Detecting Errors in Parsing of Speech Transcript

Introduction

find errors primarily due to active linguistic repair in the parsing of Speech Transcript

transcript of NPTEL Lectures. Standard English Language tokenized text separated into sentences

analyze: Course 101101058, Lectures 14 to 19

observation set: Course 101101058, Lectures 14 and 15

small set of intuitive rules, say $\mathscr{R}$ - pinpoint a particular parse structure as grammatically incorrect / show possibility of linguistic repair

Parser

Spacy's dependency parser

- industrial grade
- better results than Stanford CoreNLP
- set of rich linguistic features
- uses a trained supervised neural model to create modules
- higher accuracies on available extrinsic and intrinsic tasks
- consistent
- intuitive sense

    - ❏ handling conjunctions ‘cc’(s)

Repair

alteration that is suggested or made by a speaker, the addressee, or audience in order to correct or clarify a previous conversational contribution

may occur at any of several points following the contribution in question, perhaps occurring in accordance with a conventional order of preference

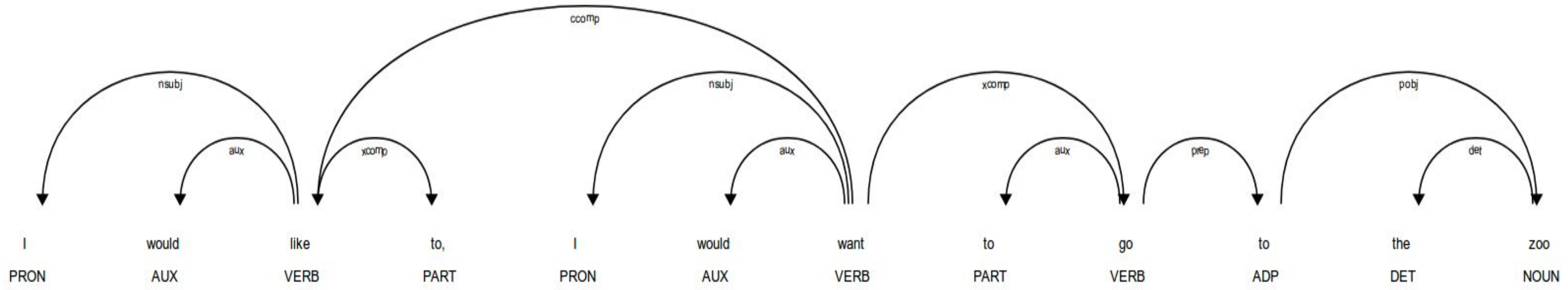
self-initiated repair: speaker of the utterance makes a repair without any prompting from another participant.

… to be continued

Violations

"I would like to, I would want to go to the zoo."

xcomp leading to a single ‘to’
incomplete phrase
xcomp must lead to a complete clause, missing in this case



Parser at Work

S_3 = “So all of these together has ensured that we have transonic compressors today , which are partly supersonic , partly subsonic , which means the compressors can go supersonic that means the flow can be in entirely supersonic and such compressors , supersonic compressors are indeed used in special circumstances specially in a rocket motors where you need very very compact compressors .”

S_8 = “Very high compression ratio would normally require supersonic flow in both rotor and stator and this is what is often done in as I mentioned some of the rocket motor compressors on other hand in most of the air craft engines as of today only the rotor is supersonic where as the stators and by enlarge subsonic but , as we can see here it is entirely possible for stator to also go supersonic .”

