Japanese Word Segmentation using Character-level Recurrent Networks with Dictionary Information

Taishi Ikeda

Abstract

This paper presents a neural word segmentation (NWS) model that uses dictionary information. Recently, many NWS models have been proposed and achieved state-of-the-art results. However, it is not clear how we incorporate the information of a word dictionary into a character-level NWS model. In this work, we propose a method of incorporating features based on a word dictionary and word embeddings directly into the model. The experimental results show that our method is robust to both Japanese balanced corpus and a microblog corpus.[1]

1 Introduction

Word identification is a fundamental step in processing of languages that have no word boundaries such as Japanese and Chinese. Its accuracy has a significant effect on downstream applications such as machine translation and dialogue systems.

In recent years, many neural word segmentation (NWS) models for Chinese have been proposed. For example, Chen et al. (2015) formulated a Chinese WS task as a character-level sequence labeling problem and proposed an NWS model based on long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997). Such neural network–based models have achieved high accuracies without feature engineering (Zheng et al., 2013; Pei et al., 2014; Cai and Zhao, 2016).

In contrast to Chinese WS, conventional methods for Japanese WS use dictionary-based approaches to perform joint WS and part-of-speech (POS) tagging (Kurohashi et al., 1994; Kudo et al., 2004; Kaji and Kitsuregawa, 2014; Morita et al., 2015). A dictionary gives a tractable way to build a lattice from an input sentence. A correct path in the lattice is selected from all candidate paths using machine learning techniques to output segmented words with POS tags. Such conventional methods have improved the accuracy of WS for words that do not appear frequently in the training data with the help of dictionary information (Nakagawa, 2004).

Since a word dictionary improves the accuracy of Japanese WS, it is expected that we can improve the performance of NWS models by incorporating a word dictionary. However, since the character-based neural models perform tagging at character units, it is not straightforward how we can incorporate dictionary information within such character-based NWS models.

In this work, we propose a method to solve this issue. In our method, we introduce two kinds of dictionary features (see Section. 3.2): (i) a binary feature whether a subsequence of the character sequence exists in the dictionary or not, and (ii) word embeddings matched with subsequences. We conducted an experiment to demonstrate that incorporating the dictionary features improves the NWS model, and our method is robust to both Japanese balanced corpus and a microblog corpus.

2 Neural Word Segmentation

In this section, we provide a description of the NWS model proposed by Chen et al. (2015). They formulated a Chinese WS task as a character-level sequence labeling problem and proposed an NWS model based on the LSTM. In this work, we also use their NWS model, and then combine our dictionary features (Section. 3.2).

The aim of WS is to assign a tag sequence $Y = y_1, y_2, \cdots, y_n$ to an input sentence $X = x_1, x_2, \cdots, x_n$, where x_t is the t-th character in a sentence and $y_t \in T$ is the tag of x_t. Note that

[1] Our source code is publicly available at XXX

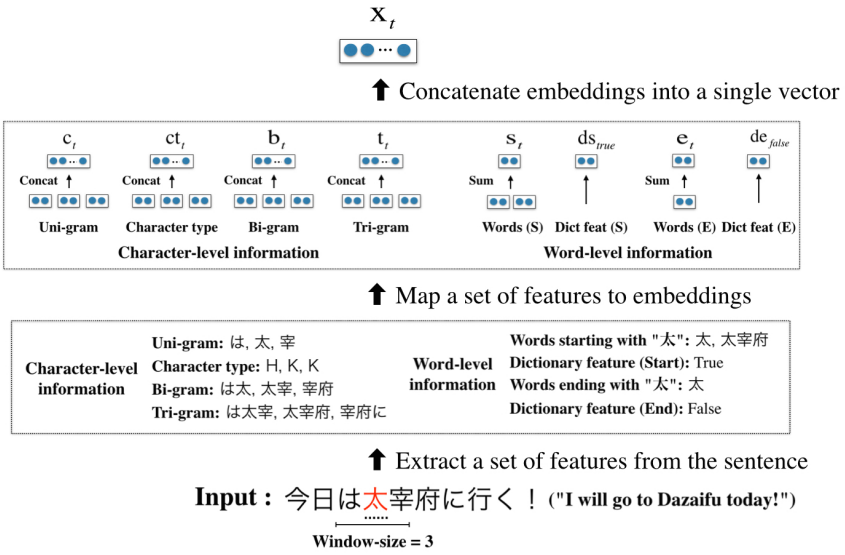

Figure 1: An example of the process of creating a feature vector from the input sentence at time $t = 4$.

$T = \{B, M, E, S\}$ indicates Begin, Middle, and End of a multi-character segmentation, and a Single character segmentation, respectively.

