Stochastic Tokenization with a Language Model for Neural Text Classification

Tatsuya Hiraoka[1], **Hiroyuki Shindo**[2], and **Yuji Matsumoto**[2,3]

[1]Tokyo Institute of Technology
[2]Nara Institute of Science and Technology
[3]RIKEN Center for Advanced Intelligence Project (AIP)
[1]tatsuya.hiraoka@nlp.c.titech.ac.jp
[2]{shindo,matsu}@is.naist.jp

Abstract

For unsegmented languages such as Japanese and Chinese, tokenization of a sentence has a significant impact on the performance of text classification. Sentences are usually segmented with words or subwords by a morphological analyzer or byte pair encoding and then encoded with word (or subword) representations for neural networks. However, segmentation is potentially ambiguous, and it is unclear whether the segmented tokens achieve the best performance for the target task. In this paper, we propose a method to simultaneously learn tokenization and text classification to address these problems. Our model incorporates a language model for unsupervised tokenization into a text classifier and then trains both models simultaneously. To make the model robust against infrequent tokens, we sampled segmentation for each sentence stochastically during training, which resulted in improved performance of text classification. We conducted experiments on sentiment analysis as a text classification task and show that our method achieves better performance than previous methods.

1 Introduction

Tokenization is a fundamental problem in text classification such as sentiment analysis (Tang et al., 2014; Kim, 2014; dos Santos and Gatti, 2014), topic detection (Lai et al., 2015; Zhang et al., 2015), and spam detection (Liu and Jia, 2012; Liu et al., 2016). In text classification with neural networks, sentence representation is calculated based on tokens that compose the sentence. Specifically, a sentence is first tokenized into meaningful units such as characters, words, and subwords (Zhang et al., 2015; Joulin et al., 2017). Then, the token embeddings are looked up and fed into a neural network encoder such as a feed-forward neural network (Iyyer et al., 2015), a convolutional neural network (CNN) (Kim, 2014; Kalchbrenner et al., 2014), or a long short-term memory (LSTM) network (Wang et al., 2016a,b).

For English and other languages that use the Latin alphabet, the whitespace is a good indicator of word segmentation. However, tokenization is a non-trivial problem in unsegmented languages such as Chinese and Japanese since they have no explicit word boundaries. For these languages, tokenizers based on supervised machine learning with a dictionary (Zhang et al., 2003; Kudo, 2006) have been used to segment a sentence into units (Lai et al., 2015). In addition, we use a neural network-based word segmenter to tokenize a raw corpus in Chinese text classification (Zhou et al., 2016; Zhang and Yang, 2018). In machine translation, subword tokenization with byte pair encoding (BPE) addresses the problem of unknown words and improves performance (Sennrich et al., 2016).

However, segmentation is potentially ambiguous, and it is unclear whether preset tokenization offers the best performance for target tasks. To address this problem, in this paper, we propose a new tokenization strategy that segments a sentence stochastically and trains a classification model with various segmentations. During training, our model first segments sentences into tokens stochastically with the language model and then feeds the tokenized sentences into a neural text classifier. The text classifier is trained to decrease the cross-entropy loss for true labels, and the language model is also learned with the sampled tokenization. This enables the model to segment the test dataset by taking into account recent tokenization in training. We find that sampling the tokens of a sentence stochastically renders the text classifier more robust to tokenization. Additionally, updating the language model improves the performance of the test set.

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 1620–1629
Florence, Italy, July 28 - August 2, 2019.

