# Transition-Based Neural Word Segmentation

**Meishan Zhang**[1] and **Yue Zhang**[2] and **Guohong Fu**[1]
1. School of Computer Science and Technology, Heilongjiang University, Harbin, China
2. Singapore University of Technology and Design
mason.zms@gmail.com,
yue_zhang@sutd.edu.sg,
ghfu@hotmail.com

## Abstract

Character-based and word-based methods are two main types of statistical models for Chinese word segmentation, the former exploiting sequence labeling models over characters and the latter typically exploiting a transition-based model, with the advantages that word-level features can be easily utilized. 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 word-based segmentation framework. Experimental results demonstrate that word features lead to comparable performances to the best systems in the literature, and a further combination of discrete and neural features gives top accuracies.

## 1 Introduction

Statistical word segmentation methods can be categorized character-based (Xue, 2003; Tseng et al., 2005) and word-based (Andrew, 2006; Zhang and Clark, 2007) 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 et al., 2004; Tseng et al., 2005) uses B/I/E/S labels to mark a character as the beginning, internal (neither beginning nor end), end and only-character (both beginning and end) of a word, respectively, employing conditional random field (CRF) to model the correspondence between the input character sequence and output label sequence. For each character, features are extracted from a five-character context window and a two-label history window. Subsequent work explores different label sets (Zhao et al., 2006), feature sets (Shi and Wang, 2007) and semi-supervised learning (Sun and Xu, 2011), reporting state-of-the-art accuracies.

| | character-based | word-based |
|---|---|---|
| discrete | Peng et al. (2004)<br>Tseng et al. (2005) | Andrew (2006)<br>Zhang and Clark (2007) |
| neural | Zheng et al. (2013)<br>Chen et al. (2015b) | **this work** |

Figure 1: Word segmentation methods.

Recently, 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, convolution neural network[1] (Zheng et al., 2013), tensor neural network (Pei et al., 2014), recursive neural network (Chen et al., 2015a) and long-short-term-memory (LSTM) (Chen et al., 2015b) have been used to derive neural feature representations from input word sequences, which are fed into a CRF inference layer.

In this paper, we investigate the effectiveness of word embedding features for neural network segmentation using transition-based models. Since it is challenging to integrate word features to the CRF inference framework of the existing

[1] The term in this paper is used to denote the neural network structure with convolutional layers, which is different from the typical convolution neural network that has a pooling layer upon convolutional layers (Krizhevsky et al., 2012).

*Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 421–431,
Berlin, Germany, August 7-12, 2016. ©2016 Association for Computational Linguistics

| step | action | buffer($\cdots w_{-1} w_0$) | queue($c_0 c_1 \cdots$) |
|---|---|---|---|
| 0 | - | $\phi$ | 中 国 $\cdots$ |
| 1 | *SEP* | 中 | 国 外 $\cdots$ |
| 2 | *APP* | 中国 | 外 企 $\cdots$ |
| 3 | *SEP* | 中国 外 | 企 业 $\cdots$ |
| 4 | *APP* | 中国 外企 | 业 务 $\cdots$ |
| 5 | *SEP* | 中国 外企 业 | 务 发 $\cdots$ |
| 6 | *APP* | 中国 外企 业务 | 发 展 $\cdots$ |
| 7 | *SEP* | $\cdots$ 业务 发 | 展 迅 速 |
| 8 | *APP* | $\cdots$ 业务 发展 | 迅 速 |
| 9 | *SEP* | $\cdots$ 发展 迅 | 速 |
| 10 | *APP* | $\cdots$ 发展 迅速 | $\phi$ |

Figure 2: Segmentation process of "中国 (Chinese) 外企 (foreign company) 业务 (business) 发展 (develop) 迅速 (quickly)".

character-based methods, we take inspiration from word-based discrete segmentation instead. In particular, we follow Zhang and Clark (2007), using the transition-based framework to decode a sentence from left-to-right incrementally, scoring partially segmented results using both character-level and word-level features. Beam-search is applied to reduce error propagation and large-margin training with early-update (Collins and Roark, 2004) is used for learning from inexact search.

We replace the discrete word and character features of Zhang and Clark (2007) with word and character embeddings, respectively, and change their linear model into a deep neural network. Following Zheng et al. (2013) and Chen et al. (2015b), we use convolution neural networks to achieve local feature combination and LSTM to learn global sentence-level features, respectively. The resulting model is a word-based neural segmenter that can leverage rich embedding features. Its correlation with existing work on Chinese segmentation is shown in Figure 1.

Results on standard benchmark datasets show the effectiveness of word embedding features for neural segmentation. Our method achieves state-of-the-art results without any preprocess based on external knowledge such as Chinese idioms of Chen et al. (2015a) and Chen et al. (2015b). We release our code under GPL for research reference.[2]

## 2 Baseline Discrete Model

We exploit the word-based segmentor of Zhang and Clark (2011) as the baseline system. It incrementally segments a sentence using a transition system, with a state holding a partially-segmented sentence in a buffer $s$ and ordering the next incoming characters in a queue $q$. Given an input Chinese sentence, the buffer is initially empty and the queue contains 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 buffer. The actions include:

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

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

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 correct output can be derived using action sequence "*SEP APP SEP APP SEP APP SEP APP SEP APP*", as shown in Figure 2.

