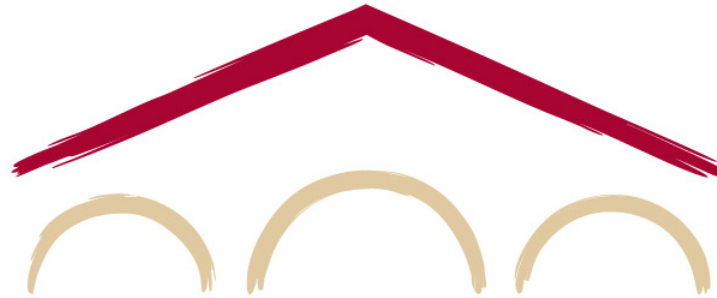


Natural Language Processing with Deep Learning

CS224N/Ling284



Christopher Manning
Lecture 4: Dependency Parsing

Lecture Plan

Syntactic Structure and Dependency parsing

1. Syntactic Structure: Consistency and Dependency (30 mins)
2. Dependency Grammar and Treebanks (15 mins)
3. Transition-based dependency parsing (15 mins)
4. Neural dependency parsing (20 mins)

Reminders/comments:

In Assignment 3, out on Tuesday, you build a neural dependency parser using PyTorch!

Start installing and learning PyTorch (Ass 3 is quite scaffolded)

Come to the PyTorch tutorial, Friday 1:30pm Friday (find it on Canvas)

Final project discussions – **come meet with us**; focus of Thursday class in week 4

1. The linguistic structure of sentences – two views: Constituency = phrase structure grammar = context-free grammars (CFGs)

Phrase structure organizes words into nested constituents

Starting unit: words

the, cat, cuddly, by, door

Words combine into phrases

the cuddly cat, by the door

Phrases can combine into bigger phrases

the cuddly cat by the door

The linguistic structure of sentences – two views: Constituency = phrase structure grammar = context-free grammars (CFGs)

Phrase structure organizes words into nested constituents.

the cat
a dog
large in a crate
barking on the table
cuddly by the door
large barking

talk to

walked behind

Two views of linguistic structure: Dependency structure

- Dependency structure shows which words depend on (modify, attach to, or are arguments of) which other words.

Look in the large crate in the kitchen by the door

Why do we need sentence structure?

Humans communicate complex ideas by composing words together into bigger units to convey complex meanings

Listeners need to work out what modifies [attaches to] what

A model needs to understand sentence structure in order to be able to interpret language correctly

Prepositional phrase attachment ambiguity

San Jose cops kill man with knife

Text Paper Translate Listen Close

San Jose cops kill man with knife

BBC Sign in News Sport Weather Shop Reel Travel

NEWS

Home Video World US & Canada UK Business Tech Science Stories

Science & Environment

Scientists count whales from space

By Jonathan Amos
BBC Science Correspondent

Prepositional phrase attachment ambiguity

Scientists count whales from space



Scientists count whales from space



PP attachment ambiguities multiply

- A key parsing decision is how we ‘attach’ various constituents
 - PPs, adverbial or participial phrases, infinitives, coordinations,

The board approved [its acquisition] [by Royal Trustco Ltd.]
[of Toronto]
[for \$27 a share]
[at its monthly meeting].

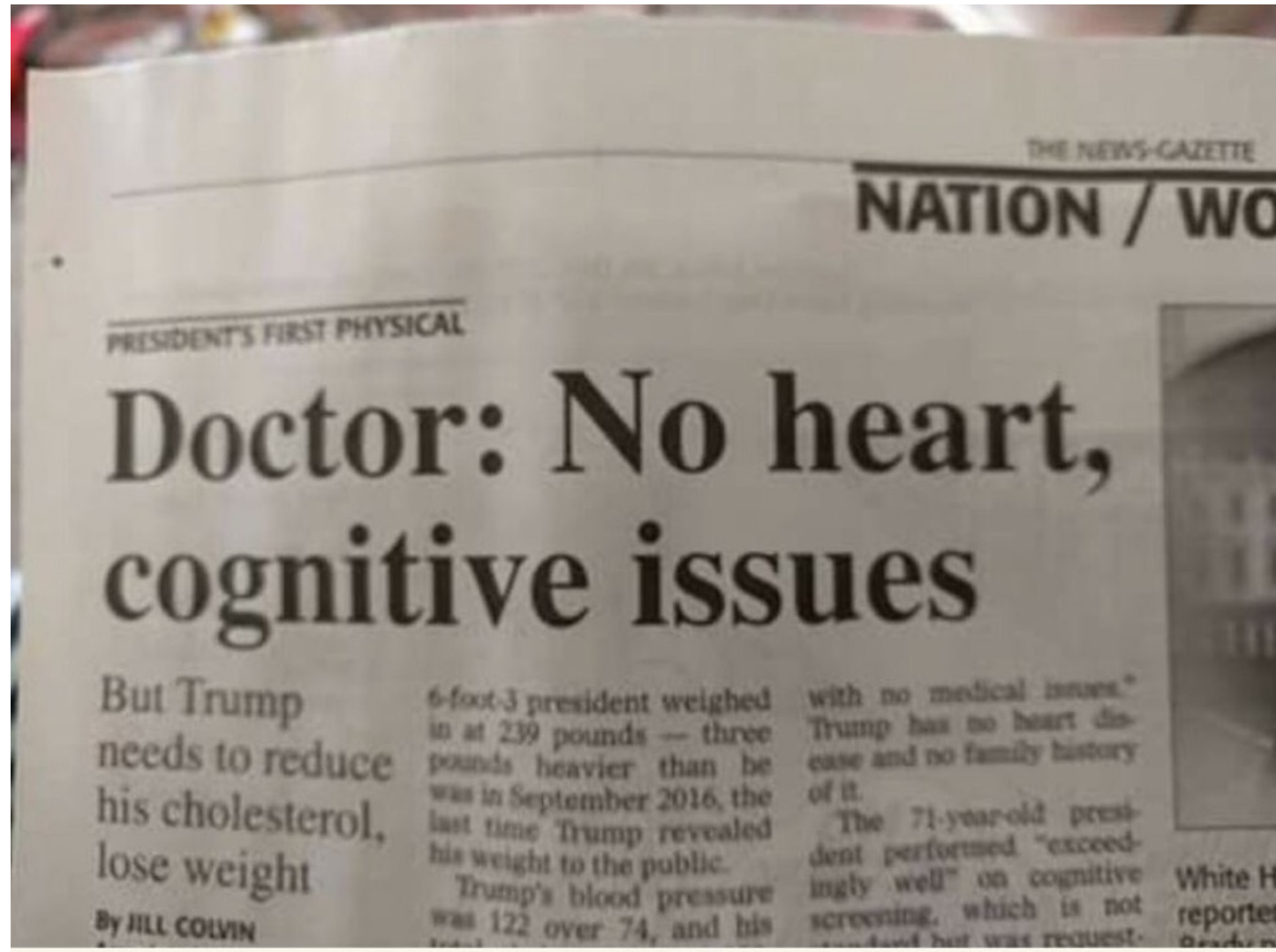
- Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$
- An exponentially growing series, which arises in many tree-like contexts:
 - E.g., the number of possible triangulations of a polygon with $n+2$ sides
 - Turns up in triangulation of probabilistic graphical models (CS228)....

