Transition-Based Disfluency Detection using LSTMs

Shaolei Wang[1], **Wanxiang Che**[*1], **Yue Zhang**[2], **Meishan Zhang**[3] and **Ting Liu**[1]
[1]Center for Social Computing and Information Retrieval, Harbin Institute of Technology, China
[2]Singapore University of Technology and Design
[3]School of Computer Science and Technology, Heilongjiang University, China
{slwang, car, tliu}@ir.hit.edu.cn
yue_zhang@sutd.edu.sg, mason.zms@gmail.com

Abstract

We model the problem of disfluency detection using a transition-based framework, which incrementally constructs and labels the disfluency chunk of input sentences using a set of transition actions without syntax information. Compared with sequence labeling methods, it can capture non-local chunk-level features; compared with joint parsing and disfluency detection methods, it is free for noise in syntax. Experiments show that our model achieves state-of-the-art F-score on both the commonly used English Switchboard test set and a set of in-house annotated Chinese data.

1 Introduction

Disfluency detection is the task of recognizing non-fluent word sequences in spoken language transcripts (Zayats et al., 2016; Wang et al., 2016; Wu et al., 2015). As shown in Figure 1, standard annotation of disfluency structure (Shriberg, 1994) indicates the reparandum (words that are discarded, or corrected by the following words), the interruption point (+) marking the end of the reparandum, the associated repair, and an optional interregnum after the interruption point (filled pauses, discourse cue words, etc.).

Ignoring the interregnum, disfluencies can be categorized into three types: restarts, repetitions, and corrections, based on whether the repair is empty, the same as the reparandum or different, respectively. Table 1 gives a few examples. Interregnums are easy to detect as they often consist of fixed phrases (e.g. "uh", "you know"). However, reparandums are more difficult to detect, because they can be in arbitrary form. Most previous disfluency detection work focuses on detecting reparandums.

I want a flight [$\underbrace{\textit{to Boston}}_{\text{RM}}$ + $\underbrace{\{\textit{um}\}}_{\text{IM}}$ $\underbrace{\text{to Denver}}_{\text{RP}}$]

Figure 1: Sentence with disfluencies annotated in English Switchboard corpus. RM=Reparandum, IM=Interregnum, RP=Repair. The preceding RM is corrected by the following RP.

Type	Annotation
repair	[I just + I] enjoy working
repair	[we want + {well} in our area we want] to
repetition	[it's + {uh} it's] almost like
restart	[we would like +] let's go to the

Table 1: Different types of disfluencies.

The main challenges of detecting reparandums include that they vary in length, may occur in different locations, and are sometimes nested. For example, the longest reparandum in our training set has fifteen words. Hence, it is very important to capture long-range dependencies for disfluency detection. Since there is large parallelism between the reparandum chunk and the following repair chunk (for example, in Figure 1, the reparandum begins with *to* and ends before another occurrence of *to*), it is also useful to exploit chunk-level representation, which explicitly makes use of resulted infelicity disfluency chunks.

Common approaches take disfluency detection as a sequence labeling problem, where each sentential word is assigned with a label (Zayats et al., 2016; Hough and Schlangen, 2015; Qian and Liu, 2013; Georgila, 2009). These methods achieve good performance, but are not powerful enough to capture complicated disfluencies with longer spans or distances. Another drawback of these approaches is that they are unable to exploit chunk-

*Email corresponding.

Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 2785–2794
Copenhagen, Denmark, September 7–11, 2017.

level features. Semi-CRF (Ferguson et al., 2015) is used to alleviate this issue to some extent. Semi-CRF models still have their inefficiencies because they can only use the local chunk information limited by the markov assumption when decoding.

A different line of work (Rasooli and Tetreault, 2013; Honnibal and Johnson, 2014; Wu et al., 2015) adopts transition-based parsing models for disfluency detection. This line of work can be seen as a joint of disfluency detection and parsing. The main advantage of the joint models is that they can capture long-range dependency of disfluencies as well as chunk-level information. However, they introduce additional annotated syntactic structure, which is very expensive to produce, and can cause noise by significantly enlarging the output search space.

