Neural Word Segmentation Learning for Chinese[*]

Deng Cai and Hai Zhao[†]
Department of Computer Science and Engineering
Key Lab of Shanghai Education Commission
for Intelligent Interaction and Cognitive Engineering
Shanghai Jiao Tong University, Shanghai, China
thisisjcykcd@gmail.com, zhaohai@cs.sjtu.edu.cn

Abstract

Most previous approaches to Chinese word segmentation formalize this problem as a character-based sequence labeling task so that only contextual information within fixed sized local windows and simple interactions between adjacent tags can be captured. In this paper, we propose a novel neural framework which thoroughly eliminates context windows and can utilize complete segmentation history. Our model employs a gated combination neural network over characters to produce distributed representations of word candidates, which are then given to a long short-term memory (LSTM) language scoring model. Experiments on the benchmark datasets show that without the help of feature engineering as most existing approaches, our models achieve competitive or better performances with previous state-of-the-art methods.

1 Introduction

Most east Asian languages including Chinese are written without explicit word delimiters, therefore, word segmentation is a preliminary step for processing those languages. Since Xue (2003), most methods formalize the Chinese word segmentation (CWS) as a sequence labeling problem with character position tags, which can be handled with supervised learning methods such as Maximum Entropy (Berger et al., 1996; Low et al., 2005) and Conditional Random Fields (Lafferty et al., 2001; Peng et al., 2004; Zhao et al., 2006a). However, those methods heavily depend on the choice of handcrafted features.

Recently, neural models have been widely used for NLP tasks for their ability to minimize the effort in feature engineering. For the task of CWS, Zheng et al. (2013) adapted the general neural network architecture for sequence labeling proposed in (Collobert et al., 2011), and used character embeddings as input to a two-layer network. Pei et al. (2014) improved upon (Zheng et al., 2013) by explicitly modeling the interactions between local context and previous tag. Chen et al. (2015a) proposed a gated recursive neural network to model the feature combinations of context characters. Chen et al. (2015b) used an LSTM architecture to capture potential long-distance dependencies, which alleviates the limitation of the size of context window but introduced another window for hidden states.

Despite the differences, all these models are designed to solve CWS by assigning labels to the characters in the sequence one by one. At each time step of inference, these models compute the tag scores of character based on (i) context features within a fixed sized local window and (ii) tagging history of previous *one*.

Nevertheless, the tag-tag transition is insufficient to model the complicated influence from previous segmentation decisions, though it could sometimes be a crucial clue to later segmentation decisions. The fixed context window size, which is broadly adopted by these methods for feature engineering, also restricts the flexibility of modeling diverse distances. Moreover, word-level information, which is being the greater granularity unit

[*] This paper was partially supported by Cai Yuanpei Program (CSC No. 201304490199 and No. 201304490171), National Natural Science Foundation of China (No. 61170114 and No. 61272248), National Basic Research Program of China (No. 2013CB329401), Major Basic Research Program of Shanghai Science and Technology Committee (No. 15JC1400103), Art and Science Interdisciplinary Funds of Shanghai Jiao Tong University (No. 14JCRZ04), and Key Project of National Society Science Foundation of China (No. 15-ZDA041).

[†] Corresponding author.

Models		Characters	Words	Tags
character based	(Zheng et al., 2013), ...	$c_{i-2}, c_{i-1}, c_i, c_{i+1}, c_{i+2}$	-	$t_{i-1}t_i$
	(Chen et al., 2015b)	$c_0, c_1, \ldots, c_i, c_{i+1}, c_{i+2}$	-	$t_{i-1}t_i$
word based	(Zhang and Clark, 2007), ...	c in w_{j-1}, w_j, w_{j+1}	w_{j-1}, w_j, w_{j+1}	-
	Ours	$c_0, c_1, \ldots, c_i$	$w_0, w_1, \ldots, w_j$	-

Table 1: Feature windows of different models. $i(j)$ indexes the current character(word) that is under scoring.

as suggested in (Huang and Zhao, 2006), remains unemployed.

To alleviate the drawbacks inside previous methods and release those inconvenient constrains such as the fixed sized context window, this paper makes a latest attempt to re-formalize CWS as a direct segmentation learning task. Our method does not make tagging decisions on individual characters, but directly evaluates the relative likelihood of different segmented sentences and then search for a segmentation with the highest score. To feature a segmented sentence, a series of distributed vector representations (Bengio et al., 2003) are generated to characterize the corresponding word candidates. Such a representation setting makes the decoding quite different from previous methods and indeed much more challenging, however, more discriminative features can be captured.

Though the vector building is word centered, our proposed scoring model covers all three processing levels from character, word until sentence. First, the distributed representation starts from character embedding, as in the context of word segmentation, the n-gram data sparsity issue makes it impractical to use word vectors immediately. Second, as the word candidate representation is derived from its characters, the inside character structure will also be encoded, thus it can be used to determine the word likelihood of its own. Third, to evaluate how a segmented sentence makes sense through word interacting, an LSTM (Hochreiter and Schmidhuber, 1997) is used to chain together word candidates incrementally and construct the representation of partially segmented sentence at each decoding step, so that the coherence between next word candidate and previous segmentation history can be depicted.