Coordination scope ambiguity

Shuttle veteran and longtime NASA executive Fred Gregory appointed to board

Shuttle veteran and longtime NASA executive Fred Gregory appointed to board

Coordination scope ambiguity



Verb Phrase (VP) attachment ambiguity



The screenshot shows the top navigation bar of The Guardian website. It includes a dark blue header with a user profile icon, a search icon, and a menu icon (three dots). The word "theguardian" is displayed in white. Below this is a light gray breadcrumb trail: "home > world > americas", followed by a dark gray button labeled "asia" and a black button labeled "≡ all". The main content area has a white background. The section "Rio de Janeiro" is highlighted in blue. The headline "Mutilated body washes up on Rio beach to be used for Olympics beach volleyball" is displayed in a large, black, serif font.

theguardian

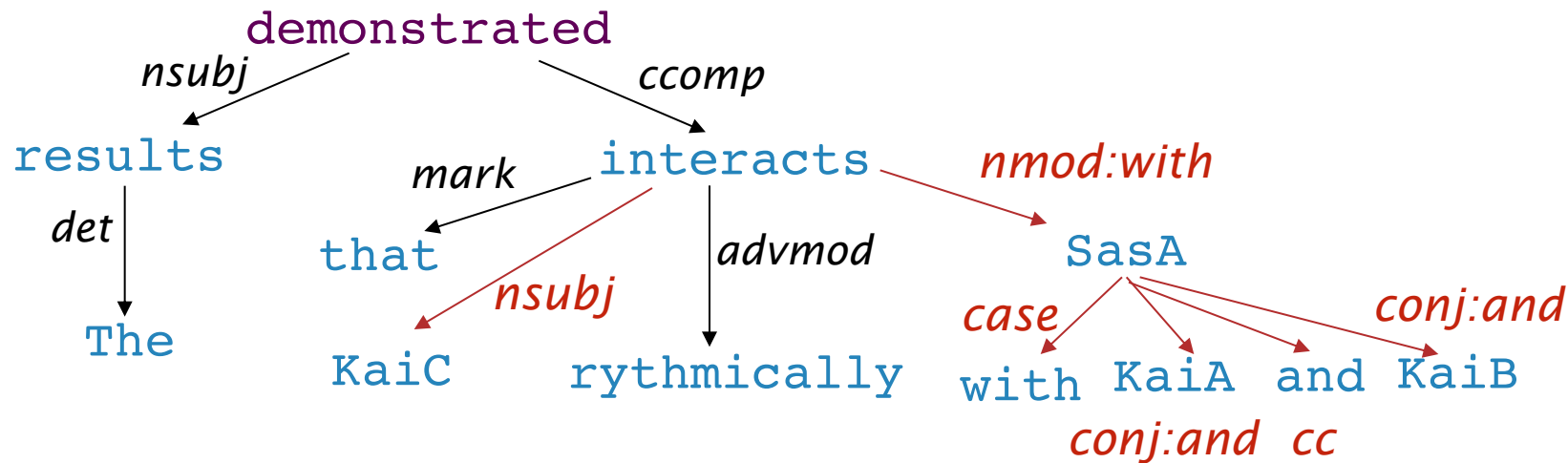
home > world > americas asia ≡ all

Rio de Janeiro

Mutilated body washes up on Rio beach to be used for Olympics beach volleyball

6/29/16, 1:48 PM

Dependency paths help extract semantic interpretation – simple practical example: extracting protein-protein interaction



KaiC \leftarrow nsubj interacts nmod:with \rightarrow SasA

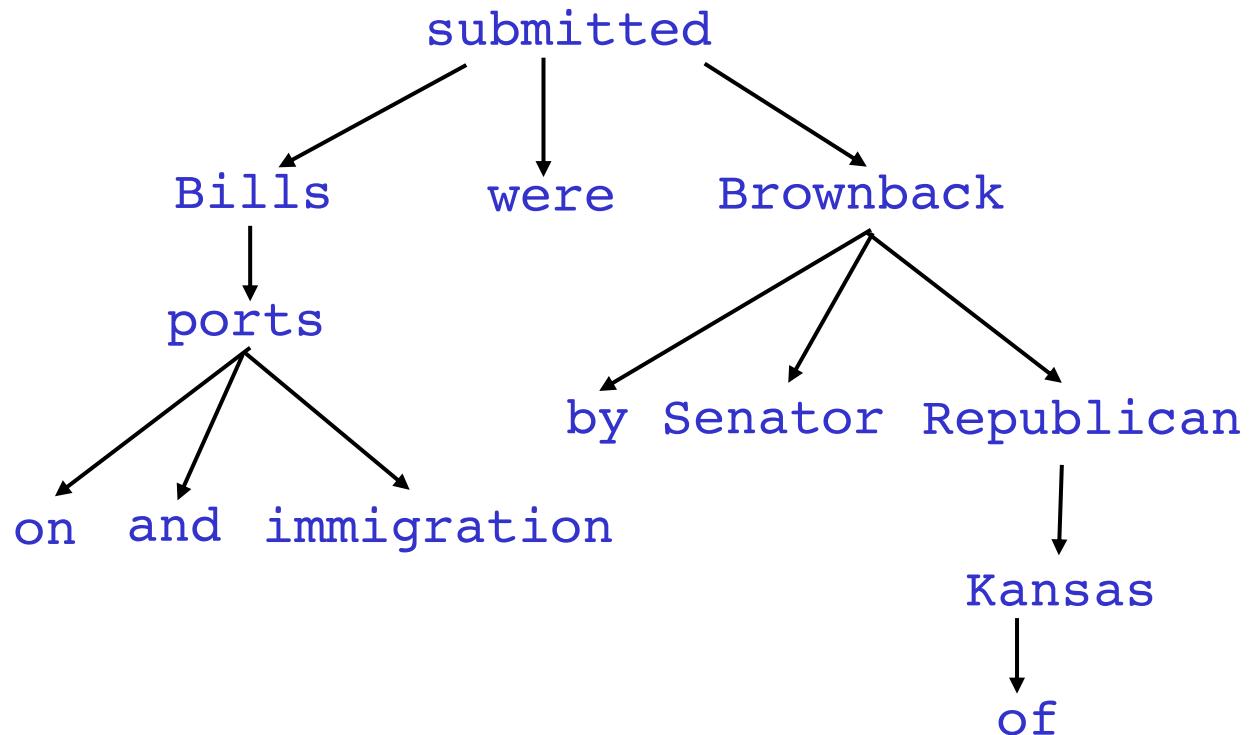
KaiC \leftarrow nsubj interacts nmod:with \rightarrow SasA conj:and \rightarrow KaiA

KaiC \leftarrow nsubj interacts nmod:with \rightarrow SasA conj:and \rightarrow KaiB

[Erkan et al. EMNLP 07, Fundel et al. 2007, etc.]

2. Dependency Grammar and Dependency Structure

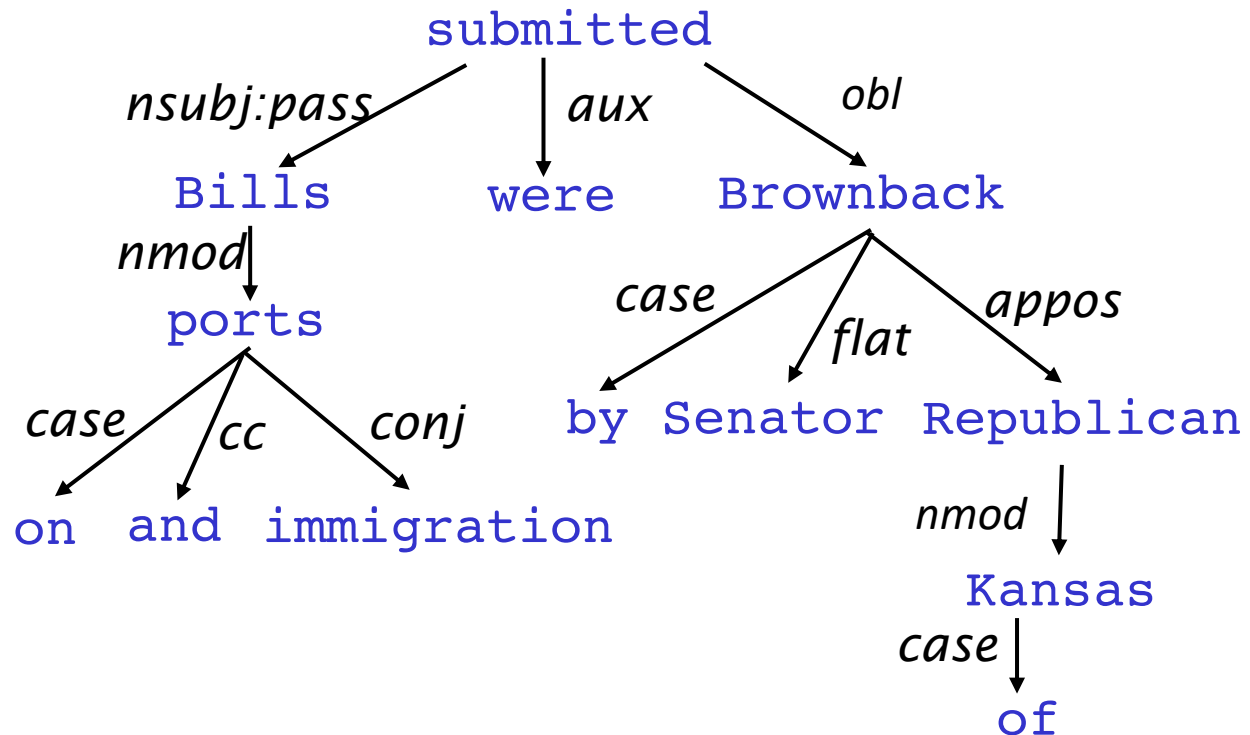
Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called **dependencies**



Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic 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.)

