Neural Constituency Parsing of Speech Transcripts

Paria Jamshid Lou
Macquarie University
Sydney, Australia

Yufei Wang
Macquarie University
Sydney, Australia

{paria.jamshid-lou,yufei.wang}@hdr.mq.edu.au

Mark Johnson
Macquarie University
Sydney, Australia
mark.johnson@mq.edu.au

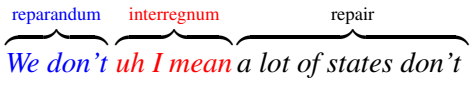
Abstract

This paper studies the performance of a neural self-attentive parser on transcribed speech. Speech presents parsing challenges that do not appear in written text, such as the lack of punctuation and the presence of speech disfluencies (including filled pauses, repetitions, corrections, etc.). Disfluencies are especially problematic for conventional syntactic parsers, which typically fail to find any EDITED disfluency nodes at all. This motivated the development of special disfluency detection systems, and special mechanisms added to parsers specifically to handle disfluencies. However, we show here that neural parsers can find EDITED disfluency nodes, and the best neural parsers find them with an accuracy surpassing that of specialized disfluency detection systems, thus making these specialized mechanisms unnecessary. This paper also investigates a modified loss function that puts more weight on EDITED nodes. It also describes tree-transformations that simplify the disfluency detection task by providing alternative encodings of disfluencies and syntactic information.

1 Introduction

While a great deal of effort has been expended on parsing written text, parsing speech (either transcribed or ASR output) has received less attention. Parsing speech is important because speech is the easiest and most natural means of communication, it is increasingly used as an input modality in human-computer interactions. Speech presents parsing challenges that do not appear in written text, such as the lack of punctuation and sentence boundaries, speech recognition errors and the presence of speech disfluencies (including filled pauses, repetitions, corrections, etc.) (Kahn et al., 2005). Of the major challenges associated with transcribed speech, we focus here on speech disfluencies, which are frequent in spontaneous speech.

Disfluencies include filled pauses ("um", "uh"), parenthetical asides ("you know", "I mean"), interjections ("well", "like") and partial words ("wou-", "oper-"). One type of disfluency which is especially problematic for conventional syntactic parsers are speech repairs. Following the analysis of Shriberg (1994), a speech repair consists of three main parts; the *reparandum*, the *interregnum* and the *repair*. As illustrated in the following example, the reparandum *we don't* is the part of the utterance that is replaced or repaired, the interregnum *uh I mean* (which consists of a filled pause *uh* and a discourse marker *I mean*) is an optional part of the disfluency, and the repair *a lot of states don't* replaces the reparandum. The fluent version is obtained by removing the reparandum and the interregnum.



(1)

In the Switchboard treebank corpus (Mitchell et al., 1999) the reparanda, filled pauses and discourse markers are dominated by EDITED, INTJ and PRN nodes, respectively (see Figure 1). Of these disfluency nodes, EDITED nodes pose a major problem for conventional syntactic parsers, as the parsers typically fail to find any EDITED nodes at all. Conventional parsers mainly capture tree-structured dependencies between words, while the relation between reparandum and repair is quite different: the repair is often a "rough copy" of the reparandum, using the same or very similar words in roughly the same order (Charniak and Johnson, 2001; Johnson and Charniak, 2004). The "rough copy" dependencies are strong evidence of a disfluency, but conventional syntac-

tic parsers cannot capture them. Moreover, the reparandum and the repair do not form conventional syntactic phrases, as illustrated in Figure 1, which is an additional difficulty when integrating disfluency detection with syntactic parsing. This motivated the development of special disfluency detection systems which find and remove disfluent words from the input prior to parsing (Charniak and Johnson, 2001; Kahn et al., 2005; Lease and Johnson, 2006), and special mechanisms added to parsers specifically to handle disfluencies (Rasooli and Tetreault, 2013; Honnibal and Johnson, 2014; Yoshikawa et al., 2016; Tran et al., 2018).

