Two views of linguistic structure: Constituency = phrase structure grammar = context-free grammars (CFGs)

Phrase structure organizes words into nested constituents.

Basic 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

Two views of linguistic structure: Constituency = phrase structure grammar = context-free grammars (CFGs)

Phrase structure organizes words into nested constituents.

Can represent the grammar with CFG rules

Basic unit: words

the, cat, cuddly, by, door

Det N Adj P N

Words combine into phrases

the cuddly cat, by the door

NP -> Det Adj N PP -> P NP

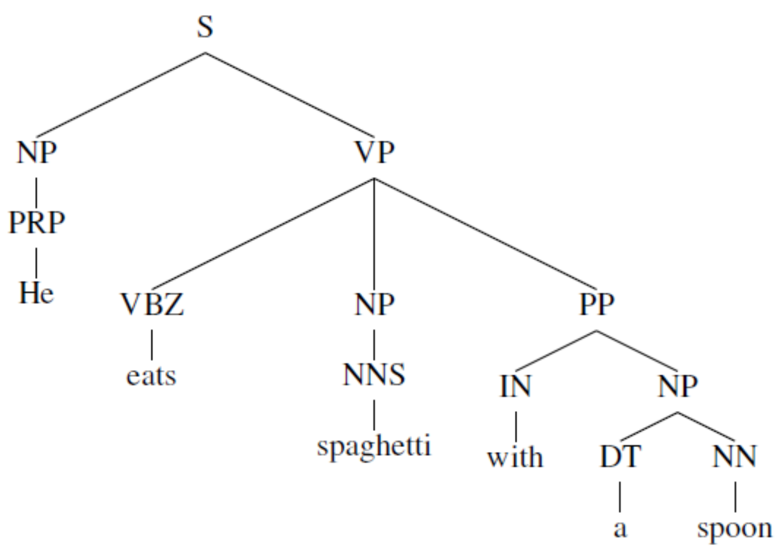
Phrases can combine into bigger phrases

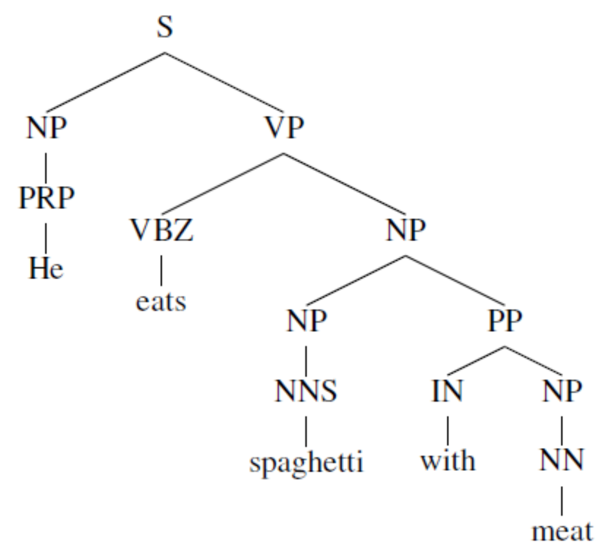
the cuddly cat by the door

NP -> NP PP

Example Constituency Trees

- PP attachment ambiguities in constituency structure





Two views of linguistic structure: Dependency structure

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

Look for the large barking dog by the door in a crate

Two views of linguistic structure: Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.
  - Determiners, adjectives, and (sometimes) verbs modify nouns

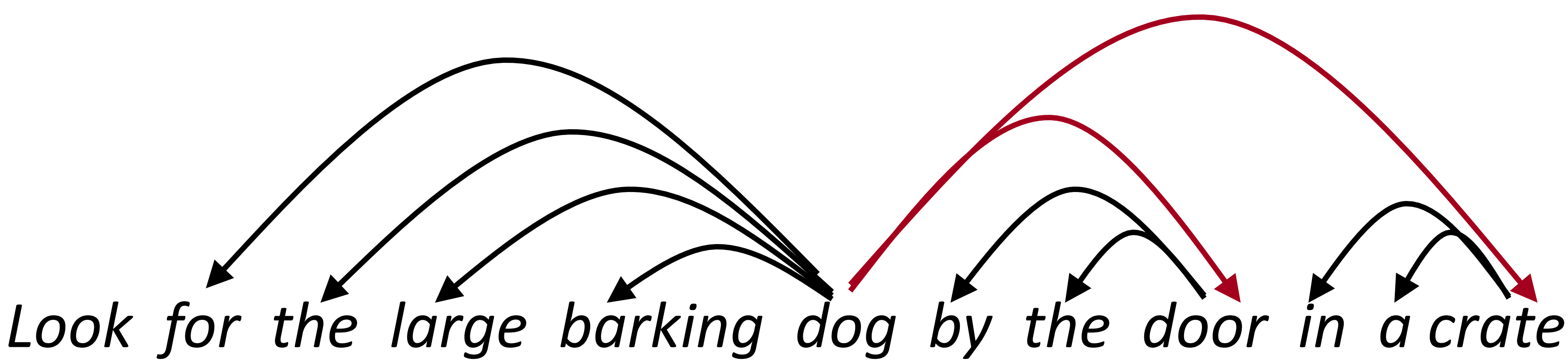
Look for the large barking dog by the door in a crate

