Université de Montréal

rali recherche appliquée en linguistique informatique

Mila

Natural Language Processing with Deep Learning

IFT6289, Winter 2022

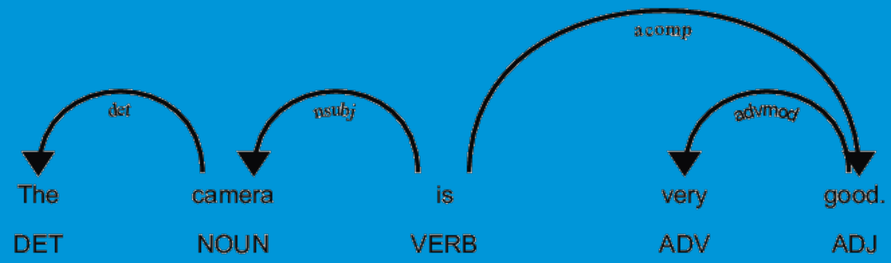
Lecture 16: Dependency Parsing

Bang Liu

Lecture outline

1. Syntactic Structure: Constituency and Dependency
2. Transition-based Dependency Parsing
3. Graph-based Dependency Parsing
4. Neural Dependency Parsing
5. Finding Syntax in Word Representations

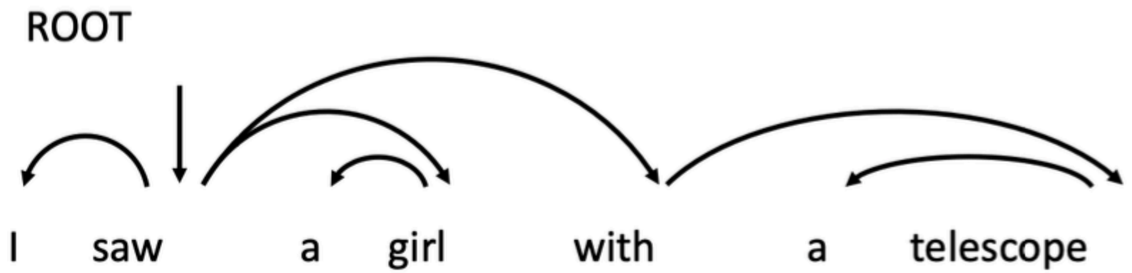
Syntactic Structure: Constituency and Dependency



What is dependency?

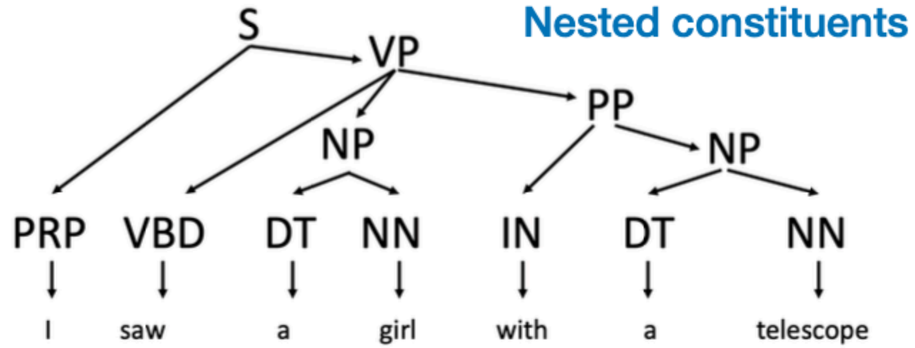
Two Most Common of Linguistic Tree Structures

- **Dependency Trees** focus on relations between words



Words directly linked to each other

- **Phrase Structure** models the structure of a sentence



Constituency Parse
generated from
Context Free Grammars
(CFGs)

Pāṇini's grammar of Sanskrit (c. 5th century BCE)

https://en.wikipedia.org/wiki/P%C4%81%E1%B9%87ini

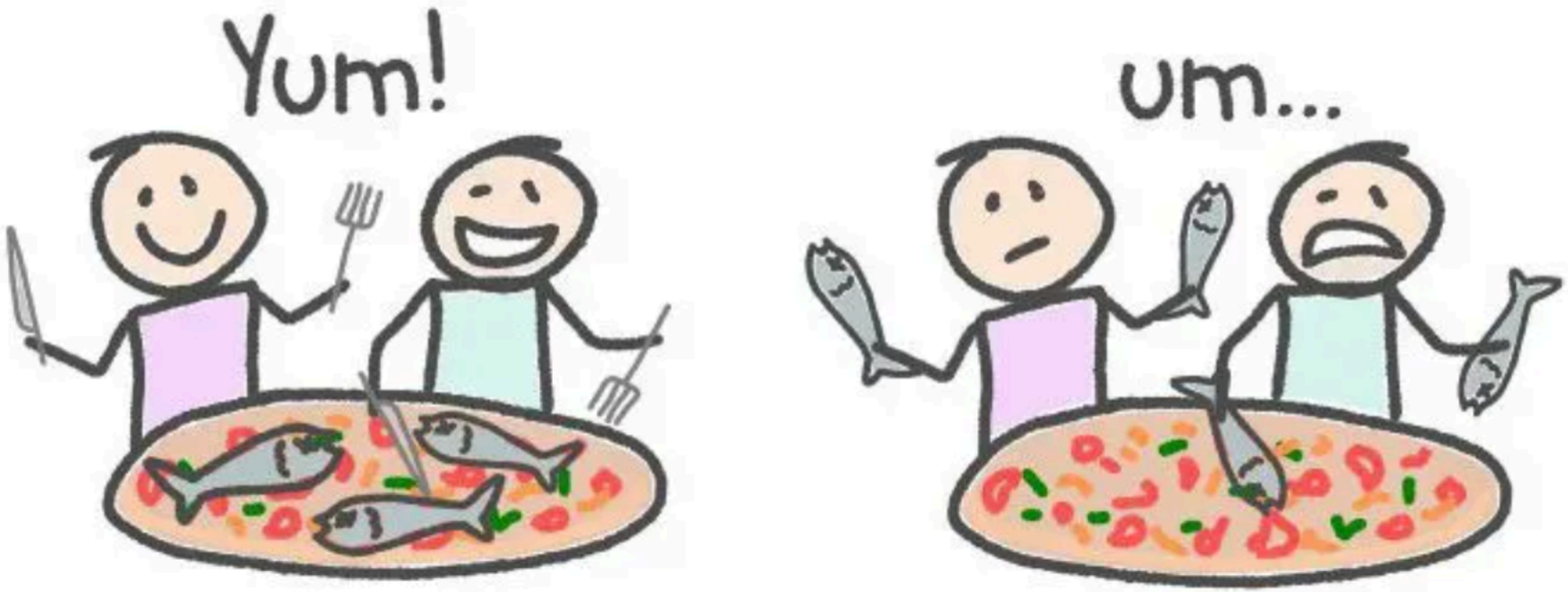
Dependency Grammar/Parsing History

- The idea of dependency structure goes back a long way
  - To Pāṇini's grammar (c. 5th century BCE)
  - Basic approach of 1st millennium Arabic grammarians
- Constituency/context-free grammars is a new-fangled invention
  - 20th century invention (R.S. Wells, 1947; then Chomsky)
- Modern dependency work often sourced to L. Tesnière (1959)
  - Was dominant approach in "East" in 20th Century (Russia, China, …)
  - Good for free-er word order languages
- Among the earliest kinds of parsers in NLP, even in the US:
  - David Hays, one of the founders of U.S. computational linguistics, built early (first?) dependency parser (Hays 1962)

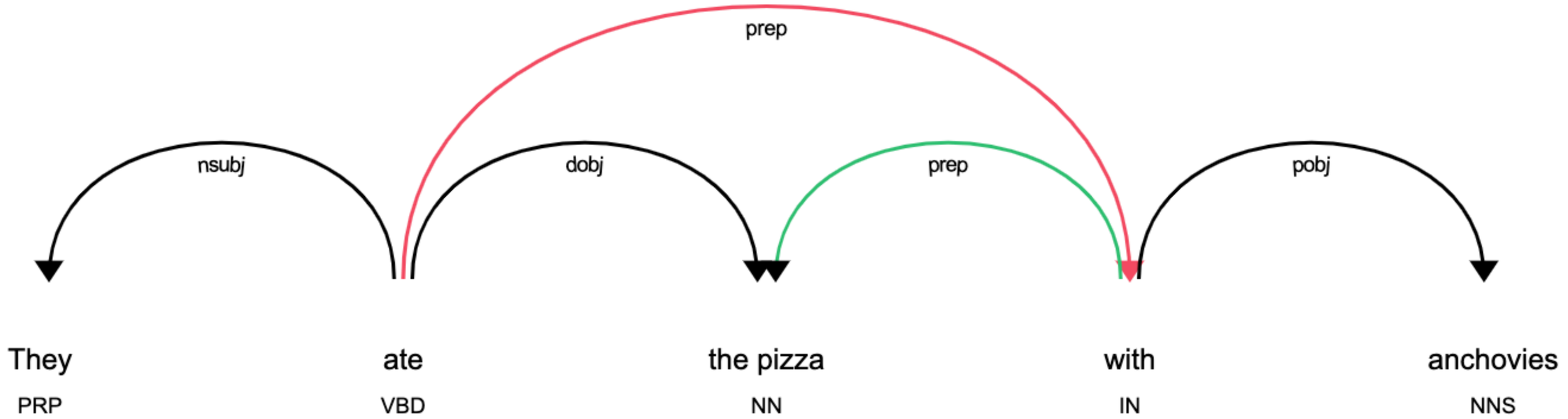
(slide credit: Stanford CS224N, Chris Manning)

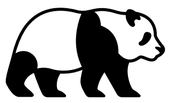
Disambiguation

“ They ate the pizza with anchovies ”

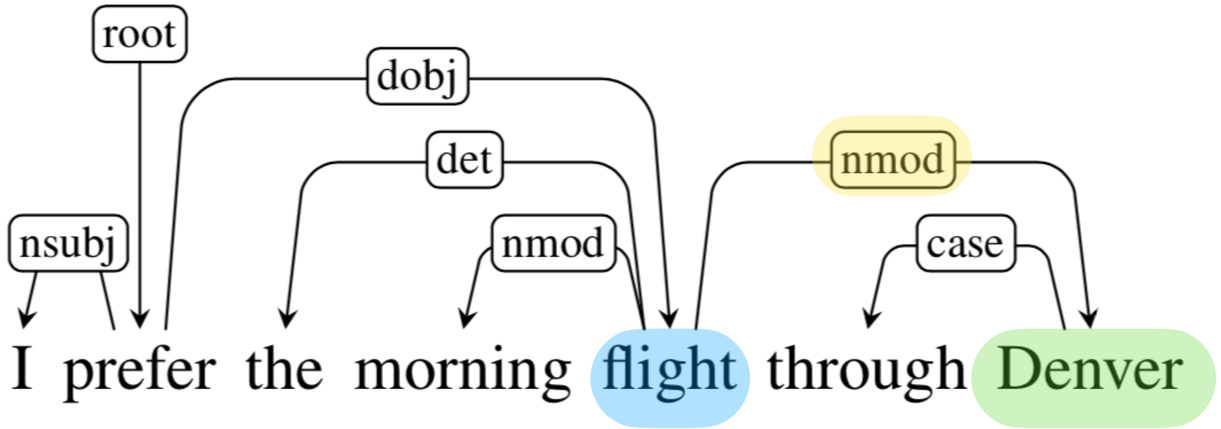


Disambiguation





Dependency Structure



- Consists of relations between lexical items, normally *binary*, *asymmetric* relations ("arrows") called **dependencies**
- The arrows are commonly typed with the name of grammatical relations (subject, prepositional object, apposition, etc)

- The arrow connects a **head** (governor) and a **dependent** (modifier)
- Usually, dependencies form a tree (single-head, connected, acyclic)

Dependency Relations

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

Figure 14.2 Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

https://web.stanford.edu/~jurafsky/slp3/14.pdf

Dependency Relations

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*.

