Accurate Linear-Time Chinese Word Segmentation via Embedding Matching

Jianqiang Ma
SFB 833 and Department of Linguistics
University of Tübingen, Germany
`jma@sfs.uni-tuebingen.de`

Erhard Hinrichs
SFB 833 and Department of Linguistics
University of Tübingen, Germany
`eh@sfs.uni-tuebingen.de`

Abstract

This paper proposes an *embedding matching* approach to Chinese word segmentation, which generalizes the traditional sequence labeling framework and takes advantage of distributed representations. The training and prediction algorithms have linear-time complexity. Based on the proposed model, a greedy segmenter is developed and evaluated on benchmark corpora. Experiments show that our greedy segmenter achieves improved results over previous neural network-based word segmenters, and its performance is competitive with state-of-the-art methods, despite its simple feature set and the absence of external resources for training.

1 Introduction

Chinese sentences are written as character sequences without word delimiters, which makes word segmentation a prerequisite of Chinese language processing. Since Xue (2003), most work has formulated Chinese word segmentation (CWS) as *sequence labeling* (Peng et al., 2004) with character position tags, which has lent itself to structured discriminative learning with the benefit of allowing rich features of *segmentation configurations*, including (i) *context* of character/word n-grams within local windows, (ii) *segmentation history* of previous characters, or the combinations of both. These feature-based models still form the backbone of most state-of-the art systems.

Nevertheless, many feature weights in such models are inevitably poorly estimated because the number of parameters is so large with respect to the limited amount of training data. This has motivated the introduction of low-dimensional, real-valued vectors, known as *embeddings*, as a tool to deal with the sparseness of the input. Embeddings allow linguistic units appearing in similar contexts to share similar vectors. The success of embeddings has been observed in many NLP tasks. For CWS, Zheng et al. (2013) adapted Collobert et al. (2011) and uses character embeddings in local windows as input for a two-layer network. The network predicts individual character position tags, the transitions of which are learned separately. Mansur et al. (2013) also developed a similar architecture, which labels individual characters and uses character bigram embeddings as additional features to compensate the absence of sentence-level modeling. Pei et al. (2014) improved upon Zheng et al. (2013) by capturing the combinations of context and history via a tensor neural network.

Despite their differences, these CWS approaches are all sequence labeling models. In such models, the target character can only influence the prediction as features. Consider the the segmentation configuration in (1), where the dot appears before the target character in consideration and the box (□) represents any character that can occur in the configuration. In that example, the known history is that the first two characters 中国 'China' are joined together, which is denoted by the underline.

(1) $\underline{\text{中国}}$·□ 格外 (where □ ∈ {风, 规, ...})

(2) $\underline{\text{中国风}}$ $\underline{\text{格外}}$ 'China-style especially'

(3) $\underline{\text{中国}}$ $\underline{\text{规格}}$ $\underline{\text{外}}$ 'besides Chinese spec.'

For possible target characters, 风 'wind' and 规 'rule', the correct segmentation decisions for them are *opposite*, as shown in (2) and (3), respectively. In order to correctly predict both, current models can set higher weights for target character-specific features. However, in general, 风 is more likely to start a new word instead of joining the existing one as in this example. Given such conflicting evidence, models can rarely find optimal feature weights, if they exist at all.

The crux of this *conflicting evidence* problem is that similar configurations can suggest opposite decisions, depending on the target character and vice versa. Thus it might be useful to treat segmentation decisions for distinct characters separately. And instead of predicting general segmentation decisions given configurations, it could be beneficial to model the *matching* between configurations and *character-specific* decisions.

To this end, this paper proposes an embedding matching approach (Section 2) to CWS, in which embeddings for both input and output are learned and used as representations to counteract sparsities. Thanks to embeddings of character-specific decisions (actions) serving as both input features and output, our hidden-layer-free architecture (Section 2.2) is capable of capturing prediction histories in similar ways as the hidden layers in recurrent neural networks (Mikolov et al., 2010). We evaluate the effectiveness of the model via a linear-time greedy segmenter (Section 3) implementation. The segmenter outperforms previous embedding-based models (Section 4.2) and achieves state-of-the-art results (Section 4.3) on a benchmark dataset. The main contributions of this paper are:

- A novel embedding matching model for Chinese word segmentation.
- Developing a greedy word segmenter, which is based on the matching model and achieves competitive results.
- Introducing the idea of character-specific segmentation action embeddings as both feature and output, which are cornerstones of the model and the segmenter.

2 Embedding Matching Models for Chinese Word Segmentation

We propose an embedding based matching model for CWS, the architecture of which is shown in Figure 1. The model employs trainable embeddings to represent both sides of the matching, which will be specified shortly, followed by details of the architecture in Section 2.2.

