CS769 Advanced NLP

Syntactic Parsing 2: Dependency Parsing

Junjie Hu

UNIVERSITY OF WISCONSIN–MADISON

Slides adapted from Zhisong

https://junjiehu.github.io/cs769-spring23/

Goal for Today

- **Transition-based parsing (Discriminative)**:
  - Decoding algorithm: **shift-reduce (Aho & Ullman, 1972)**
  - Modeling: feature-based, neural network, pre-trained models

- **Graph-based parsing (Discriminative)**:
  - Decoding algorithm: **Chu-Liu-Edmonds (1965, 1967)**
  - Modeling: feature-based, neural network, pre-trained models

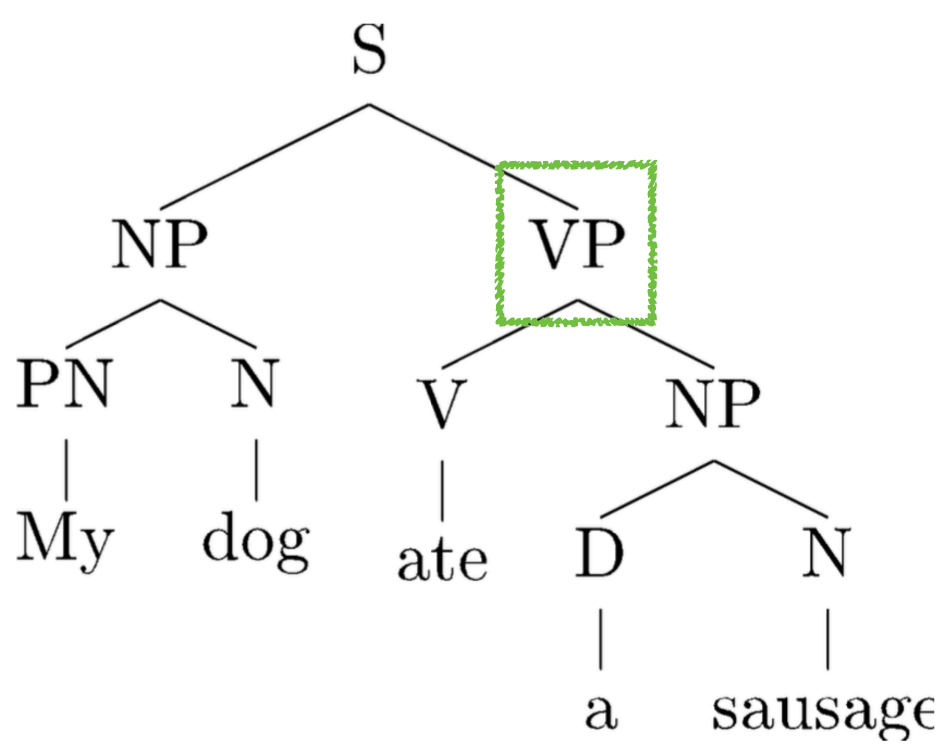
- **Data resources**: labeled data for supervised prediction, or zero-shot prediction
  - Universal dependencies
  - Cross-lingual transfer learning

Recap: Syntactic Parsing

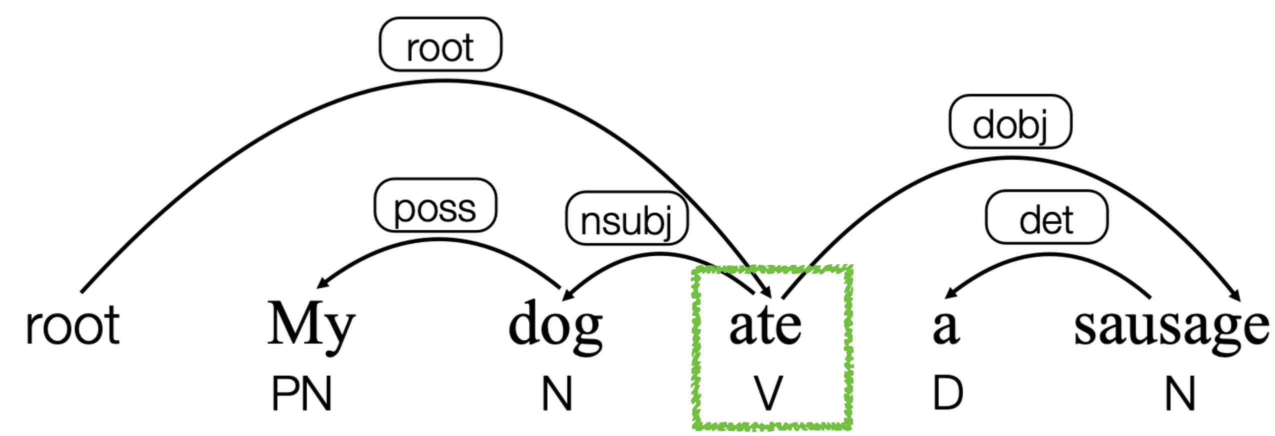
- Two types of linguistic structures:

Constituency tree: internal nodes for phrases
Dependency tree: only input words as nodes

Dependency-tree properties:
1. No multiple edges between two words
2. Each word (except root) has only one head
3. No cycles
4. (Optional) Projective (i.e., no cross edges)



Constituency (aka phrase structure) tree:
Focus on the structure of the sentence



Dependency tree:
Focus on relations between words

Predicting relations between two words

- Dependency tree consists of (*head*, relation, **dependent**) triples

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

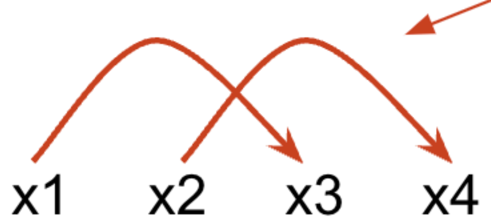
Universal Dependency relations

(de Marneffe et al., 2014)

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	**Description**
CONJ	Conjunct
CC	Coordinating conjunction

Property of Parse 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.

Abstraction of Dependency Parsing

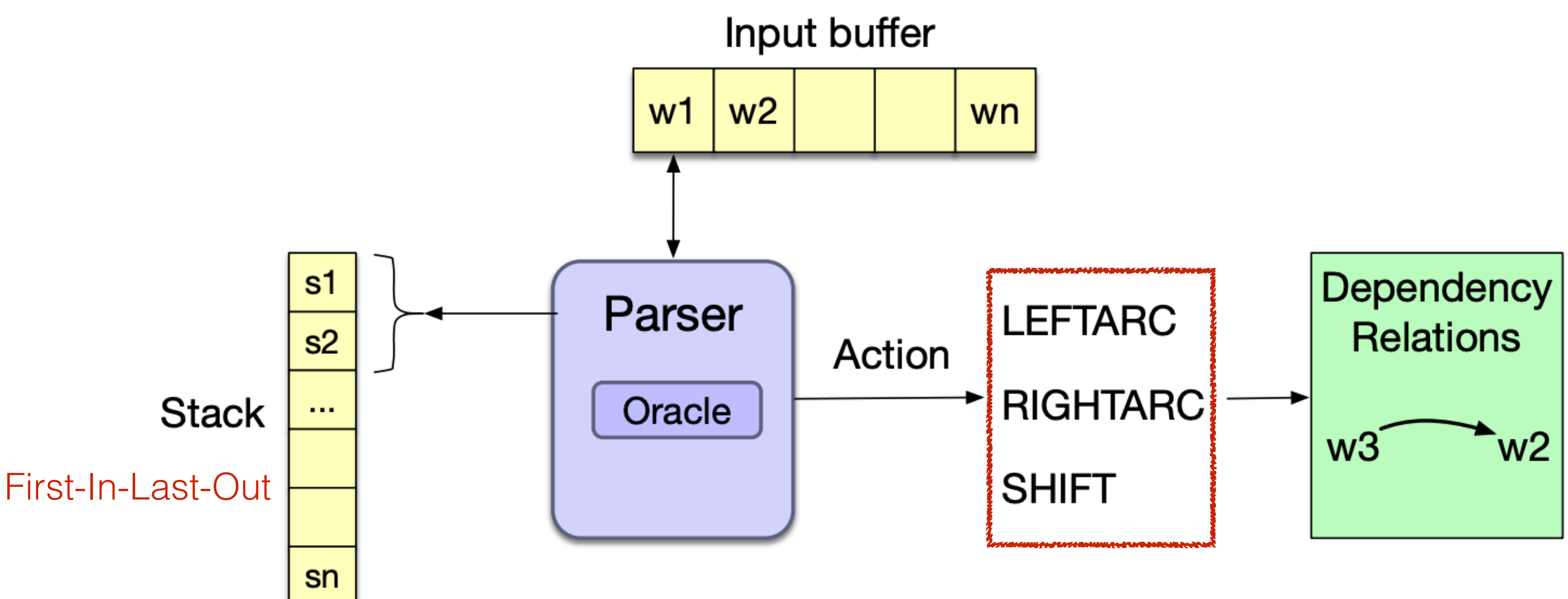
- **Link prediction** task

- Given an ordered list of nodes (words), predict the links between nodes, under some constraints (no multiple links, single-head, acyclic, projective)

Transition-based Parsing

Transition-based Parsing

- Developed for analyzing programming languages (Aho, Ullman, 1972)
- **Stack**: data structure to build the parse tree. Initially empty.
- **Input buffer**: stores tokens to be parsed. Initially contains the sentence.
- **Relation set**: stores the predicted arcs. Initially empty.
- Parser takes actions on the parse by a predictor called **Oracle**.