Inspired by the above observations, we investigate a transition-based model without syntactic information. Our model incrementally constructs and labels the disfluency chunks of input sentences using an algorithm similar to transition-based depency parsing. As shown in Figure 2, the model state consists of four components: (i) O, a conventional sequential LSTM (Hochreiter and Schmidhuber, 1997) to store the words that have been labeled as fluency. (ii) S, a stack LSTM to represent partial disfluency chunks, which captures chunk-level information. (iii) A, a conventional sequential LSTM to represent history of actions. (iiii) B, a Bi-LSTM to represent words that have not yet been processed. A sequence of transition actions are used to consume input tokens and construct the output from left to right. To reduce error propagation, we use beam-search (Collins and Roark, 2004) and scheduled sampling (Bengio et al., 2015), respectively.

We evaluate our model on the commonly used English Switchboard test set and a in-house annotated Chinese data set. Results show that our model outperforms previous state-of-the-art systems. The code is released[1].

2 Background

For a background, we briefly introduce transition-based parsing and its extention for joint disfluency detection. An arc-eager transition-based parsing system consists of a stack σ containing words being processed, a buffer β containing words to be processed and a memory A storing dependency

[1]https://github.com/hitwsl/transition_disfluency

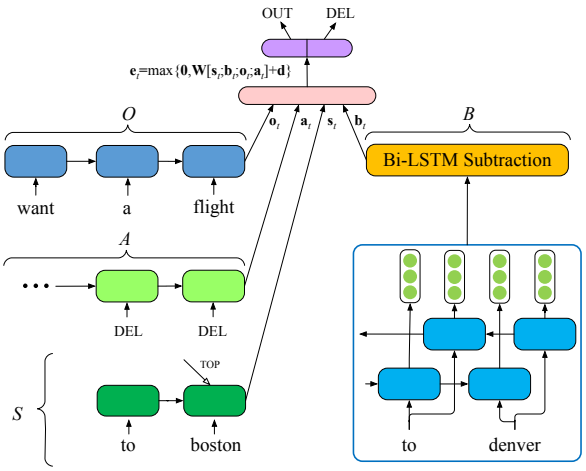

Figure 2: model state when processing the sentence "want a flight to boston to denver".

arcs which have been generated. There are four types of transition actions (Nivre, 2008)

- *Shift* : Remove the front of the buffer and push it to the stack.
- *Reduce* : Pop the top of the stack.
- *LeftArc* : Pop the top of the stack, and link the popped word to the front of the buffer.
- *RightArc* : Link the front of the buffer to the top of the stack, remove the front of the buffer and push it to the stack.

Many neural network parsers have been constructed under this framework, such as (Dyer et al., 2015), who use different LSTM structure to represent information from σ to β.

For disfluency detection, the input is a sentence with disfluencies from automatic speech recognition (ASR). We denote the word sequence as $w_1^n = (w_1, ..., w_n)$. The output of the task is a sequence of binary tags denoted as $D_1^n = (d_1, ..., d_n)$, where each d_i corresponds to the word w_i, indicating whether w_i is a disfluent word or not. Hence the task can be modeled as searching for the best sequenc D^* given the stream of words w_1^n

$$D^* = argmax_D P(D_1^n | w_1^n)$$

Wu et al. (2015) proposes a statistical transition-based disfluency detection model, which performs disfluency detection and parsing jointly by augmenting the Shift-Reduce algorithm with a binary classifier transition (BCT) action:

- *BCT* : Classify whether the current word is disfluent or not. If it is, remove it from the buffer, push it into the stack which is similar to *Shift* and then mark it as disfluent. Otherwise the original parser transition actions will be used.

Disfluency detection and parsing are jointly optimized

$$(D^*, T^*) = argmax_{D,T} \prod_{i=1}^{n} P(d_i|w_1^i, T_1^{i-1}) \times P(T_1^i|w_1^i, T_1^{i-1}, d_i),$$

where T_1^i is the partial tree after word w_i is consumed, d_i is the disfluency tag of w_i. $P(T_1^i|.)$ is the parsing model and $P(d_1^i|.)$ is the disfluency model used to predict the disluency tags on the contexts of partial trees that have been built.

3 Our Transition-Based Model

The BCT model serves as a state-of-the-art transition-based baseline. However, it requires that the training data contains both syntax trees and disfluency annotations, which reduces the practicality of the algorithm. Also, BCT does not explicitly make use of resulting infelicity disfluency chunks. Being a discrete model, the performance relies heavily on manual feature engineering.

To address the constraints above, we apply a transition-based neural model for disfluency detection that does not use any syntax information. Our transition-based method incrementally constructs and labels the disfluency chunk of input sentences by performing a sequence of actions. The task is modeled as

$$(D^*, T^*) = argmax_{D,T} \prod_{i=1}^{n} P(d_i, T_1^i|w_1^i, T_1^{i-1}),$$

where T_1^i is the partial model state after word w_i is consumed. d_i is the disfluency tag of w_i.