**Search.** Based on the transition system, the decoder searches for an optimal action sequence for a given sentence. Denote an action sequence as $A = a_1 \cdots a_n$. We define the score of $A$ as the total score of all actions in the sequence, which is computed by:

$$score(A) = \sum_{a \in A} score(a) = \sum_{a \in A} w \cdot f(s, q, a),$$

where $w$ is the model parameters, $f$ is a feature extraction function, $s$ and $q$ are the buffer and queue of a certain state before the action $a$ is applied.

The feature templates are shown in Table 1, which are the same as Zhang and Clark (2011). These base features include three main source of information. First, characters in the front of the queue and the end of the buffer are used for scoring both *separate* and *append* actions (e.g. $c_0$). Second, words that are identified are used to guide *separate* actions (e.g. $w_0$). Third, relevant information of identified words, such as their lengths and first/last characters are utilized for additional features (e.g. $len(w_{-1})$).

We follow Zhang and Clark (2011) in using beam-search for decoding, shown in Algorith 1, where $\Theta$ is the set of model parameters. Initially the beam contains only the initial state. At each step, each state in the beam is extended by applying both *SEP* and *APP*, resulting in a set of new states, which are scored and ranked. The top *B* are

[2] https://github.com/SUTDNLP/NNTransitionSegmentor.

| Feature templates | Action |
|---|---|
| $c_{-1}c_0$ | *APP, SEP* |
| $w_{-1}$, $w_{-1}w_{-2}$, $w_{-1}c_0$, $w_{-2}len(w_{-1})$<br>$start(w_{-1})c_0$, $end(w_{-1})c_0$<br>$start(w_{-1})end(w_{-1})$, $end(w_{-2})end(w_{-1})$<br>$w_{-2}len(w_{-1})$, $len(w_{-2})w_{-1}$<br>$w_{-1}$, where $len(w_{-1}) = 1$ | *SEP* |

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

**Algorithm 1** Beam-search decoding.

```
function DECODE($c_1 \cdots c_n$, $\Theta$)
  $agenda \leftarrow$ { ($\phi$, $c_1 \cdots c_n$, score=0.0) }
  for $i$ in $1 \cdots n$
    $beam \leftarrow$ { }
    for $cand$ in $agenda$
      $new \leftarrow$ SEP($cand$, $c_i$, $\Theta$)
      ADDITEM(beam, new)
      new $\leftarrow$ APP(cand, $c_i$, $\Theta$)
      ADDITEM($beam$, $new$)
    $agenda \leftarrow$ TOP-B($beam$, $B$)
  $best \leftarrow$ BESTITEM($agenda$)
  $w_1 \cdots w_m \leftarrow$ EXTRACTWORDS($best$)
```

used as the beam states 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. Online leaning with max-margin is used, which is given in section 4.

## 3 Transition-Based Neural Model

We use a neural network model to replace the discrete linear model for scoring transition action sequences. For better comparison between discrete and neural features, the overall segmentation framework of the baseline is used, which includes the incremental segmentation process, the beam-search decoder and the training process integrated with beam-search (Zhang and Clark, 2011). In addition, the neural network scorer takes the similar feature sources as the baseline, which includes character information over the input, word information of the partially constructed output, and the history sequence of the actions that have been applied so far.

The overall architecture of the neural scorer is shown in Figure 3. Given a certain state configuration $(s, q)$, we use three separate recurrent neural networks (RNN) to model the word sequence $\cdots w_{-1}w_0$, the character sequence $\cdots c_{-1}c_0c_1\cdots$, and the action sequence $\cdots a_{-1}a_0$, respectively, resulting in three dense real-valued vectors $\{\mathbf{r}_w, \mathbf{r}_c$ and $\mathbf{r}_a\}$, respectively. All the three feature vectors are used scoring the *SEP* action. For *APP*, on the other hand, we use only the character and action features $\mathbf{r}_c$ and $\mathbf{r}_a$ because the last word $w_0$ in the buffer is a partial word. Formally, given $\mathbf{r}_w, \mathbf{r}_c, \mathbf{r}_a$, the action scores are computed by:

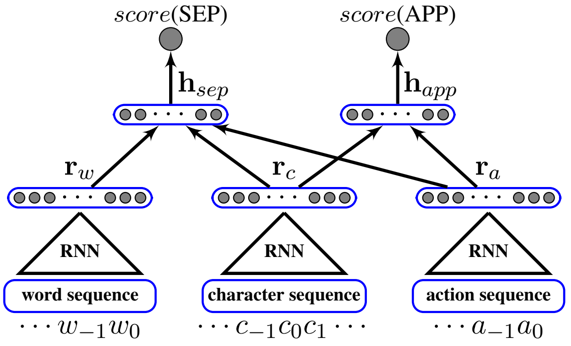


Figure 3: Scorer for the neural transition-based Chinese word segmentation model. We denote the last word in the buffer as $w_0$, the next incoming character as $c_0$ in the queue in consistent with Figure 2, and the last applied action as $a_0$.

$$score(\text{SEP}) = \mathbf{w}_{sep}\mathbf{h}_{sep}$$
$$score(\text{APP}) = \mathbf{w}_{app}\mathbf{h}_{app}$$

