Journal of Artificial Intelligence Research 63 (2018) 923-953 Submitted 07/18; published 12/18

Transition-Based Neural Word Segmentation Using Word-Level Features

Meishan Zhang MASON.ZMS@GMAIL.COM
School of Computer Science and Technology,
Heilongjiang University, Harbin, China

Yue Zhang FRCCHANG@GMAIL.COM
Westlake University, Hangzhou, China

Guohong Fu GHFU@HOTMAIL.COM
School of Computer Science and Technology,
Heilongjiang University, Harbin, China

Abstract

Character-based and word-based methods are two different solutions for Chinese word segmentation, the former exploiting sequence labeling models over characters and the latter using word-level features. Neural models have been exploited for character-based Chinese word segmentation, giving high accuracies by making use of external character embeddings, yet requiring less feature engineering. In this paper, we study a neural model for word-based Chinese word segmentation, by replacing the manually-designed discrete features with neural features in a transition-based word segmentation framework. Experimental results demonstrate that word features lead to comparable performance to the best systems in the literature, and a further combination of discrete and neural features obtains top accuracies on several benchmarks.

1. Introduction

The task of word segmentation is to chunk an input sequence of characters into a word sequence. For example, given the sentence "中国外企业务发展迅速" (The business of a foreign company in China develops quickly) in Chinese, the output word sequence can be "中国 (Chinese) 外企 (foreign company) 业务 (business) 发展 (develop) 迅速 (quickly)". Word segmentation is a first step to natural language processing and text mining tasks for Chinese, Japanese and Thai.

Data-driven word segmentation methods can be categorized into **character-based** (Xue, 2003; Tseng, Chang, Andrew, Jurafsky, & Manning, 2005) and **word-based** (Andrew, 2006; Zhang & Clark, 2007; Sun, Zhang, Matsuzaki, Tsuruoka, & Tsujii, 2009) approaches. The former casts word segmentation as a sequence labeling problem, using segmentation tags on characters to mark their relative positions inside words. The latter, in contrast, ranks candidate segmented outputs directly, extracting both character and full-word features. An influential character-based word segmentation model (Peng, Feng, & McCallum, 2004; Tseng et al., 2005) uses B/I/E/S labels to mark a character as the beginning, internal (neither beginning nor end), end of word and single-character words, respectively, employing conditional random fields (CRF) to model the correspondence between the input character sequence and output label sequence. For each character, features are extracted

	character-based	word-based
discrete	Peng et al. (2004) Tseng et al. (2005)	Andrew (2006) Zhang and Clark (2007)
neural	Zheng et al. (2013) Pei et al. (2014) Chen et al. (2015b)	**this work (Zhang et al., 2016)** Liu et al. (2016) Cai and Zhao (2016)

Figure 1: Word segmentation methods.

from a five-character context window and a two-label history window. Subsequent work explores different label sets (Zhao, Huang, Li, & Lu, 2006), feature sets (Shi & Wang, 2007; Zhang, Wang, Sun, & Mansur, 2013) and semi-supervised learning (Sun & Xu, 2011; Wang, Kazama, Tsuruoka, Chen, Zhang, & Torisawa, 2011), reporting state-of-the-art accuracies.

With the rise of deep learning, neural network models have been investigated for the character tagging approach. The main idea is to replace manual discrete features with automatic real-valued features, which are derived automatically from distributed character representations using neural networks. In particular, convolutional neural network (Zheng, Chen, & Xu, 2013), tensor neural network (Pei, Ge, & Chang, 2014), recursive neural network (Chen, Qiu, Zhu, & Huang, 2015a) and long-short-term-memory (LSTM) (Chen, Qiu, Zhu, Liu, & Huang, 2015b) have been used to derive neural feature representations from input character sequences, which are fed into a CRF inference layer.

While the above methods leverage deep neural networks over characters for effective feature derivation, it has nevertheless been shown that additional input features beyond character embeddings can be useful. For example, Peng et al. (2004) have demonstrated that word-level features such as (sub)word information are a kind of highly effective features in a discrete semi-CRF model. Similarly Zhang and Clark (2007) have made full use of word-level features by using an incremental decoding model. Output word information can also be directly useful for neural segmentation models.

In this paper, we investigate the effectiveness of word-based neural Chinese word segmentation (Zhang, Zhang, & Fu, 2016; Liu, Che, Guo, Qin, & Liu, 2016; Cai & Zhao, 2016). Its correlation with existing work on Chinese word segmentation is shown in Figure 1. Since it is challenging to integrate word features to the CRF inference framework of the existing character-based methods, we take inspiration from word-based discrete segmentation model instead. In particular, we follow Zhang and Clark (2007), using a transition-based framework (Zhang & Clark, 2011; Zhou, Zhang, Cheng, Huang, Dai, & Chen, 2017) to segment a sentence incrementally from left to right, scoring partially segmented results by using both character-level and word-level features. We replace the discrete word and character features of Zhang and Clark (2007) with neural word and character representations, respectively, and change their linear model into a deep neural network. Following Chen et al. (2015b), we build LSTMs over the input character sequence and the output word sequence, and exploit the output features for scoring. Similar to Zhang and Clark (2011), beam-search is applied to reduce error propagation and online large-margin training with early-update (Collins & Roark, 2004) is used for learning from inexact search. The resulting model is a word-based neural segmenter that can leverage rich character and word-level features.

We conduct experiments on several benchmark datasets to thoroughly examine the effectiveness of neural word features. Results show the effectiveness of word and subword level features for neural Chinese word segmentation. With pretrained character and word embeddings, our method achieves state-of-the-art results. In addition, a combination of our neural features and the traditional discrete features results in further improved performance. We conduct a number of experimental analysis for deeper understanding our proposed neural model.

This article is a much extended version of our conference paper (Zhang et al., 2016), with significant improvements of the neural model and extended discussions. First, we optimize the neural feature sets for the transition-based model, by exploiting only minimal LSTM features from characters and words and removing unimportant features. Second, we propose an improved method to train word embeddings, with which the performance of the neural model increases significantly. Third, we make comparisons of the max-margin training and the max-likelihood training, empirically demonstrating that the former is more suitable for our model. Finally we conduct more analysis for deeper understanding on transition-based neural segmentation. We make our codes and models publicly available under GPL at https://github.com/zhangmeishan/NNTranSegmentor.

2. Baseline Transition Based Discrete Model

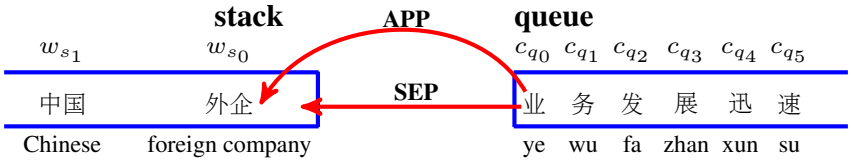
We exploit the word-based segmentor of Zhang and Clark (2011) as the baseline system, which adopts the transition-based structural learning framework to formalize the word segmentation problem. The transition-based framework has been shown effective for a number of NLP tasks such as POS tagging, syntax parsing and relation extraction (Collins, 2002; Zhang & Clark, 2011; Li & Ji, 2014; Zhou, Zhang, Huang, & Chen, 2015; Andor, Alberti, Weiss, Severyn, Presta, Ganchev, Petrov, & Collins, 2016).

2.1 The Transition System

The transition-based framework treats word segmentation as a state transition process, incrementally segmenting a sentence from left to right, where a state represents a partially-segmented output, and a transition action incrementally processes the segmentation of each character. A *state* holds partially-segmented sentence in a stack s and order the next incoming characters in a queue q. Given an input Chinese sentence, the *initial state* has an empty stack and the queue containing all characters of the sentence. A sequence of *transition actions* are used to consume characters in the queue and build the output sentence in the stack. After the incremental processing, we obtain an *end state* with the segmented sentence in the stack and no characters in the queue. The end state denotes a full word segmentation result for the input sentence. The actions include:

- `Append` (APP), which removes the first character from the queue, and appends it to the last word in the stack;

- `Separate` (SEP), which moves the first character of the queue onto the stack as a new (sub) word.



(a) An example state, where arrows denote the inserting positions of the next character (i.e. APP will append the next character onto the end of the last word, and SEP will insert a space).

step	last action	buffer($\cdots w_{s_1} w_{s_0}$)	queue($c_{q_0} c_{q_1} \cdots$)
0	-	ϕ	中 国 ···
1	*SEP*	中	国 外 ···
2	*APP*	中国	外 企 ···
3	*SEP*	中国 外	企 业 ···
4	*APP*	中国 外企	业 务 ···
5	*SEP*	中国 外企 业	务 发 ···
6	*APP*	中国 外企 业务	发 展 ···
7	*SEP*	··· 业务 发	展 迅 速
8	*APP*	··· 业务 发展	迅 速
9	*SEP*	··· 发展 迅	速
10	*APP*	··· 发展 迅速	ϕ

(b) The state-transition process.

Figure 2: The transition system of our model, where the resulting segmented sentence is "中国 (Chinese) 外企 (foreign company) 业务 (business) 发展 (develop) 迅速 (quickly)".

Figure 2(a) shows an example state and possible actions could be applied on it. Given the input sequence of characters "中国外企业务发展迅速" (The business of foreign company in China develops quickly), the gold-standard output "中国 (Chinese) 外企 (foreign company) 业务 (business) 发展 (develop) 迅速 (quickly)" can be derived by using action sequence "*SEP APP SEP APP SEP APP SEP APP SEP APP*", as shown in Figure 2(b). After the state transitions, the word sequence on the final stack is the output.

2.2 Model

Based on the transition system, our baseline segmentation model searches for an optimal end state ST_n for a given sentence $c_1 \cdots c_n$, which is constructed by a certain action sequence $A = a_1 \cdots a_n$. The model score of a state is calculated as the total score of the actions that are used to build the state:

$$\mathrm{score}(\mathrm{ST}_n) = \sum_{i=1}^{n} \mathrm{score}(a_i | \mathrm{ST}_{i-1}), \tag{1}$$

where $\mathrm{score}(a_i|\mathrm{ST}_{i-1})$ denotes the action score at the ith step, and $\mathrm{ST}_0 \cdots \mathrm{ST}_{n-1}$ are the sequence of intermediate states to reach ST_n.

Feature templates	Action
$c_{q_0-1}c_{q_0}$, $cat(c_{q_0-1})cat(c_{q_0})cat(c_{q_1})$	*APP, SEP*
w_{s_0}, $w_{s_0}w_{s_1}$, $w_{s_0}c_{q_0}$, $w_{s_1}len(w_{s_0})$, $start(w_{s_0})c_{s_0}$, $end(w_{s_0})c_{s_1}$, $start(w_{s_0})end(w_{s_0})$, $end(w_{s_1})end(w_{s_0})$, $w_{s_1}len(w_{s_0})$, $len(w_{s_1})w_{s_0}$, w_{s_0}, where $len(w_{s_0}) = 1$	*SEP*

Table 1: Feature templates for the baseline model, where w_i denotes the word in the stack, c_i denotes the character in the queue, as shown in Figure 2, $cat(.)$ denotes the category of one character, $start(.)$, $end(.)$ and $len(.)$ denote the first, last character and length of a word, respectively.

