Long Short-Term Memory for Japanese Word Segmentation

Yoshiaki Kitagawa and Mamoru Komachi
Tokyo Metropolitan University
6-6 Asahigaoka, Hino, Tokyo 191-0065, Japan
kitagawa-yoshiaki@ed.tmu.ac.jp, komachi@tmu.ac.jp

Abstract
This study presents a Long Short-Term Memory (LSTM) neural network approach to Japanese word segmentation (JWS). Previous studies on Chinese word segmentation (CWS) succeeded in using recurrent neural networks such as LSTM and gated recurrent units (GRU). However, in contrast to Chinese, Japanese includes several character types, such as hiragana, katakana, and kanji, that produce orthographic variations and increase the difficulty of word segmentation. Additionally, it is important for JWS tasks to consider a global context, and yet traditional JWS approaches rely on local features. In order to address this problem, this study proposes employing an LSTM-based approach to JWS. The experimental results indicate that the proposed model achieves state-of-the-art accuracy with respect to various Japanese corpora.

1 Introduction
Word segmentation is a fundamental task in Japanese language processing. Especially, errors in word segmentation in East Asian languages, such as Japanese and Chinese, that lack a trivial word segmentation process can cause problems for downstream NLP applications. Thus, it is crucial to perform accurate word segmentation for Japanese NLP tasks.

In order to achieve high accuracy, almost all modern methods in JWS utilize discriminative models with extensive feature engineering. However, machine learning-based methods tend to require hand-crafted feature templates and suffer from data sparseness. Thus, neural network models have been investigated for various NLP tasks to address the problem of feature engineering (Liu et al., 2015; Sutskever et al., 2014; Socher et al., 2013; Turian et al., 2010; Mikolov et al., 2013a). Neural network models enable the use of dense feature vectors termed as embeddings learned through representation learning (Mikolov et al., 2013b).

Another important problem in JWS corresponds to context modeling. Traditional JWS methods employ feature templates to expand local features in a fixed window. However, global information beyond the window is not considered. Conversely, recurrent neural network models grasp long distance information due to Long Short-Term Memory (LSTM) and record state-of-the-art accuracy in Chinese word segmentation (Chen et al., 2015b). However, it is not clear as to whether the LSTM approach is also effective in JWS since there are many types of character sets in Japanese that produce orthographic variations.

Therefore, we propose an LSTM network for Japanese word segmentation by incorporating character-level embeddings and long distance dependency. The main contributions of this study are as follows:

- We propose an LSTM model for JWS and investigate methods to utilize sparse features, such as character type, character N-gram, and dictionary features.
- The experimental results indicate that the proposed word segmentation model achieves state-of-the-art performance in both token-level and sentence-level accuracy with respect to various datasets.

2 LSTM for Japanese Word Segmentation
Machine learning-based approaches for word segmentation build a classifier from an annotated
corpus to classify whether word boundaries exist around a target character. In word segmentation, each character is assigned to several labels such as \{B, I, E, S\}, \{B, I, E\}, and \{B, I\} to indicate the segmentation. \{B, I, E, S\} represent Begin, Inside, End, and Single, respectively.

Classification of these labels is performed by running Viterbi algorithm over a word lattice (Kudo et al., 2004; Nakagawa, 2004; Kaji and Kitsuregawa, 2013) or by independently performing predictions (Neubig et al., 2011). However, the previous approaches use feature templates to expand window-based local features and suffer from data sparseness and lack of global information in a sentence.

In order to address this problem, we propose character-based embeddings and Long Short Term Memory (LSTM) network for Japanese word segmentation (JWS). Figure 1 shows an overview of the proposed framework. The model is similar to previous studies on CWS (Chen et al., 2015) although it incorporates character-based N-gram embeddings and a word dictionary sparse feature.