Figure 14.3 Examples of core Universal Dependency relations.

https://web.stanford.edu/~jurafsky/slp3/14.pdf

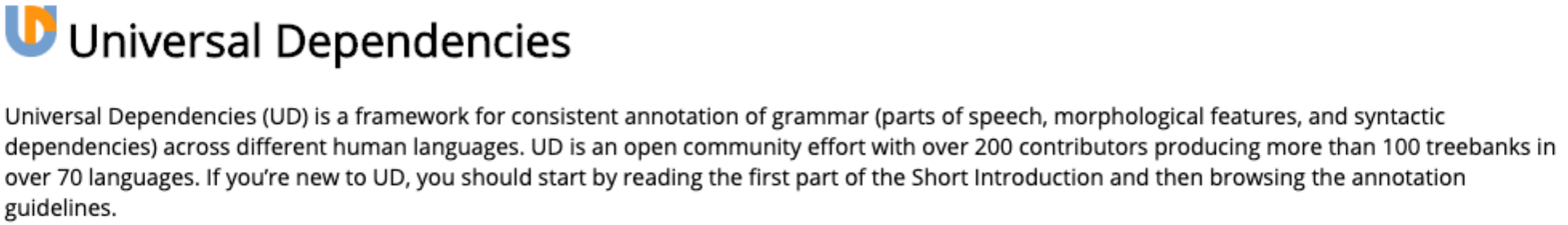
Dependency Treebanks

- The major English dependency treebank: converting from Penn Treebank using rule-based algorithms
  - (De Marneffe et al, 2006): Generating typed dependency parses from phrase structure parses
  - (Johansson and Nugues, 2007): Extended Constituent-to-dependency Conversion for English

Stanford Dependencies (English)

- Universal Dependencies: more than 100 treebanks in 70 languages were collected since 2016

Universal Dependencies (Multilingual)



https://universaldependencies.org/

Universal Dependencies

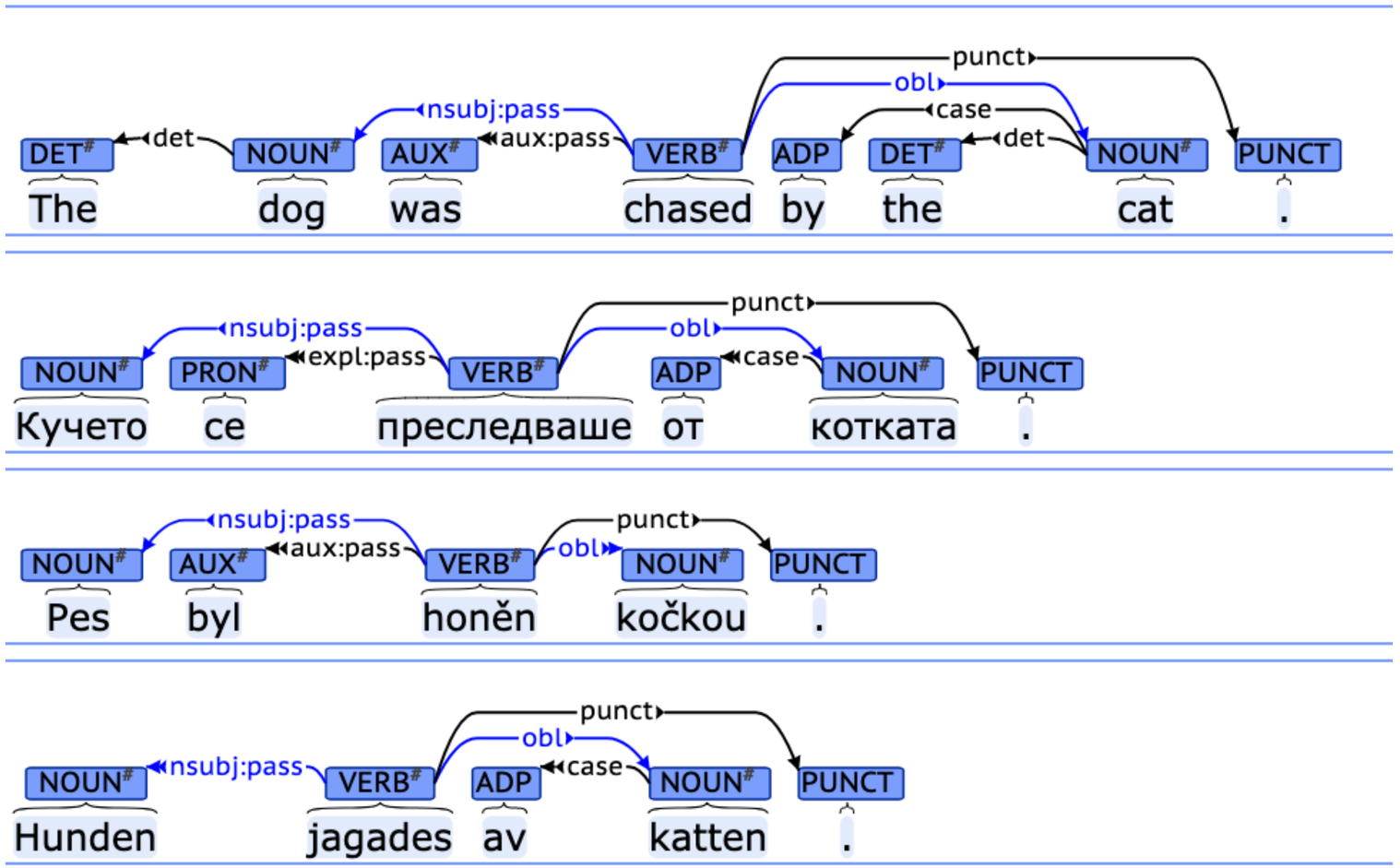
- Developing cross-linguistically consistent treebank annotation for many languages

- Goals:
  - Facilitating multilingual parser development
  - Cross-lingual learning
  - Parsing research from a language typology perspective.

Universal Dependencies

Afrikaans	1	49K		IE, Germanic
Akkadian	1	1K		Afro-Asiatic, Semitic
Amharic	1	10K		Afro-Asiatic, Semitic
Ancient Greek	2	416K		IE, Greek
Arabic	3	1,042K	W	Afro-Asiatic, Semitic
Armenian	1	36K		IE, Armenian
Assyrian	1	<1K		Afro-Asiatic, Semitic
Bambara	1	13K		Mande
Basque	1	121K		Basque
Belarusian	1	13K		IE, Slavic
Breton	1	10K	W	IE, Celtic
Bulgarian	1	156K		IE, Slavic
Buryat	1	10K		Mongolic
Cantonese	1	13K		Sino-Tibetan
Catalan	1	531K		IE, Romance
Chinese	5	161K	W	Sino-Tibetan
Classical Chinese	1	55K		Sino-Tibetan
Coptic	1	25K		Afro-Asiatic, Egyptian
Croatian	1	199K	W	IE, Slavic
Czech	5	2,222K	W	IE, Slavic
Danish	2	100K		IE, Germanic
Dutch	2	307K	W	IE, Germanic
English	6	603K	W	IE, Germanic
Erzya	1	15K		Uralic, Mordvin
Estonian	2	461K		Uralic, Finnic
Faroese	1	10K	W	IE, Germanic
Finnish	3	377K	W	Uralic, Finnic
French	8	1,156K	W	IE, Romance
Galician	2	164K		IE, Romance
German	4	3,409K	W	IE, Germanic
Gothic	1	55K		IE, Germanic
Greek	1	63K	W	IE, Greek
Hebrew	1	161K		Afro-Asiatic, Semitic
Hindi	2	375K	W	IE, Indic
Hindi English	1	26K		Code switching
Hungarian	1	42K		Uralic, Ugric
Indonesian	2	141K	W	Austronesian, Malayo-Sumbawan
Irish	1	23K		IE, Celtic
Italian	6	781K	W	IE, Romance
Japanese	5	1,688K	W	Japanese
Karelian	1	3K		Uralic, Finnic
Kazakh	1	10K	W	Turkic, Northwestern
Komi Zyrian	2	3K		Uralic, Permic
Korean	5	446K	W	Korean

Universal Dependencies



https://universaldependencies.org/introduction.html

Universal Dependencies

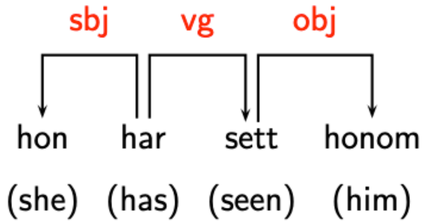
Manning's Law:

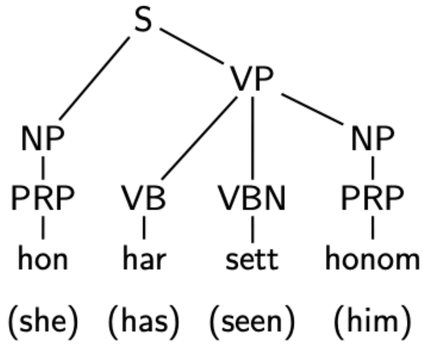
- UD needs to be satisfactory for analysis of individual languages.
- UD needs to be good for linguistic typology.
- UD must be suitable for rapid, consistent annotation.
- UD must be suitable for computer parsing with high accuracy.
- UD must be easily comprehended and used by a non-linguist.
- UD must provide good support for downstream NLP tasks.

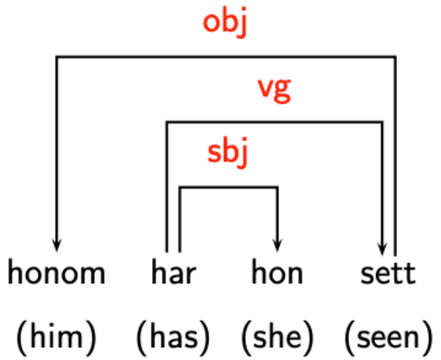
Why we need dependency when we already have constituency?

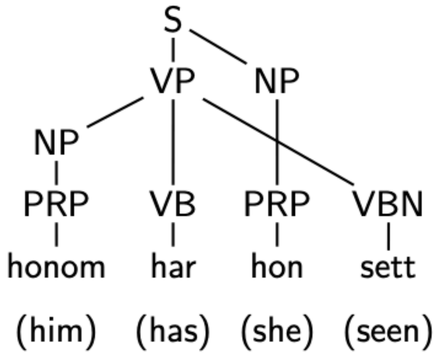
Advantages of Dependency Structure

- More suitable for free word order languages









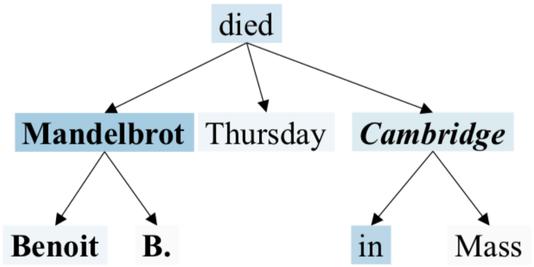
Advantages of Dependency Structure

- More suitable for free word order languages
- The predicate-argument structure is more useful for many applications

Relation Extraction

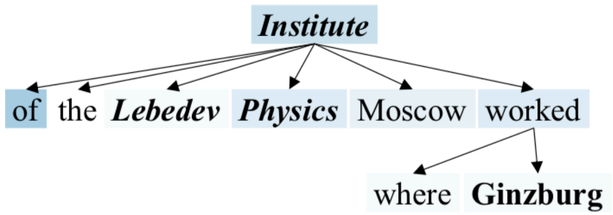
Relation: *per:city_of_death*

Benoit B. Mandelbrot, a maverick mathematician who developed an innovative theory of roughness and applied it to physics, biology, finance and many other fields, died Thursday in ***Cambridge***, Mass.



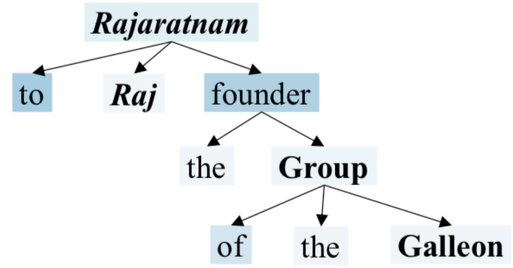
Relation: *per:employee_of*

In a career that spanned seven decades, Ginzburg authored several groundbreaking studies in various fields -- such as quantum theory, astrophysics, radio-astronomy and diffusion of cosmic radiation in the Earth's atmosphere -- that were of "Nobel Prize caliber," said Gennady Mesyats, the director of the ***Lebedev Physics Institute*** in Moscow, where **Ginzburg** worked .



Relation: *org:founded_by*

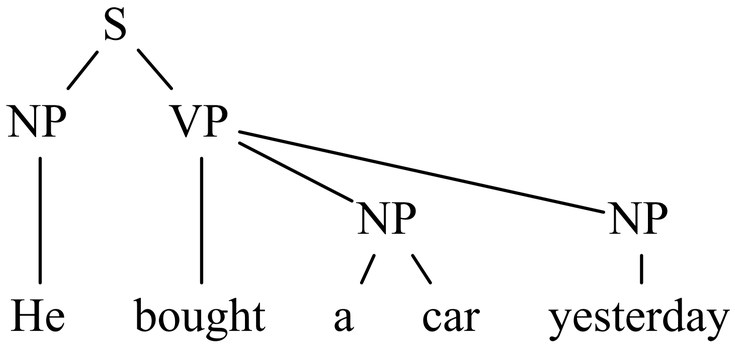
Anil Kumar, a former director at the consulting firm McKinsey & Co, pleaded guilty on Thursday to providing inside information to ***Raj Rajaratnam***, the founder of the **Galleon Group**, in exchange for payments of at least $ 175 million from 2004 through 2009.