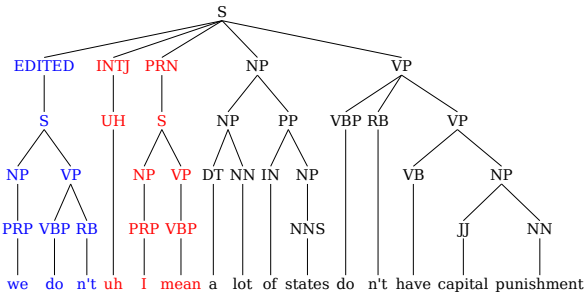


Figure 1: An example parse tree from the Switchboard corpus – *We don't uh I mean a lot of states don't have capital punishment,* where reparandum *We don't*, filled pause *uh* and discourse marker *I mean* are dominated by EDITED, INTJ and PRN nodes.

In this paper, we investigate the performance of a neural self-attentive constituency parser on speech transcripts. We show that an "off-the-shelf" self-attentive parser, unlike conventional parsers, can detect disfluent words with a performance which is competitive to or better than specialized disfluency detection systems. In summary, the main contributions of this paper are:

- We show that the self-attentive constituency parser sets a new state-of-the-art for syntactic parsing of transcribed speech,
- A neural constituency parser can detect EDITED words with an accuracy surpassing that of specialized disfluency detection models,
- We demonstrate that syntactic information helps the neural syntactic parsing detect disfluent words more accurately,
- Replacing the constituent-based representation of disfluencies with a word-based representation of disfluencies improves the detection of disfluent words,
- Modifying the training loss function to put more weight on EDITED nodes during training also improves disfluency detection.

2 Related Work

Speech recognition errors, unknown sentence boundaries and disfluencies are three major problems addressed by previous work on parsing speech. In this work, we focus on the problem of disfluency detection in parsing human-transcribed speech, where we assume that sentence boundaries are given and there are no word recognition errors. This section reviews approaches that add special mechanisms to parsers to handle disfluencies as well as specialized disfluency detection models.

2.1 Joint Parsing and Disfluency Detection

Many speech parsers adopt a transition-based dependency approach to (i) find the relationship between head words and words modifying the heads, and (ii) detect and remove disfluent words and their dependencies from the sentence. Transition-based parsers can be augmented with new parse actions to specifically handle disfluent words (Rasooli and Tetreault, 2013; Honnibal and Johnson, 2014; Yoshikawa et al., 2016; Wu et al., 2015). A classifier is trained to choose between the standard and the augmented parse actions at each time step. Using pattern-match features in the classifier significantly improves disfluency detection (Honnibal and Johnson, 2014). This reflects the fact that parsing based models use pattern-matching to capture the "rough copy" dependencies that are characteristic of speech disfluencies.

Speech parsing models usually use lexical features. One recent approach (Tran et al., 2018) integrates lexical and prosodic cues in an encoder-decoder constituency parser. Prosodic cues result in very small performance gain in both parsing and disfluency detection. Augmenting the parser with a location-aware attention mechanism is specially useful for detecting disfluencies (Tran et al., 2018).

In general, parsing models are poor at detecting disfluencies, mainly due to "rough copy" dependencies in disfluent sentences, which are difficult for conventional parsers to detect.

2.2 Specialized Disfluency Detection Models

Disfluency detection models often use a sequence tagging technique to assign a single label to each

word of a sequence. Previous work shows that LSTMs and CNNs operating on words alone are poor at disfluency detection (Zayats et al., 2016; Wang et al., 2016; Jamshid Lou et al., 2018). The performance of state-of-the-art disfluency detection models depends heavily on hand-crafted pattern match features, which are specifically designed to find "rough copies". One recent paper (Jamshid Lou et al., 2018) augments a CNN model with a new kind of layer called an *auto-correlational layer* to capture "rough copy" dependencies. The model compares the input vectors of words within a window to find identical or similar words. The addition of the auto-correlational layer to a "vanilla" CNN significantly improves the performance over the baseline CNN model. The results are competitive to models using complex hand-crafted features or external information sources, indicating that the auto-correlation model learns "rough copies".

One recent paper (Wang et al., 2018) introduces a semi-supervised approach to disfluency detection. Their self-attentive model is the current state-of-the-art result in disfluency detection. The common factor in Wang et al. (2018) and the approach presented here is the self-attentive transformer architecture, which suggests that this architecture is capable of detecting disfluencies with very high accuracy. The work we present goes beyond the work of Wang et al. (2018) in also studying the impact of jointly predicting syntactic structure and disfluencies (so it can be understood as a kind of multi-task learning). We also investigate the impact of different ways of representing disfluency information in the context of a syntactic parsing task.