Decoding: Shift-Reduce Algorithm

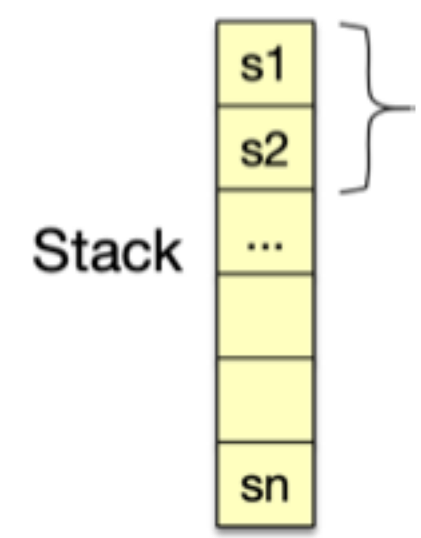
MaltParser (Nivre et al. 2006)

- The parser goes through the sentence (buffer) from left to right

- At each time, the oracle makes a **transition** action based on the current **state (a.k.a. configuration)** of the stack, buffer, relation set:

$$\text{Action}^* = \max P(\text{Action}|\text{State})$$

  - **LeftArc**: add a left arc from the first word at the top of the stack to the second word (s2 $\leftarrow$ s1); remove the dependent (s2)
  - **RightArc**: add a right arc from the second word at the top of the stack to the first word (s2 $\rightarrow$ s1); remove the dependent (s1)
  - **Shift**: Shift the word from the buffer to the stack



Decoding: Shift-Reduce Algorithm

MaltParser (Nivre et al. 2006)

- The Oracle for greedily selecting the appropriate transition is trained by supervised learning on labeled data.
- During testing, we simply apply the Oracle's predictions to get the parse tree
- Runtime complexity: $O(n)$ where n=word length

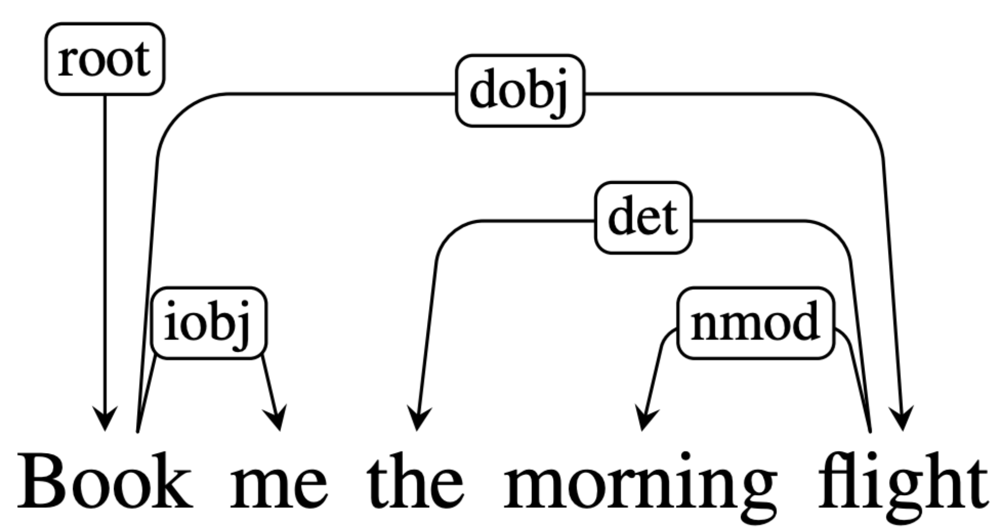
function DEPENDENCYPARSE(*words*) **returns** dependency tree

state ← {[root], [*words*], [] } ; initial configuration
while *state* **not final**
  t ← ORACLE(*state*) ; choose a transition operator to apply
  state ← APPLY(*t*, *state*) ; apply it, creating a new state
return *state*

Shift-Reduce (Example)

Example: Book me the morning flight

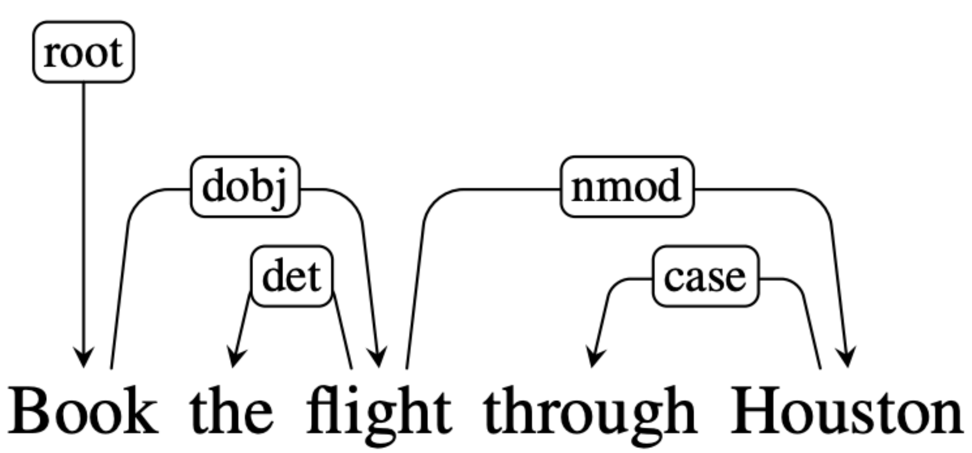
Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, 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	



Learning: Create Training Data

- Given a sentence and a dependency tree, simulate the shift-reduce process to create **(state, action)** pairs
  - Choose **LeftArc** if the top two words on the stack has a left arc in the ground-truth parse tree.
  - Choose **RightArc** if **(1)** the top two words on the stack has a right arc in the ground-truth parse tree **and (2)** all the dependents of top first word has been assigned.
  - Otherwise, choose **Shift**.

(sentence, tree) pairs:



$$P(\text{Action}|\text{State}) = \text{score}(\text{Action}, \text{State})$$

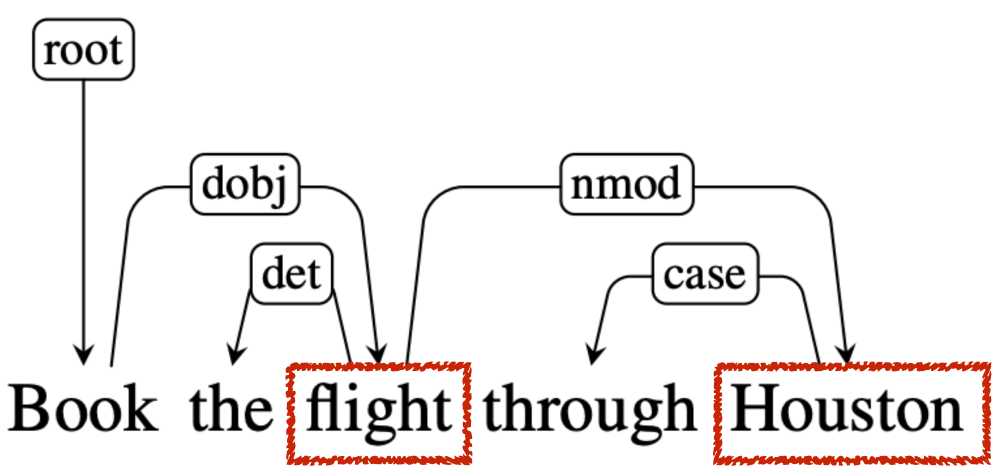
Learning: Create Training Data

- Why the RightArc's condition is important?
  - Need to make sure “flight” has assigned its dependents; otherwise “flight” will be reduced and its dependent “Houston” cannot find the arc to “flight”

State:

Stack	Word buffer	Relations
[root, book, flight]	[through, Houston]	(the ← flight)

(sentence, tree) pairs:



Action: Shift? or (book→flight)?

