

Article

# Dependency Parsing with Transformed Feature

Fuxiang Wu

School of Astronautics, Beihang University, Beijing 100191, China; fxwuedu@buaa.edu.cn

Academic Editor: Günter Neumann

Received: 2 November 2016; Accepted: 16 January 2017; Published: 21 January 2017

**Abstract:** Dependency parsing is an important subtask of natural language processing. In this paper, we propose an embedding feature transforming method for graph-based parsing, transform-based parsing, which directly utilizes the inner similarity of the features to extract information from all feature strings including the un-indexed strings and alleviate the feature sparse problem. The model transforms the extracted features to transformed features via applying a feature weight matrix, which consists of similarities between the feature strings. Since the matrix is usually rank-deficient because of similar feature strings, it would influence the strength of constraints. However, it is proven that the duplicate transformed features do not degrade the optimization algorithm: the margin infused relaxed algorithm. Moreover, this problem can be alleviated by reducing the number of the nearest transformed features of a feature. In addition, to further improve the parsing accuracy, a fusion parser is introduced to integrate transformed and original features. Our experiments verify that both transform-based and fusion parser improve the parsing accuracy compared to the corresponding feature-based parser.

**Keywords:** transform-based parser; feature-based parser; transformed feature; feature weight matrix; parser fusion

---

## 1. Introduction

Recently, traditional models based on discrete features have been employed in many typical natural language processing tasks, such as named entity recognition [1], word segmentation [2], semantic similarity assessment [3], etc. In dependency parsing, there are many feature-based parsers that have achieved high parsing accuracy such as ZPar [4], DuDuPlus [5], etc. There are two main kinds of parsing algorithms: graph-based parsing [6–9] and transition-based parsing [10,11]. The graph-based parsing searches for the best dependency tree from all trees that are formed with words in a sentence. However, a huge number of trees of a long sentence will slow down the searching of the best tree. The typical transition-based parsing makes decisions with the best score in the parsing process by using a greedy strategy based on a shift–reduce algorithm. The greedy strategy makes its computational complexity much lower than that of the graph-based parsing. Thus, many researches focus on integrating neural network structures into it and exploiting word embedding. Chen and Manning [12] introduced the neural network classifier and continuous representations into a greedy transition-based dependency parser, in order to improve the accuracy. Dyer et al. [11] employed stack long short-term memory recurrent neural networks in a transition-based parser and yielded another state-of-the-art parsing accuracy. Word embedding (distributed word representation) is a widely used method to represent words as lower-dimensional real vectors and is beneficial to the parsers. Usually, it is learned by unsupervised learning approaches. This method is developed based on the distributional hypothesis, thus containing semantic and syntactic information.

Most of the parsers will omit the feature strings, denoted as un-indexed feature strings, of which occurrence frequency is lower than a threshold. The lower frequency is partly due to some nonfrequent words, but they may carry some useful information because some synonyms of the nonfrequent words

are frequent words. Therefore, to utilize un-indexed feature strings and alleviate the sparse problem, we propose a transform-based dependency parsing based on the graph-based parsing algorithm. Our work changes nothing in the traditional feature-based parser but adds a transforming stage between the feature extracting stage and the learning or decoding stage. In the stage, a sparse feature weight matrix is employed to dynamically transform original features into transformed features. Thus, the parser directly utilizes word embeddings so as to form a feature weight matrix to transform features to consider the lower frequency features, which shows another implementation to incorporate embeddings effectively. Furthermore, we also propose a fusion parser that combines a transform-based parser and a feature-based parser. The experimental results show that our proposed parsers are more accurate compared to the corresponding feature-based parsers and indicate that un-indexed feature strings are beneficial to the parser.

## 2. Related Works

Le and Zuidem [13] proposed an infinite-order generative dependency model by using an Inside-Outside Recursive Neural Network, which allowed information to flow both bottom-up and top-down. Reranking with this model showed competitive improvement in parsing accuracy. Zhu et al. [14] introduced a recursive convolutional neural network model to capture syntactic and compositional-semantic representations. By using the model to rank dependency trees in a list of  $k$ -best candidates, their parser achieved very competitive parsing accuracy. Chen and Manning [12] introduced a neural network classifier for a greedy, transition-based dependency parser, which used a small number of dense features by exploiting continuous representations, in order to improve the accuracy. Bansal et al. [10] extracted bucket features by creating an indicator feature per dimension of the word vector with the discontinuity bucketing function, clustered bit string features like Brown Clustering in addition to traditional features, which improved the accuracy. Chen et al. [9] proposed a method to learn feature embeddings on a large dataset automatically which is parsed with a pre-trained parser to improve the parsing accuracy with feature embeddings and traditional features. By using their FE-based feature templates, they generated the embedding features to improve the parsing accuracy. The features using in parsing are selected by their occurring numbers, such as features occurring more than once used in Bansal et al. [10], to alleviate the sparse problem and noise influence. However, the omitted features do contain useful information because their infrequency may be caused by rare words while some of them have frequent synonyms.