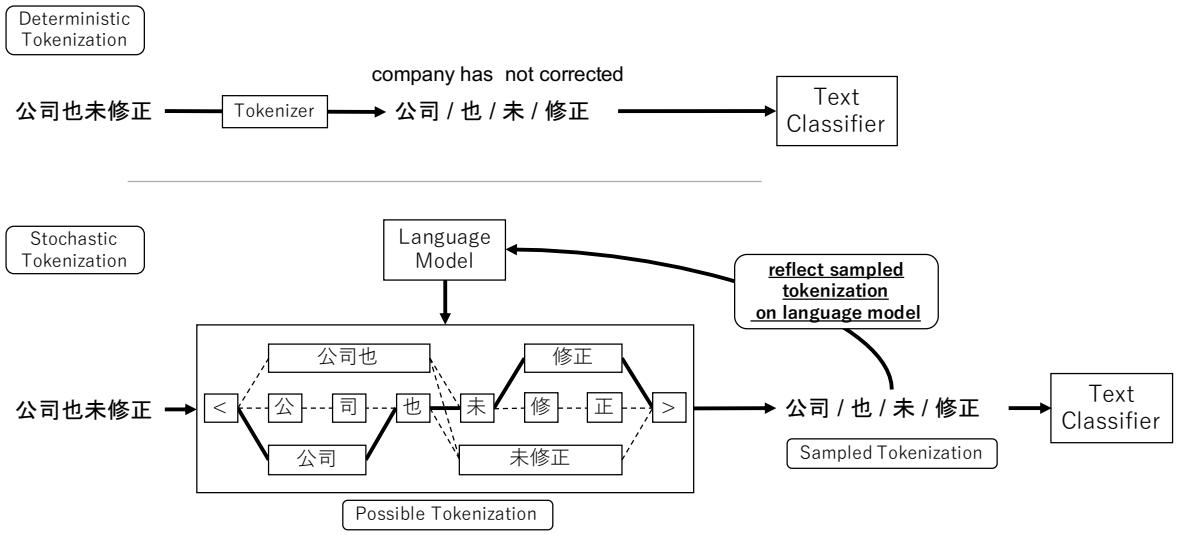

Figure 1: (Top) Schematic of the previous classification model with deterministic tokenization, and (bottom) our proposed model, which tokenizes a raw sentence stochastically with a language model and updates it by sampled tokens in the training phase. $<$ and $>$ in the lattice are special tokens indicating the beginning and end of a sentence, respectively. We input a tokenized sentence into the neural text classifier, and it is trained with its gold label.

2 Neural Text Classification

Text classification refers to the classifying of a sentence into a corresponding label. Typically, a neural network text classifier represents the sentence $s = t_1 ... t_n ... t_N$ as a vector $\boldsymbol{v}_s$ and predicts the distribution of labels by transforming the vector. For example, $\boldsymbol{v}_s$ is given by a forward LSTM as

$$\begin{aligned} \boldsymbol{c}_n^{\text{token}}, \boldsymbol{h}_n^{\text{token}} &= \text{LSTM}(\boldsymbol{c}_{n-1}^{\text{token}}, \boldsymbol{h}_{n-1}^{\text{token}}, \boldsymbol{v}_{t_n}) \\ \boldsymbol{v}_s &= \boldsymbol{h}_N^{\text{token}} \end{aligned} \quad (1)$$

where t_n is the n-th token composing a sentence of length N, and $\boldsymbol{v}_{t_n}$ is the vector for token t_n. $\boldsymbol{h}$ and $\boldsymbol{c}$ are output vectors and cell states of LSTM, respectively. The N-th output vector $\boldsymbol{h}_N^{\text{token}}$ of LSTM is assigned to the token vector $\boldsymbol{v}_s$.

The token vector $\boldsymbol{v}_t$ is obtained by concatenating a token-level representation $\boldsymbol{v}^{\text{token}}$ and a character-level representation $\boldsymbol{v}^{\text{char}}$ as follows:

$$\boldsymbol{v}_t = \boldsymbol{W}^{\text{cat}}(\boldsymbol{v}_t^{\text{token}}; \boldsymbol{v}_t^{\text{char}}) + \boldsymbol{b}^{\text{cat}} \quad (2)$$

where $\boldsymbol{v}_t^{\text{token}}$ is extracted from a lookup table, and $\boldsymbol{v}_t^{\text{char}}$ is calculated by a single-layered and unidirectional LSTM from embeddings of the characters composing the token as well as the token-level LSTM (1). $\boldsymbol{W}^{\text{cat}}$ and $\boldsymbol{b}^{\text{cat}}$ are parameters.

The probability $p(y_s = u|\boldsymbol{v}_s)$ that the sentence class y_s is a u-th class is calculated by a decoder with a linear layer as

$$p(y_s = u|\boldsymbol{v}_s) = \text{softmax}(\boldsymbol{W}^{\text{dec}}\boldsymbol{v}_s + \boldsymbol{b}^{\text{dec}})_u \quad (3)$$

where $\boldsymbol{W}^{\text{dec}}$ and $\boldsymbol{b}^{\text{dec}}$ are the parameters, and $\text{softmax}(\cdot)$ refers to the softmax function. $(\cdot)_u$ is the u-th element of a vector. The neural text classifier is trained with the optimizer to minimize cross-entropy loss for gold labels.

3 Proposed Model

3.1 Model Outline

We focus on the tokenization of neural text classification. During the training phase of text classification, the proposed model tokenizes an input sentence stochastically in every epoch with a language model. A neural text classifier takes the tokenized sentence and predicts a label for the sentence. In the evaluation, our model tokenizes the test set by the Viterbi algorithm with a language model.

When sampling tokenization in training, we consider that the model can achieve higher performance by tokenizing test data under the same criterion used in training. For example, when a classification model is trained with the word "anthropology" tokenized as "an/thro/polo/gy," the similar word "anthropological" in the test data should be tokenized as "an/thro/polo/gical" rather than "anthro/polo/g/ical." To realize this, our model

updates its language model depending on the currently sampled tokens in the training phase.