Hand-crafted Features

- Extract the features from the top two words (s1, s2) on the stack, and the words (b1, b2, …) on the buffer.
- Train a **logistic regression** model on the scores weighted by features

$$P(\text{Action}|\text{State}) = \text{softmax}(Wf(\text{Action}, \text{State}) + b)$$

Feature template:
(Uni-gram features)

$$\langle s_1.w, op\rangle, \langle s_2.w, op\rangle \langle s_1.t, op\rangle, \langle s_2.t, op\rangle$$
$$\langle b_1.w, op\rangle, \langle b_1.t, op\rangle \langle s_1.wt, op\rangle$$

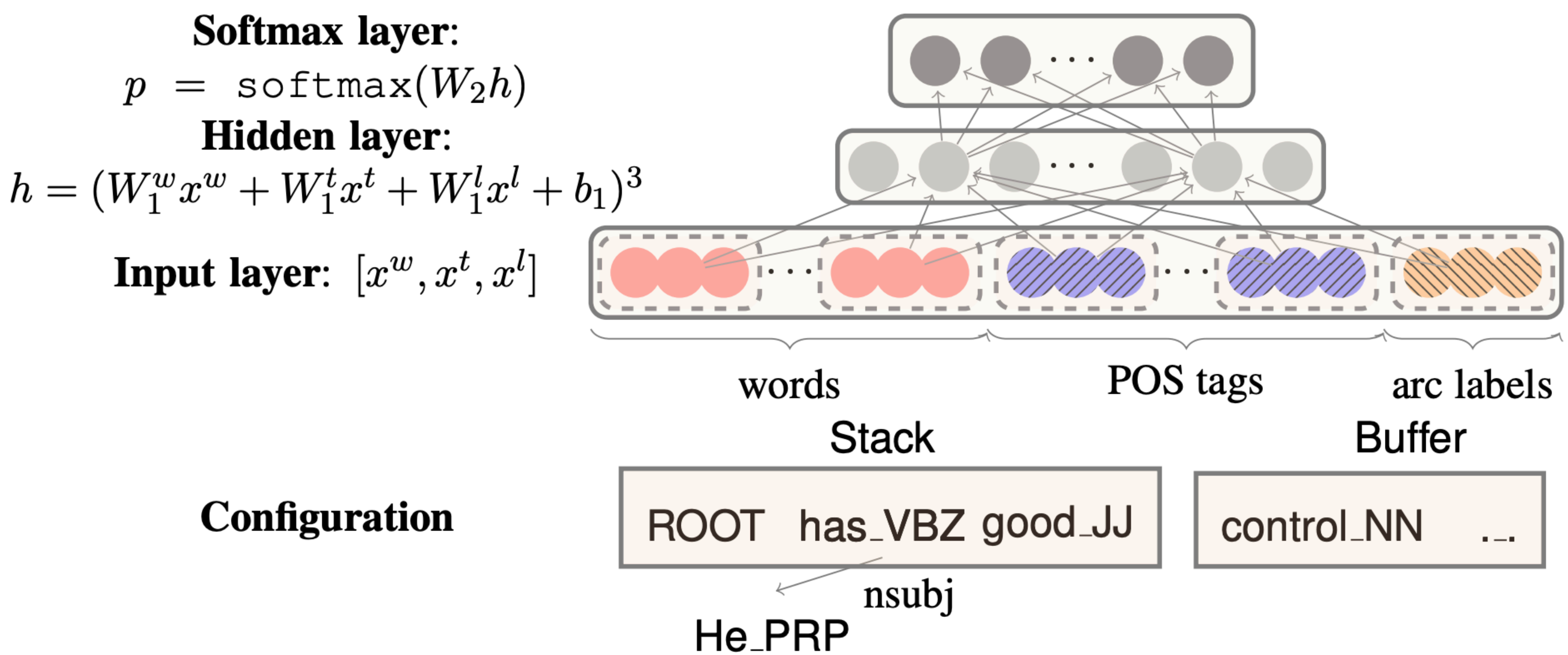
E.g., concrete features for **Shift** operator:

$$\langle s_1.w = \textit{flights}, op = \textit{shift}\rangle$$
$$\langle s_2.w = \textit{canceled}, op = \textit{shift}\rangle$$
$$\langle s_1.t = \textit{NNS}, op = \textit{shift}\rangle$$
$$\langle s_2.t = \textit{VBD}, op = \textit{shift}\rangle$$
$$\langle b_1.w = \textit{to}, op = \textit{shift}\rangle$$
$$\langle b_1.t = \textit{TO}, op = \textit{shift}\rangle$$
$$\langle s_1.wt = \textit{flightsNNS}, op = \textit{shift}\rangle$$

- Sparsity! Millions of features!

Neural Shift-Reduced Parser

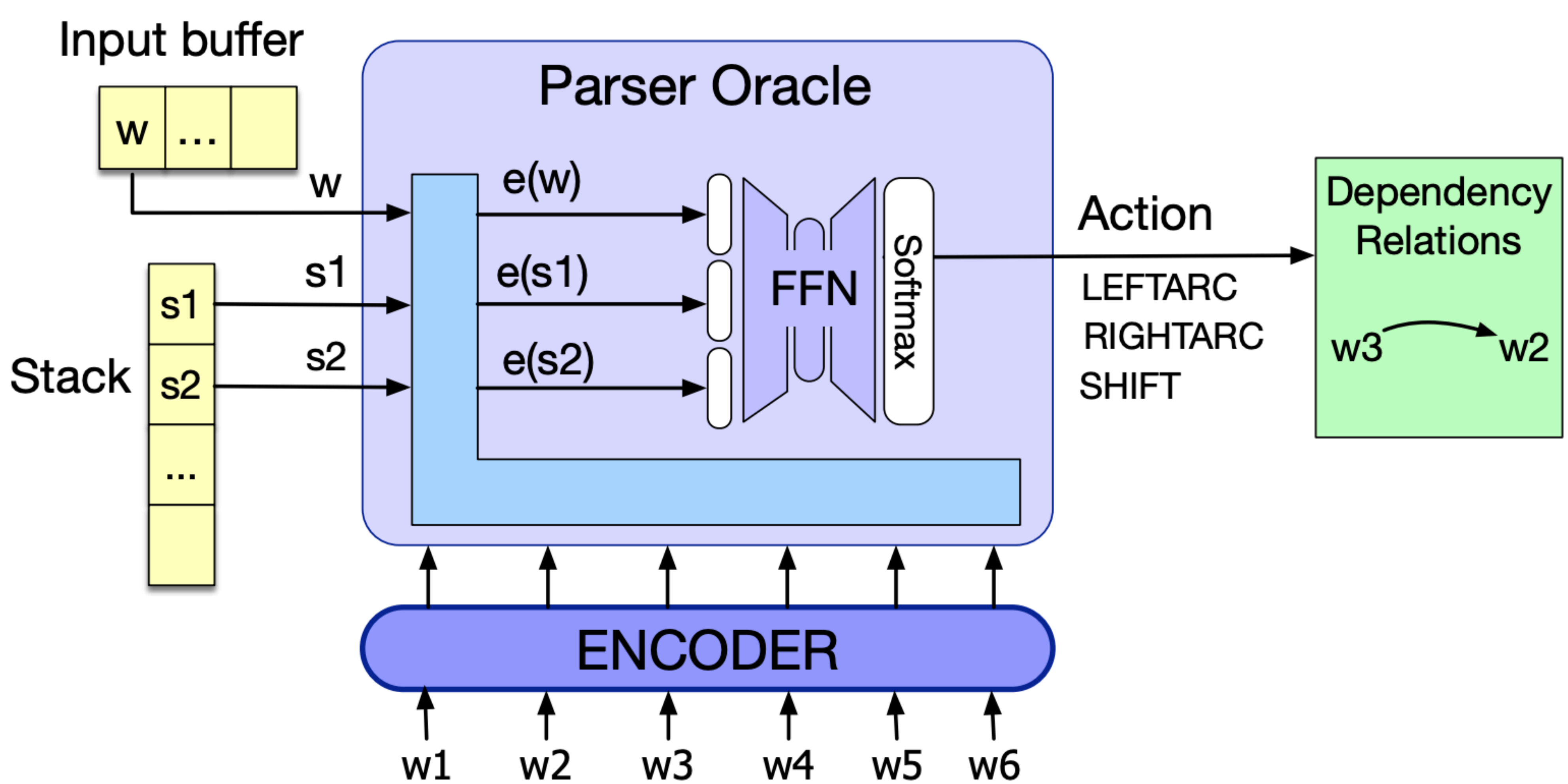
- Use embeddings to learn the feature of **configuration (state)** of the stack, buffer, relation, and compute the score between state and action by a feedforward network (Chen & Manning, 2014)



- **Words**: (1) top 3 words on stack/buffer; (2) 1st, 2nd leftmost/rightmost children of top two stack words; (3) left/right-most grandchildren of top two stack words -> 18 words in total
- **POS tags**: POS tags of selected words.
- **Arc labels**: arc labels of selected words excluding (1) those 6 words on stack/buffer

Neural Network

- Instead of just using word embedding, use a neural network to encode the sentence first, and then take the embeddings of the top two words on the stack and first word on the buffer to pass through a feedforward network. (Kiperwasser 2016, Kulmizev et al., 2019)



Shift-Reduce Parsing

- Pros:
  - Process input from left to right only once — Fast O(n)
  - Can make use of rich features from current configuration
- Cons:
  - Decisions are local and greedy
  - Cannot make use of information from right of the attachment point (i.e., limited look-ahead window).