## 3. Graph-Based Dependency Parsing

A dependency tree describes the relationships among the words in a sentence. All nodes in the tree are words. Figure 1 shows an example of a dependency tree, where an arc indicates a dependency relation from a head to a modifier. "ROOT" is an artificial root token. Table 1 describes the main symbols used here. In this section, we first introduce basic dependency parsing. Then, the transform-based dependency parsing is presented. We employ a feature-based parser with the graph-based parsing algorithm as our underlying parser. Given a segmented sentence  $x$  and its dependency tree  $y$ , the score of  $y$  for  $x$  is defined as follows:

$$s(x, y, \omega) = \omega \cdot f(x, y), \quad (1)$$

where  $\omega \in R^N$  is a weight vector for the features learned in a training process;  $f(x, y)$  is a feature vector (given data  $x$  and  $y$ , we abbreviate it as  $f$  for simplification), in which features refer to indexed feature strings extracted from a sentence  $x$  and  $y$ . After the features are extracted, the Margin Infused Relaxed Algorithm (MIRA) [15,16] is adopted to learn the weight vector  $\omega$  by the following way:

$$\begin{aligned}
 & \text{Minimize } \|\omega' - \omega\|, \\
 & \text{s.t. } s(x, y, \omega') - s(x, \bar{y}, \omega') \geq L(y, \bar{y}), \\
 & \forall (x, y) \in \mathbb{T}, \bar{y} \in ds(x),
 \end{aligned} \tag{2}$$

where  $\omega'$  is an optimized weight vector;  $\mathbb{T}$  is the training treebank;  $ds(x)$  is a set of all feasible dependency trees of  $x$ , and  $L(y, \bar{y})$  is the number of words with the incorrect parent in predicted tree  $\bar{y}$ . The condition in Equation (2) states that the parser with optimized  $\omega'$  separates  $y$  from any other tree  $\bar{y} \in ds(x)$  with a distance no less than  $L(y, \bar{y})$ , and the minimum cost function results in preventing the norm of the weight vector from “blow-up”. As every feature is corresponding to a feature string indexed in a lookup map, we will not distinguish them unless the feature strings are un-indexed, which are numerous because of the huge incorrect dependent tree or feature clipping.

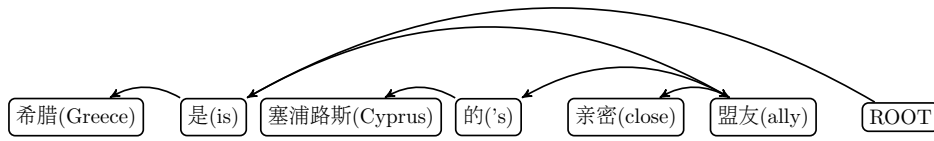


Figure 1. An example of dependency tree.

Table 1. List of symbols used in this paper.

|                    |  |                      |  |
|--------------------|--|----------------------|--|
| $x$                | : segmented sentence                                   | $y$                  | : dependency tree for a sentence   |
| $\mathbb{T}$       | : training treebank                                    | $ x $                | : number of words in $x$   |
| $w_i$              | : the $i$ th word of a sentence $x$                    | $T$                  | : feature template   |
| $C$                | : feature string                                       | $C_i$                | : the $i$ th indexed feature string  |
| $tp(C_i)$          | : index of the feature template generating $C_i$       | $C_{\mathbb{T}}$     | : set of indexed feature strings extracted from $T$ , $\{C_i\}_{i \in [1, N]}$ |
| $ds(x)$            | : set of all feasible dependency trees of $x$          | $L(y, \bar{y})$      | : loss function for $\bar{y}$ from $y$   |
| $\tilde{f}(\cdot)$ | : transformed feature vector function                  | $\tilde{f}_i(\cdot)$ | : transformed feature indicating function corresponding to $C_i$               |
| $f(\cdot)$         | : feature vector function                              | $f_i(\cdot)$         | : feature indicating function corresponding to $C_i$                           |
| $N$                | : number of indexed feature strings                    | $U$                  | : number of total feature strings.   |
| $c$                | : window size for word embedding                       | $\eta$               | : hyper-parameter for fusion decoding  |
| $Q$                | : number of nearest transformed features for a feature | $\omega$             | : weight vector for features   |
| $\tilde{\omega}$   | : weight vector for transformed features               | $\Theta$             | : feature weight matrix  |
| $\theta_i$         | : $i$ th row of $\Theta$                               | $\sigma_w(\cdot)$    | : similarity between two words via embeddings                                  |
| $\sigma(\cdot)$    | : similarity between two feature strings               |                      |  |

### 3.1. Feature Similarity Information

The vector  $f$  is generated from the parts, which are factored from the dependency trees, and the  $i$ th element  $f_i$  is the occurrence number of the corresponding indexed feature string  $C_i$  extracted from the parts by applying the  $tp(C_i)$ th template. The feature template  $T$  draws information from a part, and for the second-order part, the templates normally consist of some atomic templates, namely,  $T_w^{h+j}$ ,  $T_p^{h+j}$ ,  $T_w^{c+j}$ ,  $T_p^{c+j}$ ,  $T_w^{d+j}$ ,  $T_p^{d+j}$ ,  $T_{dhdc}$  and  $T_{dhd}$ , where the superscript  $h + j$  or  $d + j$  represents the position that is the  $j$ th word after the parent or child, and the subscript  $w$ ,  $p$ ,  $dhdc$  and  $dhd$  denote the types, namely, the surface word, the part-of-speech (POS) tag, the direction between  $h$  and  $d$  and the direction in  $h$ ,  $d$  and  $c$ , respectively. Thus, the following equation depicts the  $i$ th feature template:

$$T_i = \{ \cup_{itm \in \{w, p\}} \cup_{pt \in \{h, d, c\}} \cup_{k=-n_T(i, pt, itm)}^{n_T(i, pt, itm)} T_{itm}^{pt+k} \} \cup T_{dhdc}^{n_T(i, dhdc)} \cup T_{dhd}^{n_T(i, dhd)}, \tag{3}$$

where function  $n_T(i, pt, itm)$  returns the number of the atomic templates for the position  $pt \in \{h, d, c\}$  and the type  $itm \in \{w, p\}$  in the  $i$ th feature template, and the binary operator  $\cup$  concatenates the two input strings. Since  $T_w^h$  catches a word in a part while it draws its synonym in another part, feature strings extracted from the same feature template may be similar. Therefore, there are many related feature strings, each of which contain information about other feature strings.