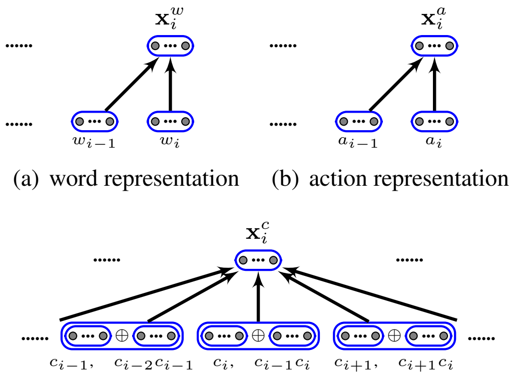
where

$$\mathbf{h}_{sep} = \tanh(W_{sep}[\mathbf{r}_w, \mathbf{r}_c, \mathbf{r}_a] + \mathbf{b}_{sep})$$
$$\mathbf{h}_{app} = \tanh(W_{app}[\mathbf{r}_c, \mathbf{r}_a] + \mathbf{b}_{app})$$

$W_{sep}, W_{app}, \mathbf{b}_{sep}, \mathbf{b}_{app}, \mathbf{w}_{sep}, \mathbf{w}_{app}$ are model parameters.

The neural networks take the embedding forms of words, characters and actions as input, for extracting $\mathbf{r}_w, \mathbf{r}_c$ and $\mathbf{r}_a$, respectively. We exploit the LSTM-RNN structure (Hochreiter and Schmidhuber, 1997), which can better capture non-local syntactic and semantic information from a sequential input, yet reducing gradient explosion or diminishing during training.

In general, given a sequence of input vectors $\mathbf{x}_0 \cdots \mathbf{x}_n$, the LSTM-RNN computes a sequence of hidden vectors $\mathbf{h}_0 \cdots \mathbf{h}_n$, respectively, with each $\mathbf{h}_i$ being determined by the input $\mathbf{x}_i$ and the previous hidden vector $\mathbf{h}_{i-1}$. A cell structure $\mathbf{ce}$ is



(a) word representation (b) action representation

(c) character representation

Figure 4: Input representations of LSTMS for $\mathbf{r}_a$ (actions) $\mathbf{r}_w$ (words) and $\mathbf{r}_c$ (characters).

used to carry long-term memory information over the history $\mathbf{h}_0 \cdots \mathbf{h}_i$ for calculating $\mathbf{h}_i$, and information flow is controlled by an input gate **ig**, an output gate **og** and a forget gate **fg**. Formally, the calculation of $\mathbf{h}_i$ using $\mathbf{h}_{i-1}$ and $\mathbf{x}_i$ is:

$$\begin{aligned}
\mathbf{ig}_i &= \sigma(W_{ig}\mathbf{x}_i + U_{ig}\mathbf{h}_{i-1} + V_{ig}\mathbf{ce}_{i-1} + \mathbf{b}_{ig}) \\
\mathbf{fg}_i &= \sigma(W_{fg}\mathbf{x}_i + U_{fg}\mathbf{h}_{i-1} + V_{fg}\mathbf{ce}_{i-1} + \mathbf{b}_{fg}) \\
\mathbf{ce}_i &= \mathbf{fg}_i \odot \mathbf{ce}_{i-1} + \\
&\qquad \mathbf{ig}_i \odot \tanh(W_{ce}\mathbf{x}_i + U_{ce}\mathbf{h}_{i-1} + \mathbf{b}_{ce}) \\
\mathbf{og}_i &= \sigma(W_{og}\mathbf{x}_i + U_{og}\mathbf{h}_{i-1} + V_{og}\mathbf{ce}_i + \mathbf{b}_{og}) \\
\mathbf{h}_i &= \mathbf{og}_i \odot \tanh(\mathbf{ce}_i),
\end{aligned}$$

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

When used to calculate $\mathbf{r}_w$, $\mathbf{r}_c$ and $\mathbf{r}_a$, the general LSTM structure above is given different input sequences $\mathbf{x}_0 \cdots \mathbf{x}_n$, according to the word, character and action sequences, respectively.

### 3.1 Input representation

**Words.** Given a word $w$, we use a looking-up matrix $E_w$ to obtain its embedding $\mathbf{e}_w(w)$. The matrix can be obtained through pre-training on large size of auto segmented corpus. As shown in Figure 4(a), we use a convolutional neural layer upon a two-word window to obtain $\cdots \mathbf{x}_{-1}^w \mathbf{x}_0^w$ for the LSTM for $\mathbf{r}_w$, with the following formula:

$$\mathbf{x}_i^w = \tanh\left(W_w[\mathbf{e}_w(w_{i-1}), \mathbf{e}_w(w_i)] + \mathbf{b}_w\right)$$

**Actions.** We represent an action $a$ with an embedding $\mathbf{e}_a(a)$ from a looking-up table $E_a$, and apply the similar convolutional neural network to obtain $\cdots \mathbf{x}_{-1}^a \mathbf{x}_0^a$ for $\mathbf{r}_a$, as shown in Figure 4(b). Given the input action sequence $\cdots a_{-1}a_0$, the $\mathbf{x}_i^a$ is computed by:

$$\mathbf{x}_i^a = \tanh\left(W_a[\mathbf{e}_a(a_{i-1}), \mathbf{e}_a(a_i)] + \mathbf{b}_a\right)$$