In the neural architecture, character based embeddings for context characters are extracted by the lookup table layer and concatenated into a single vector \(x_t \in \mathbb{R}^{H_1}\), where \(H_1\) is the size of input layer. Then \(x_t\) is passed into the next layer that performs linear transformation \(W_1\) followed by an element-wise activation function \(g\) such as sigmoid and \(\tanh\) functions:

\[
h_t = g(W_1 x_t + b_1)
\]  

(1)

where \(W_1 \in \mathbb{R}^{H_2 \times H_1}\), \(b_1 \in \mathbb{R}^{H_2}\), and \(h_t \in \mathbb{R}^{H_2}\). Additionally, \(H_2\) denotes a hyper-parameter which indicates the number of hidden units in the hidden layer, \(b_1\) denotes a bias vector, and \(h_t\) denotes the resulting hidden vector. The final output is obtained by performing a softmax function after a similar linear transformation \(W_2\) to the hidden vector as follows:

\[
y_t = \text{softmax}(W_2 h_t + b_2)
\]  

(2)

where \(W_2 \in \mathbb{R}^{T \times H_2}\), \(b_2 \in \mathbb{R}^T\), and \(y_t \in \mathbb{R}^T\). Thus, \(b_2\) denotes a bias vector, and \(y_t\) denotes the distribution vector for each possible label. In JWS, the most prevalent label set corresponds to \{B, I, E, S\} although the label sets do not significantly affect accuracy in our preliminary experiments.

The RNN addresses the problem of lack of history by using recurrent hidden units in which output at each time is dependent on that of the previous time. The recurrent neural network (RNN) is demonstrated as successful with respect to several NLP tasks such as language modeling (Mikolov et al., 2010) and text generation (Sutskever et al., 2011). However, it fails to propagate dependencies over a long distance because of the vanishing and exploding gradient problem (Hochreiter and Schmidhuber, 1997).

The LSTM provides addresses the problem by incorporating memory units to learn when to forget previous information and when to update memory cells given new information.

2.1 Character-Level Features

This section discusses character-level features. As shown in Table 1, this paper introduces Character embedding and Character type embedding, and their N-gram for JWS. We will describe the character vector \(c_t\) for JWS below. Formally, the character vector \(c_t\) is defined as follows:

\[
c_t = l_t \oplus e_t
\]  

(3)

where \(\oplus\) denotes concatenation of the vectors, and \(l_t\) and \(e_t\) denote character embeddings and character type embeddings, respectively. These embeddings are fed to the input layer.

In the following subsections, we discuss three features that are frequently used in JWS, and describe their realization as embeddings in the proposed architecture.

2.1.1 Character Embeddings

In a word segmentation task, a character dictionary \(C\) of size \(|C|\) is often created. Traditional machine-learning approaches that use feature templates treat each character independently as a one-hot vector. However, it is natural for a neural network model to represent discrete data as distributed vectors termed as embeddings (Bengio et al., 2003; Collobert and Weston, 2008). Representation learning is an actively studied topic in NLP because it overcomes the data sparseness problem. Thus, the same practice is followed to represent each character as a real-valued vector \(v_c\) in \(\mathbb{R}^d\) where \(d\) is the dimensionality of the vector space. With respect to each character, the corresponding character embedding \(v_c\) is selected by a lookup table.
2.1.2 Character Type Embeddings

Character embeddings are extremely effective in identifying prefixes and postfixes. However, they could be too sparse while crossing a word boundary. In order to address this problem, it is helpful to exploit character types, such as hiragana, katakana, and kanji (e.g., ひらがな, カタカナ, 漢字), for Japanese word segmentation (Neubig et al., 2011). For example, katakana sequence tends to correspond to a loan word, and a transition from a character type to another is likely to correspond to a word boundary (Nagata, 1999).

2.1.3 Character-Based N-gram Embeddings

In addition to character type, the N-gram is effective in JWS (Neubig et al., 2011). Thus, character type sequence information is incorporated as embeddings. Each character is converted to a one-hot vector corresponding to its character type. A one-hot vector is composed of either hiragana, katakana, kanji, alphabet, number, symbol, start symbol and terminal symbol. The advantages of a deep neural network include dealing with a sparse vector by converting it to a dense vector. This enables the utilization of a sparse feature such as character tri-gram. Additionally, a character-based N-gram is effective for sentence similarity and part-of-speech tagging (Wieting et al., 2016) and for Japanese morphological analysis (Neubig et al., 2011). Therefore, N-gram is used for character and character type embeddings. More precisely, a one-hot vector is created for each unigram as well as for each bi-gram and tri-gram. Each embedding is selected by a lookup table as well as uni-gram embeddings.

