11-711 Advanced NLP

Dependency Parsing: Algorithms, Models and Resources

Zhisong Zhang

Outline

- Algorithms: The **output decomposition** and decoding algorithms.
    - **Graph-based**
    - **Transition-based**

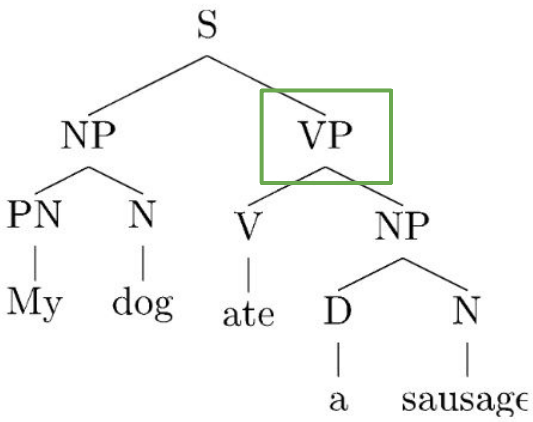
- Models: The **modeling** of the factorized pieces.
    - **Feature-based**
    - **Neural network**
    - **Pre-trained models**

- Resources: Looking closer to the **data resources** for dependency parsing.
    - **Universal dependencies**
    - **Cross-lingual transfer**

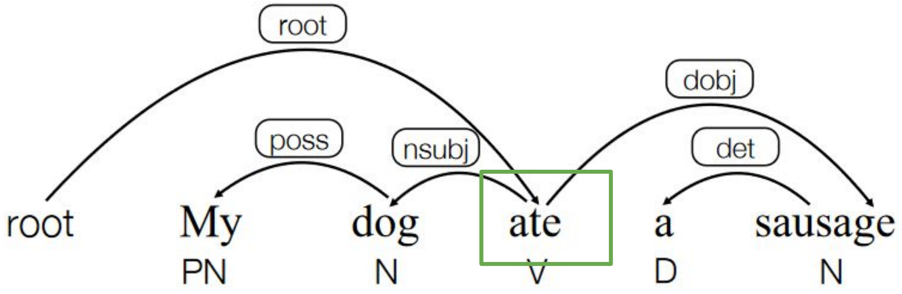
Recap.: Constituency and dependency trees

Phrase-structure trees: internal nodes for phrases
Dependency trees: only input words as nodes

Dep-tree properties:
1. No multi-dege;
2. Head=1;
3. No cycles;
4. (optional) Projective.



constituency (aka phrase-structure) tree

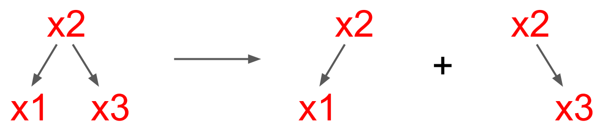


dependency tree

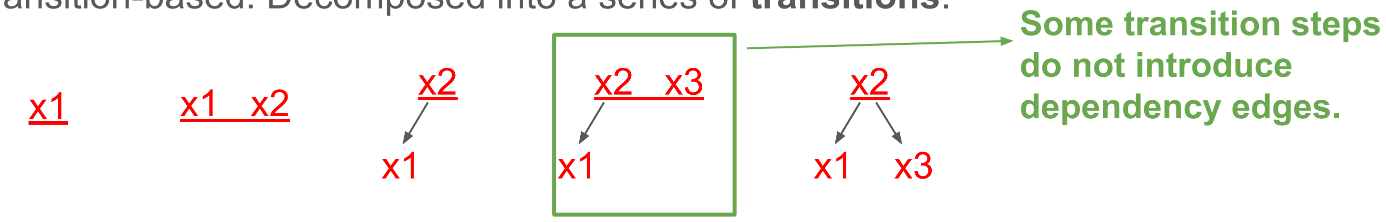
Algorithms: Overview

- Dependency parsing is a **Structured Prediction** problem, where we need to find a way to **decompose** the complex target.

- Conventionally speaking, there are **two categories**:
  - Graph-based: Decomposed into **individual** sub-trees.



  - Transition-based: Decomposed into a series of **transitions**.



Algorithms: Graph-based

Let's assume we have a “magic” scoring model.

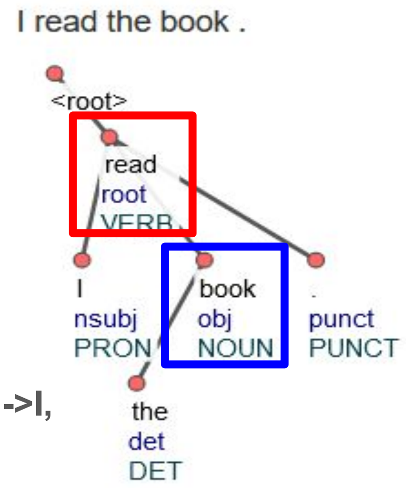
In graph-based methods, the full tree is decomposed into individual **sub-trees**.

