Incremental Parsing of Disfluent, Unsegmented Speech Transcripts

Anonymous
Department
Institution
Address
email

Abstract

Conversational speech turns are often long, motivating the use of a segmentation model to pre-process the input into shorter units. We instead propose a transition-based model that processes entire turns, jointly performing dependency parsing and disfluency detection. Segment boundaries can then be recovered from the syntactic structure if they are required.

We compare the system against a pipeline approach, where the input is segmented with a CRF model before parsing. We find that the joint model achieves a 0.3% improvement in disfluency detection, a 0.6% improvement in segmentation, and a 2.0% improvement in parse accuracy, setting a new state-of-the-art of 87.9% UAS.

1 Introduction

Previous speech understanding systems have required that the input be pre-segmented into sentence-like units. However, sentence boundaries are not always clear in the speech input, which lacks case distinctions and punctuation, and is not reliably segmented by pauses. For example, the following turn from the Switchboard conversational speech transcripts is segmented into three utterances, at the slashes:

(1) uh and really we were really forced into keeping a budget because i ’m i ’m paid once a month which sort of sort of forces some uh uh restrictions /
and you need to make sure all your bills are paid /
uh about yourself

An annotator could reasonably omit the first boundary, analysing the first two units as a single conjoined sentence. We suspect this arbitrariness in sentence boundary location might be problematic for syntactic parsing, since the boundaries could accidentally constrain the syntactic analyses in ways that are inconsistent with the parser’s expectations. This motivated us to explore approaches where the syntactic parser itself is responsible for identifying the locations of sentence boundaries. Our approach here is to associate entire discourse turns with a single syntactic structure that consists of a sequence of sentences. While the length of these turns makes conventional $O(n^3)$ parsing algorithms impractical, incremental transition-based parsers now achieve high accuracy, while running in time linear in the length of the input string. Our parsing model is based on the system described by Honnibal and Johnson (2014), who showed that a joint model of disfluency detection and parsing produced state-of-the-art accuracy on both tasks, when gold-standard segment boundaries are available.

We tried two ways of parsing unsegmented inputs. Our first approach introduces a new transition, Break, to explicitly predict segment boundaries during parsing. The second approach leaves the segmentation implicit until parsing is complete, at which point the boundaries can be read-off from the predicted parse.

We find that both ways of parsing unsegmented input result in superior accuracy to a pipeline approach, where the input is pre-segmented by a CRF model. The joint models prove advantageous at all three evaluations: segmentation, disfluency detection and parsing. The improvement is particularly substantial for parsing, where the pipeline approach results in 85.9% UAS, while the best joint model achieves 87.9%, a new state-of-the-art. Our model is also highly efficient, processing approximately 900 words per second.

A flight to $\underbrace{\text{um}}_{\text{FP}}\ \underbrace{\text{Boston}}_{\text{RM}}\ \underbrace{\text{I mean}}_{\text{IM}}\ \underbrace{\text{Denver}}_{\text{RP}}$ Tuesday

Figure 1: A sentence with disfluencies annotated in the style of Shriberg (1994) and the Switchboard corpus. FP=Filled Pause, RM=Reparandum, IM=Interregnum, RP=Repair.

Disfluency type	Example	Freq.
Speech-repair	*to* **Boston** *uh Denver*	32,310
Filled pauses	**um**, **uh**	20,502
Edit terms	**I mean**	3,447
Discourse	**well**, **you know**	21,412
Segment Conjunctions	**and**, **and so**	25,624

Table 1: Frequencies of different disfluency types in Sections 2 and 3 of the Switchboard MRG files.

2 Spoken Language Understanding

A verbatim, unpunctuated speech transcript has very different linguistic characteristics from well-edited written text, even in the absence of speech-recognition errors. Speech transcripts pose two challenges for natural language understanding technologies in particular. The **segmentation** problem arises because the speech stream is continuous, and pauses are less syntactically informative than careful punctuation (Gregory et al., 2004). The **disfluency** problem arises due to language performance problems — speakers *um* and *uh*, edit their utterances on the fly, and frequently insert parentheticals.

Well-edited written text can be segmented into sentences easily, using punctuation and capitalisation. Sentence boundary detection systems typically achieve accuracies above 99% on clean text input, while state-of-the-art speech segmentation systems achieve only 96-97% accuracy (i.e. a 3-times higher error-rate). Segmentation is typically a *pre*-process for parsing, limiting segmentation systems to ngram-based features, and acoustic cues — which have proven difficult to utilise (Liu et al., 2005). These local features cannot accurately identify segment boundaries, if segments are identified with tensed clauses, since they can contain multiple long-range dependencies.