Two views of linguistic structure: Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.
  - Determiners, adjectives, and (sometimes) verbs modify nouns
  - We will also treat prepositions as modifying nouns

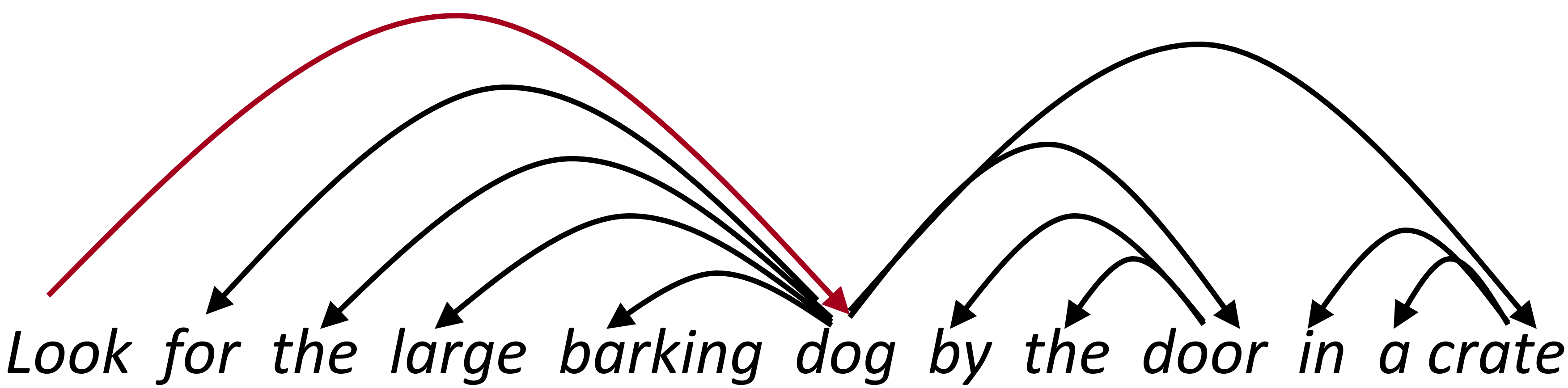
Two views of linguistic structure: Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.
  - Determiners, adjectives, and (sometimes) verbs modify nouns
  - We will also treat prepositions as modifying nouns
  - The prepositional phrases are modifying the main noun phrase



Two views of linguistic structure: Dependency structure

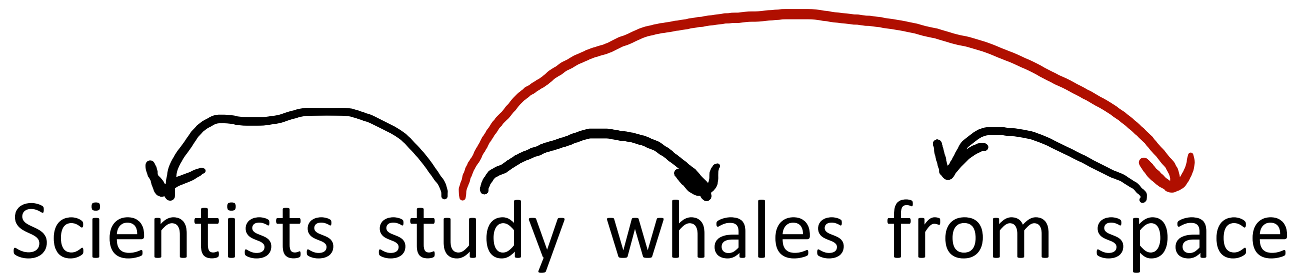
- Dependency structure shows which words depend on (modify or are arguments of) which other words.
  - Determiners, adjectives, and (sometimes) verbs modify nouns
  - We will also treat prepositions as modifying nouns
  - The prepositional phrases are modifying the main noun phrase
  - The main noun phrase is an argument of “look”

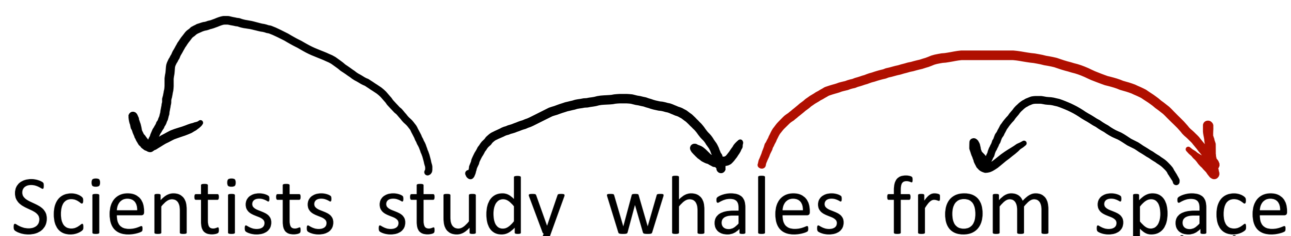


Ambiguity: PP attachments

Scientists study whales from space

PP attachment ambiguities in dependency structure





Attachment ambiguities

- 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].