In the baseline segmentation model, the action score can be calculated by:

$$\text{score}(a|\text{ST}) = \boldsymbol{\theta} \cdot \boldsymbol{f}(\text{ST}, a) \tag{2}$$

where ST and a are a state and a next action, respectively, $\boldsymbol{\theta}$ is the model parameter and $\boldsymbol{f}$ is a feature extraction function. The score models the next action a when current state ST is given.

Equation 1 can be calculated incrementally. Given one current state $\text{ST}_{j-1}(j \geq 1)$, and the next action a_j, the score of the next state ST_j can be computed by:

$$\begin{aligned} \text{score}(\text{ST}_j) &= \sum_{i=1}^{j} \text{score}(a_i|\text{ST}_{i-1}) \\ &= \sum_{i=1}^{j-1} \text{score}(a_i|\text{ST}_{i-1}) + \text{score}(a_j|\text{ST}_{j-1}) \\ &= \text{score}(\text{ST}_{j-1}) + \text{score}(a_j|\text{ST}_{j-1}). \end{aligned} \tag{3}$$

The score of all initial states is set as zero ($\text{score}(\text{ST}_0) = 0$).

Our feature extraction function $\boldsymbol{f}$ is defined according to the set of feature templates shown in Table 1, which are similar to Zhang and Clark (2011). The features differ by the action to take. These base features include two main source of information. First, character level features that can be extracted without knowing the identified words, such as the first and second characters (c_0, c_1) on the queue and the top character on the stack (c_{-1}). These features are used for scoring both *separate* and *append*. We also exploit the category information of characters, where one character is classified into one of the five categories: Chinese character, letter, punctuation, digit and other. Second, the word level information that has been identified already is used to guide *separate* actions. These extracted atomic features include word forms, lengths (e.g. $len(w_{-1})$) and the first/last characters (e.g. $start(w_{-1})$ / $end(w_{-1})$).

2.3 Search

We follow Zhang and Clark (2011) in using beam-search for decoding, shown in Algorithm 1, where Θ is the set of model parameters (for the baseline model, $\Theta = \boldsymbol{\theta}$, the parameter

Algorithm 1 Beam-search decoding, where Θ is the set of all model parameters.

function DECODE($c_1 \cdots c_n$, Θ)
  agenda $\leftarrow$ { (ϕ (empty stack), $c_1 \cdots c_n$ (queue), score=0.0) }
  for k **in** $1 \cdots n$
    list $\leftarrow$ { }
    for *candidate* **in** *agenda*
      new $\leftarrow$ APPLY(SEP, *candidate*, c_k, Θ)
      ADDITEM(list, new)
      new $\leftarrow$ APPLY(APP, candidate, c_k, Θ)
      ADDITEM(*list*, *new*)
    agenda $\leftarrow$ TOP-B(*list*, *B*)
  best $\leftarrow$ BESTITEM(*agenda*)
  $w_1 \cdots w_m \leftarrow$ EXTRACTWORDS(*best*)

vector). Initially the beam contains only the initial state, where ϕ denotes the empty stack, and $c_1 \cdots c_n$ denotes the queue containing all sentential characters. At each step, each state in the beam is extended by applying both *SEP* and *APP*, accompanied by the character in the queue being processed by the two actions, resulting in a set of new states, which are scored and ranked according to Equation 3. The top B states are used as the beam for the next step. The same process replaces until all input character are processed, and the highest-scored state in the beam is taken for output.

2.4 Training

To train the model parameter $\boldsymbol{\theta}$, we exploit online leaning with early-update, using standard back-propagation algorithm based on two widely-used objective functions, a max-margin objective and a max-likelihood objective, respectively, which are described in Section 5.

3. Transition-Based Neural Model

We propose to use neural features instead in the same transition system, first representing atomic features by low-dimensional real vectors, and then using non-linear neural layers such as LSTMs to compose these features automatically. Thus all features in the neural segmentation model are dense features, alleviating the feature sparsity problem naturally, and meanwhile it is free of feature engineering. For better comparison between discrete and neural features, the overall segmentation framework of the baseline is kept, which includes the incremental segmentation process, the beam-search decoder and the training process integrated with beam-search (Zhang & Clark, 2011; Zhou et al., 2017). In addition, the neural network scorer takes the similar feature sources as the baseline, which includes both the input character sequence and the partially constructed output word sequence.

The overall architecture of the neural-based scorer is illustrated in Figure 3. Given a state ST, we use two separate recurrent neural networks (RNN) for feature extraction. On the one hand, a bi-directional RNN is used to model the input character sequence $c_1 \cdots \cdots c_n$, and on the other hand a left-to-right RNN is used to model the identified word

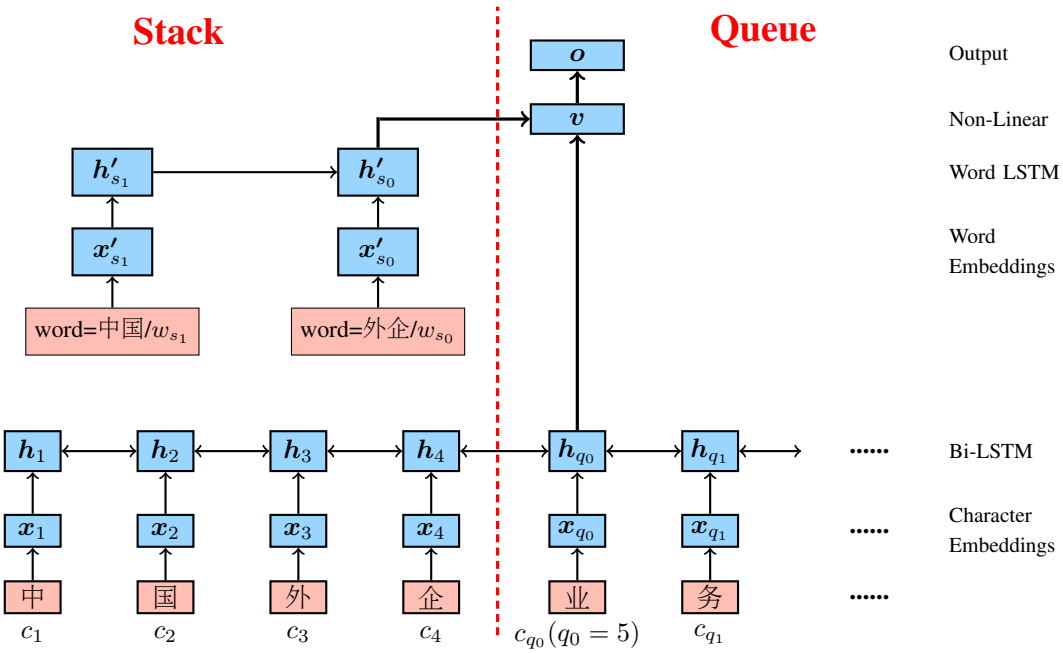


Figure 3: Scorer for the neural transition-based Chinese word segmentation model.

sequence $w_{s_m} \cdots w_{s_0}$. We exploit two dense real-valued vectors $\boldsymbol{h}_{q_0}$ (hidden output of the next coming character) and $\boldsymbol{h}'_{s_0}$ (hidden output of the last identified word) as source features to represent one state, both of which can encode the entire character/word sequences globally, benefiting from the RNN neural network structure, while our baseline discrete model is only able to use the local information from a limited window size.

We adopt a non-linear feed-forward neural layer to combine $\boldsymbol{h}_{q_0}$ and $\boldsymbol{h}'_{s_0}$ resulting in a feature vector $\boldsymbol{v}$ to represent current state ST, which is used to score the next actions. The feature vector $\boldsymbol{v}$ conceptually corresponds to the discrete feature vector $\boldsymbol{f}$ in our baseline model. While the baseline model obtains the final feature vector by manual engineering, the neural model obtains the feature vector automatically. In the following subsection, we describe our neural network model in detail.

3.1 Feature Representation

The neural networks take the neural representations of words and characters as input, for extracting $\boldsymbol{h}_{q_0}$ and $\boldsymbol{h}'_{s_0}$, respectively, and then represent a state by a non-linear composition. We exploit the LSTM-RNN structure (Hochreiter & Schmidhuber, 1997), which can better capture non-local syntactic and semantic information from a sequential input, yet reducing gradient explosion or vanishing during training.

LSTM-RNN. In general, given a sequence of input vectors $\boldsymbol{x}_1 \cdots \boldsymbol{x}_n$, the LSTM-RNN computes a sequence of hidden vectors $\boldsymbol{h}_1 \cdots \boldsymbol{h}_n$, respectively, with each $\boldsymbol{h}_t$ being determined by the input $\boldsymbol{x}_t$ and the previous hidden vector $\boldsymbol{h}_{t-1}$. A cell structure $\boldsymbol{c}$ is used to carry long-term memory information over the history $\boldsymbol{h}_1 \cdots \boldsymbol{h}_{t-1}$ for calculating $\boldsymbol{h}_t$, and information flow is controlled by an input gate $\boldsymbol{i}$, an output gate $\boldsymbol{o}$ and a forget gate $\boldsymbol{f}$. We use the

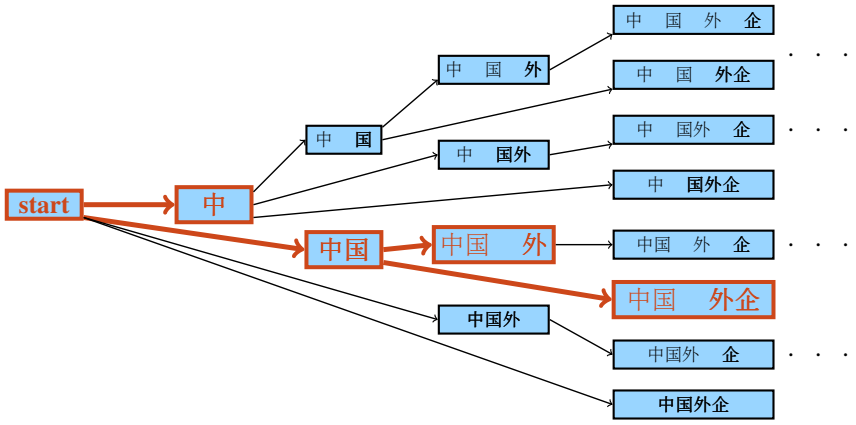


Figure 4: An example computation graph of word-level LSTM, where each node denotes one LSTM state, and the last word in the node denotes the input source of the last step. The entire search space is displayed, which can be pruned by beam search decoding. The brown thick paths show historical LSTM states with the gold-standard actions.