3.1 Transition-Based Disfluency Detection

Our model incrementally constructs and labels the disfluency chunks of input sentences, where a state is represented by a tuple (O, S, A, B):

- *output* (O) : the *output* is used to represent the words that have been labeled as fluent.
- *stack* (S) : *stack* is used to represent the partially constructed disfluency chunk.
- *action* (A) : *action* is used to represent the complete history of actions taken by the transition system.
- *buffer* (B) : *buffer* is used to represent the sentences that have not yet been processed.

Given an input disfluent sentence, the *stack*, *output* and *action* are initially empty and the *buffer* contains all words of the sentence, a sequence of transition actions are used to consume words in the *buffer* and build the output sentence:

- OUT: which moves the first word in the *buffer* to the *output* and clears out the *stack* if it is not empty.
- DEL: which moves the first word in the *buffer* to the *stack*.

Search Algorithm

Based on the transition system, the decoder searches for an optimal action sequence for a given sentence. The system is initialized by pushing all the input words and their representations (of § 3.3) onto B in the reverse order, such that the first word is at the top of B, and S, O and A each contains an empty-stack token.

At each step, the system computes a composite representation of the model states (as determined by the current configurations of B, S, O, and A), which is used to predict an action to take. Decoding completes when B is empty (except for the empty-stack symbol), regardless of the state of S. Since each token in B is either moved directly to O or S every step, the total number of actions equals to the length of input sentence. Table 2 shows the sequence of operations required to process the sentence "want a flight to boston to denver".

As shown in Figure 2, the model state representation at time t, which is written as e_t, is defined as:

$$e_t = max\{0, W[s_t; b_t; o_t; a_t] + d\},$$

where W is a learned parameter matrix, s_t is the representation of S, b_t is the representation of B, o_t is the representation of O, a_t is the representation of A, d is a bias term. $(W[s_t; b_t; o_t; a_t] + d)$ is passed through a component-wise rectified linear unit (ReLU) for nonlinearity (Glorot et al., 2011).

Step	Action	Output	Stack	Buffer
0		[]	[]	[a, flight, to, boston, to, denver]
1	OUT	[a]	[]	[flight, to, boston, to, denver]
2	OUT	[a, flight]	[]	[to, boston, to, denver]
3	DEL	[a, flight]	[to]	[boston, to, denver]
4	DEL	[a, flight]	[to, boston]	[to, denver]
5	OUT	[a, flight, to]	[]	[denver]
6	OUT	[a, flight, to, denver]	[]	[]

Table 2: Segmentation process of *a flight to boston to denver*

Finally, the model state e_t is used to compute the probability of the action at time t as:

$$p(z_t|e_t) = \frac{exp(g_{z_t}^{\mathrm{T}} e_t + q_{z_t})}{\sum_{z' \in \mathcal{A}(S,B)} exp(g_{z'}^{\mathrm{T}} e_t + q_{z'})},$$

where g_z is a column vector representing the embedding of the transition action z, and q_z is a bias term for action z. The set $\mathcal{A}(S, B)$ represents the valid actions that may be taken given the current state. Since $e_t = f(s_t, b_t, a_t, o_t)$ encodes information about all previous decisions made by the transition system, the probability of any valid sequence of transition actions z conditioned on the input can be written as:

$$p(z|w) = \prod_{t=1}^{|z|} p(z_t|e_t)$$

We then have

$$\begin{aligned}(D^*, T^*) &= argmax_{D,T} \prod_{i=1}^{|z|} P(d_i, T_1^i | w_1^i, T_1^{i-1}) \\ &= argmax_{D,T} \prod_{t=1}^{|z|} p(z_t|e_t),\end{aligned}$$

where the disfluency detection task is merged into the transition-based system.

Beam Search

The mainly drawback of greedy search is error propagation. An incorrect action will have a negative influence to its subsequent actions, leading to an incorrect output sequence. One way to reduce error propagation is beam-search. Because the number of actions taken always equals to the number of input sentence for every valid path, it is straightforward to use beam search. We use beam-search for both training and testing. The early update strategy from Collins and Roark (2004) is applied for training. In particular, each training sequence is decoded, and we keep track of the location of the gold path in the beam. If the gold path falls out of the beam at step t, decoding process is stopped and parameter update is performed using the gold path as a positive example, and beam items as negative examples. We also use the global optimization method (Andor et al., 2016; Zhou et al., 2015) to train our beam-search model.