Algorithm 1 outlines our model. After setting the initial language model and a classifier model, every sentence in a mini-batch for training is tokenized stochastically, and the language model is updated based on the tokenized sentence. An example of our model's processing is illustrated at the bottom of figure 1. Compared with conventional text classification with deterministic tokenization, our model incorporates a language model into the training process and trains in both tokenization and text classification simultaneously.

Algorithm 1 Learning Algorithm

```
set/train a language model LM
set a classifier model CM
while epoch < maxEpoch do
  for each miniBatch do
    for each sentence s in miniBatch do
      t_s = tokenize s with LM
      update LM with t_s
    end for
    update CM with miniBatch
  end for
end while
```

3.2 Nested Unigram Language Model

To sample tokens for a sentence, we employed a nested unigram language model, which was proposed as a Bayesian framework for word segmentation (Goldwater et al., 2009). When a token t consists of M characters; that is, $t = c_1 ... c_m ... c_M$, its unigram probability $p(t)$ in a text data is given as

$$p(t) = \frac{\text{count}(t) + \alpha p_{\text{base}}(t)}{\sum_{\hat{t}} \text{count}(\hat{t}) + \alpha} \tag{4}$$

where $\text{count}(t)$ is a function that returns the number of tokens t in the text data. $p_{\text{base}}(t)$ gives the basic probability of the token t with a character-level language model:

$$p_{\text{base}}(t : c_1 ... c_M) = p_{\text{uni}}(c_1) \prod_{m=2}^{M} p_{\text{bi}}(c_m | c_{m-1}) \tag{5}$$

To deal with a token that includes an unknown character, both $p_{\text{uni}}(c_m)$ and $p_{\text{bi}}(c_m|c_{m-1})$ are also calculated by a smoothed language model. A smoothed character unigram probability $p_{\text{uni}}(c_m)$ is given as

$$\begin{aligned} p_{\text{uni}}(c_m) &= \frac{\text{count}(c_m) + \beta(\frac{1}{Y})}{Y + \beta} \\ Y &= \sum_{\hat{c}} \text{count}(\hat{c}) \end{aligned} \tag{6}$$

A smoothed character bigram probability $p_{\text{bi}}(c_m|c_{m-1})$ is also given as

$$p_{\text{bi}}(c_m | c_{m-1}) = \frac{\text{count}(c_m | c_{m-1}) + \gamma p_{\text{uni}}(c_m)}{\text{count}(c_{m-1}) + \gamma} \tag{7}$$

where Y is the total number of characters, and $\text{count}(c_m|c_{m-1})$ is the number of character bigrams. $1/Y$ in (6) and $p_{\text{uni}}(c_m)$ in (7) are base probabilities of the character unigram and the character bigram, respectively. α, β, and γ are hyperparameters for smoothing language models. By setting higher values for these hyperparameters, the model associates a higher probability to out-of-vocabulary (OOV) tokens. The result of this association is that the model selects OOV tokens more frequently when sampling.

We use a dictionary-based morphological analyzer or unsupervised word segmentation to tokenize a corpus initially, and the language model is initialized with the tokenized corpus.

3.3 Sampling Tokenization

With the nested unigram language model introduced above, the tokenization of a sentence is sampled from the distribution $P(\boldsymbol{t}|s)$ where $\boldsymbol{t}$ is possible tokenization for the sentence. A probability of tokenization is obtained by a nested language model (4) as $p(\boldsymbol{t}|s) = \prod_{t \in \boldsymbol{t}} p(t)$.

Following (Kudo, 2018) and (Mochihashi et al., 2009), we employ a dynamic programming (DP) technique called forward filtering backward sampling (FFBS) (Scott, 2002) to sample tokens stochastically. With FFBS, we can sample tokens in a sentence from a distribution considering all possible tokenizations within the limit of the maximum token length l. In the forward calculation of FFBS, a DP Table D is calculated as follows:

$$\begin{aligned} \boldsymbol{D}[i][j] &= p(s_{i-j:i}) \sum_{k=1}^{min(i-j,l)} \boldsymbol{D}[i-j][k] \\ \boldsymbol{D}[0][1] &= 1 \end{aligned} \tag{8}$$

where i is the index of a character in a sentence s composed of $c_1 ... c_{i-j} ... c_i ... c_I$, and j is the length

of a token. $s_{i-j:i}$ is a token that consists of $c_{i-j}...c_i$, and $p(s_{i-j:i})$ is given by (4). $\boldsymbol{D}[i][j]$ is the marginalized probability that the token $s_{i-j:i}$ appears in the sentence.

An example of the forward calculation is illustrated in Figure 2. In the figure, the probability of a two-length token that ends with the sixth character is calculated recursively when the maximum length of a word is 3.

After completing Table $\boldsymbol{D}$, we can sample tokenization from the tail of a sentence with $\boldsymbol{D}$. Note that our model uses the whitespaces in the sentence as the token boundaries when processing languages indicating word boundaries such as English.