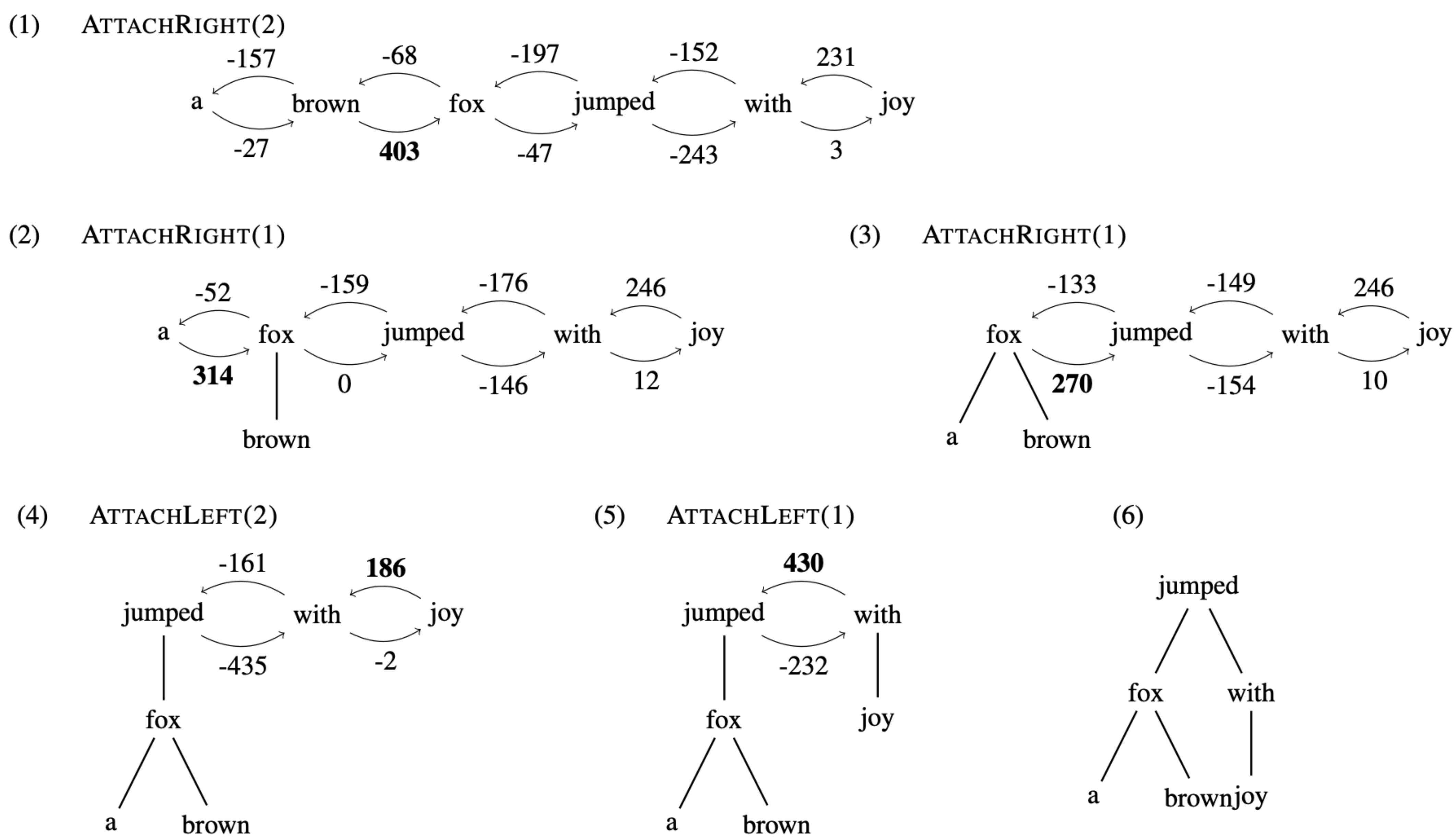
Alternative Algorithm: Beam Search

- Rather than greedily taking the **max** to predict the action from **P(Action|State)** at each time step, we can do **beam search**
  - Expand K top-scoring candidates for each action sequence in the beams, produce K^2 possible action sequences.
  - Pruning and keep only the top K scoring action sequences.

- Recover action mistakes at early states, usually perform better than greedy search, but slower.

Alternative Algorithm: 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 (**AttachRight & AttachLeft**), both create new arcs

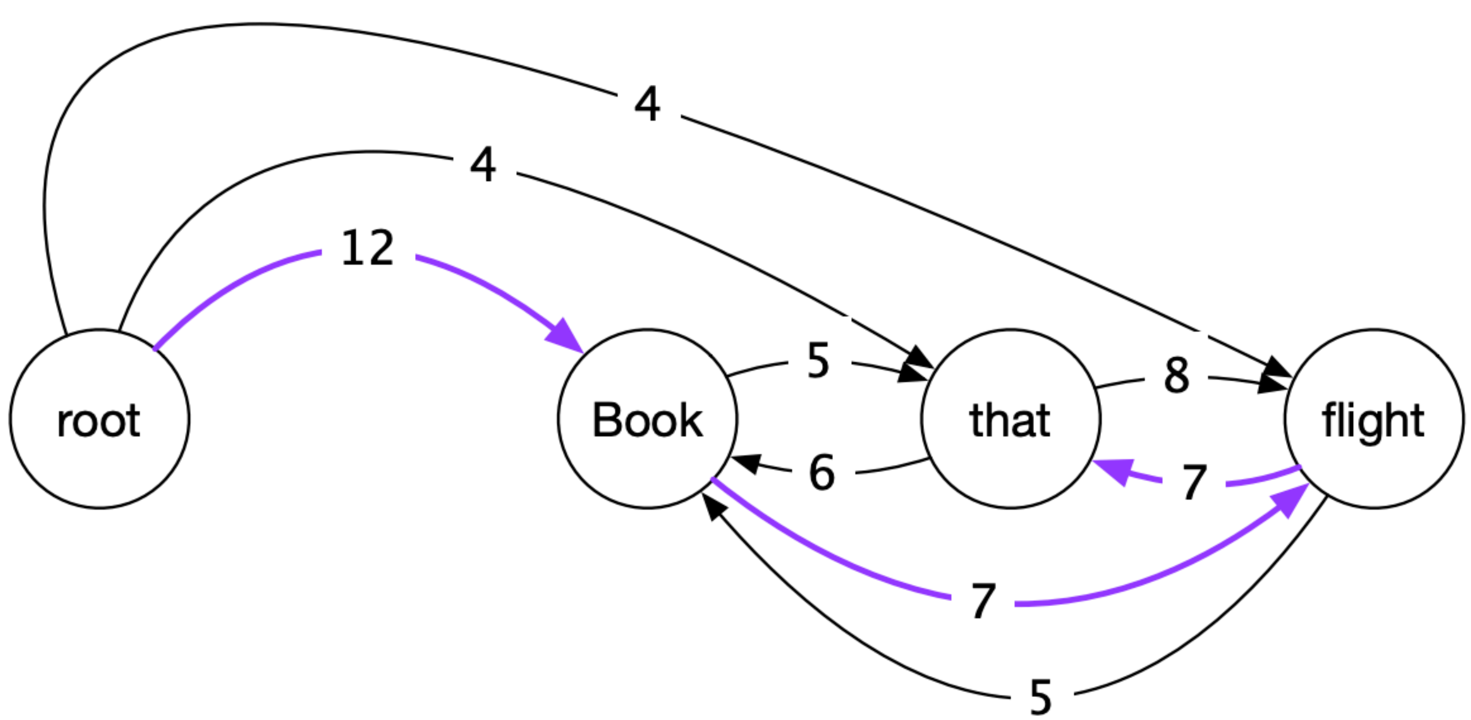


Graph-based Parsing

Maximum Spanning Tree (MST)

- Define a fully connected graph G with scores over edges
- Finding the best parse $\hat{T}(S)$ for a sentence S is equivalent to find a MST over G
- Assume the total score can be **(first-order) edge-factored**.

$$\hat{T}(S) = \underset{t \in \mathcal{G}_S}{\operatorname{argmax}} \operatorname{Score}(t, S) \qquad \operatorname{Score}(t, S) = \sum_{e \in t} \operatorname{Score}(e)$$



Decoding Algorithm: Chu-Liu Edmonds (CLE, 1965)

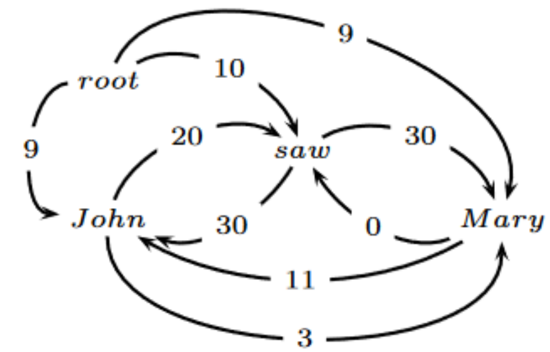
Chu-Liu-Edmonds(G, s)