LSTM-RNN variation proposed by Gers, Schraudolph, and Schmidhuber (2002) in this version. Formally, the calculation of $\boldsymbol{h}_t$ using $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ is:

$$\begin{aligned}
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_i\boldsymbol{x}_t + \boldsymbol{U}_i\boldsymbol{h}_{t-1} + \boldsymbol{V}_i\boldsymbol{c}_{t-1} + \boldsymbol{b}_i) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f\boldsymbol{x}_t + \boldsymbol{U}_f\boldsymbol{h}_{t-1} + \boldsymbol{V}_f\boldsymbol{c}_{t-1} + \boldsymbol{b}_f) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tanh(\boldsymbol{W}_c\boldsymbol{x}_t + \boldsymbol{U}_c\boldsymbol{h}_{t-1} + \boldsymbol{b}_c) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o\boldsymbol{x}_t + \boldsymbol{U}_o\boldsymbol{h}_{t-1} + \boldsymbol{V}_o\boldsymbol{c}_t + \boldsymbol{b}_o) \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t),
\end{aligned} \tag{4}$$

where $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, \boldsymbol{b}$ are model parameters, and $\odot$ denotes Hadamard product.

Encoding characters. To calculate $\boldsymbol{h}_{q_0}$, the above LSTM structure takes embeddings of the input character sequence as inputs, i.e. $\boldsymbol{x}_i = \text{emb}(c_i)$, where $\text{emb}(c_i)$ is obtained by from a look up matrix $\boldsymbol{E}^c$. We use left-to-right LSTM and right-to-left LSTMs over $\boldsymbol{x}_1 \cdots \boldsymbol{x}_n$, obtaining $\overrightarrow{\boldsymbol{h}}_1 \cdots \overrightarrow{\boldsymbol{h}}_n$ and $\overleftarrow{\boldsymbol{h}}_1 \cdots \overleftarrow{\boldsymbol{h}}_n$. Then we concatenate $\overrightarrow{\boldsymbol{h}}_i$ and $\overleftarrow{\boldsymbol{h}}_i$, obtaining $\boldsymbol{h}_i = \overrightarrow{\boldsymbol{h}}_i \oplus \overleftarrow{\boldsymbol{h}}_i$. For each state, we first locate the position of q_0, and then fetch the corresponding hidden output $\boldsymbol{h}_{q_0}$.

Encoding words. To calculate $\boldsymbol{h}'_{s_0}$, the same LSTM structure above is used on the embedding sequence of existing output words $\boldsymbol{x}'_{s_m} \cdots \boldsymbol{x}'_{s_0}$ (noting that $s_m \cdots s_0$ are consecutive). $\boldsymbol{x}'_{s_i} = \text{emb}(w_{s_i})$, where $\text{emb}(w_{s_i})$ is taken from a looking-up table $\boldsymbol{E}^w$. As the word sequence is built during the decoding progress, only a left-to-right word-level LSTM can be computed. We use the last hidden output $\boldsymbol{h}'_{s_0}$ of the word LSTM for disambiguation.

The calculation of the word-level LSTM is not as straightforward as the character-level LSTM since the word sequence is produced incrementally and dynamically. Thus the word-level LSTM is also built incrementally with each action sequence. Figure 4 shows an example to illustrate the construction of all possible word LSTM state sequences given the sentence

"中国外企...." We can see that a tree-alike structure of LSTM states is constructed. Every path from the start node to one intermiddle/leaf tree node reflects one sequential left-to-right LSTM.

State representation. Given $\boldsymbol{h}_{q_0}$ and $\boldsymbol{h}'_{s_0}$, we obtain the feature representation vector $\boldsymbol{v}$ for the state ST as follow:

$$\boldsymbol{v} = \tanh(\boldsymbol{W}'[\boldsymbol{h}_{q_0} \oplus \boldsymbol{h}'_{s_0}] + \boldsymbol{b}') \tag{5}$$

where $\boldsymbol{W}'$ and $\boldsymbol{b}'$ are model parameters and $\oplus$ denotes vector concatenation.

3.2 Scoring Actions

Given the state representation $\boldsymbol{v}$, the score of a SEP/APP action can be computed by:

$$\begin{aligned} \boldsymbol{o} &= \boldsymbol{W}\boldsymbol{v} \\ score(a|\mathrm{ST}) &= \boldsymbol{o}_a, \end{aligned} \tag{6}$$

where $\boldsymbol{W}$ is one model parameter, and $\boldsymbol{o}$ is a two-dimensional vector that denotes the action scores on a certain state, and $a \in \{\mathrm{SEP}, \mathrm{APP}\}$. Finally, based on the newly defined $score(a|\mathrm{ST})$ for the neural model, we can compute the state scores incrementally by using Equation 3, the same as the baseline discrete model.

3.3 Search and Training

We use the same global learning and beam-search algorithms as the baseline model. Here the set of model parameters Θ covers the parameters of all neural layers, including embeddings, character/word-level LSTMs, non-linear feed-forward layer for state representation and the output layer for action scoring.

3.4 Differences from Zhang et al. (2016)

The neural network structure above has several minor differences compared with the structure presented in our original conference version (Zhang et al., 2016). First, we simplify the neural network structures by using the same sources of features for action disambiguation. For example, the conference version adopts the word LSTM features only for SEP actions, while in this article we use them for both SEP and APP actions, and the preliminary results demonstrate that this simplified structure can archive slightly better performances. The advantage can be further enlarged by exploration of pretrained subword embeddings. Second, we minimize the sources of features, keeping the model as simple as possible. For example, we discard the action LSTM features of Zhang et al. (2016), the character type embeddings, the word length embeddings. We also remove the character-bigram embeddings, which has been shown important for Chinese segmentation in previous work (Pei et al., 2014; Yang, Zhang, & Dong, 2017). Third, we propose a novel effective method to train word-level embeddings that is capable of handling subwords, which potentially makes the information from character bigram embeddings be redundant.

3.5 Integrating Discrete and Neural Features

Our model can be extended by integrating the baseline discrete features into the feature layer. In particular,

$$score(a|\mathrm{ST}) = \alpha \boldsymbol{o}_a + (1-\alpha)\big(\boldsymbol{\theta} \cdot \boldsymbol{f}(\mathrm{ST}, a)\big) \tag{7}$$

where $\boldsymbol{o}_a$ and $\boldsymbol{\theta} \cdot \boldsymbol{f}(\mathrm{ST}, a)$ are the scores produced by the neural features and the discrete features, respectively, and $\alpha \in [0, 1]$ is a factor to scale the two kinds of scores. We use a simple score addition operation to make integration of these two kinds of features.

4. Embeddings

As described in Section 3, the basic inputs of our neural word segmentation model are embeddings of characters and words. Such embeddings can be handled using different methods. A straightforward method is to initialize the embeddings randomly, and fine tune them along with model training. However, one drawback is that we are only able to learn effective embeddings for characters and words that occur in the segmentation training corpus. For low-frequency and out-of-vocabulary (OOV) characters and words, it is difficult to obtain their embeddings. One common way to solve this issue is to pretrain embeddings on a very large scale raw corpus (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013; Levy & Goldberg, 2014; Pennington, Socher, & Manning, 2014; Peters, Neumann, Iyyer, Gardner, Clark, Lee, & Zettlemoyer, 2018), which can give embeddings of much larger character/word vocabularies, and meanwhile avoid the sparsity problem of mapping them into their vector representations.

We exploit the skip-gram model (Mikolov et al., 2013) to pretrain character and word embeddings from a large-scale corpus. The key idea is to build a classifier to predict one unit's surrounding contexts, where the training examples can be constructed automatically from raw sequential sentences without any human supervision. The model parameters of the classifier consist of: (1) the unit embeddings, and (2) the context embeddings. The pretrained unit embeddings can capture syntactic/semantic information of the vocabulary units, which are further used as initialization in other supervised models.

4.1 Character Embeddings

The character embeddings are pretrained using the *word2vec*[1] tool (Mikolov et al., 2013). Give an input raw sentence $c_1 c_2 \cdots c_n$, for each character c_i, the model extracts several training examples $\{< c_i, c_j >\}$ from the sentence, where $j = i \pm k$ and $(k \in [1, c])$. And then a classifier is built, aiming to predict c_j based on c_i. Standard cross-entropy loss is applied during training, which can be formalized as:

$$\begin{aligned} obj(< c_i, c_j >, W_{\text{context}}, W_{\text{source}}) &= -\log \mathrm{Prob}(c_j|c_i) \\ &= -\log \frac{\exp(\boldsymbol{\mu}_{c_j} \boldsymbol{\nu}_{c_i}^T)}{\sum_{c'} \exp(\boldsymbol{\mu}_{c'} \boldsymbol{\nu}_{c_i}^T)} \\ &= \log \sum_{c'} \exp(\boldsymbol{\mu}_{c'} \boldsymbol{\nu}_{c_i}^T) - \boldsymbol{\mu}_{c_j} \boldsymbol{\nu}_{c_i}^T, \end{aligned} \tag{8}$$

1. http://word2vec.googlecode.com/

where $\boldsymbol{\mu}_c$ denotes the character embedding as context by looking up from W_{context}, and $\boldsymbol{\nu}_c$ denotes the character embedding by looking up from W_{source} as the classifier input, and c' denotes all possible characters. The total number of target categories equals the size of character vocabulary, which could be very large, thus the negative sampling is applied to appropriate the denominator.

We pretrain character embeddings on a large raw corpus, and the resulting parameter W_{source} is used to initialize the looking-up matrix E^c in our transition neural model. We use a context size by $k = 2$ and the negative example size 5 for word2vec.

4.2 Word Embeddings

We can use the same method described above to pretrain word embeddings. The only problem is the acquisition of large-scale segmented corpus, whose basic input units are words. In this work, we use our baseline discrete model to this end. First, we train a segmentation model by using the same corpus which is used to train the final neural model. And then we use the model to automatically segment the large-scale raw corpus which is used for character embeddings. This method is also exploited in our original conference version (Zhang et al., 2016). Note that the pretrained word embeddings include information from the baseline segmentation model, which is a statistical segmentor. Thus the exploration of word embeddings in our model can be viewed as a form of uptraining, which intuitively demonstrates the effectiveness of our neural model.

There exists one issue when using such word embeddings. In our transition-based neural model, the last word on the stack is not always a full word, while the auto-segmented corpus contains only full words. Thus we are only able to pretrain full word embeddings, which leaves our model uninformed about partial words in the incremental segmentation process, potentially confusing unfinished words with full words. For example, the resulted embeddings do not include the meanings of "中"(China) and "外"(foreign) as prefix subwords which are intuitively very useful in our neural model. In addition, a number of subwords which have never occurred as full words are OOV embedding words. One simple method is to model these subwords by zero vectors, which is used in Zhang et al. (2016). The method can be problematic because it does not differentiate different subwords.