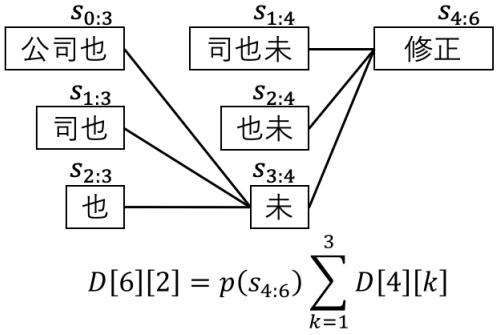

Figure 2: An example of forward calculation of forward filtering backward sampling for a Chinese sentence used in figure 1 with maximum length 3. In this figure, we illustrate the calculation of $\boldsymbol{D}[6][2]$.

3.4 Updating of the Language Model

To update the language model with a tokenized sentence, we follow the updating method of blocked Gibbs sampling for unsupervised word segmentation (Mochihashi et al., 2009). Before sampling tokenization, the token counts of the sentence are removed from the language model (4) and the new tokenization is sampled with the language model. After sampling, the language model is updated by adding the token counts in a currently tokenized sentence.

Specifically, $\mathrm{count}(t)$ in (4) is reduced for every token t included in a sentence. $\mathrm{count}(c)$ is also reduced for all character cs included by t. We handle the adding process in the same way.

By updating the language model, when evaluating the classifier on validation and test datasets, our model can reproduce the segmentation sampled in the training phase. This updating method ensures that the tokenization is consistent between training and evaluation, particularly for a sentence containing a low frequency phrase.

3.5 Embedding for Unfixed Vocabulary

Since our model does not limit the vocabulary, there are many ways to tokenize a single sentence. To use token-level representations, we typically employ a lookup embedding mechanism, which requires a fixed vocabulary. In our model, however, the vocabulary changes as the language model is updated.

We, therefore, introduce word embeddings with continuous cache inspired by (Grave et al., 2016; Kawakami et al., 2017; Cai et al., 2017). This method enables the proposed model to assign token-level representations to recently sampled tokens. Although embeddings of older tokens are discarded from the cache memory, we assume that meaningful tokens to solve the task appear frequently, and they remain in the cache during training if the size of the cache is large enough. By updating representations in the cache, the model can use token-level information adequately.

In our embedding mechanism with a cache component, the model has a list $\boldsymbol{Q}$ that stores $|\boldsymbol{Q}|$ elements of recent tokenization history. The model also keeps a lookup table $\boldsymbol{V}^{\mathrm{cache}}$ composed of token-level vectors corresponding to tokens cached in $\boldsymbol{Q}$. A token t is stored in $\boldsymbol{Q}$, and each element in $\boldsymbol{Q}$ has a unique index q to extract the representation from $\boldsymbol{V}^{\mathrm{cache}}$.

A token-level embedding of the token $\boldsymbol{v}_t^{\mathrm{token}}$ is obtained by extracting a vector $\boldsymbol{v}_q^{\mathrm{cache}}$ from $\boldsymbol{V}^{\mathrm{cache}}$. q is an index corresponding to the token t if t is in the list $\boldsymbol{Q}$; otherwise, the oldest token in $\boldsymbol{Q}$ drops from the list, and we assign its index q to the new token t. The representation for the new token $\boldsymbol{v}_q^{\mathrm{cache}}$ is initialized with $\boldsymbol{v}_t^{\mathrm{char}}$ mentioned in section 2, and the vector for the old token that drops from the list is discarded. This embedding process is described as:

$$\boldsymbol{v}_t^{\mathrm{token}} = \begin{cases} \boldsymbol{V}^{\mathrm{cache}}\boldsymbol{k}_t & (t \in \boldsymbol{Q}) \\ \boldsymbol{v}_t^{\mathrm{char}} & (\mathrm{otherwise}) \end{cases} \quad (9)$$

where $\boldsymbol{k}_t$ is a one-hot vector whose q-th element indicating t is 1 and 0 otherwise.

A token representation obtained by cache-embedding is used as a lookup representation $\boldsymbol{v}^{\mathrm{token}}$ and transformed into a concatenated token vector $\boldsymbol{v}_t$ by (2). The lookup table $\boldsymbol{V}^{\mathrm{cache}}$ is dealt with as a general lookup table, and we update it

with gradients obtained by backward calculation from the loss function. In the evaluation phase, $\boldsymbol{Q}$ is not changed by unknown tokens in the validation and test set.

4 Experiments

4.1 Setup

Dataset: To evaluate the differences caused by tokenization and embedding, we conducted experiments on short-text sentiment classification tasks. We exploited formal and informal corpora in Chinese, Japanese, and English.

The NTCIR-6 dataset (Seki et al., 2007) contains newspaper articles on 32 topics in Chinese, Japanese, and English. We extracted only the sentences attached with three-class sentiment labels. and 1 to 28 topics were used for training, 29 to 30 topics for validation, and 31 to 32 topics for testing.