** Refer to the Report for a set of 15 examples*

Parser Passes

not grammatically sound

demonstrate active linguistic repair

dependency parse is pretty much what we would expect

parser does a neat job, appears convoluted but has no apparent errors

ungrammaticality leads to ‘weirdness’ of the parse structure

by hand, similar to machine

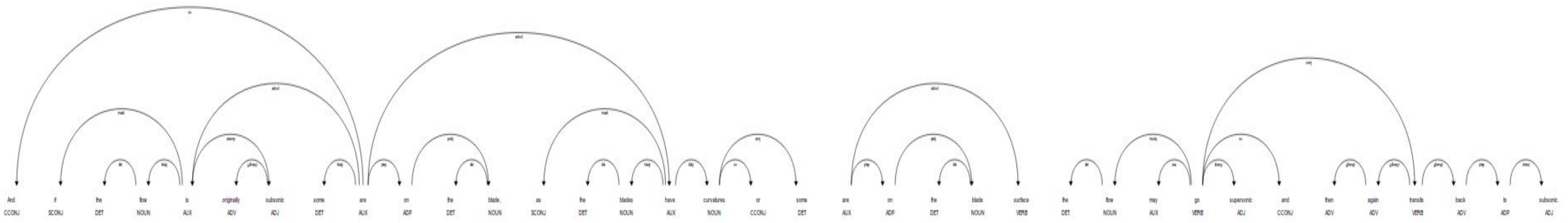
machine does a reasonable or better job

Hence, not Errors

Parser Fails

“And if the flow is originally subsonic some are on the blade, as the blades have curvatures or some are on the blade surface the flow may go supersonic and then again transits back to subsonic.”

- not a connected graph
- three subtrees
- valid sentence, components must share some sort of connection
- ungrammaticality
- repair



Self Repair

1. Repetitions: the speaker repeats some part of the utterance. E.g. I * I like it.
2. Revisions (content replacement): the speaker modifies some part of the utterance. E.g. We * I like it.
3. Restarts (also called false starts): a speaker abandons an utterance or constituent and then starts over. E.g. It’s also * I like it.

Source: Automatic Disfluency Identification in Conversational Speech Using Multiple Knowledge Sources, Liu et al., 2003.

Features Used for Disfluency Detection

- acoustic features
- prosodic features
- standard language model
- part-of-speech (POS) based LM
- rule-based knowledge

Naive Approach

- Word Overlaps

bigram overlap within a sentence does not capture repair effectively

encountered some bigram overlaps in sentences having repair in our observation set

verification set of 865 sentences,

maximum windows of size = 31

bigram overlaps = 2

not an effective metric - need to capture '*meaning*'

Capturing Meaning

- need to move beyond surface level
- capture information about tokens, substructures
- compare substructure information to check for instances of valid repair
- effectively capture meanings for comparison

Repetitions in Small Windows

default window size = 7

in some window of seven words, if we find two lexical nodes having very similar meanings

classify as a possible lexical repetition in a small window

stop words and function words are not considered

What we achieved

S: “Now , we know , that the degree of reaction should never be 0 anywhere on the blade or definitely , never be less than 0 anywhere on the rotor blade .”

utterance_1 = “should never be 0 anywhere on the blade”
utterance_2 = “never be less than 0 anywhere on the rotor blade .”

- classified by our system as violation + repetition with parse:

utterance_2 repairs utterance_1, by providing a clarification - therefore a correction, or rather a modification

#*note*: would not get captured purely looking at lexical overlap

* for more examples, refer to our notebook

Graphs

rely on the compositionality of the hierarchical structure

experiment with various structures

- dependency
- constituency
- + anaphora markers
- GCNNs

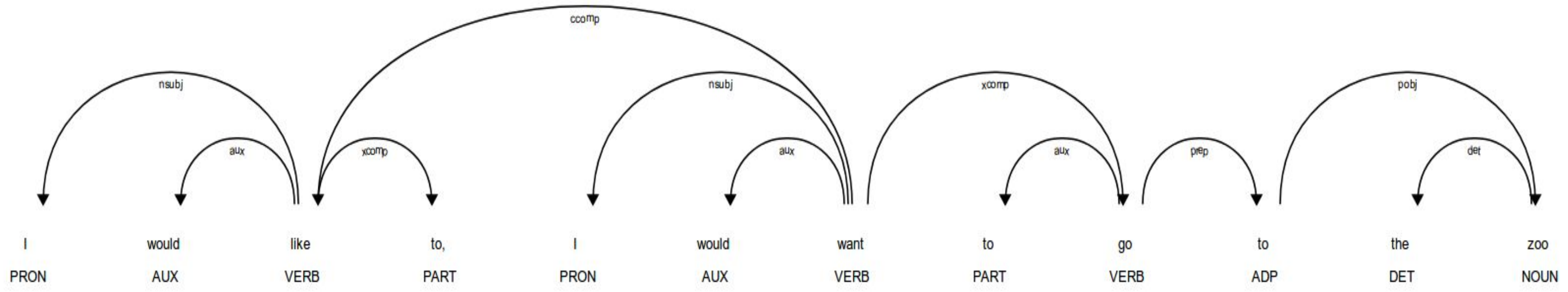
But, no need to modify the graph module

Capturing Information

compose meanings of individual words

get meanings & positional information for each “substructure”

if “very similar” and not part of same structure, flag repair



Substructure Feature Specification

- Size
- Positional
  - Order in the Depth First Search traversal
  - Shortest distance between substructure heads
- Meaning Representation
- Node Structure