First, we transform X to a sequence of feature vectors $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$, where $\mathbf{x}_t \in \mathbb{R}^d$ is the feature vector of the t-th character. The process of extracting the feature vectors is the main part of our method and, we describe it in detail in Section 3. Next, using the obtained vector sequence $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ as input, the LSTM computes each hidden vector $\mathbf{h}_t \in \mathbb{R}^\ell$.

In the output layer for predicting a tag sequence, we employ conditional random fields (CRF) to model tag dependencies. The vector $\mathbf{p}_t \in \mathbb{R}^{|T|}$ corresponding to each tag is computed as follows:

$$\mathbf{p}_t = \mathbf{W}_{tag}\mathbf{h}_t + \mathbf{b}_{tag},$$

where $\mathbf{W}_{tag} \in \mathbb{R}^{|T|\times\ell}$ is a weight matrix and $\mathbf{b}_{tag} \in \mathbb{R}^{|T|}$ a bias vector. The score of the tag sequence Y for the input sentence X is computed as follows:

$$s(X, Y) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} p_{y_i}, \qquad (1)$$

where $\mathbf{A} \in \mathbb{R}^{|T|\times|T|}$ is a matrix of transition scores between tags. During training, we minimize the negative log-likelihood of the correct tag sequence using stochastic gradient-based optimization techniques in order to obtain the optimal parameters θ as follows:

$$-\ln p(Y|X;\theta) = -s(X, Y) + \ln \sum_{\tilde{y}\in Y_x} e^{s(X,\tilde{y})},$$

where $\tilde{y} \in Y_x$ is all possible tag sequences for X. During decoding, we predict the output sequence to maximize Eq. (1) using dynamic programming.

3 Feature Extraction

In this section, we describe how to create a feature vector at each time step from an input sentence. As an example, Figure 1 shows the process of creating a feature vector for the 4-th character "太".

3.1 Character-level Information

The uni-gram character embedding $\mathbf{c}_t \in \mathbb{R}^u$ is represented in $\mathbf{E}_{uni} \in \mathbb{R}^{|Uni|\times u}$, where $|Uni|$ is the number of uni-grams in the training data and u is the dimension of the uni-gram embedding, and then we concatenate the embeddings based on the window-size k. For example, when $k = 3$, the concatenated embedding is $\mathbf{c}_{t-1} \oplus \mathbf{c}_t \oplus \mathbf{c}_{t+1}$, where $\oplus$ is the operation of vector concatenation.

As the same way, we represent bi-gram, tri-gram, and character-type embeddings in the corresponding lookup tables $\mathbf{E}_{bi} \in \mathbb{R}^{|Bi|\times b}$, $\mathbf{E}_{tri} \in \mathbb{R}^{|Tri|\times tr}$, and $\mathbf{E}_{ct} \in \mathbb{R}^{|CT|\times ct}$, and concatenate them based on k.

3.2 Word-level Information

In an example of the sentence "今日は太宰府に行く！" (meaning "I will go to Dazaifu today!") shown in Figure 1, the words starting with the character "太" in the dictionary are "太" and "太宰府". We extract word embeddings corresponding to "太" and "太宰府" from the word embed-

ding matrix $\mathbf{E}_s \in \mathbb{R}^{|V| \times s}$, where $|V|$ is the number of vocabulary in a word dictionary and s is the dimension of the word embedding. Then we create a vector $\mathbf{s}_t \in \mathbb{R}^s$ by summing the extracted word embeddings. This embedding $\mathbf{s}_t$ contains the information of the words starting with "太". While this feature is proposed by Neubig et al. (2011) and they use it as a discrete feature, we use it as a s-dimensional real-valued word embedding. Moreover, these word embeddings can be pre-trained from a large-scale unlabeled data (Mikolov et al., 2013).

In the same way, we create a vector $\mathbf{e}_t$ by looking up words ending with the character "太" from the word embedding matrix $\mathbf{E}_e \in \mathbb{R}^{|V| \times e}$.

Here, we introduce a new dictionary feature, which indicates whether there are any words in the dictionary matching with the character sequence from time t. Given $s(c_t)$, the number of words in the dictionary starting with the character c_t in the context of the sentence X, we introduce the following features:

$$\begin{aligned} \mathbf{ds}_{true} &\in \mathbb{R}^j \quad \text{if } s(c_t) > 1, \\ \mathbf{ds}_{false} &\in \mathbb{R}^j \quad \text{otherwise.} \end{aligned}$$

Because most characters constitute themselves as a word in the dictionary, we set the threshold occurrence frequency of a character to be greater than 1.