Scheduled Sampling

Scheduled sampling (Bengio et al., 2015) can also be used to reduce error propagation. The training goal of the greedy baseline is to maximize the likelihood of each action given the current model state, which means that the correct action is taken at each step. Doing inference, the action predicted by the model itself is taken instead. This discrepancy between training and inference can yield errors that accumulate quickly along the searching process. Scheduled sampling is used to solve the discrepancy by gently changing the training process from a fully guided scheme using the true previous action, towards a less guided scheme which mostly uses the predicting action instead. We take the action gaining higher $p(z_t|e_t)$ with a certain probability p, and a probability $(1-p)$ for the correct action when training.

3.2 State Representation

For better capturing non-local context information, we use LSTM structures to represent different components of each state, including *buffer*, *action*, *stack*, and *output*. In particular, we exploit LSTM-Minus (Wang and Chang, 2016) to model the *buffer* segment, conventional LSTM to model the *action* and *ouptut* segment, and stack LSTM (Dyer et al., 2015) to model the *stack* segments, which demonstrates highly effectively in parsing task.

Buffer Representation

In order to construct more informative representation, we use a Bi-LSTM to represent the *buffer* following the work of Wang and Chang (2016), where the subtraction between a unidirectional

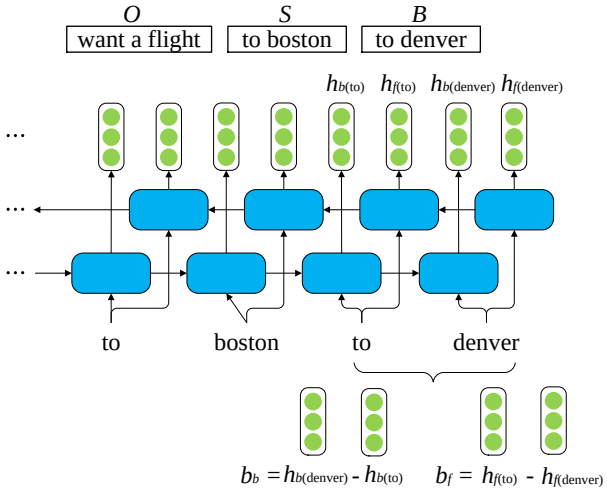

Figure 3: Illustration for learning *buffer* representation based on a Bi-LSTM, $h_f(*)$ and $h_b(*)$ indicate the hidden vectors of forward and backward LSTM respectively.

LSTM hidden vectors is utilized to represent a segment's information. We perform a similar method in a Bi-LSTM to obtain the representation of the *buffer*. The forward and backward subtractions for the *buffer* can be described as $b_f = h_f(l) - h_f(f)$ and $b_b = h_b(f) - h_b(l)$, respectively, where $h_f(f)$ and $h_f(l)$ are the hidden vectors of the first and the last words in the forward LSTM, respectively. Similarly, $h_b(f)$ and $h_b(l)$ are the hidden vectors of the first and the last words in the backward LSTM, respectively. Then these subtractions are concatenated as the representation of the *buffer* $b_t = b_f \oplus b_b$. As illustrated in Figure 3, the forward and backward subtractions for *buffer* are $b_f = h_f(to) - h_f(denver)$ and $b_b = h_b(denver) - h_b(to)$, respectively. Here to is the first word in *buffer* and $denver$ is the last. Then b_f and b_b are concatenated as the representation of *buffer*.

Action Representation

We represent an action a with an embedding $e_a(a)$ from a looking-up table E_a , and apply a conventional LSTM to represent the complete history of actions taken by the transition system. Once an action a is taken, the embedding $e_a(a)$ will be added to the right-most position of the LSTM.

Stack Representation

We use a stack LSTM (Dyer et al., 2015) to represent partial disfluency chunk. The stack LSTM tries to augment the conventional LSTM with a "stack pointer". For a conventional LSTM, new inputs are always added in the right-most position; but in a stack LSTM, the current location of the stack pointer determines which cell in the LSTM provides c_{t-1} and h_{t-1} when computing the new memory cell contents. In addition to adding elements to the end of the sequence, the stack LSTM provides a *pop* operation which moves the stack pointer to the previous element. Thus, the LSTM can be understood as a stack implemented so that contents are never overwritten, When the action OUT is taken, the *stack* is cleared by moving the stack pointer to the initial position. When the action DEL is taken, the representation of the *buffer* will be added directly to the stack LSTM.