Attachment ambiguities

- 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]

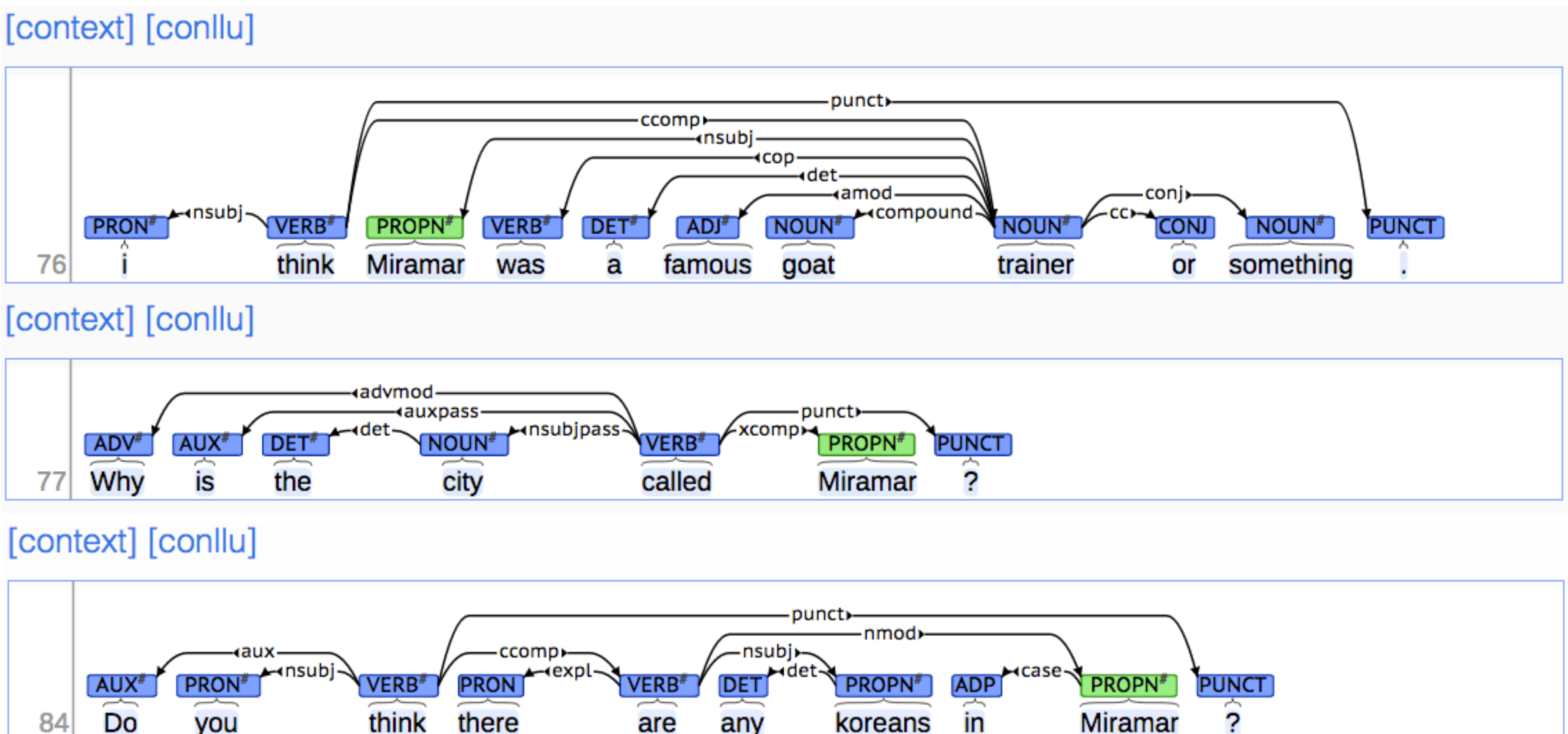
[for $27 a share]

[at its monthly meeting].

- Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$
- An exponentially growing series, which arises in many tree-like contexts
- But normally, we assume nesting.

The rise of annotated data: Universal Dependencies treebanks

[Universal Dependencies: http://universaldependencies.org/ ;
cf. Marcus et al. 1993, The Penn Treebank, *Computational Linguistics*]



