Subword Encoding in Lattice LSTM for Chinese Word Segmentation

Jie Yang, Yue Zhang, Shuailong Liang
Singapore University of Technology and Design
jieynlp@gmail.com, yue.zhang@wias.org.cn, shuailong_liang@mymail.sutd.edu.sg

Abstract

We investigate a lattice LSTM network for Chinese word segmentation (CWS) to utilize words or subwords. It integrates the character sequence features with all subsequences information matched from a lexicon. The matched subsequences serve as information shortcut tunnels which link their start and end characters directly. Gated units are used to control the contribution of multiple input links. Through formula derivation and comparison, we show that the lattice LSTM is an extension of the standard LSTM with the ability to take multiple inputs. Previous lattice LSTM model takes word embeddings as the lexicon input, we prove that subword encoding can give the comparable performance and has the benefit of not relying on any external segmentor. The contribution of lattice LSTM comes from both lexicon and pretrained embeddings information, we find that the lexicon information contributes more than the pretrained embeddings information through controlled experiments. Our experiments show that the lattice structure with subword encoding gives competitive or better results with previous state-of-the-art methods on four segmentation benchmarks. Detailed analyses are conducted to compare the performance of word encoding and subword encoding in lattice LSTM. We also investigate the performance of lattice LSTM structure under different circumstances and when this model works or fails.

Introduction

Different from the English-like languages whose words are separated naturally, it is necessary to segment character sequence as word sequence in many East Asian languages, such as Chinese. Chinese word segmentation (CWS) has been thoughtfully studied from statistical methods to recent deep learning approaches. Most of them formalize CWS as a sequence labeling problem (Xue and others 2003).

Neural network based Chinese word segmentation has attracted significant research attention due to its ability of non-linear feature representation and combination. Typical neural CWS models utilize the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) or Convolution Neural Network (CNN) (LeCun et al. 1989) as feature extractor, and take the character unigram and bigram embeddings as inputs (Pei, Ge, and Chang 2014; Yang, Zhang, and Dong 2017). Those models already achieve state-of-the-art performance on many CWS benchmarks (Zhou et al. 2017; Wang and Xu 2017; Yang, Zhang, and Dong 2017).

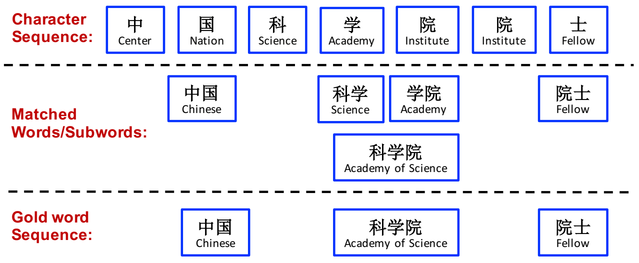

Figure 1: Segmentation with ambiguous words.

It has been shown that word information is beneficial to word segmentation (Zhang and Clark 2007; Zhang, Zhang, and Fu 2016; Cai and Zhao 2016). Zhang and Clark (2007) built a transition-based word segmentor by utilizing handcrafted word features, Zhang, Zhang, and Fu (2016) and Cai and Zhao (2016) extended the transition framework as neural models which utilize the word embeddings.

One limitation of the above word-based models, however, is that word information can be utilized only for readily recognized words, namely those that are already in the output candidates. However, ambiguous words in a context can provide additional information for disambiguation. For instance, in Figure 1, the word "科学院(Academy of Sciences)" and "学院(Academy)" can be useful for determining the correct segmentation, which is "科学院/(Academy of Sciences/)", despite that "学院(Academy)" is not in the correct output.

Zhang and Yang (2018) proposed a lattice LSTM structure which can utilize the ambiguous words information in the named entity recognition (NER) task. The lattice structure is based on character LSTM sequence but leverages word information by using extra "shortcut paths" to link the memory cell between the start and the end characters of the word. To control the contribution of each "shortcut path", gated recurrent unit is used in each path. The final memory cell of character is the weighted sum of all the "shortcut paths". The "shortcut paths" are constructed by directly matching the sentence with a word lexicon, while the word lexicon comes from auto-segmented text. In this way, the lattice LSTM based NER system requires the segmentor information, although in an indirect way.

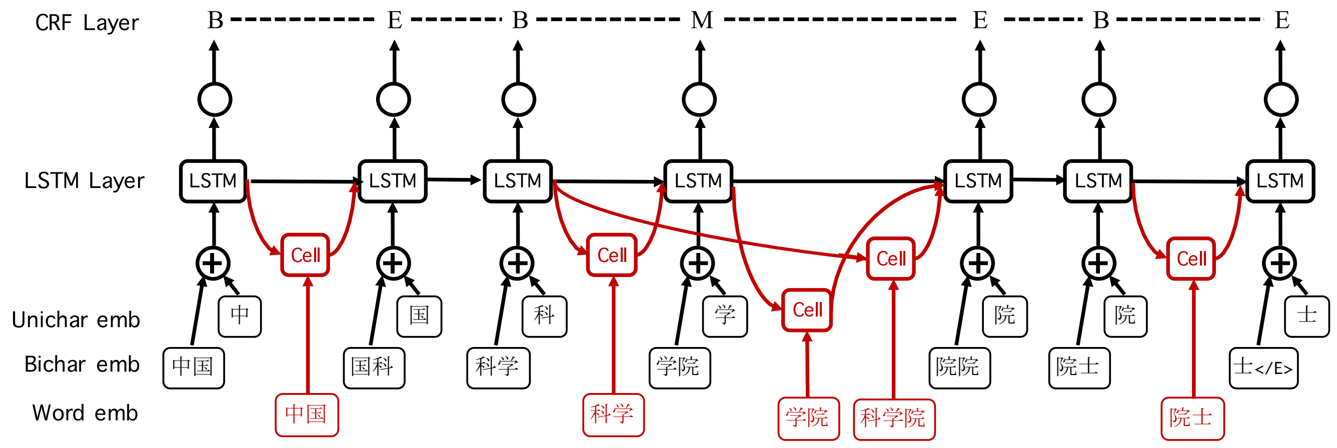

Figure 2: Models. Only forward LSTM is illustrated here.