Bansal et al. [10] proposed a new bucket feature template  $B_k$  to identify the equivalence of two embedding at the  $k$ th dimension. Therefore, this will introduce a similarity  $\sigma_k(C_i, C_j)$  between two feature strings  $C_i$  and  $C_j$  extracted with the same feature template:

$$\begin{aligned} \sigma_k(C_i, C_j) = & \prod_{pt \in \{h, d, c\}} \left\{ \prod_{k=-n_T(i, pt, p)}^{n_T(i, pt, p)} \delta(T_p^{pt+k}(C_i), T_p^{pt+k}(C_j)) \cdot \right. \\ & \left. \prod_{k=-n_T(i, pt, w)}^{n_T(i, pt, p)} \delta(B_k(T_w^{pt+k}(C_i)), B_k(T_w^{pt+k}(C_j))) \right\} \cdot \\ & \delta(T_{dhdc}^{n_T(i, \cdot, dhdc)}(C_i), T_{dhdc}^{n_T(i, \cdot, dhdc)}(C_j)) \cdot \\ & \delta(T_{dhd}^{n_T(i, \cdot, dhd)}(C_i), T_{dhd}^{n_T(i, \cdot, dhd)}(C_j)), \end{aligned} \quad (4)$$

where  $T_*(C)$  returns the atomic string extracted with the atomic template  $T_*$  in the feature string  $C$  or returns an empty string if the used template does not contain  $T_*$ ;  $B_k(w)$  returns the value of the embedding of  $w$  at the  $k$ th dimension. Chen et al. [9] utilizes feature embeddings instead of word embeddings, and introduces a new feature embedding template that consists of the index of the drawing dimension, the feature template extracting that feature string, and the drawing function  $B_k$ . Thus, this also defines a similarity  $\sigma_k(C_i, C_j)$  between two feature strings  $C_i$  and  $C_j$ ,

$$\sigma_k(C_i, C_j) = \delta(tp(C_i), tp(C_j)) \cdot \delta(B_k(C_i), B_k(C_j)), \quad (5)$$

where  $B_k(C)$  draws the value of the embedding of  $C$  at the  $k$ th dimension.

However, there are many omitted (un-indexed) feature strings due to occurring lower or extracted from the incorrect tree, and they may carry some useful information. Given an un-indexed feature string, its embedding is unknown. Thus, we utilize its inner structure to calculate the similarity from the indexed feature. Similar to the work of Bansal et al. [10], we exploit another method, directly computing the similarity via word embeddings as follows:

$$\begin{aligned} \sigma(C_i, C_j) = & \prod_{pt \in \{h, d, c\}} \left\{ \prod_{k=-n_T(i, pt, p)}^{n_T(i, pt, p)} \delta(T_p^{pt+k}(C_i), T_p^{pt+k}(C_j)) \cdot \right. \\ & \left. \prod_{k=-n_T(i, pt, w)}^{n_T(i, pt, p)} \sigma_w(T_w^{pt+k}(C_i), T_w^{pt+k}(C_j)) \right\} \cdot \\ & \delta(T_{dhdc}^{n_T(i, \cdot, dhdc)}(C_i), T_{dhdc}^{n_T(i, \cdot, dhdc)}(C_j)) \cdot \\ & \delta(T_{dhd}^{n_T(i, \cdot, dhd)}(C_i), T_{dhd}^{n_T(i, \cdot, dhd)}(C_j)), \end{aligned} \quad (6)$$

where  $\sigma_w$  calculates the similarity between two words via embeddings.

### 3.2. Transform-Based Parsing

Given a training and testing data, we first assume that there are  $N$  indexed feature strings and  $U$  total feature strings. Together with Equation (6), a feature matrix  $\Theta$  is obtained by the following equation to measure the similarity between the feature strings

$$\Theta = [\sigma(C_i, C_j)]_{i=1}^N \underset{j=1}{U}. \quad (7)$$

Thus, a transformed feature  $\tilde{f}_i$  is represented as a row  $\theta_i$  of the feature weight matrix  $\Theta$ . Like  $f_i$ ,  $\tilde{f}_i$  corresponds to the indexed feature string  $C_i$ . Similar to the score Equation (1), the following equation depicts the transform-based score of a dependency tree

$$s'(x, y) = \tilde{\omega} \cdot \tilde{f}(x, y), \quad (8)$$

where  $\tilde{\omega}$  is a weight vector for transformed features, and  $\tilde{f}$  is a transformed feature vector extracted from  $x$  and  $y$ . The transformed features are generated by  $\Theta \cdot \hat{f}(x, y)$ , where  $\hat{f}(x, y)$  is the extending extractor of  $f(x, y)$  and returns the un-indexed feature strings as well. After substituting Equation (9) into Equation (2), it can be re-written in the following form

$$\begin{aligned} & \text{Minimize } \|\tilde{\omega}' - \tilde{\omega}\|, \\ & \text{s.t. } s'(x, y, \tilde{\omega}') - s'(x, \bar{y}, \tilde{\omega}') \geq L(y, \bar{y}), \\ & \forall (x, y) \in \mathbb{T}, \bar{y} \in ds(x), \end{aligned} \tag{9}$$

where  $\tilde{\omega}'$  is an optimized weight vector for transformed features.