**Characters.** We make embeddings for both character unigrams and bigrams by looking-up matrixes $E_c$ and $E_{bc}$, respectively, the latter being shown to be useful by Pei et al. (2014). For each character $c_i$, the unigram embedding $\mathbf{e}_c(c_i)$ and the bigram embedding $\mathbf{e}_{bc}(c_{i-1}c_i)$ are concatenated, before being given to a CNN with a convolution size of 5. For the character sequence $\cdots c_{-1}c_0c_1 \cdots$ of a given state $(s, q)$, we compute its input vectors $\cdots \mathbf{x}_{-1}^c \mathbf{x}_0^c \mathbf{x}_1^c \cdots$ for the LSTM for $\mathbf{r}_c$ by:

$$\begin{aligned}
\mathbf{x}_i^c = \tanh\big(&W_c[\mathbf{e}_c(c_{i-2}) \oplus \mathbf{e}_{bc}(c_{i-3}c_{i-2}), \\
&\cdots, \mathbf{e}_c(c_i) \oplus \mathbf{e}_{bc}(c_{i-1}c_i), \cdots, \\
&\mathbf{e}_c(c_{i+2}) \oplus \mathbf{e}_{bc}(c_{i+1}c_{i+2})] + \mathbf{b}_c\big)
\end{aligned}$$

For all the above input representations, the looking-up tables $E_w$, $E_a$ , $E_c$, $E_{bc}$ and the weights $W_w$, $W_a$, $W_c$, $\mathbf{b}_w$, $\mathbf{b}_a$, $\mathbf{b}_c$ are model parameters. For calculating $\mathbf{r}_w$ and $\mathbf{r}_a$, we apply the LSTMs directly over the sequences $\cdots x_{-1}^w x_0^w$ and $\cdots x_{-1}^a x_0^a$ for words and actions, and use the outputs $\mathbf{h}_0^w$ and $\mathbf{h}_0^a$ for $\mathbf{r}_w$ and $\mathbf{r}_a$, respectively. For calculating $\mathbf{r}_c$, we further use a bi-directional extension of the original LSTM structure. In particular, the base LSTM is applied to the input character sequence both from left to right and from right to left, leading to two hidden node sequences $\cdots \mathbf{h}_{-1}^{cL}\mathbf{h}_0^{cL}\mathbf{h}_1^{cL} \cdots$ and $\cdots \mathbf{h}_{-1}^{cR}\mathbf{h}_0^{cR}\mathbf{h}_1^{cR} \cdots$, respectively. For the current character $c_0$, $\mathbf{h}_0^{cL}$ and $\mathbf{h}_0^{cR}$ are concatenated to form the final vector $\mathbf{r}_c$. This is feasible because the character sequence is input and static, and previous work has demonstrated better capability of bi-directional LSTM for modeling sequences (Yao and Zweig, 2015).

### 3.2 Integrating discrete features

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

$$\begin{aligned}
score(\text{SEP}) &= \mathbf{w'}_{sep}(\mathbf{h}_{sep} \oplus f_{sep}) \\
score(\text{APP}) &= \mathbf{w'}_{app}(\mathbf{h}_{app} \oplus f_{app}),
\end{aligned}$$

where $\mathbf{f}_{sep}$ and $\mathbf{f}_{app}$ represent the baseline sparse vector for *SEP* and *APP* features, respectively, and $\oplus$ denotes the vector concatenation operation.

**Algorithm 2** Max-margin training with early-update.

```
function TRAIN($c_1 \cdots c_n$, $a_1^g \cdots a_n^g$, $\Theta$)
    agenda ← { ($\phi$, $c_1 \cdots c_n$, score=0.0) }
    for $i$ in $1 \cdots n$
        beam ← { }
        for cand in agenda
            new ← SEP(cand, $c_i$, $\Theta$)
            if $\{a_i^g \neq$ SEP$\}$ new.score += $\eta$
            ADDITEM(beam, new)
            new ← APP(cand, $c_i$, $\Theta$)
            if $\{a_i^g \neq$ APP$\}$ new.score += $\eta$
            ADDITEM(beam, new)
        agenda ← TOP-B(beam, B)
        if {ITEM($a_1^g \cdots a_i^g$) $\notin$ agenda}
            $\Theta = \Theta - f\big($BESTITEM(agenda)$\big)$
            $\Theta = \Theta + f\big($ITEM(($a_1^g \cdots a_i^g$)$\big)$
            return
    if {ITEM($a_1^g \cdots a_n^g$) $\neq$ BESTITEM(agenda)}
        $\Theta = \Theta - f\big($BESTITEM(agenda)$\big)$
        $\Theta = \Theta + f\big($ITEM(($a_1^g \cdots a_n^g$)$\big)$
```

## 4 Training

To train model parameters for both the discrete and neural models, we exploit online learning with early-update as shown in Algorithm 2. A max-margin objective is exploited,[3] which is defined as:

$$L(\Theta) = \frac{1}{K}\sum_{k=1}^{K} l(A_k^g, \Theta) + \frac{\lambda}{2} \parallel \Theta \parallel^2$$

$$l(A_k^g, \Theta) = \max_{A} \big(score(A_k, \Theta) + \eta \cdot \delta(A_k, A_k^g)\big) - score(A_k^g, \Theta),$$

where $\Theta$ is the set of all parameters, $\{A_k^g\}_{n=1}^K$ are gold action sequences to segment the training corpus, $A_k$ is the model output action sequence, $\lambda$ is a regularization parameter and $\eta$ is used to tune the loss margins.