In this work, we extend the lattice LSTM structure in Zhang and Yang (2018) by using subword encoding which does not rely on any external segmentor[1]. We further examine the lattice LSTM which utilizes ambiguous words or subwords information on CWS tasks. Different from Zhang and Yang (2018) which takes the word embeddings as the lattice input, we additionally examine the Byte Pair Encoding (BPE) (Gage 1994) algorithm to encode the subword and construct the lattice LSTM with subword embeddings. The construction of subword embeddings does not rely on large segmented text which is necessary when building word embeddings.

To our knowledge, we are the first to use the BPE for word segmentation. The comparison between word and subword embeddings are thoroughly studied. The contributions of word/subword lexicon and their pretrained embeddings are also investigated through controlled experiments. Experiments on four benchmarks show that the subword encoding in lattice LSTM can give comparable results with word embeddings, and they both can achieve state-of-the-art segmentation performance. In the end, we analyze two segmentation examples which show failed case for word encoding for lattice LSTM ("Lattice+Word") and subword encoding for lattice LSTM ("Lattice+Subword"), respectively.

Related Work

Statistical word segmentation has been studied for decades (Sproat et al. 1996). State-of-the-art models are either using the sequence labeling methods e.g. CRF (Lafferty, McCallum, and Pereira 2001) with character features (Peng, Feng, and McCallum 2004; Zhao et al. 2006) or taking the transition-based models with word features (Zhang and Clark 2007; Sun 2010).

Similarly, neural word segmentors replace the handcrafted features with neural representations. Chen et al. (2015a) and Chen et al. (2015b) build neural CRF segmentors with GRU and LSTM to extract the representation on character embeddings, respectively. Cai and Zhao (2016) and Zhang, Zhang, and Fu (2016) directly score the word sequences with beam search by utilizing both word information and character embeddings, both show comparable results with neural CRFs. Yang, Zhang, and Dong (2017) extend the word-based neural segmentor through pretraining the character representations with multi-task training on four external tasks, and observe significant improvement. Zhou et al. (2017) improve the segmentor with better character embeddings which include pre-segmented word context information through a new embedding training method on a large auto-segmented corpus.

Lattice RNNs have been used to model speech tokenization lattice (Sperber et al. 2017) and multi-granularity segmentation for NMT (Su et al. 2017). Zhang and Yang (2018) proposed a lattice LSTM for Chinese NER. It integrates the character sequence features and all matched word embeddings into a sequence labeling model, leading to a powerful NER system. Zhu, Sobhani, and Guo (2016) proposed a DAG-structured LSTM structure which is similar to the lattice LSTM model, the DAG-LSTM binarizes the paths in the merging process but lattice LSTM merges the paths using gate controlled summation. Chen et al. (2017) also built a DAG-LSTM structure for word segmentation. Different from their model which uses no memory cell in the word path, our lattice LSTM assigns one memory cell for each word path and merges them in the end character of the word. In addition, our model consistently gives better performance.

BPE is a data compression algorithm which iteratively merges the most frequent pair of bytes in a sequence as a new byte. In this work, we use BPE algorithm to merge characters rather than bytes in the text corpus, constructing the subwords which represent the most frequent character compositions in corpus level. It has been successfully used in neural machine translation by capturing the most frequent subwords instead of words (Sennrich, Haddow, and Birch 2016).

[1]Our code is released at `https://github.com/jiesutd/SubwordEncoding-CWS`.

Models

We take the state-of-the-art LSTM-CRF framework as our baseline. For an input sentence with m characters $s = c_1, c_2, \dots, c_m$, where c_i denotes the ith character, the segmentor is to assign each character c_i with a label l_i, where $l_i \in \{B, M, E, S\}$ (Xue and others 2003). The label B, M, E and S represent the *begin, middle, end* of a word and single character word, respectively. Figure 2 shows the segmentor framework on input character sequence "中国科学院院士 (Fellow of the Chinese Academy of Sciences)", where the black part represents the baseline LSTM-CRF model and the red part shows the lattice structure.

Embedding Layer

As shown in Figure 2, for each input character c_i, the corresponding character unigram embeddings and character bigram embeddings are represented as $\mathbf{e}_{c_i}$ and $\mathbf{e}_{c_i c_{i+1}}$, respectively. The character representation is calculated as following:

$$\mathbf{x}_i = \mathbf{e}_{c_i} \oplus \mathbf{e}_{c_i c_{i+1}}, \tag{1}$$

where $\oplus$ represents *concatenate* operation.

Unlike Zhang, Zhang, and Fu (2016) which uses a window to strengthen the local features, or Zhou et al. (2017) which adds a non-linear layer before the LSTM layer, we feed the character representation $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m)$ into a bidirectional LSTM directly.