Graph $G = (V, E)$

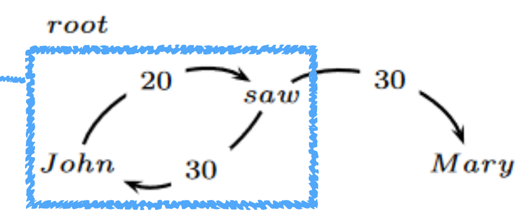
Edge weight function $s : E \rightarrow \mathbb{R}$

1. Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$
2. Let $G_M = (V, M)$
3. If G_M has no cycles, then it is an MST: return G_M
4. Otherwise, fi nd a cycle C in G_M
5. Let $G_C = \text{contract}(G, C, s)$
6. Let $\boldsymbol{y} = \text{Chu-Liu-Edmonds}(G_C, s)$
7. Find a vertex $x \in C$ s. t. $(x', x) \in \boldsymbol{y}, (x'', x) \in C$
8. return $\boldsymbol{y} \cup C - \{(x'', x)\}$

Greedy maximum

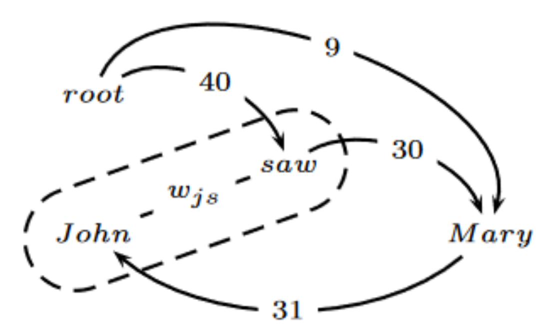


The first step of the algorithm is to find, for each word, the highest scoring incoming edge



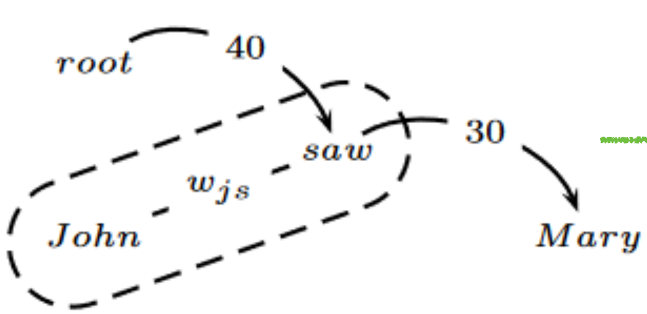
Contract

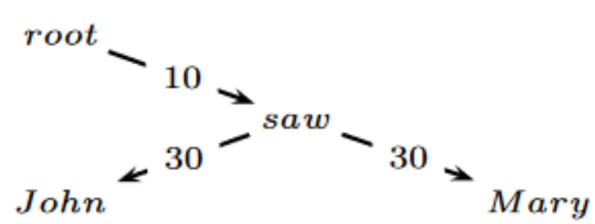
have a cycle, so we will contract it into a single node and recalculate edge weights according to Figure 3.



Rerun CLE

Recursive & Repack





Final parse tree

How to score each edge?

- Feature-based: with **manually designed features** and linear model
- (Early) **Neural network**: with atom input features and NN scorer
- (Recent) **Pre-trained encoder**: with contextualized representations

Hand-crafted Features

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

Table 1: Features used by system. p-word: word of parent node in dependency tree. c-word: word of child node. p-pos: POS of parent node. c-pos: POS of child node. p-pos+1: POS to the right of parent in sentence. p-pos-1: POS to the left of parent. c-pos+1: POS to the right of child. c-pos-1: POS to the left of child. b-pos: POS of a word in between parent and child nodes.

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

a) p-word, p-pos = <read, verb>; c-word, c-pos = <book, noun>; …

b) p-word, p-pos, c-word = <read, verb, book>; ….

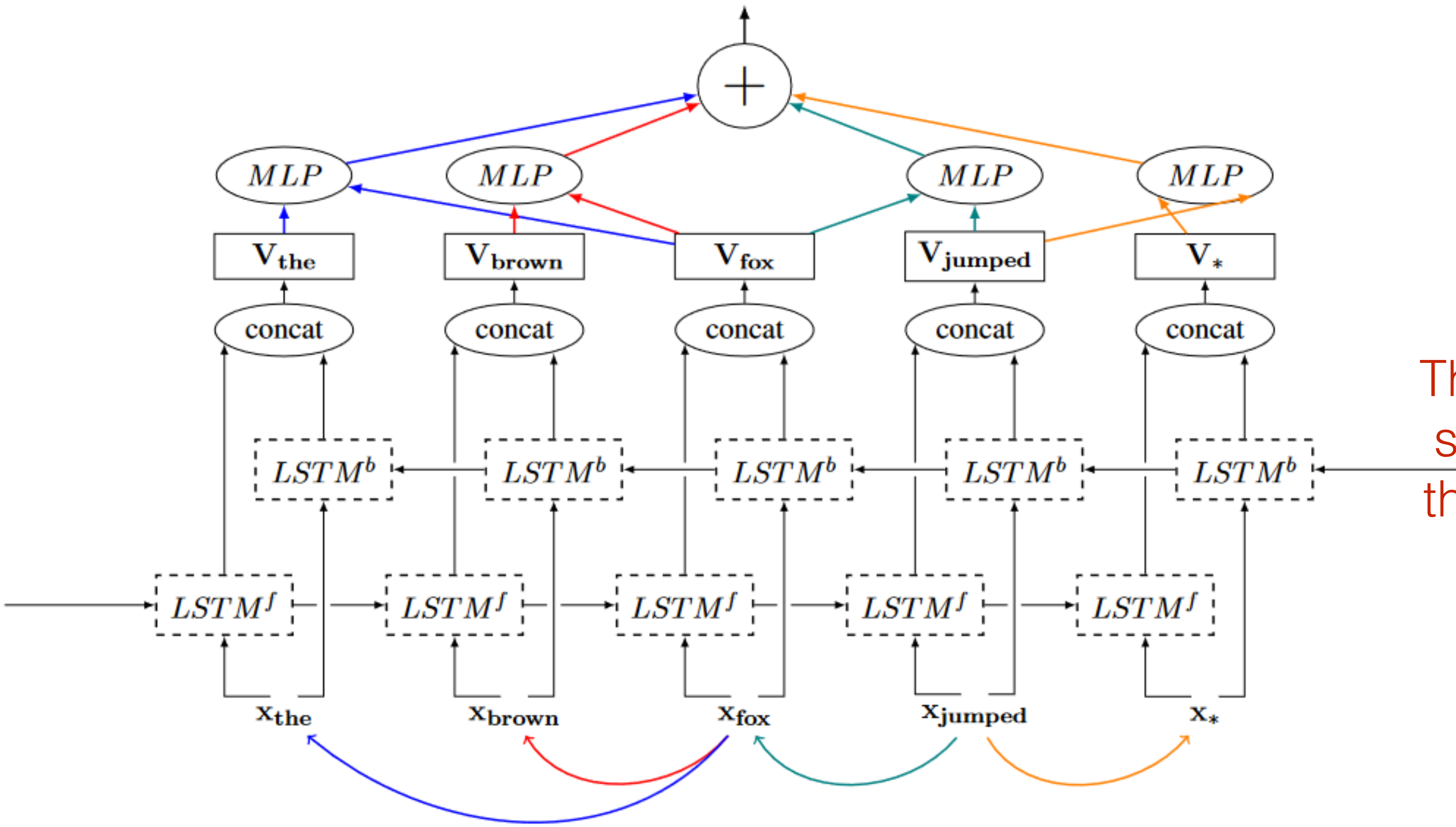
c) p-pos, p-pos+1, c-pos-1, c-pos = <verb, det, det, noun>; ….

$$\text{score}(e) = Wf(e)$$ - Sparsity! Millions of features!

Neural Network

(Kiperwasser & Goldberg, 2016)

- Remember that for parsing, we have full input sentence as the input! We can use **any NN encoders** (e.g., Bi-LSTM) to get the **word representations**, and then **edge representations** (fusion of two words).



