Detecting Speech Repairs Incrementally Using a Noisy Channel Approach

Simon Zwarts, Mark Johnson, Robert Dale

Department of Computing
Macquarie University

COLING 2010
Research goals

- Spontaneous speech often contains disfluencies
  
  I want a flight to Boston, uh, I mean, to Denver on Friday which we’d like to detect and delete in order to produce a more fluent transcript

- Current disfluency detection/correction systems process entire sentences at a time

- An incremental speech disfluency detector/corrector could better integrate with incremental speech recognition
  - and ultimately might not require sentence segmentation

- We describe an incremental version of the Charniak and Johnson (2004) TAG-based model

- We also propose two new metrics to measure how quickly and accurately an incremental disfluency system detects disfluencies
Speech errors in (transcribed) speech

• Filled pauses:
  
  \( I \text{ think it’s, } \text{uh}, \text{ refreshing to see the, } \text{uh}, \text{ support } \ldots \)

• Parentheticals:

  \( But, \text{ you know, I was reading the other day } \ldots \)

• Speech repairs:

  \( Why \text{ didn’t he, why didn’t she stay at home? } \)

• Ungrammatical constructions:

  \( My \text{ friends } is \text{ visiting me? } \)
Why focus on speech repairs?

- *Filled pauses* are easy to recognize (in transcripts at least)
- *Parentheticals* are easy to detect (e.g., parsing)
- "*Ungrammatical*" *constructions* aren’t necessarily fatal
  - Statistical parsers learn mapping of sentences to parses in training corpus
- *Speech repairs* warrant special treatment, since standard PCFG-based parsers misanalyse them
Shriberg’s analysis of speech repairs

*I want a flight to Boston, uh, I mean to Denver on Friday*

- The Interregnum is usually lexically (and prosodically marked), but can be empty
- Repairs can cross syntactic boundaries

  *Why didn’t she, uh, why didn’t he stay at home?*

  and interfere with syntactic parsing
- The Repair is often “roughly” a copy of the Reparandum

  \[ \Rightarrow \text{identify repairs by looking for “rough copies”} \]
- The Reparandum is often short (only 1–2 words long)

  \[ \Rightarrow \text{word-by-word classifiers can be quite successful} \]
- The Reparandum and Repair can be completely unrelated
Noisy channel approach to disfluency detection

- Goal: recover the most likely source string \( \hat{x} \) given observed string \( u \)

\[
\hat{x} = \arg\max_x \Pr(x|u) = \arg\max_x \Pr(u|x) \Pr(x)
\]
The language model

• Given the observed sentence

\[ u = \text{I want a flight to Boston, uh, to Denver on Friday} \]

the (true) source sentence is

\[ x = \text{I want a flight to Denver on Friday} \]

• The language model estimates \( \Pr(x) \)
  
  here we use a **bigram language model**
  
  \[
  \Pr(x) = \Pr(I | $) \Pr(\text{want} | I) \Pr(a | \text{want}) \Pr(\text{flight} | a) \\
  \Pr(\text{to} | \text{flight}) \Pr(\text{Denver} | \text{to}) \Pr(\text{on} | \text{Denver}) \\
  \Pr(\text{Friday} | \text{on}) \Pr($ | \text{Friday})
  \]
• Channel model is a transducer generating surface:source pairs \( u_i : x_i \):
  
a:a flight:flight to:0 Boston:0 uh:0 I:0 mean:0 to:0 Denver:Denver

• Crossing dependencies \( \Rightarrow \) channel model is a TAG
  ▶ TAG does not reflect grammatical structure (but LM can)
  ▶ right branching finite state model of non-repairs and interregnum
  ▶ adjunction used to describe copy dependencies in repair
Sample TAG derivation (simplified)

(I want) a flight to Boston uh I mean to Denver on Friday …

Start state: $N_{want}$

TAG rule: $N_{want}$, resulting structure:

TAG rule: $N_a$, resulting structure:

$R_{flight:flight}$
Sample TAG derivation (cont)

*(I want) a flight* to Boston uh I mean to Denver on Friday …

```
N_{want}
  /   \
 a:a  N_a
 /  \                  /  \
flight:flight  R_{flight:flight}  to:0  R_{to:to}
    /       \
   I↓  R^*_{flight:flight}  to:to  I↓
previous structure  TAG rule  resulting structure
```
(I want) a flight to Boston uh I mean to Denver on Friday …

previous structure

resulting structure

TAG rule
(I want) a flight to Boston uh I mean to Denver on Friday …
(I want) a flight to Boston uh I mean to Denver on Friday …
Training Data

- Switchboard corpus (1.3M training words) annotates reparandum, interregnum and repair (we ignore punctuation and partial words)
  