The rise of annotated data

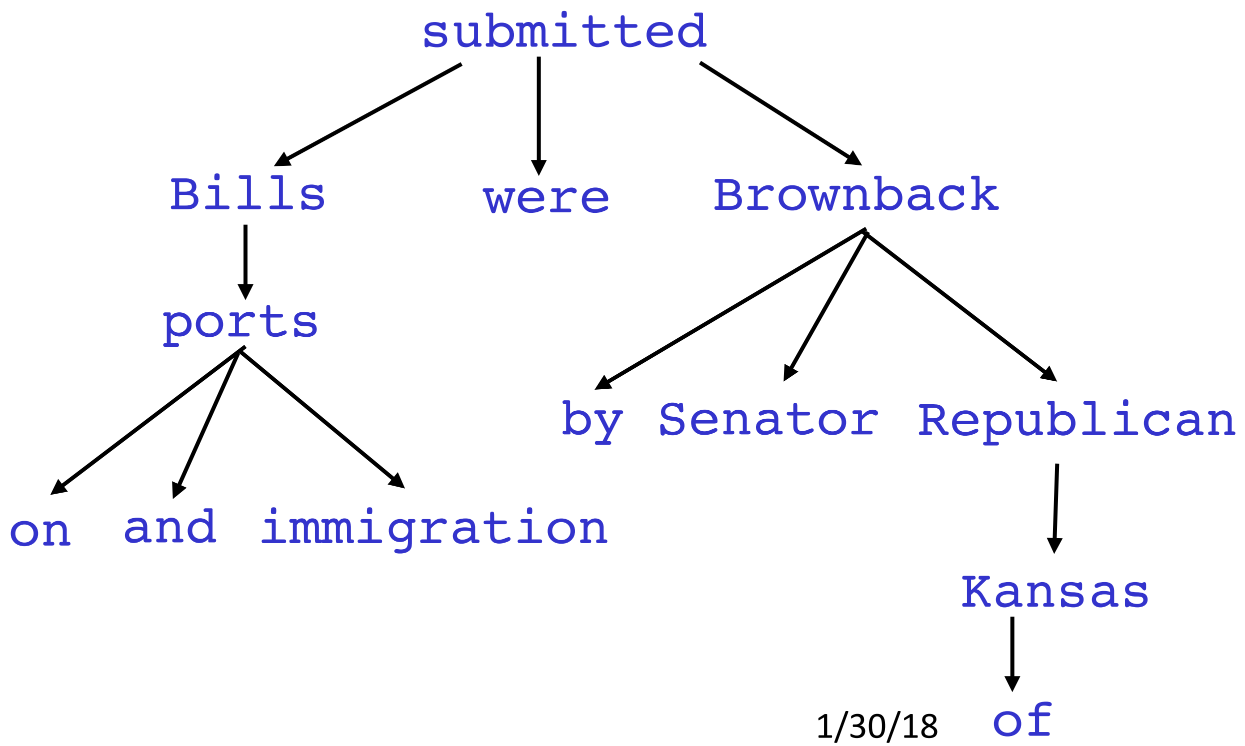
Starting off, building a treebank seems a lot slower and less useful than building a grammar

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 systems

Dependency Grammar and Dependency Structure

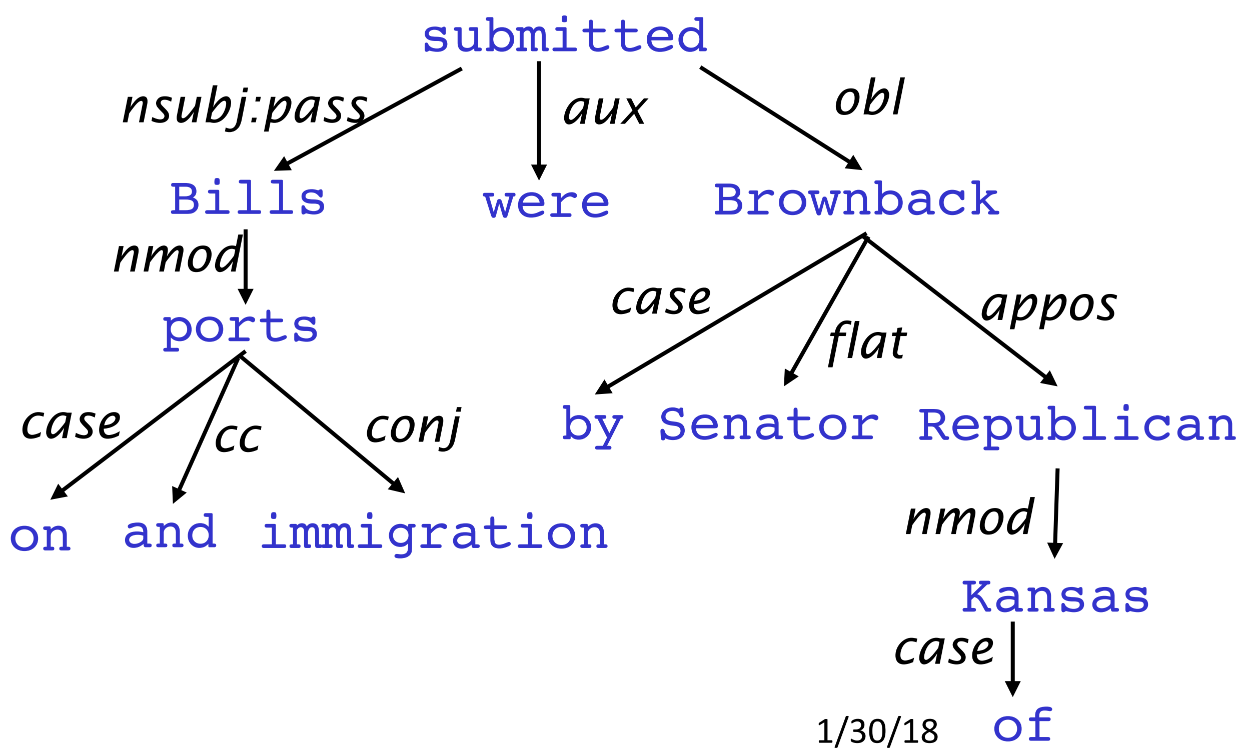
Dependency syntax postulates that syntactic structure consists of relations between lexical items, normally binary asymmetric relations (“arrows”) called dependencies



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.)