### 3.3. Optimizing Influence

Because of the similarity of some feature strings, the rank  $K$  of  $\Theta$  may be less than  $N$  and  $U$ , and  $N - K$  rows of  $\Theta$  can be represented by the other rows. We can find  $K$  linearly independent rows, and re-array them in the first  $K$  rows. Then, the  $i$ th ( $i > K$ ) row  $\theta_i$  of  $\Theta$ , the representation of the  $i$ th transformed feature representation, can be constructed as follows:

$$\begin{aligned} \theta_i = & \sum_{j=1}^K \alpha_{i,j} \cdot \theta_j \quad \exists \{\alpha_{i,j}\}_{j=1..K} \in R, \\ & i = K + 1, \dots, N, \end{aligned} \tag{10}$$

where  $\alpha_{i,j}$  is the coefficient of basis  $\theta_j$ . Thus,  $N - K$  transformed features are redundant, but it can be proven that it does not influence the optimization procedure of MIRA. For this purpose, Problem (9) can be rewritten as Equation (11)

$$\begin{aligned} & \text{Minimize } \|\dot{\omega}\|, \\ & \text{s.t. } (\tilde{f}(x, y) - \tilde{f}(x, k))^T \cdot \dot{\omega} \geq \\ & L(y, k) - s'(x, y, \tilde{\omega}) + s'(x, k, \tilde{\omega}) \quad k \in ds(x), \end{aligned} \tag{11}$$

where  $\dot{\omega} = \tilde{\omega}' - \tilde{\omega}$ . The dual problem of Problem (11) can be written as follows:

$$\begin{aligned} & \text{Maximize } \frac{1}{2} \lambda^T \cdot D \cdot \lambda + e^T \cdot \lambda, \\ & \text{subject to } \lambda \geq 0, \end{aligned} \tag{12}$$

where  $D$  and  $e$  are as follows:

$$\begin{cases} e = \left[ (L(y, t) - s'(x, y, \tilde{\omega}) + s'(x, t, \tilde{\omega}))^T \right]_{t \in ds(x)}, \\ D = -A \cdot A^T, \\ A = \left[ (\tilde{f}(x, y) - \tilde{f}(x, t))^T \right]_{t \in ds(x)}. \end{cases} \tag{13}$$

Since there are  $N - K$  dependent transformed features, we firstly drop them to explore the change of Problem (12). Let  $D^K, e^K$  be the dropped version of  $D, e, D = H^T \cdot D^K \cdot H$  since the rank  $rank(D^K) = rank(D)$ , where  $H$  is a full-rank matrix. Moreover, it can be proved that Problem (12) can be rewritten in the following form if the feasible region is  $H \cdot \lambda \geq 0$ ,

$$\begin{aligned} & \text{Maximize } \frac{1}{2} \lambda^T \cdot D^K \cdot \lambda + e^{K^T} \cdot H^{-1} \cdot \lambda, \\ & \text{subject to } \lambda \geq 0. \end{aligned} \tag{14}$$

Therefore, redundant transformed features will introduce an additional transforming matrix  $H^{-1}$  transforming  $e$  and rotating the feasible region. However, the dropped version (14) and original version (12) can be calculated with quadratic programming in the same way, such as Hildreth's algorithm [17]. After  $\lambda$  is calculated,  $\dot{\omega}$  can be computed directly by  $A^T \cdot \lambda$ . Therefore, redundant

transformed features do not affect the optimization but would weight the constraints due to  $e^{k^T} \cdot H^{-1}$ , which may hurt the performance.

The problem is alleviated by only considering the  $Q$  nearest transformed features for a feature as the approach results in a very sparse matrix  $\Theta$ . When  $Q$  is larger, transform-based features will catch more information of feature similarity. Hence, when the metrics of similarity  $\sigma(C_i, C_j)$  are reliable,  $Q$  should be assigned with a relatively large value to fully exploit the similarity between features.

### 3.4. Fusion Decoding

Fusion is proved to be helpful in classification tasks [18,19]. Considering that original and transformed features describe different views of a span, we attempt to fuse them together for possible improvement. Like some reranking frameworks such as the forest reranking algorithm [20] without needing the K-best output, we employ a fusion decoding to integrate original and transformed features. The scoring function of this fusion decoding combines transform-based and feature-based scoring functions as follows:

$$\hat{s}(x, y) = ((1 - \eta) \cdot \omega + \eta \cdot \tilde{\omega} \cdot \Theta) \cdot f(x, y), \quad (15)$$

where  $\eta$  is a hyper-parameter for fusion decoding.

## 4. Implementation of Transform-Based Parsing

To verify the effectiveness of the transformed features and compare with the base features in a graph-based parsing algorithm, we adopt a projective parser learned with MIRA to build the transform-based parser. Without loss of generality, we only consider first-order features over dependency parts [6] and second-order features over grandchild and sibling parts [21,22]. The training process of the transform-based parser can be divided into several stages as follows.