To our best knowledge, our proposed approach to CWS is the first attempt which explicitly models the entire contents of the segmenter's state, including the complete history of both segmentation decisions and input characters. The comparisons of feature windows used in different models are shown in Table 1. Compared to both sequence labeling schemes and word-based models in the past, our model thoroughly eliminates context windows and can capture the complete history of segmentation decisions, which offers more possibilities to effectively and accurately model segmentation context.

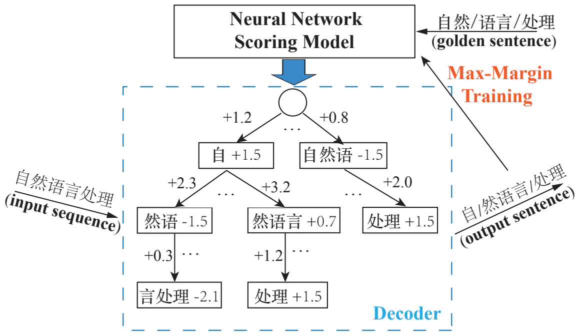

Figure 1: Our framework.

2 Overview

We formulate the CWS problem as finding a mapping from an input character sequence x to a word sequence y, and the output sentence y^* satisfies:

$$y^* = \arg\max_{y \in \text{GEN}(x)} \left(\sum_{i=1}^{n} \text{score}(y_i | y_1, \cdots, y_{i-1}) \right)$$

where n is the number of word candidates in y, and GEN(x) denotes the set of possible segmentations for an input sequence x. Unlike all previous works, our scoring function is sensitive to the complete contents of partially segmented sentence.

As shown in Figure 1, to solve CWS in this way, a neural network scoring model is designed to evaluate the likelihood of a segmented sentence. Based on the proposed model, a decoder is developed to find the segmented sentence with the highest score. Meanwhile, a max-margin method

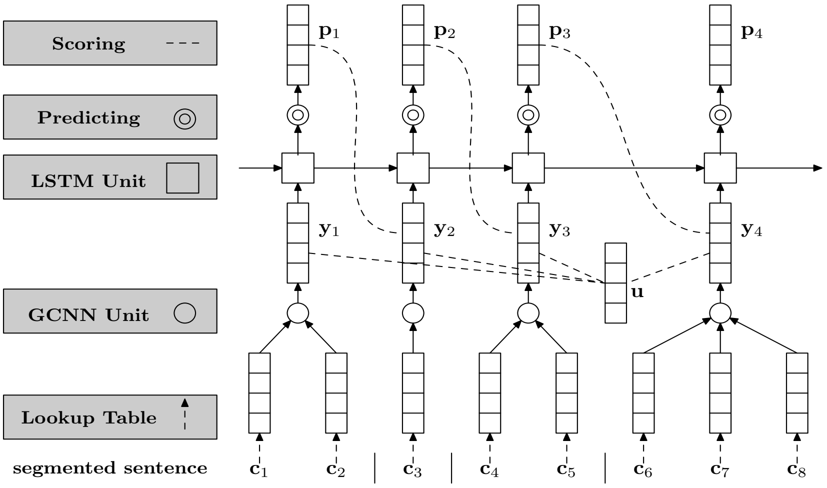

Figure 2: Architecture of our proposed neural network scoring model, where $\mathbf{c}_i$ denotes the i-th input character, $\mathbf{y}_j$ denotes the learned representation of the j-th word candidate and $\mathbf{p}_k$ denotes the prediction for the $(k+1)$-th word candidate.

is utilized to perform the training by comparing the structured difference of decoder output and the golden segmentation. The following sections will introduce each of these components in detail.

3 Neural Network Scoring Model

The score for a segmented sentence is computed by first mapping it into a sequence of word candidate vectors, then the scoring model takes the vector sequence as input, scoring on each word candidate from two perspectives: (1) how likely the word candidate itself can be recognized as a legal word; (2) how reasonable the link is for the word candidate to follow previous segmentation history immediately. After that, the word candidate is appended to the segmentation history, updating the state of the scoring system for subsequent judgements. Figure 2 illustrates the entire scoring neural network.

3.1 Word Score

Character Embedding. While the scores are decided at the word-level, using word embedding (Bengio et al., 2003; Wang et al., 2016) immediately will lead to a remarkable issue that rare words and out-of-vocabulary words will be poorly estimated (Kim et al., 2015). In addition, the character-level information inside an n-gram can be helpful to judge whether it is a true word. Therefore, a lookup table of character embeddings is used as the bottom layer.

Formally, we have a character dictionary D of size $|D|$. Then each character $c \in D$ is represented as a real-valued vector (character embedding) $\mathbf{c} \in \mathbb{R}^d$, where d is the dimensionality of the vector space. The character embeddings are then stacked into an embedding matrix $\mathbf{M} \in \mathbb{R}^{d \times |D|}$. For a character $c \in D$, its character embedding $\mathbf{c} \in \mathbb{R}^d$ is retrieved by the embedding layer according to its index.