Baseline LSTM Layer

LSTM (Hochreiter and Schmidhuber 1997) is an advanced recurrent neural network (RNN) with extra memory cells which are used to keep the long-term information and alleviate the gradient vanishing problem. Equation 2 shows the calculation of $\overrightarrow{\mathbf{h}}_i$ which is the forward LSTM representation of character c_i .

$$\begin{aligned}
\begin{bmatrix} \mathbf{o}_i \\ \mathbf{f}_i \\ \widetilde{\mathbf{c}}_i \end{bmatrix} &= \begin{bmatrix} \sigma \\ \sigma \\ tanh \end{bmatrix} \Big(\mathbf{W}^\top \begin{bmatrix} \mathbf{x}_i \\ \overrightarrow{\mathbf{h}}_{i-1} \end{bmatrix} + \mathbf{b} \Big) \\
\mathbf{i}_i &= \mathbf{1} - \mathbf{f}_i \\
\mathbf{c}_i &= \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \widetilde{\mathbf{c}}_i \\
\overrightarrow{\mathbf{h}}_i &= \mathbf{o}_i \odot tanh(\mathbf{c}_i)
\end{aligned} \tag{2}$$

where $\mathbf{i}_i, \mathbf{f}_i$ and $\mathbf{o}_i$ denote a set of input, forget and output gates, respectively. We choose the coupled LSTM structure (Greff et al. 2017) which sets the input gate $\mathbf{i}_i = \mathbf{1} - \mathbf{f}_i$. $\mathbf{c}_i$ is the memory cell of character c_i. $\mathbf{W}^\top$ and $\mathbf{b}$ are model parameters. $\sigma()$ represents the sigmoid function.

For each input sentence $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m)$, we calculate both the forward and backward LSTM representation as follows:

$$\begin{aligned}
\overrightarrow{\mathbf{h}}_1, \overrightarrow{\mathbf{h}}_2, \dots, \overrightarrow{\mathbf{h}}_m &= \overrightarrow{LSTM}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m) \\
\overleftarrow{\mathbf{h}}_1, \overleftarrow{\mathbf{h}}_2, \dots, \overleftarrow{\mathbf{h}}_m &= \overleftarrow{LSTM}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m),
\end{aligned} \tag{3}$$

where $\overrightarrow{LSTM}$ and $\overleftarrow{LSTM}$ represent the forward and backward LSTM, respectively. To incoperate the information from both sides, the hidden vector of character c_i is the concatenation of the representations in both directions:

$$\mathbf{h}_i = \overrightarrow{\mathbf{h}}_i \oplus \overleftarrow{\mathbf{h}}_i \tag{4}$$

A CRF layer (Eq. 9) is used on top of the hidden vectors $(\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_m)$ to perform label prediction.

Lattice LSTM Layer

The lattice LSTM adds "shortcut paths" (red part in Figure 2) to LSTM. The input of the lattice LSTM model is character sequence and all subsequences which are matched words in a lexicon $\mathbb{D}$. $\mathbb{D}$ is collected from auto-segmented Gigaword corpus or BPE encoding. Following Zhang and Yang (2018), we use $w_{b,e}$ to represent the subsequence that has a start character index b and a end character index e, and the embeddings of the subsequence is represented as $\mathbf{e}_{w_{b,e}}$.

During the forward lattice LSTM calculation, the "cell" in Figure 2 of a subsequence $w_{b,e}$ takes the hidden vector of the start character $\overrightarrow{\mathbf{h}}_b$ and the subsequence (word or subword) embeddings $\mathbf{e}_{w_{b,e}}$ as input, an extra LSTMcell (without output gate) is applied to calculate the memory vector of the sequence $\mathbf{c}_{w_{b,e}}$:

$$\begin{aligned}
\begin{bmatrix} \mathbf{i}_{b,e} \\ \mathbf{f}_{b,e} \\ \widetilde{\mathbf{c}}_{b,e} \end{bmatrix} &= \begin{bmatrix} \sigma \\ \sigma \\ tanh \end{bmatrix} \Big(\mathbf{W}_s^\top \begin{bmatrix} \mathbf{e}_{w_{b,e}} \\ \overrightarrow{\mathbf{h}}_b \end{bmatrix} + \mathbf{b}_s \Big) \\
\mathbf{c}_{b,e} &= \mathbf{f}_{b,e} \odot \mathbf{c}_b^c + \mathbf{i}_{b,e} \odot \widetilde{\mathbf{c}}_{b,e}
\end{aligned} \tag{5}$$

where $\mathbf{c}_{b,e}$ is the memory cell of the shortcut path starting from character c_b to character c_e. $\mathbf{W}_s^\top$ and $\mathbf{b}_s$ are model parameters of the shortcut path LSTM. Different from the standard LSTMcell which calculates both memory and output vectors, we calculate only the memory cell of the shortcut path.

The subsequence output memory vector $\mathbf{c}_{b,i}$ links to the end character c_i as the input to calculate the hidden vector $\overrightarrow{\mathbf{h}}_i$ of c_i. For character c_i with multiple subsequence memory cell inputs[2], we define the input set as $\mathbb{C}_i = \{\mathbf{c}_{b,i} | b \in \{b' | w_{b',i} \in \mathbb{D}\}\}$, we assign a unique gate for each subsequence input to control its contribution:

$$\mathbf{i}_{b,i} = \sigma\big(\mathbf{W}^{g\top} \begin{bmatrix} \mathbf{x}_i \\ \mathbf{c}_{b,i} \end{bmatrix} + \mathbf{b}^g\big) \tag{6}$$

where $\mathbf{W}^{g\top}$ and $\mathbf{b}^g$ are model parameters for the gate.

Until now, we have calculated the subsequence memory inputs $\mathbb{C}_i$ and their control gates $\mathbb{I}_i = \{\mathbf{i}_{b,i} | b \in \{b' | w_{b',i} \in \mathbb{D}\}\}$. Following the idea of coupled LSTM (Greff et al. 2017) which keeps the sum of input and forget gate as $\mathbf{1}$, we normorize all the subsequence gates $\mathbb{I}_i$ with the standard

[2]e.g. The first "院(College)" in Figure 2 takes two subsequence memory vectors of both "学院(Academy)" and "科学院(Academy of Sciences)" as input.