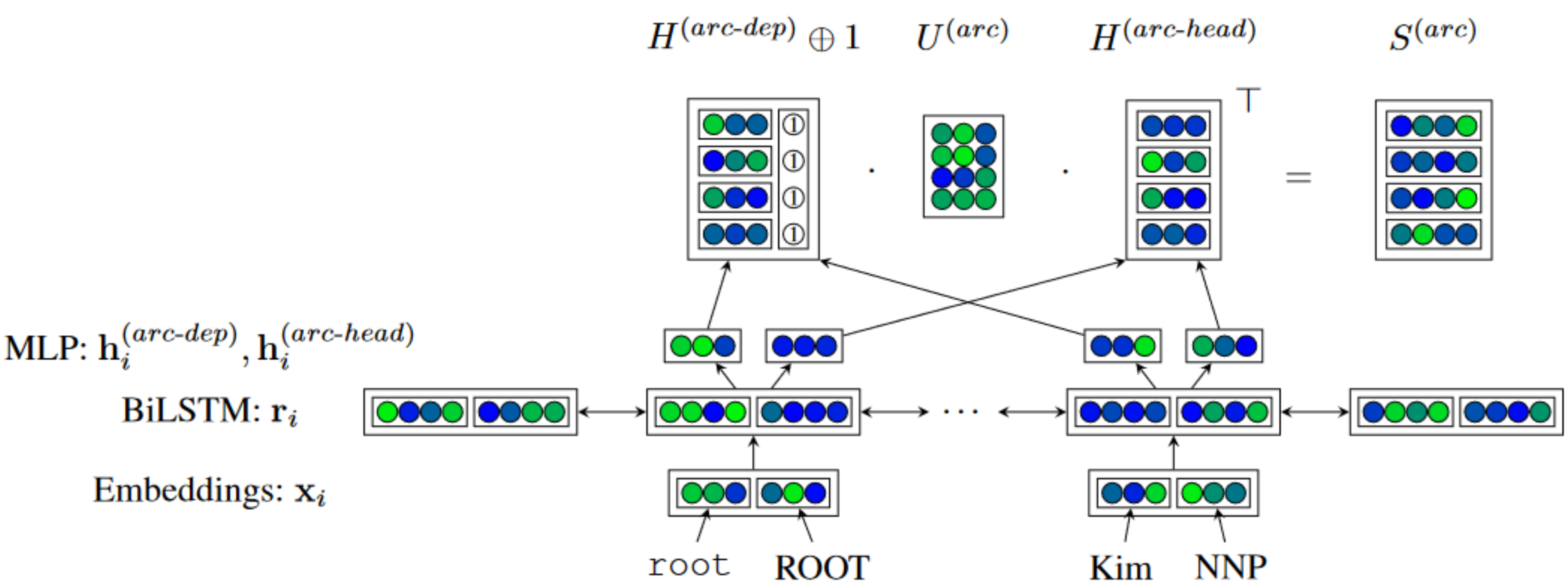
The inputs to the final scorer now contains the information of the full sentence.

Ground-truth edge

Deep Biaffine Scorer

(Ducat & Manning, 2017)

- Probably nowadays the "standard" parsing scorer architecture.
- Intuition:
  - For each word, learn two representations for the word being a head and a dependent
  - Biaffine function to compute scores between possible head-dip pairs



Progress over Years

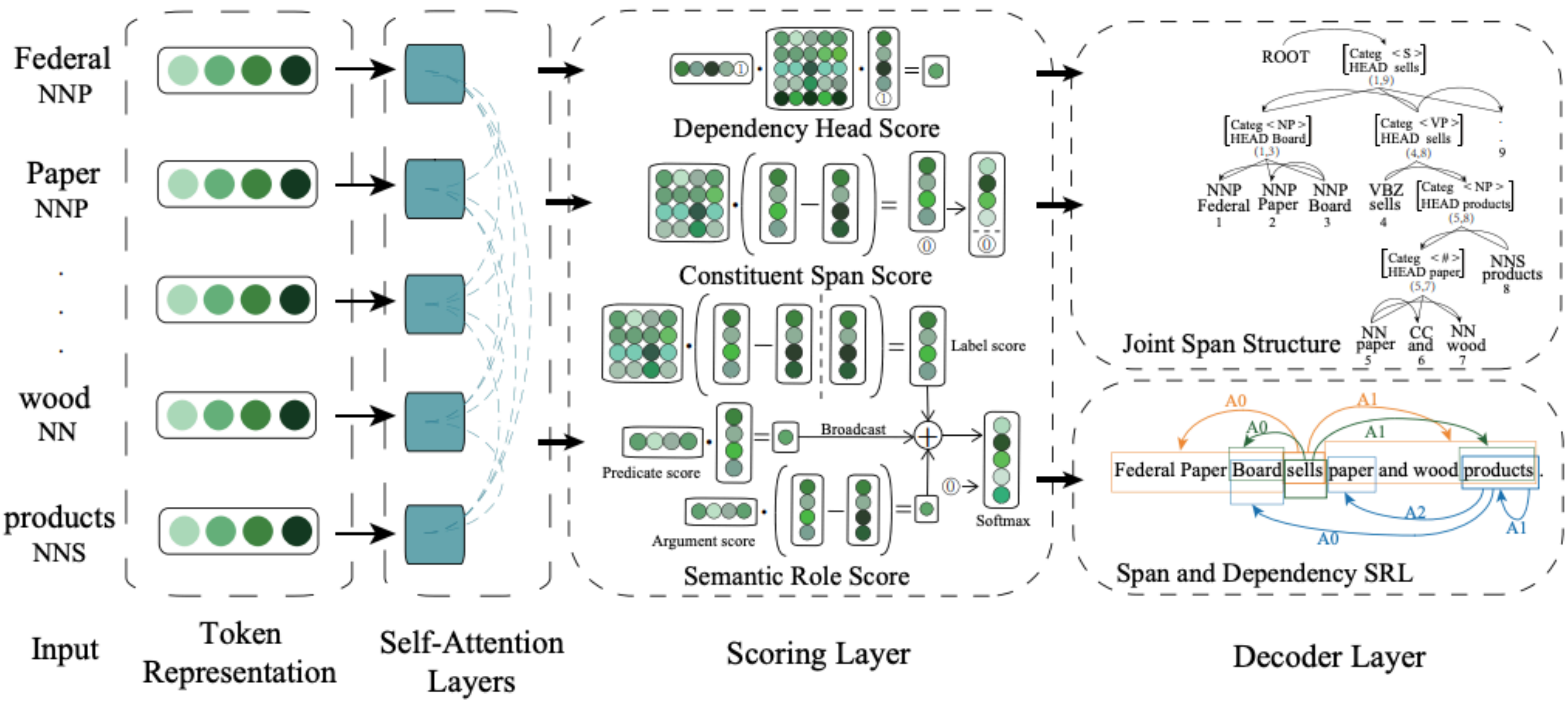
- Move towards fine-tuning pre-trained models (e.g., XLNet, Bert)
- Results: https://nlpprogress.com/english/dependency_parsing.html

Model	POS	UAS	LAS	Paper / Source	Code
Label Attention Layer + HPSG + XLNet (Mrini et al., 2019)	97.3	97.42	96.26	Rethinking Self-Attention: Towards Interpretability for Neural Parsing	Official
ACE + fine-tune (Wang et al., 2020)	-	97.20	95.80	Automated Concatenation of Embeddings for Structured Prediction	Official
HPSG Parser (Joint) + XLNet (Zhou et al, 2020)	97.3	97.20	95.72	Head-Driven Phrase Structure Grammar Parsing on Penn Treebank	Official
Second-Order MFVI + BERT (Wang et al., 2020)	-	96.91	95.34	Second-Order Neural Dependency Parsing with Message Passing and End-to-End Training	Official
CVT + Multi-Task (Clark et al., 2018)	97.74	96.61	95.02	Semi-Supervised Sequence Modeling with Cross-View Training	Official
CRF Parser (Zhang et al., 2020)	-	96.14	94.49	Efficient Second-Order TreeCRF for Neural Dependency Parsing	Official

Pre-trained Model + Multi-task

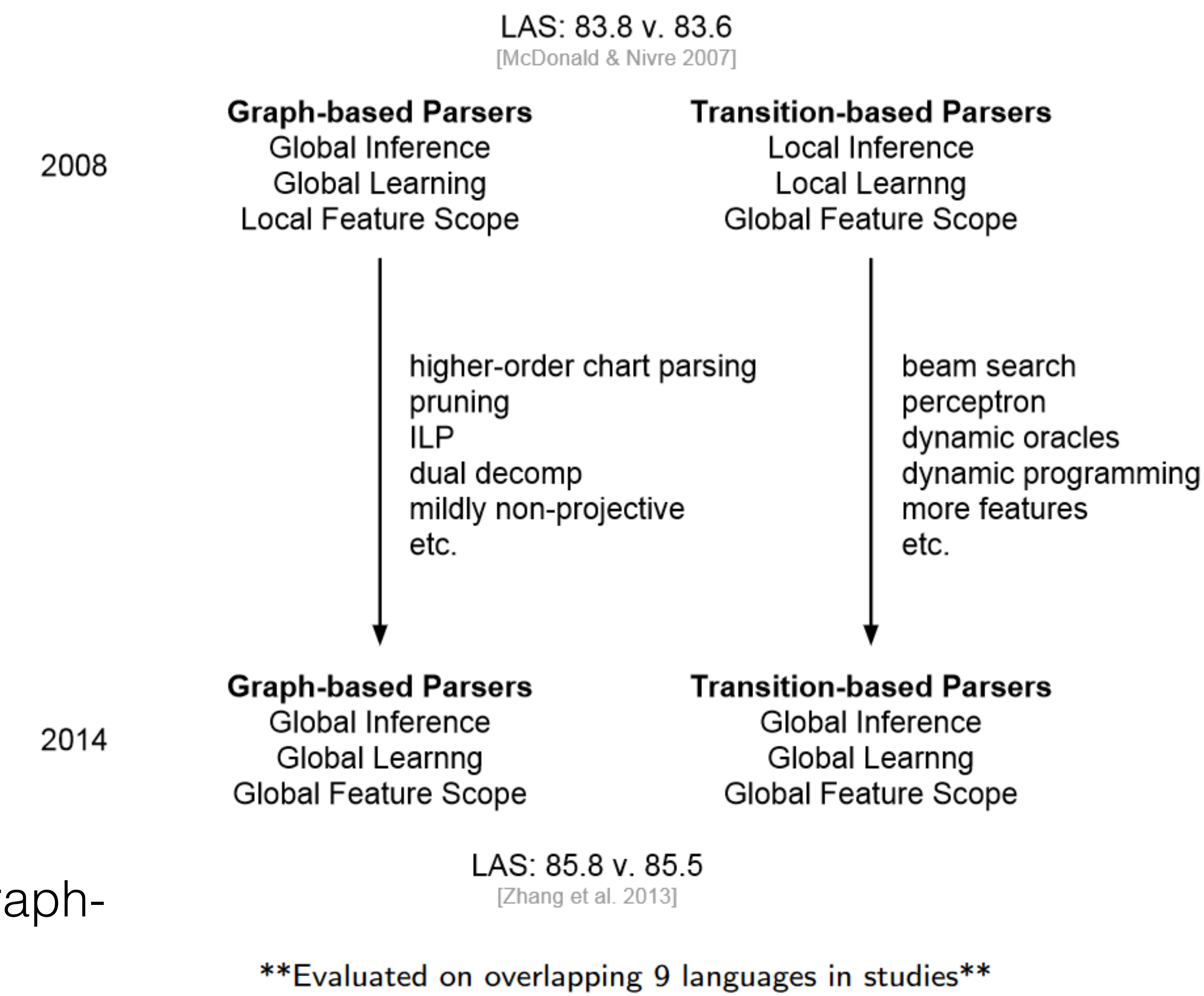
(Zhou et al. 2020)