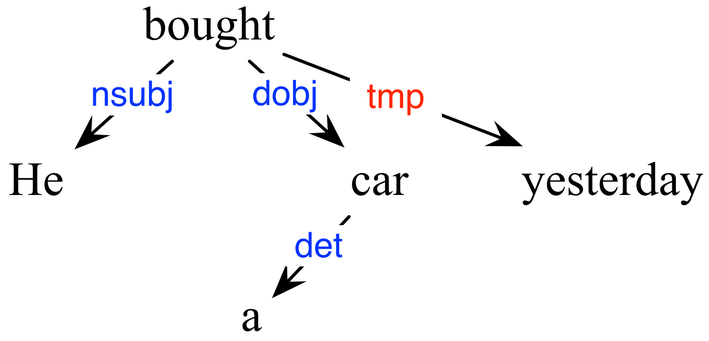


Dependency Formalisms

Dependency Structure

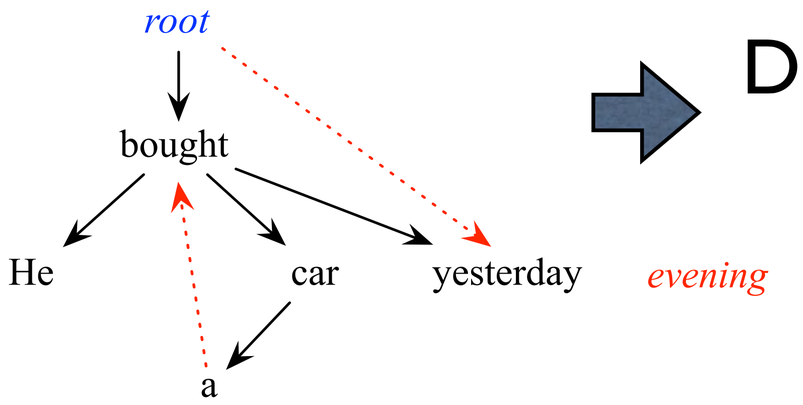
- Constituent structure
  - Starts with the bottom level constituents (tokens).
  - Group smaller constituents into bigger constituents (phrases).
- Dependency structure
  - Starts with vertices (tokens).
  - Build a graph by adding edges between vertices (arcs).





Dependency Graph

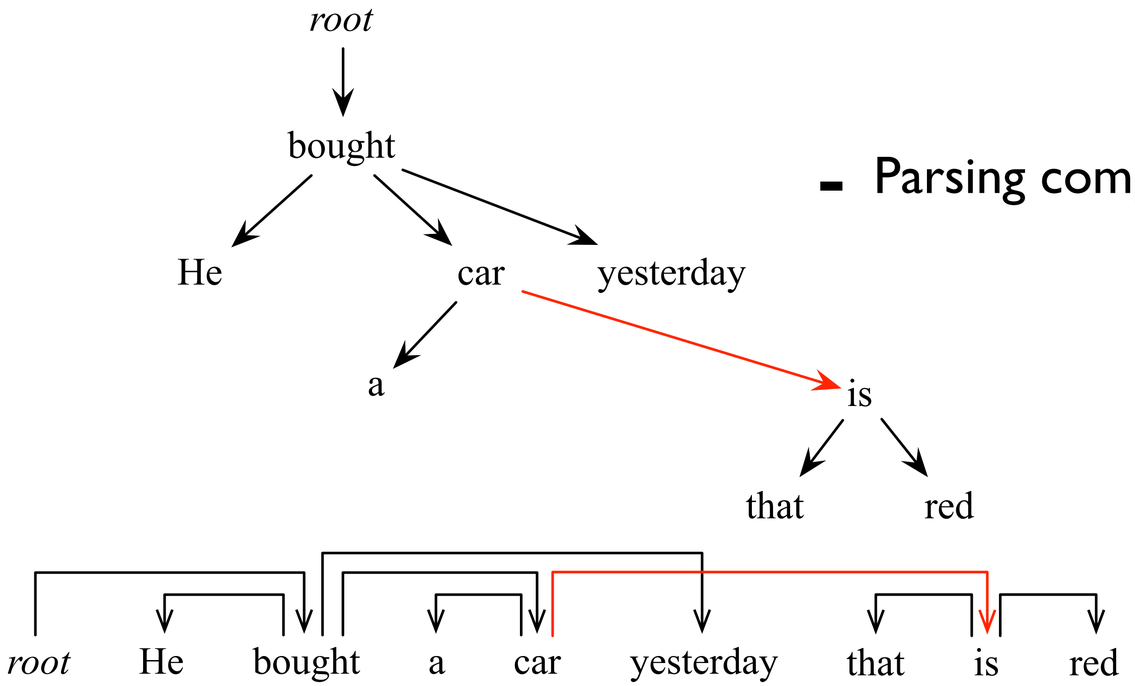
- For a sentence $s = w_1 \dots w_n$, a dependency graph $G_s = (V_s, A_s)$
  - $V_s = \{w_0 = \textbf{root}, w_1, \dots, w_n\}$.
  - $A_s = \{(w_i, r, w_j) : i \neq j, w_i \in V_s, w_j \in V_s - \{w_0\}, r \in R_s\}$.
    - R_s = a subset of dependency relations in s.
- A well-formed dependency graph
  - Root
  - Single head
  - Connected
  - Acyclic



Dependency Tree

Projectivity

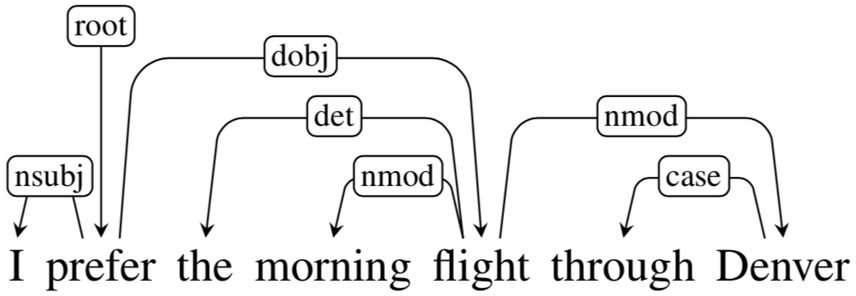
- A projective dependency tree has no crossing arc when all vertices are lined up in linear order and arcs are drawn above.
- e.g., *He bought a car yesterday that is red*.



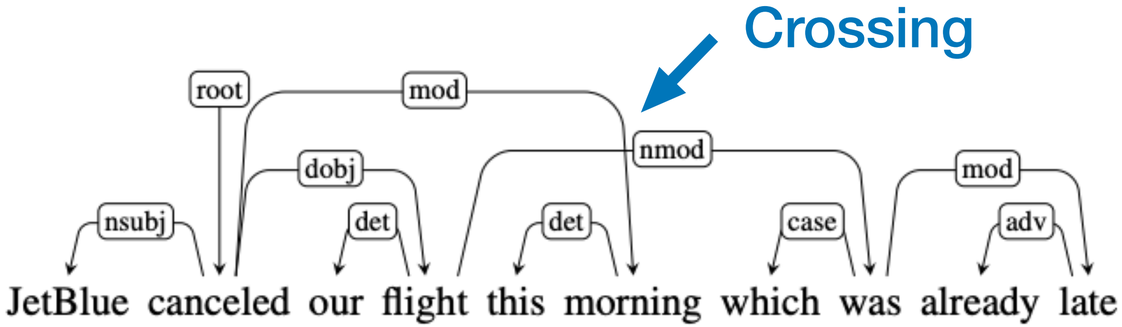
- Parsing complexity: O(n) vs. O(n^2).

Projectivity

- **Definition**: there are no crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words



projective



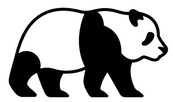
non-projective

Non-projectivity arises due to long distance dependencies or in languages with flexible word order.

This class: focuses on projective parsing

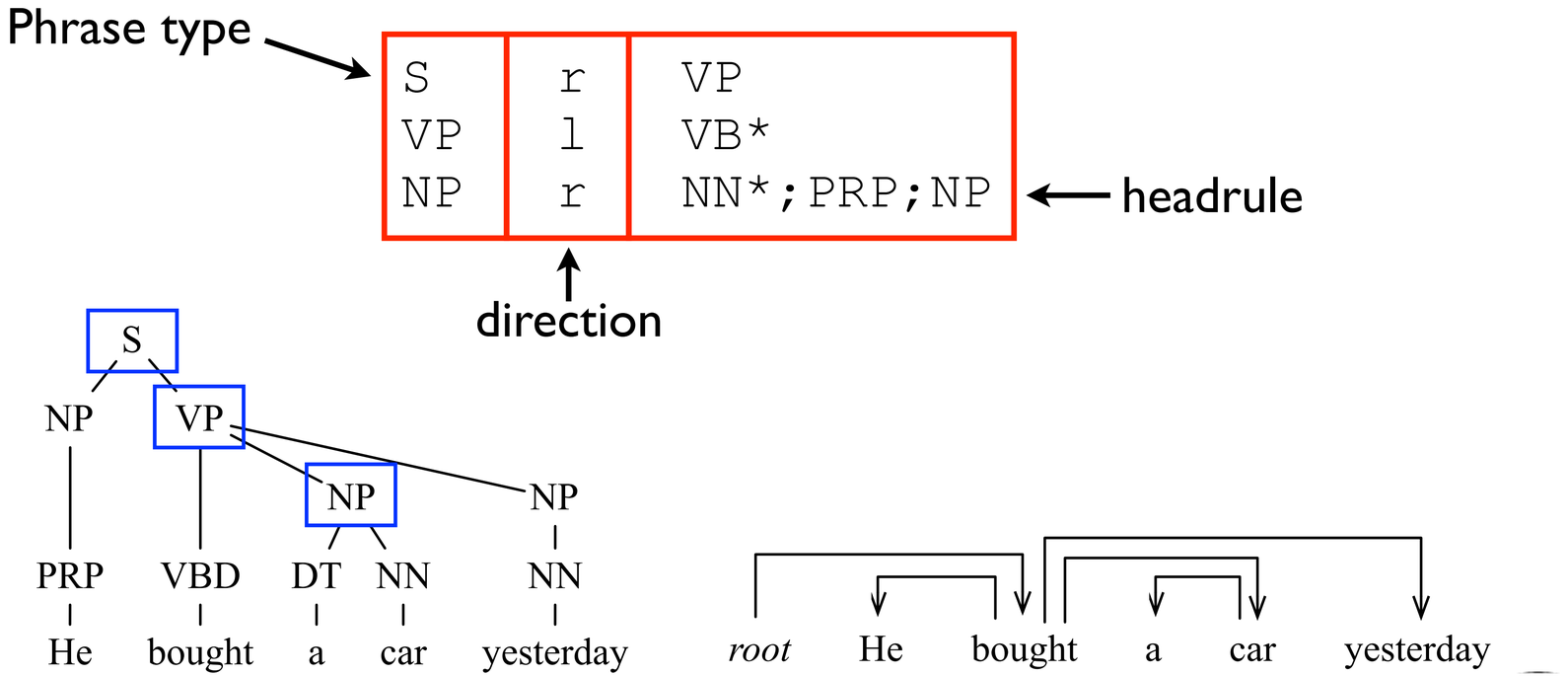
Dataset	# Sentences	(%) Projective
English	39,832	99.9
Chinese	16,091	100.0
Czech	72,319	76.9
German	38,845	72.2

How to get dependency structure?