Gated Combination Neural Network. In order to obtain word representation through its characters, in the simplest strategy, character vectors are integrated into their word representation using a weight matrix $\mathbf{W}^{(L)}$ that is shared across all words with the same length L, followed by a non-linear function $g(\cdot)$. Specifically, $\mathbf{c}_i$ $(1 \leq i \leq L)$ are d-dimensional character vector representations respectively, the corresponding word vector $\mathbf{w}$ will be d-dimensional as well:

$$\mathbf{w} = g(\mathbf{W}^{(L)} \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_L \end{bmatrix}) \qquad (1)$$

where $\mathbf{W}^{(L)} \in \mathbb{R}^{d \times Ld}$ and g is a non-linear function as mentioned above.

Although the mechanism above seems to work well, it can not sufficiently model the complicated combination features in practice, yet.

Gated structure in neural network can be useful for hybrid feature extraction according to (Chen et al., 2015a; Chung et al., 2014; Cho et al., 2014), we therefore propose a gated combination neural network (GCNN) especially for character compositionality which contains two types of gates,

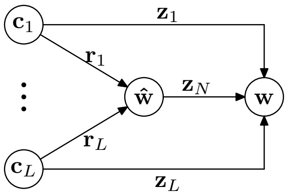

Figure 3: Gated combination neural network.

namely *reset gate* and *update gate*. Intuitively, the reset gates decide which part of the character vectors should be mixed while the update gates decide what to preserve when combining the characters information. Concretely, for words with length L, the word vector $\mathbf{w} \in \mathbb{R}^d$ is computed as follows:

$$\mathbf{w} = \mathbf{z}_N \odot \hat{\mathbf{w}} + \sum_{i=1}^{L} \mathbf{z}_i \odot \mathbf{c}_i$$

where $\mathbf{z}_N, \mathbf{z}_i$ $(1 \leq i \leq L)$ are update gates for new activation $\hat{\mathbf{w}}$ and governed characters respectively, and $\odot$ indicates element-wise multiplication.

The new activation $\hat{\mathbf{w}}$ is computed as:

$$\hat{\mathbf{w}} = \tanh(\mathbf{W}^{(L)} \begin{bmatrix} \mathbf{r}_1 \odot \mathbf{c}_1 \\ \vdots \\ \mathbf{r}_L \odot \mathbf{c}_L \end{bmatrix})$$

where $\mathbf{W}^{(L)} \in \mathbb{R}^{d \times Ld}$ and $\mathbf{r}_i \in \mathbb{R}^d$ $(1 \leq i \leq L)$ are the reset gates for governed characters respectively, which can be formalized as:

$$\begin{bmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_L \end{bmatrix} = \sigma(\mathbf{R}^{(L)} \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_L \end{bmatrix})$$

where $\mathbf{R}^{(L)} \in \mathbb{R}^{Ld \times Ld}$ is the coefficient matrix of reset gates and σ denotes the sigmoid function.

The update gates can be formalized as:

$$\begin{bmatrix} \mathbf{z}_N \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_L \end{bmatrix} = \exp(\mathbf{U}^{(L)} \begin{bmatrix} \hat{\mathbf{w}} \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_L \end{bmatrix}) \odot \begin{bmatrix} 1/\mathbf{Z} \\ 1/\mathbf{Z} \\ \vdots \\ 1/\mathbf{Z} \end{bmatrix}$$

where $\mathbf{U}^{(L)} \in \mathbb{R}^{(L+1)d \times (L+1)d}$ is the coefficient matrix of update gates, and $\mathbf{Z} \in \mathbb{R}^d$ is the normalization vector,

$$\mathbf{Z}_k = \sum_{i=1}^{L} [\exp(\mathbf{U}^{(L)} \begin{bmatrix} \hat{\mathbf{w}} \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_L \end{bmatrix})]_{d \times i + k}$$

where $0 \leq k < d$.

According to the normalization condition, the update gates are constrained by:

$$\mathbf{z}_N + \sum_{i=1}^{L} \mathbf{z}_i = \mathbf{1}$$

The gated mechanism is capable of capturing both character and character interaction characteristics to give an efficient word representation (See Section 6.3).

Word Score. Denote the learned vector representations for a segmented sentence y with $[\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n]$, where n is the number of word candidates in the sentence. *word score* will be computed by the dot products of vector $\mathbf{y}_i (1 \leq i \leq n)$ and a trainable parameter vector $\mathbf{u} \in \mathbb{R}^d$.

$$\text{Word\_Score}(\mathbf{y}_i) = \mathbf{u} \cdot \mathbf{y}_i \qquad (2)$$

It indicates how likely a word candidate by itself is to be a true word.

3.2 Link Score

Inspired by the recurrent neural network language model (RNN-LM) (Mikolov et al., 2010; Sundermeyer et al., 2012), we utilize an LSTM system to capture the coherence in a segmented sentence.