Subword embeddings. Here we present a novel method to train mixed word/subword embeddings based on the auto-segmented corpus. Our idea is motivated by the extension of the skip-gram model by Levy and Goldberg (2014), which allows the model to consider arbitrary context features. Given one segmented sentences $w_1 w_2 \cdots w_n$, the standard way to pretrain full word embeddings exploiting context features from the surrounding words. For example, we can construct four context features w_{i-2}, w_{i-1}, w_{i+1} and w_{i+2} for word w_i. This is equal to the basic method by using the word2vec tool. In order to pretrain subword embeddings as well, we decompose a full word into prefixes and suffixes in this article, producing additional sentential sequences with subwords. Given one sentence $w_1 \cdots w_n$, assuming that the focus word $w_i = c_i \cdots c_k \cdots c_j$ $(j > k \geq i)$, we produce a new sequence with subwords by splitting the word at the inner character position k: $w_1 \cdots w_{i-1} w_{i,\text{p}} w_{i,\text{s}} w_{i+1} \cdots w_n$, where $w_{i,\text{p}} = c_i \cdots c_k$ and $w_{i,\text{s}} = c_{k+1} \cdots c_j$. Then we can extract the following context features to train subword embedding for $w_{i,\text{p}}$: $(1) w_{i-2} \circ \text{F}$, $(2) w_{i-1} \circ \text{F}$, $(3) w_{i,\text{s}} \circ \text{S}$ and $(4) w_{i+1} \circ \text{F}$, and for the fullword w_{i-1} (other related full words

Method	Sentence	Example instances for word2vec pretraining	
		Source	Context features
baseline	今年 学术 **研讨会** 如期 举行	**研讨会**	今年∘F 学术∘F 如期∘F 举行∘F
ours	今年 学术 **研讨会** 如期 举行	**研讨会**	今年∘F 学术∘F 如期∘F 举行∘F
	今年 学术 **研 讨会** 如期 举行	**研**	今年∘F 学术∘F **讨会**∘S 如期∘F
		学术	start∘F 今年∘F **研**∘P **讨会**∘S
		如期	**研**∘P **讨会**∘S 举行∘F end∘F
	今年 学术 **研讨 会** 如期 举行	**研讨**	今年∘F 学术∘F **会**∘S 如期∘F
		学术	start∘F 今年∘F **研讨**∘P **会**∘S
		如期	**研讨**∘P **会**∘S 举行∘F end∘F

Table 2: Comparison between the baseline word embedding method and our method with subword embedding pretraining, where the input sentence is "今年(this year) 学术(academy) 研讨会(seminar) 如期(on schedule) 举行(hold)". "start" and "end" are two pseudo words to mark the beginning and ending of a sentene, and here we focus on the word "研讨会".

are similar), we extract features (1)$w_{i-3} \circ$ F, (2)$w_{i-2} \circ$ F, (3)$w_{i,\mathrm{p}} \circ$ P and (4)$w_{i,\mathrm{s}} \circ$ S, where F, P, S denote full word, prefix and suffix, respectively.

There are three things to note. First, we treat full words and subwords equally when they are used as source embeddings (Levy & Goldberg, 2014), while as contextual features, full words, prefix subwords and suffix subwords are treated differently by the corresponding attribute marks. Second, we only pretrain embeddings for full words and prefix subwords, since we never use suffix subwords in the transition-based word segmentation models. Third, in order to have a large coverage, we extract all possible prefix subwords for each word when pretraining subword embeddings.

Table 2 shows an example to illustrate the differences between the baseline pretraining method of Zhang et al. (2016) and our improved method with subword pretraining in this article. We list only several representative instances. For the baseline method, all context features are full words, and the mark F is just used for clear comparison. We use the extended word2vec tool[2] by Levy and Goldberg (2014) to train the new word embeddings.

5. Training

We exploit online learning with early-update (Zhang & Clark, 2011) to train parameters for both the discrete and neural models as shown in Algorithm 2. Training is performed over each training instance. First, we apply a standard beam search algorithm to segment an input sentence based on current model parameters Θ. For each state in the beam (agenda), we can derive two subsequent states by applying SEP and APP, respectively. We keep only the top B states with highest scores. Different from the decoding algorithm of the testing phase, we add a margin η during decoding if one action is incorrectly predicted, leading to wrong states with higher scores. The margin ensures the score of the gold-standard action

2. https://github.com/BIU-NLP/word2vecf

Algorithm 2 Online learning with early-update.

function TRAIN($c_1 \cdots c_n$, $a^g_1 \cdots a^g_n$, Θ)
  agenda $\leftarrow$ { (ϕ (empty stack), $c_1 \cdots c_n$ (queue), score=0.0) }
  list $\leftarrow$ { }
  $\text{ST}^g_0 \leftarrow$ { ϕ, $c_1 \cdots c_n$}
  for k **in** $1 \cdots n$
    CLEAR(*list*)
    for *candidate* **in** *agenda*
      new $\leftarrow$ APPLY(SEP, *candidate*, c_k, Θ)
      if $\{a^g_k \neq \text{SEP}\}$ new.*score* += η
      ADDITEM(list, new)
      new $\leftarrow$ APPLY(APP, candidate, c_k, Θ)
      if $\{a^g_k \neq \text{APP}\}$ *new*.score += η
      ADDITEM(*list*, *new*)
    agenda $\leftarrow$ TOP-B(*list*, *B*)
    $\text{ST}^g_k \leftarrow$ APPLY(a^g_k, ST^g_{k-1})
    if $\{\text{ST}^g_k \notin \text{agenda}\}$ **break**
  APPLYLOSS(*list*, ST^g_k)
  BACKPROPAGATION()
  MODELUPDATE(Θ)

be η greater than the highest-scored incorrect action after training is finished. In particular, when the gold-standard state is pruned off the agenda, a model update is executed.

We compute loss by contrasting the gold-standard state against the negative states at the last step. Standard back-propagation is performed. In this work, we investigate two widely-used training objective functions, namely max-margin and max-likelihood, respectively. Max-margin training has been extensively exploited in transition-based models (Huang & Sagae, 2010; Zhang & Clark, 2011; Zhu, Zhang, Chen, Zhang, & Zhu, 2013; Zhang, Zhang, Che, & Liu, 2014a), while max-likelihood training is recently used for transition-based neural models (Zhou et al., 2015), which has been analyzed both empirically and theoretically by Andor et al. (2016).

Here we denote the two training methods in one framework, shown by Algorithm 2, which are different in the loss computation. For max-margin training, the loss function is defined as:

$$\begin{aligned} l(\text{ST}^g_k, \Theta) &= \max_{\text{ST}_k} \big(score(\text{ST}_k) + \eta \cdot \sum_{i=1}^{k} \delta(A_i, A^g_i) \big) - score(\text{ST}^g_k) \\ &= \max_{\text{ST}_k} \big(\overline{score}(\text{ST}_k) \big) - score(\text{ST}^g_k), \end{aligned} \tag{9}$$

where $A_1 \cdots A_k$ is the action sequence to generate ST_k, $A^g_1 \cdots A^g_k$ is the action sequence to generate ST^g_k (the gold-standard state at step k), $\delta(\cdot)$ denotes the Hamming distance between the two input sequences and $\overline{score}(\text{ST}_k)$ refers to the score computed by using Algorithm 2 with the margin η. We follow previous work exploiting subgradients to approximate the real gradients, since the above objective function is non-differentiable. The

Algorithm 3 Gradient computation for the max-margin objective.

function ApplyLoss-MaxMargin(*list,* ST_k^g)
  best ← BestItem(*list*)
  ST_k^g.gradient = -1
  best.gradient = 1

Algorithm 4 Gradient computation for the max-likelihood objective.

function ApplyLoss-LikeLihood(*list,* ST_k^g)
  scores ← { }
  for i **in** $1 \cdots$ sizeof(*list*)
    scores.append(list_i.score)
  $\boldsymbol{p}$ = SoftMax(scores)
  for i **in** $1 \cdots$ sizeof(*list*)
    list_i.gradient = $\boldsymbol{p}_i$ - IsGold (list_i)

gradients of the loss function with respect to the state scores can be computed by Algorithm 3, where the gradient of the gold-standard state is set as -1.0 and the gradient of the highest-score state is set as 1.0 directly.

For the max-likelihood objective, the loss function is computed as:

$$\begin{aligned} l(\mathrm{ST}_k^g, \Theta) &= -\log \mathrm{Prob}(\mathrm{ST}_k^g) \\ &= -\log \frac{\exp\big(score(\mathrm{ST}_k^g)\big)}{\sum_{\mathrm{ST}_k} \exp\big(score(\mathrm{ST}_k)\big)} \\ &= -\log \frac{\exp\big(score(\mathrm{ST}_k^g)\big)}{\sum_{\mathrm{ST}_k \in \mathrm{list}} \exp\big(score(\mathrm{ST}_k)\big)} \\ &= \log \sum_{\mathrm{ST}_k \in \mathrm{list}} \exp\big(score(\mathrm{ST}_k)\big) - score(\mathrm{ST}_k^g), \end{aligned} \tag{10}$$

where $\mathrm{Prob}(\mathrm{ST}_k^g)$ denotes the probability of the gold-standard state, computed by normalizing over all possible states. Since the number of possible states increases exponentially by the step, we use the states in the *list* of Algorithm 2 to approximate the sum. The corresponding gradients of the loss function to the state scores can be computed by Algorithm 4. First we apply the softmax function to obtain the probabilities of all states in the *list*, then the state gradient is calculated directly by the state probability subtracting with a one-zero value indicating whether it is a gold-standard state (one) or not (zero). In particular, the value of η should be zero when the max-likelihood training is exploited.

To avoid overfitting, we augment the objective with one additional L_2 regularization term, thus our final objective can be formalized as follows:

$$L(\Theta) = l(\mathrm{ST}_k^g, \Theta) + \frac{\lambda}{2} \parallel \Theta \parallel^2 \tag{11}$$

where λ is a regularization parameter. Our goal is to minimize the above loss function over all training examples. For discrete models, the only model parameter is $\boldsymbol{\theta}$, while for neural

network models the model parameters include $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}$ and $\boldsymbol{b}$, as well as the looking-up matrixes $\boldsymbol{E}$. We use standard back-propagation to learn the gradients from the loss function (LeCun, Bottou, Orr, & Müller, 2012), and online AdaGrad (Duchi, Hazan, & Singer, 2011) to minimize the objective function for both the discrete and neural models. All the matrix and vector parameters are initialized by uniform sampling in $(-0.01, 0.01)$.

6. Experiments

We conduct much extended experiments based on the conference version (Zhang et al., 2016). In addition to showing the original results on CTB6, PKU and MSR, the following is discussed. First, we test the out-of-vocabulary (oov) performance of various models, and also evaluate model performances on one out-of-domain dataset `Zhuxian`. Second, we compare two different training objectives, namely max-margin and max-likelihood. Third, we test the effectiveness of our proposed subword embeddings, and in addition perform feature ablation analysis to evaluate several potentially useful features as well. Last, we conduct deep analysis for discrete and neural feature combination, showing the optimal combination hyper-parameter and explaining the performance gains from two different aspects.