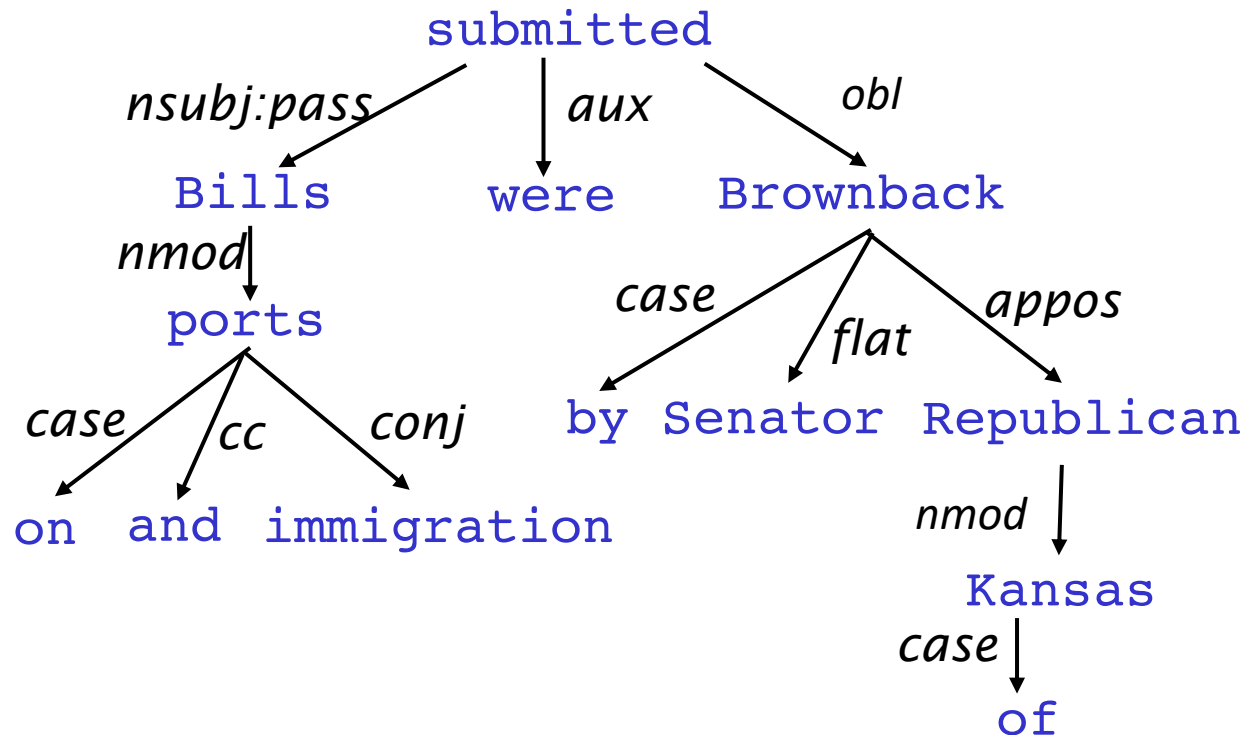


Dependency Grammar and Dependency Structure

Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called **dependencies**

An arrow connects a **head** (governor, superior, regent) with a **dependent** (modifier, inferior, subordinate)

Usually, dependencies form a tree (a connected, acyclic, single-root graph)



Pāṇini's grammar (c. 5th century BCE)



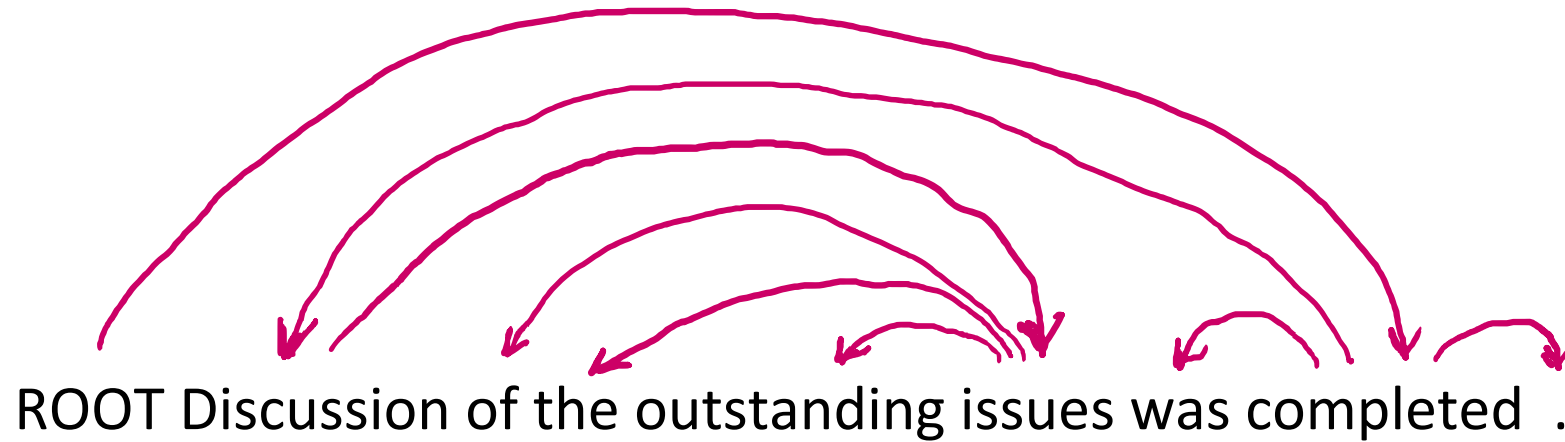
Gallery: <http://wellcomeimages.org/indexplus/image/L0032691.html>

CC BY 4.0 File: Birch bark MS from Kashmir of the Rupavatra Wellcome L0032691.jpg

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 grammar is a new-fangled invention
 - 20th century invention (R.S. Wells, 1947; then Chomsky 1953, etc.)
- Modern dependency work is often sourced to Lucien Tesnière (1959)
 - Was dominant approach in “East” in 20th Century (Russia, China, ...)
 - Good for free-er word order, inflected languages like Russian (or Latin!)
- Used in some of the earliest 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) and published on dependency grammar in *Language*

Dependency Grammar and Dependency Structure

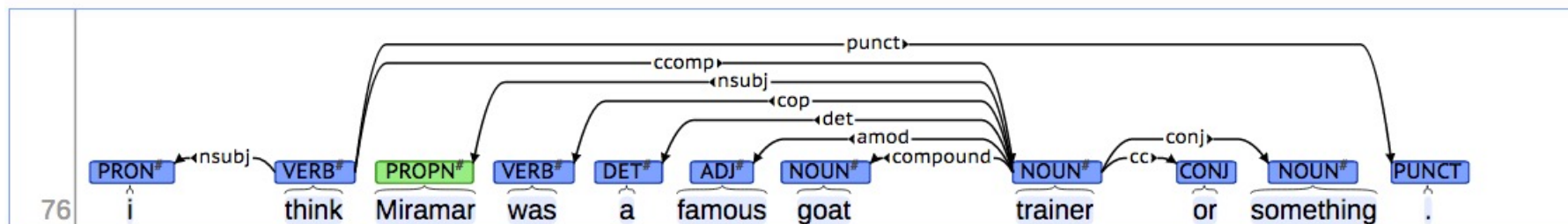


- Some people draw the arrows one way; some the other way!
 - Tesnière had them point from head to dependent – we follow that convention
- We usually add a fake ROOT so every word is a dependent of precisely 1 other node