2.1 Segmentation as Configuration-Action Matching

Output. The word segmentation output of a character sequence can be described as a sequence of *character-specific segmentation actions*. We use **separation** (***s***) and **combination** (***c***) as possible actions for each character, where a separation action starts a new word with the current character, while a combination action appends the character to the preceding ones. We model character-action *combinations* instead of atomic, character independent actions. As a running example, sentence (4b) is the correct segmentation for (4a), which can be represented as the sequence (猫 -***s***, 占 -***s***, 领 -***c***, 了 -***s***, 婴 -***s***, 儿 -***c***, 床 -***c***) .

(4) a. 猫占领了婴儿床
b. 猫 占领 了 婴儿床
c. ‘The cat occupied the crib’

Input. The input are the segmentation configurations for each character under consideration, which are described by context and history features. The *context features* of captures the characters that are in the same sentence of the current character and the *history features* encode the segmentation actions of previous characters.

- **Context features**. These refer to character *unigrams* and *bigrams* that appear in the local context window of h characters that centers at c_i, where c_i is 领 in example (4) and $h = 5$ is used in this paper. The template for features are shown in Table 1. For our example, the uni- and bi-gram features would be: 猫, 占, 领, 了, 婴 and 猫占, 占领, 领了, 了婴, respectively.
- **History features**. To make inference tractable, we assume that only previous l character-specific actions are relevant, where $l = 2$ for this study. In our example, 猫 -***s*** and 战 -***s*** are the history features. Such features capture partial information of syntactic and semantic dependencies between previous *words*, which are clues for segmentation that pure character contexts could not provide. A dummy character START is used to represent the absent (left) context characters in the case of the first l characters in a sentence. And the predicted action for the START symbol is always ***s***.

Matching. CWS is now modeled as the matching of the input (segmentation configuration) and output (two possible character-specific actions) for each character. Formally, a matching model learns

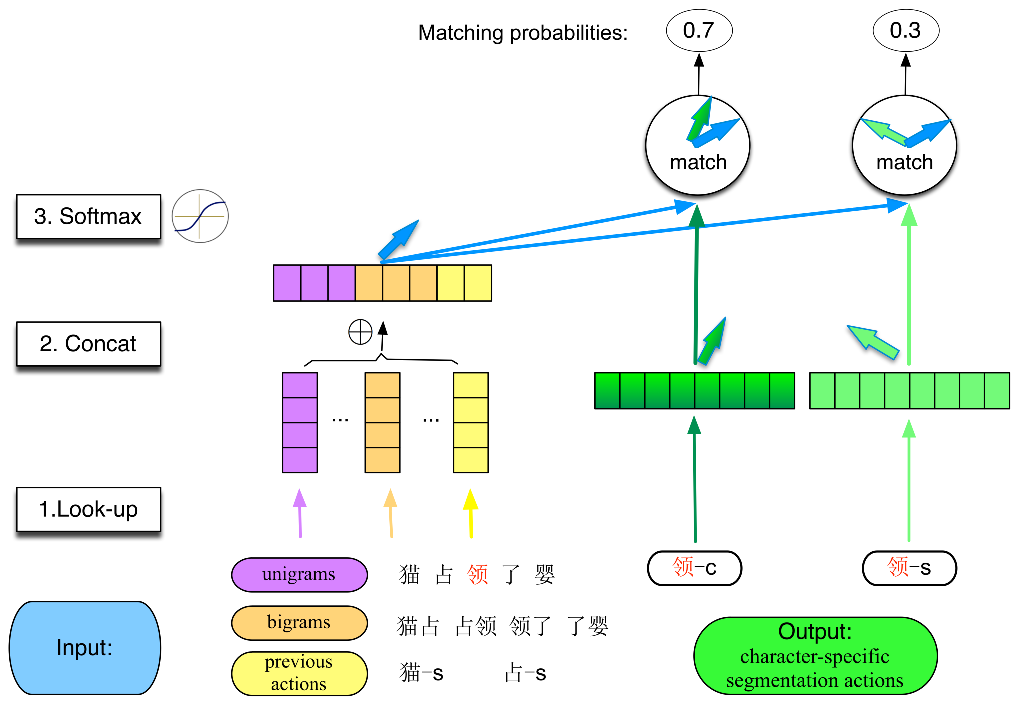

Figure 1: **The architecture of the embedding matching model for CWS**. The model predicts the segmentation for the character 领 in sentence (4), which is the second character of word 占领 'occupy'. Both feature and output embeddings are trainable parameters of the model.