1. *Building Feature Lookup Map*: An indexed set  $\{C_i\}_{i \in [1, |C_T|]}$  of feature strings is generated by enumerating the feature strings, where occurrence frequency is more than five, in a training corpus and assigning their indexes in occurring order. Then, they form a lookup map to identify the indexed feature string.
2. *Caching Sentence Transformed Features*: The feature strings of every sentence in the training corpus and its possible dependency trees are extracted. Then, for each feature string  $C_*$  (indexed or un-indexed), the column  $\theta_{C_*}$  of  $\theta$  is constructed and cached, where the similarity  $\sigma_w$  between words  $w_1$  and  $w_2$  is defined as the dot product of their embeddings.
3. *Training*: The parameters are learned with MIRA, and it is known that the graph-based dependency parsing is time-consuming.

In the stage of Caching Sentence Transformed Features, a feature string  $C$  may occur many times and is non-indexed. Moreover, the amount of the indexed features strings is usually several million. Thus, the vector  $\theta_C$  should be dynamically calculated with Equation (6) and cached to accelerate the extraction.

## 5. Experiments

In this paper, the transform-based parser is compared with the feature-based parser on the English Penn Treebank (PTB) and Chinese Treebank (CTB) version 5.0 [23] with gold-standard segmentation and POS tags. As the feature transform stage can be integrated into any order parsers, without loss of generality, we only conduct the experiments on second order parsers.

### 5.1. Experiments Setup

An open-source conversion utility Penn2Malt with head rules compiled by Zhang and Clark [24] is employed to generate CoNLL dependencies on CTB data. The same split of the CTB5 as defined



by Zhang and Clark [24] is adopted except that the training corpus is *smaller* than the original due to the time and storage consuming of the training. In order to reserve the statistical characteristics of the original training corpus, we construct the training corpus by selecting *every five sentences* in the original training corpus, and the split is as Table 2. For English, we use the head rules provided by Yamada and Matsumoto for the converting, and the POS tags are predicted by the Stanford POS tagger (*accuracy*  $\approx$  97.2%).

**Table 2.** The data split for training, testing and development.

|     | Training                          | Testing            | Development        |
|-----|-----------------------------------|--------------------|--------------------|
| PTB | One-Fifth of {2–21}               | 22                 | 23                 |
| CTB | One-Fifth of {001–815, 1001–1136} | 816–885, 1137–1147 | 886–931, 1148–1151 |

There are several popular models for efficiently and effectively learning word vectors, such as Global Vectors for word representation (Glove) [25], skip-gram with negative-sample (SGNS) in Word2Vec [26,27] and syntax relative skip-gram with negative-sample (SSGNS) [28].

According to experiments reported by Suzuki et al. [29], GloVe and SGNS provide similar performance in many tasks. We only consider two models-SGNS and SSGNS in this paper. The training dataset is the Daily Xinhua News Agency part of the Chinese Gigaword Fifth Edition (LDC2011T13), and it is segmented by a Stanford Word Segmenter [30], which is trained with the training dataset in Table 2. Since a word may have different POS tags with different meanings, attaching the POS tag of a word to itself may alleviate the vague meaning in word representation learning. Hence, Stanford Postagger [31] is employed to tag the word in the training dataset.

## 5.2. Results and Analysis

We employ a feature-based parser with second order parsing built by Ma and Zhao [32] as the baseline (the parser is a fourth order parser, but we constrain it to second order as all experiments are elevated under the second order). Parsing accuracies are measured by unlabeled attachment score (UAS), the percentage of words with the correct head, and labeled attachment score (LAS), the percentage of words with the correct head and label, which ignores any token with PU tag (punctuation).

### 5.2.1. Parsing Accuracy

At first, we evaluate our parser without considering the un-indexed features, but the accuracies of transform-based parsing are smaller than that of baseline. Henceforth, the experiments are done with the un-indexed features at default. The final scores evaluated on the test corpus are shown in Table 3. In the table, Trans is the same as the transition-based parser with stack long short term memory (LSTM) introduced by Dyer et al. [11], except that it adopts the arc-eager transition system to use dynamic oracles [33] and dropout to prevent over-fitting; OrgEM0 represents the parser with embedding dictionary learned by the SGNS model; OrgEM1 represents the parser with the dictionary based on the SSGNS model; MixOrgEM0 represents the fusion pipelines with OrgEM0. The UAS (LAS) of OrgEM0 are 0.757% (0.804%) higher than that of Baseline, and the scores of OrgEM1 are slightly better than those of OrgEM0. Similarly, the UAS (LAS) of OrgEM0 are 0.579% (0.324%) higher than that of Trans. The fusion parsers MixOrgEM0 and MixOrgEM1 integrate feature information and further improve the parsing accuracy, where UAS and LAS increase as much as 1.175% and 1.284%, respectively. For English, we observe similar experiments to those of Chinese. The UAS (LAS) of OrgEM0 are 0.617% (0.545%) higher than that of Baseline. The fusion versions improve the accuracy as well. Therefore, the transform-based parser improves parsing accuracy, and the SSGNS model performs better than the SGNS model in transform-based parsing because of the consideration of the word order.