- **Scoring**: $\mathbf{score}(\text{Tree}) = \sum_{s} \boxed{\mathbf{score}}(\text{SubTree})$
- **Decoding**: $T(\text{sent.}) = \text{argmax}_{T}\ \mathbf{score}(T)$

- We will examine the simplest case:

  first-order (arc-factored)

$\text{score}(\text{Tree}) \propto \sum_{(m, h)} \text{score}(\text{modifier}, \text{head})$

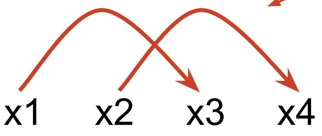


We have 5 edges here: {root->read, read->I, read->book, read->., book->the}

Algorithms: Constraints

Let's look at the **properties** of a well-formed dependency tree:

Constraints	Violation Example	Decoding Algorithm
No multiple-edges	x1 -> x2; x1 -> x2;	Enumeration (Binary class.)
Single-head	x1 -> x2 <- x3	Enumeration (Head class.)
No-cycle (**acyclic**)	x1 -> x2; x2 -> x1;	**Chu-Liu-Edmonds**
No-cross (**projective**)	x1 -> x3; x2 -> x4;	**Eisner's DP**



Dependency-version of CYK.

Algorithms: Chu-Liu-Edmonds

Examples and figures borrowed from (McDonald et al., 2005)

Exact algorithm for non-projective first-order parsing: (McDonald et al, 2005)

Chu-Liu-Edmonds(G, s)

Graph $G = (V, E)$

Edge weight function $s : E \rightarrow \mathbb{R}$

Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$

Let $G_M = (V, M)$

If G_M has no cycles, then it is an MST: return G_M

Otherwise, find a cycle C in G_M

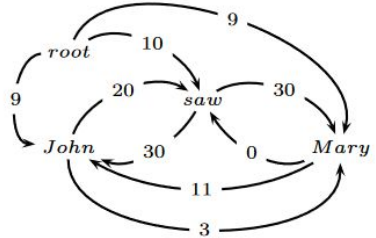
Let $G_C = \text{contract}(G, C, s)$

Let $\boldsymbol{y} = \text{Chu-Liu-Edmonds}(G_C, s)$

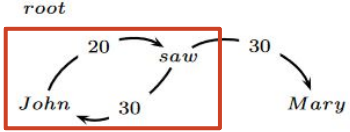
Find a vertex $x \in C$ s. t. $(x', x) \in \boldsymbol{y}, (x'', x) \in C$

return $\boldsymbol{y} \cup C - \{(x'', x)\}$

Greedy maximum



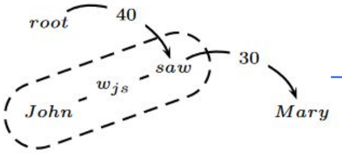
The first step of the algorithm is to find, for each word, the highest scoring incoming edge

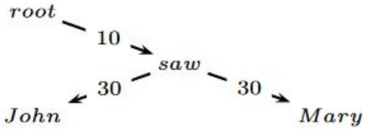


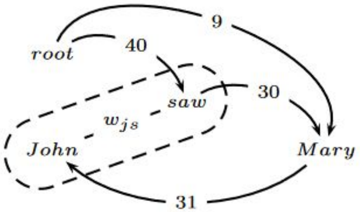
Contract

have a cycle, so we will contract it into a single node and recalculate edge weights according to Figure 3.

Recursive & Repack







Algorithms: Transition-based

Examples and figures borrowed from Speech and Language Processing (3rd ed. draft)

Transition-based method performs parsing by a series of (shift-reduce) transitions.

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

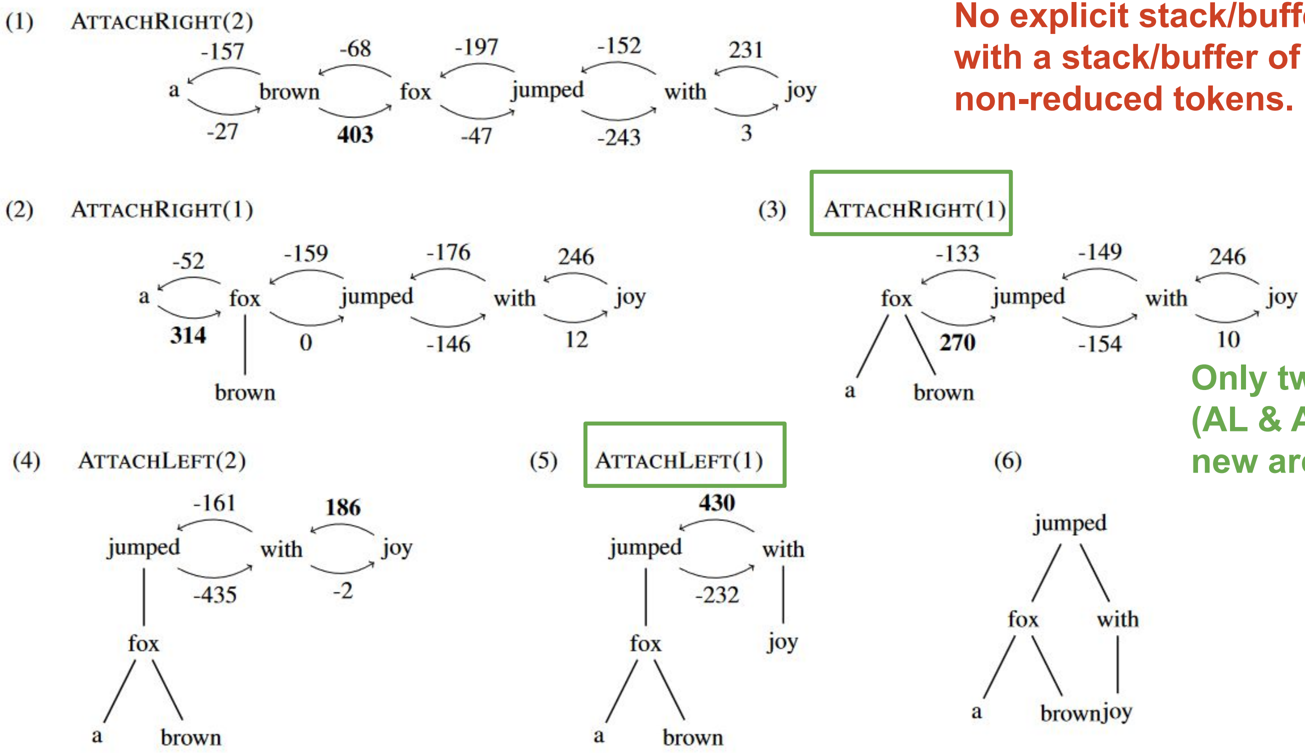
Broadly speaking, transition-based methods are **not constrained to** shift-reduce systems: (Bohet et al., 2016): **incremental** building with a series of actions.