Group	Feature template
unigram	$c_{i-2}, c_{i-1}, c_i, c_{i+1}, c_{i+2}$
bigram	$c_{i-2}c_{i-1}, c_{i-1}c_i, c_ic_{i+1}, c_{i+1}c_{i+2}$

Table 1: Uni- and bi-gram feature template

the following function:

$$g\,(\,b_1b_2...b_n, a_1a_2...a_n)$$
$$= \prod_{j=1}^{n} f\left(b_j(a_{j-2}, a_{j-1}; c_{j-\frac{h}{2}}...c_{j+\frac{h}{2}}), a_j\right) \quad (1)$$

where $c_1c_2...c_n$ is the character sequence, b_j and a_j are the segmentation configuration and action for character c_j, respectively. In (1), $b_j(a_{j-2}, a_{j-1}; c_{j-\frac{h}{2}}...c_{j+\frac{h}{2}})$ indicates that the configuration for each character is a function that depends on the actions of the previous l characters and the characters in the local window of size h.

Why embedding. The above matching model would suffer from sparsity if these outputs (character-specific action a_j) were directly encoded as one-hot vectors, since the matching model can be seen as a sequence labeling model with $C \times L$ outputs, where L is the number of original labels while C is the number of unique characters. For Chinese, C is at the order of $10^3 - 10^4$. The use of embeddings, however, can serve the matching model well thanks to their low dimensionality.

2.2 The Architecture

The proposed architecture (Figure 1) has three components, namely look-up table, concatenation and softmax function for matching. We will go through each of them in this section.

Look-up table. The mapping between features/outputs to their corresponding embeddings are kept in a *look-up table*, as in many previous embedding related work (Bengio et al., 2003; Pei et al., 2014). Such features are extracted from the training data. Formally, the embedding for each distinct feature d is denoted as $Embed(d) \in \mathbb{R}^N$, which is a real valued vector of dimension N. Each feature is retrieved by its unique index. The retrieval of the embeddings for the output actions is similar.

Concatenation. To predict the segmentation for the target character c_j, its feature vectors are *concatenated* into a single vector, the *input embedding*, $\mathbf{i}(b_j) \in \mathbb{R}^{N \times K}$, where K is the number of features used to describe the configuration b_j.

Softmax. The model then computes the dot product of the input embedding $\mathbf{i}(b_j)$ and each of the two *output embeddings*, $\mathbf{o}(a_{j,1})$ and $\mathbf{o}(a_{j,2})$, which represent the two possible segmentation actions for the target character c_j, respectively. The exponential of the two raw scores are normalized to obtain probabilistic values $\in [0,1]$.

We call the resulting scores *matching probabilities*, which denote probabilities that actions match the given segmentation configuration. In our example, 领 -***c*** has the probability of 0.7 to be the correct action, while 领 -***s*** is less likely with a lower probability of 0.3. Formally, the above matching procedure can be described as a *softmax* function, as shown in (2), which is also an individual f term in (1).

$$f(\,b_j, a_{j,k}) = \frac{exp\,(\mathbf{i}(b_j)\cdot\mathbf{o}(a_{j,k}))}{\sum_{k'} exp\,\big(\mathbf{i}(b_j)\cdot\mathbf{o}(a_{j,k'})\big)} \qquad (2)$$

In (2), $a_{j,k}$ ($1 \le k \le 2$) represent two possible actions, such as 领 -***c*** and 领 -***s*** for 领 in our example. Note that, to ensure the input and output are of the same dimension, for each character specific action, the model trains two distinct embeddings, one $\in \mathbb{R}^N$ as feature and the other $\in \mathbb{R}^{N\times K}$ as output, where K is the number of features for each input.

Best word segmentation of sentence. After plugging (2) into (1) and applying (and then dropping) logarithms for computational convenience, finding the best segmentation for a sentence becomes an optimization problem as shown in (3). In the formula, $\hat{Y}$ is the best action sequence found by the model among all the possible ones, $Y = a_1a_2...a_n$, where a_j is the predicted action for the character c_j ($1 \le j \le n$), which is either c_j-***s*** or c_j-***c***, such as 领 -***s*** and 领 -***c***.

$$\hat{Y} = \operatorname*{argmax}_{Y} \sum_{j=1}^{n} \frac{exp\,(\mathbf{i}(b_j)\cdot\mathbf{o}(a_j))}{\sum_k exp\,(\mathbf{i}(b_j)\cdot\mathbf{o}(a_{j,k}))} \qquad (3)$$

3 The Greedy Segmenter

Our model depends on the actions predicted for the previous two characters as history features. Traditionally, such scenarios call for dynamic programming for exact inference. However, preliminary experiments showed that, for our model, a Viterbi search based segmenter, even supported by conditional random field (Lafferty et al., 2001) style training, yields similar results as the greedy search based segmenter in this section. Since the greedy segmenter is much more efficient in training and testing, the rest of the paper will focus on the proposed greedy segmenter, the details of which will be described in this section.

3.1 Greedy Search

Initialization. The first character in the sentence is made to have two left side characters that are dummy symbols of START, whose predicted actions are always START-***s***, i.e. separation.