As well as being continuous, unscripted speech is frequently disfluent. Figure 1 shows part of a disfluent utterance, annotated according to Shriberg (1994). Speech repairs are particularly problematic for syntactic parsers, because of the complicated dependency structures that can arise between the reparandum, repair and the fluent sentence (Johnson and Charniak, 2004).

Length	Segmented	Unsegmented
1-2	26,140	10,225
2-5	15,727	6,914
5-10	23,283	6,885
10-20	19,738	8,934
20-50	6,406	9,765
50-100	161	2,106
100-200	0	257
200-422	0	20
Total	91,455	45,106

Table 2: Input lengths in the Switchboard training corpus, with and without gold-standard segmentation. For instance, with utterances pre-segmented, there are 161 sentences with between 51 and 100 tokens; in the unsegmented corpus, there are 2,106 sentences with lengths in that range.

2.1 The Switchboard Corpus

The Switchboard portion of the Penn Treebank (Marcus et al., 1993) contains 1,126 transcripts of telephone calls between strangers on an assigned topic. Every file has been annotated for speech-repairs and other disfluencies (filled pauses, parentheticals, discourse markers, etc); these annotations are provided in the DPS files. Syntactic brackets (MRG files) are available for 619,236 of the 1,482,845 words in the training sections of the corpus (2 and 3). All of the transcripts have also been annotated for various speech *metadata*, including utterance segmentation, speech repairs per Shriberg (1994), and non-repair disfluencies, such as filled pauses. Table 1 shows the frequency of these disfluencies in the training corpus.

Table 2 shows how segmentation affects the length of training inputs. Without any utterance segmentation, the inputs consist of whole turns, using the gold-standard diarisation in the MRG files. Segmentation doubles the number of segments (and so halves their length, on average); i.e., on average each turn contains one sentence-medial segment boundary.

We follow previous work on spoken language understanding by lower-casing the text and removing punctuation and partial words (words tagged XX and words ending in '-'). However, we depart from Honnibal and Johnson (2014) in not removing one-token sentences, not removing filled pauses as a pre-process, and not re-tokenising the common parentheticals *you know* and *i mean*. We avoid these extra pre-processing steps in favour of extended disfluency processing, by subtyping the Edit transition with different disfluency labels, as described in Section 3.3.

$(\sigma, i\|\beta, \mathbf{A}, \mathbf{L}) \vdash (\sigma\|i, \beta, \mathbf{A}, \mathbf{L})$	S
$(\sigma\|i\|j, \beta, \mathbf{A}, \mathbf{L}) \vdash (\sigma\|i, \beta, \mathbf{A}(j) = i, \mathbf{L}(j) = \ell)$	R$_\ell$
$(\sigma\|i, j\|\beta, \mathbf{A}, \mathbf{L}) \vdash (\sigma, j\|\beta, \mathbf{A}(i) = j, \mathbf{L}(i) = \ell)$	L$_\ell$
$(\sigma\|i, \beta, \mathbf{A}, \mathbf{L}) \vdash (\sigma\|x_1\|...\|x_n, \beta, \mathbf{A}(\gamma) = \gamma, \mathbf{L}(\gamma) = \ell)$	E$_\ell$
Where	
$x_1...x_n$ are the former left children of i	
γ is i and its rightward subtree	

Figure 2: Transition system for the parser, with σ denoting the stack, β denoting the buffer, $\mathbf{A}$ denoting a vector of head indices, and $\mathbf{L}$ a vector of arc labels. The transitions are the arc-hybrid **S**hift, **R**ight and **L**eft, and the Honnibal and Johnson (2014) **E**dit. The R, L and E transitions are parameterised by label, ℓ.

3 Joint Disfluency Detection and Parsing

Our model is based on the Honnibal and Johnson (2014) joint incremental disfluency detection and parsing system. This section provides a brief description of the model, and highlights our departures from it. Our novel contributions are described from Section 4 onwards.

3.1 Transition-based Dependency Parsing

We follow recent work on speech parsing in using an incremental, transition-based dependency parser (Rasooli and Tetreault, 2013; Honnibal and Johnson, 2014). A transition-based parser (Nivre, 2003) consists of a configuration, and a set of actions (or 'transition system'). Actions are chosen from the transition-system and applied to the state, until a terminal configuration is reached.