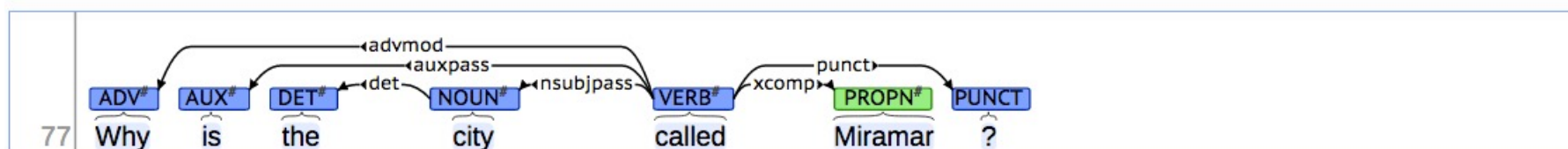
The rise of annotated data & Universal Dependencies treebanks

Brown corpus (1967; PoS tagged 1979); Lancaster-IBM Treebank (starting late 1980s);
Marcus et al. 1993, The Penn Treebank, *Computational Linguistics*;
Universal Dependencies: <http://universaldependencies.org/>

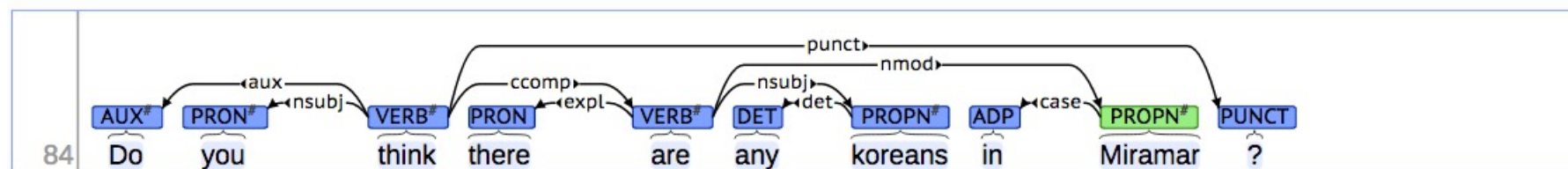
[context] [conllu]



[context] [conllu]



[context] [conllu]



The rise of annotated data

Starting off, building a treebank seems a lot slower and less useful than writing a grammar (by hand)

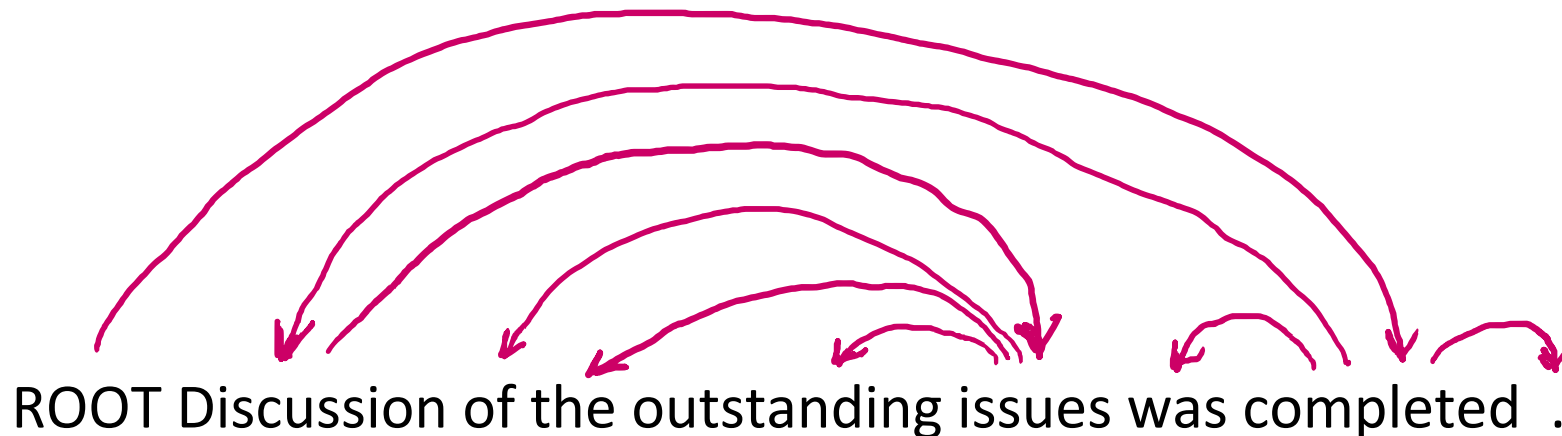
But a treebank gives us many things

- Reusability of the labor
 - Many parsers, part-of-speech taggers, etc. can be built on it
 - Valuable resource for linguistics
- Broad coverage, not just a few intuitions
- Frequencies and distributional information
- A way to evaluate NLP systems

Dependency Conditioning Preferences

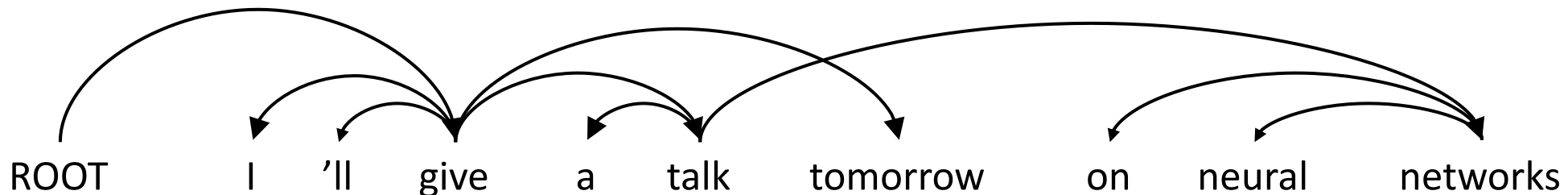
What are the straightforward sources of information for dependency parsing?

1. Bilexical affinities The dependency [discussion → issues] is plausible
2. Dependency distance Most dependencies are between nearby words
3. Intervening material Dependencies rarely span intervening verbs or punctuation
4. Valency of heads How many dependents on which side are usual for a head?



Dependency Parsing

- A sentence is parsed by choosing for each word what other word (including ROOT) it is a dependent of
- Usually some constraints:
 - Only one word is a dependent of ROOT
 - Don't want cycles $A \rightarrow B, B \rightarrow A$
- This makes the dependencies a tree
- Final issue is whether arrows can cross (be **non-projective**) or not



Projectivity

- Definition of a **projective parse**: There are no crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words
- Dependencies corresponding to a CFG tree must be **projective**
 - I.e., by forming dependencies by taking 1 child of each category as head
- Most syntactic structure is projective like this, but dependency theory normally does allow non-projective structures to account for displaced constituents
 - You can't easily get the semantics of certain constructions right without these nonprojective dependencies



3. Methods of Dependency Parsing

1. Dynamic programming

Eisner (1996) gives a clever algorithm with complexity $O(n^3)$, by producing parse items with heads at the ends rather than in the middle

2. Graph algorithms

You create a Minimum Spanning Tree for a sentence

McDonald et al.'s (2005) $O(n^2)$ MSTParser scores dependencies independently using an ML classifier (he uses MIRA, for online learning, but it can be something else)

Neural graph-based parser: Dozat and Manning (2017) et seq. – very successful!

3. Constraint Satisfaction

Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.

4. “Transition-based parsing” or “deterministic dependency parsing”

Greedy choice of attachments guided by good machine learning classifiers

E.g., MaltParser (Nivre et al. 2008). Has proven highly effective. And fast.