Long Short-Term Memory Networks. The LSTM neural network (Hochreiter and Schmidhuber, 1997) is an extension of the recurrent neural network (RNN), which is an effective tool for sequence modeling tasks using its hidden states for history information preservation. At each time step t, an RNN takes the input $\mathbf{x}_t$ and updates its recurrent hidden state $\mathbf{h}_t$ by

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

where g is a non-linear function.

Although RNN is capable, in principle, to process arbitrary-length sequences, it can be difficult to train an RNN to learn long-range dependencies due to the vanishing gradients. LSTM addresses this problem by introducing a memory cell to preserve states over long periods of time, and controls the update of hidden state and memory cell by three types of gates, namely *input gate*, *forget gate* and *output gate*. Concretely, each step of LSTM takes input $\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}$ and produces

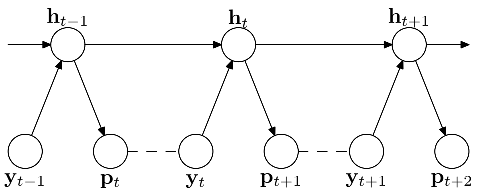

Figure 4: Link scores (dashed lines).

$\mathbf{h}_t, \mathbf{c}_t$ via the following calculations:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}^i\mathbf{x}_t + \mathbf{U}^i\mathbf{h}_{t-1} + \mathbf{b}^i)\\
\mathbf{f}_t &= \sigma(\mathbf{W}^f\mathbf{x}_t + \mathbf{U}^f\mathbf{h}_{t-1} + \mathbf{b}^f)\\
\mathbf{o}_t &= \sigma(\mathbf{W}^o\mathbf{x}_t + \mathbf{U}^o\mathbf{h}_{t-1} + \mathbf{b}^o)\\
\hat{\mathbf{c}}_t &= \tanh(\mathbf{W}^c\mathbf{x}_t + \mathbf{U}^c\mathbf{h}_{t-1} + \mathbf{b}^c)\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \hat{\mathbf{c}}_t\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}$$

where $\sigma, \odot$ are respectively the element-wise sigmoid function and multiplication, $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t, \mathbf{c}_t$ are respectively the input gate, forget gate, output gate and memory cell activation vector at time t, all of which have the same size as hidden state vector $\mathbf{h}_t \in \mathbb{R}^H$.

Link Score. LSTMs have been shown to outperform RNNs on many NLP tasks, notably language modeling (Sundermeyer et al., 2012).

In our model, LSTM is utilized to chain together word candidates in a left-to-right, incremental manner. At time step t, a prediction $\mathbf{p}_{t+1} \in \mathbb{R}^d$ about next word $\mathbf{y}_{t+1}$ is made based on the hidden state $\mathbf{h}_t$:

$$\mathbf{p}_{t+1} = \tanh(\mathbf{W}^p\mathbf{h}_t + \mathbf{b}^p)$$

link score for next word $\mathbf{y}_{t+1}$ is then computed as:

$$\text{Link\_Score}(\mathbf{y}_{t+1}) = \mathbf{p}_{t+1} \cdot \mathbf{y}_{t+1} \qquad (3)$$

Due to the structure of LSTM, the prediction vector $\mathbf{p}_{t+1}$ carries useful information detected from the entire segmentation history, including previous segmentation decisions. In this way, our model gains the ability of sequence-level discrimination rather than local optimization.

3.3 Sentence score

Sentence score for a segmented sentence y with n word candidates is computed by summing up word scores (2) and link scores (3) as follow:

$$s(y_{[1:n]}, \theta) = \sum_{t=1}^{n}(\mathbf{u} \cdot \mathbf{y}_t + \mathbf{p}_t \cdot \mathbf{y}_t) \qquad (4)$$

where θ is the parameter set used in our model.

4 Decoding

The total number of possible segmented sentences grows exponentially with the length of character sequence, which makes it impractical to compute the scores of every possible segmentation. In order to get exact inference, most sequence-labeling systems address this problem with a Viterbi search, which takes the advantage of their hypothesis that the tag interactions only exist within adjacent characters (Markov assumption). However, since our model is intended to capture complete history of segmentation decisions, such dynamic programming algorithms can not be adopted in this situation.

Algorithm 1 Beam Search.

```
Input: Parameters $\theta$, beam size $k$, maximum
       word length $w$ and input character sequence
       $c[1:n]$
Output: Approx. $k$ best segmentations
$\pi[0] \leftarrow \{(score = 0, \mathbf{h} = \mathbf{h}_0, \mathbf{c} = \mathbf{c}_0)\}$
for $i = 1$ to $n$ do
    ▷ Generate Candidate Word Vectors
    $X \leftarrow \emptyset$
    for $j = \max(1, i - w)$ to $i$ do
        $\mathbf{w}$ = GCNN-Procedure($c[j:i]$)
        $X$.add($(index = j - 1, word = \mathbf{w})$)
    end for
    ▷ Join Segmentation
    $Y \leftarrow \{\ y$.append($x$) $\mid y \in \pi[x.index]$ and
        $x \in X\}$
    ▷ Filter $k$-Max
    $\pi[i] \leftarrow k\text{-}\arg\max_{y \in Y} y.score$
end for
return $\pi[n]$
```