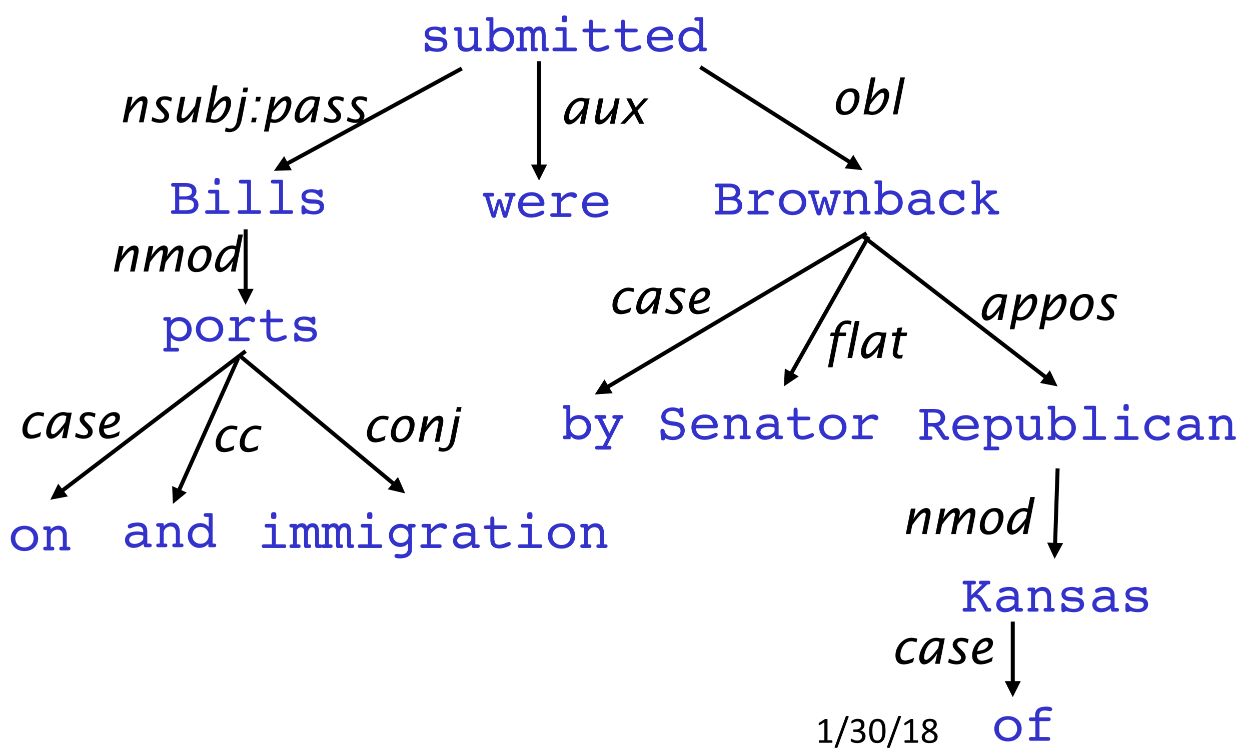


Dependency Grammar and Dependency Structure

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

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

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



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

Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)
https://web.stanford.edu/~jurafsky/slp3/14.pdf

Pāṇini's grammar (c. 5th century BCE)

Gallery: http://wellcomeimages.org/indexplus/image/L0032691.html
 File:Birch bark MS from Kashmir of the Rupavatra Welcome 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 grammars is a more recent invention
  - 20th century (R.S. Wells, 1947)
- Modern dependency work often linked to work of L. Tesnière (1959)
  - Was dominant approach in "East" (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)

Dependency Grammar and Dependency Structure

- Some people draw the arrows one way; some the other way!
  - Tesnière had them point from head to dependent…
  - Ours will point from head to dependent
- Usually add a fake ROOT so every word is a dependent of precisely 1 other node

Dependency Conditioning Preferences

What are the sources of information for dependency parsing?