Greedy transition-based parsing [Nivre 2003]



- A simple form of greedy discriminative dependency parser
- The parser does a sequence of bottom-up actions
 - Roughly like “shift” or “reduce” in a shift-reduce parser – CS143, anyone?? – but the “reduce” actions are specialized to create dependencies with head on left or right
- The parser has:
 - a stack σ , written with top to the right
 - which starts with the ROOT symbol
 - a buffer β , written with top to the left
 - which starts with the input sentence
 - a set of dependency arcs A
 - which starts off empty
 - a set of actions

Basic transition-based dependency parser

Start: $\sigma = [\text{ROOT}]$, $\beta = w_1, \dots, w_n$, $A = \emptyset$

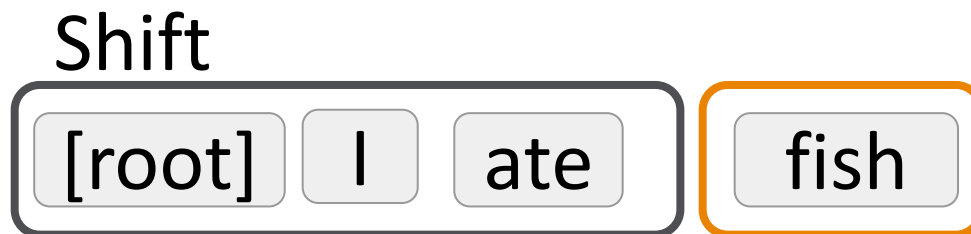
1. Shift $\sigma, w_i | \beta, A \rightarrow \sigma | w_i, \beta, A$
2. Left-Arc_r $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_j, \beta, A \cup \{r(w_j, w_i)\}$
3. Right-Arc_r $\sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_i, \beta, A \cup \{r(w_i, w_j)\}$

Finish: $\sigma = [w]$, $\beta = \emptyset$

Arc-standard transition-based parser

(there are other transition schemes ...)

Analysis of “I ate fish”



Start: $\sigma = [\text{ROOT}]$, $\beta = w_1, \dots, w_n$, $A = \emptyset$

1. Shift $\sigma, w_i | \beta, A \rightarrow \sigma | w_i, \beta, A$

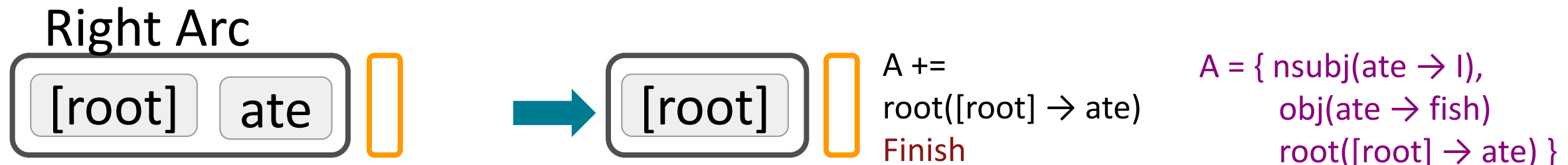
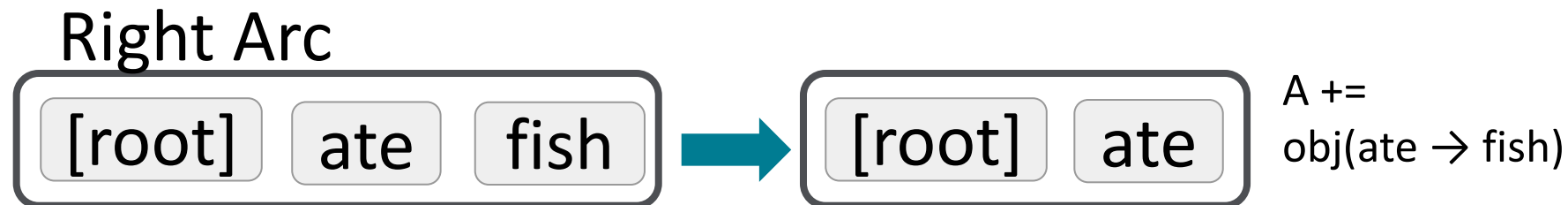
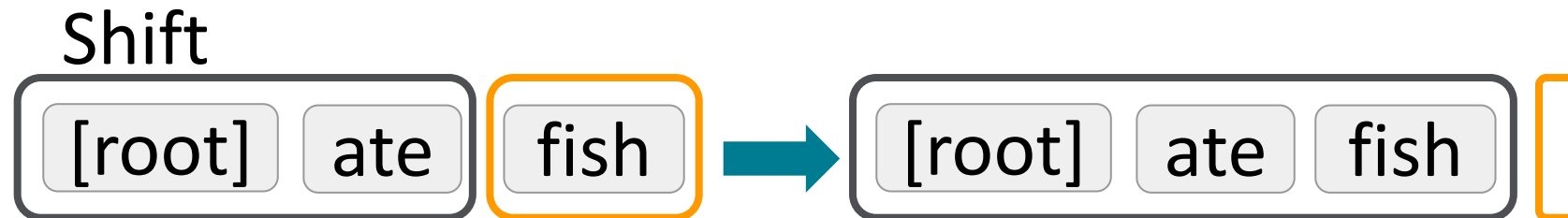
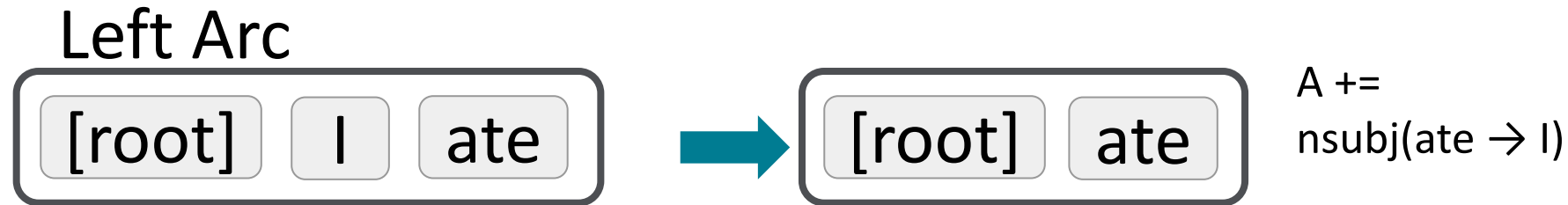
2. Left-Arc_r $\sigma | w_i | w_j, \beta, A \rightarrow$
 $\sigma | w_j, \beta, A \cup \{r(w_j, w_i)\}$

3. Right-Arc_r $\sigma | w_i | w_j, \beta, A \rightarrow$
 $\sigma | w_i, \beta, A \cup \{r(w_i, w_j)\}$

Finish: $\sigma = [w]$, $\beta = \emptyset$

Arc-standard transition-based parser

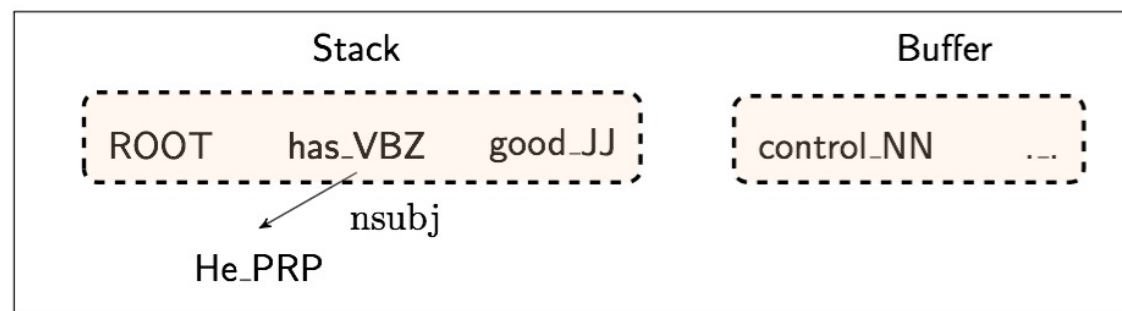
Analysis of “I ate fish”



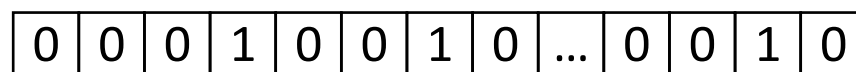
MaltParser [Nivre and Hall 2005]