The embedding vectors \( l_t \) and \( e_t \) are defined as follows:

\[
l_t = l_{t-2:t} \oplus l_{t-1:t} \oplus l_{t} \tag{4}
\]

\[
e_t = e_{t-2:t} \oplus e_{t-1:t} \oplus e_{t} \tag{5}
\]

where \( l_{a:b} \) denotes the embedding for the strings from \( a \) to \( b \). The same holds for \( e_t \).

2.2 Incorporating Word Dictionary

Character embeddings, character type embeddings, and their N-gram extensions perform an excellent job with respect to learning character-based features from an annotated corpus. However, character-based JWS models lack word-level information that is useful to determine the character sequences constituting a word. Thus, a Japanese morphological analyzer typically uses a dictionary. It is essential for JWS that uses a word lattice during decoding to use word-level information such as a unigram and a bigram although this is not necessary for character-based JWS approaches.
However, it is not trivial to encode dictionary information into a neural network architecture. Tsuboi (2014) suggests that it is not effective to learn both dense continuous and sparse discrete vector representations in the same layer. Thus, we followed the same practice to create a sparse dictionary vector although it is used for the input to the final output layer as shown in Figure 1 instead of learning embeddings.

Figure 2 illustrates the creation of a dictionary vector. The dictionary vector is composed of three parts as follows: left side feature $L$, right side feature $R$, and inside feature $I$. For example, $L2$ is activated if a word with a length corresponding to 2 exists in the dictionary on the left side of the prediction point. If the length of the word exceeds a certain threshold, the word length is cut off with respect to the length. In the study, 4 is adopted as the threshold following Neubig et al. (2011). In contrast to $L$ and $R$, $I2$ is fired if there exists a word that spans across the boundary and possesses length 2. It should be noted that $I$ is activated only if the length of the word exceeds 1 based on its definition. Finally, the feature vectors are concatenated to a single vector termed as a dictionary vector $d_t$.

The dictionary vector $d_t$ is concatenated to the current hidden vector. It should be noted that the current hidden vector $h_t$ is on top of the LSTM network. Formally, the new hidden vector $h'_t$ is defined as follows:

$$h'_t = h_t \oplus d_t$$  \hspace{1cm} (6)

### 2.3 Training

In this study, a cross entropy error is adopted as a loss function. Given an output vector $y_t$, the loss to a correct distribution corresponding to $c_t$ is computed as follows:

$$\text{loss} = \sum_t -i_t \log y_t + \frac{1}{2}\lambda \|\theta\|^2$$  \hspace{1cm} (7)

where $i_t$ denotes correct label distribution, $\lambda$ denotes a hyper parameter of L2 regularization, and $\theta$ indicates all parameters of the model.

Following (Socher et al., 2013), the diagonal variant of AdaGrad (Duchi et al., 2011) with mini batches is used to minimize the objective. The update for the $i$-th parameter $\theta_{t,i}$ at time step $t$ is defined as follows:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sum_{\tau=t}^\tau g_{\tau,i}^2} g_{t,i}$$  \hspace{1cm} (8)

where $\alpha$ denotes the initial learning rate, and $g_{\tau} \in \mathbb{R}^{|\theta_i|}$ denotes the subgradient at time step $\tau$ for parameter $\theta_i$.

### 3 Experiments

We evaluated the proposed neural word segmentation method on several JWS corpora. In order
to evaluate neural network architectures, we prepare a feed forward network (FFNN) and a recurrent neural network (RNN) for JWS. The FFNN is illustrated by a dotted line in Figure 1. Additionally, RNN uses the same inputs as LSTM although it does not use any LSTM units.

The experiments are separated into two parts. First, neural network architectures and features are compared with previous state-of-the-art method on a balanced corpus (See top of Table 3). Second, the proposed method is evaluated on a newspaper corpus annotated with a different segmentation criterion (See Table 4).

### 3.1 Datasets

We evaluate the methods with respect to two different datasets, namely a popular Japanese corpus BCCWJ (Balanced Corpus of Contemporary Written Japanese) (Maekawa et al., 2014) and another widely used Japanese corpus Kyoto University Corpus version 4.0 (KC). The BCCWJ is composed of various domains while KC only includes the newswire domain. The details of the corpora are shown in Table 1. The train and test split of BCCWJ are followed according to the Project Next NLP\(^1\). We adopted the same train and test split of KC used in previous studies (Kudo et al., 2004; Uchimoto et al., 2001).