Others

the Dynamic Programming algorithms for building substructures

querying using pointer optimizations

capturing the meaning

training on observation samples + toy examples

setting hyperparameters

efficient implementation for better runtime

Takeaway

can use the same structure to efficiently detect parse errors

creation of a global substructure set can be achieved O (n) hops, *n* being the sentence length

graph + small window = all sorts of possible repair

Results

	counts
Sentences	865
Not a Tree	40
'xcomp' Condition Violated	2
Possible Repetitions using Window	253
Possible Repetitions using Parse	627
Bigrams repeated	2
Parsing Constraint violated	42
Violation + Repetition with Window	18
Violation + Repetition with Parse	30

set	%
violating parse constraints (V)	4.86
V + repair with Window	42.86
V + repair with Parse	71.43

Qualitative Verification

S: “ You can indeed choose intermediate values , you do not have to choose only 0.5 or 1 or 2 , you can indeed choose intermediate values , but that well give you an idea , prima-facie a first cut idea based on two-dimensional cascade understanding how much delta beta , that particular blade section can actually produce . ”

repeated utterance = “you can indeed choose intermediate values”

- classified by our system as violation + repetition with parse:

Hence, it works

Future

- tree-LSTMs
- siamese Nets
- other types of Hierarchical Structures

Extra Slides

Algorithms and Optimizations

Modules for Detecting Errors

- set of four necessary and sufficient modules
- can be augmented with more rules

Handle Parse Constraint Violations

M2: Given the root node of a tree, Find if it violates any ‘xcomp’ conditions

```
Method: Requires a ‘node’ as a parameter
let all_children ← set of all children of node
if all_children is an empty set and the dependency label of arc to node is ‘xcomp’,
    then return False,
        else return True

    else, let subtree_result ← True,
    iterate over all_children with iterator child: DO
        update subtree_result as (subtree_result and M2 (child))

    return subtree_result
```

Handle Parse Constraint Violations..

```
M3: Given the root node, find the size of the tree:
Method: Requires a ‘node’ as a parameter
let all_children ← set of all children of node
if all_children is an empty set,
    then we have encountered a leaf node and we return 1
    else,
        let subtree_size ← 1

        iterate over all_children with iterator child: DO
                update subtree_size as (subtree_size + M3 (child))

        return subtree_size
```

Handle Parse Constraint Violations...

```
M1: Given a set of nodes S, Find the Root Node
Method: Requires a set of nodes, S as a parameter
iterate over S with iterator node: DO
    if parent of node is the node itself,
        then return node as the Root

M4: Given a set of nodes, check if the graph is connected or not
Method: Requires a set of nodes, S as a parameter
let root_token ← M1 (S) and let tree_size ← M3 (root_token)
if tree_size equals size of S,
    then return 1,
        else return 0
```

Foundations

global store ‘all_structs’ - store structural information about subtrees rooted at every node in the graph DS - for a sentence

populating iteratively would have taken a complexity of O(N ^ 2), N being the number of nodes

DP on trees: Reduces the Complexity to O(N)