3 Neural Constituency Parser

We use the self-attentive constituency parser introduced by Kitaev and Klein (2018) and train it on the Switchboard corpus of transcribed speech (we describe the training and evaluation conditions in more detail in Section 4). The self-attentive parser achieves state-of-the-art performance on WSJ data, which is why we selected it as the best "off-the-shelf" parsing model. The constituency parser uses a self-attentive transformer (Vaswani et al., 2017) as an encoder and a chart-based parser (Stern et al., 2017) as a decoder, as reviewed in the following sections.

3.1 Self-Attentive Encoder

The encoder of a transformer is a stack of n identical layers, each consists of two stacked sublayers: a multi-head attention mechanism, and a point-wise fully connected network. The inputs to the encoder first flow through a self-attention sublayer, which helps the encoder attends to several words in the sentence as it encodes a specific word. Because the model lacks recurrent layers, this sublayer is the only mechanism which propagates information between positions in the sentence. The self-attention maps the input to three vectors called query, key and value and defines an attention function as mapping a query and a set of key-value pairs to an output vector. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. Each self-attention sublayer has several attention heads, where each head has its own query, key and value weight matrices. The multi-head attention allows the model to jointly attend to information from several different positions. The outputs of the self-attention layer are fed to a feed-forward neural network, which is applied to each position independently. For further detail, see Vaswani et al. (2017).

We believe that the self-attention mechanism is especially useful for detecting disfluencies in a sentence. In pilot experiments we found that similar LSTM-based parsers, such as the AllenNLP parser (Gardner et al., 2018), were much worse at disfluency detection than the self-attentive parser. As shown in Figure 1, the "rough copy" similarity between the repair and the reparandum is a strong indicator of disfluency. "Rough copies" involve same or very similar words in roughly same word order; for example, in the Switchboard training data, over 60% of the words in the reparandum are exact copies of the words in the repair. Using the multi-head self-attention mechanism the model can presumably learn to focus on "rough copies" when detecting a reparandum.

3.2 Tree Score and Chart Parse Decoder

A chart-based parser scores a tree as a sum of potentials on its labeled constituent spans as follows:

$$s(T) = \sum_{(i,j,l) \in T} s(i, j, l) \qquad (2)$$

where $s(i, j, l)$ is a score of a constituent located between string positions i and j with label l. At test time, a modified CYK algorithm is used to find the highest scoring parse tree for a given sentence.

$$\hat{T} = \underset{T}{\operatorname{argmax}}\, s(T) \tag{3}$$

Given the gold tagged tree $T^\star$, we train the model by minimizing a hinge loss:

$$max\left(0, \max_{T \neq T^\star}[s(T) + \triangle(T, T^\star)] - s(T^\star)\right) \tag{4}$$

where $\triangle$ is the Hamming loss on labeled spans. For further detail, see Kitaev and Klein (2018) and Stern et al. (2017).

3.3 External Embedding and Edited Loss

Peters et al. (2018) have recently introduced a new approach for word representation called Embeddings from Language Models (ELMo) which has achieved state-of-the-art results in various NLP tasks. These embeddings are produced by a LSTM language model (LM) which inputs words and characters and generates a vector representation for each word of the sentence. The ELMo output is a concatenation of both the forward and backward LM hidden states. We found that using external ELMo embedding as the only lexical representation used by the model leads to the highest EDITED word f-score. Following Kitaev and Klein (2018), we use a trainable weight matrix to project the ELMo pretrained weights of 1024 dimension to a 512-dimensional content representation. We tried different combinations of input including predicted POS tags, character LSTM and word embeddings with ELMo, but the result was either worse or not significantly better than when using ELMo alone.

The sole change we made to the self-attentive parser was to modify the loss function, so it puts more weight onto EDITED nodes. We show below that this improves the model's ability to recover EDITED nodes. We modify the tree scoring in 2 as follows:

$$s(T) = \sum_{(i,j,l) \in T} w_l \; s(i, j, l) \tag{5}$$

where w_l depends on the label l. We only used two different values of w_l here, one for EDITED nodes and one for all other node labels. We treat these as hyperparameters, and tune them to maximize EDITED nodes f-score (this is F(S_E) in Section 4.1 below).