In addition, we introduce another dictionary feature indicating the word existence in the dictionary. Given $e(c_t)$, the number of words in the dictionary ending with the character c_t in the context of the sentence X, we introduce the following features:

$$\begin{aligned} \mathbf{de}_{true} &\in \mathbb{R}^j \quad \text{if } e(c_t) > 1, \\ \mathbf{de}_{false} &\in \mathbb{R}^j \quad \text{otherwise.} \end{aligned}$$

Finally, we concatenate the extracted embeddings based on both the character- and word-level information into a single feature vector $\mathbf{x}_t$. The feature vector $\mathbf{x}_t$ is given to the LSTM as input. Thus, using the dictionary features and word embeddings, we can inject the dictionary knowledge in the NWS model.

4 Experiments

In order to test the effectiveness of the proposed method, we use the Balanced Corpus of Contemporary Written Japanese (BCCWJ) (Maekawa et al., 2014) as in-domain data and a microblog corpus (Kaji and Kitsuregawa, 2014) as out-domain data. As an evaluation metric, we use the balanced F-measure to evaluate the performance of WS.

4.1 Datasets

For evaluating the performance on the BCCWJ, the manually annotated part of the BCCWJ (core data) is divided into the training, development and evaluation data according to the following division standard: We use 2,983 sentences from ClassA-1[2] as the evaluation data. The remaining part of the corpus, a total of 55,948, is used as the training and 500 as the development data. As the dictionary, we use UniDic 2.1.2, which contains 756,460 entries.

For evaluating the performance on the microblog corpus, we use the Kyoto University Text Corpus (Kawahara et al., 2002) and Kyoto University Web Document Leads Corpus (Hangyo et al., 2012) as the training data. We randomly choose 500 sentences from the combined corpora as the development data, and use 47,642 sentences in the remaining part of the corpora as the training data. As the dictionary for these corpora, we use JUMAN dictionary[3], which contains 702,359 entries, provided with JUMAN (Kurohashi et al., 1994). As the evaluation data, we use a microblog corpus (Kaji and Kitsuregawa, 2014), which contains 1831 sentences collected from Twitter.

As a large-scale unlabeled data, we use 3,512,522 sentences from the BCCWJ excluding ClassA-1.

4.2 Hyper-parameters

Referring to Chen et al. (2015)'s experiments, we set the dimensions of uni-gram embeddings, character types, bi-gram embeddings, tri-gram embeddings, word embeddings and dictionary feature embeddings to 100, 10, 50, 50, 100, 100, respectively. The uni-gram, bi-gram, tri-gram and word embeddings are pre-trained from the large-scale unlabeled data using the Gensim implementation of word2vec (Řehůřek and Sojka, 2010). Other weights and embeddings are randomly initialized from a uniform distribution with range [-0.08, 0.08]. The dimension of the LSTM hidden

[2] http:/plata.ar.media.kyoto-u.ac.jp/mori/research/topics/PST/NextNLP.html

[3] There are two kinds of word segmentation standards in Japanese: JUMAN standard and BCCWJ's short unit words. We choose different dictionaries according to each dataset.

	Precision	Recall	F1
LSTM-CRF (Baseline)	98.93	99.01	98.97
+Trigram	99.11	99.11	99.11
+DictFeat	99.27	99.24	99.25
+Trigram, DictFeat	99.40	99.39	99.39
+AutoSegWords*	98.94	99.07	99.01
+AutoSegWords	99.25	99.25	99.25
+DictFeat, AutoSegWords*	99.34	99.38	99.36
+DictFeat, AutoSegWords	99.49	99.47	99.48
+Trigram, DictFeat, AutoSegWords*	99.37	99.34	99.35
+Trigram, DictFeat, AutoSegWords	**99.50**	**99.50**	**99.50**
KyTea (Neubig et al., 2011)	98.42	98.37	98.39
+Dictionary	99.18	99.10	99.14

Table 1: Results on the BCCWJ ClassA-1. The embedding with * symbol is initialized at random.

	Precision	Recall	F1
LSTM-CRF (Baseline)	82.02 (12,445/15,174)	86.89 (12,445/14,324)	84.38
+Trigram, DictFeat	82.90 (12,628/15,233)	88.16 (12,628/14,324)	85.45
+DictFeat, AutoSegWords	**83.03** (12,572/15,142)	87.78 (12,572/14,324)	85.34
+Trigram, DictFeat, AutoSegWords	82.93 (12,654/15,259)	88.34 (12,654/14,324)	**85.55**
JUMAN (0.7) (Kurohashi et al., 1994)	79.44 (12,711/16,001)	88.74 (12,711/14,324)	83.83
JUMAN++ (1.01) (Morita et al., 2015)	77.73 (12,822/16,495)	**89.51** (12,822/14,324)	83.21

Table 2: Results on the microblog corpus

vectors is set to 150. The character types are used: Hiragana, Katakana, Kanji, Numerals, Alphabets, and Others. The window-size is set to 3.