Iteration. The algorithms predicts the action for each character c_j, one at a time, in a left-to-right, incremental manner, where $1 \le j \le n$ and n is the sentence length. To do so, it first extracts context features and history features, the latter of which are the predicted character-specific actions for the previous two characters. Then the model matches the concatenated feature embedding with embeddings of the two possible character-specific actions, c_j-***s*** and c_i-***c***. The one with higher matching probability is predicted as segmentation action for the character, which is irreversible. After the action for the last character is predicted, the segmented word sequence of the sentence is built from the predicted actions deterministically.

Hybrid matching. Character-specific embeddings are capable of capturing subtle word formation tendencies of individual characters, but such representations are incapable of covering matching cases for unknown target characters. Another minor issue is that the action embeddings for certain low frequent characters may not be sufficiently trained. To better deal with these scenarios, We also train two embeddings to represent character-independent segmentation actions, ALL-***s*** and ALL-***c***, and use them to average with or substitute embeddings of infrequent or unknown characters, which are either insufficiently trained or nonexistent. Such strategy is called *hybrid matching*, which can improve accuracy.

Complexity. Although the total number of actions is large, the matching for each target character only requires the two actions that correspond to that specific character, such as 领 -***s*** and 领 -***c*** for 领 in our example. Each prediction is thus similar to a softmax computation with two outputs, which costs constant time C. Greedy search ensures that the total time for predicting a sentence of n characters is $n \times C$, i.e. *linear time complexity*, with a minor overhead for mapping actions to segmentations.

3.2 Training

The training procedure first predicts the action for the current character with current parameters, and then optimizes the log likelihood of correct segmentation actions in the gold segmentations to update parameters. Ideally, the matching probability for the correct action embedding should be 1 while that of the incorrect one should be 0. We minimize the *cross-entropy loss function* as in (4) for the segmentation prediction of each character c_j to pursue this goal. The loss function is convex, similar to that of maximum entropy models.

$$J = -\sum_{k=1}^{K} \delta\left(a_{j,k}\right) \log \frac{exp\left(\mathbf{i} \cdot \mathbf{o}(a_{j,k})\right)}{\sum_{k'} exp\left(\mathbf{i} \cdot \mathbf{o}(a_{j,k'})\right)} \quad (4)$$

where $a_{j,k}$ denotes a possible action for c_j and $\mathbf{i}$ is a compact notation for $\mathbf{i}(b_j)$. In (4), $\delta(a_{j,k})$ is an *indicator function* defined by the following formula, where $\hat{a_j}$ denotes the correct action.

$$\delta(a_{j,k}) = \begin{cases} 1, & \text{if } a_{j,k} = \hat{a_j} \\ 0, & \text{otherwise} \end{cases}$$

To counteract over-fitting, we add L2 regularization term to the loss function, as follows:

$$J = J + \sum_{k=1}^{K} \frac{\lambda}{2} \left(||\mathbf{i}||^2 + ||\mathbf{o}(a_{j,k})||^2\right) \quad (5)$$

The formula in (4) and (5) are similar to that of a standard softmax regression, except that both input and output embeddings are parameters to be trained. We perform *stochastic gradient descent* to update input and output embeddings in turn, each time considering the other as constant. We give the gradient (6) and the update rule (7) for the input embedding $\mathbf{i}(b_j)$ ($\mathbf{i}$ for short), where $\mathbf{o}_k$ is a short notation for $\mathbf{o}(a_{j,k})$. The gradient and update for output embeddings are similar. The α in (7) is the learning rate, which we use a linear decay scheme to gradually shrink it from its initial value to zero. Note that the update for the input embedding $\mathbf{i}$ is actually performed for the feature embeddings that form $\mathbf{i}$ in the concatenation step.

$$\frac{\partial J}{\partial \mathbf{i}} = \sum_{k} \left(f\left(b_j, a_{j,k}\right) - \delta\left(a_{j,k}\right)\right) \cdot \mathbf{o}_k + \lambda \mathbf{i} \quad (6)$$

$$\mathbf{i} = \mathbf{i} - \alpha \frac{\partial J}{\partial \mathbf{i}} \quad (7)$$

Complexity. For each iteration of the training process, the time complexity is also linear to the input character number, as compared with search, only a few constant time operations of gradient computation and parameter updates are performed for each character.

4 Experiments

4.1 Data and Evaluation Metric

In the experiments, we use two widely used and freely available[1] manually word-segmented corpora, namely, **PKU** and **MSR**, from the second SIGHAN international Chinese word segmentation bakeoff (Emerson, 2005). Table 2 shows the details of the two dataset. All evaluations in this paper are conducted with official training/testing set split using official scoring script.[2]

	PKU	**MSR**
Word types	5.5×10^4	8.8×10^4
Word tokens	1.1×10^6	2.4×10^6
Character types	5×10^3	5×10^3
Character tokens	1.8×10^6	4.1×10^6

Table 2: Corpus details of PKU and MSR

The segmentation accuracy is evaluated by precision (P), recall (R), **F-score** and $\mathbf{R}_{oov}$, the recall for out-of-vocabulary words. Precision is defined as the number of correctly segmented words divided by the total number of words in the segmentation result. Recall is defined as the number of correctly segmented words divided by the total number of words in the gold standard segmentation. In particular, R_{oov} reflects the model generalization ability. The metric for overall performance, the evenly-weighted F-score is calculated as in (8):

$$F = \frac{2 \times P \times R}{P + R} \quad (8)$$

To comply with CWS evaluation conventions and make comparisons fair, we distinguish the following two settings:

- *closed-set*: no extra resource other than training corpora is used.
- *open-set*: additional lexicon, raw corpora, etc are used.