4 Experiments

We evaluate the self-attentive parser on the Penn Treebank-3 Switchboard corpus (Mitchell et al., 1999). Following Charniak and Johnson (2001), we split the Switchboard corpus into training, dev and test sets as follows: training data consists of the sw[23]∗.mrg files, dev data consists of the sw4[5-9]∗.mrg files and test data consists of the sw4[0-1]∗.mrg files. Except as explicitly noted below, we remove all partial words (words tagged XX and words ending in "-") and punctuation from data, as they are not available in realistic ASR applications (Johnson and Charniak, 2004).

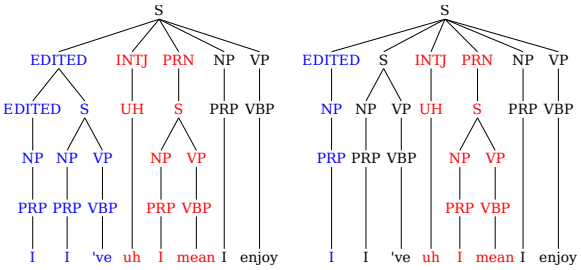
4.1 Evaluation Metrics

We evaluate the self-attentive parser in terms of parsing accuracy and disfluency detection performance. We report *precision* (P), *recall* (R) and *f-score* (F) for both *constituent spans* (S) and *word positions* (W), treating each word position as labeled by all the constituents that contain that word. We also consider subsets of constituent spans and word positions; specifically: (i) S_E, the set of constituent spans labeled EDITED, (ii) W_E, the set of word positions dominated by one or more EDITED nodes, and (iii) W_{EIP}, the set of word positions dominated by one or more EDITED, INTJ or PRN nodes.

We demonstrate the evaluation metrics with an example here. Consider the gold and predicted parse trees illustrated in Figure 2. The constituency trees are viewed as a set of labeled spans over the words of the sentence, where constituent spans are pairs of string positions. As explained earlier, we ignore punctuation and partial words when calculating evaluation scores. To calculate fscore for a span, i.e., F(S), the gold, predicted and correct labeled spans are counted. In this case, the number of predicted, gold and correctly predicted spans is 13, 14 and 12.

Since a parse tree with EDITED nodes identifies certain words as EDITED, we can evaluate how accurately a parser classifies words as EDITED (i.e. F(W_E)). Continuing with the example in Figure 2, the number of predicted, gold and correctly predicted EDITED words is 1, 3 and 1.

Similarly, we can also measure how well the parser can identify all disfluency words, i.e., the words dominated by EDITED, INTJ or PRN nodes. Continuing with the example in Figure 2, the number of predicted, gold and correctly pre-



(a) Gold tree (b) Predicted tree

$_0$ *I* $_1$ *I* $_2$ *'ve* $_3$ *uh* $_4$ *I* $_5$ *mean* $_6$ *I* $_7$ *enjoy* $_8$

Figure 2: An example gold and predicted parse tree.

dicted EDITED, INTJ and PRN words is 4, 6, 4.

4.2 Model Training

We use randomized search (Bergstra and Bengio, 2012) to tune the optimization and architecture parameters of the model on the dev set. We optimize the model for its performance on parsing EDITED nodes $F(S_E)$. The hyperparameters include dimensionality of the model, learning rate, edited loss weight, dropout, number of layers and heads as shown in Table 1. All other hyperparameters not mentioned here are the same as in Kitaev and Klein (2018).

Configuration	Parser
hidden label dim	340
model dim	2048
non-EDITED label weight	0.7
EDITED label weight	2
learning rate	0.0006
learning rate warmup steps	110
step decay factor	0.52
num heads	7
num layers	4
attention dropout	0.27
relu dropout	0.09
residual dropout	0.26
elmo dropout	0.57
tag embedding dropout	0.35
word embedding dropout	0.2

Table 1: Hyperparameter setting for the self-attentive constituency parser.