- Jointly optimize dependency parsing, constituency parsing, semantic role labeling



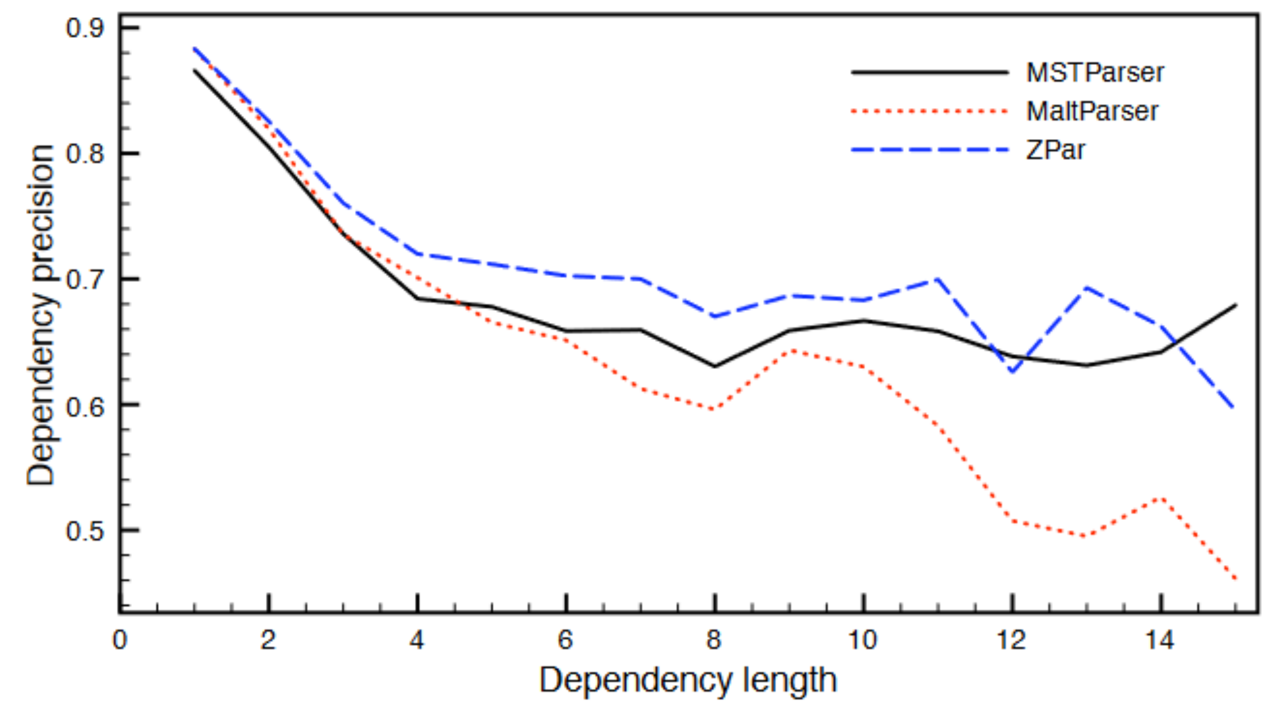
Transition vs Graph parsing

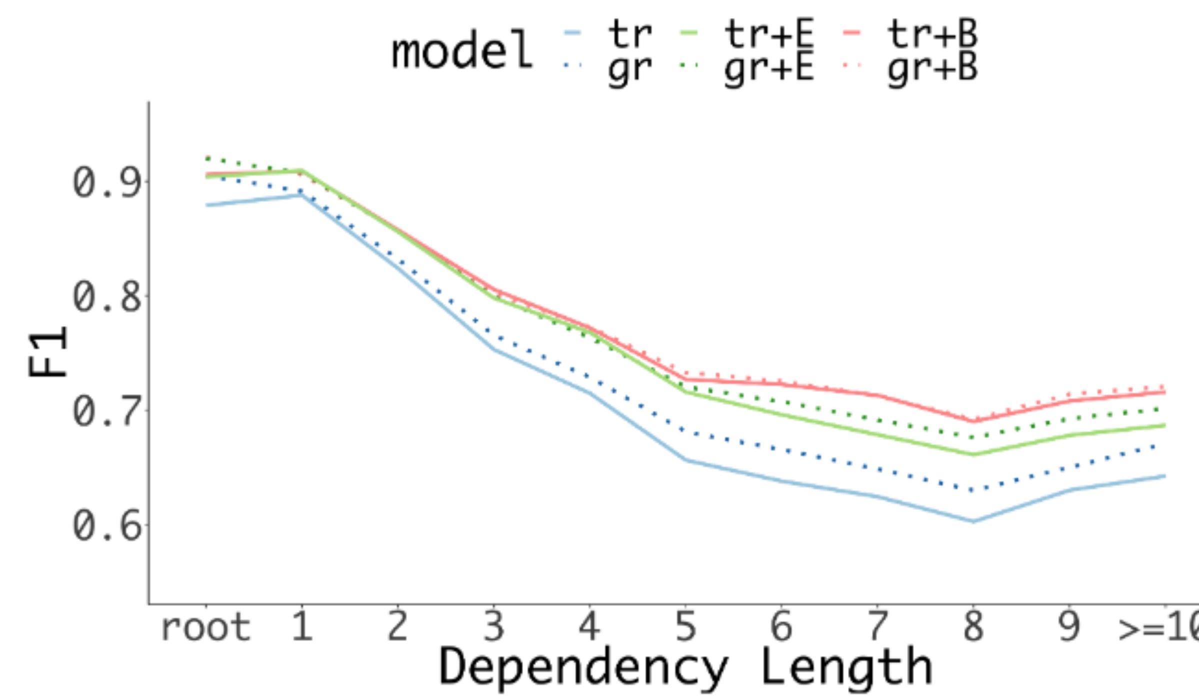
- Graph-based
  - Local factorization
  - Global inference
  - Mostly CLE $O(n^3)$
  - some $O(n\log n + n^2)$ (Gabow et al. 1986)
- Transition-based
  - Local normalization
  - Rich output features
  - Linear time $O(n)$ with shift-reduce
- Both can reach similar results, but Graph-based produces projective tree while transition-based may not



Transition vs Graph parsing

- Deep Contextualized Word Embeddings in Transition-based and Graph-based Dependency Parsing — **A Tale of Two Parsers Revisited** (Kulmizev et al. 2019)
    - Pre-trained model allows parsers to **pack info about global sentence structure** into local feature representations.
    - They benefit transition-based parsers more than graph-based parsers, making the two approaches **approximately equivalent** in terms of both accuracy and error profile.





Models after Pre-training

- A “huge change” to the parsing models?
  - **Maybe not** (only changing the scorers)?
  - The basic parsing paradigms are almost the same.

- However, this indeed brings changes:
  - Somehow **blur the distinctions** between graph-/transition-based methods
  - Computational complexity:
    - (CPU-oriented), graph $O(n^3)$ > transition $O(n)$
    - (GPU-oriented), graph (easier to parallelize) <= transition (not so GPU-friendly?)

- What stills remains interesting is not shifted towards the **data resources**.