See (Nivre, 2003) and (Yamada and Matsumoto, 2003) as typical examples.

Examples and figures borrowed from (Goldberg and Elhadad, 2010)

Algorithms: Easy-first

Non-directional **easy-first** parsing: (Goldberg and Elhadad, 2010)



No explicit stack/buffer, or with a stack/buffer of non-reduced tokens.

Only two types of actions (AL & AR), both create new arcs.

Algorithms: Tagging

With proper encoding, a dependency tree can be **cast as a sequence**, and the problem can be formulated as a sequence tagging problem: (Strzyz et al., 2019)

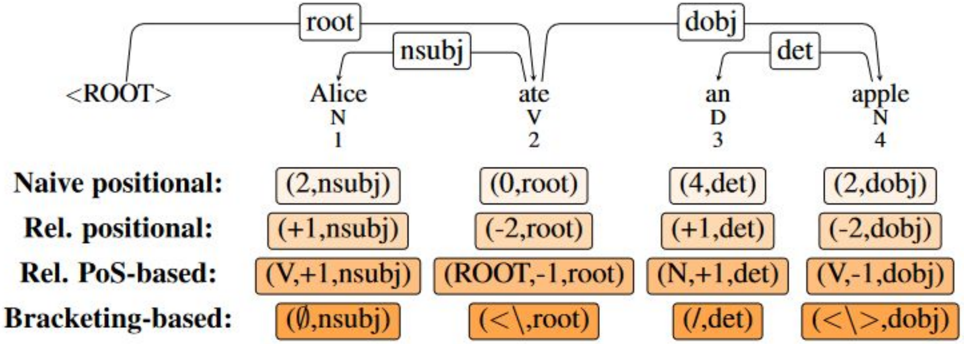


Figure 1: Types of encoding on an example tree.

In some way, still similar to a transition system with attaching transitions. But not necessarily built incrementally.

Algorithms: There's much more

A great tutorial (EACL 2014) on parsing algorithms (McDonald and Nivre, 2014).

- Graph-based:
  - Projective higher-order parsing (more complex CKYs).
  - Non-projective higher-order parsing (approximate methods).
  - Pruning methods.
- Transition-based:
  - Different varieties of transition systems.
  - DP with transition-based parsing.
  - Spurious ambiguity and dynamic oracle.

And surely there are ways to combine them: (Zhang and Clark, 2008)

(Nice title: "**A Tale of Two Parsers**: investigating and combining graph-based and transition-based dependency parsing using beam-search")*

Borrowed from (McDonald and Nivre, 2014)