Table 3. Results of the test corpus.

|     |     | Trans  | Baseline | OrgEM0  | MixOrgEM0 | OrgEM1  | MixOrgEM1 |
|-----|-----|--------|----------|---------|-----------|---------|-----------|
| CTB | UAS | 84.13% | 83.952%  | 84.709% | 85.127%   | 84.730% | 84.948%   |
|     | LAS | 82.25% | 81.770%  | 82.574% | 83.054%   | 82.602% | 82.905%   |
| PTB | UAS | 91.12% | 90.982%  | 91.599% | 92.017%   | 91.52%  | 91.828%   |
|     | LAS | 89.87% | 89.59%   | 90.135% | 90.603%   | 90.122% | 90.325%   |

5.2.2. Parameter Selection

The parameters selection is conducted on CTB, and we use the same parameters in the experiments of English. The two hyper-parameters ( $Q, win$ ) are determined by finding the maximum score on the development corpus.  $\eta$  is searched in every experience in the fusion pipeline because of the low cost of enumerating it. The experiments based on embedding dictionaries with SGNS model trained by the famous tool-kits Word2Vec. Firstly, for the hyper-parameter  $win$ , its searching range is [1, 10]. The parsing accuracy of the development corpus are compared in Figure 2. It is shown that the transform-based parser can improve the parsing accuracy with the embedding dictionaries for different window sizes. The best performance is achieved at  $win = 3$ , and the UAS and LAS are 1.08% and 0.95% higher than those of the baseline, respectively. Figure 3 indicates that the fusion pipeline can integrate the original and transformed features to improve the parsing performance further. The fusion parser is a pure feature based parser (baseline) at  $\eta = 0$ , while the parser is a pure transform based parser at  $\eta = 1$ . The best performance of the fusion pipeline is achieved at ( $win = 1, \eta = 0.8$ ), and UAS and LAS are 1.16% and 1.14% higher than the scores of the baseline, respectively. Therefore, the window size is chosen as  $win = 1$  in the following experiences.

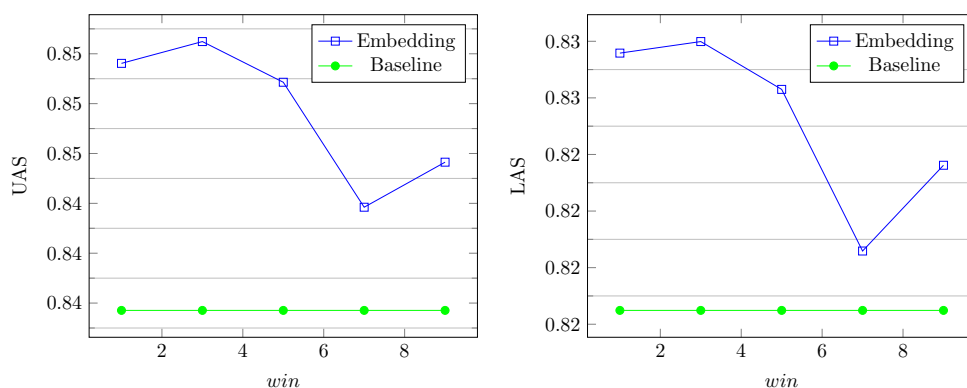


Figure 2. The parsing accuracy under different win.

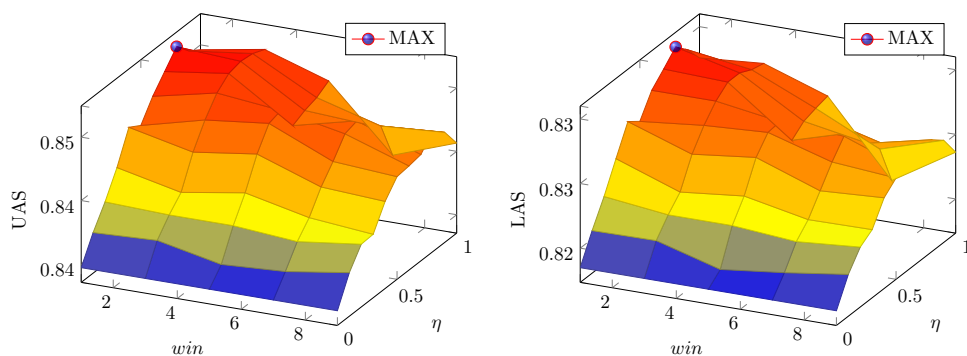


Figure 3. The fusion parsing accuracy under different win and  $\eta$ .



Secondly, for the hyper-parameter  $Q$ , its searching range is  $[5, 100]$  due to the limit in computing resources. The experiments are shown in Figure 4. The larger the  $Q$ , the more relevant transformed features are retrieved for a feature, and the larger the size of transformed feature caches of sentences. However, the score is decreasing with the increasing of  $Q$ , and it reaches the maximum at  $Q = 10$ . This indicates that a larger  $Q$  may result in more noise in transformed feature caches of sentences. Thus, it is partially caused by noise in word representation. Figure 5 shows the scores of the fusion pipeline. The biggest UAS improvement is achieved at  $(Q = 40 \ \& \ \eta = 0.6)$ . Thus, we select the hyper-parameters as  $(win = 1 \ \& \ Q = 40)$  in the following experiences.

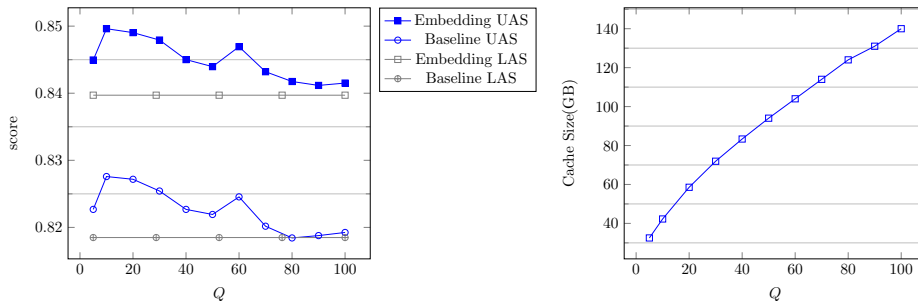


Figure 4. The parsing accuracy and cache size under different  $Q$ .