Dataset	Type	Train	Dev	Test
CTB6	Sentence	23.4k	2.08k	2.80k
	Word	641k	59.9k	81.6k
	Char	1.06m	100k	134k
PKU	Sentence	17.2k	1.91k	1.95k
	Word	1.01m	99.9k	104k
	Char	1.66m	164k	173k
MSR	Sentence	78.2k	8.69k	3.99k
	Word	2.12m	247k	107k
	Char	3.63m	417k	184k
Weibo	Sentence	20.1k	2.05k	8.59k
	Word	421k	43.7k	188k
	Char	689k	73.2k	316k

Table 1: Statistics of datasets.

LSTM input gate $\mathbf{i}_i$ to ensure their sum equals to $\mathbf{1}$ (Eq. 7).

$$\begin{aligned}\boldsymbol{\alpha}_{b,i} &= \frac{exp(\mathbf{i}_{b,i})}{exp(\mathbf{i}_i) + \sum\limits_{\mathbf{i}_{b',i} \in \mathbb{I}_i} exp(\mathbf{i}_{b',i})} \\ \boldsymbol{\alpha}_{i} &= \frac{exp(\mathbf{i}_{i})}{exp(\mathbf{i}_i) + \sum\limits_{\mathbf{i}_{b',i} \in \mathbb{I}_i} exp(\mathbf{i}_{b',i})}\end{aligned} \tag{7}$$

$\boldsymbol{\alpha}_{b,i}$ and $\boldsymbol{\alpha}_i$ are the subsequence memory gate and the standard LSTM input gate after the normalization, respectively. The final forward lattice LSTM representation $\overrightarrow{\mathbf{h}}_i$ of character c_i is calculated as:

$$\begin{aligned}\begin{bmatrix}\mathbf{o}_i\\ \mathbf{f}_i\\ \widetilde{\boldsymbol{c}}_i\end{bmatrix} &= \begin{bmatrix}\sigma\\ \sigma\\ tanh\end{bmatrix}\Big(\mathbf{W}^\top\begin{bmatrix}\mathbf{x}_i\\ \overrightarrow{\mathbf{h}}_{i-1}\end{bmatrix}+\mathbf{b}\Big)\\ \mathbf{i}_i &= \mathbf{1} - \mathbf{f}_i\\ \mathbf{c}_i &= \sum_{\boldsymbol{c}_{b,i}\in\mathbb{C}_i} \boldsymbol{\alpha}_{b,i}\odot\boldsymbol{c}_{b,i} + \boldsymbol{\alpha}_i\odot\widetilde{\boldsymbol{c}}_i\\ \overrightarrow{\mathbf{h}}_i &= \mathbf{o}_i\odot tanh(\mathbf{c}_i)\end{aligned} \tag{8}$$

where $\mathbf{W}^\top$ and $\mathbf{b}$ are the model parameters which are the same with the standard LSTM in Eq. 2. Compare with Eq. 2, Eq. 8 has a more complex memory calculation step which integrates both the standard character LSTM memory $\widetilde{\boldsymbol{c}}_i$ and all the matched subsequence memory inputs $\mathbb{C}_i$. In this respect, we can regard the lattice LSTM as an extension of the standard LSTM with the ability of taking multiple inputs.

The backward lattice LSTM representation $\overleftarrow{\mathbf{h}}_i$ has a symmetrical calculation process as above. To give a fair comparison with the baseline bi-directional LSTM structure, we use the bi-directional lattice LSTM whose final hidden vector $\mathbf{h}_i$ is the concatenation of the hidden vectors on both lattice LSTM directions. The same CRF layer as the baseline is used on top of the lattice LSTM layer.

Decoding and Training

A standard CRF layer is used. The probability of a label sequence $y = l_1, l_2, \dots, l_m$ is

Parameter	Value	Parameter	Value
char emb size	50	bigram emb size	50
word emb size	50	subword emb size	50
char dropout	0.5	lattice dropout	0.5
LSTM layer	1	LSTM hidden	200
learning rate *lr*	0.01	*lr* decay	0.05

Table 2: Hyper-parameter values.

$$P(y|s) = \frac{exp(\sum\limits_{i=1}^{m}(F(l_i) + L(l_{i-1}, l_i)))}{\sum\limits_{y'\in\mathbb{C}(s)} exp(\sum\limits_{i=1}^{m}(F(l'_i) + L(l'_{i-1}, l'_i)))}, \tag{9}$$

where $\mathbb{C}(s)$ is the set of all possible label sequences on sentence s and y' is an arbitary label sequence. $F(l_i) = \mathbf{W}^{l_i}\mathbf{h}_i + b^{l_i}$ is the emission score from hidden vector $\mathbf{h}_i$ to label l_i. $L(l_{i-1}, l_i)$ is the transition score from l_{i-1} to l_i. $\mathbf{W}^{l_i}$ and b^{l_i} are model parameters specific to label l_i.

Viterbi algorithm (Viterbi 1967) is used to decode the highest scored label sequence over the input sequence. During training, we choose sentence-level log-likelihood as the loss function.

$$Loss = \sum_{i=1}^{N} log(P(y_i|s_i)), \tag{10}$$

where y_i is the gold labels of sentence s_i.

Experiments

Experimental Settings

Data. We take the Chinese Treebank 6.0 (CTB6) (Xue et al. 2005) as our main dataset and split the train/dev/test following Zhang, Zhang, and Fu (2016). We also evaluate our model on another three standard Chinese word segmentation datasets: PKU, MSR, and Weibo. PKU and MSR are taken from the SIGHAN 2005 bake-off (Emerson 2005) with standard data split. Different from the CTB6/PKU/MSR which are mainly based on formal news text, Weibo dataset is collected from informal social media in the NLPCC 2016 shared task (Qiu, Qian, and Shi 2016), the standard split is used. Table 1 shows the details of the four investigated datasets.