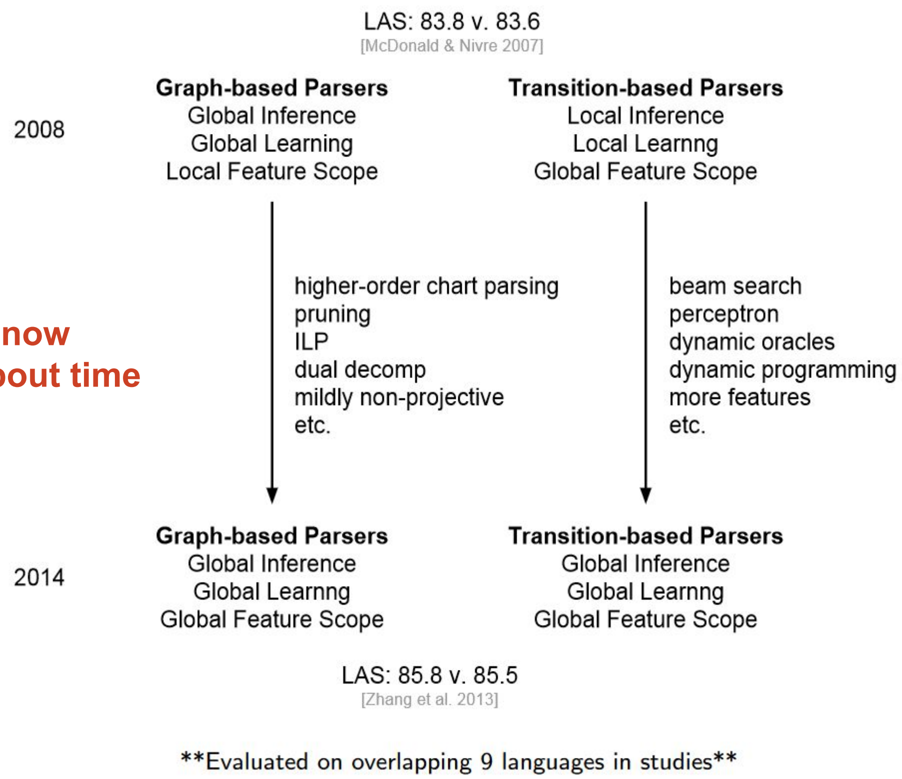
Algorithms: Summary

Typically:

- Graph-based:
  - Local factorization.
  - Global inference.
  - Mostly O(N^3)+.
- Transition-based:
  - Local normalization.
  - Rich output features.
  - Linear time (with shift-reduce).

Things change now when talking about time complexity.

These two can reach similar results, but with different characteristics.



Models: Overview

The above only mentions **inference algorithms**, all of them need **a model** to do the **scoring** of the decomposed parts:

Graph-based: **score**(m, h)   Transition-based: **score**(action|state)

How to design the input representations and the scoring model?

- Feature-based: with **manually designed features** and linear model
- (Early) **neural network**: with atom input features and NN scorer.
- With **contextualized representations**, especially with pre-trained encoders.

Models: Feature based

Borrowed from (McDonald et al., 2005)

Things are similar for transition-based methods.

a)

Basic Uni-gram Features
p-word, p-pos
p-word
p-pos
c-word, c-pos
c-word
c-pos

b)

Basic Big-ram Features
p-word, p-pos, c-word, c-pos
p-pos, c-word, c-pos
p-word, c-word, c-pos
p-word, p-pos, c-pos
p-word, p-pos, c-word
p-word, c-word
p-pos, c-pos

c)

In Between POS Features
p-pos, b-pos, c-pos
Surrounding Word POS Features
p-pos, p-pos+1, c-pos-1, c-pos
p-pos-1, p-pos, c-pos-1, c-pos
p-pos, p-pos+1, c-pos, c-pos+1
p-pos-1, p-pos, c-pos, c-pos+1

I[pron] **read**[verb] the[det] **book**[noun] .[punct] -> **score**(**book**, **read**)

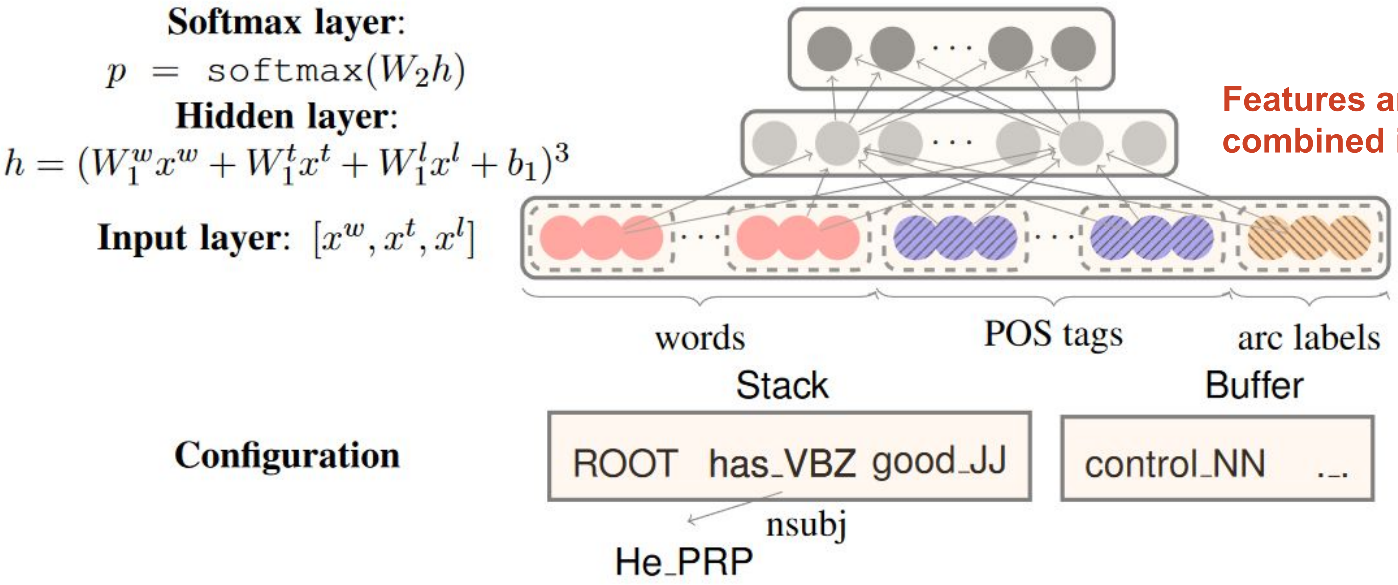
- a) “p-read&p-verb”, “p-read”, “p-verb”, “c-book&c-noun”, …
- b) “p-read&p-verb&c-book&c-noun”, “p-read&c-book&c-noun”, …
- c) “p-verb&b-det&c-noun, “p-verb&p-det+1&c-det-1&c-noun”, ...