With respect to word-level features, Neubig et al. (2011) do not use any external dictionary but use the dictionary created from the training corpus. Hence, the same scenario is adopted, and all the words in training corpus are added although singletons are omitted to prevent overfitting on the training data as described in (Neubig et al., 2011). In order to analyze the effect of the dictionary feature, we recreate a larger dictionary created from both training and test sets. This is termed as gold dict in Table 3.

### 3.2 Tools

In the experiments, we use a popular JWS tool KyTea (ver.0.4.6)\(^2\) that implements (Neubig et al., 2011). We train a KyTea model by the provided scripts for training. It internally creates a dictionary as described above. Pre-trained KyTea models adopt their own word segmentation criterion extended from that of BCCWJ, and thus KyTea models are re-trained to ensure a fair comparison.

Additionally, we implement neural network-based JWS models including FFNN, RNN, and LSTM by using a neural network framework termed as Chainer (ver 1.4.0)\(^3\)(Tokui et al., 2015).

### 3.3 Hyper Parameters

We investigate several parameter combinations inspired by previous studies (Chen et al., 2015b) in the preliminary experiments. The complete set of parameters used in the study is shown in Table 2. The BCCWJ development set is used for tuning hyper parameters. Based on Figure 3, it is observed that the proposed method is very fast to learn although a CPU is used. The duration corresponds to 24 h per epoch.

**Window size.** In the preliminary experiments, the results indicate that window size 5 is better than others in terms of both accuracy or time. Window size 7 did not significantly improve the F1 score.

---

\(^1\)https://goo.gl/QCxxwB  
\(^2\)http://www.phontron.com/kytea/index.html  
\(^3\)http://chainer.org
from window size 5, and thus window size 5 is selected due to the trade-off between accuracy and time.

**Dimension of character type embeddings.** The dimension of character embeddings is fixed by following (Chen et al., 2015b). However, we search six dimensions of character type embeddings. The dimension of character type embeddings is selected as corresponding to 10 because 10 yields a better performance than others with respect to both accuracy or time as indicated by the preliminary experiments. Although window sizes 20 and 50 are competitive with respect to window size 10, window size 10 is selected given the time complexity.

**Label set.** In CWS, the label set \{B, I, E, S\} is often used. In contrast, various label sets are adopted in JWS. We explore three label sets and find that \{B, I, E, S\} is slightly better than the others.

**Learning rate.** In this task, the learning rate largely affects accuracy. Although a learning rate of 0.1 appears to exceed those in other NLP tasks, a small learning rate (such as 0.01) degrades accuracy and significantly affects learning time. Thus, the learning rate of 0.1 is selected for all the experiments.

### 3.4 Results

Tables 3 and 4 show the experimental results for the BCCWJ Corpus and Kyoto Corpus. In both corpora, the LSTM-based method outperformed the state-of-the-art method (Neubig et al., 2011). Table 5 illustrates the performance of the two methods per domain breakdown. The accuracy of the proposed method in terms of token-level F1 and sentence-level accuracy exceeded those of others in four domains out of six, and this resulted in improvements in the overall performance.

### 4 Discussion

**Models.** Table 3 shows that LSTM is superior to FFNN and RNN by using the same feature set (character embeddings only). It demonstrates the effectiveness of modeling a context by LSTM.

**Character type.** Comparing LSTM with LSTM + char type, F1 improves by 0.25 points. The result shows that character type embeddings are useful in JWS. However, the advantage of using character type embeddings is smaller when compared to that when the model is changed to LSTM. This suggests that the choice of architecture has a greater effect on the final accuracy.

**Dictionary feature.** The addition of a dictionary feature to LSTM + char type improves F1 by 0.37. This result shows that dictionary feature is effective in JWS. However, the addition of the dictionary feature to LSTM + char type + N-gram does not result in any notable difference. It is assumed that character based N-gram embeddings subsume the dictionary feature because the dictionary is created from the training corpus. Additional experiments using the gold dictionary created from the test corpus support this hypothesis.

**N-gram embedding.** A comparison of LSTM + char type with LSTM + char type + N-gram indicates N-gram embeddings significantly improve the performance of the model by a large margin. Traditional machine learning-based approaches, such as CRF and SVM, do not use the advantage of the sparse features while the LSTM-based proposed model successfully exploits this information.