6.1 Experimental Settings

Data. We use three benchmark datasets for evaluation, namely CTB6, PKU and MSR. The CTB6 corpus is taken from the Penn Chinese Treebank 6.0, and the PKU and MSR corpora can be obtained from BakeOff 2005 (Emerson, 2005). For Chinese word segmentation, there are several granularity criteria to segment one sentence, and CTB6, PKU and MSR take three styles of segmentation, each having a guideline. We follow Zhang et al. (2014a), splitting the CTB6 corpus into training, development and testing sections. For the PKU and MSR corpora, only the training and test datasets are specified and we randomly split 10% of the training sections for development. Additionally, we evaluate the cross-domain performances as well, using the annotated data of Zhang, Zhang, Che, and Liu (2014b), which takes the raw corpus of the free Internet novel "Zhuxian" as basic source, and follows the same segmentation guideline of CTB6 (Xue, Xia, Chiou, & Palmer, 2005). Thus we use the model trained on CTB6 to test our proposed neural model. Table 3 shows the overall statistics of the four datasets.

The Chinese Gigaword corpus (LDC2011T13) is used to pretrain character and word embeddings. We clean the data and remove several noisy sentences (i.e. the percentage of Chinese characters in one sentence is below 20% and the sentence length is lower than 3 or larger than 200), and a total of 32 million sentences are kept. All datasets including annotated and raw corpora are preprocessed by replacing all the Chinese characters into simplified characters, and as well by replacing all full-width characters into half-width characters. As a whole, we obtained vector representations for 11,327 characters, 2,130,331 full words and 7,544,738 full/sub words (additional $7{,}544{,}738 - 2{,}130{,}331 = 5{,}414{,}407$ subwords by our new word pretraining method).

Evaluation. We use word-level precision (P), recall (R) and their F-measure (F) as the major metrics for evaluation. Concretely, a word is treated as correct if only the predicted span is exactly matching with the gold-standard span. In addition, in order to test the

		CTB6	Zhuxian	PKU	MSR
Training	#sent	23k		17k	78k
	#word	641k		1,010k	2,122k
	#characters	1,056k		1,663k	3,633k
	average word length	1.65		1.65	1.71
Development	#sent	2.1k	0.8k	1.9k	8.7k
	#word	60k	20k	100k	246k
	#characters	100k	28k	164k	417k
	average word length	1.67	1.39	1.64	1.69
Test	#sent	2.8k	1.4k	1.9k	4.0k
	#word	82k	34k	104k	106k
	#characters	134k	48k	173k	184k
	average word length	1.64	1.40	1.66	1.73

Table 3: Statistics of datasets.

model performance on unknown words, we use another metric R_{oov}, where one word is treated as out-of-vocabulary if it has never occurred in the training corpus.

Hyper-parameters. The hyper-parameter values are tuned according to preliminary results on the development corpus. We set the dimension size of the basic input character embeddings and word embeddings to 50. The dimension sizes of all the hidden layers of the neural model are set to 100. Finding no significant improvement when the sizes become larger, we use this value for simplicity and efficiency. The initial learning rate for Adagrad is set to 0.01, the regularization term in the training objective is set to 10^{-8}, and the value of η in max-margin training is set to 0.2. The models are not highly sensitive to the above hyper-parameters, with the performances remaining stable when they are set within certain ranges. There is one hyper-parameter, the iteration number of training, that we must tune carefully. We train different models on the corresponding training datasets for 20 iterations, and select the best iteration model according to their development performances.

6.2 Development Results

We perform several development experiments on the CTB6 development dataset.

6.2.1 Embeddings and Beam Size

We study the influence of beam size on the baseline and neural models. Here, we can either fine-tune or fix the embeddings during training. In case of fine-tuning, only characters and words in the training data can be learned, while the out-of-vocabulary (OOV) embeddings could not be used effectively. In addition, following Dyer, Ballesteros, Ling, Matthews, and Smith (2015) we randomly set the low-frequency characters and words (i.e. characters occur less than 2 times and words occur less than 4 times) in the training data as OOVs in order to learn OOV character and word embeddings, while avoiding overfitting. For the CTB6 dataset, if fine-tuning is adopted, the vocabulary sizes of characters and words are 3,791 and 115,537 (prefix subwords are included), respectively. If the pretrained embeddings are not fine-tuned, we can utilize all embeddings learned from the large raw auto-segmented corpus,

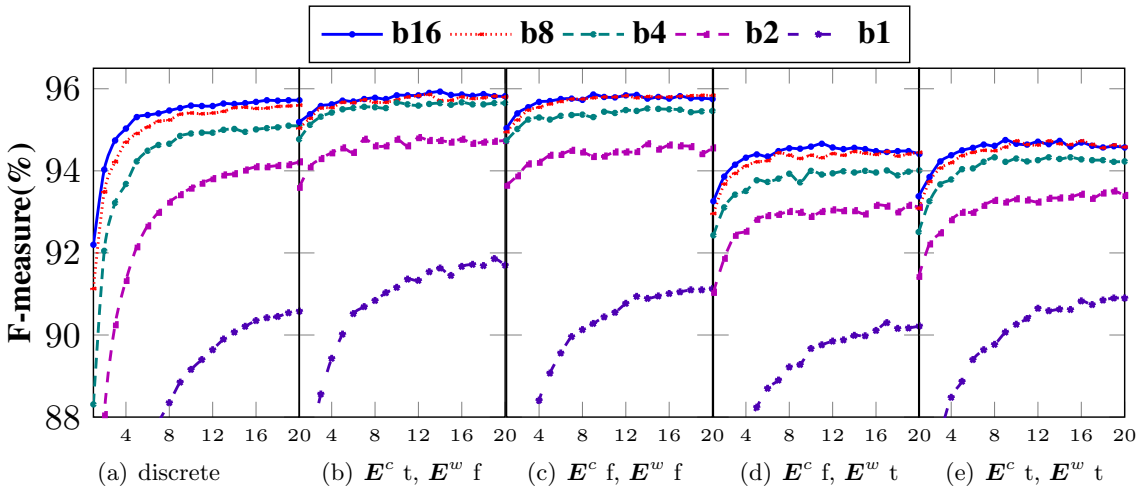


Figure 5: F-measures against the training epoch using beam sizes 1, 2, 4, 8 and 16, respectively, where t and f denote tuning and not tuning (fixing) the corresponding look up matrix.

thus the vocabularies are the same as pretrained word2vec results (character vocabulary size: 11,327, word vocabulary size: 7,544,738).

Figure 5 shows the development results, where the training curves of the discrete baseline are shown in Figure 5(a) and these of the neural model without and with fine tuning are shown by Figure 5(b-e), respectively. We show the development performances for every epoch over the entire training set. All the models are trained by the max-margin objective, as this method can give better performances. The trends of the discrete and neural models in term of different beam sizes are highly similar. The performance increases with a larger beam size in all settings, which demonstrates the usefulness of beam-search. When the beam increases into 16, the gains levels out.

From the curves, we can see that fine-tuning *character* embeddings impacts the neural models marginally, while fine-tuning *word* embeddings can bring significant changes on the development performances. The main reason can be that the total number of Chinese characters is relatively small, and most of them are covered by the training corpus. Thus fine-tuning does not lead to much loss of pretrained embedding information. On the other hand fine-tuning does not bring much gains either. However, it is a very different case for word embeddings, for which the vocabulary size is much larger than that of the segmentation training corpus. By using fine-tuning, we are unable to exploit the pretrained embeddings of OOV words (98.5% of the total vocabulary size) properly. These embeddings encode information from our baseline discrete model, which could be intuitively useful for Chinese word segmentation.

Further, we consider another scenario where randomly initialized embeddings are exploited. In this situation, we must fine-tune the embeddings to obtain good performances. Thus we investigate the scenario by fine-tuning both character and word embeddings. We

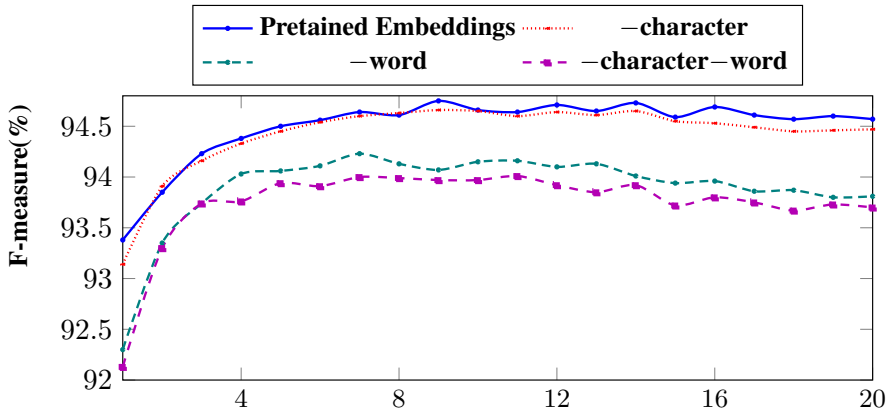


Figure 6: F-measures against the training epoch by using pretrained/random embeddings.

fix the beam size to 16. Figure 6 shows the results. By removing pretrained character embeddings, the performances show slight drops. But the performances drop more significantly when pretrained word embeddings are replaced with random embeddings. The results show that pretrained embeddings are useful for the neural model, which is consistent with observations by previous work (Pei et al., 2014; Chen et al., 2015b).

Based on the above observation, we set the beam size of all the transition-based models to 16, and exploit fine-tuned character embeddings and fixed word embeddings in our final models. More stable and better performances can be obtained by this setting.

In particular, here we use a single model to pretrain word and subword embeddings, projecting the two types of embeddings into a shared vector space. This is important: If the word and subword embeddings are pretrained separately, we find that the performance decreases significantly. Based on the final selected setting, the F-measure value drops from 95.93 to 93.54, which is reasonable as a single word-level LSTM receives inputs from two different vector spaces. We have exploited one alternative, using two separate word-level LSTMs for full words and subwords, respectively. The two LSTMs are computed at the SEP actions and APP actions, resepctively. The F value of this model is 95.86, which is not significantly different from our final model, while our model is simpler.

6.2.2 Training Objective

We investigate the influence of different training objectives on our transition-based neural model. Figure 7 shows the results. A max-likelihood training objective leads to quick convergence. However, the F-measure value decreases gradually after the fourth iteration, which may be due to overfitting on the training dataset. The convergence speed by using max-margin training is slightly slower, but after more iterations, max-margin training gives better performances on the developmental dataset. We thus adopt max-margin training in the transition-based neural word segmentation models.