Sparsity problem! There can be millions of features!

Models: Neural network

Borrowed from (Chen and Manning, 2014)

Things are similar for graph-based methods*.*



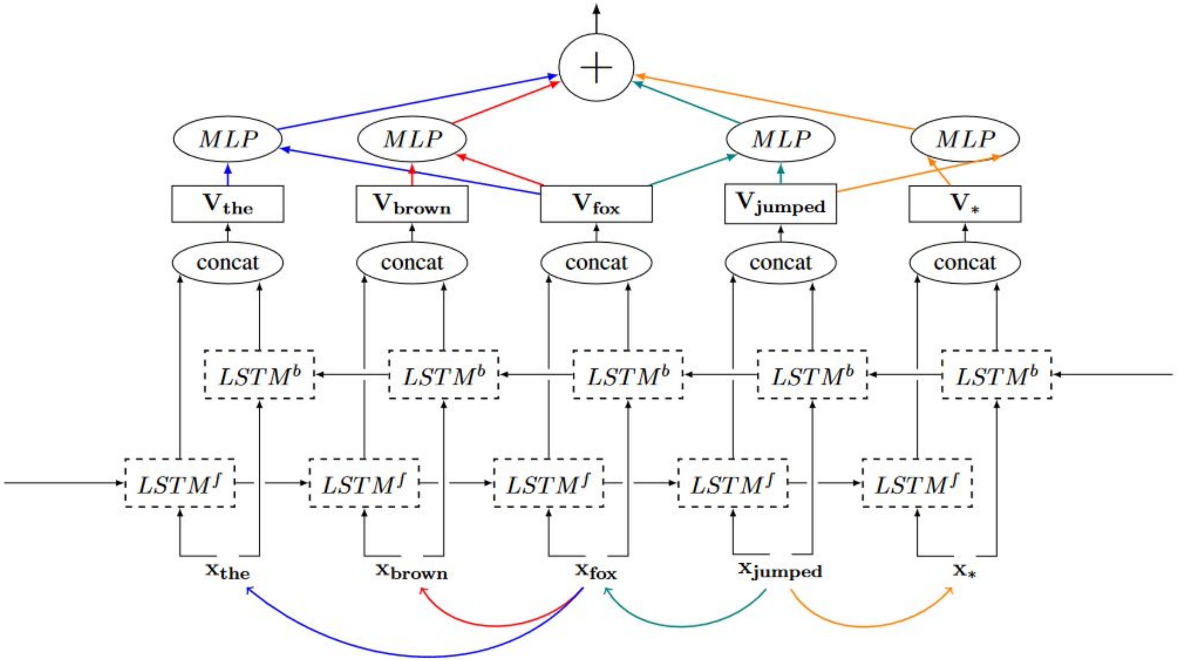
Features are automatically combined inside NN.

Only atomic inputs are fine, no longer manual feature combinations.

Borrowed from (Kiperwasser and Goldberg, 2016)

Models: With contextualized representations

Remember that for parsing, we have **full input sentence** as the input!



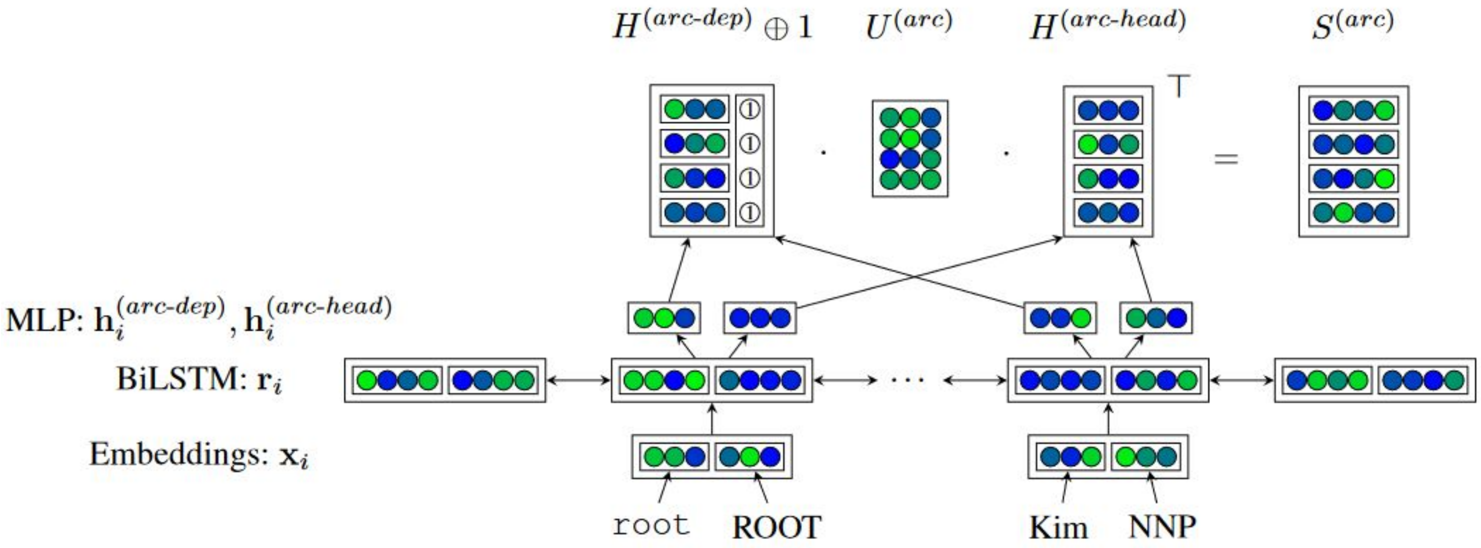
The inputs to the final scorer now contains the information of the full sentence.