Hyperparameters. Table 2 shows the chosen hyperparameters in our model. We do not tune the hyperparameters based on each dataset but keep them same among all datasets. Standard gradient descent (SGD) with a learning rate decay is used as the optimizer. The embedding sizes of character unigram/bigram and word/subword are all in 50 dimensions. Dropout (Srivastava et al. 2014) is used on both the character input and the word/subword input to prevent overfitting.

Embeddings. We take the same character unigram embeddings, bigram embeddings and word embeddings with Zhang, Zhang, and Fu (2016), which pretrain those embeddings using word2vec (Mikolov et al. 2013) on Chinese Gi-

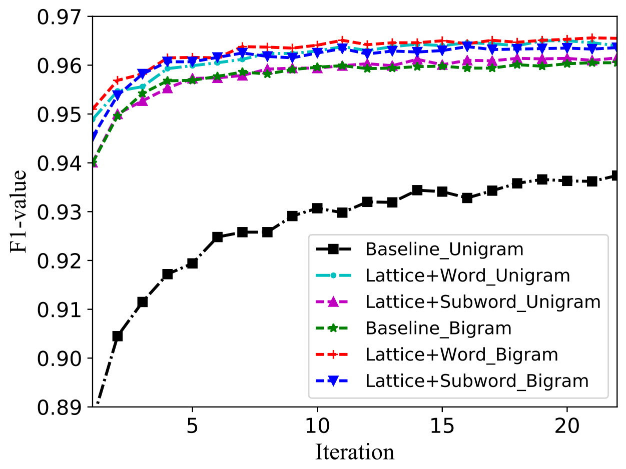

Figure 3: F1-value against training iterations.

gaword corpus[3]. The vocabulary of subword is constructed with 200000 merge operations and the subword embeddings are also trained using word2vec (Heinzerling and Strube 2018). Trie structure (Fredkin 1960) is used to accelerate the building of lattice (matching words/subwords). All the embeddings are fine-tuned during training.

Development Experiments

We perform development experiments on CTB6 development dataset to investigate the contribution of character bigram information and the word/subword information. Figure 3 shows the iteration curve of F-scores against different numbers of training iterations with different character representations. "_Unigram" means model using only character unigram information and "_Bigram" represents the model using both character unigram and bigram information (concatenating their embeddings). The performance of the baseline with only character unigram information is largely behind the others. By integrating the character bigram information, baseline model has a significant improvement. If we replace the character bigram with "Lattice+Word" or "Lattice+Subword", the model performance is even better. This proves that our lattice LSTM structure has a better ability to disambiguate the characters than character bigram information, we attribute this ability to the gate control mechanism which filters the noisy words/subwords.

Zhang and Yang (2018) observed that character bigram information has a negative effect on "Lattice+Word" structure on Chinese NER task, while it is different on Chinese segmentation task. We find the character bigram information gives significant improvements on both "Lattice+Word" and "Lattice+Subword" structures. This is likely because character bigrams are informative but ambiguous, they can provide more useful character disambiguation evidence in segmentation task than in NER task where "Lattice+Word" already works well in disambiguating characters.

[3]`https://catalog.ldc.upenn.edu/LDC2011T13`.

Models	**CTB6**	**SIGHAN**		**Weibo**
		PKU	**MSR**	
Zheng, Chen, and Xu (2013)	–	92.4	93.3	–
Pei, Ge, and Chang (2014)	–	95.2	97.2	–
Ma and Hinrichs (2015)	–	95.1	96.6	–
Zhang, Zhang, and Fu (2016)*	96.0	95.7	97.7	–
Xu and Sun (2016)	95.8	96.1	96.3	–
Cai et al. (2017)	–	95.8	97.1	–
Yang, Zhang, and Dong (2017)*	**96.2**	**96.3**	97.5	**95.5**
Zhou et al. (2017)	**96.2**	96.0	**97.8**	–
Baseline	95.8	95.3	97.4	95.0
Lattice+Word	**96.3**	**95.9**	97.7	95.1
Lattice+Subword	96.1	95.8	**97.8**	**95.3**

Table 3: Main results (F1). * represents model utilizing external supervised information.

Results

We evaluate our model on four datasets. The main results and the recent state-of-the-art neural CWS models are listed in Table 3. Zhang, Zhang, and Fu (2016) integrated both discrete features and neural features in a transition-based framework. Xu and Sun (2016) proposed the dependency-based gated recursive neural network to utilize long distance dependencies. Yang, Zhang, and Dong (2017) utilized the character representations which are jointly trained on several tasks such as punctuation prediction, POS tagging, and heterogeneous segmentation task. Zhou et al. (2017) trained the character embeddings by including the segmentation label information from large auto-segmented text. While our "Lattice+Subword" does not need those steps. These works are orthogonal to and can be integrated to our lattice LSTM model.

As shown in Table 3, the lattice LSTM models have significant improvement from the baseline on all datasets. The "Lattice+Word" model gives 11.9%, 12.8%, 11.5%, 2.0% error reductions on CTB6/PKU/MSR/Weibo datasets, respectively. And "Lattice+Subword" model has 7.14%, 10.6%, 15.4%, 6.0% error reductions on the four datasets, respectively. The reason for the different error reductions on different datasets is discussed in the analysis section. "Lattice+Word" and "Lattice+Subword" gives the best performance than other models on CTB6 and MSR, respectively. In PKU dataset, our lattice LSTM is slightly behind the model of Yang, Zhang, and Dong; Zhou et al.; Xu and Sun (2017; 2017; 2016), while the first two models utilize the external supervised or semi-supervised information and Xu and Sun (2016) preprocess the dataset by replacing all the Chinese idioms, leading the comparison not entirely fair. In conclusion, both word encoding and subword encoding can help the lattice LSTM model gives comparable performance to the state-of-the-art word segmentation models.