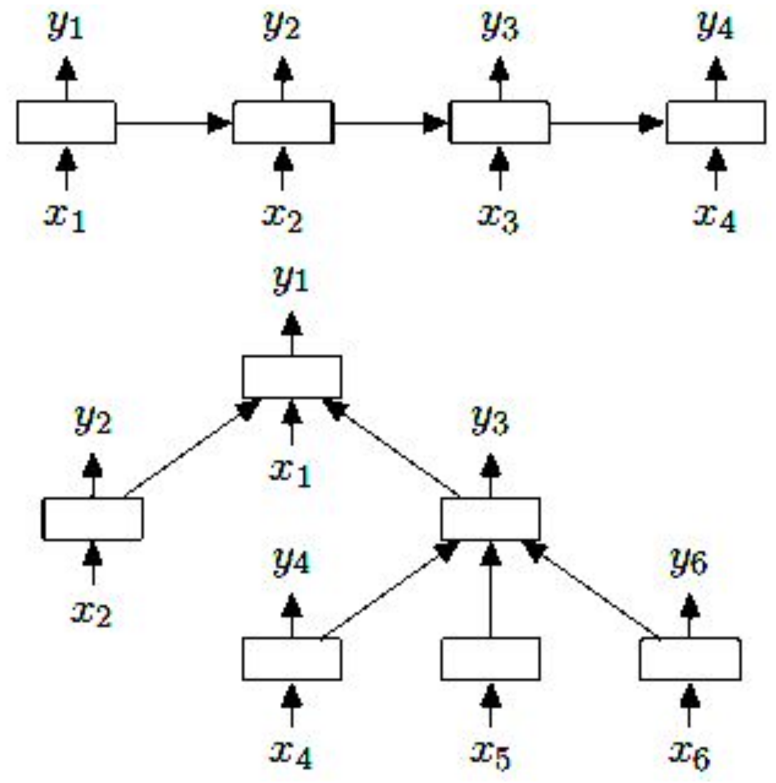
```
dp [‘child’] = {'size': 1, 'hash': the word embedding of node text, 'node': the child node}
dp [‘node’] = {‘size’: 1 + sum of dp [‘i’] where ‘i’ is a child of node,
               ‘hash’: centroid of embedding of text in node with all hash [‘i’];
                                           where ‘i’ is a child of node,
               ‘node’: the present node}
```

Pointers

parent pointers

reduces number of steps of finding the height of a node from (2 * N - 1) to H

need of search discounted



Utilizing the Optimizations

check if a struct is descendant of another: O(max (height (node_1), height (node_2)))

2 variables: struct_A and struct_B

```
assign struct_B to be the struct having the higher node
initialize an iterator variable token with ‘node’ parameter in struct_A

while parent of token is not the same the node: DO
    if parent of token is the same as the ‘node’ parameter of struct_B,
        then return True
otherwise, return False
```

Feature Specification

Feature Engineering

1. length of the shortest path between the 2 nodes
2. difference in their indices of appearance in the depth-first search traversal

feature 2 - easily computed using our store - maintains all nodes in order of their depth-first traversal

get feature 1 in linear time:

path_length (A, B) = path_length (A, root) + path_length (B, root)
- 2 * path_length (root, LCA(A, B)) — (I)

- where LCA (A, B) is the least common ancestor of A and B

Feature Engineering...

Obtain LCA(A, B) in linear time

1. create 2 paths: Path_A storing nodes from A to the root, Path_B storing nodes from B to the root
2. reverse each path and iterate over them from the root
3. hit an uncommon node, we return the previous node as the LCA
4. completely traversed one out of the two paths, we return the last node of the said path as the LCA

takes 3 * height_of_tree operations, Complexity ∈ O (height_of_tree).

then, using (`I`), we obtain the value for feature 1

Similarity Check

normalize hashed embeddings of each struct by their respective node sizes

return:

1. The cosine similarity
2. The L2 norm of the difference

between their computed centroid embeddings

Repair using the Parse Structure

Primary Repair Finder - operates on the global store of all structs

```
iterate over pairs (struct1, struct2) of structs: DO
    ignore the following cases:
     a.  the two structs have nodes which appeared which were less than ‘nhops’
         hops away in the DFS traversal of the tree
     b.  the shortest path between the two struct nodes is of length less than ‘lpath’
     c.  the two struct nodes are bound by an ancestor-descendant relationship

    else, get similarity measure between struct1 and struct2
        if cosine similarity is less than threshold ‘tcos’, and the L2 norm of difference
        is less than threshold ‘tnorm’, then declare that repair has been detected
```

Hyperparameter Tuning

human in the loop

manually set parameter values to check what works best for a small set of sentences from our observation set bootstrapped with a few toy examples

nhops = 6
lpath = 8
tcos = 0.975
tnorm = 1.15

Thank You