Transition based dependency parsing

Parsing
ISCL-BA-06

Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2020/21

version: 284d23c @2021-02-08

Dependency parsing
an overview

- Dependency parsing has many similarities with context-free parsing (e.g., the result is a tree)
- They also have some different properties (e.g., number of edges and depth of trees are limited)
- The process involves discovering the relations between words in a sentence
  - Determine the head of each word
  - Determine the relation type
- Dependency parsing can be
  - grammar-driven (hand crafted rules or constraints)
  - data-driven (rules/model is learned from a treebank)

Grammar-driven dependency parsing

- Grammar-driven dependency parsers typically based on
  - lexicalized CF parsing
  - constraint satisfaction problem
    - start from fully connected graph, eliminate trees that do not satisfy the constraints
    - exact solution is intractable, often employ heuristics, approximate methods
    - sometimes 'soft', or weighted, constraints are used
  - Practical implementations exist
- Our focus will be on data-driven methods

Dependency parsing
common methods for data-driven parsers

- Almost any modern/practical dependency parser is statistical
- There are two main approaches:

Graph-based search for the best tree structure, for example
  - find minimum spanning tree (MST)
  - adaptations of CF chart parser (e.g., CKY)

  (in general, computationally more expensive)

Transition-based similar to shift-reduce (LR(k)) parsing
  - Single pass over the sentence, determine an operation (shift or reduce) at each step
  - Linear time complexity
  - We need an approximate method to determine the best operation

Shift-Reduce parsing
a refresher through an example

Grammar

S $\rightarrow$ P | S + P | S − P
P $\rightarrow$ Num | P × Num | P / Num

Parser states/actions

Stack	Input buffer	Action
	2 + 3 × 4	shift
2	+ 3 × 4	reduce (P → Num)
P	+ 3 × 4	reduce (S → P)
S	+ 3 × 4	shift
S +	3 × 4	shift
S + 3	× 4	reduce (P → Num)
S + P	× 4	shift
S + P ×	4	shift
S + P × 4		reduce (P → P × Num)
S + P		reduce (S → S + P)
S		accept

Transition-based parsing
differences from shift-reduce parsing

- The shift-reduce (LR) parsers for formal languages are deterministic, actions are determined by a table lookup
- Natural language sentences are ambiguous, a dependency parser's actions cannot be made deterministic
- Operations are (somewhat) different: instead of reduce (using phrase-structure rules) we use *arc* operations connecting two words with a labeled arc
- More operations may be defined (e.g., to deal with non-projectivity)

Transition based parsing

- Use a *stack* and a *buffer* of unprocessed words
- Parsing as predicting a sequence of transitions like
  - Left-Arc: mark current word as the head of the word on top of the stack
  - Right-Arc: mark current word as a dependent of the word on top of the stack
  - Shift: push the current word on to the stack
- Algorithm terminates when all words in the input are processed
- The transitions are not naturally deterministic, best transition is predicted using a machine learning method

A typical transition system

$$(\underbrace{\sigma \mid \overbrace{w_i}^{\text{stack top}}}_{\text{stack}}, \underbrace{\overbrace{w_j}^{\text{next word}} \mid \beta}_{\text{buffer}}, \underbrace{A}_{\text{arcs}})$$

Left-Arc$_r$: $(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_j \mid \beta, A \cup \{(w_j, r, w_i)\})$
- pop w_i,
- add arc (w_j, r, w_i) to A (keep w_j in the buffer)

Right-Arc$_r$: $(\sigma \mid w_i, w_j \mid \beta, A) \Rightarrow (\sigma, w_i \mid \beta, A \cup \{(w_i, r, w_j)\})$
- pop w_i,
- add arc (w_i, r, w_j) to A,
- move w_i to the buffer

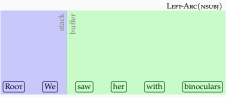
Shift: $(\sigma, w_j \mid \beta, A) \Rightarrow (\sigma \mid w_j, \beta, A)$
- push w_j to the stack
- remove it from the buffer