To make our model efficient in practical use, we propose a beam-search algorithm with dynamic programming motivations as shown in Algorithm 1. The main idea is that any segmentation of the first i characters can be separated as two parts, the first part consists of characters with indexes from 0 to j that is denoted as y, the rest part is the word composed by $c[j+1:i]$. The influence from previous segmentation y can be represented as a triple ($y.score$, $y.\mathbf{h}$, $y.\mathbf{c}$), where $y.score$, $y.\mathbf{h}$, $y.\mathbf{c}$ indicate the current score, current hidden state vector and current memory cell vector respectively. Beam search ensures that the total time for segmenting a sentence of n characters is $w \times k \times n$, where w, k are maximum word length and beam size respectively.

5 Training

We use the max-margin criterion (Taskar et al., 2005) to train our model. As reported in (Kummerfeld et al., 2015), the margin methods generally outperform both likelihood and perception methods. For a given character sequence $x^{(i)}$, denote the correct segmented sentence for $x^{(i)}$ as $y^{(i)}$. We define a structured margin loss $\Delta(y^{(i)}, \hat{y})$ for predicting a segmented sentence $\hat{y}$:

$$\Delta(y^{(i)}, \hat{y}) = \sum_{t=1}^{m} \mu \mathbf{1}\{y^{(i),t} \neq \hat{y}^t\}$$

where m is the length of sequence $x^{(i)}$ and μ is the discount parameter. The calculation of margin loss could be regarded as to count the number of incorrectly segmented characters and then multiple it with a fixed discount parameter for smoothing. Therefore, the loss is proportional to the number of incorrectly segmented characters.

Given a set of training set Ω, the regularized objective function is the loss function $J(\theta)$ including an ℓ_2 norm term:

$$J(\theta) = \frac{1}{|\Omega|} \sum_{(x^{(i)}, y^{(i)}) \in \Omega} l_i(\theta) + \frac{\lambda}{2} ||\theta||_2^2$$

$$l_i(\theta) = \max_{\hat{y} \in \text{GEN}(x^{(i)})} (s(\hat{y}, \theta) + \Delta(y^{(i)}, \hat{y}) - s(y^{(i)}, \theta))$$

where the function $s(\cdot)$ is the sentence score defined in equation (4).

Due to the hinge loss, the objective function is not differentiable, we use a subgradient method (Ratliff et al., 2007) which computes a gradient-like direction. Following (Socher et al., 2013), we use the diagonal variant of AdaGrad (Duchi et al., 2011) with minibatchs to minimize the objective. The update for the i-th parameter at time step t is as follows:

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}$$

where α is the initial learning rate and $g_{\tau,i} \in \mathbb{R}^{|\theta_i|}$ is the subgradient at time step τ for parameter θ_i.

6 Experiments

6.1 Datasets

To evaluate the proposed segmenter, we use two popular datasets, PKU and MSR, from the second International Chinese Word Segmentation Bakeoff (Emerson, 2005). These datasets are commonly used by previous state-of-the-art models and neural network models.

Character embedding size	$d = 50$
Hidden unit number	$H = 50$
Initial learning rate	$\alpha = 0.2$
Margin loss discount	$\mu = 0.2$
Regularization	$\lambda = 10^{-6}$
Dropout rate on input layer	$p = 0.2$
Maximum word length	$w = 4$

Table 2: Hyper-parameter settings.

Both datasets are preprocessed by replacing the continuous English characters and digits with a unique token. All experiments are conducted with standard Bakeoff scoring program[1] calculating precision, recall, and F1-score.

6.2 Hyper-parameters

Hyper-parameters of neural network model significantly impact on its performance. To determine a set of suitable hyper-parameters, we divide the training data into two sets, the first 90% sentences as training set and the rest 10% sentences as development set. We choose the hyper-parameters as shown in Table 2.

We found that the character embedding size has a limited impact on the performance as long as it is large enough. The size 50 is chosen as a good trade-off between speed and performance. The number of hidden units is set to be the same as the character embedding. Maximum word length determines the number of parameters in GCNN part and the time consuming of beam search, since the words with a length $l > 4$ are relatively rare, 0.29% in PKU training data and 1.25% in MSR training data, we set the maximum word length to 4 in our experiments.[2]

Dropout is a popular technique for improving the performance of neural networks by reducing overfitting (Srivastava et al., 2014). We also drop the input layer of our model with dropout rate 20% to avoid overfitting.

[1] http://www.sighan.org/bakeoff2003/score

[2] This 4-character limitation is just for consistence for both datasets. We are aware that this is a too strict setting, especially which makes additional performance loss in a dataset with larger average word length, i.e., MSR.

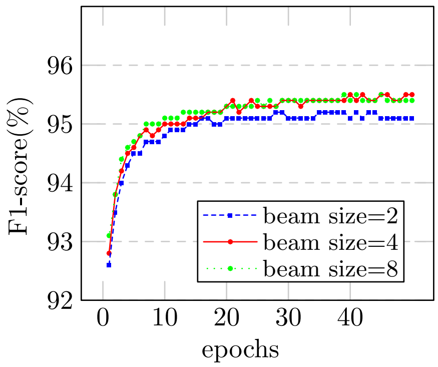

Figure 5: Performances of different beam sizes on PKU dataset.