The number of epochs is set to 50, and we report the result on the evaluation set at the epoch when the best F-measure on the development data is achieved. Parameter optimization is done by stochastic gradient descent (SGD) with a learning rate of 0.005 and a gradient clipping of 5.0. We decay the learning rate when results on the development data do not improve after 5 consecutive epochs. The dropout method is applied to the input layer of the LSTM with dropout rate of 0.2.

4.3 Results and Discussion

As a baseline, we used the NWS model with the uni-gram embeddings, character types and bi-gram embeddings (we refer to this as LSTM-CRF). We evaluate the effectiveness of our method by adding the dictionary features and word embeddings to the LSTM-CRF.

Table 1 lists the results of our method and the baselines in the experiments on the BCCWJ. We used KyTea, an open source implementation of the pointwise method for Japanese WS (Neubig et al., 2011) as a baseline, and the last row "+Dictionary" in Table 1 means KyTea with dictionary features.

Firstly, our proposed method augmented with the dictionary features (**+DictFeat**) outperforms the baselines. This result suggests that the performance for Japanese WS is improved by incorporating the dictionary features.

Secondly, we evaluated the effectiveness of word embeddings. Here, we employ a following strategy with regard to how to obtain pre-trained word embeddings. After we segmented sentences from the large-scale unlabeled data by using the LSTM-CRF (+Trigram, DictFeat), word embeddings are pre-trained from these automatically segmented sentences. Then, our proposed method (**+AutoSegWords**) augmented with word embeddings outperforms the baselines. As the results shown in Table 1, our methods using pre-trained word embeddings obtain a significant improvement as opposed to the ones using random embeddings. Finally, we got the best result when we used all feature vectors on the experiments.

Table 2 lists the results of our methods and the baselines in the experiments on the microblog corpus. We used open source implementations, JUMAN (Kurohashi et al., 1994) and JUMAN++ (Morita et al., 2015) as the baselines.

Some papers have reported low accuracy on the microblog data including lexical variants and emoticons (Kaji and Kitsuregawa, 2014; Takahasi and Mori, 2015). The lattice-based approach JUMAN++ outputs 16,495 words as opposed to 14,324 correct words in the evaluation data. This over-segmentation problem is caused by OOV words such as lexical variants and emoticons being divided into known words. This lowers the precision of WS, and deteriorates the overall accuracy on the microblog data in Japanese. In contrast, our system (LSTM-CRF +Trigram, DictFeat, AutoSegWords) outputs 15,259 words, much less than the lattice-based approaches. As a result, the precision rises, and our methods significantly outperform the baselines on the microblog data.

5 Conclusion

In this paper, we proposed a method that incorporates dictionary information into the NWS model. We conducted an experiment to demonstrate that incorporating the information of a word dictionary improves the performance of the NWS model on both Japanese balanced and microblog corpora. For future work, we plan to expand our model as joint WS and POS tagging in a multi-task learning manner (Collobert et al., 2011).

References

Deng Cai and Hai Zhao. 2016. Neural word segmentation learning for Chinese. In *Proc. ACL*.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Pengfei Liu, and Xuanjing Huang. 2015. Long short-term memory neural networks for Chinese word segmentation. In *Proc. EMNLP*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12(Aug).

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8).

Nobuhiro Kaji and Masaru Kitsuregawa. 2014. Accurate word segmentation and pos tagging for Japanese microblogs: Corpus annotation and joint modeling with lexical normalization. In *Proc. EMNLP*.

Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. 2004. Applying conditional random fields to Japanese morphological analysis. In *Proc. EMNLP*.

Sadao Kurohashi, Toshihisa Nakamura, Yuji Matsumoto, and Makoto Nagao. 1994. Improvements of Japanese morphological analyzer juman. In *Proceedings of The International Workshop on Sharable Natural Language*.

Kikuo Maekawa, Makoto Yamazaki, Toshinobu Ogiso, Takehiko Maruyama, Hideki Ogura, Wakako Kashino, Hanae Koiso, Masaya Yamaguchi, Makiro Tanaka, and Yasuharu Den. 2014. Balanced corpus of contemporary written Japanese. *Language Resources and Evaluation* 48(2).

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* .

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

Tetsuji Nakagawa. 2004. Chinese and Japanese word segmentation using word-level and character-level information. In *Proc. COLING*.

Graham Neubig, Yosuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust, adaptable Japanese morphological analysis. In *Proc. ACL*.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2014. Max-margin tensor neural network for Chinese word segmentation. In *Proc. ACL*.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*.

Fumihiko Takahasi and Shinsuke Mori. 2015. Keyboard logs as natural annotations for word segmentation. In *Proc. EMNLP*.

Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. 2013. Deep learning for Chinese word segmentation and POS tagging. In *Proc. EMNLP*.