". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 5682–5689. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16368.

Zhang, Yue and Stephen Clark (June 2008). "Joint Word Segmentation and POS Tagging Using a Single Perceptron". In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, pp. 888–896. URL: https://www.aclweb.org/anthology/P08-1101.

Zhang, Yue and Stephen Clark (Oct. 2010). "A Fast Decoder for Joint Word Segmentation and POS-Tagging Using a Single Discriminative Model". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, pp. 843–852. URL: https://www.aclweb.org/anthology/D10-1082.

Zhao, Hai, Chang-Ning Huang, Mu Li, and Bao-Liang Lu (2006). "Effective Tag Set Selection in Chinese Word Segmentation via Conditional Random Field

Modeling". In: *Proceedings of the 20th Pacific Asia Conference on Language, Information and Computation*. Association for Computational Linguistics. URL: http://aclweb.org/anthology/Y/Y06/Y06-1012 (visited on 12/04/2018).

Zhao, Hai and Chunyu Kit (2008). "Exploiting unlabeled text with different unsupervised segmentation criteria for Chinese word segmentation". In: *Research in Computing Science* 33, pp. 93–104.

Zheng, Xiaoqing, Hanyang Chen, and Tianyu Xu (Oct. 2013a). "Deep Learning for Chinese Word Segmentation and POS Tagging". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 647–657. URL: https://www.aclweb.org/anthology/D13-1061.

Zheng, Xiaoqing, Hanyang Chen, and Tianyu Xu (Oct. 2013b). "Deep Learning for Chinese Word Segmentation and POS Tagging". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 647–657. URL: http://www.aclweb.org/anthology/D13-1061 (visited on 12/04/2018).

Zhikov, Valentin, Hiroya Takamura, and Manabu Okumura (Oct. 2010). "An Efficient Algorithm for Unsupervised Word Segmentation with Branching Entropy and MDL". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, pp. 832–842. URL: https://www.aclweb.org/anthology/D10-1081.

Zhou, Hao, Zhenting Yu, Yue Zhang, Shujian Huang, XIN-YU DAI, and Jiajun Chen (Sept. 2017). "Word-Context Character Embeddings for Chinese Word Segmentation". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 760–766. URL: https://www.aclweb.org/anthology/D17-1079 (visited on 12/04/2018).

Zhou, Zhi-Hua and Ming Li (Nov. 2005). "Tri-training: exploiting unlabeled data using three classifiers". In: *IEEE Transactions on Knowledge and Data Engineering* 17.11, pp. 1529–1541. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.186.

List of Major Publications

Tolmachev, Arseny, Daisuke Kawahara, and Sadao Kurohashi (Nov. 2018). "Juman++: A Morphological Analysis Toolkit for Scriptio Continua". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, pp. 54–59. URL: https://www.aclweb.org/anthology/D18-2010.

Tolmachev, Arseny, Daisuke Kawahara, and Sadao Kurohashi (July 2019). "Shrinking Japanese Morphological Analyzers With Neural Networks and Semi-supervised Learning". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 2744–2755. URL: https://www.aclweb.org/anthology/N19-1281.

Tolmachev, Arseny, Daisuke Kawahara, and Sadao Kurohashi (Mar. 2020). "Design and Structure of The Juman++ Morphological Analyzer". In: *Journal of Natural Language Processing* 27.1.

Tolmachev, Arseny and Sadao Kurohashi (Sept. 2017a). "Automatic Extraction of High-Quality Example Sentences for Word Learning Using a Determinantal Point Process". In: *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 133–142. DOI: 10.18653/v1/W17-5014. URL: https://www.aclweb.org/anthology/W17-5014.

Tolmachev, Arseny, Sadao Kurohashi, and Daisuke Kawahara (Apr. 2022). "Automatic Japanese Example Extraction for Flashcard-based Foreign Language Learning". In: *Journal of Information Processing (To Appear)*.

List of Other Publications

Tolmachev, Arseny and Sadao Kurohashi (Mar. 2017b). "Kotonoha: An Example Sentence Based Spaced Repetition System". In: *Proceedings of the Twenty-third Annual Meeting of the Association for Natural Language Processing*. Tsukuba, Japan, pp. 847–850. URL: http://www.anlp.jp/proceedings/annual_meeting/2017/pdf_dir/E5-4.pdf.

Tolmachev, Arseny and Sadao Kurohashi (Mar. 2018). "Juman++ v2: A Practical and Modern Morphological Analyzer". In: *Proceedings of 24th Annual Meeting of Japanese Natural Language Processing Society*. Okayama, Japan, pp. 917–920. URL:

http://www.anlp.jp/proceedings/annual_meeting/2018/pdf_dir/A5-2.pdf.

Tolmachev, Arseny, Hajime Morita, and Sadao Kurohashi (Mar. 2016). "A Grammar and Dependency Aware Search System for Japanese Sentences". In: *Proceedings of the Twenty-second Annual Meeting of the Association for Natural Language Processing*. Sendai, Japan, pp. 593–596. URL: http://www.anlp.jp/proceedings/annual_meeting/2016/pdf_dir/P15-4.pdf.