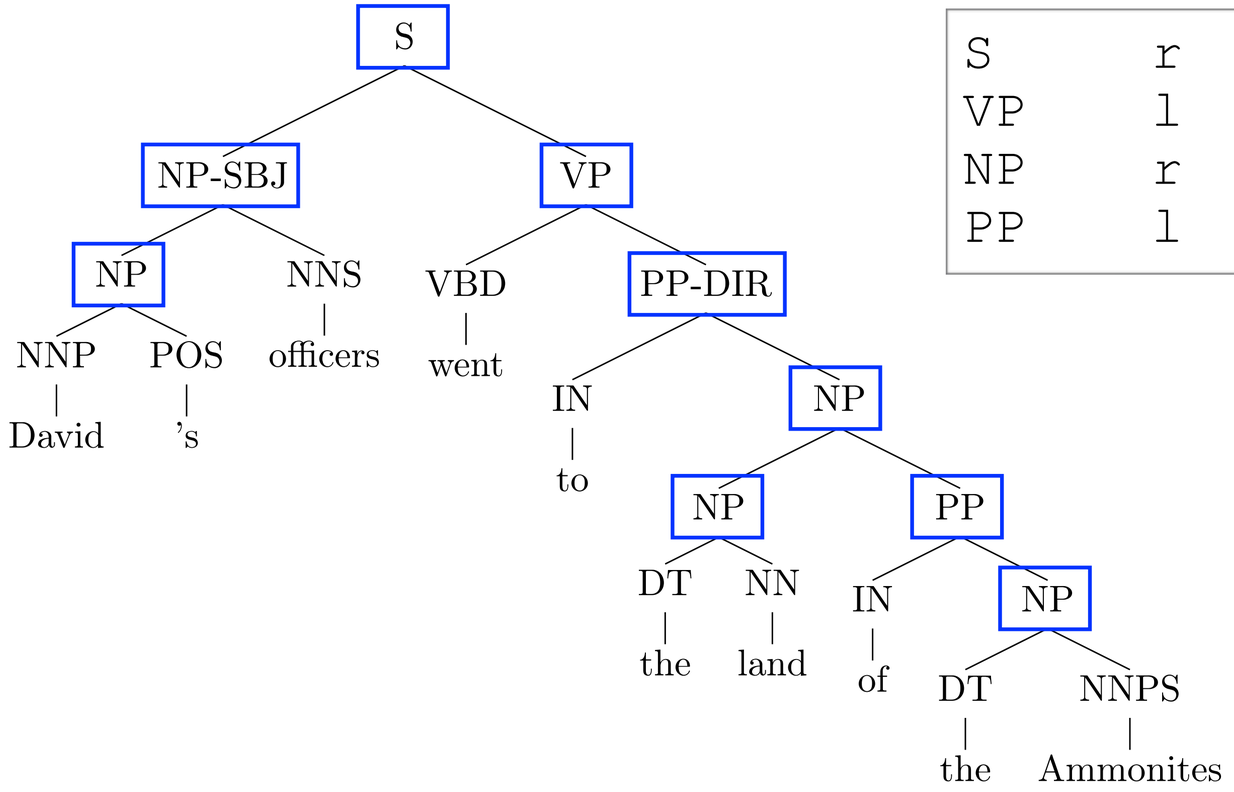


Constituent to Dependency

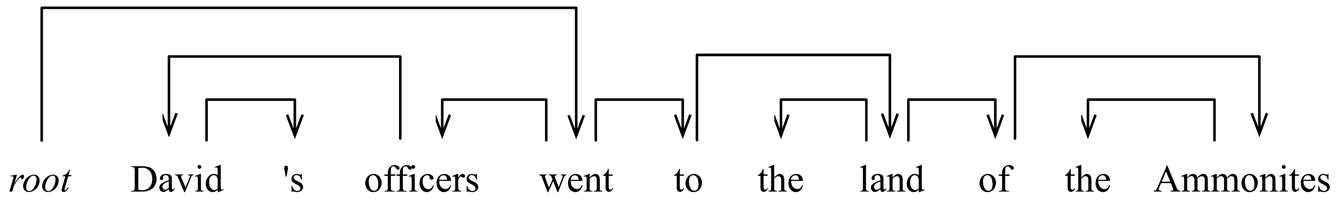
- Head-finding rules (i.e., head-percolation rules, headrules)
  - Constituent trees can be converted into dependency trees.
  - Apply headrules recursively to find the head of each constituent.



Constituent to Dependency



S	r	VP
VP	l	VB*
NP	r	NN*;PRP;NP
PP	l	IN

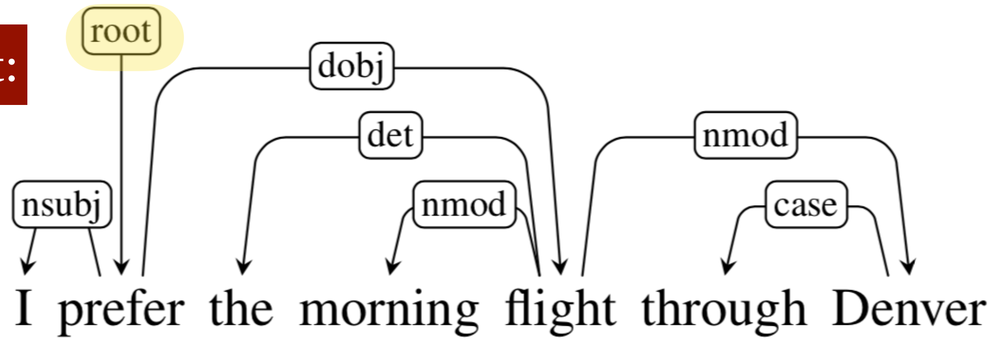


Dependency Parsing

Input:

I prefer the morning flight through Denver

Output:



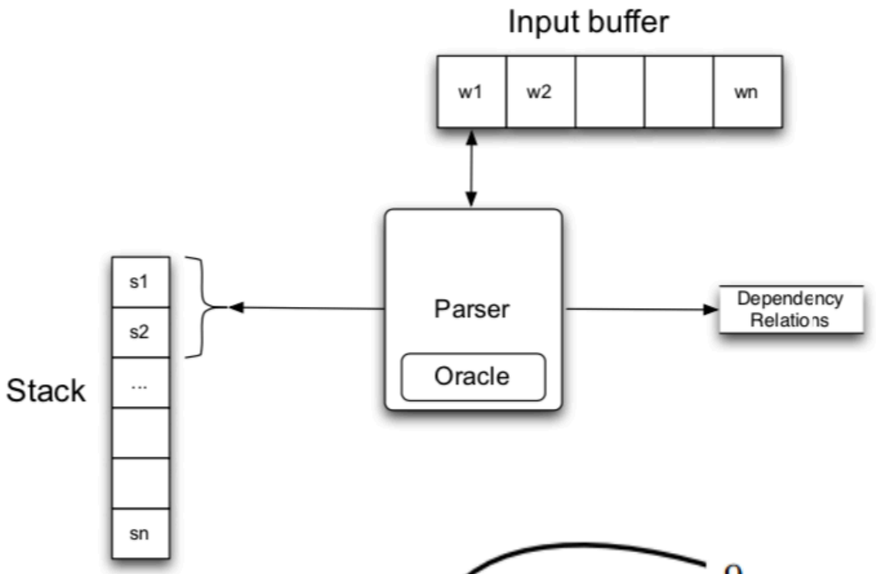
- A sentence is parsed by choosing for each word what other word is it a dependent of (and also the relation type)
- We usually add a fake ROOT at the beginning so every word has one head
- Usually some constraints:
  - Only one word is a dependent of ROOT
  - No cycles: A —> B, B —> C, C —> A

Learning from data: treebanks!

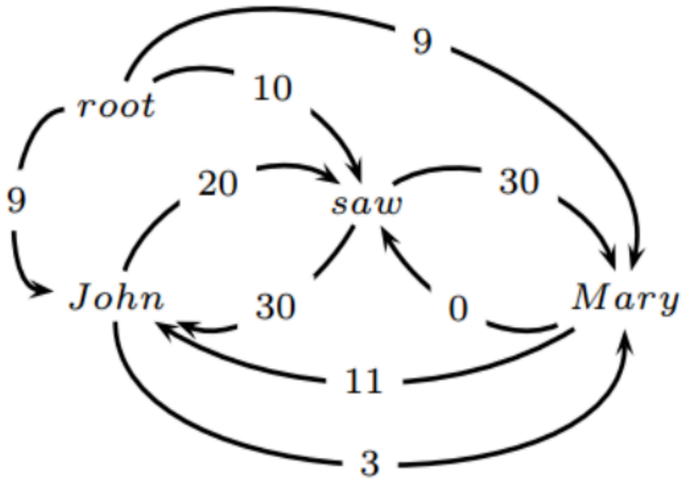
Two Families of Algorithms

Transition-based dependency parsing

- Also called “shift-reduce parsing”



Graph-based dependency parsing



Two Families of Algorithms

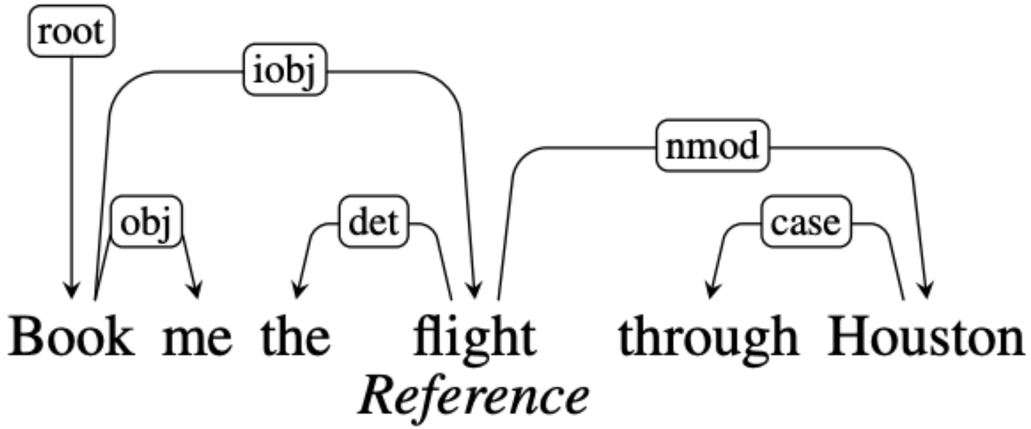
- Transition-based parsing
  - Transition: an operation searching for a dependency relation between each pair of tokens (e.g., Shift, Reduce).
  - Greedy search that finds local optima (locally optimized transitions) → do better for local dependencies.
  - Projective: O(n), non-projective: O(n^2).
- Graph-based parsing
  - Build a complete graph with directed/weighted edges and find a tree with the highest score (sum of all weighted edges).
  - Exhaustive search that finds for the global optimum (maximum spanning tree) → do better for long-distance dependencies.
  - Projective: O(n^3), non-projective: O(n^2).

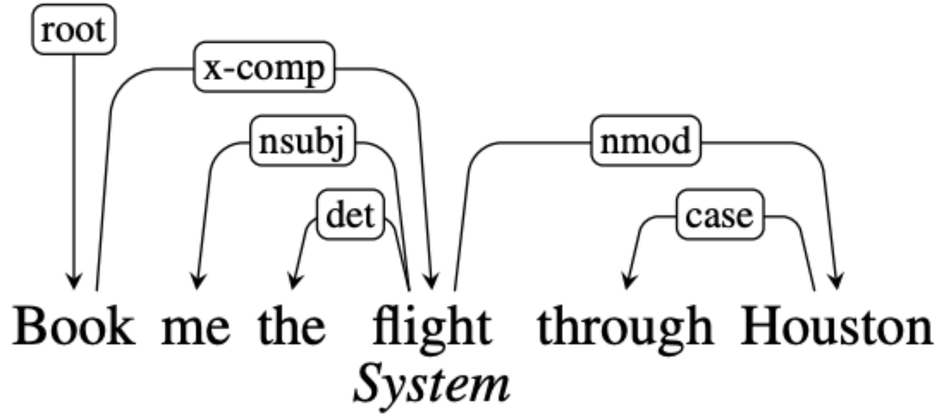
Transition-based Parsing

- Projective parsing: $O(n)$
  - Bottom-up: Yamada & Matsumoto, 2003.
  - Top-down, bottom-up: Nivre, 2003. Shift-reduce parsing
  - Beam search: Zhang & Clark, 2008.
  - Dynamic programming: Huang & Sagae, 2010.
  - Selectional branching: Choi & McCallum, 2013.
- Non-projective parsing: $O(n^2)$
  - Exhaustive search: Covington, 2001.
  - Reordering tokens: Nivre, 2009 (linear-time in practice).
  - Selective search: Choi & Palmer, 2011 (linear-time in practice).

Evaluation

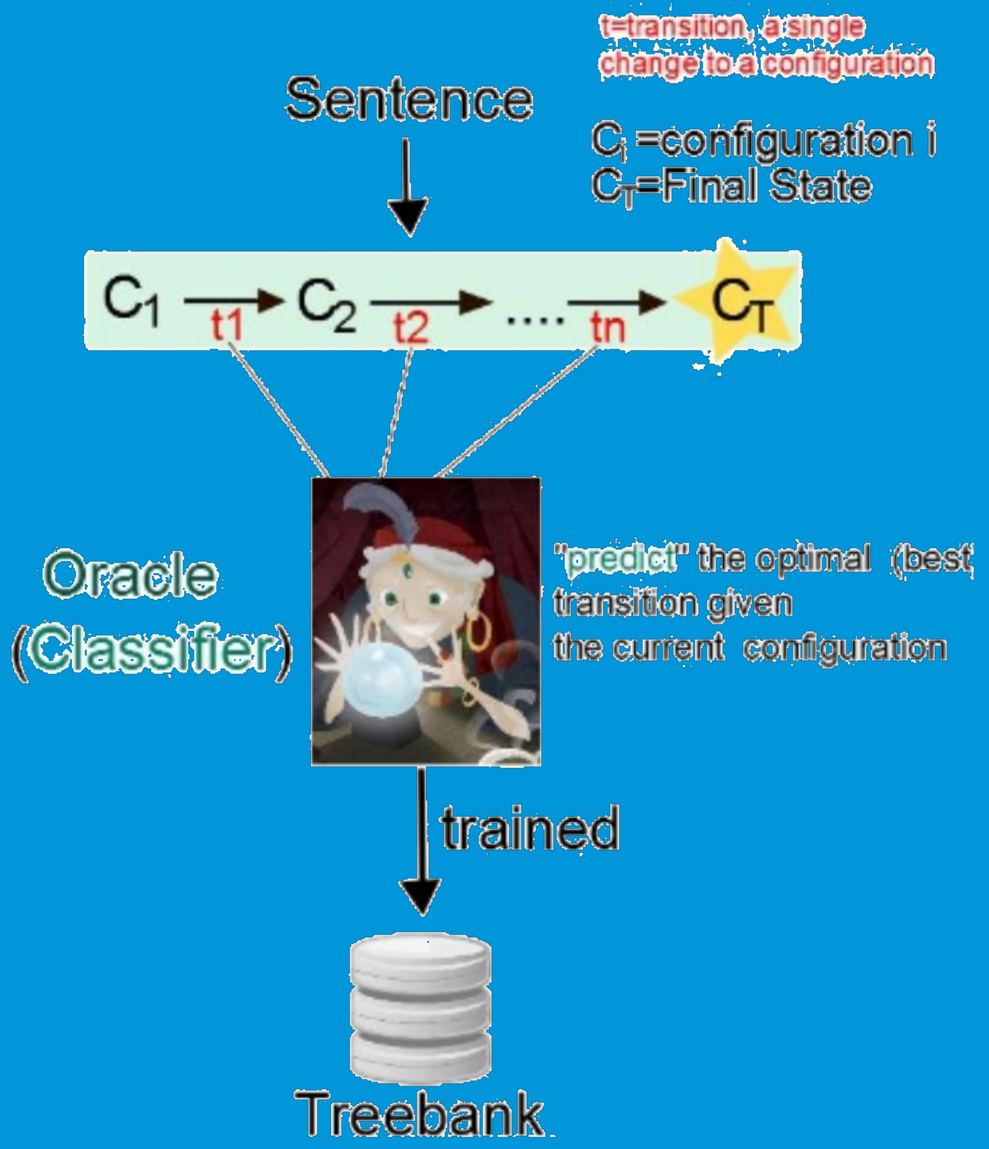
- Unlabeled attachment score (UAS)
  = percentage of words that have been assigned the correct head
- Labeled attachment score (LAS)
  = percentage of words that have been assigned the correct head & label





UAS = ? LAS = ?