As a social short-text dataset, we used Twitter datasets in the Japanese[1] and English[2] experiments. These datasets were annotated with five-class sentiment labels in Japanese and two-class sentiment labels in English, and 21,000 sentences were randomly selected in a well-balanced manner. We split the corpus into 18,000 for training, 2,000 for validation, and 1,000 for testing in both Japanese and English.

We used ChnSentiCorp (HOTEL)[3] as another dataset for Chinese sentiment classification on informal texts. We did not use any other resource except for the training datasets.

Model: To compare the results from different tokenization on text classification, we used the simple neural text classifier described in section 2 in all the experiments [4]. As the token vector $\boldsymbol{v}_t$ mentioned in (1), we used representations by different tokenization and embedding.

We initialized randomly and trained token-level and character-level representations with a classification task. The sizes of a token representation and a character representation were set as 512 and 128, respectively, and the size of a sentence representation was 1,024. The sentence representation was projected to a label-size vector depending on the dataset, and probabilities for labels were obtained using the softmax function.

The main results are shown in Table 1. We compared the scores obtained by models trained with different tokenization. In the table, "dictionary" means the model trained with dictionary-based tokenization. Chinese and Japanese datasets are tokenized by Jieba[5] and MeCab, respectively, and the English dataset is tokenized by original whitespaces. As a baseline model that samples tokenization, we employed SentencePiece implementation[6]. We used SentencePiece in both options with/without sampling ("subword" / "subword+samp"). We set the subword size as 6,000 for NTCIR in English and 8,000 for the others[7].

Our model is denoted as "proposed" in the table. "sp" represents the proposed method whose language model is initialized with dictionary-based tokenization, and "unsp" represents the model initialized with unsupervised word segmentation. The sizes of the cache for the proposed model were the same as the sizes of the subword vocabulary for SentencePiece. We set the maximum length of the token for our method as eight for every language. When initializing the language model with a dictionary-based tokenization, the corpus was retokenized into tokens shorter than eight characters depending on the language model. The hyperparameters for smoothing were set as $\alpha = \beta = \gamma = 1$ in both pretraining for unsupervised word segmentation and training for classification.

Dropout layers were used for embedding and sentence representations with a rate of 0.5. We used the softmax cross-entropy loss for optimization, and the parameters were optimized by Adam (Kingma and Ba, 2014). We trained the models in 30 epochs, and the model with the highest score on the validation dataset was selected and evaluated on the test dataset. In this paper, we report the average F1 score in five experiments.

[1] http://bigdata.naist.jp/~ysuzuki/data/twitter/

[2] https://www.kaggle.com/c/twitter-sentiment-analysis2

[3] http://tjzhifei.github.io/resource.html

[4] We conducted the same experiment with Deep Average Network (DAN) (Iyyer et al., 2015) rather than LSTM and obtained similar results. We report the experiment with the LSTM classifier because the results are more significant than the results with DAN.

[5] https://github.com/fxsjy/jieba

[6] https://github.com/google/sentencepiece

[7] Although we should have set the subword size for the English NTCIR as 8,000 as well as the other datasets, we had to use 6,000 because the English dataset was too small to make more than 8,000 subwords with SentencePiece.

	Chinese		Japanese		English	
	NTCIR	HOTEL	NTCIR	TWITTER	NTCIR	TWITTER
dictionary	50.21	85.28	55.54	65.00	49.52	71.40
subword	50.95	86.45	52.87	66.25	52.19	72.65
subword+samp	**51.32**	**87.61**	51.36	66.25	53.90	73.15
proposed(sp)	50.91	86.62	**58.27**	66.50	**56.73**	73.66
proposed(unsp)	49.54	87.29	53.07	**67.75**	54.09	**74.80**

Table 1: F1 scores (%) from the models trained with different methods of tokenization. The highest scores among all methods are highlighted by bold font, and the highest scores among unsupervised tokenization models are highlighted with an underline.

4.2 Results

First, we analyzed the overall results of the experiment. The highest scores among all tokenization methods are highlighted by bold font in Table 1. As shown in the table, the proposed method obtained the best scores in the Japanese and English datasets. SentencePiece with a sampling option, however, scored the highest in the Chinese datasets. This is because the Chinese vocabulary is larger than the Japanese and English vocabularies. In other words, Chinese datasets have a larger number of types of n-grams. We consider that the cache size of the proposed method is not sufficient to store meaningful words to solve the task of the Chinese dataset.

Second, we focus on the results by the supervised methods ("dictionary" and "proposed(sp)"). The language model of the proposed method "sp" is initialized by corpus segmented by the "dictionary" method and trained by sampled tokenization while training a classifier. The table shows that the scores from our method surpassed the dictionary-based segmentation for all datasets. We conclude that the proposed method is superior to the method that trains a classifier with dictionary-based tokenization.