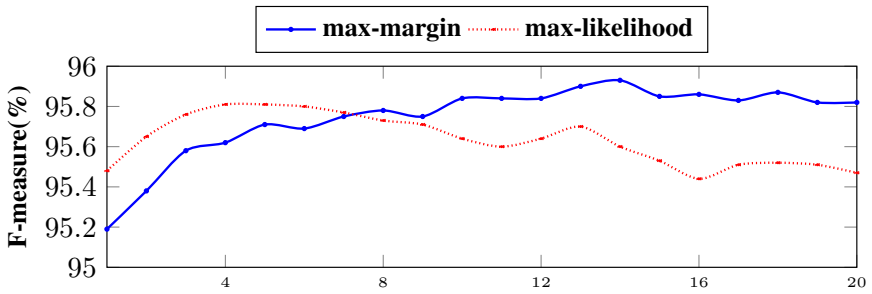


Figure 7: F-measures against the training epoch by using different training methods.

Model	Description	P	R	F	△F
neural	final neural model	96.01	95.86	95.93	—
core features					
-character	removing character LSTM	95.72	95.60	95.66	-0.27
-subword	only full word embeddings	94.87	94.91	94.89	-1.04
-word	removing word LSTM	90.80	90.90	90.85	-5.08
-LSTM(+CONV)	convolution, character/word window: 5/2	95.28	95.52	95.40	-0.53
redundant (less informative) features					
+character bigram	embedding	95.96	95.78	95.87	-0.06
+character category	embedding	96.09	95.74	95.91	-0.02
+word length	embedding	95.92	95.92	95.92	-0.01
+first character	fetching from character LSTM	95.88	95.92	95.90	-0.03
+all included characters	average pooling over character LSTM	95.98	95.92	95.95	+0.02
+action	another LSTM (Zhang et al., 2016)	96.01	95.79	95.90	-0.03

Table 4: Feature ablation experiments.

6.2.3 Feature Ablation

We conduct feature ablation experiments to study the effects of the word and character embedding features to the neural model. The results are shown in Table 4. Character embeddings (`-char`) contribute only 0.27% to the neural segmentation model, similar to our finding in Zhang et al. (2016). The role of character embeddings is not as important as demonstrated by previous character-based neural models (Pei et al., 2014; Chen et al., 2015b). One possible reason can be that the global word-level information alone is sufficient for action prediction in most contexts. The observation indicates that we can obtain competitive performances in the transition-based model as well by using pure word-level features. One potential benefit is that the efficiency of our model, which can save the time cost by the character LSTM. The developmental speed is increased from 40.0 to 57.7 sentences per second after character LSTM is removed.

`-subword` denotes the exploration of pretrained word embeddings based only on full words, which is the same as our original conference paper (Zhang et al., 2016). Without embeddings of subwords, the performance drops significantly, by close to 1%. By removing

both full and sub word embeddings (-word), our transition-based neural model gives an F-measure of only 90.85%, nearly 5% below of the final development performance, demonstrating the importance of word-level features. Intuitively, we can explain the effectiveness of the word-level LSTM by the example in Figure 2(b). As shown in step 4, the last word "外企(foreign company)" is a highly important clue for predicting the next word "业务(business)".

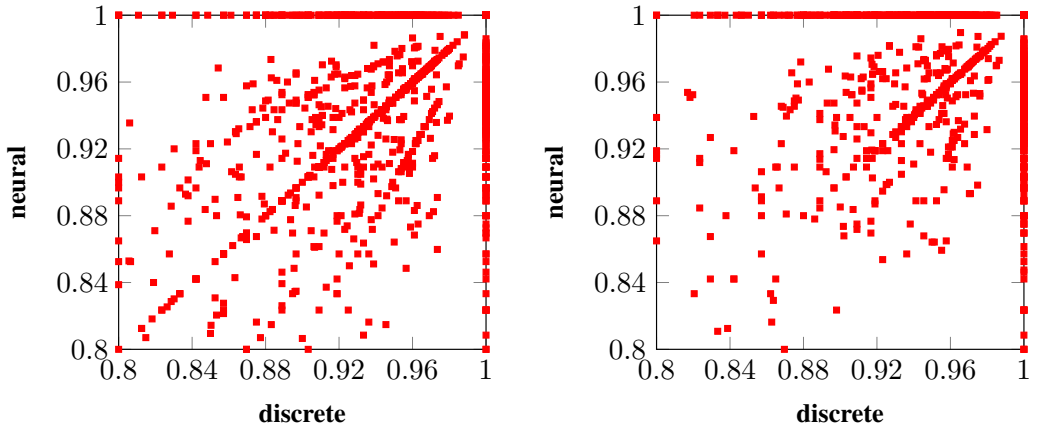
-LSTM(+CONV) is used to verify the effectiveness of LSTM. The major benefit of LSTM is that it can capture long distance connections inner an input sequence. Here we replace all LSTM neural structures by convolutional structures. We adopt a window size of 5 ($c_{q_0-2} \cdots c_{q_0+2}$) for encoding character sequences, and a window size of 2 (w_{s_0} and w_{s_1}) for encoding word sequence, which includes similar local information as the discrete baseline. According to Table 4, removing LSTM reduces the final performances by 0.53%, indicating the usefulness of LSTM structures. In particular, we find that the transition-based neural model by convolutional neural structures performs worse than the baseline discrete model (convolution: 95.40 v.s. discrete: 95.72). Considering that both models exploit similar local features, the observation further verifies the importance of long distance features for Chinese word segmentation.

In addition, we also test the effects of other potential informative features, including: (1) character-level information such as character bigram embeddings, character category embeddings, which are exploited by concatenated with character embeddings for character-level LSTM; (2) word-level features from the last word such as length embedding, vector representation of the first character derived from the character LSTM, and vector representation of all included characters by averaging over their character LSTM outputs, which are exploited by concatenation with word embeddings; and (3) action-level LSTM features, which have been used in our original conference paper (Zhang et al., 2016) but shown ineffective by Yang et al. (2017). The ineffectiveness of the action-level LSTM has been observed in Kuncoro, Ballesteros, Kong, Dyer, Neubig, and Smith (2017) also, who perform joint syntax parsing and language modeling.

We find that these features do not significantly influence our transition-based neural model. The obversion is slightly different from the conference version (Zhang et al., 2016), for example, the embeddings of character bigrams and the action-level LSTM features (i.e., the two kinds of features lead to increases of 0.22% and 0.28% on the same data in the conference version, respectively). For the former, one possible reason may be that the enhanced word-level embeddings now cover subwords as well, which makes character bigrams less informative, since they have overlapping information. For the latter, the possible reason can include: (1) the actions are simple, (2) there is no apparent distance dependency among the action sequence and (3) existing word LSTM features already cover action sequence information. According to the results, we conclude that the proposed neural model is able to achieve state-of-the-art performances based on a highly concise neural network structure.

6.2.4 Discrete and Neural Features

Prior work has shown the effectiveness of integrating discrete and neural features for several NLP tasks (Turian, Ratinov, & Bengio, 2010; Wang & Manning, 2013; Durrett & Klein, 2015; Zhang & Zhang, 2015). We investigate the usefulness of such integration to our word-



(a) Compared against the gold segmentations.

(b) Compared against the predicted segmentations of the combined model

Figure 8: Sentence-level F-measure comparisons for the discrete and neural models.

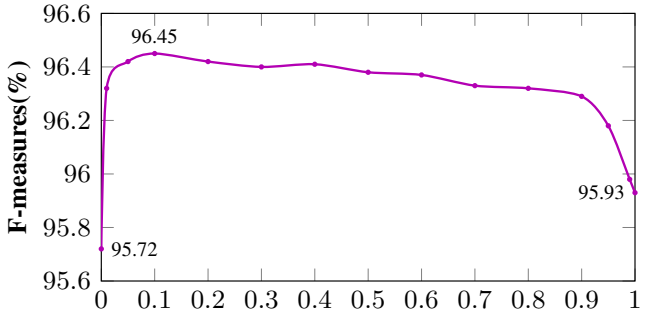


Figure 9: Integration performances with respect to the scale hyper-parameter α.

based segmentor on the development dataset. In particular, we conduct two experiments. First, we compare the error distributions between the discrete and the neural models. Intuitively, different error distributions are necessary for improvements by integration. We draw a scatter graph to show their differences, with the (x, y) values denoting the F scores of the two models with respect to sentences, respectively. As shown in Figure 8(a), the points are rather scattered, showing the differences of the two models. In other words, the discrete model and the neural model are expertise under different contexts, being complementary with each other. Thus a combination may result in better performances.

Second, we directly examine the results by integrating discrete and neural features. As shown in Equation 7, we have a hyper-parameter α to scale the scores of the two separate feature sets. Figure 9 shows the performances. When $\alpha = 0$ or $\alpha = 1$, the single model performances are reported. We can see that the integration model achieves the best performance when $\alpha = 0.1$, and the concrete value is 96.45, showing an increase of 0.52% over the neural model, which is the better one of the two individual models.

Models	P	R	F	R_{oov}
our transition-based word models				
discrete	95.35	95.20	95.28	73.95
neural	95.77	95.47	95.62	76.58
combined	**96.02**	**96.06**	**96.04**	**80.18**
character-based CRF models				
discrete	95.14	95.20	95.17	70.68
neural	95.47	95.45	95.46	74.16
combined	95.69	95.80	95.74	76.95
other models				
(Liu et al., 2016)	N/A	N/A	95.48	N/A
(Zhang et al., 2014a)	N/A	N/A	95.71	N/A
(Wang et al., 2011)	95.83	95.75	95.79	N/A
(Zhang & Clark, 2011)	95.46	94.78	95.13	71.18

Table 5: Main results on CTB6 test dataset.

In order to study the similarities between the combined model and the individual models, we draw a scatter graph of the two models in Figure 8(a) using the combined segmentation outputs as the gold-standard reference. Intuitively, the F scores can be regarded as similarities between two different outputs. As shown in Figure 8(b), it is difficult to see any bias of the combined model favored the discrete or neural models, since the points in the figure are also highly scattered.

6.3 Final Results

Table 5 shows the final results on the CTB6 test dataset. The transition-based models using neural features achieve better performances than those using discrete features. The observation is different from our conference paper (Zhang et al., 2016), thanks to the enhanced pretrained word embeddings by the neural model. The combined model with both discrete and neural features gives the best performances, which is consistent with our developmental findings and our conference paper. We report the models' performance in recognizing OOV words as well. The results show that the neural models can give better OOV performances, which possibly benefits from the pretrained word embeddings.

For a more thorough comparison, we implement discrete, neural and combined *character-based* Chinese word segmentation models as well.[3] In particular, the character-based discrete model is a CRF tagging model using character unigrams, bigrams, trigrams and tag transitions (Tseng et al., 2005), and the character-based neural model exploits a bi-directional LSTM layer to model character sequences and a CRF layer for output (Chen et al., 2015b).[4] The bi-directional character LSTM is different from our transition-based neural model in that character bigram embeddings are also exploited as inputs to LSTMs,

3. Publicly available under Apache License 2.0 at https://github.com/zhangmeishan/NNCRFSegmentor.
4. Bi-directional LSTM is slightly better than a single left-right LSTM used in Chen et al. (2015b).