For the discrete models, $f(\cdot)$ denotes the features extracted according to the feature templates in Table 1. For the neural models, $f(\cdot)$ denotes the corresponding $\mathbf{h}_{sep}$ and $\mathbf{h}_{app}$. Thus only the output layer is updated, and we further use back-propagation to learn the parameters of the other layers (LeCun et al., 2012). We use online AdaGrad (Duchi et al., 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 $(-0.01, 0.01)$.

| | | CTB60 | PKU | MSR |
|---|---|---|---|---|
| Training | #sent | 23k | 17k | 78k |
| | #word | 641k | 1,010k | 2,122k |
| Development | #sent | 2.1k | 1.9k | 8.7k |
| | #word | 60k | 100k | 246k |
| Test | #sent | 2.8k | 1.9k | 4.0k |
| | #word | 82k | 104k | 106k |

Table 2: Statistics of datasets.

| Type | hyper-parameters |
|---|---|
| Network structure | $d(\mathbf{h}_{sep}) = 100, d(\mathbf{h}_{app}) = 80$<br>$d(\mathbf{h}_i^a) = 20, d(\mathbf{x}_i^a) = 20$<br>$d(\mathbf{h}_i^w) = 50, d(\mathbf{x}_i^w) = 50$<br>$d(\mathbf{h}_i^{cL}) = d(\mathbf{h}_i^{cR}) = 50, d(\mathbf{x}_i^c) = 50$<br>$d(\mathbf{e}_w(w_i)) = 50, d(\mathbf{e}_a(a_i)) = 20$<br>$d(\mathbf{e}_c(c_i)) = 50, d(\mathbf{e}_{bc}(c_{i-1}c_i)) = 50$ |
| Training | $\lambda = 10^{-8}, \alpha = 0.01, \eta = 0.2$ |

Table 3: Hyper-parameter values in our model.

## 5 Experiments

### 5.1 Experimental Settings

**Data.** We use three datasets for evaluation, namely CTB6, PKU and MSR. The CTB6 corpus is taken from Chinese Treebank 6.0, and the PKU and MSR corpora can be obtained from BakeOff 2005 (Emerson, 2005). We follow Zhang et al. (2014), 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. Table 1 shows the overall statistics of the three datasets.

**Embeddings.** We use *word2vec*[4] to pre-train word, character and bi-character embeddings on Chinese Gigaword corpus (LDC2011T13). In order to train full word embeddings, the corpus is segmented automatically by our baseline model.

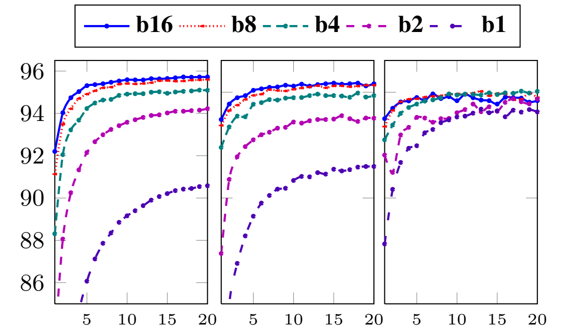
**Hyper-parameters.** The hyper-parameter values are tuned according to development performances. We list their final values in Table 3.

### 5.2 Development Results

To better understand the word-based neural models, we perform several development experiments. *All* the experiments in this section are conducted on the CTB6 development dataset.

[3] Zhou et al. (2015) find that max-margin training did not yield reasonable results for neural transition-based parsing, which is different from our findings. One likely reason is that when the number of labels is small max-margin is effective.

[4] http://word2vec.googlecode.com/



(a) discrete (b) neural(-tune) (c) neural(+tune)

Figure 5: Accuracies against the training epoch using beam sizes 1, 2, 4, 8 and 16, respectively.

### 5.2.1 Embeddings and beam size

We study the influence of beam size on the baseline and neural models. Our neural model has two choices of using pre-trained word embeddings. We can either fine-tune or fix the embeddings during training. In case of fine-tuning, only words in the training data can be learned, while embeddings of out-of-vocabulary (OOV) words could not be used effectively.[5] In addition, following Dyer et al. (2015) we randomly set words with frequency 1 in the training data as the OOV words in order to learn the OOV embedding, while avoiding overfitting. If the pretrained word embeddings are not fine-tuned, we can utilize all word embeddings.

Figure 5 shows the development results, where the training curve of the discrete baseline is shown in Figure 5(a) and the curve of the neural model without and with fine tuning are shown in 5(b) and 5(c), respectively. The performance increases with a larger beam size in all settings. When the beam increases into 16, the gains levels out. The results of the discrete model and the neural model without fine-tuning are highly similar, showing the usefulness of beam-search.

On the other hand, with fine-tuning, the results are different. The model with beam size 1 gives better accuracies compared to the other models with the same beam size. However, as the beam size increases, the performance increases very little. The results are consistent with Dyer et al. (2015), who find that beam-search improves the results only slightly on dependency parsing. When a beam size of 16 is used, this model performs the worst compared with the discrete model and the neural model without fine-tuning. This is likely because the fine-tuning of embeddings leads to overfitting of in-vocabulary words, and underfitting over OOV words. Based on the observation, we exploit fixed word embeddings in our final models.

| Model | P | R | F |
|---|---|---|---|
| neural | 95.21 | 95.69 | 95.45 |
| -word | 91.81 | 92.00 | 91.90 |
| -character unigram | 94.89 | 95.56 | 95.22 |
| -character bigram | 94.93 | 95.53 | 95.23 |
| -action | 95.00 | 95.31 | 95.17 |
| +discrete features (combined) | 96.38 | 96.22 | 96.30 |

Table 4: Feature experiments.