Output Representation

We use a conventional LSTM to represent the *output*. When the action OUT is taken, the representation of the *buffer* will be added directly to the right-most position of the LSTM. Because the words in the *output* are a continuous subsequence of the final output sentence with disfluencies removed, the LSTM representation can be seen as a pseudo language model and thus has the ability to keep the generated sentence grammatical, which is very important for disfluency detection.

3.3 Token Embeddings

We use four vectors to represent each input token: a learned word embedding w; a fixed word embedding $\widetilde{w}$; a learned POS-tag embedding p; and a hand-crafted feature representation d. The four vectors are concatenated, transformed by a matrix V and fed to a rectified layer to learn a feature combination:

$$x = max\{0, V[\widetilde{w}; w; p; d] + b\},$$

where V means vector concatenation.

Following the work of Wang et al. (2016), we extract two types of hand-crafted discrete features (as shown in Table 3) for each token in a sentence, and incorporate them into our neural networks by translating them into a 0-1 vector d. The dimension of d is 78, which equals to the number of discrete features. For a token x_t, d_i fires if x_t matches the i-th pattern of the feature templates. The duplicate features indicate whether x_t has a *duplicated* word/POS-tag in certain distance. The *similarity* features indicate whether the surface string of x_t resembles its surrounding words.

duplicate features
$Duplicate(i, w_{i+k}), -15 \leq k \leq +15$ and $k \neq 0$: if w_i equals w_{i+k}, the value is 1, others 0
$Duplicate(p_i, p_{i+k}), -15 \leq k \leq +15$ and $k \neq 0$: if p_i equals p_{i+k}, the value is 1, others 0
$Duplicate(w_i w_{i+1}, w_{i+k} w_{i+k+1}), -4 \leq k \leq +4$ and $k \neq 0$: if $w_i w_{i+1}$ equals $w_{i+k} w_{i+k+1}$, the value is 1, others 0
$Duplicate(p_i p_{i+1}, p_{i+k} p_{i+k+1}), -4 \leq k \leq +4$ and $k \neq 0$: if $p_i p_{i+1}$ equals $p_{i+k} p_{i+k+1}$, the value is 1, others 0
similarity features
$fuzzyMatch(w_i, w_{i+k}), k \in \{-1, +1\}$: $similarity = 2 * num\_same\_letters/(len(w_i) + len(w_{i+k}))$. if $similarity > 0.8$, the value is 1, others 0

Table 3: Discrete features used in our transition-based neural networks. p-POS tag. w-word.

4 Experiments

4.1 Settings

Dataset. Our training data include the Switchboard portion of the English Penn Treebank (Marcus et al., 1993) and a in-house Chinese data set. For English, two annotation layers are provided: one for syntactic bracketing (MRG files), and the other for disfluencies (DPS files). The Switchboard annotation project was not fully completed. Because disfluency annotation is cheaper to produce, many of the DPS training files do not have matching MRG files. Only 619,236 of the 1,482,845 tokens in the DPS disfluency detection training data have gold-standard syntactic parses. To directly compare with transition-based parsing methods (Honnibal and Johnson, 2014; Wu et al., 2015), we also use the subcorpus of PARSED/MRG/SWBD. Following the experiment settings in Charniak and Johnson (2001), the training subcorpus contains directories 2 and 3 in PARSED/MRG/SWBD and directory 4 is split into test, development sets and others. Following Honnibal and Johnson (2014), we lower-case the text and remove all punctuations and partial words[2]. We also discard the 'um' and 'uh' tokens and merge 'you know' and 'i mean' into single tokens. Automatic POS-tags generated from pocket_crf (Qian and Liu, 2013) are used as POS-tag in our experiments.

For Chinese experiments, we collect 25k spoken sentences from meeting minutes, which are transcribed using the iflyrec toolkit[3], and annotate them with only disfluency annotations according to the guideline proposed by Meteer et al. (1995).

[2]words are recognized as partial words if they are tagged as 'XX' or end with '-'

[3]the iflyrec toolkit is available at http://www.iflyrec.com/

4.2 Neural Network Training

Pretrained Word Embeddings. Following Dyer et al. (2015) and Wang et al. (2016), we use a variant of the skip n-gram model introduced by Ling et al. (2015), named "structured skip n-gram", to create word embeddings. The AFP portion of English Gigaword corpus (version 5) is used as the training corpus. Word embeddings for Chinese are trained on Chinese baike corpus. We use an embedding dimension of 100 for English, 300 for chinese.