4.3 Edited Loss

Our best dev model (see Table 1) uses an edited loss that puts more weight on EDITED nodes and less weight on non-EDITED nodes. To explore the effect of edited loss, we retrained the best model with an equally weighted loss. The results in Table 2 indicate that differential weighting improves parsing EDITED nodes as well as EDITED word detection. It also rebalances the precision vs. recall trade-off and slightly increases overall parsing accuracy F(S).

	equal weight	different weight
$P(S_E)$	83.0	83.3
$R(S_E)$	91.6	91.4
$F(S_E)$	87.1	87.2
F(S)	92.8	93
$F(W_E)$	86.9	87.5

Table 2: Parsing precision $P(S_E)$, recall $R(S_E)$ and f-score $F(S_E)$ of EDITED nodes, parsing f-score F(S) and EDITED word f-score $F(W_E)$ on the Switchboard dev set for the equally and differentially weighted loss.

4.4 Modifying the Training Data

We investigate the effect of modifying the training data on the performance of the parser.

4.4.1 Simplified Tree Structures

We use different tree-transformations to explore the effect of different amounts of and encodings of disfluencies and syntactic information on the performance of the model.

- **Baseline:** Parse trees as they appear in the Switchboard corpus. A sample is shown in Figure 3.

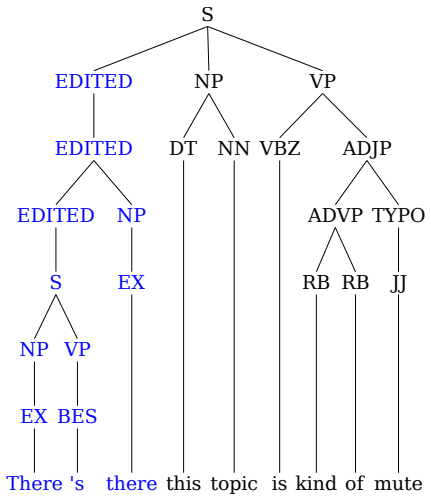


Figure 3: Baseline parse trees as they appear in the Switchboard corpus

- **Transformation PosDisfl:** Pushing disfluency nodes (i.e. EDITED, INTJ and PRN) down to POS tags, as shown in Figure 4.

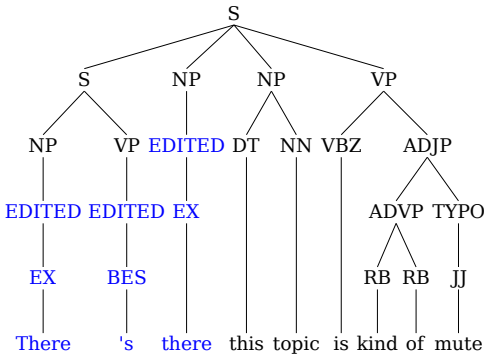


Figure 4: Transformation PosDisfl, where disfluency nodes are pushed down to POS tags.

- **Transformation NoSyntax:** Deleting all non-disfluency nodes, as shown in Figure 5.

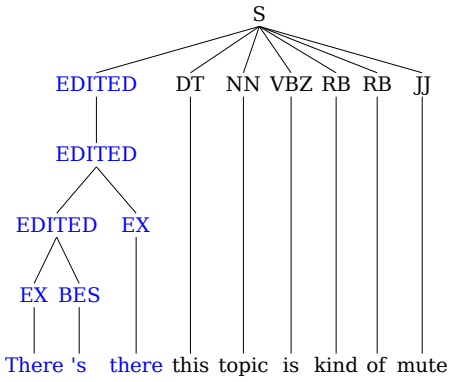


Figure 5: Transformation NoSyntax, where all non-disfluency nodes are deleted.

- **Transformation PosDisfl+NoSyntax:** Pushing disfluency nodes down to POS tags and deleting all non-disfluency nodes, as shown in Figure 6.

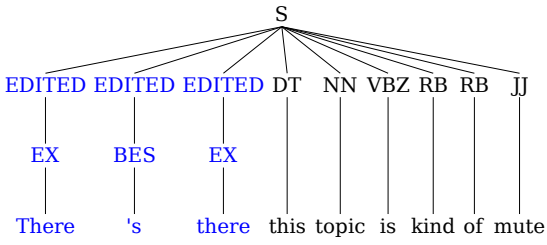


Figure 6: Transformation PosDisfl+NoSyntax, where disfluency nodes are pushed down to POS tags and all non-disfluency nodes are deleted.

- **Transformation TopDisfl:** Deleting all disfluency nodes but the top ones, as shown in Figure 7.