A configuration $c = (\sigma, \beta, \mathbf{A}, \mathbf{L})$, where σ and β are disjoint sets of word indices termed the *stack* and *buffer* respectively, $\mathbf{A}$ is a vector of head indices, and $\mathbf{L}$ is a vector of dependency labels. A dependency arc from a head h to a dependent d with label ℓ is represented $\mathbf{A}(d) = h$, $\mathbf{L}(d) = \ell$. A word d is marked disfluent by setting its head to itself, i.e. $\mathbf{A}(d) = d$. Labels are used to distinguish the different types of disfluencies, e.g. filled pauses, speech repairs, etc. A vertical bar is used to denote concatenation to the stack or buffer, e.g. $\sigma|i$ indicates a stack with the topmost element i and remaining elements σ.

3.2 Arc-Hybrid Transition System

We depart from Honnibal and Johnson (2014) in using the arc-hybrid system (Kuhlmann et al., 2011), instead of the arc-eager system (Nivre, 2003). The two transition systems achieve comparable accuracy (Goldberg and Nivre, 2013), but we find the arc hybrid system slightly simpler.

The arc-hybrid system, shown in Figure 2, defines the Left-Arc in the same way as the Nivre (2003) arc-eager system, but the Right-Arc creates an arc between the top two words of the stack, following the arc-standard definition.

Unlike the arc-eager system, arcs are only created when a word is *popped*. This means that there are never arcs to words on the stack. The stack, buffer, and the words that have been assigned heads are three disjoint sets.

The arc-hybrid system maintains the simplicity advantages of arc-standard, which have motivated Huang and Sagae (2010) and others to continue working with it; but allows training oracles to be defined easily, due to the *arc decomposable* property that Goldberg and Nivre (2013) show it shares with the arc-eager system.

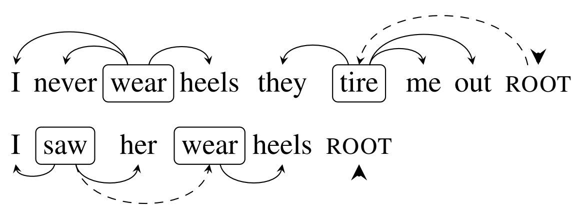
3.3 The Edit Transition

We employ the Edit transition defined by Honnibal and Johnson (2014), to handle speech repairs. The Edit transition marks the word i on top of the stack $\sigma|i$ as disfluent, along with its rightward descendents — i.e., all words in the sequence $i...j-1$, where j is the leftmost edge of the word at the start of the buffer. It then restores the words both preceding and formerly governed by i to the stack.

Honnibal and Johnson (2014) only apply the Edit transition to speech-repair disfluencies. They pre-process the input to remove *uh* and *um* tokens, and merge the common parenthetical *you know* into a single token. We instead extend the Edit transition to the other disfluency types described in Table 1, with the exception of segment conjunctions, which are simply conjunctions that occur at the beginning of a segment.

3.4 Training and Decoding

We follow Honnibal and Johnson (2014) in employing beam-search decoding, and use their training strategy: the Sun et al. (2009) latent-variable variant of the Collins (2002) structured perceptron, with weight updates calculated with the Huang et al. (2012) *maximum violation* strategy. We also employ the path-length normalisation technique that Honnibal and Johnson (2014) recommend, to deal with the variable-length transition histories the Edit transition may introduce: when calculating the figure-of-merit for the beam, we use the *mean transition score*, instead of the *total transition score*.



I never wear heels they tire me out ROOT
I saw her wear heels ROOT

Figure 3: Two parse states, showing a segmentation decision to be made using Strategy 1, implicit segmentation. Words remaining on the stack are circled, and current arcs are drawn with solid lines. The arrow indicates the start of the buffer. In the top state, the correct move is a Left-Arc as there is no dependency between *wear* and *tired*. In the lower state, the Right-Arc is correct. The dependencies that would be added by these moves are shown with dashed lines.

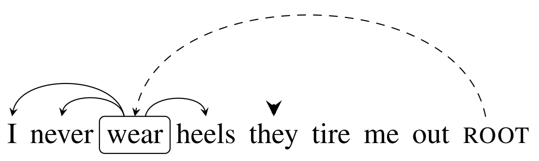


Figure 4: A parse state illustrating segmentation using Strategy 2, the Break transition. The governor of the previous segment, *wear*, is the only word on the stack (circled), and the first word of the next segment, *they*, is at the start of the buffer (indicated by an arrow). After the Break transition is applied, *wear* is popped from the stack, and the dashed arc is added from the ROOT symbol.

4 Joint Segmentation and Parsing

We now describe two ways of encoding utterance segmentation decisions into the parser's transition system. In the first strategy, the segmentation is determined from the parse structure; in the second strategy, a distinct transition, Break, is added to the transition system to insert segment boundaries. The two strategies are evaluated in Section 6. We find that despite being quite different, the two strategies achieve similar performance.