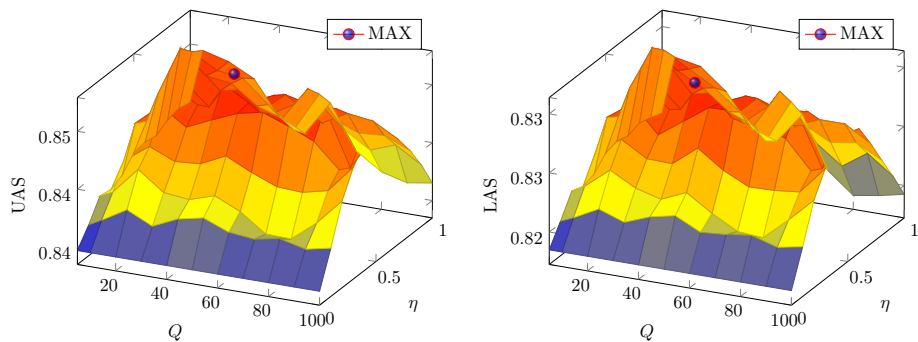


Figure 5. The fusion parsing accuracy under different  $Q$  and  $\eta$ .

In addition, for the parser with the SSGNS model, UAS and LAS are 85.07% and 82.7%, respectively. They are higher than the corresponding scores in the parser with the SGNS model. The dictionary learned by the SSGNS model considers the word order and may contain more syntactic information. Therefore, the parser would be beneficial from the dictionary with more syntactical information. Figure 6 shows the scores of the fusion pipeline are higher than those with the SGNS model.

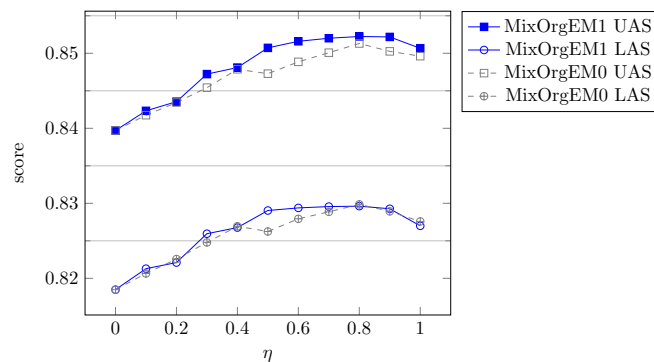


Figure 6. The fusion parsing accuracy with the SSGNS model under different  $\eta$ .

## 6. Conclusions

In this paper, we propose a transform-based parser that utilizes transformed features instead of original features to exploit all feature strings. The transformed features are generated by using a feature weight matrix. It is proved that the redundant weights of transformed features, which are caused by rank deficiency of the feature weight matrix, does not affect the learning of MIRA. The result shows that the scores of the transform-based parsers with un-indexed feature strings are higher than those of the corresponding feature-based parser. Because original and transformed features play the same role in the learning algorithm, the fusion parser can be simply constructed by integrating the information of original and transformed features. The results show that the fusion parser outperforms the transform-based parser in terms of parsing accuracy.

Although our parser is a second-order parser, it can be adopted into fourth-order or even higher order parsers and some reranking frameworks to improve the parsing accuracy. With word embedding, the improvement of generalization is on the surface word level, and more investigations will be conducted for improvement of generalization in the level of the inner syntactical structure. In the future, we plan to customize the similarity function between feature strings to utilize the semantic and syntactic information indicated by further embeddings.

**Conflicts of Interest:** The author declares no conflict of interest.

## References

1. Cho, H.C.; Okazaki, N.; Miwa, M.; Tsujii, J. Named entity recognition with multiple segment representations. *Inf. Process. Manag.* **2013**, *49*, 954–965.
2. Sun, X.; Zhang, Y.; Matsuzaki, T.; Tsuruoka, Y.; Tsujii, J. Probabilistic Chinese word segmentation with non-local information and stochastic training. *Inf. Process. Manag.* **2013**, *49*, 626–636.
3. Jiang, Y.; Zhang, X.; Tang, Y.; Nie, R. Feature-based approaches to semantic similarity assessment of concepts using Wikipedia. *Inf. Process. Manag.* **2015**, *51*, 215–234.
4. Zhang, Y.; Clark, S. Syntactic Processing Using the Generalized Perceptron and Beam Search. *Comput. Linguist.* **2011**, *37*, 105–151.
5. Chen, W.; Kazama, J.; Uchimoto, K.; Torisawa, K. Exploiting Subtrees in Auto-Parsed Data to Improve Dependency Parsing. *Comput. Intell.* **2012**, *28*, 426–451.
6. McDonald, R.; Crammer, K.; Pereira, F. Online Large-Margin Training of Dependency Parsers. In Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL '05), Ann Arbor, MI, USA, 25–30 June 2005; pp. 91–98.
7. McDonald, R.; Pereira, F.; Ribarov, K.; Hajič, J. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT '05), Vancouver, BC, Canada, 6–8 October 2005; pp. 523–530.
8. Wu, F.; Zhou, F. Hybrid dependency parser with segmented treebanks and reparsing. In *Chinese Intelligent Automation Conference*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 336, pp. 53–60.
9. Chen, W.; Zhang, M.; Zhang, Y. Distributed Feature Representations for Dependency Parsing. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2015**, *23*, 451–460.
10. Bansal, M.; Gimpel, K.; Livescu, K. Tailoring Continuous Word Representations for Dependency Parsing. Available online: [http://ttic.uchicago.edu/~klivescu/papers/bansal\\_etal\\_ACL2014.pdf](http://ttic.uchicago.edu/~klivescu/papers/bansal_etal_ACL2014.pdf) (accessed on 18 January 2017).
11. Dyer, C.; Ballesteros, M.; Ling, W.; Matthews, A.; Smith, N.A. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. *arXiv* **2015**, arXiv:1505.08075.
12. Chen, D.; Manning, C.D. A Fast and Accurate Dependency Parser Using Neural Networks. Available online: <http://cs.stanford.edu/people/danqi/papers/emnlp2014.pdf> (accessed on 18 January 2017).
13. Le, P.; Zuidema, W. The Inside-Outside Recursive Neural Network model for Dependency Parsing. Available online: <http://www.emnlp2014.org/papers/pdf/EMNLP2014081.pdf> (accessed on 18 January 2017).
14. Zhu, C.; Qiu, X.; Chen, X.; Huang, X. A Re-Ranking Model for Dependency Parser with Recursive Convolutional Neural Network. *arXiv* **2015**, arXiv:1505.05667.