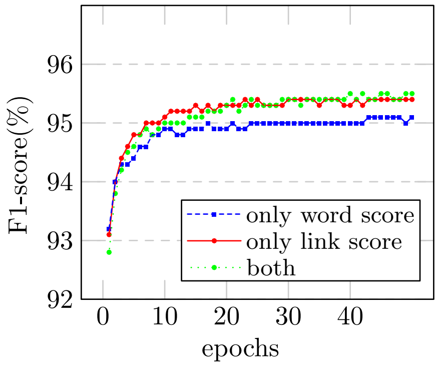

Figure 6: Performances of different score strategies on PKU dataset.

6.3 Model Analysis

Beam Size. We first investigated the impact of beam size over segmentation performance. Figure 5 shows that a segmenter with beam size 4 is enough to get the best performance, which makes our model find a good balance between accuracy and efficiency.

GCNN. We then studied the role of GCNN in our model. To reveal the impact of GCNN, we re-implemented a simplified version of our model, which replaces the GCNN part with a single nonlinear layer as in equation (1). The results are listed in Table 3, which demonstrate that the performance is significantly boosted by exploiting the GCNN architecture (94.0% to 95.5% on F1-score), while the best performance that the simplified version can achieve is 94.7%, but using a much larger character embedding size.

models	P	R	F
Single layer ($d = 50$)	94.3	93.7	94.0
GCNN ($d = 50$)	95.8	95.2	95.5
Single layer ($d = 100$)	94.9	94.4	94.7

Table 3: Performances of different models on PKU dataset.

	PKU		MSR	
+Dictionary	ours	theirs	ours	theirs
(Chen et al., 2015a)	94.9	**95.9**	95.8	96.2
(Chen et al., 2015b)	94.6	95.7	95.7	**96.4**
This work	**95.7**	-	**96.4**	-

Table 4: Comparison of using different Chinese idiom dictionaries.[4]

Link Score & Word Score. We conducted several experiments to investigate the individual effect of link score and word score, since these two types of scores are intended to estimate the sentence likelihood from two different perspectives: the semantic coherence between words and the existence of individual words. The learning curves of models with different scoring strategies are shown in Figure 6.

The model with only word score can be regarded as the situation that the segmentation decisions are made only based on local window information. The comparisons show that such a model gives moderate performance. By contrast, the model with only link score gives a much better performance close to the joint model, which demonstrates that the complete segmentation history, which can not be effectively modeled in previous schemes, possesses huge appliance value for word segmentation.

6.4 Results

We first compare our model with the latest neural network methods as shown in Table 4. However, (Chen et al., 2015a; Chen et al., 2015b) used an extra preprocess to filter out Chinese idioms according to an external dictionary.[3] Table 4 lists the results (F1-scores) with different dictionaries, which show that our models perform better when under the same settings.

[3] In detail, when a dictionary is used, a preprocess is performed before training and test, which scans original text to find out Chinese idioms included in the dictionary and replace them with a unique token. This treatment does not strictly follow the convention of closed-set setting defined by SIGHAN Bakeoff, as no linguistic resources, either dictionary or corpus, other than the training corpus, should be adopted.

[4] The dictionary used in (Chen et al., 2015a; Chen et al., 2015b) is neither publicly released nor specified the exact source until now. We have to re-run their code using our selected dictionary to make a fair comparison. Our dictionary has been submitted along with this submission.

Models	PKU			MSR		
	P	R	F	P	R	F
(Zheng et al., 2013)	92.8	92.0	92.4	92.9	93.6	93.3
(Pei et al., 2014)	93.7	93.4	93.5	94.6	94.2	94.4
(Chen et al., 2015a)*	94.6	94.2	94.4	94.6	95.6	95.1
(Chen et al., 2015b) *	94.6	94.0	94.3	94.5	95.5	95.0
This work	**95.5**	**94.9**	**95.2**	**96.1**	**96.7**	**96.4**
+Pre-trained character embedding						
(Zheng et al., 2013)	93.5	92.2	92.8	94.2	93.7	93.9
(Pei et al., 2014)	94.4	93.6	94.0	95.2	94.6	94.9
(Chen et al., 2015a)*	94.8	94.1	94.5	94.9	95.9	95.4
(Chen et al., 2015b)*	95.1	94.4	94.8	95.1	96.2	95.6
This work	**95.8**	**95.2**	**95.5**	**96.3**	**96.8**	**96.5**

Table 5: Comparison with previous neural network models. Results with * are from our runs on their released implementations.[5]

Models	PKU	MSR	PKU	MSR
(Tseng et al., 2005)	95.0	96.4	-	-
(Zhang and Clark, 2007)	94.5	97.2	-	-
(Zhao and Kit, 2008b)	95.4	97.6	-	-
(Sun et al., 2009)	95.2	97.3	-	-
(Sun et al., 2012)	95.4	97.4	-	-
(Zhang et al., 2013)	-	-	96.1*	97.4*
(Chen et al., 2015a)	94.5	95.4	96.4*	97.6*
(Chen et al., 2015b)	94.8	95.6	96.5*	97.4*
This work	95.5	96.5	-	-

Table 6: Comparison with previous state-of-the-art models. Results with * used external dictionary or corpus.