4.1 Strategy 1: Implicit segmentation

The first strategy we present uses the Left-Arc to attach each segment-governor to the ROOT symbol, which we place at the end of the input. This way of encoding the segmentation decisions has the governors accumulate on the stack, with the final decision to place segment boundaries between them only made when the buffer is exhausted. Because the arc-hybrid system is used, rather than the arc-eager system, the governor of a word is not determined when it is pushed onto the stack — only when it is popped. Thus, even when the buffer is exhausted, the parser may decide to create a dependency between two words, instead of using the Left-Arc to attach them to the ROOT symbol.

Figure 3 shows two similar parse states, where this decision arises. In the first state, the correct decision is **Left-Arc**, because a segment boundary should be inserted between *wear* and *tires*. In the second state, the correct decision is **Right-Arc**, because *wear* is an argument of *saw*. Such decisions may be difficult, so delaying them until the potential left and right heads can be compared in the same state may be advantageous.

4.2 Strategy 2: Explicit Break transition

The second strategy we present uses a specific Break transition. The design of the transition was inspired by the work of Zhang et al. (2013), who describe a joint transition-based model of dependency parsing and punctuation prediction. The task of predicting sentence-final punctuation, and our task of utterance segmentation, are closely related, so we experimented with their approach.

Zhang et al. (2013) train their model to insert sentence-final punctuation when the stack is fully connected and the first word of a new sentence is at the start of the buffer. Our Break transition operates in a similar fashion. The transition is applied when there is exactly one word on the stack, and the word at the start of the buffer has no leftward children. The word on the stack is arced to the ROOT symbol, and the stack is popped:

$$(i, j|\beta|n, \mathbf{A}, \mathbf{D}) \vdash (\emptyset, j|\beta|n, \mathbf{A}(i) = n, \mathbf{L}(i) = \text{R})$$
$$\text{Where } \{x, ..., j : \mathbf{A}(x) = j\} = \emptyset$$

The transition is designed to be applied early, instead of late: it should be applied when the first word of the new segment is at the start of the buffer. This is guaranteed by the pre-condition, which prevents it from being applied if the word at the start of the buffer has any leftward children.

Figure 4 shows a state at which the transition should be applied. The governor of the previous segment is the only word remaining on the stack, and the word at the start of the buffer has no children, as it is the start of a new segment. In the resulting state (shown below), the governor has been popped from the stack (it is no longer circled), and it has been arced to the ROOT symbol.

4.3 Training Oracle

We follow Honnibal and Johnson (2014) in training our model using the latent-variable structured perceptron algorithm (Sun et al., 2009). As each training example is received, we search for the highest-scoring transition sequence, and the highest-scoring *gold* transition sequence. The search for the best gold-standard sequence requires a training oracle, which maps a parse-state and a gold-standard derivation to a set of gold-standard actions. An action is considered gold-standard if it is a step towards the best possible continuation, i.e. the transition sequence that will yield the highest-scoring analysis reachable from the current configuration.

Goldberg and Nivre (2013) give a training oracle for the arc-hybrid system. The oracle amounts to the following rules, where β_0 refers to the first word of the buffer, σ_0 refers to the top word of the stack, and σ_1 refers to the second word on the stack. We denote the vector of gold-standard head indices $\mathbf{G}$, with $\mathbf{G}(d) = h$ asserting that an arc from h to d is in the gold-standard. The arc-hybrid training oracle consists of the following rules, which determine which actions (abbreviated S, R and L here) are gold-standard:

1. If the stack is empty, S is the only gold action;
2. If $\mathbf{G}(\beta_0) = \sigma_0$, S is a gold action;
3. If there are any other arcs between β_0 and the stack, S is not a gold-standard action;
4. If $\mathbf{G}(\sigma_0) = \sigma_1$, R is a gold action;
5. If $\mathbf{G}(\sigma_0) = \beta_0$, L is a gold action;
6. If there are any other arcs between σ_0 and the buffer, neither L nor R are gold actions.

The Honnibal and Johnson (2014) Edit transition adds the following rules to the oracle above. Recall that $\mathbf{G}(d) = d$ asserts that a word d is disfluent in the gold-standard:

1. If $\mathbf{G}(\sigma_0) = \sigma_0$, E is a gold action;
2. If $\mathbf{G}(\sigma_0) \neq \sigma_0$, E is not a gold action;
3. If $\mathbf{G}(\beta_0) = \beta_0$, both L and S are gold actions;
4. If $\mathbf{G}(\sigma_1) = \sigma_1$ and $\mathbf{G}(\sigma_0) = \sigma_0$, R is a gold action;
5. If $\mathbf{G}(\sigma_0) = \sigma_0$ but $\mathbf{G}(\beta_0) \neq \beta_0$, L is not a gold action;
6. If $\mathbf{G}(\sigma_0) = \sigma_0$ but $\mathbf{G}(\sigma_1) \neq \sigma_1$, R is not a gold action.