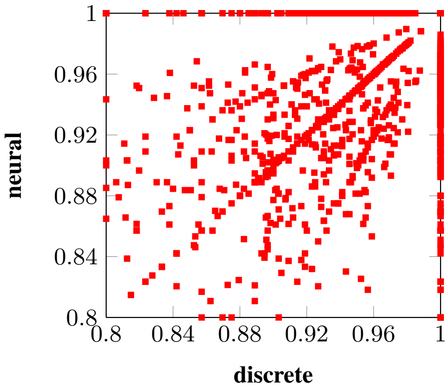


Figure 6: Sentence accuracy comparisons for the discrete and the neural models.

### 5.2.2 Feature ablation

We conduct feature ablation experiments to study the effects of the word, character unigram, character bigram and action features to the neural model. The results are shown in Table 4. Word features are particularly important to the model, without which the performance decreases by 4.5%. The effects of the character unigram, bigram and action features are relatively much weaker.[6] This demonstrates that in the word-based incremental search framework, words are the most crucial information to the neural model.

### 5.2.3 Integrating discrete features

Prior work has shown the effectiveness of integrating discrete and neural features for several NLP tasks (Turian et al., 2010; Wang and Manning,

[5] We perform experiments using random initialized word embeddings as well when fine-tune is used, which is a fully supervised model. The performance is slightly lower.

[6] In all our experiments, we fix the character unigram and bigram embeddings, because fine-tuning of these embeddings results in little changes.

| Models | P | R | F |
|---|---|---|---|
| word-based models | | | |
| discrete | 95.29 | 95.26 | 95.28 |
| neural | 95.34 | 94.69 | 95.01 |
| combined | **96.11** | **95.79** | **95.95** |
| character-based models | | | |
| discrete | 95.38 | 95.12 | 95.25 |
| neural | 94.59 | 94.92 | 94.76 |
| combined | 95.63 | 95.60 | 95.61 |
| other models | | | |
| Zhang et al. (2014) | N/A | N/A | 95.71 |
| Wang et al. (2011) | 95.83 | 95.75 | 95.79 |
| Zhang and Clark (2011) | 95.46 | 94.78 | 95.13 |

Table 5: Main results on CTB60 test dataset.

2013; Durrett and Klein, 2015; Zhang and Zhang, 2015). We investigate the usefulness of such integration to our word-based segmentor on the development dataset. We study it by two ways. 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 of each point denoting the F-measure scores of the two models with respect to sentences, respectively. As shown in Figure 6, the points are rather dispersive, showing the differences of the two models.

Further, we directly look at the results after integration of both discrete and neural features. As shown in Table 4, the integrated model improves the accuracies from 95.45% to 96.30%, demonstrating that the automatically-induced neural features contain highly complementary information to the manual discrete features.

### 5.3 Final Results

Table 6 shows the final results on CTB6 test dataset. For thorough comparison, we implement discrete, neural and combined character-based models as well.[7] 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[8] and a CRF layer for output (Chen et al., 2015b).[9] The combined model uses the same method for integrating discrete and neural features as our word-based model.

| Models | PKU | MSR |
|---|---|---|
| our word-based models | | |
| discrete | 95.1 | 97.3 |
| neural | 95.1 | 97.0 |
| combined | 95.7 | **97.7** |
| character-based models | | |
| discrete | 94.9 | 96.8 |
| neural | 94.4 | 97.2 |
| combined | 95.4 | 97.2 |
| other models | | |
| Cai and Zhao (2016) | 95.5 | 96.5 |
| Ma and Hinrichs (2015) | 95.1 | 96.6 |
| Pei et al. (2014) | 95.2 | 97.2 |
| Zhang et al. (2013a) | **96.1** | 97.5 |
| Sun et al. (2012) | 95.4 | 97.4 |
| Zhang and Clark (2011) | 95.1 | 97.1 |
| Sun (2010) | 95.2 | 96.9 |
| Sun et al. (2009) | 95.2 | 97.3 |

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

The word-based models achieve better performances than character-based models, since our model can exploit additional word information learnt from large auto-segmented corpus. We also compare the results with other models. 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. We achieve slightly better accuracies. Zhang et al. (2014) is a joint segmentation, POS-tagging and dependency parsing model, which can exploit syntactic information.

To compare our models with other state-of-the-art models in the literature, we report the performance on the PKU and MSR datasets also.[10] Our combined model gives the best result on the MSR dataset, and the second best on PKU. The method of Zhang et al. (2013a) gives the best performance on PKU by co-training on large-scale data.

### 5.4 Error Analysis

To study the differences between word-based and character-based neural models, we conduct error analysis on the test dataset of CTB60. First,

[7] The code is released for research reference under GPL at https://github.com/SUTDNLP/NNSegmentation.

[8] We use a concatenation of character unigram and bigram embeddings at each position as the input to LSTM, because our experiments show that the character bigram embeddings are useful, without which character-based neural models are significantly lower than their discrete counterparts.

[9] Bi-directional LSTM is slightly better than a single left-right LSTM used in Chen et al. (2015b).

[10] 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.