Hyper-Parameters. Both the Bi-LSTMs and the stack LSTMs have two hidden layers and their dimensions are set to 100. Pretrained word embeddings have 100 dimensions and the learned word embeddings have also 100 dimensions. Pos-tag embeddings have 12 dimensions. The dimension of action embeddings is set to 20.

4.3 Performance On English Swtichboard

We build two baseline systems using CRF and Bi-LSTM, respectively. The hand-crafted discrete features of CRF refer to those in Ferguson et al. (2015). For the Bi-LSTM model, the token embedding is the same with our transition-based method. Table 4 shows the result of our model on both the development and test sets. Beam search improves the F-score form 87.1% to 87.5%, which is consistent with the finding of Buckman et al. (2016) on the LSTM parser of (Dyer et al., 2015) (improvements by about 0.3 point). Scheduled sampling achieves the same improvements compared to beam-search. Because of high training speed, we conduct subsequent experiments based on scheduled sampling.

We compare our transition-based neural model to five top performing systems. Our model outperforms the state-of-the-art, achieving a 87.5% F-

Method	Dev			Test		
	P	R	F1	P	R	F1
CRF	93.9	78.3	85.4	91.7	75.1	82.6
Bi-LSTM	94.1	79.3	86.1	91.7	80.6	85.8
greedy	91.4	83.7	87.4	91.1	83.3	87.1
+beam	93.6	83.6	88.3	92.8	82.7	**87.5**
+schedualed	92.3	84.3	88.1	91.1	84.1	**87.5**

Table 4: Experiment results on the development and test data of English Switchboard data.

Method	**P**	**R**	**F1**
Our	91.1	84.1	**87.5**
Attention-based (Wang et al., 2016)	91.6	82.3	86.7
Bi-LSTM (Zayats et al., 2016)	91.8	80.6	85.9
semi-CRF (Ferguson et al., 2015)	90.0	81.2	85.4
UBT (Wu et al., 2015)	90.3	80.5	85.1
M^3N (Qian and Liu, 2013)	-	-	84.1

Table 5: Comparison with previous state-of-the-art methods on the test set of English Switchboard.

score as shown in Table 5. It achieves 2.4 point improvements over UBT (Wu et al., 2015), which is the best syntax-based method for disfluency detection. The best performance by linear statistical sequence labeling methods is the semi-CRF method of Ferguson et al. (2015), achieving a 85.4% F-score leveraging prosodic features. Our model obtains a 2.1 point improvement compared to this. Our model also achieves 0.8 point improvement over the neural attention-based model (Wang et al., 2016), which regards the disfluency detection as a sequence-to-sequence problem. We attribute the success to the strong ability to learn global chunk-level features and the good state representation such as the stack-LSTM.

4.4 Result On DPS Corpus

As described in section 3.1, to directly compare with the transition-based parsing methods (Honnibal and Johnson, 2014; Wu et al., 2015), we only use MRG files, which are less than the DPS files. In fact, many methods, such as Qian and Liu (2013), have used all the DPS files as training data. We are curious about the performance of our system using all the DPS files. Following the experimental settings of Johnson and Charniak (2004), the corpus is split as follows: main training consisting of all sw[23]*.dps files, development training consisting of all sw4[5-9]*.dps files and test training consisting of all sw4[0-1]*.mrg files. Table 6 shows the result on the DPS files.

Method	**P**	**R**	**F1**
Our	93.1	83.5	**88.1**
Bi-LSTM	92.4	82.0	86.9
M^3N^* (Qian and Liu, 2013)	90.6	78.7	84.2
CRF	91.8	77.2	83.9

Table 6: Test result of our transition-based model using DPS files for training.

Method	Dev			Test		
	P	R	F1	P	R	F1
Our	68.9	40.4	50.9	77.2	37.7	**50.6**
Bi-LSTM	60.1	41.3	48.9	65.3	38.2	48.2
CRF	73.7	33.5	46.1	77.7	32.0	45.3

Table 7: performance on Chinese annotated data

The result of M^3N^* comes from our experiments with the toolkit[4] released by Qian and Liu (2013), which use the same data set and pre-processing. Our model achieves a 88.1% F-score by using more training data, obtaining 0.6 point improvement compared with the system training on MRG files. The performance is far better than the sequence labeling methods that use DPS files for training.