- We have left to explain how we choose the next action 🙋
 - Answer: **Stand back, I know machine learning!**
- Each action is predicted by a discriminative classifier (e.g., softmax classifier) over each legal move
 - Max of 3 untyped choices (max of $|R| \times 2 + 1$ when typed)
 - Features: top of stack word, POS; first in buffer word, POS; etc.
- There is NO search (in the simplest form)
 - But you can profitably do a beam search if you wish (slower but better): You keep k good parse prefixes at each time step
- The model's accuracy is *fractionally* below the state of the art in dependency parsing, but
- It provides **very fast linear time parsing**, with high accuracy – great for parsing the web

Conventional Feature Representation



binary, sparse
dim = $10^6 - 10^7$

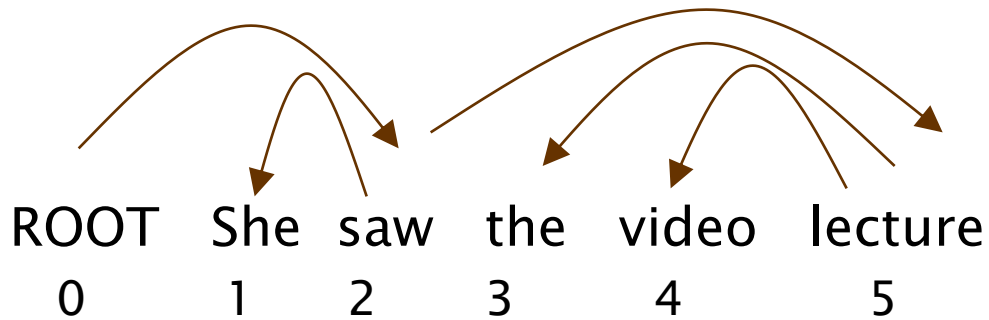


Feature templates: usually a combination of 1–3 elements from the configuration

Indicator features

$s1.w = \text{good} \wedge s1.t = \text{JJ}$
 $s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$
 $lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$
 $lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$

Evaluation of Dependency Parsing: (labeled) dependency accuracy



$$\text{Acc} = \frac{\text{\# correct deps}}{\text{\# of deps}}$$

$$\text{UAS} = 4 / 5 = 80\%$$

$$\text{LAS} = 2 / 5 = 40\%$$

Gold

1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed

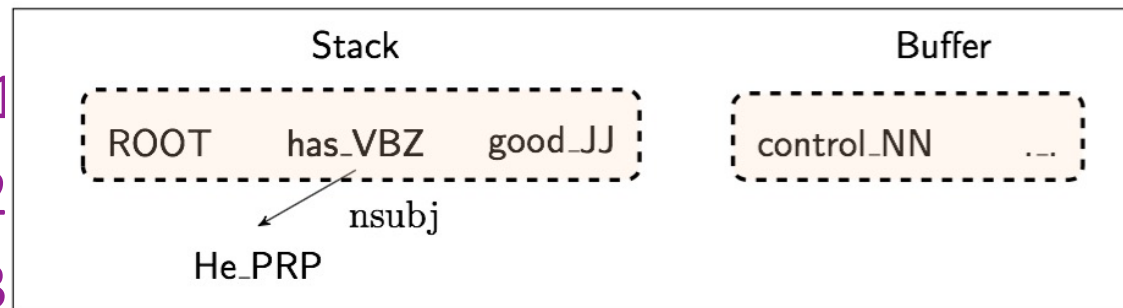
1	2	She	nsubj
2	0	saw	root
3	4	the	det
4	5	video	nsubj
5	2	lecture	ccomp

Handling non-projectivity

- The arc-standard algorithm we just presented only builds projective dependency trees
- Possible directions to head:
 1. Just declare defeat on nonprojective arcs 🙄
 2. Use dependency formalism which only has projective representations
 - A CFG only allows projective structures; you promote head of projectivity violations
 3. Use a postprocessor to a projective dependency parsing algorithm to identify and resolve nonprojective links
 4. Add extra transitions that can model at least most non-projective structures (e.g., add an extra SWAP transition will allow any non-projectivity, cf. bubble sort)
 5. Move to a parsing mechanism that does not use or require any constraints on projectivity (e.g., the graph-based MSTParser or Dozat and Manning (2017))

4. Why do we gain from a neural dependency parser? Indicator Features Revisited

- Problem #1
- Problem #2
- Problem #3



dense

dim = ~1000

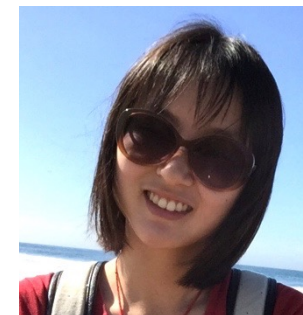
0.1	0.9	-0.2	0.3	...	-0.1	-0.5
-----	-----	------	-----	-----	------	------

More than 95% of parsing time is consumed by
feature computation

Neural Approach:

- $s1.w = \text{good} \wedge s1.t = \text{JJ}$
- $s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$
- $lc(s2).t = \text{PRP} \wedge s2.t = \text{VBZ} \wedge s1.t = \text{JJ}$
- $lc(s2).w = \text{He} \wedge lc(s2).l = \text{nsubj} \wedge s2.w = \text{has}$

A neural dependency parser [Chen and Manning 2014]

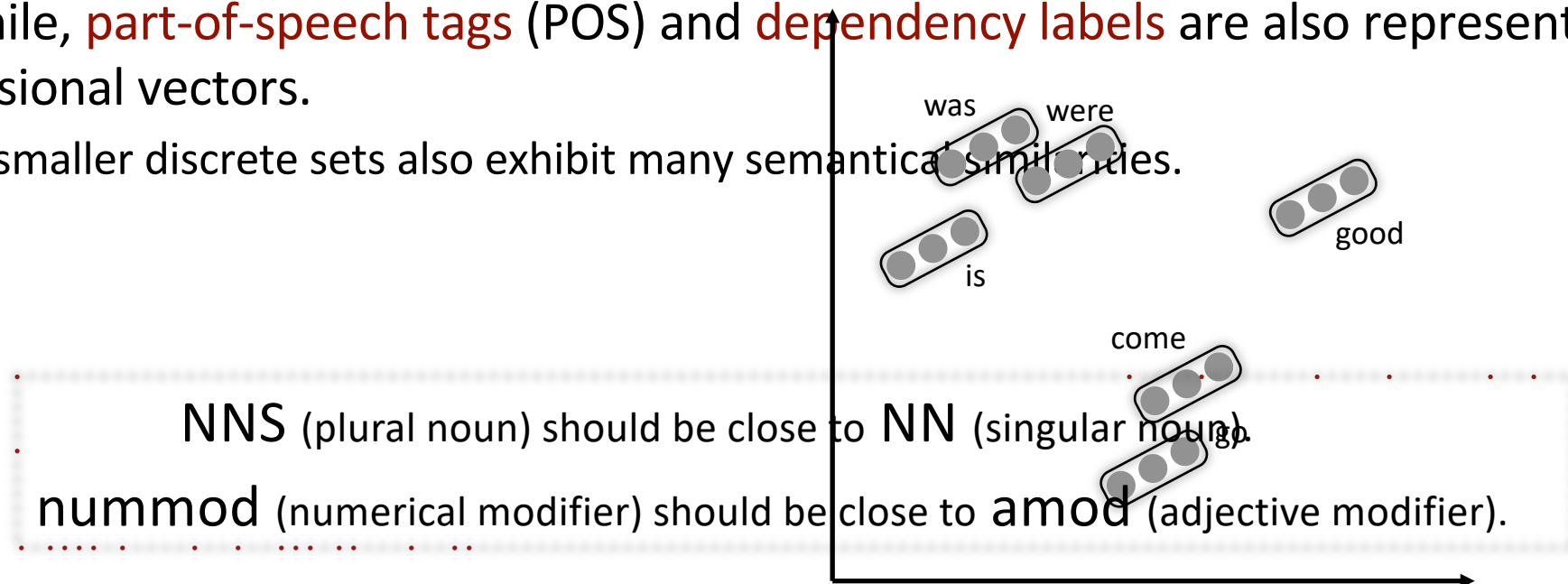


- Results on English parsing to Stanford Dependencies:
 - Unlabeled attachment score (UAS) = head
 - Labeled attachment score (LAS) = head and label