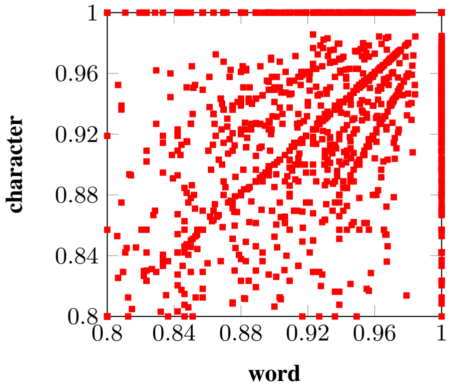


Figure 7: Sentence accuracy comparisons for word- and character-based neural models.

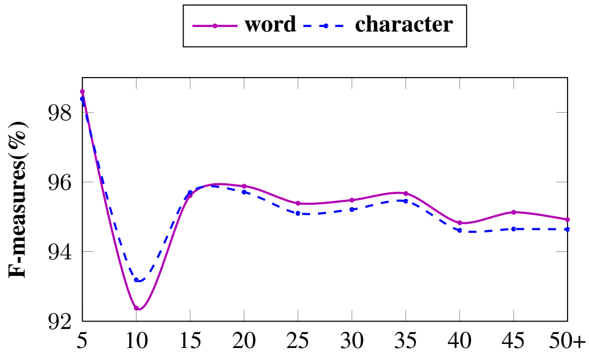


Figure 8: F-measure against character length.

we examine the error distribution on individual sentences. Figure 7 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 line, 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 sentence lengths. We divide the test sentences into ten bins, with bin $i$ denoting sentence lengths in $[5 * (i - 1), 5 * i]$. Figure 8 shows the results. According to the figure, we observe that word-based neural model is relatively weaker for sentences with length in $[5, 10]$, while can better tackle long sentences.

Third, we compare the two neural models by their capabilities of modeling words with different lengths. Figure 9 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.

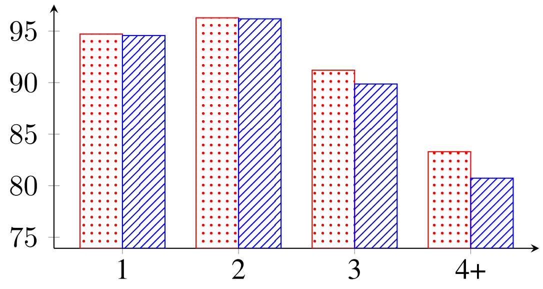


Figure 9: F-measure against word length, where the boxes with red dots denote the performances of word-based neural model, and the boxes with blue slant lines denote character-based neural model.

## 6 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 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 et al., 2011). A tag transition matrix is used for inference, which makes the model effectively. Most subsequent work on neural segmentation followed this method, improving the extraction of emission features by using more complex neural network structures.

Mansur et al. (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 neural-based models using word-level embedding features (Morita et al., 2015; Liu et al., 2016; Cai and Zhao, 2016), which are different from our work in the basic framework. For instance, Liu et al. (2016) follow Andrew (2006) using a semi-CRF for structured inference.

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 et al. (2014), who integrated features of character-based and word-based segmentors.

Transition-based framework with beam search has been widely exploited in a number of other NLP tasks, including syntactic parsing (Zhang and Nivre, 2011; Zhu et al., 2013), information extraction (Li and Ji, 2014) and the work of joint models (Zhang et al., 2013b; Zhang et al., 2014). Recently, the effectiveness of neural features has been studied for this framework. In the natural language parsing community, it has achieved great success. Representative work includes Zhou et al. (2015), Weiss et al. (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.

## 7 Conclusion

We proposed a word-based neural model for Chinese segmentation, which exploits not only character embeddings as previous work does, but also word embeddings pre-trained from large scale corpus. The model achieved comparable performances compared with a discrete word-based baseline, and also the state-of-the-art character-based neural models in the literature. We further demonstrated that the model can utilize discrete features conveniently, resulting in a combined model that achieved top performances compared with previous work. Finally, we conducted several comparisons to study the differences between our word-based model with character-based neural models, showing that they have different error characteristics.

## Acknowledgments

We thank the anonymous reviewers, Yijia Liu and Hai Zhao for their constructive comments, which help to improve the final paper. This work is supported by National Natural Science Foundation of China (NSFC) under grant 61170148, Natural Science Foundation of Heilongjiang Province (China) under grant No.F2016036, the Singapore Ministry of Education (MOE) AcRF Tier 2 grant T2MOE201301 and SRG ISTD 2012 038 from Singapore University of Technology and Design. Yue Zhang is the corresponding author.

## References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the ACL 2016*.

Galen Andrew. 2006. A hybrid markov/semi-markov conditional random field for sequence segmentation.

In *Proceedings of the 2006 Conference on EMNLP*, pages 465–472, Sydney, Australia, July.

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

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, and Xuanjing Huang. 2015a. Gated recursive neural network for chinese word segmentation. In *Proceedings of the 53nd ACL*, pages 1744–1753, July.

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

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July.

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

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

Greg Durrett and Dan Klein. 2015. Neural crf parsing. In *Proceedings of the 53nd ACL*, pages 302–312, July.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53nd ACL*, pages 334–343, July.

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

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

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2012. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer.

Qi Li and Heng Ji. 2014. Incremental joint extraction of entity mentions and relations. In *Proceedings of the ACL 2014*.