Models	PKU		MSR	
	F	R_{oov}	F	R_{oov}
our transition-based word models				
discrete	95.14	62.18	97.17	65.26
neural	95.62	67.20	97.03	71.14
combined	95.93	**69.87**	**97.87**	**74.13**
character-based CRF models				
discrete	95.00	58.24	96.83	56.72
neural	95.34	61.38	97.30	72.11
combined	95.68	68.36	97.44	73.27
other models				
(Liu et al., 2016)	95.67	—	97.58	—
(Cai & Zhao, 2016)	95.5	—	96.5	—
(Ma & Hinrichs, 2015)	95.1	—	96.6	—
(Pei et al., 2014)	95.2	—	97.2	—
(Zhang et al., 2013)	**96.1**	—	97.5	—
(Sun, Wang, & Li, 2012)	95.4	—	97.4	—
(Zhang & Clark, 2011)	95.12	60.58	97.25	67.38
(Sun, 2010)	95.2	—	96.9	—
(Sun et al., 2009)	95.2	—	97.3	—

Table 6: Main results on PKU and MSR test datasets.

because our experiments show that character bigram embeddings are highly useful for character-based neural models, without which the performances can drop significantly. The combined model uses the same method as the model integration of discrete and neural features for our transition-based model.

The transition-based word models achieve better performances than the character-based CRF models, since the models can exploit additional word information learnt from large auto-segmented corpus. We also compare the results with other methods in the literature. In particular, character-based CRF models perform poorly on OOV words compared with their word-based counterparts, which could be a problem for out-of-domain testdata. (Wang et al., 2011) is a semi-supervised model that exploits word statistics from auto-segmented raw corpus, which is similar with our combined model in using semi-supervised word information. (Zhang et al., 2014a) is a joint segmentation, POS-tagging and dependency parsing model, which can exploit syntactic information. We achieve slightly better performances compared with these methods.

Finally, we report performances on the PKU and MSR datasets also. As shown in Table 6, our combined model gives the best result on the MSR dataset, and the second best on PKU. The method of (Zhang et al., 2013) gives the best performance on PKU by co-training on large-scale data. The results of (Chen et al., 2015a) and (Chen et al., 2015b) are not listed, because they take a preprocessing step by replacing Chinese idioms with a uniform symbol in their test data, which can greatly influence the fairness of model comparisons. We only list the OOV performances of the models implemented by us, since

Models	P	R	F	R_{oov}
our transition-based word models				
discrete	88.30	86.56	87.42	65.06
neural	86.33	87.09	86.71	67.05
combined	87.82	**90.23**	**89.01**	**75.26**
character-based CRF models				
discrete	87.80	86.73	87.26	65.21
neural	86.58	86.44	86.51	66.49
combined	**88.54**	88.28	88.41	73.94
other models				
(Zhang & Clark, 2011)	88.42	87.13	87.77	66.88

Table 7: Main results on Zhuxian test dataset.

we find that different work usually have different preprocessing techniques, which causes the OOV performances incomparable.

Cross domain. We evaluate the model performances on an out-of-domain dataset in order to test the robustness of the transition-based neural models For these tests, we train models on the CTB6 training and development datasets, reporting their performances on the Zhuxian test dataset, whose segmentation style is the same as CTB6. Table 7 shows the detailed results. For the transition-based models, neural features can result in better performances on OOV words. For example, the domain words such as "**烧火**棍"(fire stick) and "**小池**镇"(Xiao Chi Town) can be correctly recognized by the transition-based neural model with more cases than other models. These words are extremely difficult because they have subwords (the bold part) which could be full words as well. In the transition-based neural model, subword embeddings can be helpful for recognition of these words. Again, the integrated transition-based model gives the best performance on the dataset.

6.4 Comparing Word-Based and Character-Based Neural Models

To study the differences between word-based and character-based neural models, we conduct error analysis on the test dataset of CTB6. First, we examine the error distribution on individual sentences. Figure 10 shows the F-measure values of each test sentence by word- and character-based neural models, respectively, where the x-axis value denotes the F-measure value of the word-based neural model, and the y-axis value denotes its performance of the character-based neural model. We can see that the majority scatter points are off the diagonal, demonstrating strong differences between the two models. This results from the differences in feature sources.

Second, we study the F-measure distribution of the two neural models with respect to the sentence length by the number of words. We divide the test sentences into eleven bins, with bin i denoting sentence lengths in $[5*(i-1), 5*i]$. The number of sentences in each bin ranges from 120 to 380. Figure 11 shows the results. According to the figure, we observe that the word-based neural model is relatively weaker for sentences with length in $[1, 10]$, while better tackling long sentences.

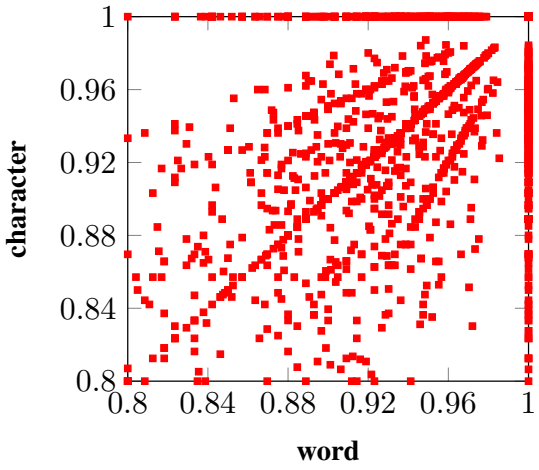


Figure 10: Sentence-level F-measure comparisons for word- and character-based neural models.

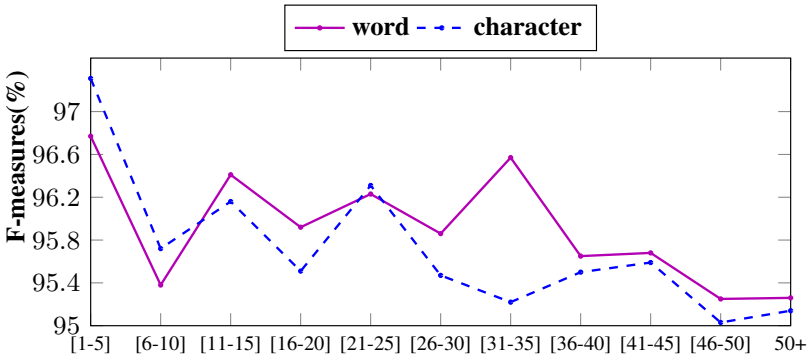


Figure 11: F-measure against sentence length.

Third, we compare the two neural models by their capabilities of modeling words with different lengths. Figure 12 shows the results. The performances are lower for words with lengths beyond 2, and the performance drops significantly for words with lengths over 3. Overall, the word-based neural model achieves comparable performances with the character-based model, but gives significantly better performances for long words, in particular when the word length is over 3. This demonstrates the advantage of word-level features.

7. Related Work

Xue (2003) was the first to propose a character-tagging method to Chinese word segmentation, using a maximum entropy model to assign B/I/E/S tags to each character in the input sentence separately. Peng et al. (2004) showed that better results can be achieved by global learning using a CRF model. This method has been followed by most subsequent models

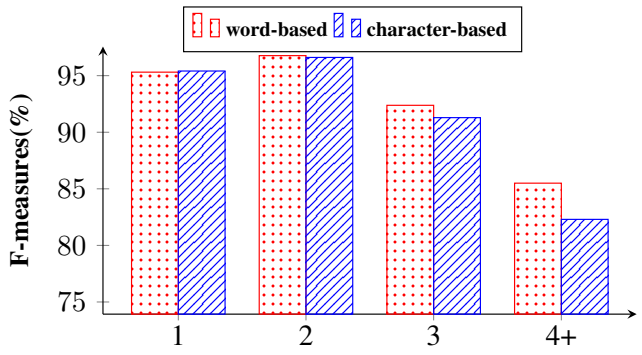


Figure 12: F-measure against word length, where the boxes with red dots denote the performances of word-based neural model, and the boxes with blue slanted lines denote character-based neural model.

in the literature (Tseng et al., 2005; Zhao, 2009; Sun et al., 2012). The most effective features have been character unigrams, bigrams and trigrams within a five-character window, and a bigram tag window. Special characters such as alphabets, numbers and date/time characters are also differentiated for extracting features.

Zheng et al. (2013) built a neural network segmentor, which essentially substitutes the manual discrete features of Peng et al. (2004), with dense real-valued features induced automatically from character embeddings, using a deep neural network structure (Collobert, Weston, Bottou, Karlen, Kavukcuoglu, & Kuksa, 2011). A tag transition matrix is used for inference, which makes the model effective. Most subsequent work on neural segmentation followed this method, improving the extraction of emission features by using more complex neural network structures.

Mansur, Pei, and Chang (2013) experimented with embeddings of richer features, and in particular character bigrams. Pei et al. (2014) used a tensor neural network to achieve extensive feature combinations, capturing the interaction between characters and tags. Chen et al. (2015a) used a recursive network structure to the same end, extracting more combined features to model complicated character combinations in a five-character window. Chen et al. (2015b) used a LSTM model to capture long-range dependencies between characters in a sentence. Xu and Sun (2016) proposed a dependency-based gated recursive neural network to efficiently integrate local and long-distance features. The above methods are all character-based models, making no use of full word information. In contrast, we leverage both character embeddings and word embeddings for better accuracies.

For word-based segmentation, Andrew (2006) used a semi-CRF model to integrate word features, Zhang and Clark (2007) used a perceptron algorithm with inexact search, and Sun et al. (2009) used a discriminative latent variable model to make use of word features. Recently, there have been several word-based neural models (Liu et al., 2016; Cai & Zhao, 2016), among which our conference version (Zhang et al., 2016) can be regarded as one representative model. Morita, Kawahara, and Kurohashi (2015) suggested a two-stage method,

using a word-level RNN language model to rerank the outputs from a baseline model, thus it is not a word-based segmentation model alone, but a seminal example using word-level neural features. Liu et al. (2016) follow Andrew (2006) using a semi-CRF for structured inference. Cai and Zhao (2016) exploit a different decoding methd to produce word sequence incrementally. Compared with these work, the transition-based framework can be more intuitive for utilizing word-level features in a left-to-right incremental processing.

We followed the global learning and beam-search framework of Zhang and Clark (2011) in building a word-based neural segmentor. The main difference between our model and that of Zhang and Clark (2011) is that we use a neural network to induce feature combinations directly from character and word embeddings. In addition, the use of a bi-directional LSTM allows us to leverage non-local information from the word sequence, and look-ahead information from the incoming character sequence. The automatic neural features are complementary to the manual discrete features of Zhang and Clark (2011). We show that our model can accommodate the integration of both types of features. This is similar in spirit to the work of Sun (2010) and Wang, Voigt, and Manning (2014), who integrated features of character-based and word-based segmentors.

