Joint Transition-based Dependency Parsing and Disfluency Detection for Automatic Speech Recognition Texts

Masashi Yoshikawa and **Hiroyuki Shindo** and **Yuji Matsumoto**
Graduate School of Information and Science
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara, 630-0192, Japan
{ masashi.yoshikawa.yh8, shindo, matsu }@is.naist.jp

Abstract

Joint dependency parsing with disfluency detection is an important task in speech language processing. Recent methods show high performance for this task, although most authors make the unrealistic assumption that input texts are transcribed by human annotators. In real-world applications, the input text is typically the output of an automatic speech recognition (ASR) system, which implies that the text contains not only disfluency noises but also recognition errors from the ASR system. In this work, we propose a parsing method that handles both disfluency and ASR errors using an incremental shift-reduce algorithm with several novel features suited to ASR output texts. Because the gold dependency information is usually annotated only on transcribed texts, we also introduce an alignment-based method for transferring the gold dependency annotation to the ASR output texts to construct training data for our parser. We conducted an experiment on the Switchboard corpus and show that our method outperforms conventional methods in terms of dependency parsing and disfluency detection.

1 Introduction

Spontaneous speech is different from written text in many ways, one of which is that it contains disfluencies, that is, parts of the utterance that are corrected by the speaker during the utterance. NLP system performance is reported to deteriorate when there are disfluencies, for example, with SMT (Cho et al., 2014). Therefore, it is desirable to preprocess the speech before passing it to other NLP tasks.

There are a number of studies that address the problem of detecting disfluencies. Some of these studies include dependency parsing (Honnibal and Johnson, 2014; Wu et al., 2015; Rasooli and Tetreault, 2014), whereas others are dedicated systems (Qian and Liu, 2013; Ferguson et al., 2015; Hough and Purver, 2014; Hough and Schlangen, 2015; Liu et al., 2003). Among these studies, Honnibal (2014) and Wu (2015) address this problem by adding a new action to transition-based dependency parsing that removes the disfluent parts of the input sentence from the stack. Using this approach, they achieved high performance in terms of both dependency parsing and disfluency detection on the Switchboard corpus.

However, the authors assume that the input texts to parse are transcribed by human annotators, which, in practice, is unrealistic. In real-world applications, in addition to disfluencies, the input texts contain ASR errors; these issues might degrade the parsing performance. For example, proper nouns that are not contained in the ASR system vocabulary may break up into smaller pieces, yielding a difficult problem for the parsing unit (Cheng et al., 2015):

REF: what can we get at **Litanfeeth**
HYP: what can we get *it leaks on feet*

In this work, we propose a method for joint dependency parsing and disfluency detection that can robustly parse ASR output texts. Our parser handles both disfluencies and ASR errors using an incremental shift-reduce algorithm, with novel features that consider recognition errors of the ASR system.

Furthermore, to evaluate dependency parsing per-

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1036–1041,
Austin, Texas, November 1-5, 2016.

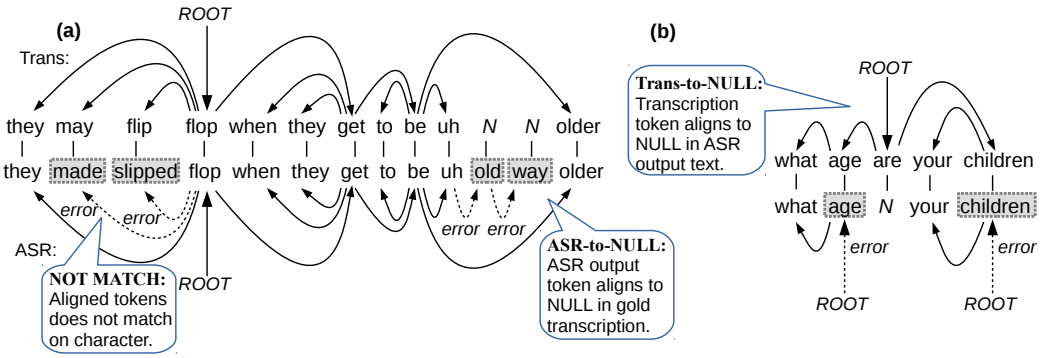

Figure 1: Examples of three problematic cases. Above shows the gold transcription and its tree, below shows the aligned ASR output and its newly transferred tree, where the dotted edges are ASR error edges.

formance on real human utterances, we create a tree-annotated corpus that contains ASR errors. [1]

2 Data Creation

To evaluate dependency parsing performance on real speech texts, we must create a tree-annotated corpus of ASR output texts.