Parser	UAS	LAS	sent. / s
MaltParser	89.8	87.2	469
MSTParser	91.4	88.1	10
TurboParser	92.3	89.6	8
C & M 2014	92.0	89.7	654

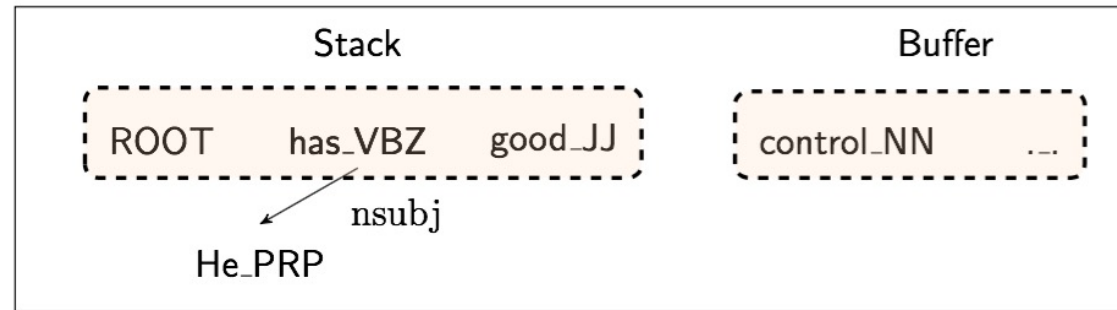
First win: Distributed Representations

- We represent each word as a d -dimensional dense vector (i.e., word embedding)
 - Similar words are expected to have close vectors.
- Meanwhile, **part-of-speech tags** (POS) and **dependency labels** are also represented as d -dimensional vectors.
 - The smaller discrete sets also exhibit many semantical similarities.



Extracting Tokens & vector representations from configuration

- We extract a set of tokens based on the stack / buffer positions:



	word	POS	dep.
s1	good	JJ	∅
s2	has	VBZ	∅
b1	control	NN	∅
lc(s1)	∅	+	+
rc(s1)	∅	∅	∅
lc(s2)	He	PRP	nsubj
rc(s2)	∅	∅	∅

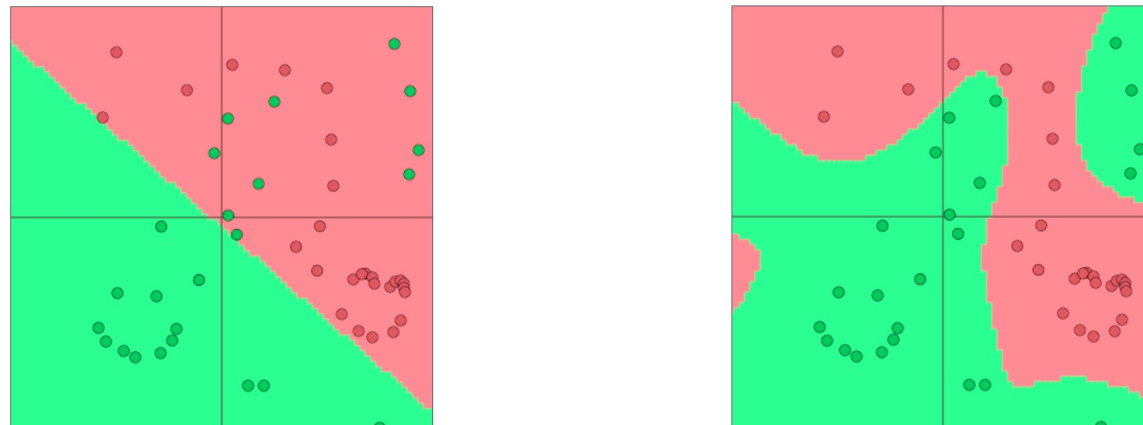
} A concatenation of the vector representation of all these is the neural representation of a configuration

Second win: Deep Learning classifiers are non-linear classifiers

- A **softmax classifier** assigns classes $y \in \mathcal{C}$ based on inputs $x \in \mathbb{R}^d$ via the probability:

$$p(y|x) = \frac{\exp(W_y \cdot x)}{\sum_{c=1}^C \exp(W_c \cdot x)}$$

- **Traditional ML classifiers** (including Naïve Bayes, SVMs, logistic regression and softmax classifier) are not very powerful classifiers: they only **give linear decision boundaries**
- But **neural networks** can use multiple layers to learn much more complex **nonlinear decision boundaries**



Simple feed-forward neural network multi-class classifier

Softmax probabilities

Log loss (cross-entropy error) will be back-propagated to the embeddings

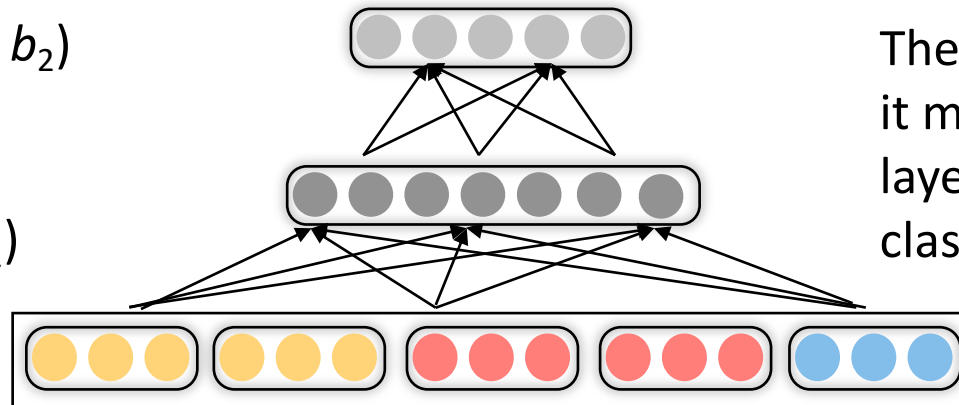
Output layer y

$$y = \text{softmax}(Uh + b_2)$$

Hidden layer h

$$h = \text{ReLU}(Wx + b_1)$$

Input layer x



The hidden layer re-represents the input — it moves inputs around in an intermediate layer vector space—so it can be easily classified with a (linear) softmax

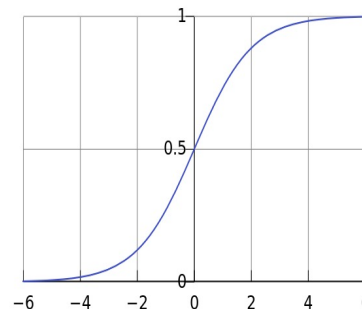
x is result of lookup

$$x_{(i, \dots, i+d)} = Le$$

lookup + concat

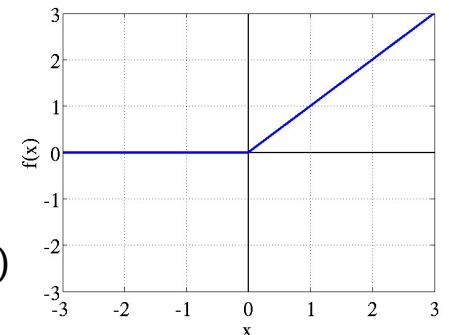
logistic = “sigmoid”