In the implicit segmentation strategy, the segment boundaries are inserted via the standard Left-Arc transition, so no adjustment to the training oracle is required. In the explicit strategy, the new Break transition is used to insert the segment boundaries.

The Break transition is gold-standard if and only if its pre-conditions are met, and the word on top of the stack and the word at the start of the buffer do not belong to the same segment. If this is the case, B is the only gold-standard action.

5 Experiments

We use the Switchboard portion of the Penn Treebank (Marcus et al., 1993), as described in Section 2.1, to train and evaluate our models. Unfortunately, this complicates comparison with most of the prior work on utterance segmentation, which used the RT'04 shared-task data distributed by DARPA. The RT'04 data covers part of the Switchboard corpus text, but was reanalysed for segmentation and disfluency detection, with slightly different annotation conventions. Previous work has found it difficult to reconcile the RT annotations with those provided by the Penn Treebank (Bies et al., 2006). We use the Switchboard corpus for consistency with previous work on disfluency detection and parsing (Qian and Liu, 2013; Rasooli and Tetreault, 2013; Honnibal and Johnson, 2014).

We follow the pre-processing and dependency conversion steps described in Section 2.1: the text was lower-cased, partial words were removed, and the phrase-structure trees were converted into projective-dependency parses using the Stanford Dependency Converter (de Marneffe et al., 2006). We use the standard train/dev/test split from Charniak and Johnson (2001). We follow Honnibal and Johnson (2014) in using the SPARSEVAL (Roark et al., 2006a) metric to evaluate our parser, which measures the dependency accuracy of words marked fluent in the gold-standard.

We follow Johnson and Charniak (2004) and others in restricting our disfluency evaluation to speech repairs, which we identify as words that have a node labelled EDITED as an ancestor in the Switchboard phrase-structure trees. We also follow them in training only on the MRG files, giving us 619,236 words of training data instead of the 1,482,845 used by other disfluency detection systems, such as Qian and Liu (2013).

We test for statistical significance in our results by training 20 models for each experimental configuration, using different random seeds. The random seeds control how the sentences are shuffled during training, which the perceptron model is quite sensitive to. We use the Wilcoxon rank-sums non-parametric test.

The main hyper-parameter of our model is the beam-width. Our beam is notably narrower than Honnibal and Johnson (2014), who employ a beam-width of 64. We found that only small accuracy improvements were obtained with beams wider than 8. We use a beam-width of 12.

5.1 Features

We base our feature set on the arc-hybrid features used by Goldberg and Nivre (2013), with the additional disfluency features used by Honnibal and Johnson (2014). Our specific feature templates are provided in the supplementary materials.

The templates refer to various combinations of the word-form, part-of-speech tag, and the Brown cluster (Brown et al., 1992) of various tokens in the context. The tokens are the top three words of the stack, the left and right subtree of the top word of the stack, the first three words of the buffer, and the left subtree of the first word of the buffer. Additionally, we follow Honnibal and Johnson (2014) in using the leftmost and rightmost edge of the top word of the stack and the first word of the buffer as contextual tokens. We used the Brown cluster mapping computed by Liang (2005). We follow Honnibal and Johnson (2014) in using 4- and 6-bit prefixes of the clusters in our feature templates, as initially suggested by Koo and Collins (2010).

5.2 Qian and Liu (2013) Disfluency Detector

Our parser performs disfluency detection jointly during parsing, using the Edit transition described by Honnibal and Johnson (2014). For comparison, we also trained and evaluated the disfluency detection system of Qian and Liu (2013) on both segmented and unsegmented input.

The Qian and Liu (2013) system uses a cascade of M3N sequence-tagging models. The first pass detects filler-words, the next pass uses the filler-word predictions as features, and two subsequent passes detect disfluencies. The system is a good comparison point because it achieves equivalent, state-of-the-art accuracy to the Honnibal and Johnson (2014) system, with a very different algorithm.

	Segmentation			
	Acc.	P	R	F
CRF	96.6	87.4	78.0	82.4
Joint (explicit)	96.9	84.0	85.9	84.9
Joint (implicit)	96.9	83.9	86.2	85.0

Table 3: Segmentation evaluation on the development set, for the CRF model and our joint approaches.