"Lattice+Subword" works better than "Lattice+Word" on MSR and Weibo datasets, while the latter gives more improvement on CTB6 and PKU datasets. Based on the results of lattice LSTM on the four examined datasets, "Lattice+Subword" has a comparable performance with "Lattice+Word".

Lexicon vs. Pretrained Emb	**P**	**R**	**F1**	**ER%**	$\mathbf{R}_{IV}$	$\mathbf{R}_{OOV}$
Baseline	95.93	95.62	95.78	0	96.70	77.36
Lattice+Subword Rand Emb	96.13	95.82	95.97	-4.5	96.85	78.37
Lattice+Subword Emb	96.23	95.90	96.07	-6.9	96.86	79.79
Lattice+Word Rand Emb	96.26	96.12	96.19	-9.7	97.00	81.28
Lattice+Word Emb	96.36	96.16	96.27	-11.6	97.05	81.02

Table 4: Lexicon and embeddings on CTB6.

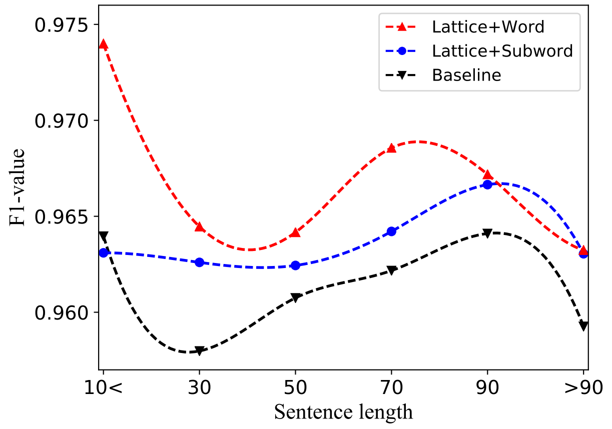

Figure 4: F1-value against the sentence length.

Analysis

Lexicon and Embeddings. Table 4 shows the model results on CTB6 test data. To distinguish the contribution from word/subword lexicon and their pretrained embeddings, we add another set of experiments by using the same word/subword lexicon with randomly initialized embeddings[4]. As shown in Table 4, the lattice LSTM models consistently outperform the baseline. Lattice LSTM with pretrained word embeddings gives the best result, with an 11.6% error reduction. The word auxiliary lattice LSTM outperforms the model with subword information on CTB6 dataset. The contribution of error reduction by the lexicon in "Lattice+Word" and "Lattice+Subword" are 4.5% and 9.7%, respectively. We can estimate the contribution of pretrained embeddings for "Lattice+Word" and "Lattice+Subword" are $(6.9\% - 4.5\%) = 2.4\%$ and $(11.6\% - 9.7\%) = 1.9\%$, respectively. The comparison between pretrained embeddings and randomly initialized embeddings shows that both pretrained embeddings and lexicon are useful to lattice LSTM, and lexicon contributes more than the pretrained embeddings.

OOV Analysis. Table 4 also shows the recall of in-vocabulary ($\mathbf{R}_{IV}$) and out-of-vocabulary ($\mathbf{R}_{OOV}$) words, respectively. As shown in the table, the recall of out-of-vocabulary words can be largely improved with the lattice structure ($2.43\%, 3.66\%$ absolute improvement for subword encoding and word encoding, respectively). The $\mathbf{R}_{OOV}$ of "Lattice+Subword" are largely improved ($+1.42\%$) with the pretrained subword embeddings. It is interesting that the $\mathbf{R}_{OOV}$ of "Lattice+Word" has a slight reduction when using pretrained word embeddings, we leave the investigation of this phenomenon in future work.

[4]Within $[-\sqrt{\frac{3}{dim}}, \sqrt{\frac{3}{dim}}]$, dim is the embedding size.

Data	**Emb**	**Split**	**#Word**	**#Match**	**Ratio** (%)	**ER** (%)
CTB6	Word	Train	641k	573k	89.35	–
		Test	81.6k	73.3k	89.79	11.9
	Subword	Train	641k	536k	83.57	–
		Test	81.6k	68.6k	84.13	7.14
PKU	Word	Train	1.01m	967k	95.89	–
		Test	104k	101k	96.63	12.8
	Subword	Train	1.01m	918k	90.87	–
		Test	104k	95.4k	91.42	10.6
MSR	Word	Train	2.12m	1.98m	93.37	–
		Test	107k	99.8k	93.38	11.5
	Subword	Train	2.12m	1.93m	91.12	–
		Test	107k	98.2k	91.91	15.4
Weibo	Word	Train	421k	370k	87.97	–
		Test	188k	162k	86.03	2.0
	Subword	Train	421k	337k	80.10	–
		Test	188k	147k	78.39	6.0

Table 5: Word/Subword coverage in lexicon. **#Word** is the number of words in the corresponding dataset, **#Match** is the number of matched words between the dataset and word/subword lexicon, **#Ratio** $= \frac{\textbf{\#Match}}{\textbf{\#Word}}$ represents the word coverage rate. **#ER** is the error reduction compared with baseline model.