Table 5 gives comparisons among previous neural network models. In the first block of Table 5, the character embedding matrix $\mathbf{M}$ is randomly initialized. The result shows that our proposed novel model outperforms previous neural network methods.

Previous works have found that the performance can be improved by pre-training the character embeddings on large unlabeled data. Therefore, we use word2vec (Mikolov et al., 2013) toolkit[6] to pre-train the character embeddings on the Chinese Wikipedia corpus and use them for initialization. Table 5 also shows the results with additional pre-trained character embeddings. Again, our model achieves better performance than previous neural network models do.

Table 6 compares our models with previous state-of-the-art systems. Recent systems such as (Zhang et al., 2013), (Chen et al., 2015b) and (Chen et al., 2015a) rely on both extensive feature engineering and external corpora to boost performance. Such systems are not directly comparable with our models. In the closed-set setting, our models can achieve state-of-the-art performance on PKU dataset but a competitive result on MSR dataset, which can attribute to too strict maximum word length setting for consistence as it is well known that MSR corpus has a much longer average word length (Zhao et al., 2010). In later experiments. we found the performance can indeed be further improved by allowing longer words. (e.g., if 6-char words allowed, F1 score on MSR will be furthermore improved 0.3%.)

For the running cost, we roughly report the current computation consuming on PKU dataset.[7] It takes about two days to finish 50 training epochs (for results in Figure 6 and the last row of Table 6) only with two cores of an Intel i7-5960X CPU. The requirement for RAM during training is less than 800MB. The trained model can be saved within 4MB on the hard disk.

7 Related Work

Neural Network Models. Most modern CWS methods followed (Xue, 2003) treated CWS as a sequence labeling problems (Zhao et al., 2006b). Recently, researchers have tended to explore neural network based approaches (Collobert et al., 2011) to reduce efforts of feature engineering (Zheng et al., 2013; Qi et al., 2014; Chen et al., 2015a; Chen et al., 2015b). They modeled CWS

[5] To make comparisons fair, we re-run their code (https://github.com/dalstonChen) without their unspecified Chinese idiom dictionary.

[6] http://code.google.com/p/word2vec/

[7] Our code is released at https://github.com/jcyk/CWS.

as tagging problem as well, scoring tags on individual characters. In those models, tag scores are decided by context information within local windows and the sentence-level score is obtained via *context-independently* tag transitions. Pei et al. (2014) introduced the tag embedding as input to capture the combinations of context and tag history. However, in previous works, only the tag of previous *one* character was taken into consideration though theoretically the complete history of actions taken by the segmenter should be considered.

Alternatives to Sequence Labeling. Besides sequence labeling schemes, Zhang and Clark (2007) proposed a word-based perceptron method. Zhang et al. (2012) used a linear-time incremental model which can also benefits from various kinds of features including word-based features. But both of them rely heavily on massive handcrafted features.

Another notable exception is (Ma and Hinrichs, 2015), which is also an embedding-based model, but models CWS as configuration-action matching. However, again, this method only uses the context information within limited sized windows.

At the same time of this work, some other neural models (Zhang et al., 2016a; Liu et al., 2016) has been proposed, which can also leverage word-level information but quite different from ours in the basic framework.

Other Techniques. The proposed model might furthermore benefit from some techniques in recent state-of-the-art systems, such as semi-supervised learning (Zhao and Kit, 2008b; Zhao and Kit, 2008a; Sun and Xu, 2011; Zhao and Kit, 2011; Zeng et al., 2013; Zhang et al., 2013), incorporating global information (Zhao and Kit, 2007; Zhang et al., 2016b), and joint models (Qian and Liu, 2012; Li and Zhou, 2012).

8 Conclusion

This paper presents a novel neural framework for the task of Chinese word segmentation, which contains three main components: (1) a factory to produce word representation when given its governed characters; (2) a sentence-level likelihood evaluation system for segmented sentence; (3) an efficient and effective algorithm to find the best segmentation.

The proposed framework makes a latest attempt to formalize word segmentation as a direct structured learning procedure in terms of the recent distributed representation framework.

Though our system outputs results that are better than the latest neural network segmenters but comparable to all previous state-of-the-art systems, the framework remains a great of potential that can be further investigated and improved in the future.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155.

Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, and Xuanjing Huang. 2015a. Gated recursive neural network for chinese word segmentation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1744–1753.

Xinchi Chen, Xipeng Qiu, Chenxi Zhu, Pengfei Liu, and Xuanjing Huang. 2015b. Long short-term memory neural networks for chinese word segmentation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1197–1206.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The 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.