Transition based parsing: example

Shift

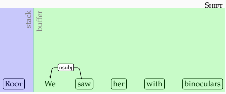
Transition based parsing: example

Left-Arc(nsubj)



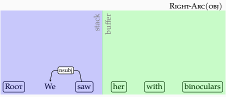
Transition based parsing: example

Shift



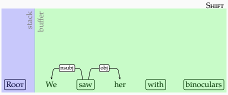
Transition based parsing: example

Right-Arc(obj)

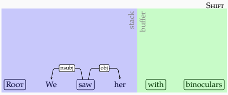


Note: We need Shift for NP attachment.

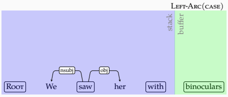
Transition based parsing: example



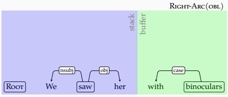
Transition based parsing: example



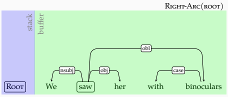
Transition based parsing: example



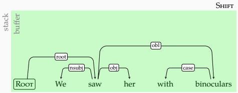
Transition based parsing: example



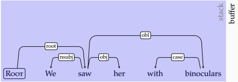
Transition based parsing: example



Transition based parsing: example



Transition based parsing: example



Making transition decisions

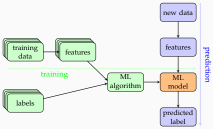
- In LR(k) parsing, the actions are deterministic: there is only one action to take on every parser state
- In transition-based dependency parsing, we have to choose the best among multiple actions
- The typical method is to train a (discriminative) classifier on features extracted from gold-standard *transition sequences*
- Almost any machine learning (classification) method is applicable

Classification

- Classification refers to *supervised* machine learning methods that predict categorical variables (e.g., POS tags or parser actions)
- The predictions are based on statistics extracted from a *training set*
- There are a large number of classification methods, just a few examples:
  - Logistic regression
  - Decision trees
  - Support vector machines
  - Memory-based learning
  - (Deep) neural networks

Supervised learning

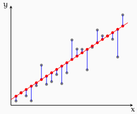
with a picture



Types of supervised learning

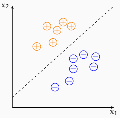
- If we want to predict a numeric value, the problem is called *regression*
  - Age of the author
  - Frequency of a word
  - Reaction time to a stimuli
- If we want to predict a label, or category, the problem is called *classification*
  - Part of Speech of a word
  - Whether document is spam or not
  - The translation of a word
  - The action to take during transition-based parsing

Supervised learning: regression

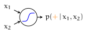


- We want to predict y form x
- Our model is the linear equation with least error
- The idea is to reduce the error on the training set

Supervised learning: classification



- We want to predict the class (+, or −) from the features (x_1 and x_2)
- A possible solution: find a function that separates the classes
- Another solution: predict the probabilities (*logistic regression*)



A note on generalization

- An important concern in machine learning is to learn to *generalize*
- A common issue with (complex) ML methods is *overfitting* – the system may learn 'memorize' the training data, rather than learning generalizations
- There are methods to prevent overfitting, e.g., *regularization*
- To make sure that there is no overfitting, you need to test your system on a separate data set

This is a very superficial introduction. You need to know more about the methods you are using so that you get the best out of these methods.

Features for transition-based parsing

- The features come from the parser configurations, for example
  - The word at the top of the stack, (peeking towards the bottom of the stack is also fine)
  - The first/second word on the buffer
  - Right/left dependents of the word on top of the stack/buffer
- For each possible 'address', we can make use of features like
  - Word form, lemma, POS tag, morphological features, word embeddings
  - Dependency relations – (w_i, r, w_j) triples
- Note that some 'address'–'feature' combinations may not be defined

Features for transition-based parsing

examples

- In transition-based parsing, transition decisions come from a classifier
- At each step during parsing, we have features like
  - `form[Stack] = saw`
  - `lemma[Stack] = see`
  - `POS[Stack] = VERB`
  - `form[Buff] = her`
  - `lemma[Buff] = she`
  - `POS[Buf] = PRON`
- We need to make a transition decision such as
  - Shift
  - Right-Arc(obj)
  - Right-Arc(obl)
  - Left-Arc(acl)
- We can use any multi-class classifier, examples in the literature include
  - SVMs
  - Decision Trees
  - Neural networks
  - ...