Transition-based Dependency Parsing



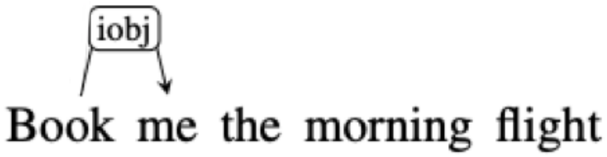
Transition-based Dependency Parsing

- The parsing process is modeled as a sequence of transitions
- A configuration consists of a stack s, a buffer b and a set of dependency arcs A: $c = (s, b, A)$

Stack: Can add arcs to 1st two words on stack

Buffer: Unprocessed words

Current graph:



Transition-based Dependency Parsing

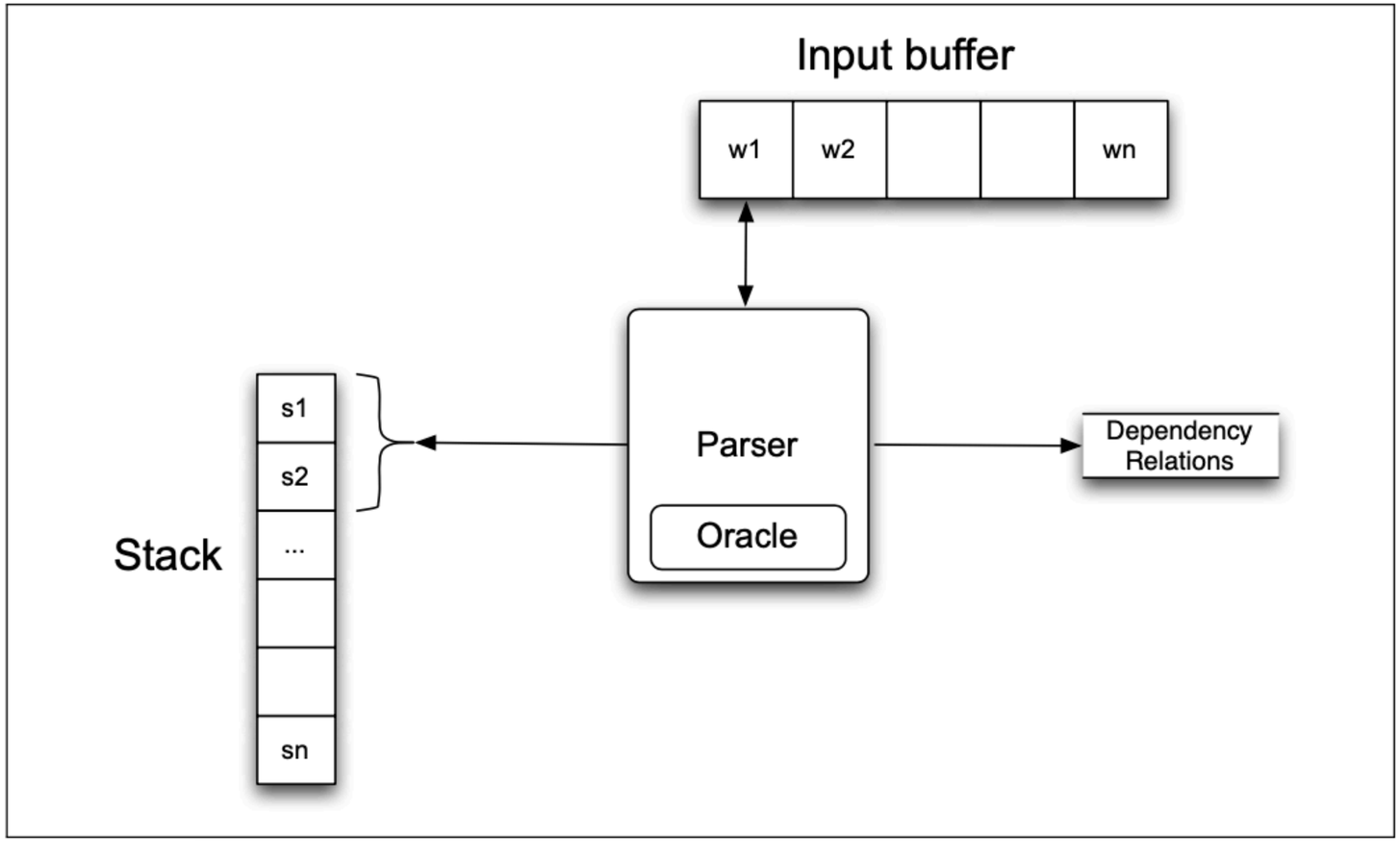


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

https://web.stanford.edu/~jurafsky/slp3/14.pdf

Transition-based Dependency Parsing

Examining the words in a single pass over the input from left to right, and take one of the following actions:

- Assign the current word as the head of some previously seen word,
- Assign some previously seen word as the head of the current word,
- Or postpone doing anything with the current word, adding it to a store for later processing.

https://web.stanford.edu/~jurafsky/slp3/14.pdf

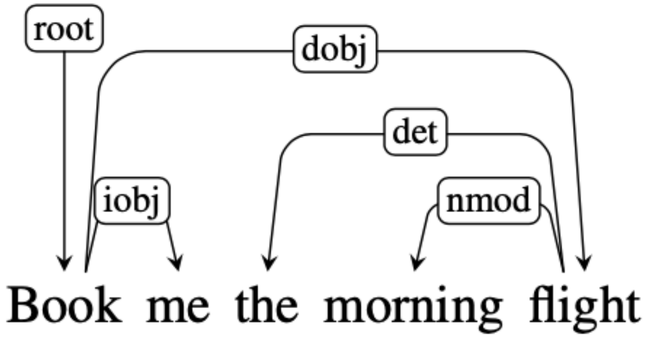
Transition-based Dependency Parsing

To make these actions more precise, we'll create three transition operators that will operate on the top two elements of the stack:

- **LEFTARC**: Assert a head-dependent relation between the word at the top of the stack and the word directly beneath it; remove the lower word from the stack.
- **RIGHTARC**: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack;
- **SHIFT**: Remove the word from the front of the input buffer and push it onto the stack.

https://web.stanford.edu/~jurafsky/slp3/14.pdf

A Running Example



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	

Figure 14.7 Trace of a transition-based parse.

https://web.stanford.edu/~jurafsky/slp3/14.pdf

Things to Note

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	

Figure 14.7 Trace of a transition-based parse.

Several things to note:

- The sequence given is not the only one that might lead to a reasonable parse
- We are assuming that the oracle always provides the correct operator at each point in the parse
- We have illustrated this example without the labels on the dependency relations. To produce labeled trees, we can parameterize the LEFTARC and RIGHTARC operators with dependency labels, as in LEFTARC(NSUBJ) or RIGHTARC(DOBJ).

https://web.stanford.edu/~jurafsky/slp3/14.pdf

Creating an Oracle

Train classifiers to play the role of the **oracle**.

- **Training data:** representative treebanks containing dependency trees.
- **Features:** manually designed or learned

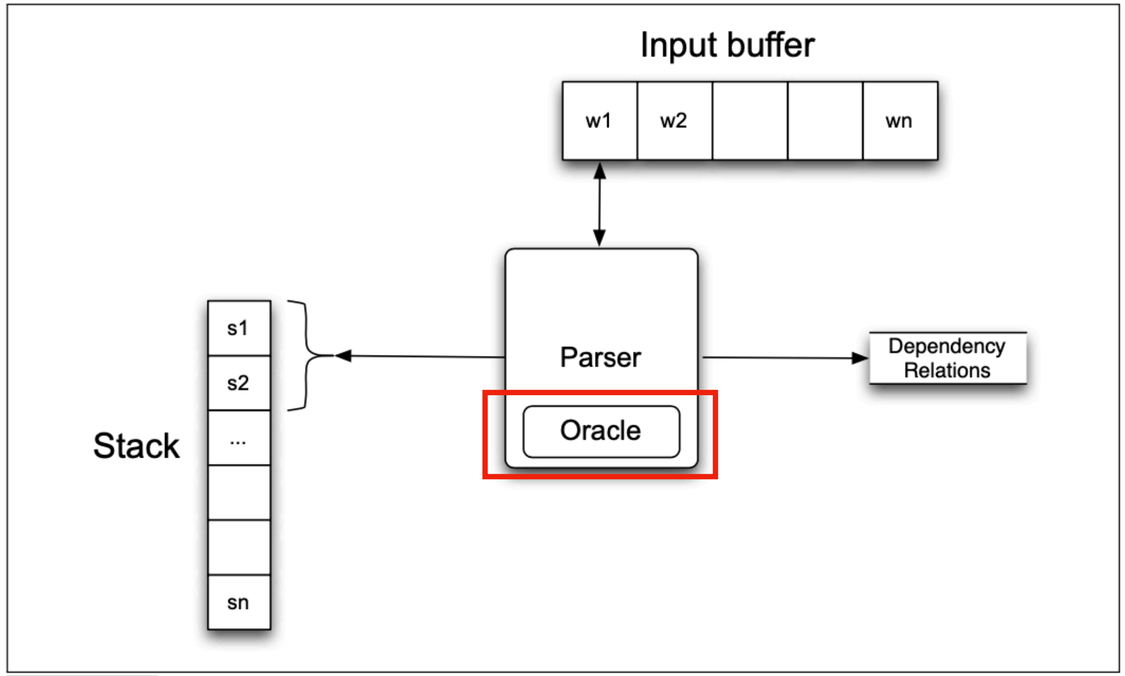


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

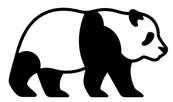
https://web.stanford.edu/~jurafsky/slp3/14.pdf

Creating an Oracle: Generate Training Data

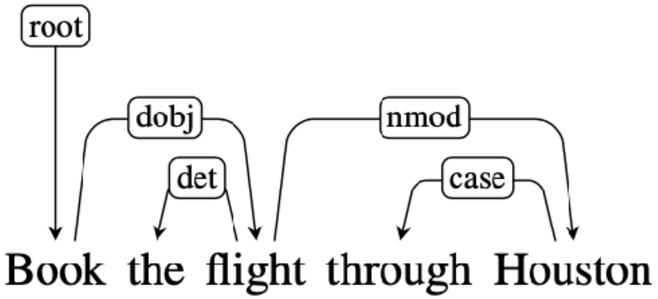
Given a reference parse and a configuration, the training oracle proceeds as follows:

- Choose **LEFTARC** if it produces a correct head-dependent relation given the reference parse and the current configuration,
- Otherwise, choose **RIGHTARC** if (1) it produces a correct head-dependent relation given the reference parse and (2) **all of the dependents of the word at the top of the stack have already been assigned,**
- Otherwise, choose **SHIFT**.

https://web.stanford.edu/~jurafsky/slp3/14.pdf



Creating an Oracle: Generate Training Data

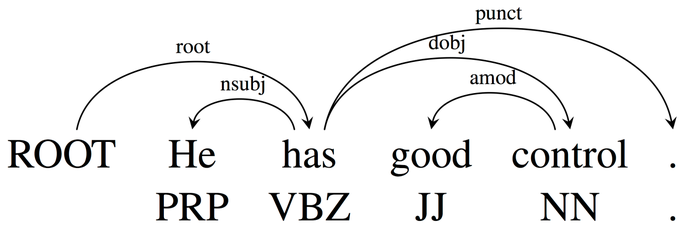


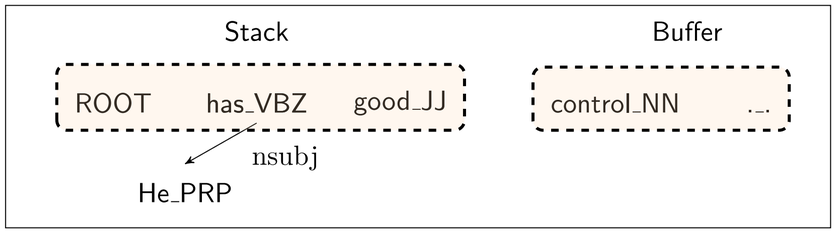
Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Figure 14.8 Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.