Transition-based methods with beam search have been exploited for a range of other NLP tasks, including syntactic parsing (Zhang & Nivre, 2011; Zhu et al., 2013), information extraction (Li & Ji, 2014) and various joint models (Zhang, Zhang, Che, & Liu, 2013; Zhang et al., 2014a). Recently, the effectiveness of neural features has been studied for this framework, achieving success for parsing and other tasks. Representative work includes Zhou et al. (2015), Weiss, Alberti, Collins, and Petrov (2015), Watanabe and Sumita (2015) and Andor et al. (2016). In this work, we apply the transition-based neural framework to Chinese segmentation, in order to exploit word-level neural features such as word embeddings.

8. Conclusion

We proposed a word-based neural model for Chinese segmentation, which exploits not only character embeddings, but also word embeddings pre-trained from large scale corpus. A transition-based framework is used to build the neural segmentation model, where decoding is performed incrementally by predicting a sequence of transition actions. We exploited LSTM neural structures to represent both the input character sequence and the partial word sequence. The resulting hidden features are used for representations of states, based on which transition actions are predicted by a feed-forward neural layer.

We conducted experiments on several benchmark datasets, which cover segmentation styles of different granularities and domains. Results demonstrated that our proposed model achieved strong performances compared with a discrete word-based baseline and the state-of-the-art character-based neural models in the literature. This shows the usefulness of word level features in the form of word and subword embeddings for neural word segmentation. We further demonstrated that the transition-based framework can also utilize discrete features, resulting in a combined model that achieved top performances compared with previous work. Finally, we conducted extended comparisons to study the differences between word-based and character-based neural models, showing their unique characteristics.

Acknowledgments

This work is supported by National Natural Science Foundation of China (NSFC) grants 61672211 and 61602160, Natural Science Foundation of Heilongjiang Province (China) grant F2016036, Special business expenses in Heilongjiang Province (China) grant 2016-KYYWF-0183. Corresponding author: Yue Zhang, E-mail: frcchang@gmail.com.

References

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., & Collins, M. (2016). Globally normalized transition-based neural networks. In *Proceedings of the 54th ACL*, pp. 2442–2452, Berlin, Germany.

Andrew, G. (2006). A hybrid markov/semi-markov conditional random field for sequence segmentation. In *Proceedings of EMNLP*, pp. 465–472, Sydney, Australia.

Cai, D., & Zhao, H. (2016). Neural word segmentation learning for chinese. In *Proceedings of ACL 2016*.

Chen, X., Qiu, X., Zhu, C., & Huang, X. (2015a). Gated recursive neural network for chinese word segmentation. In *Proceedings of the 53nd ACL*, pp. 1744–1753.

Chen, X., Qiu, X., Zhu, C., Liu, P., & Huang, X. (2015b). Long short-term memory neural networks for chinese word segmentation. In *Proceedings of EMNLP*, pp. 1197–1206.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP*, pp. 1–8.

Collins, M., & Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd ACL*, pp. 111–118, Barcelona, Spain.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, *12*, 2493–2537.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, *12*, 2121–2159.

Durrett, G., & Klein, D. (2015). Neural crf parsing. In *Proceedings of the 53nd ACL*, pp. 302–312.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., & Smith, N. A. (2015). Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53nd ACL*, pp. 334–343.

Emerson, T. (2005). The second international chinese word segmentation bakeoff. In *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing*, pp. 123–133.

Gers, F. A., Schraudolph, N. N., & Schmidhuber, J. (2002). Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, *3*(Aug), 115–143.

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

Huang, L., & Sagae, K. (2010). Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th ACL*, pp. 1077–1086.

Kuncoro, A., Ballesteros, M., Kong, L., Dyer, C., Neubig, G., & Smith, N. A. (2017). What do recurrent neural network grammars learn about syntax?. In *Proceedings of EACL*, pp. 1249–1258.

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer.

Levy, O., & Goldberg, Y. (2014). Dependency-based word embeddings. In *Proceedings of the 52nd ACL*, pp. 302–308.

Li, Q., & Ji, H. (2014). Incremental joint extraction of entity mentions and relations. In *Proceedings of the ACL 2014*.

Liu, Y., Che, W., Guo, J., Qin, B., & Liu, T. (2016). Exploring segment representations for neural segmentation models. In *Proceedings of IJCAI 2016*.

Ma, J., & Hinrichs, E. (2015). Accurate linear-time chinese word segmentation via embedding matching. In *Proceedings of the 53nd ACL*, pp. 1733–1743.

Mansur, M., Pei, W., & Chang, B. (2013). Feature-based neural language model and chinese word segmentation. In *Proceedings of IJCNLP*, pp. 1271–1277.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *NIPS*, pp. 3111–3119.

Morita, H., Kawahara, D., & Kurohashi, S. (2015). Morphological analysis for unsegmented languages using recurrent neural network language model. In *Proceedings of the 2015 Conference on EMNLP*, pp. 2292–2297.

Pei, W., Ge, T., & Chang, B. (2014). Max-margin tensor neural network for chinese word segmentation. In *Proceedings of the 52nd ACL*, pp. 293–303, Baltimore, Maryland.

Peng, F., Feng, F., & McCallum, A. (2004). Chinese segmentation and new word detection using conditional random fields. In *Proceedings of Coling 2004*, pp. 562–568, Geneva, Switzerland.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 EMNLP*, pp. 1532–1543.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the NAACL*, pp. 2227–2237.

Shi, Y., & Wang, M. (2007). A dual-layer crfs based joint decoding method for cascaded segmentation and labeling tasks.. In *IJCAI*, pp. 1707–1712.

Sun, W. (2010). Word-based and character-based word segmentation models: Comparison and combination. In *Coling 2010: Posters*, pp. 1211–1219.

Sun, W., & Xu, J. (2011). Enhancing chinese word segmentation using unlabeled data. In *Proceedings of the 2011 Conference on EMNLP*, pp. 970–979.

Sun, X., Wang, H., & Li, W. (2012). Fast online training with frequency-adaptive learning rates for chinese word segmentation and new word detection. In *Proceedings of the 50th ACL*, pp. 253–262.

Sun, X., Zhang, Y., Matsuzaki, T., Tsuruoka, Y., & Tsujii, J. (2009). A discriminative latent variable chinese segmenter with hybrid word/character information. In *Proceedings of NAACL 2009*, pp. 56–64.

Tseng, H., Chang, P., Andrew, G., Jurafsky, D., & Manning, C. (2005). A conditional random field word segmenter for sighan bakeoff 2005. In *Proceedings of the fourth SIGHAN workshop*, pp. 168–171.

Turian, J., Ratinov, L.-A., & Bengio, Y. (2010). Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 384–394.

Wang, M., & Manning, C. D. (2013). Effect of non-linear deep architecture in sequence labeling. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pp. 1285–1291, Nagoya, Japan. Asian Federation of Natural Language Processing.

Wang, M., Voigt, R., & Manning, C. D. (2014). Two knives cut better than one: Chinese word segmentation with dual decomposition. In *Proceedings of the 52nd ACL*, pp. 193–198, Baltimore, Maryland.

Wang, Y., Kazama, J., Tsuruoka, Y., Chen, W., Zhang, Y., & Torisawa, K. (2011). Improving chinese word segmentation and pos tagging with semi-supervised methods using large auto-analyzed data. In *Proceedings of 5th IJCNLP*, pp. 309–317, Chiang Mai, Thailand.

Watanabe, T., & Sumita, E. (2015). Transition-based neural constituent parsing. In *Proceedings of the 53rd ACL*, pp. 1169–1179.

Weiss, D., Alberti, C., Collins, M., & Petrov, S. (2015). Structured training for neural network transition-based parsing. In *Proceedings of the 53rd ACL*, pp. 323–333.

Xu, J., & Sun, X. (2016). Dependency-based gated recursive neural network for chinese word segmentation. In *Proceedings of ACL 2016*.

Xue, N. (2003). Chinese word segmentation as character tagging. *International Journal of Computational Linguistics and Chinese Language Processing*, *8(1)*.

Xue, N., Xia, F., Chiou, F.-D., & Palmer, M. (2005). The penn chinese treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, *11(2)*, 207–238.

Yang, J., Zhang, Y., & Dong, F. (2017). Neural word segmentation with rich pretraining. In *Proceedings of the 55th ACL*, pp. 839–849.

Zhang, L., Wang, H., Sun, X., & Mansur, M. (2013). Exploring representations from unlabeled data with co-training for Chinese word segmentation. In *Proceedings of the EMNLP 2013*, pp. 311–321, Seattle, Washington, USA.

Zhang, M., & Zhang, Y. (2015). Combining discrete and continuous features for deterministic transition-based dependency parsing. In *Proceedings of the 2015 EMNLP*, pp. 1316–1321.

Zhang, M., Zhang, Y., Che, W., & Liu, T. (2013). Chinese parsing exploiting characters. In *Proceedings of the 51st ACL*, pp. 125–134.

Zhang, M., Zhang, Y., Che, W., & Liu, T. (2014a). Character-level chinese dependency parsing. In *Proceedings of the 52nd ACL*, pp. 1326–1336, Baltimore, Maryland.

Zhang, M., Zhang, Y., Che, W., & Liu, T. (2014b). Type-supervised domain adaptation for joint segmentation and pos-tagging. In *Proceedings of the 14th EACL*, pp. 588–597, Gothenburg, Sweden. Association for Computational Linguistics.

Zhang, M., Zhang, Y., & Fu, G. (2016). Transition-based neural word segmentation. In *Proceedings of the 54th ACL*, pp. 421–431.

Zhang, Y., & Clark, S. (2007). Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of the 45th ACL*, pp. 840–847, Prague, Czech Republic.

Zhang, Y., & Clark, S. (2011). Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, *37*(1), 105–151.

Zhang, Y., & Nivre, J. (2011). Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th ACL*, pp. 188–193.

Zhao, H. (2009). Character-level dependencies in chinese: Usefulness and learning. In *Proceedings of the EACL*, pp. 879–887, Athens, Greece.

Zhao, H., Huang, C.-N., Li, M., & Lu, B.-L. (2006). Effective tag set selection in chinese word segmentation via conditional random field modeling. In *Proceedings of PACLIC*, Vol. 20, pp. 87–94. Citeseer.

Zheng, X., Chen, H., & Xu, T. (2013). Deep learning for Chinese word segmentation and POS tagging. In *Proceedings of the 2013 Conference on EMNLP*, pp. 647–657, Seattle, Washington, USA.

Zhou, H., Zhang, Y., Cheng, C., Huang, S., Dai, X., & Chen, J. (2017). A neural probabilistic structured-prediction method for transition-based natural language processing. *Journal of Artificial Intelligence Research*, *58*, 703–729.

Zhou, H., Zhang, Y., Huang, S., & Chen, J. (2015). A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd ACL*, pp. 1213–1222.

Zhu, M., Zhang, Y., Chen, W., Zhang, M., & Zhu, J. (2013). Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st ACL*, pp. 434–443.