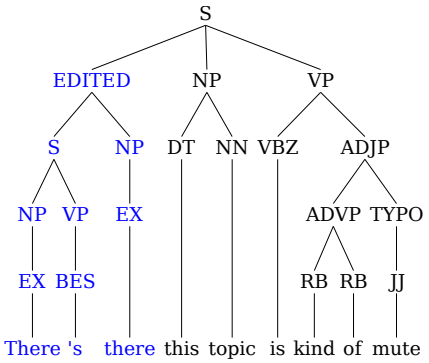


Figure 7: Transformation TopDisfl, where all disfluency nodes but the top ones are deleted.

- **Transformation TopDisfl+NoSyntax:** Deleting all nodes but the top-most disfluency nodes, as shown in Figure 8.

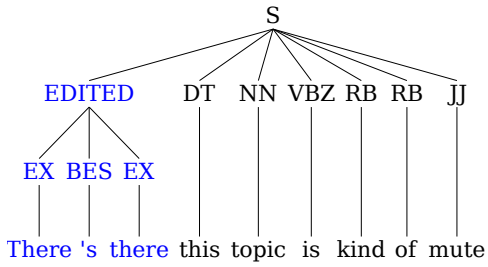


Figure 8: Transformation TopDisfl+NoSyntax, where all nodes but the top-most disfluency nodes are deleted.

We report the performance of the self-attentive parser in terms of EDITED word f-score and disfluency word f-score in Table 3. Since the transformations change the tree shapes, it is not meaningful to compare their parsing f-scores. As illustrated in Table 3, pushing disfluency nodes down to POS tags (i.e. *Transformation PosDisfl*) increases precision about 2%, resulting in 1% improvement in word f-score F(W_E). It also improves F(W_{EIP}) by 0.4%. In general, the model can take advantage of the simplified encoding of disfluency nodes (see *Transformations PosDisfl* and *TopDisfl*). Moreover, deleting all but the top-most disfluency nodes as in *Transformation TopDisfl+NoSyntax* significantly drops precision (about 20%), resulting in more than 13% decrease in EDITED word f-score. It also hurts detecting all types of disfluency (more than 7% decrease in

$F(W_{EIP})$). In general, removing syntactic structure dramatically degrades the performance of the model in terms of $F(W_E)$ and $F(W_{EIP})$, as shown in *Transformations NoSyntax*, *PosDisfl+NoSyntax* and *TopDisfl+NoSyntax*. This indicates that syntactic information is important for detecting disfluencies.

Setting	$P(W_E)$	$R(W_E)$	$F(W_E)$	$F(W_{EIP})$
Baseline	81.6	94.2	87.5	94.0
PosDisfl	83.7	94.2	**88.7**	**94.4**
NoSyntax	73.0	95.0	82.5	92.3
PosDisfl+NoSyntax	73.4	93.4	82.2	91.6
TopDisfl	81.9	94.3	87.7	93.8
TopDisfl+NoSyntax	61.3	93.1	74.0	86.7

Table 3: EDITED word precision $P(W_E)$, recall $R(W_E)$ and f-score $F(W_E)$ as well as EDITED, INTJ and PRN word f-score $F(W_{EIP})$ on the Switchboard dev set for different encodings of disfluency nodes in data. The best f-scores are shown in bold.

4.4.2 Punctuation and Partial words

As mentioned before, speech recognition models generally do not produce punctuation and partial words in their outputs. Thus, prior work has removed them from the data to make the evaluation more realistic. However, it is interesting to see what information partial words and punctuation convey about syntactic structure in general and disfluencies in particular, so we did an experiment to investigate the effect of including these in the training and test data. We use the best hyperparameter configuration on the Switchboard dev set and retrain the model on two versions of the data: (i) with partial words and (ii) with punctuation and partial words. As shown in Table 4, keeping punctuation and partial words in the training data increases EDITED word f-score by about 4%, indicating that punctuation and partial words greatly help disfluency detection. Punctuation leads to more gain in disfluency detection than partial words. Punctuation also improves the word f-score for all types of disfluencies by more than 1%.

5 Results

We selected our best model based on the dev set results (including differentially weighted loss) and compared the results achieved for the *Tree Transformation PosDisfl* and *No Tree Transformation* on the test set with previous work. Although most previous work has used the Switchboard corpus, it is sometimes difficult to compare systems directly due to different scoring metrics and differences in experimental setup, such as the use of partial words, punctuation, prosodic cues and so on. Since some studies report their results using partial words and/or punctuation, we divide prior work according to the setting they used and report the results of the self-attentive parser on the test data for each setting.