Again, things are similar for transition-based ones.

Models: Deep Biaffine Scorer

Borrowed from (Dozat and Manning, 2017)

Probably nowadays *the "standard"* parsing scorer architecture.



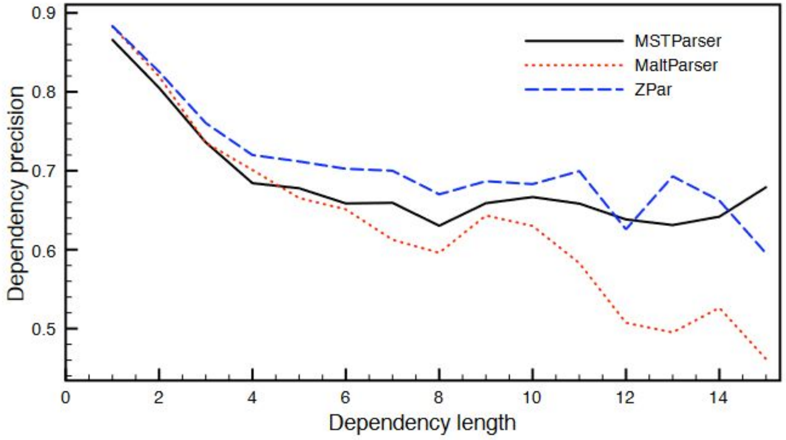
Models: With pre-trained models

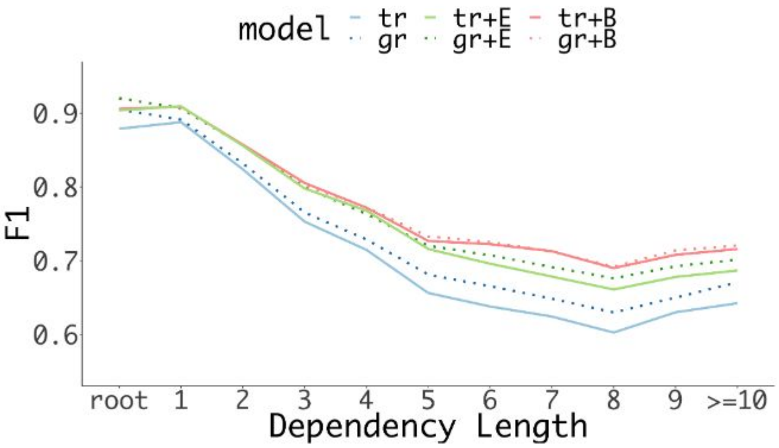
Figures borrowed from (Kulmizev et al, 2019)

As you can imagine, with recent **pre-trained contextualized embeddings**:

*Deep Contextualized Word Embeddings in Transition-Based and Graph-Based Dependency Parsing – **A Tale of Two Parsers Revisited***, (Kulmizev et al. 2019)

- *Contextualized word embeddings allow parsers **to pack information about global sentence structure** into local feature representations.*
- *They benefit transition-based parsers more than graph-based parsers, making the two approaches **virtually equivalent** in terms of both accuracy and error profile.*





Models: Summary

- A “huge change” to the parsing models?
    - **Maybe NOPE** (only changing the scorers)?
    - The **basic parsing paradigms** are almost the same.

- However, this indeed brings changes:
    - Somehow **blur the distinctions** between graph- and transition-based methods.
    - When talking about computational **complexity**:
        - (CPU-oriented), graph O(N^3) > transition O(N).
        - (GPU-oriented), graph (easier to parallelize) <= transition (not GPU-friendly)?
    - Standard parsing model: “Bert (and Bert’s friends) + Deep-Biaffine”

- What stills remains interesting is now back to **the data resources**.

Resources: Overview

There have been 6 CoNLL shared tasks related with dependency parsing:

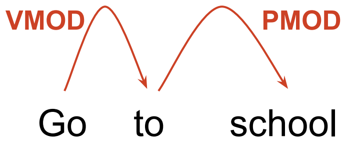
2018	Multilingual Parsing from Raw Text to Universal Dependencies	multilingual	Universal Dependencies
2017	Multilingual Parsing from Raw Text to Universal Dependencies	multilingual	Universal Dependencies
2009	Syntactic and Semantic Dependencies in Multiple Languages	multilingual	Language-specific
2008	Joint Parsing of Syntactic and Semantic Dependencies	English	Language-specific
2007	Dependency Parsing: Multilingual & Domain Adaptation	multilingual	Language-specific
2006	Multi-Lingual Dependency Parsing	multilingual	Language-specific

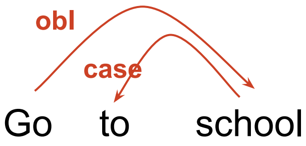
Resources: Overview

There can be multiple ways of **constructing dependency trees**, for example, for English, multiple ways of converting from constituency trees to dependencies:

Penn2Malt -> LTH-Convertor (for CoNLL tasks) ;; SD (stanford) -> UD

There are many things that **need to be specified**:





It's hard to say which one is "correct" or "better", but we need to arrive at something consistent.