4.5 Performance on Chinese

Table 7 shows the results of Chinese disfluency detection. Our model obtains a 2.4 point improvement compared with the baseline Bi-LSTM model and a 5.3 point compared with the baseline CRF model. The performance on Chinese is much lower than that on English. Apart from the smaller training set, the main reason is that the proportion of repair type disflueny is much higher.

5 Analysis

5.1 Ablation Tests

As described in section 3.1, the sate representation has four components. We explicitly compare the impact of different parts. As shown in Table 8, the F-score decreases most heavily without *stack*, which indicates that it is very necessary to capture chunk-level information for disfluency detection and our model can model it effectively. The results also show that *output*, which can be seen as a pseudo language model, has important influence on model performance. Seen from the result, history of actions represented in *action* is also useful for predicting at each step. The F-score decreases

[4]The toolkit is available at https://code.google.com/p/disfluency-detection/downloads.

Method	P	R	F1
ALL	91.1	84.1	**87.5**
- *stack*	93.5	80.6	86.5
- *action*	91.6	83.0	87.1
- *output*	89.0	84.4	86.7
- Bi-LSTM	93.6	81.4	87.1

Table 8: Results of feature ablation experiments on English Switchboard test data. "- Bi-LSTM" means using unidirectional LSTM for *buffer*

about 0.4 point, which shows that Bi-LSTM can capture more information compared to simple unidirectional LSTM.

5.2 Repetitions vs Non-repetitions

Repetition disfluencies are easier to detect and even some simple hand-crafted features can handle them well. Other types of reparandums such as repair are more complex (Zayats et al., 2016; Ostendorf and Hahn, 2013). In order to better understand model performances, we evaluate our model's ability to detect repetition vs. non-repetition (other) reparandum. The results are shown in Table 9. All the three models achieve high score on repetition reparandum. Our transition-based model is much better in predicting non-repetitions compared to CRF and Bi-LSTM. We conjecture that our transition-based structure can capture more of the reparandum/repair "rough copy" similarities by learning represention of both chunks and global state.

Method	Repetitions	Non-repetitions	Either
CRF	93.8	61.4	82.6
Bi-LSTM	93.1	65.3	85.8
OUR	93.3	68.7	**87.5**

Table 9: F-score of different types of reparandums on English Switchboard test data.

6 Related Work

Common approaches take disfluency detection as a sequence labeling problem, where each sentential word is assigned with a label (Georgila, 2009; Qian and Liu, 2013). These methods achieve good performance, but are not powerful enough to capture complicated disfluencies with longer spans or distances. Another drawback is that they have no ability to exploit chunk-level features. There are also works that try to use recurrent neural network (RNN), which can capture dependencies at any length in theory, on disfluency detection problem (Zayats et al., 2016; Hough and Schlangen, 2015). The RNN method treats sequence tagging as classification on each input token. Hence, it also has no power to exploit chunk-level features. Some works (Wang et al., 2016) regard the disfluency detection as a sequence-to-sequence problem and propose a neural attention-based model for it. The attention-based model can capture a global representation of the input sentence by using a RNN when encoding. It can strongly capture long-range dependencies and achieves good performance, but are also not powerful enough to capture chunk-level information. To capture chunk-level information, Ferguson et al. (2015) try to use semi-CRF for disfluency detection, and reports improved results. Semi-CRF models still have their inefficiencies because they can only use the local chunk information limited by the markov assumption when decoding.

Many syntax-based approaches (Lease and Johnson, 2006; Rasooli and Tetreault, 2013; Honnibal and Johnson, 2014; Wu et al., 2015) have been proposed which jointly perform dependency parsing and disfluency detection. The main advantage of joint models is that they can capture long-range dependency of disfluencies. However, it requires that the training data contains both syntax trees and disfluency annotations, which reduces the practicality of the algorithm. The performance relies heavily on manual feature engineering.

Transition-based framework has been widely exploited in a number of other NLP tasks, including syntactic parsing (Zhang and Nivre, 2011; Zhu et al., 2013), information extraction (Li and Ji, 2014) and joint syntactic models (Zhang et al., 2013, 2014).

Recently, deep learning methods have been widely used in many nature language processing tasks, such as name entity recognition (Lample et al., 2016), zero pronoun resolution (Yin et al., 2017) and word segmentation (Zhang et al., 2016). The effectiveness of neural features has also been studied for this framework (Zhou et al., 2015; Watanabe and Sumita, 2015; Andor et al., 2016). We apply the transition-based neural framework to disfluency detection, which to our knowledge has not been investigated before.