### 5 Error Analysis

#### 5.1 Effect of Domain

In order to determine the characteristics of the proposed method, we conducted an error analysis by comparing the proposed method with KyTea with respect to different domains. Thus, we computed F1 for each domain of BCCWJ, and counted the number of incorrect sentences. Table 5 summarizes token-level and sentence-level comparison between the proposed model and KyTea. We selected the pairs that exhibited a large margin in F1. Thus, we analyzed White paper, Magazine, and Book.

---

4It should be noted that the singletons in the combined corpus are removed while creating the gold dictionary, and thus the test corpus may still contain words that are not in the gold dictionary.
Table 3: Experimental results of Japanese word segmentation on BCCWJ.

| Methods                        | F1  |
|-------------------------------|-----|
| FFNN                          | 96.53 |
| RNN                           | 96.46 |
| LSTM                          | 97.00 |
| LSTM + char type              | 97.25 |
| LSTM + char type + dict       | 97.37 |
| LSTM + char type + N-gram     | 98.41 |
| LSTM + char type + N-gram + dict | 98.42 |
| LSTM + char type + N-gram + dict (gold) | 98.67 |
| KyTea 0.4.6                   | 98.34 |

Table 4: Experimental results of Japanese word segmentation on Kyoto Corpus.

| Methods                        | F1  |
|-------------------------------|-----|
| LSTM + char type + N-gram + dict | 96.47 |
| KyTea 0.4.6                   | 96.21 |

White paper. This domain comprises of official documents published by the government. Thus, kanji covers a large portion of the corpus. Additionally, the number of characters per sentence is high. In this domain, the proposed method is only inferior to Neubig et al. (2011) with respect to both F1 and the number of incorrect sentence, and this is potentially because a long sequence introduced noise to the LSTM-based models.

Magazine. This domain contains colloquial expressions as well as formal expressions. Hiragana occupies a large portion of this corpus due to the colloquial expressions. Furthermore, F1 for this domain is the lowest in both methods. The results indicate that hiragana exhibits a poor performance. However, the proposed method is more robust than KyTea in this domain. This could be due to the modeling of contextual information since the hiragana sequence tends to fall outside the local window size.

Book. This domain typically includes named entities (NE) such as a company name. This corpus is balanced in terms of the proportion of character types. Overall, the proposed model tends to be robust for compounds of different character type (e.g. Famiポート (Fami Port; multimedia vending machine)), whereas Neubig et al. (2011)’s model correctly classifies words composed of unique character type (e.g. ポストドクター (Postdoc)). It is considered that the difference between token-level and sentence-level accuracy highlights the characteristic of the methods. The proposed method typically produces less errors although it does not consistently perform word segmentation across the corpus.

5.2 Example

In order to investigate the characteristics of the proposed method from a different angle, we demonstrate actual examples of word segmentation. Table 6 shows a comparison of four examples for the current study and KyTea 0.4.6. The proposed method possesses two characteristics.

The first characteristic is that strings with the same character type tend to form a word unit. This characteristic is demonstrated by the first and second examples. In the first example, “と” and “りゅう” are different words although they possess the same character type “Hiragana,” and thus they are incorrectly combined to form a fake word. In the second example, “が” and “まんま” are also incorrectly connected. This type of error tends to occur when the character type corresponds to “Hiragana,” that includes many high-frequency ambiguous single-character particles.

Another characteristic of this method is that words including different character types tend to be broken by KyTea at the position where a character type is changed. This characteristic is demonstrated by the third example. In the third example, “ため池 (storage reservoir)” is a single word that consists of “ため (storage)” and “池 (reservoir)” although KyTea fails to recognize the word since “ため” and “池” possess different character types. In contrast, the proposed method correctly identifies the word.