Third, we analyzed the scores obtained using unsupervised methods ("subword", "subword+samp", and "proposed(unsp)"). The highest scores among the unsupervised methods are emphasized by an underline. The proposed method obtained the best scores for the Japanese and English datasets, but SentencePiece was superior for the Chinese dataset as described in the overall comparison.

Finally, we compare the proposed methods. The proposed model whose language model is initialized by a dictionary ("sp") obtained higher scores on the NTCIR dataset in every language. On the other hand, the model with unsupervised initialization scored higher on SNS dataset for all languages. From these results, we conclude that the performance of our model improved with dictionary-based segmentation for formal corpus while unsupervised initialization improved the performance of informal corpus when a generally used dictionary was employed.

5 Discussion

5.1 Cache Size

In the main experiment described in the previous section, we set the size of the cache for token-level embedding to be the same as the vocabulary of SentencePiece for a fair comparison. As explained, the scores of our model for the Chinese dataset were lower than the scores for SentencePiece with sampling. We consider that this result was caused by the cache-overflow of the vocabulary. Therefore, we conducted an additional experiment where the size of the cache was increased. The results are shown in table 2. The cache size of the model denoted as "x2" is twice the size (16,000) of the model used in Table 1. From the result, we conclude that increasing the size of the cache improves the performance of the proposed model for the Chinese datasets. We also determine that the size of the cache used in the main experiment is sufficient to store meaningful words for the task in Japanese and English.

Figure 3 shows the performances of different cache sizes on two Chinese datasets, and Table 3 shows the vocabulary sizes of the language models at the beginning of a classifier training on each dataset. From the result of the experiment on the Chinese dataset, we conclude that increasing the cache size improves performance. We also conclude that we can use the size of vocabulary of the initial language model as an indicator to set the cache size. In the figure, the performance in-

	Chinese	
	NTCIR	HOTEL
subword+samp	51.32	87.61
proposed(sp)	50.91	86.62
proposed(sp)x2	**51.45**	87.45
proposed(unsp)	49.54	87.29
proposed(unsp)x2	51.32	**88.29**

Table 2: F1 scores (%) by models with different cache size for the proposed model on Chinese datasets. The size of the cache of proposed models when "+x2" is 16,000 and 8,000 otherwise. The best model in table 1 "subword+samp" is quoted as a baseline model.

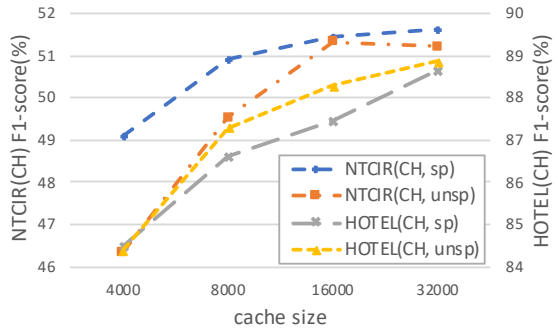

Figure 3: F1 scores (%) by models trained with different cache sizes on NTCIR(CH) and HOTEL(CH).

Lanugage	Dataset	sp	unsp
Chinese	NTCIR	29,299	28,232
	HOTEL	22,170	15,816
Japanese	NTCIR	7,623	10,139
	Twitter	33,831	17,013
English	NTCIR	11,449	3,544
	Twitter	50,733	6,859

Table 3: The volume of vocabulary of the language model initialized by dictionary (sp) and unsupervised word segmentation (unsp) for each dataset.

creases up to the cache size around the size of the initial vocabulary. In addition, we consider from the vocabulary sizes of the Japanese and English Twitter dataset that it is important to select an appropriate tokenizer for initialization. Although the initial vocabulary is huge for the Japanese and English datasets, the cache size is sufficient to store the useful words for classification. We consider the reason is that there are many similar but different words in the vast vocabulary of the Twitter dataset unlike the Chinese dataset, and the difference becomes small using the language model.

5.2 Sampling Option

Our model has two other options for sampling tokenization: a model without sampling in the training phase ("nosamp") and a model that samples tokenization without updating the language model ("samp"). The former means that the model tokenizes a sentence into the one-best tokenization with an initialized unigram language model while the latter can be described as a vocabulary-free version of SentencePiece. We tested this comparison on the models with dictionary-based initialization ("sp") and the 500-epoch pretrained models ("unsp"). Table 4 shows the results. Our proposed model updating a language model is denoted as "train."

The results show that higher scores are given by updating the language model ("train") on all the datasets. While we cannot determine comprehensively whether performance is improved by sampling without updating a language model ("samp"), from the results, we argue that the performance of the classification task is improved by sampling tokenization and updating its language model.

5.3 Case Study

Figure 4 shows distributions for each label for different tokenizations for a sentence in a validation set of a Japanese Twitter dataset. Each distribution is calculated by the same model that samples tokenization and updates its language model. In the figure, "INITIAL" means a prediction by a model inputted into a sentence tokenized by an initial language model. In other words, "INITIAL" shows the prediction by the model without updating the language model.