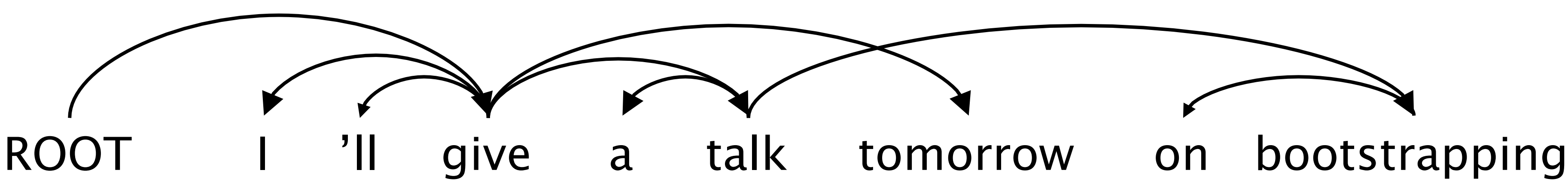
1. Bilexical affinities [discussion → issues] is plausible
2. Dependency distance mostly with nearby words
3. Intervening material

   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
  - i.e., find the right outgoing arrow from each word
- Usually some constraints:
  - Only one word is a dependent of ROOT
  - Don’t want cycles A → B, B → A
- This makes the dependencies a tree
- Final issue is whether arrows can cross (non-projective) or not



Methods of Dependency Parsing

1. Dynamic programming
2. Graph algorithms

   You create a Minimum Spanning Tree for a sentence

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

   MaltParser (Nivre et al. 2008). Has proven highly effective.

4. 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, 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: σ = [ROOT], $\beta = w_1, \ldots, w_n$, $A = \varnothing$

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

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

Arc-standard transition-based parser

(there are other transition schemes …)

Analysis of “I ate fish”

Start

[root] | I ate fish

Shift

[root] I | ate fish

Shift

[root] I ate | fish

Start: σ = [ROOT], $\beta = w_1, \ldots, w_n$, A = $\varnothing$

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

Finish: $\beta = \varnothing$

Arc-standard transition-based parser

Analysis of “I ate fish”

Left Arc

[root] I ate → [root] ate

A += nsubj(ate → I)

Shift

[root] ate | fish → [root] ate fish |

Right Arc

[root] ate fish → [root] ate

A += obj(ate → fish)

Right Arc

[root] ate | → [root] |

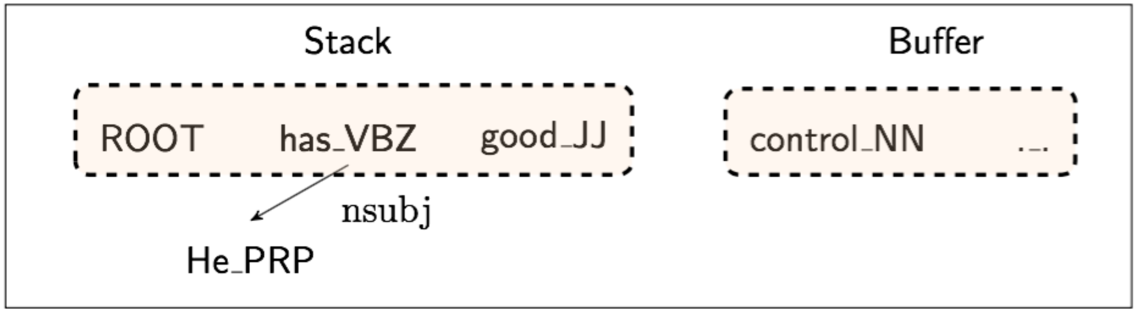
A += root([root] → ate)

Finish

MaltParser
[Nivre and Hall 2005]

- How could we choose the next action?
- Each action is predicted by a discriminative classifier (eg. SVM or logistic regression classifier) over each legal move
  - 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)
- It provides VERY fast linear time parsing
- The model's accuracy is only *slightly* below the best dependency parsers

Feature Representation



binary, sparse
dim $=10^6 \sim 10^7$

0	0	0	1	0	0	1	0	…	0	0	1	0

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

Indicator features

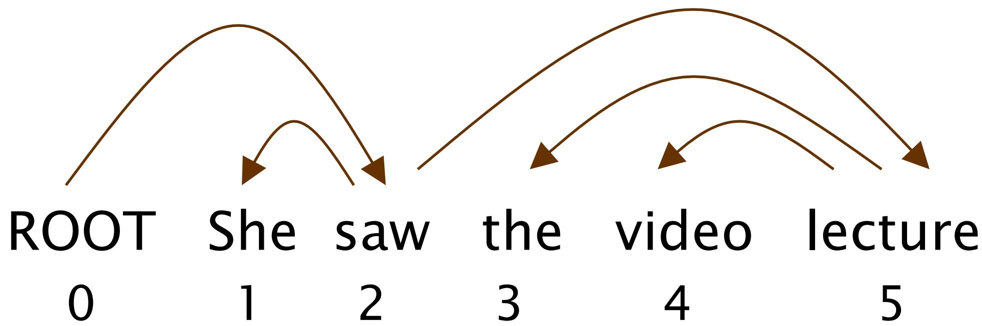
$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$