Because the system does not require syntactically-annotated training data, it can be trained from the DPS files, which make more training data available than the syntactically-annotated MRG files. Qian and Liu (2013) train and evaluate their model from the DPS annotations. However, to promote direct comparison against our syntactic disfluency detectors, we trained the system from the MRG files, and followed the syntactic disfluency annotations (i.e., words were marked disfluent if they were part of the yield of an EDITED node). Interestingly, this resulted in slightly higher accuracy than Qian and Liu (2013) report. It seems that the MRG annotated speech repairs are easier to detect than the DPS annotations.

5.3 CRF Segmenter

To evaluate our joint model, we prepared a sequence-based segmentation system, following the approach of Liu et al. (2005). We used the CRF implementation provided by the Wapiti toolkit (Lavergne et al., 2010). The segmentation problem was modelled as a binary classification task, with segment-initial tokens labelled 1 and all other tokens labelled 0. Features referred to the word, part-of-speech tag, prefix, and suffix of the target and surrounding tokens, as well as the previous two labelling decisions. Weights were learned using the L-BFGS algorithm, with an elastic-net penalty tuned on the development data. The pattern file, which describes the exact feature-templates, is attached in the supporting materials.

6 Development Results

6.1 Segmentation Accuracy

Table 3 compares the per-token segmentation accuracy of the three systems, along with their precision, recall and F-measure at identifying segment-initial words. Low recall indicates under-segmentation, while low precision indicates over-

System	Segmented *P*	Segmented *R*	Segmented *F*	Unsegmented *P*	Unsegmented *R*	Unsegmented *F*
Qian & Liu '13	90.6	80.7	85.3	83.8	76.5	80.0
Pipeline	92.4	76.8	84.1	81.9	73.6	77.6
Joint (explicit)		n/a		88.4	68.6	77.2
Joint (implicit)		n/a		88.3	68.9	77.5

Table 4: Disfluency detection evaluation on the development set, with and without gold segment boundaries.

System	UAS	LAS
Gold → Parser	90.9	88.0
CRF → Parser	86.2	83.4
Joint (explicit)	87.9	85.1
Joint (implicit)	88.1	85.3

Table 5: Unlabelled and labelled (parse) attachment scores on the development data.

segmentation. Our segmentation evaluation excludes disfluent words. When a disfluent word begins a segment, we move its segment label back to the first fluent word.

The two syntactic systems achieve equivalent accuracy, with similar precision/recall bias. Because the parser is incremental, it is easy to capture the information available to the CRF system, with the added advantage of long-range syntactic features. This gives the parsing models a small but statistically significant accuracy advantage.

6.2 Disfluency Detection Accuracy

Table 4 shows the accuracy of the systems at detecting speech-repair disfluencies. We first follow previous work in evaluating our model given gold-standard segment boundaries (**Segmented**). The **Pipeline** system is the parsing model described in Section 3, which is based on the Honnibal and Johnson (2014) model, but makes use of the arc-hybrid transition system, with a feature set adjusted accordingly. We also train the model to detect other disfluency types during parsing, by splitting the Edit transition with different labels.

While Honnibal and Johnson (2014) found that their system achieved slightly higher accuracy than Qian and Liu (2013), our model's accuracy is slightly lower. It may be that the Edit transition does not work as well with the arc-hybrid system as it does with the arc-eager system that Honnibal and Johnson employ. Alternatively, our feature set may be less well-tuned for disfluency detection.

When the gold-standard segment boundaries were replaced with predictions from the CRF model, precision fell substantially, with only a small decrease in recall. It seems that the parser dealt with segmentation errors by applying the Edit transition, since the input was ungrammatical. This suggests that there may be a way to train the model to adjust to segmentation mistakes, should a pipeline architecture prove desirable.

The two joint models achieved similar disfluency F-measure to the pipeline system, but with higher precision, and lower recall. The differences in F-measure were not statistically significant. We suspect that the syntactic disfluency models are sensitive to parse accuracy, which is reduced in the absence of gold-standard segment boundaries. However, the drop in performance of the Qian and Liu (2013) system on unsegmented input, from 85.3 to 80.0 F-measure suggests that the segment boundaries are also informative clues for disfluency detection in their own right.

6.3 Parsing Accuracy

Table 5 shows labelled (LAS) and unlabelled (UAS) dependency accuracies on the development data. With gold-standard segmentation, the system achieves 90.9% UAS, matching the state-of-the-art accuracy reported by Honnibal and Johnson (2014). When the same system was given input segmented by the CRF model described in Section 5.3, accuracy fell to 86.2%.