$$f(z) = \frac{1}{1 + \exp(-z)}$$



ReLU = Rectified Linear Unit

$$\text{ReLU}(z) = \max(z, 0)$$



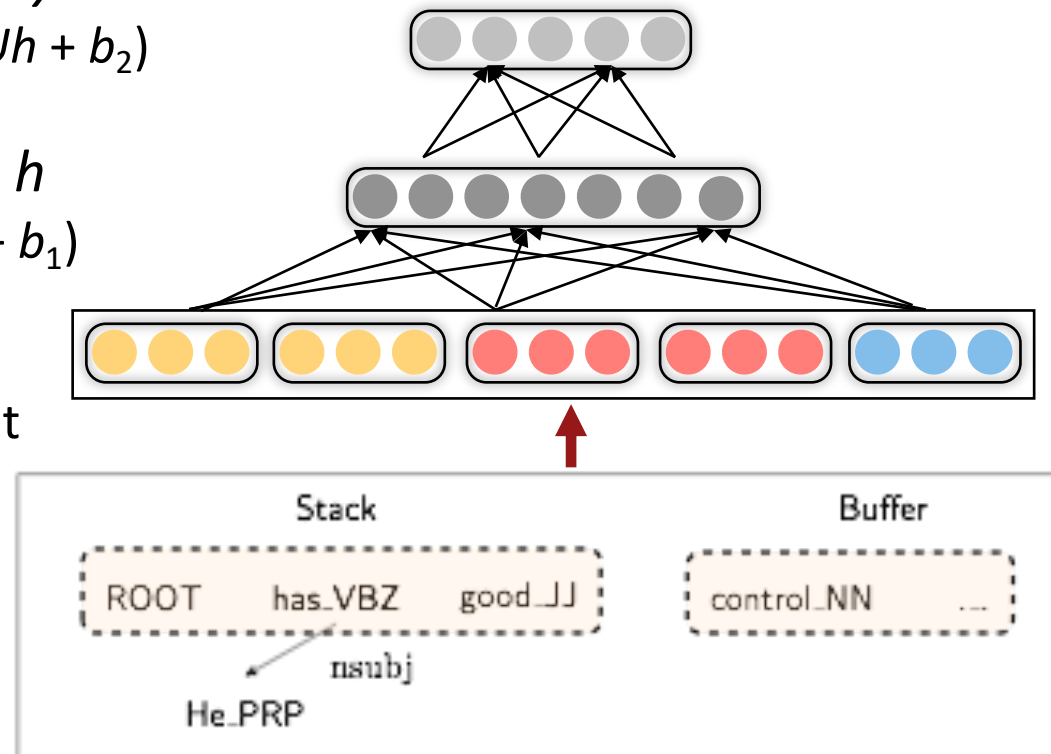
Neural Dependency Parser Model Architecture

Softmax probabilities

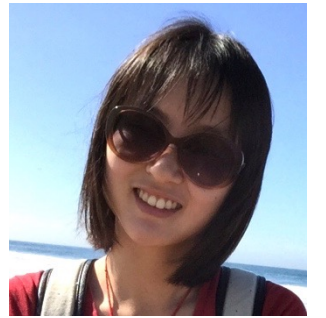
Output layer y
 $y = \text{softmax}(Uh + b_2)$

Hidden layer h
 $h = \text{ReLU}(Wx + b_1)$

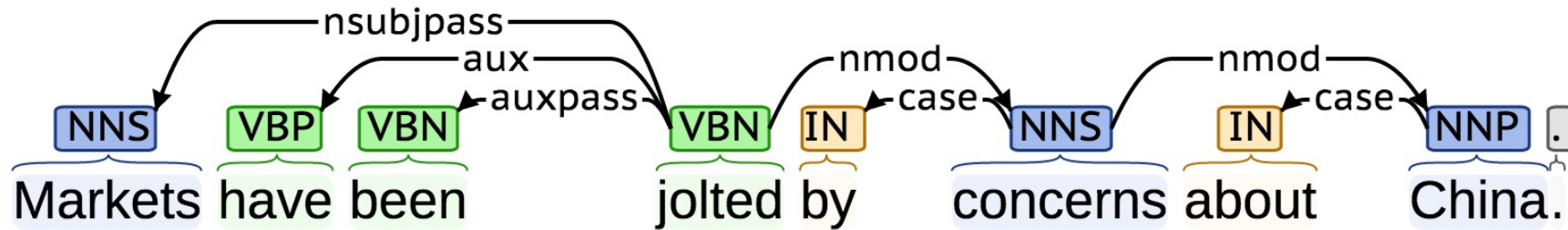
Input layer x
lookup + concat



Dependency parsing for sentence structure



Chen & Manning (2014) showed that neural networks can accurately determine the structure of sentences, supporting meaning interpretation



This paper was the first simple and successful neural dependency parser

The dense representations (and non-linear classifier) let it outperform other greedy parsers in both accuracy and speed

Further developments in transition-based neural dependency parsing

This work was further developed and improved by others, including in particular at Google

- Bigger, deeper networks with better tuned hyperparameters
- Beam search
- Global, conditional random field (CRF)-style inference over the decision sequence

Leading to SyntaxNet and the Parsey McParseFace model (2016):

“The World’s Most Accurate Parser”

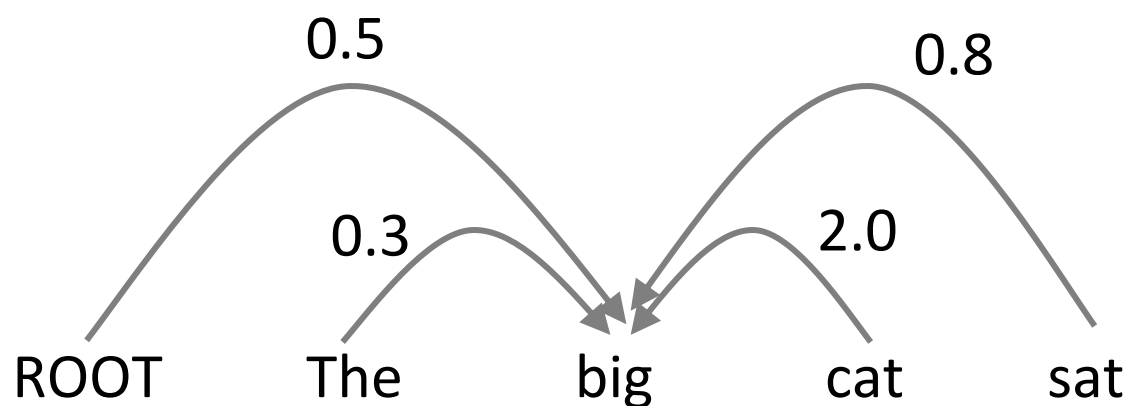
<https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html>



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

Graph-based dependency parsers

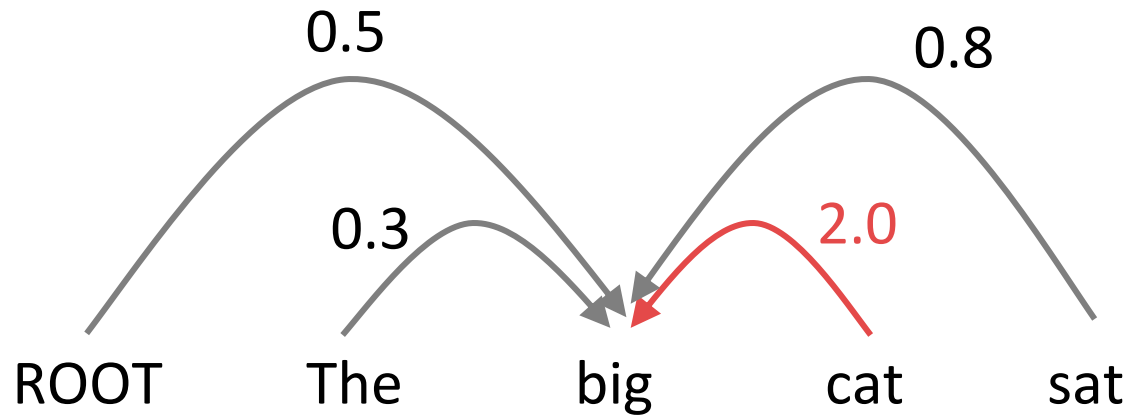
- Compute a score for every possible dependency for each word
 - Doing this well requires good “contextual” representations of each word token, which we will develop in coming lectures



e.g., picking the head for “big”

Graph-based dependency parsers

- Compute a score for every possible dependency (choice of head) for each word
 - Doing this well requires more than just knowing the two words
 - We need good “contextual” representations of each word token, which we will develop in the coming lectures
- Repeat the same process for each other word; find the best parse (MST algorithm)



e.g., picking the head for “big”

A Neural graph-based dependency parser

[Dozat and Manning 2017; Dozat, Qi, and Manning 2017]

- This paper revived interest in graph-based dependency parsing in a neural world
 - Designed a biaffine scoring model for neural dependency parsing
 - Also crucially uses a neural sequence model, something we discuss next week
- Really great results!
 - **But slower than the 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