However, there are cases where contrary results are indicated. In the fourth example, “と” and “うんざり” correspond to different words and possess the same character type “Hiragana”. An analysis of the first and second examples indicates that the proposed method tends to form a fake word that comprises of the same character type although it yields a correct segmentation result. Thus, the qualitative analysis is considered difficult.
Table 5: Token-level and sentence-level performance on various domains. The first term of the number of incorrect sentences indicates that both JWS predicted in correct results, and the second term indicates that only the corresponding method predicted an incorrect result.

| Domain            | KyTea 0.4.6 | This work |
|-------------------|-------------|-----------|
|                   | F1 | # of incorrect sent. | F1 | # of incorrect sent. |
| Yahoo! Japan Answers | 98.38 | 50+25 | 98.44 | 50+19 |
| Yahoo! Japan Blog  | 99.75 | 76+22 | 99.73 | 76+21 |
| White paper       | 99.20 | 60+21 | 99.08 | 60+24 |
| Book              | 98.15 | 63+19 | 98.28 | 63+28 |
| Magazine          | 96.70 | 73+29 | 97.25 | 73+17 |
| Newspaper         | 98.19 | 60+36 | 98.46 | 60+15 |
| All               | 98.34 | 382+152 | 98.42 | 382+124 |

Table 6: An example of the error in this study and KyTea. The character “|” indicates the word boundary, and the bold face indicates the incorrect part.

| Methods          | Example                                      | correct/incorrect |
|------------------|----------------------------------------------|-------------------|
| This work        | エルマー | とりゅう | の | 絵 | で | incorrect |
| KyTea 0.4.6      | エルマー | とりゅう | の | 絵 | で | correct   |
| This work        | うち | がまんま | その | 環境 | です | incorrect |
| KyTea 0.4.6      | うち | がまんま | その | 環境 | です | correct   |
| This work        | 七百六十| 一 | の | ため池 | など | 被害 | correct |
| KyTea 0.4.6      | 七百六十| 一 | の | ため池 | など | 被害 | incorrect |
| This work        | 思う | と うんざり | です | . | correct |
| KyTea 0.4.6      | 思う | と うんざり | です | . | incorrect |

6 Related Works

In JWS, a supervised learning approach is widely used. A popular method in JWS involves creating a word lattice by using a dictionary and using Viterbi decoding (Kudo et al., 2004; Sassano, 2002). This approach is known to yield accurate results by considering the sequence of words although it is not robust if training data differs from test data. Another popular approach employs point-wise prediction by using a local window (Neubig et al., 2011; Neubig and Mori, 2010). However, both approaches do not consider the global context because they use feature templates of a fixed length. Additionally, they both suffer from feature sparseness.

Recently, deep neural network architectures have been widely studied in the CWS task (Chen et al., 2015b,a; Pei et al., 2014). However, a deep neural network approach requires high computational cost when compared with previous approaches. In JWS, Morita et al. (2015) proposed integrating a recurrent neural network language model into JWS by interpolating it with traditional JWS. As opposed to using recurrent neural architecture as side information, word segmentation in Japanese is directly learned by using LSTM.

Recently, a neural network approach for normalization was explored (Kann et al., 2016; Ikeda et al., 2016). Kann et al. (2016) proposed a character based encoder-decoder model and achieved state-of-the-art accuracy for the task of canonical morphological segmentation. Because their method is based on unsupervised learning, it can be learned at a low cost, but it is necessary to adjust word segmentation criterion to human annotation. Ikeda et al. (2016) also presented an encoder-decoder model for Japanese text normalization. However, their model was only as good as conventional CRF even though it was trained with a large scale artificially created corpus.
7 Conclusion

In this paper, we presented an LSTM neural network approach to JWS. We proposed learning Japanese-specific features, such as character type and character N-gram, as embeddings, and dictionary features as a sparse vector. The proposed method has been shown to achieve state-of-the-art accuracy on various domains. The empirical results of the study have suggested further opportunities to investigate continuous features not only for WS but also for POS tagging.

In JWS, it is important to deal with colloquial expressions that are frequently found in dialogue-based conversations and web text (Saito et al., 2014; Sasano et al., 2013; Kaji and Kitsuregawa, 2014). It is expected that deep neural architectures, such as CNNs, may be effective in this scenario because of their ability to learn robust representations for characters and words (Ling et al., 2015).

Acknowledgments

This work was supported by the Microsoft Research Collaborative Research (CORE) Projects.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. J. Mach. Learn. Res. 3:1137–1155.

Deng Cai and Hai Zhao. 2016. Neural word segmentation learning for chinese. In ACL. pages 409–420.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, and Xuanjing Huang. 2015a. Gated recursive neural network for Chinese word segmentation. In ACL-IJCNLP. pages 1744–1753.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Pengfei Liu, and Xuanjing Huang. 2015b. Long short-term memory neural networks for Chinese word segmentation. In EMNLP. pages 1197–1206.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In ICML. pages 160–167.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research 12(Jul):2121–2159.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation 9(8):1735–1780.