$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$

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

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

Evaluation of Dependency Parsing: (labeled) dependency accuracy



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

UAS = 4 / 5 = 80%

LAS = 2 / 5 = 40%

Gold

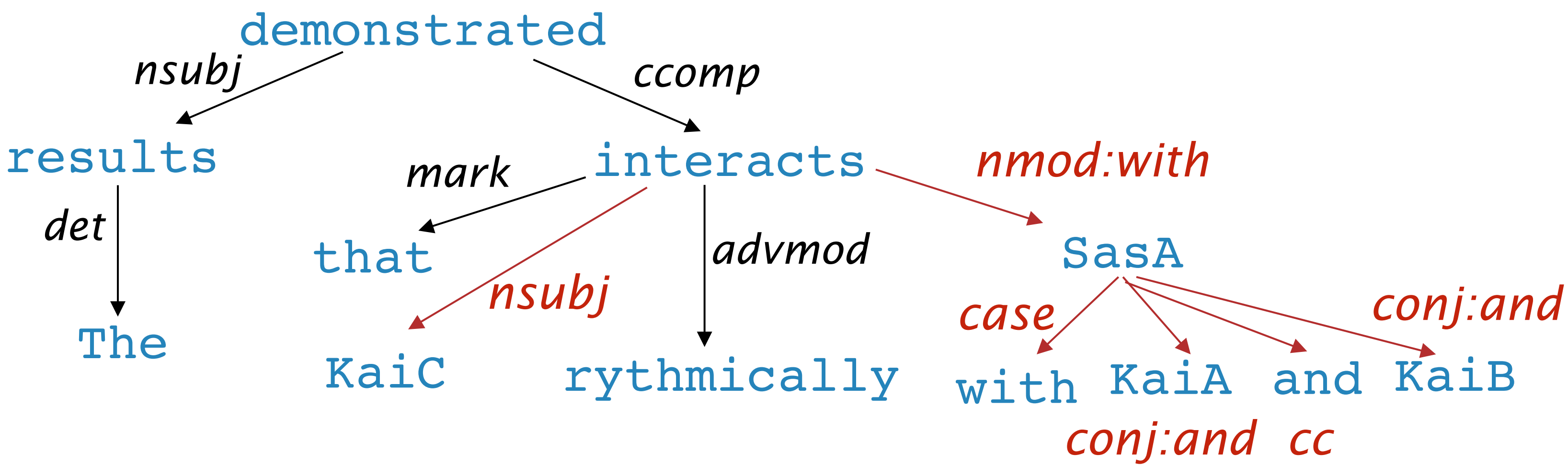
1	2	She	nsubj
2	0	saw	root
3	5	the	det
4	5	video	nn
5	2	lecture	obj

Parsed

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

Dependency paths identify semantic relations – e.g, for protein interaction

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



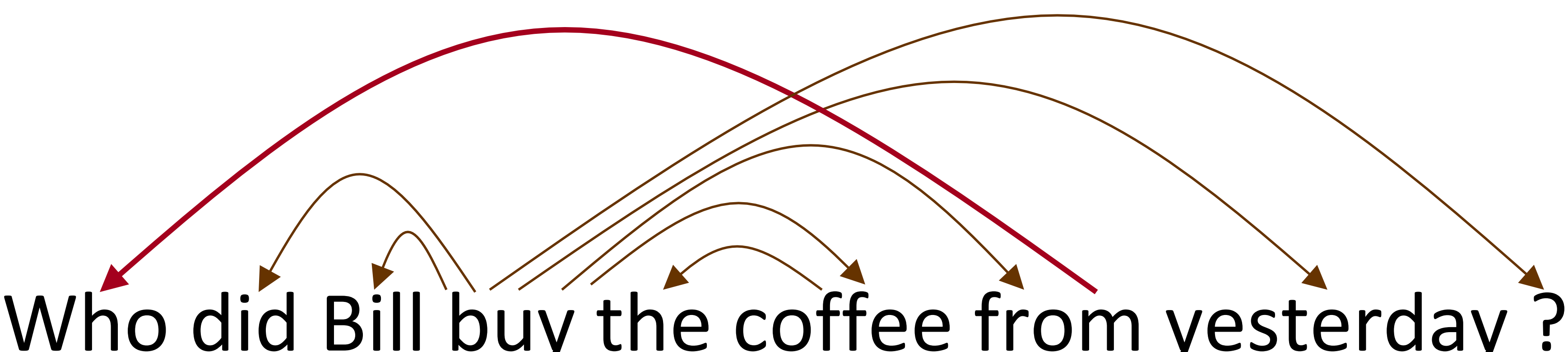
KaiC 🡐nsubj interacts nmod:with 🡒 SasA

KaiC 🡐nsubj interacts nmod:with 🡒 SasA conj:and🡒 KaiA

KaiC 🡐nsubj interacts prep_with🡒 SasA conj:and🡒 KaiB

Projectivity

- Dependencies parallel to a CFG tree must be projective
  - There must not be any crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words.
- 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