  I/PRP want/VBP a/DT flight/NN [to/TO Boston/NNP ,/, + {F uh/UH ,/, } {E I/PRP mean/VBP ,/, } to/TO Denver/NNP] on/IN Friday/NNP
  
  ▶ 5.4% of words are in a reparandum
  ▶ 31K repairs, average length: 1.6 words

- Reparandum and repair word-aligned by minimum-edit-distance, prefers identity, POS identity, similar POS

- Of the 57K alignments in the training data:
  
  ▶ 35K (62%) are identities
  ▶ 7K (12%) are insertions
  ▶ 9K (16%) are deletions
  ▶ 5.6K (10%) are substitutions (5% with same POS)
Dynamic programming algorithm for noisy channel

\[ I \text{ want a flight to Boston, uh, I mean to Denver on Friday} \]

- The most likely analysis \( \hat{x} \) generated by the noisy channel model (bigram language model + TAG channel model) can be found using dynamic programming
- Charniak and Johnson (2004) propose a \( O(n^5) \) algorithm that involves updating a table with entries of the form

\[ \langle \text{reparandum start, reparandum end, repair start, repair end} \rangle \]

\( \text{together with standard bigram trellis entries} \)

- The table entries can be computed in bottom-up left-to-right order

\[ \Rightarrow \text{an incremental version of the Charniak and Johnson model} \]
Bottom-up restricts incrementality

I want a flight to Boston, uh, I mean to Denver on Friday

The model’s two basic assumptions:
1. The repair looks like the reparandum
2. A sentence without the disfluency is fluent
don’t hold until the disfluency has been completed

I want a flight to Boston, uh, I mean, to . . .

- to Boston does not (yet) look very much like to
- taking the disfluency out, there is no fluent continuation (yet)

Pure bottom-up computation delays until the disfluency has completed and the continuation seen
Increasing incrementality with speculative completion

• We modify the algorithm to speculatively complete an incomplete repair
  ➤ *incremental completion substitution* assumes that unanalysed words in the reparandum are substitutions of (as yet unseen) words in the repair
  ➤ the probability is calculated by summing over all possible repair word substitutions

• When the actual following words are observed, we replace the speculatively completed chart cells with their true values
  ⇒ A disfluency detected by speculative completion may be revised as following words are observed
Evaluating disfluency detection

I want a flight to Boston, uh, I mean to Denver on Friday

- reparandum
- interregnum
- repair

- Fluent words are much more common than disfluent words
  ⇒ percent correct is not very informative
  ⇒ prior work reports $f$-score of fluent/disfluent labels (or other metrics)

- At the end of the sentence, the incremental algorithms produce same analyses as Charniak/Johnson algorithm
  ⇒ Incremental algorithms achieve same $f$-score (0.778) as Charniak/Johnson algorithm
Time to detection evaluation

I want a flight to Boston, uh, I mean to Denver on Friday

- reparandum
- interregnum
- repair

- Time to detection evaluates how quickly an algorithm proposes a disfluency
  - *average time to detection*: average number of words from start of reparandum to when repair is first detected
- Time to detection results:
  - No speculation: 5.1 words, with speculation: 4.6 words
  - ⇒ *speculation speeds disfluency detection by 0.5 words on average*
Delayed f-score at $k$ words

$I$ want a flight to Boston, uh, I mean to Denver on Friday

- Delayed f-score at $k$ words forces the model to label each word as fluent/disfluent when it has seen $k$ additional words
  - *delayed f-score at $k$ words*: f-score evaluated when input is $k$ words beyond word evaluated
- Delayed f-score results:

<table>
<thead>
<tr>
<th>$k$ tokens back</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>No speculation</td>
<td>0.500</td>
<td>0.558</td>
<td>0.631</td>
<td>0.665</td>
<td>0.701</td>
<td>0.714</td>
</tr>
<tr>
<td>With speculation</td>
<td>0.578</td>
<td>0.633</td>
<td>0.697</td>
<td>0.725</td>
<td>0.758</td>
<td>0.770</td>
</tr>
</tbody>
</table>

$\Rightarrow$ *Speculation does not decrease accuracy of disfluency detection*
Conclusion and future work

• It’s possible to develop an incremental version of the Charniak/Johnson disfluency detection algorithm
  ▶ Speculative completion speeds disfluency detection without decreasing accuracy

• Future work:
  ▶ develop a version that does not require sentence-segmented input
  ▶ develop models that detect disfluencies even earlier
  ▶ replace the bigram language model with an incremental parsing model
  ▶ develop methods for training disfluency models from data without disfluency annotations
  ▶ couple this with an incremental speech recogniser
Interested in statistical models for computational linguistics? We’re recruiting PhD students!

Contact Mark.Johnson@mq.edu.au or Katherine.Demuth@mq.edu.au for more information.