Resources: UD

Universal dependencies: https://universaldependencies.org/

The data is released through LINDAT/CLARIN.

Cover 100+ languages.

- The next release (v2.9) is scheduled for November 15, 2021 (data freeze on November 1).
- Version 2.8 treebanks are available at http://hdl.handle.net/11234/1-3687. 202 treebanks, 114 languages, released May 15, 2021.
- Version 2.7 treebanks are archived at http://hdl.handle.net/11234/1-3424. 183 treebanks, 104 languages, released November 15, 2020.
- Version 2.6 treebanks are archived at http://hdl.handle.net/11234/1-3226. 163 treebanks, 92 languages, released May 15, 2020.
- Version 2.5 treebanks are archived at http://hdl.handle.net/11234/1-3105. 157 treebanks, 90 languages, released November 15, 2019.
- Version 2.4 treebanks are archived at http://hdl.handle.net/11234/1-2988. 146 treebanks, 83 languages, released May 15, 2019.
- Version 2.3 treebanks are archived at http://hdl.handle.net/11234/1-2895. 129 treebanks, 76 languages, released November 15, 2018.
- Version 2.2 treebanks are archived at http://hdl.handle.net/11234/1-2837. 122 treebanks, 71 languages, released July 1, 2018.
- Version 2.1 treebanks are archived at http://hdl.handle.net/11234/1-2515. 102 treebanks, 60 languages, released November 15, 2017.
- Version 2.0 treebanks are archived at http://hdl.handle.net/11234/1-1983. 70 treebanks, 50 languages, released March 1, 2017.
  - Test data 2.0 are archived at http://hdl.handle.net/11234/1-2184. 81 treebanks, 49 languages, released May 18, 2017.
- Version 1.4 treebanks are archived at http://hdl.handle.net/11234/1-1827. 64 treebanks, 47 languages, released November 15, 2016.
- Version 1.3 treebanks are archived at http://hdl.handle.net/11234/1-1699. 54 treebanks, 40 languages, released May 15, 2016.
- Version 1.2 treebanks are archived at http://hdl.handle.net/11234/1-1548. 37 treebanks, 33 languages, released November 15, 2015.
- Version 1.1 treebanks are archived at http://hdl.handle.net/11234/LRT-1478. 19 treebanks, 18 languages, released May 15, 2015.
- Version 1.0 treebanks are archived at http://hdl.handle.net/11234/1-1464. 10 treebanks, 10 languages, released January 15, 2015.
- In general, we intend to have regular treebank releases every six months. The v2.0 and v2.2 releases were brought forward because of their usage in the CoNLL 2017 and 2018 Multilingual Parsing Shared Tasks.

Update every half year.