Yijia Liu, Wanxiang Che, Jiang Guo, Bing Qin, and Ting Liu. 2016. Exploring segment representations for neural segmentation models. In *Proceedings of IJCAI 2016*.

Jianqiang Ma and Erhard Hinrichs. 2015. Accurate linear-time chinese word segmentation via embedding matching. In *Proceedings of the 53nd ACL*, pages 1733–1743, July.

Mairgup Mansur, Wenzhe Pei, and Baobao Chang. 2013. Feature-based neural language model and chinese word segmentation. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1271–1277, Nagoya, Japan, October. Asian Federation of Natural Language Processing.

Hajime Morita, Daisuke Kawahara, and Sadao Kurohashi. 2015. Morphological analysis for unsegmented languages using recurrent neural network language model. In *Proceedings of the 2015 Conference on EMNLP*, pages 2292–2297.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2014. Max-margin tensor neural network for chinese word segmentation. In *Proceedings of the 52nd ACL*, pages 293–303, Baltimore, Maryland, June.

Fuchun Peng, Fangfang Feng, and Andrew McCallum. 2004. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of Coling 2004*, pages 562–568, Geneva, Switzerland, Aug 23–Aug 27.

Yanxin Shi and Mengqiu Wang. 2007. A dual-layer crfs based joint decoding method for cascaded segmentation and labeling tasks. In *IJCAI*, pages 1707–1712.

Weiwei Sun and Jia Xu. 2011. Enhancing chinese word segmentation using unlabeled data. In *Proceedings of the 2011 Conference on EMNLP*, pages 970–979, July.

Xu Sun, Yaozhong Zhang, Takuya Matsuzaki, Yoshimasa Tsuruoka, and Jun'ichi Tsujii. 2009. A discriminative latent variable chinese segmenter with hybrid word/character information. In *Proceedings of NAACL 2009*, pages 56–64, June.

Xu Sun, Houfeng Wang, and Wenjie Li. 2012. Fast online training with frequency-adaptive learning rates for chinese word segmentation and new word detection. In *Proceedings of the 50th ACL*, pages 253–262, July.

Weiwei Sun. 2010. Word-based and character-based word segmentation models: Comparison and combination. In *Coling 2010: Posters*, pages 1211–1219, August.

Huihsin Tseng, Pichuan Chang, Galen Andrew, Daniel Jurafsky, and Christopher Manning. 2005. A conditional random field word segmenter for sighan bakeoff 2005. In *Proceedings of the fourth SIGHAN workshop*, pages 168–171.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 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*, pages 384–394, July.

Mengqiu Wang and Christopher D. Manning. 2013. Effect of non-linear deep architecture in sequence labeling. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1285–1291, Nagoya, Japan, October. Asian Federation of Natural Language Processing.

Yiou Wang, Jun'ichi Kazama, Yoshimasa Tsuruoka, Wenliang Chen, Yujie Zhang, and Kentaro Torisawa. 2011. Improving chinese word segmentation and pos tagging with semi-supervised methods using large auto-analyzed data. In *Proceedings of 5th IJCNLP*, pages 309–317, Chiang Mai, Thailand, November.

Mengqiu Wang, Rob Voigt, and Christopher D. Manning. 2014. Two knives cut better than one: Chinese word segmentation with dual decomposition. In *Proceedings of the 52nd ACL*, pages 193–198, Baltimore, Maryland, June.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *Proceedings of the 53rd ACL*, pages 1169–1179, July.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd ACL*, pages 323–333, July.

Jingjing Xu and Xu Sun. 2016. Dependency-based gated recursive neural network for chinese word segmentation. In *Proceedings of ACL 2016*.

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

Kaisheng Yao and Geoffrey Zweig. 2015. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. *arXiv preprint arXiv:1506.00196*.

Yue Zhang and Stephen Clark. 2007. Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of the 45th ACL*, pages 840–847, Prague, Czech Republic, June.

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

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th ACL*, pages 188–193, June.

Meishan Zhang and Yue Zhang. 2015. Combining discrete and continuous features for deterministic transition-based dependency parsing. In *Proceedings of the 2015 EMNLP*, pages 1316–1321, September.

Longkai Zhang, Houfeng Wang, Xu Sun, and Mairgup Mansur. 2013a. Exploring representations from unlabeled data with co-training for Chinese word segmentation. In *Proceedings of the EMNLP 2013*, pages 311–321, Seattle, Washington, USA, October.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2013b. Chinese parsing exploiting characters. In *Proceedings of the 51st ACL*, pages 125–134, August.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2014. Character-level chinese dependency parsing. In *Proceedings of the 52nd ACL*, pages 1326–1336, Baltimore, Maryland, June.

Hai Zhao, Chang-Ning Huang, Mu Li, and Bao-Liang Lu. 2006. Effective tag set selection in chinese word segmentation via conditional random field modeling. In *Proceedings of PACLIC*, volume 20, pages 87–94. Citeseer.

Hai Zhao. 2009. Character-level dependencies in chinese: Usefulness and learning. In *Proceedings of the EACL*, pages 879–887, Athens, Greece, March.

Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. 2013. Deep learning for Chinese word segmentation and POS tagging. In *Proceedings of the 2013 Conference on EMNLP*, pages 647–657, Seattle, Washington, USA, October.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd ACL*, pages 1213–1222, July.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st ACL*, pages 434–443, August.