15. Crammer, K.; Dekel, O.; Keshet, J.; Shalev-Shwartz, S.; Singer, Y. Online passive-aggressive algorithms. *J. Mach. Learn. Res.* **2006**, *7*, 551–585.
16. Crammer, K.; Singer, Y. Ultraconservative online algorithms for multiclass problems. *J. Mach. Learn. Res.* **2003**, *3*, 951–991.
17. Censor, Y.; Zenios, S.A. *Parallel Optimization: Theory, Algorithms, and Applications*; Oxford University Press: Oxford, UK, 1997.
18. Zhang, Z.; Huang, D.Y.; Zhao, R.; Dong, M. Onset Detection Based on Fusion of Simpls and Superflux. Available online: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.445.6390&rep=rep1&type=pdf> (accessed on 18 January 2017).
19. Huang, D.Y.; Zhang, Z.; Ge, S.S. Speaker state classification based on fusion of asymmetric simple partial least squares (SIMPLS) and support vector machines. *Comput. Speech Lang.* **2014**, *28*, 392–419.
20. Hayashi, K.; Watanabe, T.; Asahara, M.; Matsumoto, Y. Third-Order Variational Reranking on Packed-Shared Dependency Forests. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11), Edinburgh, UK, 27–31 July 2011; pp. 1479–1488.
21. McDonald, R.; Pereira, F. Online Learning of Approximate Dependency Parsing Algorithms. Available online: [http://www.aclweb.org/old\\_anthology/E/E06/E06-1011.pdf](http://www.aclweb.org/old_anthology/E/E06/E06-1011.pdf) (accessed on 18 January 2017).
22. Carreras, X. Experiments with a Higher-Order Projective Dependency Parser. Available online: [http://www.aclweb.org/old\\_anthology/D/D07/D07-1.pdf#page=991](http://www.aclweb.org/old_anthology/D/D07/D07-1.pdf#page=991) (accessed on 18 January 2017).
23. Xue, N.; Xia, F.; Chiou, F.D.; Palmer, M. The Penn Chinese TreeBank: Phrase structure annotation of a large corpus. *Nat. Lang. Eng.* **2005**, *11*, 207–238.
24. Zhang, Y.; Clark, S. A Tale of Two Parsers: Investigating and Combining Graph-Based and Transition-Based Dependency Parsing Using Beam-Search. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '08), Honolulu, HI, USA, 25–27 October 2008; pp. 562–571.
25. Pennington, J.; Socher, R.; Manning, C.D. Glove: Global Vectors for Word Representation. In Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014), Doha, Qatar, 25–29 October 2014; pp. 1532–1543.
26. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. *arXiv* **2013**, arXiv:1301.3781.
27. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the Neural Information Processing Systems Foundation, Stateline, NV, USA, 5–10 December 2013; pp. 3111–3119.
28. Ling, W.; Dyer, C.; Black, A.W.; Trancoso, I. Two/Too Simple Adaptations of Word2Vec for Syntax Problems. Available online: <http://www.aclweb.org/anthology/N/N15/N15-1142.pdf> (accessed on 18 January 2017).
29. Suzuki, J.; Nagata, M. A Unified Learning Framework of Skip-Grams and Global Vectors. Available online: <http://www.aclweb.org/anthology/P/P15/P15-2031.pdf> (accessed on 18 January 2017).
30. Tseng, H.; Chang, P.; Andrew, G.; Jurafsky, D.; Manning, C. A conditional random field word segmenter for sighthan bakeoff 2005. In Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing, Jeju Island, Korea, 14–15 October 2005.
31. Toutanova, K.; Klein, D.; Manning, C.D.; Singer, Y. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL '03), Edmonton, AB, Canada, 27 May–1 June 2003; Volume 1, pp. 173–180.
32. Ma, X.; Zhao, H. Fourth-Order Dependency Parsing. Available online: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.378.9297&rep=rep1&type=pdf#page=799> (accessed on 18 January 2017).
33. Goldberg, Y.; Nivre, J. A Dynamic Oracle for Arc-Eager Dependency Parsing. Available online: <http://www.cs.bgu.ac.il/yoavg/publications/coling2012dynamic.pdf> (accessed on 18 January 2017).