Setting	$F(W_E)$	$F(W_{EIP})$
without punctuation & partial words	88.7	94.4
with partial words	89.7	94.4
with punctuations & partial words	92.2	95.5

Table 4: EDITED word $F(W_E)$ and EDITED, INTJ and PRN word f-score $F(W_{EIP})$ on the Switchboard dev set for three versions of the training data.

Table 5 shows the test performance of the self-attentive constituency parser against previous parsing models of speech transcripts. The self-attentive parser outperforms all previous models in parsing accuracy. It has also better performance than Kahn et al. (2005) and Tran et al. (2018), who used acoustic/prosodic cues from speech waveform as well as the words in the transcript.

Parsing Model	F(S)
without partial words	
self-attentive parser (PDT)	92.4
self-attentive parser (NT)	**92.7**
partial words:	
Hale et al. (2006)	71.1
Kahn et al. (2005)	86.6
Tran et al. (2018)	88.5
self-attentive parser (NT)	92.3
self-attentive parser (PDT)	**92.6**

Table 5: Parse f-score F(S) for all constituent spans on the Switchboard test set with and without partial words. NT = No Transformation and PDT = PosDisfl Transformation.

We also compare the performance of the self-attentive parser with state-of-the-art disfluency detection methods in terms of EDITED word f-score. As shown in Table 6, the self-attentive parser (with *PosDisfl Transformation*) achieves a new state-of-

the-art for detecting EDITED words. Its performance is competitive with specialized disfluency detection models that directly optimize for disfluency detection. Using partial words increases edited word f-score for *No Transformation* mode by 0.1% and for *PosDisfl Transformation* mode by 0.6%, which is not surprising as the presence of partial words is strongly correlated with the presence of a disfluency.

It is interesting to compare the self-attentive parser with the ACNN model presented in Jamshid Lou et al. (2018). They introduce a new ACNN layer which is able to learn the "rough copy" dependencies between words, for which previous models heavily relied on hand-crafted pattern-matching features. "Rough copies" are a strong indicator of disfluencies that can help the model detect reparanda (i.e. EDITED nodes). That the self-attentive parser is better than the ACNN model (Jamshid Lou et al., 2018) in detecting disfluencies may indicate that the self-attention mechanism can learn "rough copy" dependencies.

Model	**F(W_E)**
without partial words:	
Honnibal and Johnson (2014)	84.1
Jamshid Lou et al. (2018) •	84.5
Wu et al. (2015)	85.1
Ferguson et al. (2015) •	85.4
Wang et al. (2016) •	86.7
Jamshid Lou and Johnson (2017) •	86.8
self-attentive parser (NT)	86.9
Wang et al. (2017) •	87.5
self-attentive parser (PDT)	**88.1**
partial words:	
Hale et al. (2006)	41.7
Tran et al. (2018)	77.5
Kahn et al. (2005)	78.2
Rasooli and Tetreault (2013)	81.4
Zayats et al. (2016) •	85.9
self-attentive parser (NT)	87.0
self-attentive parser (PDT)	**88.7**

Table 6: Edited word f-score F(W_E) on the Switchboard test set with and without partial words. • Specialized disfluency detection models. NT = No Transformation and PDT = PosDisfl Transformation.

We also compare the performance of the self-attentive parser with Wang et al.'s (2018) self-attentive disfluency detection model in terms of disfluency (i.e. EDITED, INTJ and PRN) word f-score. As shown in Table 7, the self-attentive parser outperforms this state-of-the-art specialized self-attentive disfluency detection model.

Self-attentive Model	**F(W_{EIP})**
punctuation & partial words	
Wang et al. (2018)	91.1
self-attentive parser (NT)	93.7
self-attentive parser (PDT)	**94.0**

Table 7: EDITED, INTJ and PRN word f-score F(W_{EIP}) on the Switchboard test set with punctuation and partial words. NT = No Transformation and PDT = PosDisfl Transformation.