Sentence Length. We compare the baseline model, lattice model with subword embeddings and word embeddings based on the sentence length. Figure 4 shows the F1 distribution on CTB6 dev dataset with respect to sentence length on three models. The baseline The performance of baseline has a trough in around 30-character sentences and decreases when the sentence length over 90, this phenomenon has also been observed in transition-based neural segmentor Yang, Zhang, and Dong (2017). "Lattice+Word" has a similar performance-length curve while "Lattice+Subword" gives a more stable performance along sentence length. One possible reason is that words are built using the auto-segmented corpus whose segmentor has the similar performance-length distribution. On the other hand, subwords in BPE algorithm are built on corpus level statistic information which is not related with sentence length. Hence the "Lattice+Subword" model gives a more stable performance distribution along sentence length.

Word/Subword Coverage in lexicon. Table 5 shows the word/subword coverage rate between word/subword lexicon with four datasets. Word/subword level coverage is consistently higher than the entity level coverage in Zhang and Yang (2018). In "Lattice+Word" and "Lattice+Subword" models, higher word/subword coverage (PKU/MSR, $>$ 90%) gives better error reduction rate. And on Weibo dataset, both two lattice models have limited improvements, as the word/subword coverage in this data is the lowest. On the other hand, although "Lattice+Subword" has lower word coverages in all datasets, it gives better performance than "Lattice+Word" on MSR and Weibo datasets. This shows that both the word coverage and the quality of subsequence embeddings are critical to lattice LSTM model and lattice LSTM with subword encoding can give comparable performance with lower word coverage.

#Example 1: where Lattice+Subword fails.		
Sentence		国际狮子会帮助湖北灾民住进新居 Int'l Lions Clubs help Hubei flood victims move in new house
Gold Segmentation		国际/ 狮子会/ 帮助/ 湖北/ 灾民/ 住进/ 新居 Int'l/Lions Clubs/help/Hubei/flood victims/move in/new house
Baseline		国际/ 狮子/ 会 / 帮助/ 湖北/ 灾民/ 住进/ 新居 Int'l/ Lion/will /help/Hubei/victims/move in/new house
L+Word	**Matched**	国际,狮子,狮子会,帮助,湖北,灾民,灾民住,民住,住进,新居 Int'l,Lions,Lions Clubs,help,Hubei,victims,×,×,move in,new house
	Decode	国际/ 狮子会/ 帮助/ 湖北/ 灾民/ 住进/ 新居 Int'l/Lions Clubs/help/Hubei/flood victims/move in/new house
L+Subword	**Matched**	国际,狮子,帮助,湖北,灾民,新居 Int'l,Lions,help,Hubei,victims,new house
	Decode	国际/ 狮子/ 会 / 帮助/ 湖北/ 灾民/ 住进/ 新居 Int'l/ Lion/will /help/Hubei/victims/move in/new house
#Example 2: where Lattice+Word fails.		
Sentence		国际生物多样性日纪念大会在京举行 Int'l Biological Diversity Day COMM meeting in Beijing hold
Gold Segmentation		国际/ 生物/ 多样性/ 日/ 纪念/ 大会/ 在/ 京/ 举行 Int'l/Biological/Diversity/Day/COMM/meeting/in/Beijing/hold
Baseline		国际/ 生物/ 多样性日/ 纪念/ 大会/ 在/ 京/ 举行 Int'l/Biological/ DiversityDay COMM/meeting/in/Beijing/hold
L+Word	**Matched**	国际,生物,多样性,性日,纪念,大会,在京,京举,举行 Int'l,Biological,Diversity,×,COMM,meeting,in Beijing,×,hold
	Decode	国际/ 生物/ 多样/ 性日 / 纪念/ 大会/ 在/ 京/ 举行 Int'l/Biological/ Diverse/× /COMM/meeting/in/Beijing/hold
L+Subword	**Matched**	国际,生物多样性,多样性,纪念,大会,在京,举行 Int'l,Biological Diversity,Diversity,COMM,meeting,in Beijing,hold
	Decode	国际/ 生物/ 多样性/ 日/ 纪念/ 大会/ 在/ 京/ 举行 Int'l/Biological/Diversity/Day/COMM/meeting/in/Beijing/hold

Figure 5: Examples. Red and green color represent incorrect and correct segmentation, respectively. × represents ungrammatical word. Words with underlines are critical to the segmentation errors.

Case Study. Figure 5 shows two examples of the segmentation results on CTB6 test dataset. In example 1, both baseline and "Lattice+Subword" fail to give correct segmentation of "狮子会 (Lions Clubs)" while "Lattice+Word" can successfully distinguish it. In this example, both "Lattice+Word" and "Lattice+Subword" have the noisy matched word/subword "狮子(Lion)", but there is an extra matched word "狮子会 (Lions Clubs)" in "Lattice+Word" to provide a stronger evidence of segmentation due to the gate control mechanism. Example 2 shows another example that "Lattice+Subword" gives the right segmentation while "Lattice+Word" fails. In this case, both "Lattice+Word" and "Lattice+Subword" have the correct matched word/subword "多样性(Diversity)" while "Lattice+Word" has an extra noisy matched word "性日(×)" which misleads "Lattice+Word" to segment the sentence incorrectly. We conclude that the gate control mechanism for matched words/subwords is useful but not perfect. Based on the final performance in table 3, the lattice LSTM with gate control structure has advantages outweigh its disadvantages.

Conclusion

We investigated the use of ambiguous words and subwords for CWS with lattice LSTM. Subsequences using word embeddings collected from auto-segmented text and subword embeddings deduced from BPE algorithm are empirically compared. Results on four benchmarks show that the subword encoding works comparable with word encoding in lattice LSTM, both significantly improve the segmentation and give comparable performance to the best systems on all evaluated datasets. Experiments also show that the matched subsequence lexicon contributes more than the pretrained embeddings, this shows the potential of incorporating lattice LSTM structure with domain lexicon for cross-domain sequence labeling. We also observe that higher word/subword coverage leads to larger improvement in both "Lattice+Word" and "Lattice+Subword" models. With the case study, we find that lattice LSTM with gate control structure is useful but can still fail in some cases, which needs deeper investigation in the future.