As shown in the figure, the model predicts different labels for each tokenization. The model feeding tokenization by an updated language model predicts a true label while the model with tokenization by the initial language model predicts a wrong label with a higher probability. In this example, the difference of tokenization on "電源ボタン" and "ほしかった" has A significant effect on the prediction. Although this example was remarkable, there were many sentences where the model predicted different labels by its tokenization.

	Chinese		Japanese		English	
	NTCIR	HOTEL	NTCIR	TWITTER	NTCIR	TWITTER
sp+nosamp	50.36	86.28	53.96	66.25	52.85	72.80
sp+samp	50.27	85.28	51.98	66.25	53.33	72.40
sp+train	**50.91**	**86.62**	**58.27**	**66.50**	**56.73**	**73.66**
unsp+nosamp	49.08	86.95	52.80	65.37	53.80	73.90
unsp+samp	48.95	85.95	48.35	66.37	53.80	73.60
unsp+train	**49.54**	**87.29**	**53.07**	**67.75**	**54.09**	**74.80**

Table 4: F1 scores (%) of the ablation study to compare the sampling options of the proposed model. "samp" represents a model that samples tokenization without updating its language model while "train" updates its language model depending on the sampled tokenization.

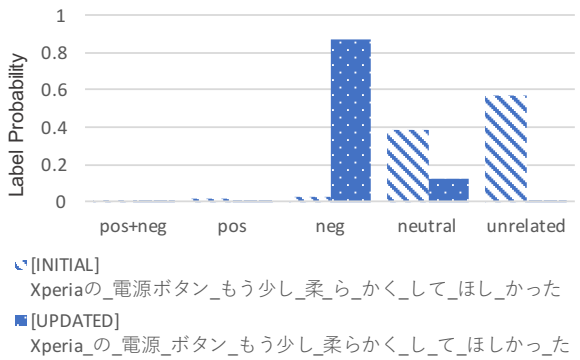

Figure 4: Label prediction by a trained model for a sentence with different tokenization. "_" denotes word boundary, and the sentence means "I wish they had made the button on the Xperia more responsive." True label for the sentence is "neg". The sentence indicated by "UPDATED" is tokenized by a language model updated with sampled tokenization while "INITIAL" is segmented by an initial language model.

5.4 News-title Classification

In addition to sentiment analysis, we also evaluated our model on other domains of text classification, topic classification. We employed Japanese web news corpus provided by Livedoor[8] and a model classified article titles into a label of nine topics. The experiment was conducted under the same condition as the sentiment analysis described in section 4.

As shown in Table 5, the proposed method with unsupervised initialization obtained the highest score. In addition to the result of sentiment analysis, we also determined that the performance improved by initializing the language model with dictionary-based tokenization. From the result, we conclude that our new tokenization strategy is effective on some classification tasks.

[8] https://www.rondhuit.com/download.html#ldcc

	F1-score
dictionary	80.31
subword	80.41
subword+samp	78.95
proposed(sp)	**81.71**
proposed(unsp)	80.46

Table 5: F1-scores (%) using the models with different tokenizations on the news-title classification task (Japanese).

6 Related Work

Our work is related to word segmentation for a neural network encoder. To tokenize a sentence into subwords without dictionary-based segmentation, BPE is commonly used in neural machine translation (NMT) (Sennrich et al., 2016). BPE forces a merger of tokens without any exceptions, and tokenization does not become natural.

The problem associate with BPE has been addressed using a language model to tokenize a sentence. (Goldwater et al., 2006, 2009) proposed unsupervised word segmentation by sampling tokenization and updating a language model with Gibbs sampling. The language model for unsupervised word segmentation is smoothed with base probabilities of words to give a probability for all possible words in a text. (Mochihashi et al., 2009) extended this to the use of blocked Gibbs sampling, which samples tokenization by a sentence. The authors introduced a nested Bayesian language model that calculates a probability of a word by hierarchical language models.

Recently, (Kudo and Richardson, 2018) proposed a subword generator for NMT, which tokenizes a sentence stochastically with a subword-level language model while (Kudo, 2018) reports improvement in performance of NMT by the idea

of sampling tokenization. Considering multiple subwords makes an NMT model robust against noise and segmentation errors. This differs from BPE in that it does not merge tokens uniquely by its frequency and differs from unsupervised word segmentation with a language model in that it limits its subword vocabulary. Our work is similar to this line of research, but we focus on NLP tasks that do not require decoding such as text classification. The proposed model is different from this work in some respects: the vocabulary is not fixed, and the language model is updated by sampled tokenization.