Given a corpus that consists of speech data, transcription text and its syntactic annotation (e.g., the Switchboard corpus), we first apply the ASR system to the speech data. Next, we perform alignment between the ASR output texts and the transcription. Then, we transfer the gold syntactic annotation to the ASR output texts based on this alignment (Figure 1). The alignment is performed by minimizing the edit distance between the two sentences. We include "NULL" tokens in this alignment to allow for some tokens not having an aligned conterpart ("*N*" tokens in the Figure 1).

In the constructed trees, there are three problematic cases based on how an ASR output text and its transcription are aligned with each other: (1) a word in the ASR output text aligns with a NULL token in the transcription (**ASR-to-NULL**), (2) a word in the gold transcription aligns with a NULL in the ASR output (**Trans-to-NULL**), and (3) two words align, but do not match exactly in terms of characters (**NOT MATCH**). To create a consistent dependency tree that spans the entire sentence, we must address each of these cases.

[1]There are also studies that tackle the problem of disfluency detection in the context of speech recognition such as (Liu et al., 2003). Our work is novel in that our aim is to extend the joint method of disfluency detection with dependency parsing so that it can be applicable to the output of ASR system.

2.1 ASR-to-NULL

In the case of ASR-to-NULL, a token from the ASR system has no corresponding token in the gold transcription. In this case, we automatically annotate a dependency relation with an "error" label such that the token's head becomes the previous word token.

Figure 1(a) shows an example of this case. In the figure, the words "old" and "way" have no corresponding words in the gold transcription. Thus, we automatically annotate the dependency relations between ("old", "uh") and ("way", "old"), respectively, with the "error" label.

2.2 Trans-to-NULL

Although NULL tokens are introduced to facilitate alignment, as these tokens in the ASR output are not actual words, we must remove them in the final tree. Without any treatment, the gold transcription tokens aligned to these tokens are also deleted along with them. This causes the child tokens in the sentence not to have heads; consequently, these child tokens are not included in the syntactic tree. To avoid this problem, we instead attach them to the head of the deleted token.

For example, in Figure 1(b), the word "are" is missing in the ASR hypothesis. Then, this token's children lose their head in the transfer process. Thus, we rescue these children by attaching them to the head of "are", which, in this case, is ROOT token.

If the head of the removed token is also of the Trans-to-NULL type, then we look for an alternative head by climbing the tree in a recursive manner, until reaching ROOT. We also label the newly created edges in this process as "error".

2.3 NOT MATCH

In cases in which two aligned tokens do not match exactly on the character level, the mismatch is regarded as an instance of a substitution type of ASR error. Therefore, we encode this fact in the label of the arc from the token to its head.

In Figure 1(a), the words "made" and "slipped" in the ASR hypothesis do not match the gold transcription tokens, "may" and "flip", respectively. Therefore, we automatically re-label the arc from each token to its head as "error".

3 Transition-based Dependency Parsing

To parse texts that contain disfluencies and ASR errors, we extend the ArcEager shift-reduce dependency parser of (Zhang and Nivre, 2011). Our proposed parser adopts the same *Shift*, *Reduce*, *LeftArc*, and *RightArc* actions as ArcEager. To this parser we add three new actions, i.e., *Edit*, *LeftArcError*, and *RightArcError*, to handle disfluencies and ASR errors.

Edit action removes a disfluent token when it is the first element of the stack. This is different from Honnibal (2014)'s *Edit* action: theirs accumulates consecutive disfluent tokens on the top of the stack and removes them all at once, whereas our method removes this kind of token one-by-one. Use of this *Edit* action guarantees that the length of the action sequence is always $2n-1$. This property is advantageous because the parser can use the standard beam search and does not require normalization, such as those adopted in (Honnibal and Johnson, 2014) and (Zhu et al., 2013).

LeftArcError and *RightArcError* act in the same way as *LeftArc* and *RightArc*, except that these act only on ASR error tokens, whereas the original *LeftArc* and *RightArc* are reserved for non ASR error tokens. Using two different kinds of *Arc* actions for the two types of tokens (ASR error or not) allows for the weights not to be shared between them, and is expected to yield improved performance.

In the experiment below, we train all of the models using structured perceptron with max violation (Huang et al., 2012). The feature set is mainly based on (Honnibal and Johnson, 2014), such as the disfluency capturing features to inquire whether the token sequence inside the two specific spans match on word forms or POS tags. We adjusted these features to inspect the content of the buffer more carefully, because our parser decides if the word token is disfluent or not every time new token is shifted and hints for the decision lies much more in the buffer.