[1]http://www.sighan.org/bakeoff2005/

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

We will report the final results of our model[3] on PKU and MSR corpora in comparison with previous embedding based models (Section 4.2) and state-of-the-art systems (Section 4.3), before going into detailed experiments for model analyses (Section 4.5).

4.2 Comparison with Previous Embedding-Based Models

Table 3 shows the results of our greedy segmenter on the PKU and MSR datasets, which are compared with embedding-based segmenters in previous studies.[4] In the table, results for both closed-set and open-set setting are shown for previous models. In the open-set evaluations, all three previous work use *pre-training* to train character ngram embeddings from large unsegmented corpora to initialize the embeddings, which will be later trained with the manually word-segmented training data. For our model, we report the close-set results only, as pre-training does not significant improve the results in our experiments (Section 4.5).

As shown in Table 3, under close-set evaluation, our model significantly outperform previous embedding based models in all metrics. Compared with the previous best embedding-based model, our greedy segmenter has achieved up to 2.2% and 25.8% absolute improvements (MSR) on F-score and R_{oov}, respectively. Surprisingly, our close-set results are also comparable to the best *open-set* results of previous models. As we will see in (Section 4.4), when using same or less character uni- and bi-gram features, our model still outperforms previous embedding based models in closed-set evaluation, which shows the effectiveness of our matching model.

Significance test. Table 4 shows the 95% confidence intervals (CI) for close-set results of our model and the best performing previous model (Pei et al., 2014), which are computed by formula (9), following (Emerson, 2005).

$$CI = 2\sqrt{\frac{F(1-F)}{N}} \tag{9}$$

where F is the F-score value and the N is the word token count of the testing set, which is 104,372 and 106,873 for PKU and MSR, respectively. We see that the confidence intervals of our results do *not* overlap with that of (Pei et al., 2014), meaning that our improvements are statistically significant.

4.3 Comparison with the State-of-the-Art Systems

Table 5 shows that the results of our greedy segmenter are competitive with the state-of-the-art supervised systems (Best05 closed-set, Zhang and Clark, 2007), although our feature set is much simpler. More recent state-of-the-art systems rely on both extensive feature engineering and extra raw corpora to boost performance, which are semi-supervised learning. For example, Zhang et al (2013) developed 8 types of static and dynamic features to maximize the co-training system that used extra corpora of Chinese Gigaword and Baike, each of which contains more than 1 billion character tokens. Such systems are not directly comparable with our supervised model. We leave the development of semi-supervised learning methods for our model as future work.

4.4 Features Influence

Table 6 shows the F-scores of our model on PKU dataset when different features are removed ('w/o') or when only a subset of features are used. Features complement each other and removing any group of features leads to a limited drop of F-score up to 0.7%. Note that features of previous (two) actions are even more informative than all unigram features combined, suggesting that intra- an inter-word dependencies reflected by action features are strong evidence for segmentation. Moreover, using same or less character ngram features, our model outperforms previous embedding based models, which shows the effectiveness of our matching model.

4.5 Model Analysis

Learning curve. Figure 2 shows that the training procedure coverages quickly. After the first iteration, the testing F-scores are already 93.5% and 95.7% for PKU and MSR, respectively, which then gradually reach their maximum within the next 9 iterations before the curve flats out.

Speed. With an unoptimized single-thread Python implementation running on a laptop with intel Core-i5 CPU (1.9 GHZ), each iteration of the training procedure on PKU dataset takes about 5 minutes, or 6,000 characters per second. The pre-

[3]Our implementation: https://zenodo.org/record/17645.

[4]The results for Zheng et al. (2013) are from the re-implementation of Pei et al. (2014).

Models	**PKU Corpus**				**MSR Corpus**			
	P	**R**	**F**	**R$_{oov}$**	**P**	**R**	**F**	**R$_{oov}$**
Zheng et al.(2013)	92.8	92.0	92.4	63.3	92.9	93.6	93.3	55.7
+ *pre-training*†	93.5	92.2	92.8	69.0	94.2	93.7	93.9	64.1
Mansur et al. (2013)	93.6	92.8	93.2	57.9	92.3	92.2	92.2	53.7
+ *pre-training*†	94.0	93.9	94.0	69.5	93.1	93.1	93.1	59.7
Pei et al. (2014)	93.7	93.4	93.5	64.2	94.6	94.2	94.4	61.4
+ *pre-training*†	94.4	93.6	94.0	69.0	95.2	94.6	94.9	64.8
+ *pre-training & bigram*†	-	-	**95.2**	-	-	-	**97.2**	-
This work (closed-set)	**95.5**	**94.6**	95.1	**76.0**	**96.6**	**96.5**	96.6	**87.2**

Table 3: Comparison with previous embedding based models. Numbers in percentage. Results with † used extra corpora for (pre-)training.