Taishi Ikeda, Hiroyuki Shindo, and Yuji Matsumoto. 2016. Japanese text normalization with encoder-decoder model. In WNUT. pages 118–126.

Nobuhiro Kaji and Masaru Kitsuregawa. 2013. Efficient word lattice generation for joint word segmentation and pos tagging in Japanese. In IJCNLP. pages 153–161.

Nobuhiro Kaji and Masaru Kitsuregawa. 2014. Accurate word segmentation and pos tagging for Japanese microblogs: Corpus annotation and joint modeling with lexical normalization. In EMNLP. pages 99–109.

Katharina Kann, Ryan Cotterell, and Hinrich Schütze. 2016. Neural morphological analysis: Encoding-decoding canonical segments. In EMNLP. pages 961–967.

Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. 2004. Applying conditional random fields to Japanese morphological analysis. In EMNLP. pages 230–237.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fernández, Silvio Amir, Luis Marujo, and Tiago Luís. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In EMNLP. pages 1520–1530.

Pengfei Liu, Xipeng Qiu, Xinchi Chen, Shiyu Wu, and Xuanjing Huang. 2015. Multi-timescale long short-term memory neural network for modelling sentences and documents. In EMNLP. pages 2326–2335.

Kikuo Maekawa, Makoto Yamazaki, Toshinobu Ogiso, Takehiko Maruyama, Hideki Ogura, Wakako Kashiho, Hanae Koiso, Masaya Yamaguchi, Makiro Tanaka, and Yasuharu Den. 2014. Balanced corpus of contemporary written Japanese. Language Resources and Evaluation 48(2):345–371.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In Inter-speech. volume 2, page 3.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013a. Distributed representations of words and phrases and their compositionality. In NIPS. pages 3111–3119.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In NIPS. pages 3111–3119.

Hajime Morita, Daisuke Kawahara, and Sadao Kurohashi. 2015. Morphological analysis for unsegmented languages using recurrent neural network language model. In EMNLP. pages 2292–2297.
Masaaki Nagata. 1999. A part of speech estimation method for Japanese unknown words using a statistical model of morphology and context. In ACL. pages 277–284.

Tetsuji Nakagawa. 2004. Chinese and Japanese word segmentation using word-level and character-level information. In ACL. page 466.

Graham Neubig and Shinsuke Mori. 2010. Word-based partial annotation for efficient corpus construction. In LREC. pages 2723–2727.

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

Wenzhe Pei, Tao Ge, and Baobao Chang. 2014. Max-margin tensor neural network for Chinese word segmentation. In ACL. pages 293–303.

Itsumi Saito, Kugatsu Sadamitsu, Hisako Asano, and Yoshihiro Matsuo. 2014. Morphological analysis for Japanese noisy text based on character-level and word-level normalization. In COLING. pages 1773–1782.

Ryohei Sasano, Sadao Kurohashi, and Manabu Okumura. 2013. A simple approach to unknown word processing in Japanese morphological analysis. In IJCNLP. pages 162–170.

Manabu Sassano. 2002. An empirical study of active learning with support vector machines for Japanese word segmentation. In ACL. pages 505–512.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In ACL. pages 455–465.

Ilya Sutskever, James Martens, and Geoffrey E. Hinton. 2011. Generating text with recurrent neural networks. In ICML. pages 1017–1024.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In NIPS. pages 3104–3112.

Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: a Next-Generation open source framework for deep learning. In NIPS Workshop.

Yuta Tsuboi. 2014. Neural networks leverage corpus-wide information for part-of-speech tagging. In EMNLP. pages 938–950.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In ACL. pages 384–394.

Kiyotaka Uchimoto, Satoshi Sekine, and Hitoshi Ishihara. 2001. The unknown word problem: a morphological analysis of Japanese using maximum entropy aided by a dictionary. In EMNLP. pages 91–99.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Charagram: Embedding words and sentences via character n-grams. In EMNLP. pages 1504–1515.

Meishan Zhang, Yue Zhang, and Guohong Fu. 2016. Transition-based neural word segmentation. In ACL. pages 421–431.