6 Conclusion and Future Work

This paper shows that using an "off-the-shelf" constituency parser achieves a new state-of-the-art in parsing transcribed speech. The self-attentive parser is effective in detecting disfluent words as it outperforms specialized disfluency detection models, suggesting that it is feasible to use standard neural architectures to perform disfluency detection as part of some other task, rather than requiring a separate disfluency detection pre-processing step. We also show that removing syntactic information hurts word f-score. That is, performing syntactic parsing and disfluency detection as a multi-task training objective yields higher disfluency detection accuracy than performing disfluency detection in isolation. Modifying encoding by indicating disfluencies at the word level leads to further improvements in disfluency detection.

In future work we hope to integrate syntactic parsing more closely with automatic speech recognition. A first step is to develop parsing models that parse ASR output, rather than speech transcripts. It may also be possible to more directly integrate an attention-based syntactic parser with a speech recogniser, perhaps trained in an end-to-end fashion.

Acknowledgments

This research was supported by a Google award through the Natural Language Understanding Focused Program, CRP 8201800363 from Data61/CSIRO, and under the Australian Research Councils Discovery Projects funding scheme (project number DP160102156). We also

thank the anonymous reviewers for their valuable comments that helped to improve the paper.

References

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305.

Eugene Charniak and Mark Johnson. 2001. Edit detection and parsing for transcribed speech. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics on Language Technologies (NAACL)*, pages 118–126, Stroudsburg, USA.

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

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia. Association for Computational Linguistics.

John Godfrey and Edward Holliman. 1993. Switchboard-1 release 2 LDC97S62. Published by: Linguistic Data Consortium, Philadelphia, USA.

John Hale, Izhak Shafran, Lisa Yung, Bonnie J. Dorr, Mary Harper, Anna Krasnyanskaya, Matthew Lease, Yang (2) Liu, Brian Roark, Matthew Snover, and Robin Stewart. 2006. Pcfgs with syntactic and prosodic indicators of speech repairs. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 161–168, Sydney, Australia. Association for Computational Linguistics.

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

Paria Jamshid Lou, Peter Anderson, and Mark Johnson. 2018. Disfluency detection using auto-correlational neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4610–4619, Brussels, Belgium.

Paria Jamshid Lou and Mark Johnson. 2017. Disfluency detection using a noisy channel model and a deep neural language model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 547–553, Vancouver, Canada.

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 (ACL'04)*, pages 33–39, Barcelona, Spain.

Jeremy Kahn, Matthew Lease, Eugene Charniak, Mark Johnson, and Mari Ostendorf. 2005. Effective use of prosody in parsing conversational speech. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT'05)*, pages 233–240, Tallinn, Estonia.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

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*, NAACL-Short '06, pages 73–76, Stroudsburg, PA, USA. Association for Computational Linguistics.

Marcus Mitchell, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. 1999. Treebank-3 LDC99T42. Published by: Linguistic Data Consortium, Philadelphia, USA.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Mohammad Sadegh Rasooli and Joel Tetreault. 2013. Joint parsing and disfluency detection in linear time. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 124–129, Seattle, USA.

Elizabeth Shriberg. 1994. *Preliminaries to a theory of speech disfluencies*. Ph.D. thesis, University of California, Berkeley, USA.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827. Association for Computational Linguistics.

Trang Tran, Shubham Toshniwal, Mohit Bansal, Kevin Gimpel, Karen Livescu, and Mari Ostendorf. 2018. Parsing speech: A neural approach to integrating lexical and acoustic-prosodic information. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 69–81, New Orleans, USA.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, Long Beach, USA.

Feng Wang, Wei Chen, Zhen Yang, Qianqian Dong, Shuang Xu, and Bo Xu. 2018. Semi-supervised disfluency detection. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 3529–3538, Santa Fe, USA.

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

Shaolei Wang, Wanxiang Che, Yue Zhang, Meishan Zhang, and Ting Liu. 2017. Transition-based disfluency detection using LSTMs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2785–2794, Copenhagen, Denmark.

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.

Masashi Yoshikawa, Hiroyuki Shindo, and Yuji Matsumoto. 2016. Joint transition-based dependency parsing and disfluency detection for automatic speech recognition texts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1036–1041.

Victoria Zayats, Mari Ostendorf, and Hannaneh Hajishirzi. 2016. Disfluency detection using a bidirectional LSTM. In *Proceedings of the 16th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 2523–2527, San Francisco, USA.