7 Conclusion

We introduced a transition-based model for disfluency detection, which does not use any syntax information, learning represention of both chunks and global contexts. Experiments showed that our model achieves the state-of-the-art F-scores on both the commonly used English Switchboard test set and a in-house annotated Chinese data set.

Acknowledgments

We thank the anonymous reviewers for their valuable suggestions. This work was supported by the National Key Basic Research Program of China via grant 2014CB340503 and the National Natural Science Foundation of China (NSFC) via grant 61370164 and 61632011. The Chinese disfluency data is annotated by IFLYTEK CO.,LTD.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany. Association for Computational Linguistics.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

Jacob Buckman, Miguel Ballesteros, and Chris Dyer. 2016. Transition-based dependency parsing with heuristic backtracking. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing; 2016 Nov 1-5; Austin, Texas, USA. Stroudsburg (USA): Association for Computational Linguistics (ACL); 2016. p. 2313-18.* ACL (Association for Computational Linguistics).

Eugene Charniak and Mark Johnson. 2001. Edit detection and parsing for transcribed speech. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–9. Association for Computational Linguistics.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China. Association for Computational Linguistics.

James Ferguson, Greg Durrett, and Dan Klein. 2015. Disfluency detection with a semi-markov model and prosodic features. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 257–262. Association for Computational Linguistics.

Kallirroi Georgila. 2009. Using integer linear programming for detecting speech disfluencies. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 109–112. Association for Computational Linguistics.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275.

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

Matthew Honnibal and Mark Johnson. 2014. Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics*, 2:131–142.

Julian Hough and David Schlangen. 2015. Recurrent neural networks for incremental disfluency detection. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Mark Johnson and Eugene Charniak. 2004. A tag-based noisy channel model of speech repairs. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 33. Association for Computational Linguistics.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Matthew Lease and Mark Johnson. 2006. Early deletion of fillers in processing conversational speech. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 73–76. Association for Computational Linguistics.

Qi Li and Heng Ji. 2014. Incremental joint extraction of entity mentions and relations. In *Proceedings of the 52nd Annual Meeting of the Association*

for Computational Linguistics (Volume 1: Long Papers), pages 402–412, Baltimore, Maryland. Association for Computational Linguistics.

Wang Ling, Chris Dyer, Alan Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Marie W Meteer, Ann A Taylor, Robert MacIntyre, and Rukmini Iyer. 1995. *Dysfluency annotation stylebook for the switchboard corpus*. University of Pennsylvania.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Mari Ostendorf and Sangyun Hahn. 2013. A sequential repetition model for improved disfluency detection. In *INTERSPEECH*, pages 2624–2628.

Xian Qian and Yang Liu. 2013. Disfluency detection using multi-step stacked learning. In *HLT-NAACL*, pages 820–825.

Mohammad Sadegh Rasooli and Joel R Tetreault. 2013. Joint parsing and disfluency detection in linear time. In *EMNLP*, pages 124–129.

Elizabeth Ellen Shriberg. 1994. *Preliminaries to a theory of speech disfluencies*. Ph.D. thesis, Citeseer.

Shaolei Wang, Wanxiang Che, and Ting Liu. 2016. A neural attention model for disfluency detection. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 278–287, Osaka, Japan. The COLING 2016 Organizing Committee.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional lstm. In *Proc. of ACL*, pages 2306–2315.

Taro Watanabe and Eiichiro Sumita. 2015. Transition-based neural constituent parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1169–1179, Beijing, China. Association for Computational Linguistics.

Shuangzhi Wu, Dongdong Zhang, Ming Zhou, and Tiejun Zhao. 2015. Efficient disfluency detection with transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 495–503. Association for Computational Linguistics.

Qingyu Yin, Yu Zhang, Weinan Zhang, and Ting Liu. 2017. A deep neural network for chinese zero pronoun resolution. In *Proceedings of the 26th International Conference on Artificial Intelligence*, IJCAI'17. AAAI Press.

Vicky Zayats, Mari Ostendorf, and Hannaneh Hajishirzi. 2016. Disfluency detection using a bidirectional lstm. *arXiv preprint arXiv:1604.03209*.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2013. Chinese parsing exploiting characters. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 125–134, Sofia, Bulgaria. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2014. Character-level chinese dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1326–1336, Baltimore, Maryland. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, and Guohong Fu. 2016. Transition-based neural word segmentation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 421–431, Berlin, Germany. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA. Association for Computational Linguistics.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1213–1222, Beijing, China. Association for Computational Linguistics.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria. Association for Computational Linguistics.