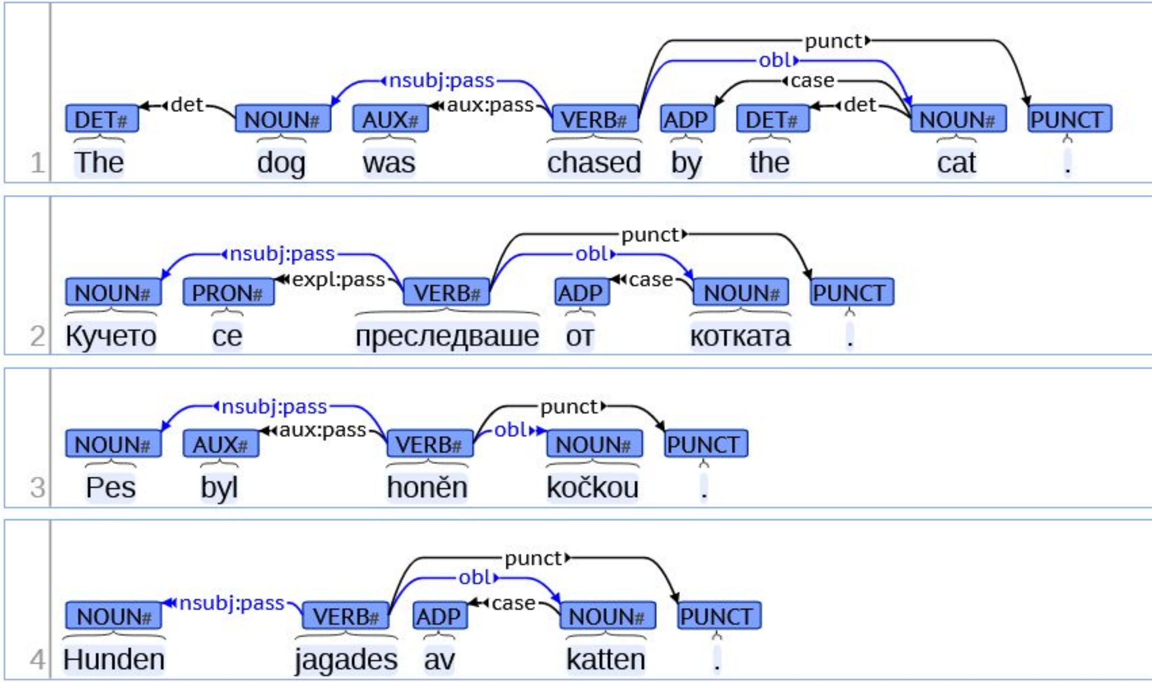
Resources: UD

The table lists the **37 universal syntactic relations** used in UD v2. It is a revised version of the relations originally described in *Universal Stanford Dependencies: A cross-linguistic typology* (de Marneffe *et al.* 2014).

	Nominals	Clauses	Modifier words	Function Words
Core arguments	nsubj obj iobj	csubj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod* discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	**MWE**	**Loose**	**Special**	**Other**
conj cc	fixed flat compound	list parataxis	orphan goeswith reparandum	punct root dep

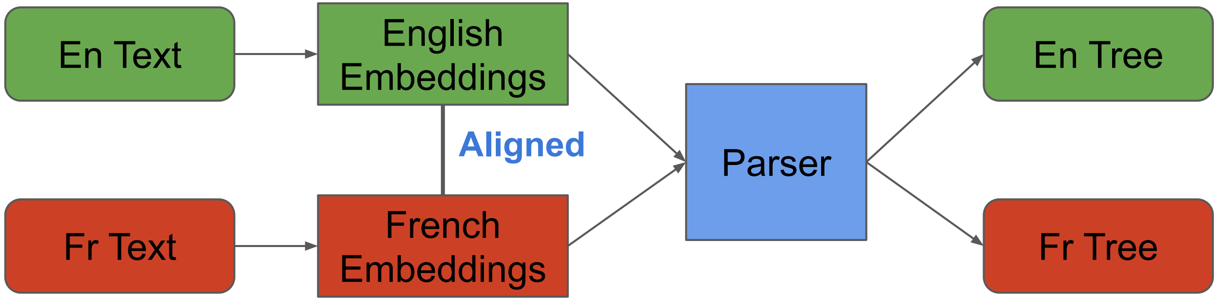
Resources: UD

“Universal Dependencies (UD) is a project that is developing ***cross-linguistically consistent*** *treebank annotation for many languages, with the goal of facilitating multilingual parser development, cross-lingual learning, and parsing research from a language typology perspective.”*



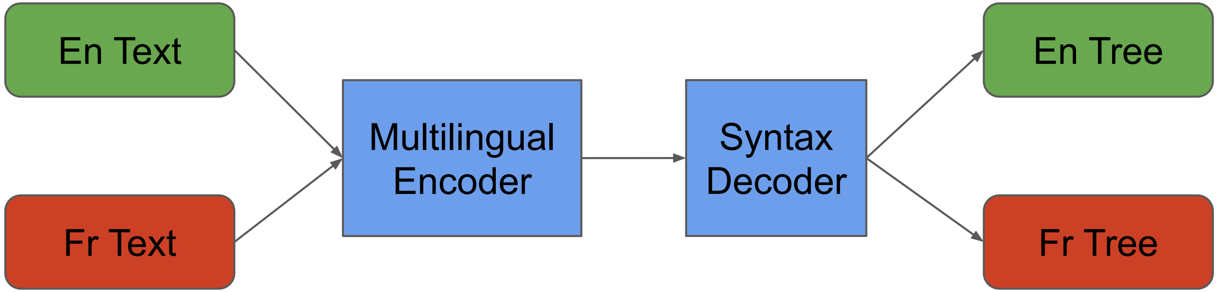
Resources: UD + Cross-lingual Transfer

- **Cross-lingual** transfer: **Transfer** from high-resource languages to low-resource ones. (* UD provides a great test-bed for this!)
- One specific interest thing is **zero-shot transfer**, where no trees for the target languages are available.
- This can be achieved with aligned multilingual word embeddings, or ...



Resources: Multilingual contextualized representations

…, or simply **multilingual contextualized pre-trained encoders**, which have been shown quite effective for cross-lingual transfer (Wu and Dredze, 2019).



Still an interesting question: how BERT/mBERT encodes syntax so that simply multilingual pre-training seems to be able to "align" syntactic information?

Resources: Problems

However, UD is not without problems:

- There can be **consistency problems** (an open collaboration project).
- Many treebanks are **converted** from constituency treebanks rather than from directly dependency annotations.
- **English-centric** (remember it’s derived from Stanford Dependencies).
- Are the UD choices the most reasonable ones?
  - Arguments and Adjuncts (Przepiórkowski and Patejuk, 2018)
  - Coordinate Structures (Kanayama et al., 2018)

Summary for dependency parsing

- **Algorithms**: graph-based & transition-based
- **Models**: feature-based -> NN -> pre-training
- **Resources**: cross-lingual consistent UD

- Nice online demo: http://lindat.mff.cuni.cz/services/udpipe/
- Nice parsers: stanza, udpipe, udify
- More on UD: https://universaldependencies.org/ ; EACL17 Tutorial

- Questions?