References

[2016] Cai, D., and Zhao, H. 2016. Neural word segmentation learning for chinese. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 409–420.

[2017] Cai, D.; Zhao, H.; Zhang, Z.; Xin, Y.; Wu, Y.; and Huang, F. 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)*, 608–615.

[2015a] Chen, X.; Qiu, X.; Zhu, C.; and Huang, X. 2015a. Gated recursive neural network for chinese word segmentation. In *Proceedings of the 53th Annual Meeting of the Association for Computational Linguistics*.

[2015b] Chen, X.; Qiu, X.; Zhu, C.; Liu, P.; and Huang, X. 2015b. Long short-term memory neural networks for chinese word segmentation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1385–1394.

[2017] Chen, X.; Shi, Z.; Qiu, X.; and Huang, X. 2017. Dag-based long short-term memory for neural word segmentation. *arXiv preprint arXiv:1707.00248*.

[2005] Emerson, T. 2005. The second international chinese word segmentation bakeoff. In *Proceedings of the fourth SIGHAN workshop on Chinese language Processing*, volume 133.

[1960] Fredkin, E. 1960. Trie memory. *Communications of the ACM* 3(9):490–499.

[1994] Gage, P. 1994. A new algorithm for data compression. *The C Users Journal* 12(2):23–38.

[2017] Greff, K.; Srivastava, R. K.; Koutník, J.; Steunebrink, B. R.; and Schmidhuber, J. 2017. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems* 28(10):2222–2232.

[2018] Heinzerling, B., and Strube, M. 2018. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In chair), N. C. C.; Choukri, K.; Cieri, C.; Declerck, T.; Goggi, S.; Hasida, K.; Isahara, H.; Maegaard, B.; Mariani, J.; Mazo, H.; Moreno, A.; Odijk, J.; Piperidis, S.; and Tokunaga, T., eds., *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

[1997] Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

[2001] Lafferty, J. D.; McCallum, A.; and Pereira, F. C. N. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, 282–289. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[1989] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. D. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1(4):541–551.

[2015] Ma, J., and Hinrichs, E. 2015. Accurate linear-time chinese word segmentation via embedding matching. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, 1733–1743.

[2013] Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[2014] Pei, W.; Ge, T.; and Chang, B. 2014. Max-margin tensor neural network for chinese word segmentation. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics*, 293–303. Association for Computational Linguistics.

[2004] Peng, F.; Feng, F.; and McCallum, A. 2004. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of the International Conference on Computational Linguistics*, 562.

[2016] Qiu, X.; Qian, P.; and Shi, Z. 2016. Overview of the nlpcc-iccpol 2016 shared task: Chinese word segmentation for micro-blog texts. In *International Conference on Computer Processing of Oriental Languages*, 901–906. Springer.

[2016] Sennrich, R.; Haddow, B.; and Birch, A. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 1715–1725.

[2017] Sperber, M.; Neubig, G.; Niehues, J.; and Waibel, A. 2017. Neural lattice-to-sequence models for uncertain inputs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1380–1389.

[1996] Sproat, R.; Gale, W.; Shih, C.; and Chang, N. 1996. A stochastic finite-state word-segmentation algorithm for chinese. *Computational linguistics* 22(3):377–404.

[2014] Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

[2017] Su, J.; Tan, Z.; Xiong, D.; Ji, R.; Shi, X.; and Liu, Y. 2017. Lattice-based recurrent neural network encoders for neural machine translation. In *AAAI*, 3302–3308.

[2010] Sun, W. 2010. Word-based and character-based word segmentation models: Comparison and combination. In *Proceedings of the International Conference on Computational Linguistics*, 1211–1219.

[1967] Viterbi, A. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory* 13(2):260–269.

[2017] Wang, C., and Xu, B. 2017. Convolutional neural network with word embeddings for chinese word segmentation. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing*, 163–172. Taipei, Taiwan: Asian Federation of Natural Language Processing.

[2016] Xu, J., and Sun, X. 2016. Dependency-based gated recursive neural network for chinese word segmentation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 567. Association for Computational Linguistics.

[2003] Xue, N., et al. 2003. Chinese word segmentation as character tagging. *Computational Linguistics and Chinese Language Processing* 8(1):29–48.

[2005] Xue, N.; Xia, F.; Chiou, F.-D.; and Palmer, M. 2005. The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural language engineering* 11(02):207–238.

[2017] Yang, J.; Zhang, Y.; and Dong, F. 2017. Neural word segmentation with rich pretraining. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, 839–849.

[2007] Zhang, Y., and Clark, S. 2007. Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, volume 45, 840.

[2018] Zhang, Y., and Yang, J. 2018. Chinese ner using lattice lstm. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

[2016] Zhang, M.; Zhang, Y.; and Fu, G. 2016. Transition-based neural word segmentation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

[2006] Zhao, H.; Huang, C.-N.; Li, M.; and Lu, B.-L. 2006. Effective tag set selection in chinese word segmentation via conditional random field modeling. In *Proceedings of the Pacific Asia Conference on Language, Information and Computation*, volume 20, 87–94. Citeseer.

[2013] Zheng, X.; Chen, H.; and Xu, T. 2013. Deep learning for chinese word segmentation and pos tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 647–657.

[2017] Zhou, H.; Yu, Z.; Zhang, Y.; Huang, S.; DAI, X.-Y.; and Chen, J. 2017. Word-context character embeddings for chinese word segmentation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 760–766.

[2016] Zhu, X.; Sobhani, P.; and Guo, H. 2016. Dag-structured long short-term memory for semantic compositionality. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 917–926.