In this paper, we address the problem of tokenization for a neural network encoder by modifying a tokenization strategy. Another approach to address this problem alters the architecture of a neural network. For example, (Zhang and Yang, 2018) employs lattice LSTM, which considers all possible tokenizations of a sentence for named entity recognition. Lattice structured RNNs are also used for neural Chinese word segmentation such as (Chen et al., 2017) and (Yang et al., 2018), and they report improvement in performance. Our work is different from these works from the perspective that we address the problem focusing on the segmentation itself not the architecture of a neural network as well as (Kudo, 2018).

We used a caching mechanism proposed to augment neural language models (Merity et al., 2016; Grave et al., 2016). This is also exploited for an open-vocabulary language model (Kawakami et al., 2017). (Cai et al., 2017) proposed a similar architecture to the caching mechanism for neural Chinese word segmentation.

7 Conclusion

In this paper, we introduced stochastic tokenization for text classification with a neural network. Our model differs from previous methods in terms of sampling tokenization that considers all possible words under the maximum length limitation. To embed various tokens, we proposed the cache mechanism for frequent words. Our model also updates the language model depending on the sampled tokenizations in the training phase. With the updated language model, the proposed model can tokenize the test dataset considering recently used tokenization in the training phase. This results in improved performance for sentiment analysis tasks on Japanese and English datasets and Chinese datasets with a larger cache. We find that the proposed model of tokenization provides an improvement in the performance of text classification with a simple LSTM classifier. We expect our model contributes to improved performance of other complex state-of-the-art encoding architectures for text classification.

Acknowledgments

We are grateful to the members of the Computational Linguistics Laboratory, NAIST and the anonymous reviewers for their insightful comments.

References

Deng Cai, Hai Zhao, Zhisong Zhang, Yuan Xin, Yongjian Wu, and Feiyue Huang. 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)*, volume 2, pages 608–615.

Xinchi Chen, Zhan Shi, Xipeng Qiu, and Xuanjing Huang. 2017. Dag-based long short-term memory for neural word segmentation. *arXiv preprint arXiv:1707.00248*.

Sharon Goldwater, Thomas L Griffiths, and Mark Johnson. 2006. Contextual dependencies in unsupervised word segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 673–680. Association for Computational Linguistics.

Sharon Goldwater, Thomas L Griffiths, and Mark Johnson. 2009. A bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112(1):21–54.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2016. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. 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)*, volume 1, pages 1681–1691.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. volume 2, pages 427–431.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for

modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 655–665.

Kazuya Kawakami, Chris Dyer, and Phil Blunsom. 2017. Learning to create and reuse words in open-vocabulary neural language modeling. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1492–1502.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Taku Kudo. 2006. Mecab: Yet another part-of-speech and morphological analyzer. http://taku910.github.io/mecab/.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75.

Taku Kudo and John Richardson. 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*, pages 66–71.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*.

Chenwei Liu, Jiawei Wang, and Kai Lei. 2016. Detecting spam comments posted in micro-blogs using the self-extensible spam dictionary. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE.

Lin Liu and Kun Jia. 2012. Detecting spam in chinese microblogs-a study on sina weibo. In *2012 Eighth International Conference on Computational Intelligence and Security*, pages 578–581. IEEE.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Daichi Mochihashi, Takeshi Yamada, and Naonori Ueda. 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: Volume 1-Volume 1*, pages 100–108. Association for Computational Linguistics.

Cicero dos Santos and Maira Gatti. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78.

Steven L Scott. 2002. Bayesian methods for hidden markov models: Recursive computing in the 21st century. *Journal of the American Statistical Association*, 97(457):337–351.

Yohei Seki, David Kirk Evans, Lun-Wei Ku, Hsin-Hsi Chen, Noriko Kando, and Chin-Yew Lin. 2007. Overview of opinion analysis pilot task at ntcir-6.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages P1715–1725.

Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. 2014. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1555–1565.

Jin Wang, Liang-Chih Yu, K Robert Lai, and Xuejie Zhang. 2016a. Dimensional sentiment analysis using a regional cnn-lstm model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 225–230.

Yequan Wang, Minlie Huang, Li Zhao, et al. 2016b. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 606–615.

Jie Yang, Yue Zhang, and Shuailong Liang. 2018. Subword encoding in lattice lstm for chinese word segmentation. *arXiv preprint arXiv:1810.12594*.

Hua-Ping Zhang, Hong-Kui Yu, De-Yi Xiong, and Qun Liu. 2003. Hhmm-based chinese lexical analyzer ictclas. In *Proceedings of the second SIGHAN workshop on Chinese language processing-Volume 17*, pages 184–187. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

Yue Zhang and Jie Yang. 2018. Chinese ner using lattice lstm. *arXiv preprint arXiv:1805.02023*.

Yujun Zhou, Bo Xu, Jiaming Xu, Lei Yang, and Changliang Li. 2016. Compositional recurrent neural networks for chinese short text classification. In *Web Intelligence (WI), 2016 IEEE/WIC/ACM International Conference on*, pages 137–144. IEEE.