Models	**PKU**		**MSR**	
	F	CI	F	CI
Pei et al.	93.5	±0.15	94.4	±0.14
This work	95.1	±0.13	96.6	±0.11

Table 4: Significance test of closed-set results of Pei et al (2014) and our model.

Model	**PKU**	**MSR**
Best05 closed-set	95.0	96.4
Zhang et al. (2006)	95.1	97.1
Zhang and Clark (2007)	94.5	97.2
Wang et al. (2012)	94.1	97.2
Sun et al. (2009)	95.2	97.3
Sun et al. (2012)	95.4	**97.4**
Zhang et al. (2013) †	**96.1**	**97.4**
This work	95.1	96.6

Table 5: Comparison with the state-of-the-art systems. Results with † used extra lexicon/raw corpora for training, i.e. in open-set setting. Best05 refers to the best closed-set results in 2nd SIGHAN bakeoff.

Feature	F-score	Feature	F-score
All features	**95.1**	uni-&bi-gram	94.6
w/o action	94.6	only action	93.3
w/o unigram	94.8	only unigram	92.1
w/o bigram	94.4	only bigram	94.2

Table 6: The influence of features. F-score in percentage on the PKU corpus.

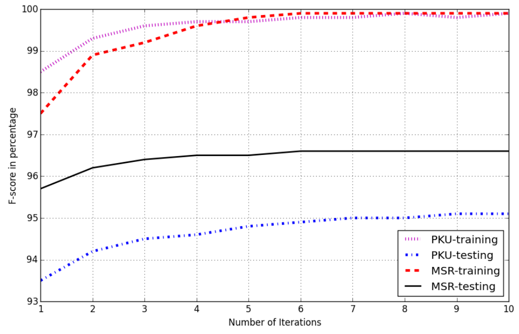

Figure 2: The learning curve of our model.

diction speed is above 13,000 character per second.

Hyper parameters. The hyper parameters used in the experiments are shown in Table 7. We initialized hyper parameters with recommendations in literature before tuning with dev-set experiments, each of which change one parameter by a magnitude. We fixed the hyper parameter to the current setting without spending too much time on tuning, since that is not the main purpose of this paper.

- **Embedding size** determines the number of parameters to be trained, thus should fit the training data size to achieve good performance. We tried the size of 30 and 100, both of which performs worse than 50. A possible tuning is to use different embedding size for different groups of features instead of setting $N_1 = 50$ for all features.
- **Context window size**. A window size of 3-5 characters achieves comparable results. Zheng et al. (2013) suggested that context window larger than 5 may lead to inferior results.
- **Initial Learning rate**. We found that several learning rates between 0.04 to 0.15 yielded very similar results as the one reported here. The training is not very sensitive to reason-

able values of initial learning rate. However, Instead of our simple linear decay of learning rate, it might be useful to try more sophisticated techniques, such as *AdaGrad* and exponential decaying (Tsuruoka et al., 2009; Sun et al., 2013).

- **Regularization**. Our model suffers a little from over-fitting, if no regularization is used. In that case, the F-score on PKU drops from 95.1% to 94.7%.
- **Pre-training**. We tried pre-training character embeddings using *word2vec*[5] with Chinese Gigaword Corpus[6] and use them to initialize the corresponding embeddings in our model, as previous work did. However, we were only able to see insignificant F-score improvements within 0.1% and observed that the training F-score reached 99.9% much earlier. We hypothesize that pre-training leads to sub-optimal local maximums for our model.
- **Hybrid matching**. We tried applying hybrid matching (Section 3.1) for target characters which are less frequent than the top f_{top} characters, including unseen characters, which leads to about 0.15% of F-score improvements.

Size of feature embed'	$N_1 = 50$
Size of output embed'	$N_2 = 550$
Window size	$h = 5$
Initial learning rate	$\alpha = 0.1$
Regularization	$\lambda = 0.001$
Hybrid matching	$f_{top} = 8\%$

Table 7: Hyper parameters of our model.