Both of the joint models, **Joint (explicit)** and **Joint (implicit)**, achieve substantially higher parse accuracies than the pipeline system. We attribute the pipeline system's loss of accuracy to error-propagation problems: segmentation errors mean that the parser will be supplied ungrammatical input, leading to low quality parses. The joint models allow the parser to segment the input while assigning the dependency parse, either by employing an explicit transition, or by simply attaching segment-governors to the root node. The two strategies yield similar accuracy, with a small but statistically significant advantage to the implicit segmentation.

7 Final Evaluation

Table 6 shows the final evaluation, using unsegmented inputs from the test set. We compare the implicit and explicit strategies for joint segmentation and parsing against the pipeline system on segmentation, speech-repair disfluency detection, and unlabelled parse accuracy. We also evaluate the state-of-the-art Qian and Liu (2013) disfluency detection system on unsegmented input, and eval-

System	Seg.	Disfl.	UAS	w/s
Qian & Liu '13	—	79.2	—	1,161
CRF → Parser	96.7	76.4	85.9	948
Joint (explicit)	97.3	76.9	87.8	890
Joint (implicit)	97.2	76.7	87.9	893

Table 6: Final evaluation scores, for segmentation, disfluency detection, parsing and efficiency, on unsegmented input.

uate the efficiency of the systems, measured in words per second.[1]

The two joint strategies performed similarly on all evaluations. The difference in segmentation accuracy was statistically significant, while the differences in parse accuracy (UAS) and disfluency detection F-measure were not. We conclude that the way in which the segmentation decisions are encoded into the transition system is relatively unimportant, so long as the decisions are made jointly during parsing.

The advantage of joint modelling over the pipeline approach (**CRF → Parser**) is clear, particularly on the parsing evaluation, where the **Joint (implicit)** model achieved 2.0% higher UAS.

Interestingly, the joint models' small advantage in segmentation accuracy over the CRF system widened on the test data: on the development data, the joint models scored 96.9%, while the CRF model scored 96.6%. On the test data, the best joint model segmented with 97.3% accuracy, while the CRF model scored 96.7%. Both differences were statistically significant.

The differences in disfluency detection F-measure between the three syntactic systems were not statistically significant. The state-of-the-art Qian and Liu (2013) system achieved significantly better accuracy. We attribute this to the loss of parse quality for the syntactic models on unsegmented input, which is linked to their disfluency detection accuracy due to their joint approach.

Finally, we note that there is little loss in efficiency from parsing the unsegmented input, and that the joint models are almost as efficient as the Qian and Liu (2013) sequence-tagging system, which only performs disfluency detection.

8 Related Work

Our model draws directly on the work of Honnibal and Johnson (2014), who showed that a joint transition-based model achieved superior results to a pipeline approach for disfluency detection and parsing. We describe a similar model, which we extend to utterance segmentation. They report 90.9% UAS and 85.8 disfluency detection F-measure on the development data, given gold-standard segmentation. Our model achieves 90.9% and 84.1 on these evaluations. We attribute the difference in disfluency detection to a lack of feature tuning in our model.

Rasooli and Tetreault (2013) and Rasooli and Tetreault (2014) also describe joint transition-based models of dependency parsing and disfluency detection. However, their system is limited to greedy search, and their disfluency detection transition operates slightly differently. The Rasooli and Tetreault (2014) system achieves 88.4% UAS and 82.6% disfluency F-measure, given gold-standard segmentation and POS tags. Although their system differs in several minor ways, it seems likely that the biggest factor in their lower accuracy is the lack of beam-search.

Zhang and Clark (2011) show that the generalised perceptron with beam-search architecture could be successfully applied to a range of tasks, including joint word segmentation and POS tagging for Chinese. Their results anticipate the recent interest in joint transtion-based dependency parsing models, such as the work of Zhang et al. (2013), who describe a joint transition-based model of dependency parsing and punctuation prediction. We based our Break transition, described in Section 4.2, on their work. Although our models are similar, there are no directly comparable results, as Zhang et al. (2013) did not investigate the effect of segmentation on parse accuracy, and did not apply their model to conversational speech.

The impact of utterance segmentation on parse quality was investigated by Kahn et al. (2004), in the context of a PCFG parsing model. They compared the effect on parse accuracy of three sentence segmentation systems: oracle segmentation, an HMM system, and naive pause-based segmentation. They showed that the HMM segmentation system achieved a 7% improvement in bracket precision and recall over the baseline segmenter, and scored 5% below the oracle segmenter.