3.1 Backoff Action Feature

With the newly proposed *LeftArcError* and *RightArcError* actions, we fear that the relatively low frequency of "error" tokens may cause the weights for these actions to be updated too infrequently to be accurately generalized. We resort to using the "backoff action feature" to avoid this situation. This means that, for each action $a \in \{LeftArc, LeftArcError\}$, the score of performing it in a state s is calculated as follow:

$$SCORE(a, s) = \mathbf{w} \cdot \mathbf{f}(a, s) + \mathbf{w} \cdot \mathbf{f}(a', s) \quad (1)$$

where a' = *LeftArcBackoff*, $\mathbf{w}$ is the weight vector and $\mathbf{f}(\cdot, \cdot)$ is the feature representation, respectively. *LeftArcBackoff* is not actual action performed by our parser, rather it is used to provide the common feature representation which both *LeftArc* and *LeftArcError* can "back off" to. *RightArc* and *RightArcError* actions also calculate their scores as in Eq.(1), with a' = *RightArcBackoff*. The scores for all the other actions are calculated in the normal way: $SCORE(a, s) = \mathbf{w} \cdot \mathbf{f}(a, s)$.

3.2 WCN Feature

To better capture which parts of the texts are likely to be ASR errors, we use additional features extracted from a word confusion network (WCN) generated by ASR models. Marin (2015) reports his observation that WCN slots with more arcs tend to correspond to erroneous region. Following (Marin, 2015), we use mean and standard deviation of arc posteriors and the highest arc posterior in each WCN slot corresponding to each word token. We include in the feature vector these real-valued features for tokens on top of the stack and the first three elements of the buffer.

4 Experiment

We conducted experiments using both the proposed parsing method and the tree-annotated corpus based on the ASR output texts. Our experiments were performed using the Switchboard corpus (Godfrey et

al., 1992). This corpus consists of speech data and its transcription texts, and subset of which is annotated with POS tags, syntactic trees and disfluency information (repair, discourse marker and interjection) based on (Shriberg, 1994). [2]

4.1 ASR Settings

To obtain the ASR output texts of the corpus, we used the off-the-shelf NeuralNet recipe (Zhang et al., 2014) presented by Kaldi.[3] We used the jackknife method to obtain the ASR output texts throughout the syntactically annotated part of the corpus. [4]

From these ASR output texts, we created the tree-annotated corpus by applying the data creation method introduced in §2. Out of all 857,493 word tokens, there are 32,606 ASR-to-NULL, 34,952 Trans-to-NULL, and 93,138 NOT MATCH cases, meaning 15.6% of all word tokens had "error" labeled arcs.

4.2 Parsing Settings

We assigned POS tags to the created corpus using the Stanford POS tagger (Toutanova et al., 2003) trained on a part of the gold Switchboard corpus. [5]

We adopt the same train/dev/test split as in (Honnibal and Johnson, 2014), although the data size reduces slightly during the process of data creation. We report the unlabeled attachment score (UAS), which indicates how many heads of fluent tokens are correctly predicted. As for disfluency detection, we report precision/recall/F1-score values following the previous work in the literature.

As a baseline (To which we refer as *Base* in the following), we use an ArcEager parser with our proposed *Edit* action and the disfluency capturing features, trained on the train part of the gold Switchboard corpus. Using this parser on ASR output test data can be seen as reproducing the typical situation, in which a parser is trained on ASR-error-free texts, but nevertheless needs to parse the ASR output texts.

[2]We converted the phrase structure trees to dependency ones using the Stanford converter (de Marneffe et al., 2006).

[3]http://kaldi-asr.org/

[4]The average Word Error Rate of resulting models were 13.9 % on the Switchboard part of HUB5 evaluation dataset: https://catalog.ldc.upenn.edu/LDC2002S09

[5]We used a part of the corpus that is annotated with POS information but not syntactic one. The performance of the tagger is evaluated on the syntactically annotated part of the corpus; the tagger has an accuracy score of 95.0%.

Model	Dep	Disfl		
	UAS	Prec.	Rec.	F1
Base	72.7	58.6	**62.2**	60.3
+ *ErrorAct*	76.3	66.0	57.6	61.5
+ *Backoff*	**76.4**	65.6	57.3	61.1
+ *WCN*	76.2	**67.9**	57.9	**62.5**

Table 1: Dependency parsing and disfluency detection results of the proposed methods. We used our created corpus as both train and test data.

Train	Test	Model	Dep	Disfl		
			UAS	Prec.	Rec.	F1
$Trans$	$Trans$	*Base*	89.7	90.4	76.8	83.1
$Trans$	ASR	*Base*	74.7	58.5	**65.6**	61.8
ASR	ASR	*Base*	72.7	58.6	62.2	60.3
ASR	ASR	*Ours*	**76.2**	**67.9**	57.9	**62.5**

Table 2: Parsing result on different train-test settings. $Trans$ refers to original Switchboard transcription text, ASR the text created through the data creation in §4.1. *Ours* is our proposed parser: *Base* + *ErrorAct* + *Backoff* + *WCN*.