5 Related Work

Word segmentation. Most modern segmenters followed Xue (2003) to model CWS as sequence labeling of character position tags, using conditional random fields (Peng et al. 2004), structured perceptron (Jiang et al., 2008), etc. Some notable exceptions are (Zhang and Clark, 2007; Zhang et al., 2012), which exploited rich word-level features and (Ma et al., 2012; Ma, 2014; Zhang et al., 2014), which explicitly model word structures. Our work generalizes the sequence labeling to a more flexible framework of matching, and predicts actions as in (Zhang and Clark, 2007; Zhang et al., 2012) instead of position tags to prevent the greedy search from suffering tag inconsistencies. To better utilize resources other than training data, our model might benefit from techniques used in recent state-of-the-art systems, such as semi-supervised learning (Zhao and Kit, 2008; Sun and Xu, 2011; Zhang et al., 2013; Zeng et al., 2013), joint models (Li and Zhou, 2012; Qian and Liu, 2012), and partial annotations (Liu et al., 2014; Yang and Vozila, 2014).

Distributed representation and CWS. Distributed representation are useful for various NLP tasks, such as POS tagging (Collobert et al., 2011), machine translation (Devlin et al., 2014) and parsing (Socher et al., 2013). Influenced by Collobert et al. (2011), Zheng et al. (2013) modeled CWS as tagging and treated *sentence-level* tag sequence as the combination of individual tag predictions and context-independent tag transition. Mansur et al. (2013) was inspired by Bengio et al. (2003) and used character bigram embeddings to compensate for the absence of sentence level optimization. To model interactions between tags and characters, which are absent in these two CWS models, Pei et al. (2014) introduced the tag embedding and used a tensor hidden layer in the neural net. In contrast, our work uses character-specific action embeddings to *explicitly* capture such interactions. In addition, our work gains efficiency by avoiding hidden layers, similar as Mikolov et al. (2013).

Learning to match. Matching heterogeneous objects has been studied in various contexts before, and is currently flourishing, thanks to embedding-based deep (Gao et al., 2014) and convolutional (Huang et al., 2013; Hu et al., 2014) neural networks. This work develops a matching model for CWS and differs from others in its “shallow” yet effective architecture.

6 Discussion

Simple architecture. It is possible to adopt standard feed-forward neural network for our embedding matching model with character-action embeddings as both feature and output. Nevertheless, we designed the proposed architecture to avoid hidden layers for simplicity, efficiency and easy-tuning, inspired by *word2vec*. Our simple architecture is effective, demonstrated by the improved results over previous neural-network word seg-

[5]https://code.google.com/p/word2vec/

[6]https://catalog.ldc.upenn.edu/LDC2005T14

menters, all of which use feed-forward architecture with different features and/or layers. It might be interesting to directly compare the performances of our model with same features on the current and feed-forward architectures, which we leave for future work.

Greedy and exact search-based models. As mentioned in Section 3, we implemented and preliminarily experimented with a segmenter that trains a similar model with exact search via Viterbi algorithm. On the PKU corpus, its F-score is 0.944, compared with greedy segmenter's 0.951. Its training and testing speed are up to 7.8 times slower than that of the greedy search segmenter. It is counter-intuitive that the performance of the exact-search segmenter is no better or even worse than that of the greedy-search segmenter. We hypothesize that since the training updates parameters with regard to search errors, the final model is "tailored" for the specific search method used, which makes the model-search combination of greedy search segmenter not necessarily worse than that of exact search segmenter. Another way of looking at it is that search is less important when the model is accurate. In this case, most step-wise decisions are correct in the first place, which requires no correction from the search algorithm. Empirically, Zhang and Clark (2011) also reported exact-search segmenter performing worse than beam-search segmenters.

Despite that the greedy segmenter is incapable of considering future labels, this rarely causes problems in practice. Our greedy segmenter has good results, compared with the exact-search segmenter above and previous approaches, most of which utilize exact search. Moreover, the greedy segmenter has additional advantages of faster training and prediction.

Sequence labeling and matching. A traditional sequence labeling model such as CRF has K (number of labels) target-character-independent weight vectors, where the target character influences the prediction via the weights of the features that contain it. In a way, a matching model can be seen as a family of "sub-models", which keeps a group of weight vectors (the output embeddings) for *each* unique target character. Different target characters activate different sub-models, allowing opposite predictions for similar input features, as the target weight vectors used are different.

7 Conclusion and Future Work

In this paper, we have introduced the matching formulation for Chinese word segmentation and proposed an embedding matching model to take advantage of distributed representations. Based on the model, we have developed a greedy segmenter, which outperforms previous embedding-based methods and is competitive with state-of-the-art systems. These results suggest that it is promising to model CWS as configuration-action matching using distributed representations. In addition, linear-time training and testing complexity of our simple architecture is very desirable for industrial application. To the best of our knowledge, this is the first greedy segmenter that is competitive with the state-of-the-art discriminative learning models.

In the future, we plan to investigate methods for our model to better utilize external resources. We would like to try using convolutional neural network to automatically encode ngram-like features, in order to further shrink parameter space. It is also interesting to study whether extending our model with deep architectures can benefit CWS. Lastly, it might be useful to adapt our model to tasks such as POS tagging and name entity recognition.