The training data

- We want features like,
  - `lemma[Stack] = duck`
  - `POS[Stack] = NOUN`
  - ...
- But treebank gives us:

```
1   Read   read   VERB  VB   Mood=Imp|VerbForm=Fin  0 root
2   on     on     ADV   RB   _                      1 advmod
3   to     to     PART  TO   _                      4 mark
4   learn  learn  VERB  VB   VerbForm=Inf           1 xcomp
5   the    the    DET   DT   Definite=Def           6 det
6   facts  fact   NOUN  NNS  Number=Plur            4 obj
7   .      .      PUNCT .    _                      1 punct
```

- The treebank has the outcome of the parser, but none of the features we expect

The training data

- The features for transition-based parsing have to be from *parser configurations*
- The data (treebanks) need to be preprocessed for obtaining the training data
- The general idea is to construct a transition sequence by performing a 'mock' parsing by using treebank annotations as an 'oracle'
- There may be multiple sequences that yield the same dependency tree, this procedure defines a 'canonical' transition sequence
- For example,

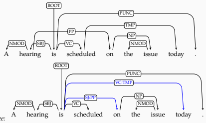
  Left-Arc$_r$ if $(\beta[0], r, \sigma[0]) \in A$
  Right-Arc$_r$ if $(\sigma[0], r, \beta[0]) \in A$
  and all dependents of $\beta[0]$ are attached
  Shift otherwise

Non-projective parsing

- The transition-based parsing we defined so far works only for projective dependencies
- One way to achieve (limited) non-projective parsing is to add special operations:
  - Swap operation that swaps tokens in swap and buffer
  - Left-Arc and Right-Arc transitions to/from non-top words from the stack
- Another method is pseudo-projective parsing:
  - preprocessing to 'projectivize' the trees before training
    - The idea is to attach the dependents to a higher level head that preserves projectivity, while marking it on the new dependency label
  - post-processing for restoring the projectivity after parsing
    - Re-introduce projectivity for the marked dependencies

Pseudo-projective parsing

Non-projective tree:
Pseudo-projective tree:



Transition based parsing: summary/notes

- Linear time, greedy, projective parsing
- Can be extended to non-projective dependencies
- We need some extra work for generating gold-standard transition sequences from treebanks
- Early errors propagate, transition-based parsers make more mistakes on long-distance dependencies
- The greedy algorithm can be extended to beam search for better accuracy (still linear time complexity)
- Reading suggestion: Jurafsky and Martin (2009, draft chapter 14): `https://web.stanford.edu/~jurafsky/slp3/14.pdf`, Kübler, McDonald, and Nivre (2009)

Next:

- Graph-based parsing: the MST

Acknowledgments, references, additional reading material

Grune, Dick and Ceriel J.H. Jacobs (2007). *Parsing Techniques: A Practical Guide.* second. Monographs in Computer Science. The first edition is available at `http://dickgrune.com/Books/PTAPG_1st_Edition/BookBody.pdf`. Springer New York. ISBN: 9780387689548.

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* second. Pearson Prentice Hall. ISBN: 978-0-13-504196-3. URL: `http://web.stanford.edu/~jurafsky/slp3/`.

Kübler, Sandra, Ryan McDonald, and Joakim Nivre (2009). *Dependency Parsing.* Synthesis lectures on human language technologies. Morgan & Claypool. ISBN: 9781598295962.

blank

blank

blank

blank