https://web.stanford.edu/~jurafsky/slp3/14.pdf

Creating an Oracle: Features





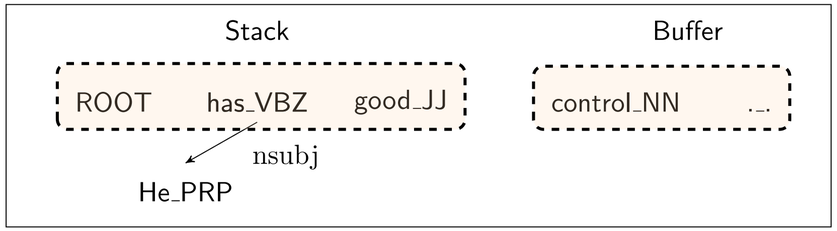
- Extract features from the configuration
- Use your favorite classifier: logistic regression, SVM…

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

Figure 14.9 Standard feature templates for training transition-based dependency parsers. In the template specifications s_n refers to a location on the stack, b_n refers to a location in the word buffer, w refers to the wordform of the input, and t refers to the part of speech of the input.

https://web.stanford.edu/~jurafsky/slp3/14.pdf

Creating an Oracle: Features



Feature templates

$s_2 . w \circ s_2 . t$

$s_1 . w \circ s_1 . t \circ b_1 . w$

$lc(s_2) . t \circ s_2 . t \circ s_1 . t$

$lc(s_2) . w \circ lc(s_2) . l \circ s_2 . w$

Features

$s_2 . w = \text{has} \circ s_2 . t = \text{VBZ}$

$s_1 . w = \text{good} \circ s_1 . t = \text{JJ} \circ b_1 . w = \text{control}$

$lc(s_2) . t = \text{PRP} \circ s_2 . t = \text{VBZ} \circ s_1 . t = \text{JJ}$

$lc(s_2) . w = \text{He} \circ lc(s_2) . l = \text{nsubj} \circ s_2 . w = \text{has}$

Usually a combination of 1-3 elements from the configuration

Binary, sparse, millions of features

Parsing with Neural Networks

A Fast and Accurate Dependency Parser using Neural Networks

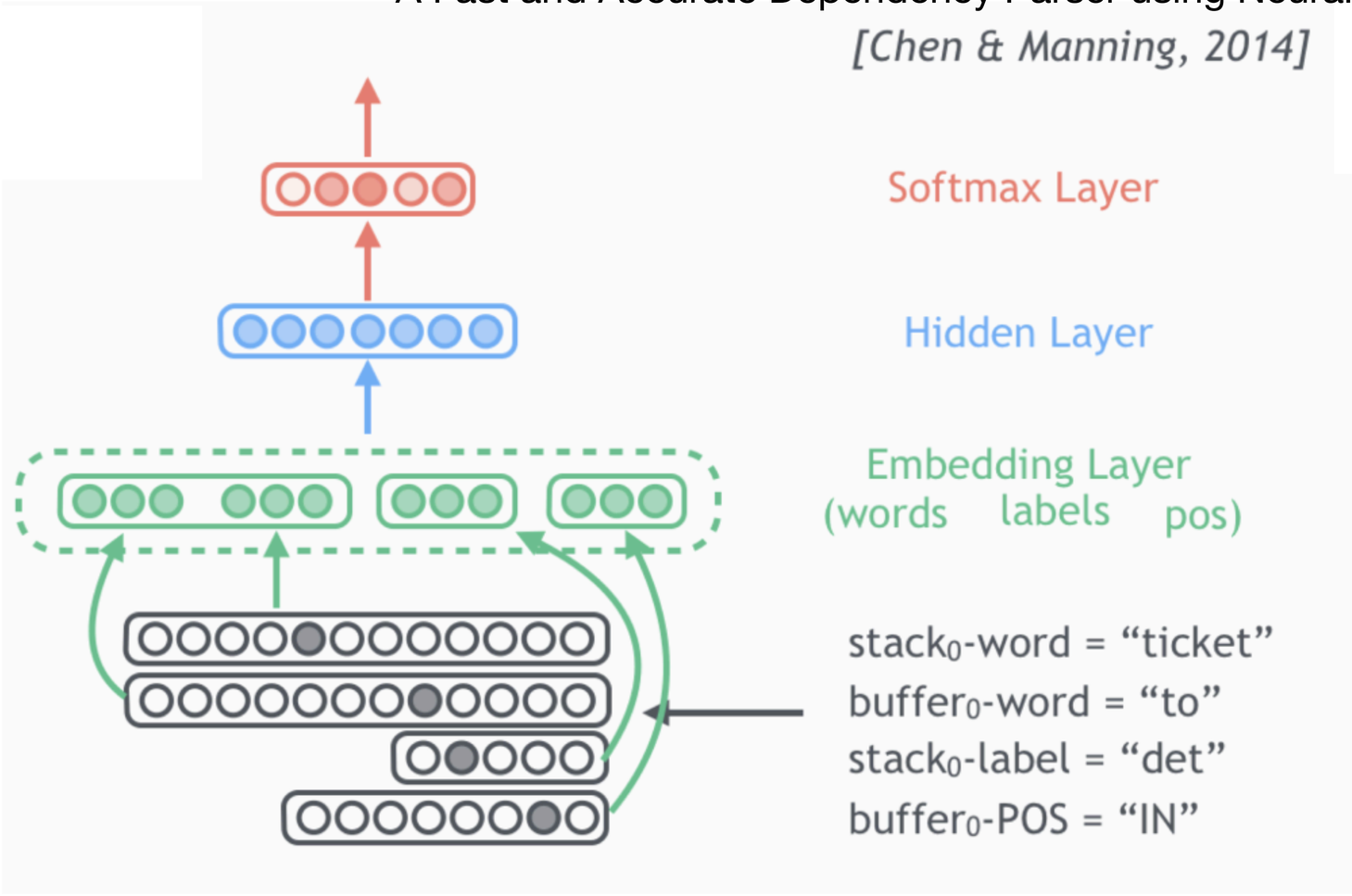
[Chen & Manning, 2014]

Representation for configuration:

- Embeddings for words/POS tags on top of stack
- Embeddings for words/POS tags at front of buffer
- Embeddings for existing arc labels at specific positions

Classifier:

- Feed-forward neural network (input representation has a fixed dimensionality)



No feature templates anymore!

Further Improvements

- Bigger, deeper networks with better tuned hyperparameters
- Beam search
- Global normalization

Method	UAS	LAS (PTB WSJ SD 3.3)
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79

Google's SyntaxNet and the Parsey McParseFace (English) model

> Announcing SyntaxNet: The World's Most Accurate Parser Goes Open Source
>
> Thursday, May 12, 2016

Arc Eager Transition System

- The arc-standard transition system described above is only one of many possible systems.
- A frequently used alternative is the arc eager transition system.
- It asserts rightward relations much sooner than in the arc standard approach.
- This is accomplished through **minor changes to the LEFTARC and RIGHTARC operators** and the addition of a new **REDUCE** operator.

- **Read 14.4.2** for more details: https://web.stanford.edu/~jurafsky/slp3/14.pdf

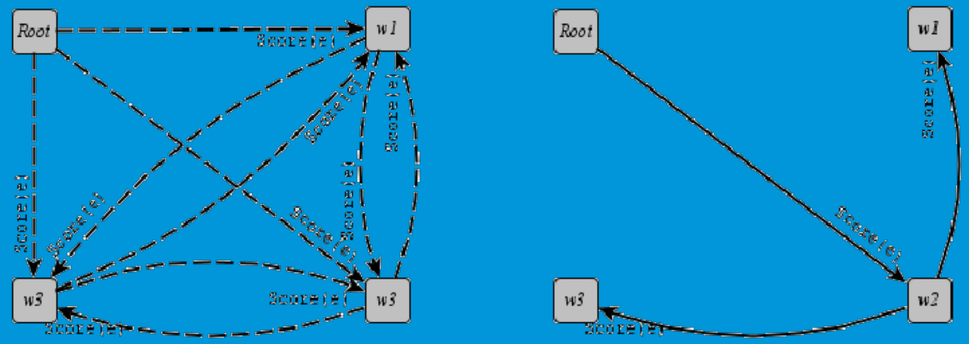
Arc Eager Transition System

- LEFTARC: Assert a head-dependent relation between the word at the front of the input buffer and the word at the top of the stack; pop the stack.
- RIGHTARC: Assert a head-dependent relation between the word on the top of the stack and the word at front of the input buffer; shift the word at the front of the input buffer to the stack.
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.
- REDUCE: Pop the stack.

Beam Search

- Transition-based approach: a single pass through the sentence, highly efficient
- Also the source of its greatest weakness – once a decision has been made it can not be undone.
- **Beam search**: not only choose the single best operation at each step (similar to text generation).

Graph-based Dependency Parsing



Graph-based Dependency Parsing

- **Basic idea**: let's predict the dependency tree directly

$$Y^* = \arg\max_{Y \in \Phi(X)} score(X, Y)$$

X: sentence, Y: any possible dependency tree

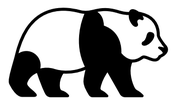
- **Factorization**:

$$score(X, Y) = \sum_{e \in Y} score(e) = \sum_{e \in Y} w^\intercal f(e)$$

Assign scores/weights to all possible edges

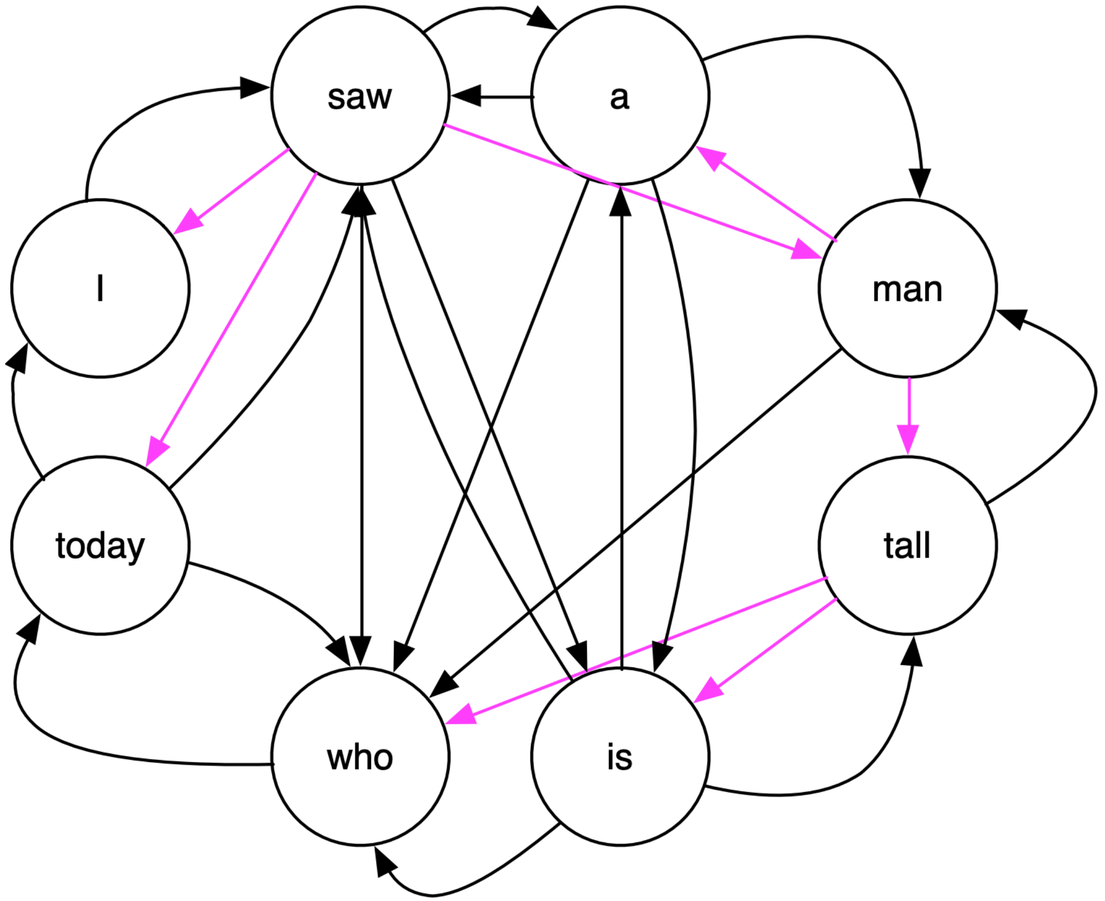
Train a model to compute these scores

- **Inference**: finding maximum spanning tree (MST) for weighted, directed graph



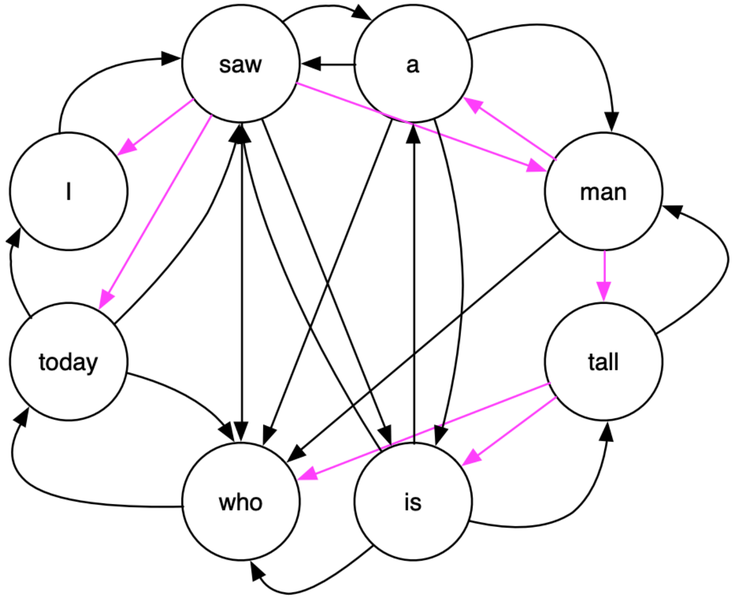
MST Parsing Inference

- We start out with a fully connected graph with a score for each edge
- N^2 edges total



MST Parsing Inference

- From this graph G, we want to find a spanning tree (tree that spans G [includes all the vertices in G])
- If the edges have weights, the best parse is the maximal spanning tree (the spanning tree with the highest total weight).