Acknowledgments

The authors would like to thank the anonymous reviewers for their very helpful and constructive suggestions. We are indebted to Çağrı Çöltekin for discussion and comments, to Dale Gerdemann, Cyrus Shaoul, Corina Dima, Sowmya Vajjala and Helmut Schmid for their useful feedback on an earlier version of the manuscript. Financial support for the research reported in this paper was provided by the German Research Foundation (DFG) as part of the Collaborative Research Center "Emergence of Meaning" (SFB 833) and by the German Ministry of Education and Technology (BMBF) as part of the research grant CLARIN-D.

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.

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.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of ACL*, pages 1370–1380.

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

Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, Li Deng, and Yelong Shen. 2014. Modeling interestingness with deep neural networks. In *Proceedings of EMNLP*, pages 2–13.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems*, pages 2042–2050.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the ACM International Conference on Information & Knowledge Management*, pages 2333–2338.

Wenbin Jiang, Liang Huang, Qun Liu, and Yajuan Lü. 2008. A cascaded linear model for joint Chinese word segmentation and part-of-speech tagging. In *Proceedings of ACL*, pages 897–904.

John Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proceedings of International Conference on Machine Learning*, pages 282–289.

Zhongguo Li and Guodong Zhou. 2012. Unified dependency parsing of Chinese morphological and syntactic structures. In *Proceedings of EMNLP*, pages 1445–1454.

Yijia Liu, Yue Zhang, Wanxiang Che, Ting Liu, and Fan Wu. 2014. Domain adaptation for CRF-based Chinese word segmentation using free annotations. In *Proceedings of EMNLP*, pages 864–874.

Jianqiang Ma, Chunyu Kit, and Dale Gerdemann. 2012. Semi-automatic annotation of Chinese word structure. In *Proceedings of the Second CIPS-SIGHAN Joint Conference on Chinese Language Processing*, pages 9–17.

Jianqiang Ma. 2014. Automatic refinement of syntactic categories in Chinese word structures. In *Proceedings of LREC*.

Mairgup Mansur, Wenzhe Pei, and Baobao Chang. 2013. Feature-based neural language model and Chinese word segmentation. In *Proceedings of IJCNLP*, pages 1271–1277.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of INTERSPEECH*, pages 1045–1048.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Wenzhe Pei, Tao Ge, and Chang Baobao. 2014. Max-margin tensor neural network for Chinese word segmentation. In *Proceedings of ACL*, pages 239–303.

Fuchun Peng, Fangfang Feng, and Andrew McCallum. 2004. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of COLING*, pages 562–571.

Xian Qian and Yang Liu. 2012. Joint Chinese word segmentation, POS tagging and parsing. In *Proceedings of EMNLP-CoNLL*, pages 501–511.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In *Proceedings of ACL*, pages 455–465.

Weiwei Sun and Jia Xu. 2011. Enhancing Chinese word segmentation using unlabeled data. In *Proceedings of EMNLP*, 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 NAACL*, 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 ACL*, pages 253–262.

Xu Sun, Yaozhong Zhang, Takuya Matsuzaki, Yoshimasa Tsuruoka, and Jun’ichi Tsujii. 2013. Probabilistic Chinese word segmentation with non-local information and stochastic training. *Information Processing & Management*, 49(3):626–636.

Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. 2009. Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. In *Proceedings of ACL-IJCNLP*, pages 477–485.

Kun Wang, Chengqing Zong, and Keh-Yih Su. 2012. Integrating generative and discriminative character-based models for Chinese word segmentation. *ACM Transactions on Asian Language Information Processing (TALIP)*, 11(2):7.

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

Fan Yang and Paul Vozila. 2014. Semi-supervised Chinese word segmentation using partial-label learning With conditional random fields. In *Proceedings of EMNLP*, page 90–98.

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 ACL*, pages 770–779.

Yue Zhang and Stephen Clark. 2007. Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of ACL*, pages 840–847.

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

Ruiqiang Zhang, Genichiro Kikui, and Eiichiro Sumita. 2006. Subword-based tagging by conditional random fields for Chinese word segmentation. In *Proceedings of NAACL*, pages 193–196.

Kaixu Zhang, Maosong Sun, and Changle Zhou. 2012. Word segmentation on Chinese mirco-blog data with a linear-time incremental model. In *Proceedings of the 2nd CIPS-SIGHAN Joint Conference on Chinese Language Processing*, pages 41–46.

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

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2014. Character-level Chinese dependency parsing. In *Proceedings of ACL*, pages 1326–1336.

Hai Zhao and Chunyu Kit. 2008. Unsupervised segmentation helps supervised learning of character tagging for word segmentation and named entity recognition. In *Proceedings of IJCNLP*, pages 106–111.

Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. 2013. Deep Learning for Chinese Word Segmentation and POS Tagging. In *Proceedings of EMNLP*, pages 647–657.