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

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

Chang-Ning Huang and Hai Zhao. 2006. Which is essential for chinese word segmentation: Character versus word. In *The 20th Pacific Asia Conference on Language, Information and Computation*, pages 1–12.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2015. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*.

Jonathan K. Kummerfeld, Taylor Berg-Kirkpatrick, and Dan Klein. 2015. An empirical analysis of optimization for max-margin nlp. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 273–279.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth Interntional Conference on Machine Learning*.

Zhongguo Li and Guodong Zhou. 2012. Unified dependency parsing of chinese morphological and syntactic structures. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1445–1454.

Yijia Liu, Wanxiang Che, Jiang Guo, Bing Qin, and Ting Liu. 2016. Exploring segment representations for neural segmentation models. *arXiv preprint arXiv:1604.05499*.

Jin Kiat Low, Hwee Tou Ng, and Wenyuan Guo. 2005. A maximum entropy approach to chinese word segmentation. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, volume 1612164, pages 448–455.

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

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *11th Annual Conference of the International Speech Communication Association*, pages 1045–1048.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Wenzhe Pei, Tao Ge, and Baobao Chang. 2014. Max-margin tensor neural network for chinese word segmentation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 293–303.

Fuchun Peng, Fangfang Feng, and Andrew McCallum. 2004. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of the 20th international conference on Computational Linguistics*, page 562.

Yanjun Qi, Sujatha G Das, Ronan Collobert, and Jason Weston. 2014. Deep learning for character-based information extraction. In *Advances in Information Retrieval*, pages 668–674.

Xian Qian and Yang Liu. 2012. Joint chinese word segmentation, pos tagging and parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 501–511.

Nathan D Ratliff, J Andrew Bagnell, and Martin Zinkevich. 2007. (approximate) subgradient methods for structured prediction. In *International Conference on Artificial Intelligence and Statistics*, pages 380–387.

Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 455–465.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Weiwei Sun and Jia Xu. 2011. Enhancing chinese word segmentation using unlabeled data. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 970–979.

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 Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 56–64.

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 Annual Meeting of the Association for Computational Linguistics*, pages 253–262.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. Lstm neural networks for language modeling. In *13th Annual Conference of the International Speech Communication Association*.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd international conference on Machine learning*, pages 896–903.

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 on Chinese language Processing*, volume 171.

Peilu Wang, Yao Qian, Hai Zhao, Frank K. Soong, Lei He, and Ke Wu. 2016. Learning distributed word representations for bidirectional lstm recurrent neural network. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

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

Xiaodong Zeng, Derek F. Wong, Lidia S. Chao, and Isabel Trancoso. 2013. Graph-based semi-supervised model for joint chinese word segmentation and part-of-speech tagging. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 770–779.

Yue Zhang and Stephen Clark. 2007. Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 840–847.

Kaixu Zhang, Maosong Sun, and Changle Zhou. 2012. Word segmentation on chinese mirco-blog data with a linear-time incremental model. In *Second CIPS-SIGHAN Joint Conference on Chinese Language Processing*, pages 41–46.

Longkai Zhang, Houfeng Wang, Xu Sun, and Mairgup Mansur. 2013. Exploring representations from unlabeled data with co-training for Chinese word segmentation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 311–321.

Meishan Zhang, Yue Zhang, and Guohong Fu. 2016a. Transition-based neural word segmentation. In *Proceedings of the 54nd Annual Meeting of the Association for Computational Linguistics*.

Zhiong Zhang, Hai Zhao, and Lianhui Qin. 2016b. Probabilistic graph-based dependency parsing with convolutional neural network. In *Proceedings of the 54nd Annual Meeting of the Association for Computational Linguistics*.

Hai Zhao and Chunyu Kit. 2007. Incorporating global information into supervised learning for chinese word segmentation. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics*, pages 66–74.

Hai Zhao and Chunyu Kit. 2008a. Exploiting unlabeled text with different unsupervised segmentation criteria for chinese word segmentation. *Research in Computing Science*, 33:93–104.

Hai Zhao and Chunyu Kit. 2008b. Unsupervised segmentation helps supervised learning of character tagging for word segmentation and named entity recognition. In *Proceedings of the Third International Joint Conference on Natural Language Processing*, pages 106–111.

Hai Zhao and Chunyu Kit. 2011. Integrating unsupervised and supervised word segmentation: The role of goodness measures. *Information Sciences*, 181(1):163–183.

Hai Zhao, Chang-Ning Huang, and Mu Li. 2006a. An improved chinese word segmentation system with conditional random field. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, volume 1082117.

Hai Zhao, Chang-Ning Huang, Mu Li, and Bao-Liang Lu. 2006b. Effective tag set selection in chinese word segmentation via conditional random field modeling. In *Proceedings of the 9th Pacific Association for Computational Linguistics*, volume 20, pages 87–94.

Hai Zhao, Chang-Ning Huang, Mu Li, and Bao-Liang Lu. 2010. A unified character-based tagging framework for chinese word segmentation. *ACM Transactions on Asian Language Information Processing*, 9(2):5.

Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. 2013. Deep learning for Chinese word segmentation and POS tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 647–657.