A weakness of the pipeline architecture employed by Kahn et al. (2004) and others is error-propagation, which arises because the earlier components forward only a single hypothesis. One way of mitigating this problem is re-ranking, which Johnson and Charniak (2004); Johnson

[1]All systems were run on a 2.4GHz Intel Xeon, with a single thread.

et al. (2004) used to improve disfluency detection, and Roark et al. (2006b) used to improve utterance segmentation. The re-ranking architecture allows subsequent models to search some of the hypothesis space from earlier in the pipeline, i.e. a disfluency or segmentation analysis can be selected in light of the parse structure it permits. We instead adopt a fully joint approach, made tractable by recent advances in incremental parsing.

9 Conclusion

Segmentation and disfluency detection make speech parsing particularly difficult, relative to understanding written text. Both disfluency detection and segmentation require syntactic features, but standard polynomial-time parsing algorithms cannot be applied accurately before segmentation and disfluency detection have been conducted. We have shown that recent advances in transition-based parsing offer a solution to this chicken-and-egg problem. We model all three problems jointly, so that the combined problem-space can be searched for a good hypothesis. We demonstrate a 2% improvement in dependency parse accuracy over the pipeline approach. Our model currently makes use of no acoustic features, which would be interesting to explore for future work. It would also be interesting to apply the model to spoken language understanding problems, as it is efficient enough to operate in real-time, and it is the first model to achieve high parsing accuracies on unsegmented spoken language transcripts.

References

Ann Bies, Stephanie Strassel, Haejoong Lee, Kazuaki Maeda, Seth Kulick, Yang Liu, Mary Harper, and Matthew Lease. 2006. Linguistic resources for speech parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18:467–479.

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*, pages 118–126. The Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics.

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

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *TACL*, 1:403–414.

Michelle Gregory, Mark Johnson, and Eugene Charniak. 2004. Sentence-internal prosody does not help parsing the way punctuation does. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 81–88. Association for Computational Linguistics, Boston, Massachusetts, USA.

Matthew Honnibal and Mark Johnson. 2014. Joint incremental dependency parsing and disfluency detection. *TACL*, 1:403–414.

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*, pages 142–151. Association for Computational Linguistics, Montréal, Canada.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1077–1086.

Mark Johnson and Eugene Charniak. 2004. A TAG-based noisy channel model of speech repairs. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 33–39.

Mark Johnson, Eugene Charniak, and Matthew Lease. 2004. An improved model for recognizing disfluencies in conversational speech. In *Rich Transcription 2004 Fall workshop (RT-04F)*. Palisades, NY.

Jeremy G. Kahn, Mari Ostendorf, and Ciprian Chelba. 2004. Parsing conversational speech using enhanced segmentation. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Short Papers*, pages 125–128. Association for Computational Linguistics, Boston, Massachusetts, USA.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–11.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 673–682. Association for Computational Linguistics, Portland, Oregon, USA.

Thomas Lavergne, Olivier Cappé, and François Yvon. 2010. Practical very large scale CRFs. In *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 504–513. Association for Computational Linguistics.

Percy Liang. 2005. *Semi-supervised learning for natural language*. Ph.D. thesis, MIT.

Yang Liu, Andreas Stolcke, Elizabeth Shriberg, and Mary Harper. 2005. Using conditional random fields for sentence boundary detection in speech. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 451–458. Association for Computational Linguistics, Ann Arbor, Michigan.

Michell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

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*, pages 820–825. Association for Computational Linguistics, Atlanta, Georgia.

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

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*, pages 48–53. Association for Computational Linguistics, Gothenburg, Sweden.

B. Roark, M. P. Harper, E. Charniak, B. Dorr, M. Johnson, J. Kahn, Y. Liu, M. Ostendorf, J. Hale, A. Krasnyanskaya, M. Lease, I. Shafran, M. Snover, R. Stewart, and L.Yung. 2006a. Sparseval: Evaluation metrics for parsing speech. In *Proceedings of Language Resource and Evaluation Conference*, pages 333–338. European Language Resources Association (ELRA), Genoa, Italy.

B. Roark, Yang Liu, M. Harper, R. Stewart, M. Lease, M. Snover, I. Shafran, Bonnie J Dorr, J. Hale, A. Krasnyanskaya, and L. Yung. 2006b. Reranking for sentence boundary detection in conversational speech. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1. IEEE, IEEE.

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

Xu Sun, Takuya Matsuzaki, Daisuke Okanohara, and Jun'ichi Tsujii. 2009. Latent variable perceptron algorithm for structured classification. In *IJCAI*, pages 1236–1242.

Dongdong Zhang, Shuangzhi Wu, Nan Yang, and Mu Li. 2013. Punctuation prediction with transition-based parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 752–760. Association for Computational Linguistics, Sofia, Bulgaria.

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