Data Resources

Data Resources

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

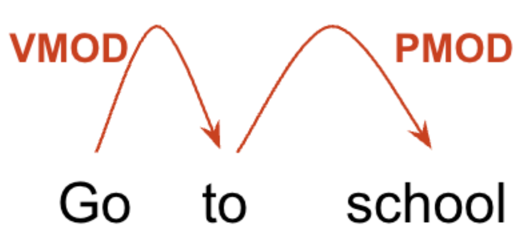
2018	Multilingual Parsing from Raw Text to Universal Dependencies	multilingual	Universal Dependency
2017	Multilingual Parsing from Raw Text to Universal Dependencies	multilingual	
2009	Syntactic and Semantic Dependencies in Multiple Languages	multilingual	Language specific
2008	Joint Parsing of Syntactic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

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.

Universal Dependency

- Updated every half year
- https://universaldependencies.org/

The data is released through LINDAT/CLARIN.

- 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.

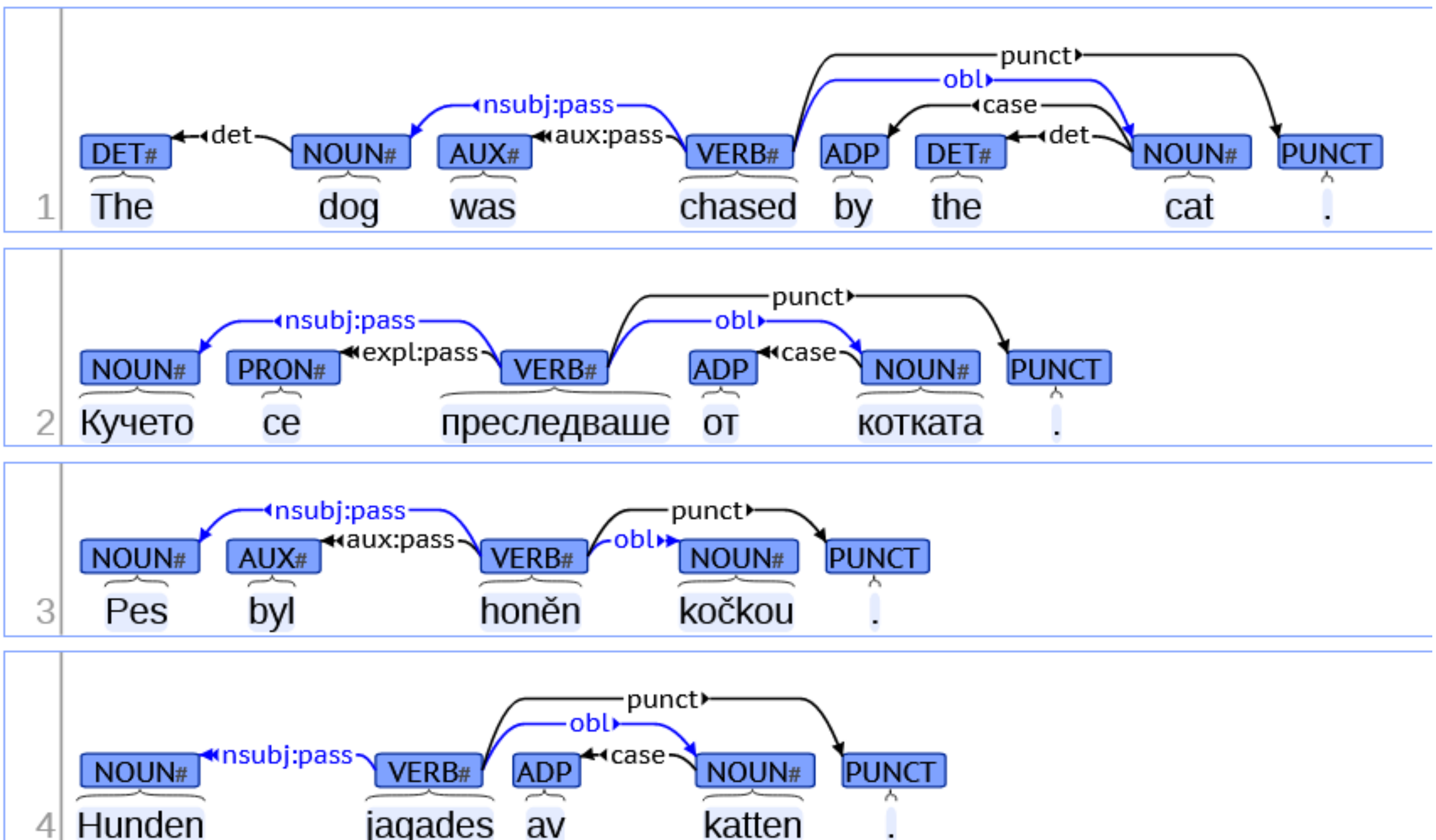
Universal Dependency

- 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

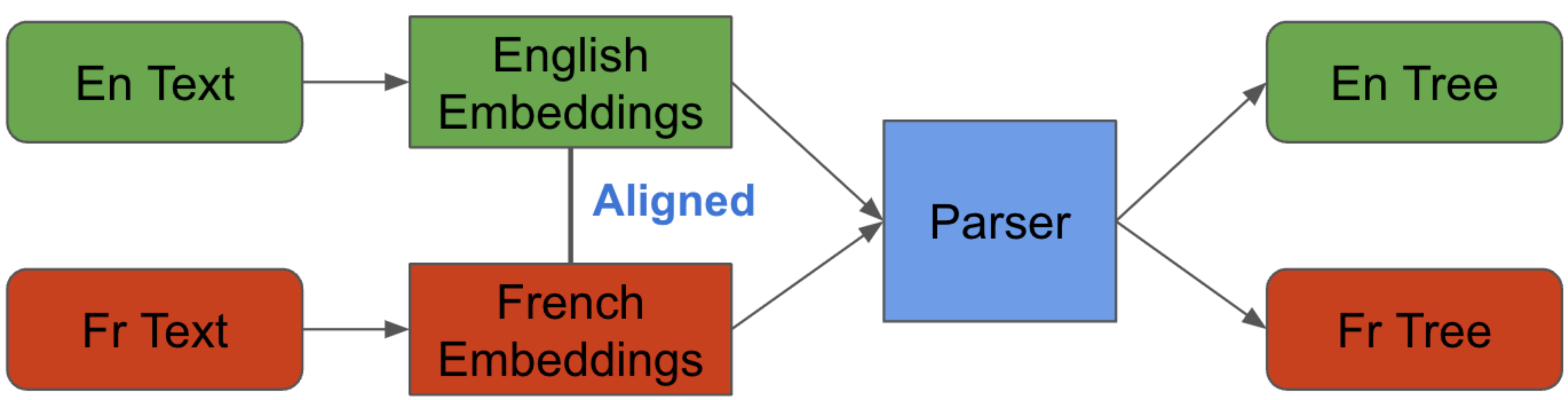
Universal Dependency

- “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.”



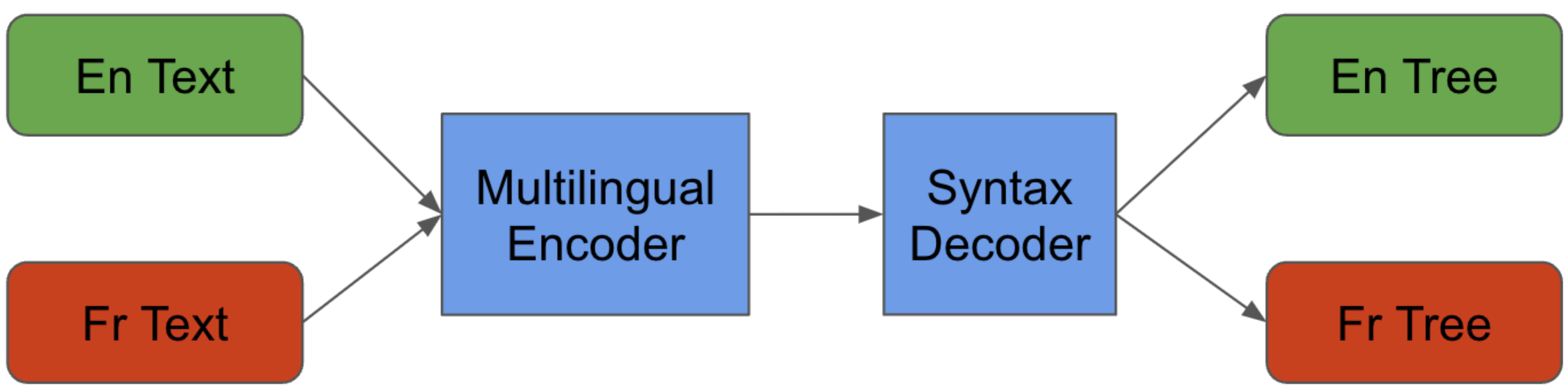
UD + Cross-lingual Transfer

- Dsfsd **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 ...



Multilingual Contextualized Representations

- 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?

UD is not Prefect

- 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)

Related Materials

- Online demo http://lindat.mff.cuni.cz/services/udpipe/
- Nice parsers: stanza, udpipe, udify
- More on UD: https://universaldependencies.org/
- EACL17 Tutorial: universaldependencies.org/eacl17tutorial/

Questions?