Graph-based Dependency Parsing

- **Training** learn parameters so the score for the gold tree is higher than for all other trees

Graph-based Dependency Parsing

- **Training** learn parameters so the score for the gold tree is higher than ~~for all other trees~~ a single best tree

Train using structured margin loss: structured perceptron

Structured Perceptron

- Simple way to train (non-probabilistic) global models
- Find the one-best, and if it's score is better than the correct answer, adjust parameters to fix this

$$\hat{Y} = \text{argmax}_{\tilde{Y} \neq Y} S(\tilde{Y} \mid X; \theta)$$

Find one best

$$\textbf{if } S(\hat{Y} \mid X; \theta) \geq S(Y \mid X; \theta) \textbf{ then}$$

If score better than reference

$$\theta \leftarrow \theta + \alpha\left(\frac{\partial S(Y|X;\theta)}{\partial \theta} - \frac{\partial S(\hat{Y}|X;\theta)}{\partial \theta}\right)$$

Increase score of ref, decrease score of one-best (here, SGD update)

$$\textbf{end if}$$

Structured Perceptron and Hinge Loss

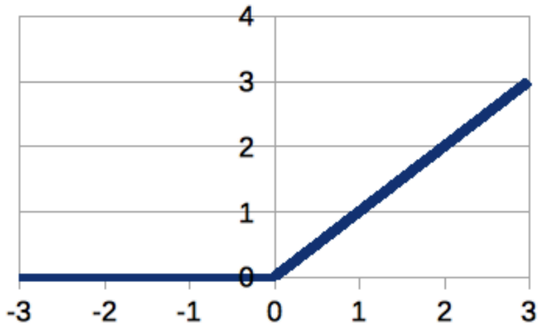
- Loss functions for structured perceptron

$$\ell_{\text{percept}}(X, Y) = \max(0, S(\hat{Y} \mid X; \theta) - S(Y \mid X; \theta))$$

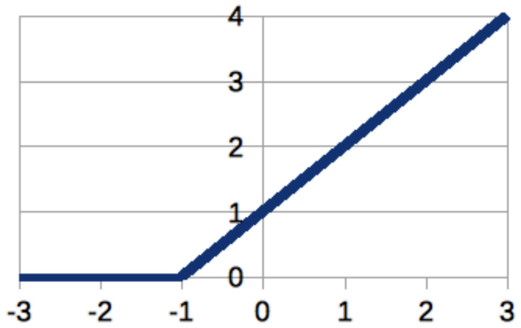
- Penalize when incorrect answer is within margin m

$$\ell_{\text{hinge}}(x, y; \theta) = \max(0, m + S(\hat{y} \mid x; \theta) - S(y \mid x; \theta))$$

Note: hinge loss can be used instead of cross-entropy loss in other places as well



Perceptron



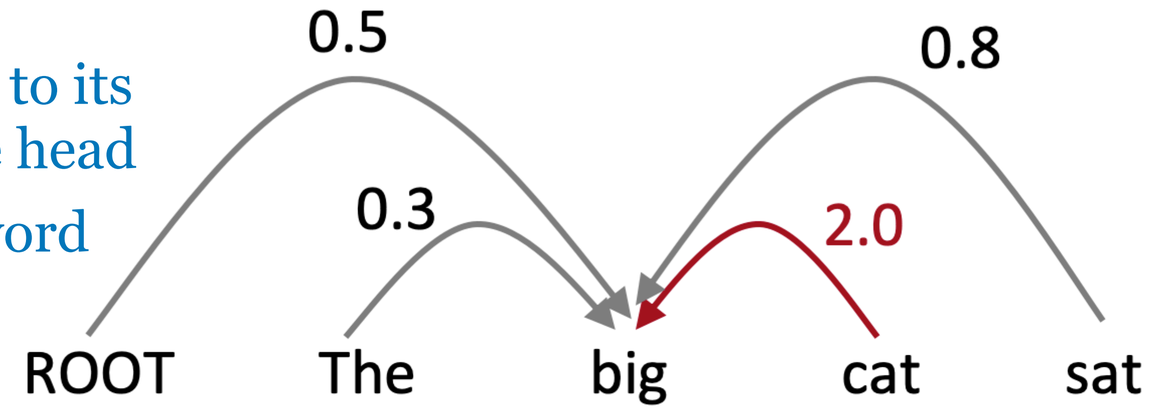
Hinge

$$\ell_{\text{ca-hinge}}(x, y; \theta) = \max(0, \text{cost}(\hat{y}, y) + S(\hat{y} \mid x; \theta) - S(y \mid x; \theta))$$

Graph-based Dependency Parsing

- **Training** learn parameters so the score for the gold tree is higher than ~~for all other trees~~ a single best tree
- **To get a good tree**
  - Compute a score for every possible dependency for each word
  - With neural networks, leverage good "contexual" representations of each word token

- Add edge from each word to its highest-scoring candidate head
- Repeat process for each word



e.g., picking the head for "big"

Neural Graph-based Dependency Parsing

- Pre-neural networks
  - MSTParser - use hard crafted features (McDonald et al, 2005)

- Neural networks - leverage better representation (“contextual” embeddings)
  - Phrase Embeddings (Pei et al, 2015)
  - BiLSTM feature extractors (Kipperwasser and Goldberg 2016)
  - BiAffine Classifier (Dozat and Manning 2017)

Graph-based Dependency Parsing

(Dozat and Manning 2017)

https://arxiv.org/pdf/1611.01734.pdf

- Great result!
- But slower than simple neural transition-based parsers
  - There are n^2 possible dependencies in a sentence of length n

Method	UAS	LAS (PTB WSJ SD 3.3
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79
Dozat & Manning 2017	**95.74**	**94.08**

Rethinking Self-Attention: Towards Interpretability in Neural Parsing

Khalil Mrini[1], Franck Dernoncourt[2], Quan Tran[2], Trung Bui[2], Walter Chang[2], and Ndapa Nakashole[1]

[1] University of California, San Diego, La Jolla, CA 92093

khalil@ucsd.edu, nnakashole@eng.ucsd.edu

[2]Adobe Research, San Jose, CA 95110

{franck.dernoncourt, qtran, bui, wachang}@adobe.com

Penn Treebank

Models are evaluated on the Stanford Dependency conversion (**v3.3.0**) of the Penn Treebank with **predicted** POS-tags. Punctuation symbols are excluded from the evaluation. Evaluation metrics are unlabeled attachment score (UAS) and labeled attachment score (LAS). UAS does not consider the semantic relation (e.g. Subj) used to label the attachment between the head and the child, while LAS requires a semantic correct label for each attachment.Here, we also mention the predicted POS tagging accuracy.

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

http://nlpprogress.com/english/dependency_parsing.html

Main idea

- **Attention mechanisms** provide arguably **explainable** attention distributions that can help to interpret predictions.
- However, **self-attention mechanisms** have multiple heads, making the combined outputs difficult to interpret.
- **Label Attention Layer**: a modified version of self-attention, where each classification label corresponds to one or more attention heads.
- New state of the art for both **constituency and dependency parsing**, in both English and Chinese.

https://www.aclweb.org/anthology/2020.findings-emnlp.65.pdf

Finding Syntax in Word Representations

Do They Encode Syntax?

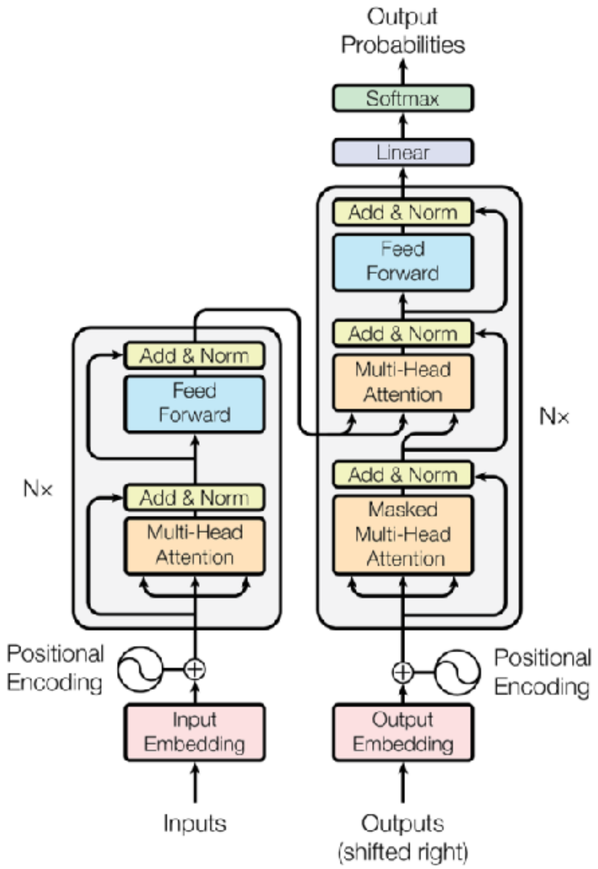


Figure 1: The Transformer - model architecture.

Q1: do pertained LMs encoded syntax through unlabeled corpus?
Q2: if yes, how to detect the syntax encoded?

 Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019

A Structural Probe for Finding Syntax

- Propose a **structural probe** to test **whether syntax trees are embedded** in a **linear transformation** of a neural network's word representation space.
- Tree structure is embedded if the transformed space has the property that **squared L2 distance between two words' vectors** corresponds to the **number of edges** between the words in the parse tree.
- To re-construct edge directions, we hypothesize a linear transformation under which the squared **L2 norm** corresponds to **the depth** of the word in the parse tree.
- What to do? Uses supervision to **find the transformations** under which these properties are best approximated for each model.

Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019

The Structural Probe

- Family of squared distance

$$d_B(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)^2 = \left(B(\mathbf{h}_i^\ell - \mathbf{h}_j^\ell)\right)^T \left(B(\mathbf{h}_i^\ell - \mathbf{h}_j^\ell)\right)$$

- Approximate through gradient descent

$$\min_B \sum_\ell \frac{1}{|s^\ell|^2} \sum_{i,j} \left| d_{T^\ell}(w_i^\ell, w_j^\ell) - d_B(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)^2 \right|$$

Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019

The Structural Probe

Transformed distance between word i, j

The linear transformation to learn (parameters of the probe)

$$d_B(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)^2 = \left(B(\mathbf{h}_i^\ell - \mathbf{h}_j^\ell)\right)^T \left(B(\mathbf{h}_i^\ell - \mathbf{h}_j^\ell)\right)$$

context-sensitive embedding of word j

$$\min_B \sum_\ell \frac{1}{|s^\ell|^2} \sum_{i,j} \left| d_{T^\ell}(w_i^\ell, w_j^\ell) - d_B(\mathbf{h}_i^\ell, \mathbf{h}_j^\ell)^2 \right|$$

length of sentence l

tree distance between word i, j

Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019

The Tree Depth Structural Probe

- replace vector distance with squared norm

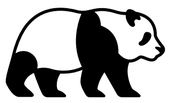
$$\|\mathbf{h}_i\|_A = (B\,\mathbf{h}_i^\ell)^T(B\,\mathbf{h}_i^\ell)$$

Another linear transformation to learn (parameters of the probe)

- Approximate ||w_i||, i.e., the depth of word i, through gradient descent

$$\min_B \sum_\ell \frac{1}{|s^\ell|^2} \sum_i \left| \|w_i\| - \|\mathbf{h}_i\|_A \right|$$

Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019



Experiments: Tree Distance

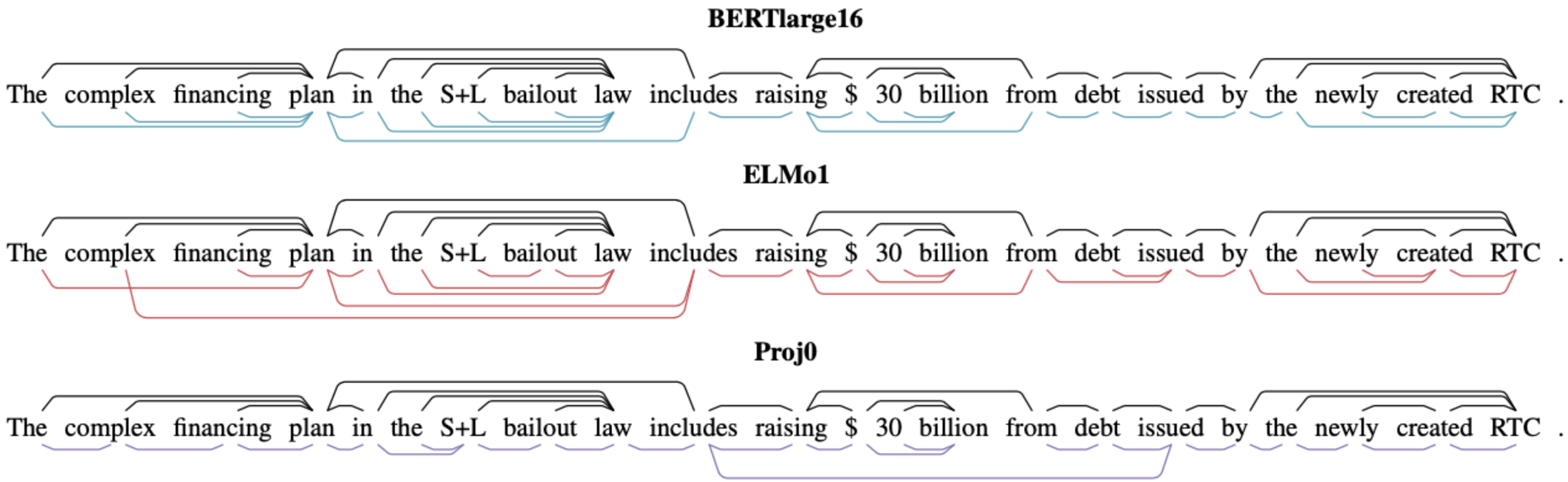


Figure 2: Minimum spanning trees resultant from predicted squared distances on BERTLARGE16 and ELMO1 compared to the best baseline, PROJ0. Black edges are the gold parse, above each sentence; blue are BERTLARGE16, red are ELMO1, and purple are PROJ0.

Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019

Experiments: Tree Depth

We evaluate models on their ability to recreate the order of words specified by their depth in the parse tree. We report the Spearman correlation betwen the true depth ordering and the predicted ordering, averaging first between sentences of the same length, and then across sentence lengths 5–50, as the "norm Spearman (NSpr.)". We also evaluate models' ability to identify the root of the sentence as the least deep, as the "root%".[6]

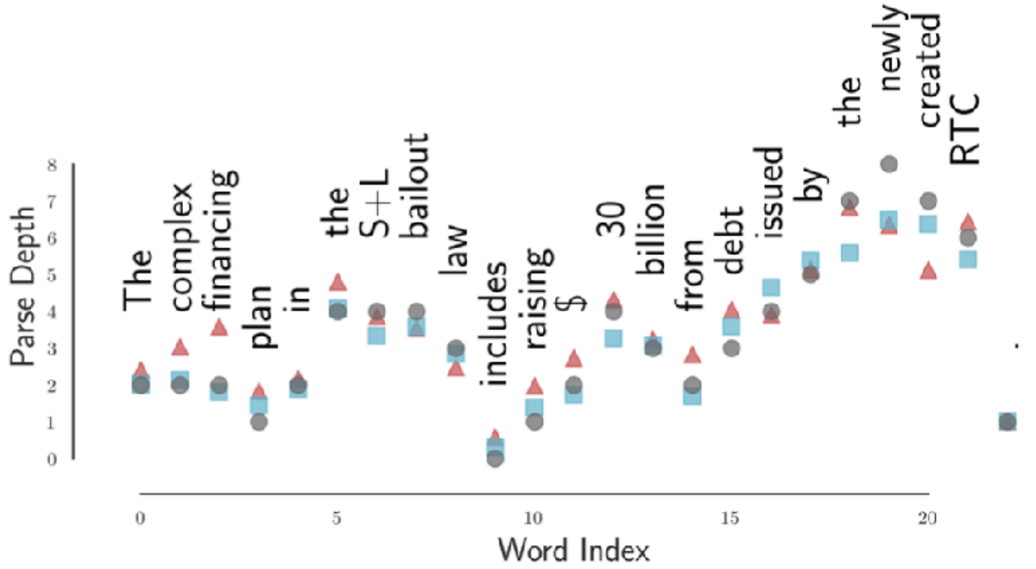
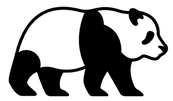


Figure 3: Parse tree depth according to the gold tree (black, circle) and the norm probes (squared) on ELMO1 (red, triangle) and BERTLARGE16 (blue, square).

Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019



Experiments: Different Model Layers

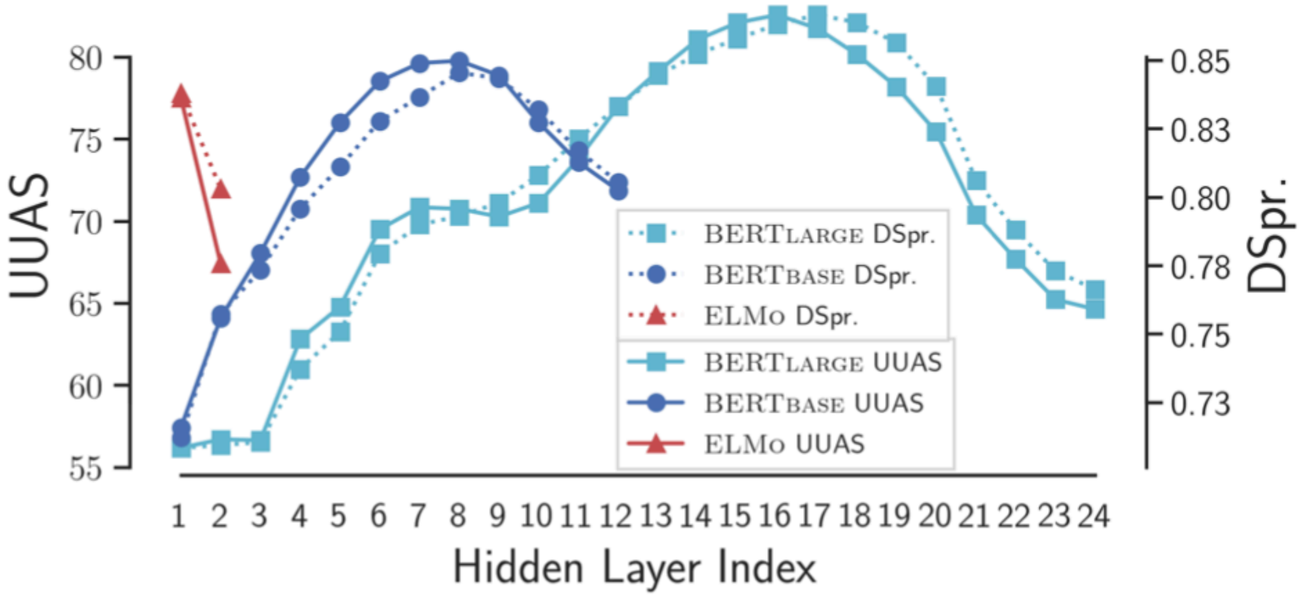
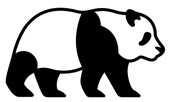


Figure 1: Parse distance UUAS and distance Spearman correlation across the BERT and ELMo model layers.

UUAS: Undirected Unlabeled Attachment Score
DSpr: the average Spearman correlation of true to predicted distances

Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019



Experiments: Rank of Matrix B

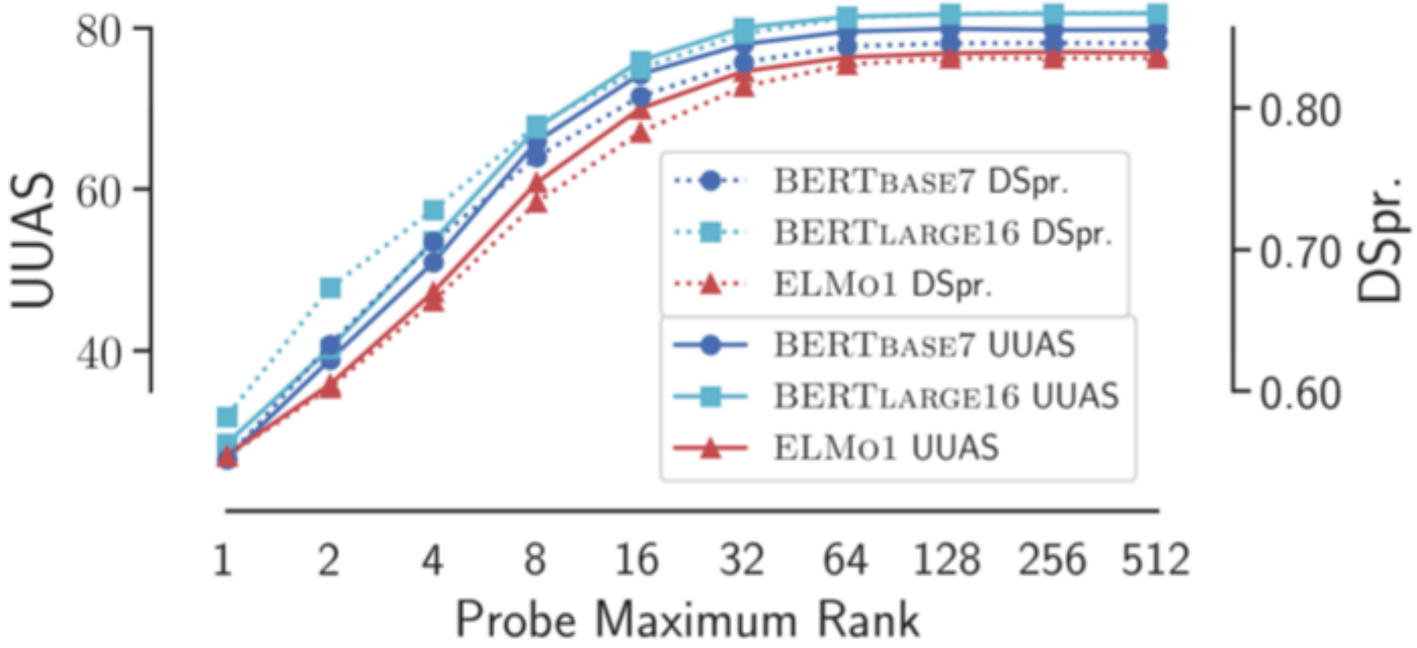


Figure 5: Parse distance tree reconstruction accuracy when the linear transformation is constrained to varying maximum dimensionality.

Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019

Take aways

- The structure of syntax trees emerges through properly defined distances and norms on two deep models' word representation spaces (ELMo and BERT).
- Different layers have differences in terms of syntax information.
- The transformation matrix B can be low-rank.
- Future work: design probes for testing the existence of different types of graph structures on any neural representation of language?

Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019

Todo

- **Reading assignment 10:**
  - Rethinking Self-Attention: Towards Interpretability in Neural Parsing https://aclanthology.org/2020.findings-emnlp.65.pdf
  - **Due date: April 15 23:59 pm, 2022 (EST timezone)**
- **Suggested Readings:**
  - Speech and Language Processing (3rd ed. draft). Dan Jurafsky and James H. Martin, Chapter 14: Dependency Parsing
  - The papers mentioned in our slides
  - Globally Normalized Transition-Based Neural Networks: https://arxiv.org/pdf/1603.06042.pdf
  - Universal Dependencies: A cross-linguistic typology: https://nlp.stanford.edu/~manning/papers/USD_LREC14_UD_revision.pdf
  - Check NLP Progress website: http://nlpprogress.com

References

1. **Stanford CS224N, Winter 2019:** https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/slides/cs224n-2019-lecture05-dep-parsing.pdf
2. **Emory University CS571**: Natural Language Processing, Jinho D. Choi
3. **SFU Nat LangLab CMPT 413/825**: Natural Language Processing
4. **Speech and Language Processing (3rd ed. draft)**, Dan Jurafsky and James H. Martin, Chapter 14: Dependency Parsing
5. Hewitt and Manning, A Structural Probe for Finding Syntax in Word Representations, ACL 2019
6. https://www.aclweb.org/anthology/2020.findings-emnlp.65.pdf

Thanks! Q&A

Bang Liu
Email: bang.liu@umontreal.ca
Homepage: http://www-labs.iro.umontreal.ca/~liubang/