4.3 Results and Analysis

In Table 1, based on the baseline *Base* parser, we report scores with the additional (and additive) use of *Left/RightArcError* actions (*ErrorAct*), the WCN feature (*WCN*), and the backoff action feature (*Backoff*), on our created corpus. Using *ErrorAct* resulted in 3.6% and 1.2% improvement in UAS and disfluency detection F1, respectively. *Backoff* contributes to further improved UAS, whereas *WCN* cause an increase in disfluency detection accuracy.

Table 2 reports performance on various train and test data settings. In Table 2, the Train and Test columns represent which data to use in training and testing; $Trans$ refers to the gold transcription text of the Switchboard corpus, and ASR the text created through the data creation in §4.1. When evaluated on the ASR texts, the parser trained on the ASR texts showed degraded performance compared to the parser trained on the gold transcription ($(\text{Train}, \text{Test}) = (ASR, ASR)$). Although both the train and test data are ASR texts and share characteristics, we did not observe domain adaptation effect. We hypothesized that the drop in the performance is due to the noisy nature of our corpus, which is created from the texts with ASR errors. Having ASR-

error-specific actions, *Left/RightArcError* mitigates this problem by separately treating the ASR error tokens and non ASR error tokens. Finally, with the newly proposed features, the parser trained on ASR texts outperforms the parser trained on the transcription texts with the improvement of 1.5% and 0.7% for UAS and disfluency detection, respectively.

However, when compared with the case of $(\text{Train}, \text{Test}) = (Trans, Trans)$, we observe significant decreases in performance in both of the tasks conducted on ASR texts. This result clearly poses a new challenge for the disfluency detection community.

5 Conclusion

In this work, we have proposed a novel joint transition-based dependency parsing method with disfluency detection. Using new actions, and new feature set, the proposed parser can parse ASR output texts robustly. We have also introduced a data construction method to evaluate dependency parsing and disfluency detection performance for real speech data. As the experimental results for ASR texts is significantly lower than that achieved for the gold transcription texts, we have clarified the need to develop a method that is robust to recognition errors in the ASR system.

6 Acknowledgements

We thank the three anonymous reviewers for their detailed and insightful comments on an earlier draft of this paper. This work was supported by JSPS KAKENHI Grant Number 15K16053, 26240035.

References

Hao Cheng, Hao Fang, and Mari Ostendorf. 2015. Open-domain name error detection using a multi-task rnn. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 737–746. Association for Computational Linguistics.

Eunah Cho, Jan Niehues, and Alex Waibel. 2014. Tight integration of speech disfluency removal into smt. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers (EACL)*, pages 43–47. Association for Computational Linguistics.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *In Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*.

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 (NAACL)*, pages 257–262. Association for Computational Linguistics.

J. J. Godfrey, E. C. Holliman, and J. McDaniel. 1992. " switchboard: Telephone speech corpus for research and development ". In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on (Volume:1)*. Proc. IEEE Int. Conf. Acoust. Speech Sig. Proc.

Matthew Honnibal and Mark Johnson. 2014. Joint incremental disfluency detection and dependency parsing. In *Transactions of the Association of Computational Linguistics Volume 2, Issue 1 (TACL)*, pages 131–142. Association for Computational Linguistics.

Julian Hough and Matthew Purver. 2014. Strongly incremental repair detection. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 78–89. Association for Computational Linguistics.

Julian Hough and David Schlangen. 2015. Recurrent neural networks for incremental disfluency detection. Interspeech 2015.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Yang Liu, Elizabeth Shriberg, and Andreas Stolcke. 2003. Automatic disfluency identification in coversational speech using multiple knowledge sources. In *In Proceedings of the 8th Eurospeech Conference*.

Marius Alexandru Marin. 2015. In *Effective Use of Cross-Domain Parsing in Automatic Speech Recognition and Error Detection. Ph.D. thesis*. University of Washington.

Xian Qian and Yang Liu. 2013. Disfluency detection using multi-step stacked learning. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics.

Mohammad Sadegh Rasooli and Joel Tetreault. 2014. Non-monotonic parsing of fluent umm i mean disfluent sentences. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers (EACL)*,

pages 48–53. Association for Computational Linguistics.

Elizabeth Shriberg. 1994. In *Preliminaries to a Theory of Speech Disfluencies. Ph.D. thesis*. University of California, Berkeley.

Kristina Toutanova, Dan Klein, Christopher Manning, and Singer Yoram. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *In Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 173–180. 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) (ACL)*, pages 495–503. 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 (ACL)*, pages 188–193. Association for Computational Linguistics.

Xiaohui Zhang, Jan Trmal, Daniel Povey, and Sanjeev Khudanpur. 2014. Improving deep neural network acoustic models using generalized maxout networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Muhua Zhu, Yue Zhang, Wenliang Chen, Miu Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 434–443. Association for Computational Linguistics.