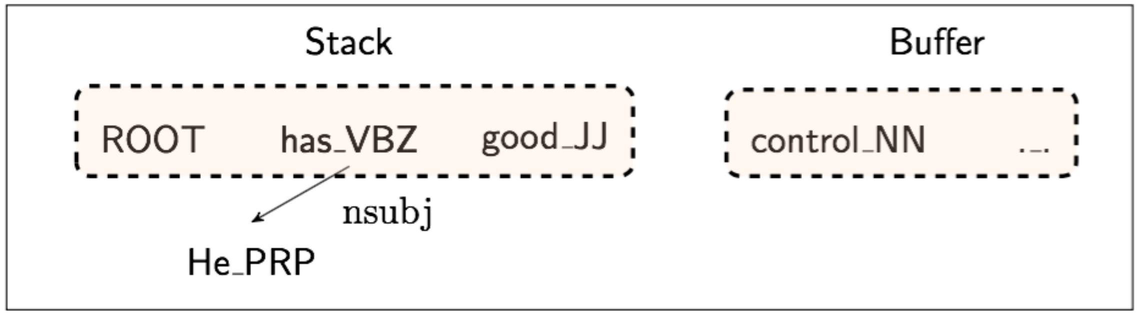


Handling non-projectivity

- The arc-standard algorithm we presented only builds projective dependency trees
- Possible directions:
  1. Just declare defeat on nonprojective arcs
  2. Use a dependency formalism which only admits projective representations (a CFG doesn’t represent such structures…)
  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, 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)

Why train 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.

Our Approach:

learn a dense and compact feature representation

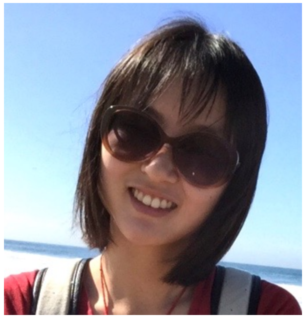
$$s1.w = \text{good} \wedge s1.t = \text{JJ}$$

$$s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$$

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

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

A neural dependency parser
[Chen and Manning 2014]



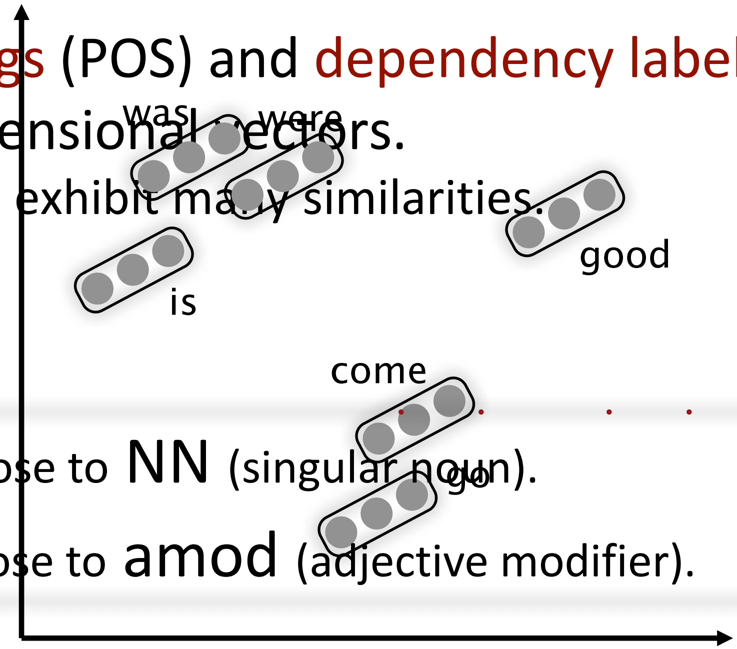
- English parsing to Stanford Dependencies:
  - Unlabeled attachment score (UAS) = head
  - Labeled attachment score (LAS) = head and label

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**

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 similarities.

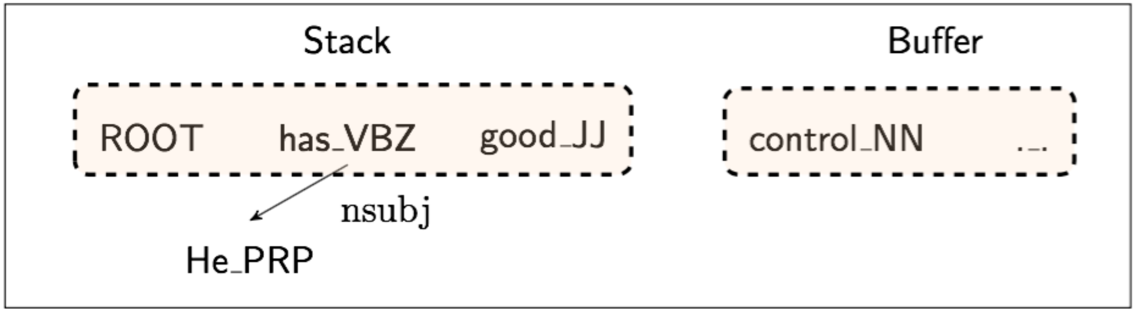


NNS (plural noun) should be close to NN (singular noun).

num (numerical modifier) should be close to amod (adjective modifier).

Extracting Tokens and then vector representations from configuration

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



	word	POS	dep.
s_1	good	JJ	∅
s_2	has	VBZ	∅
b_1	control	NN	∅
$lc(s_1)$	∅	∅	∅
$rc(s_1)$	∅	∅	∅
$lc(s_2)$	He	PRP	nsubj
$rc(s_2)$	∅	∅	∅

- We convert them to vector embeddings and concatenate them

Model Architecture

Softmax probabilities

Output layer y

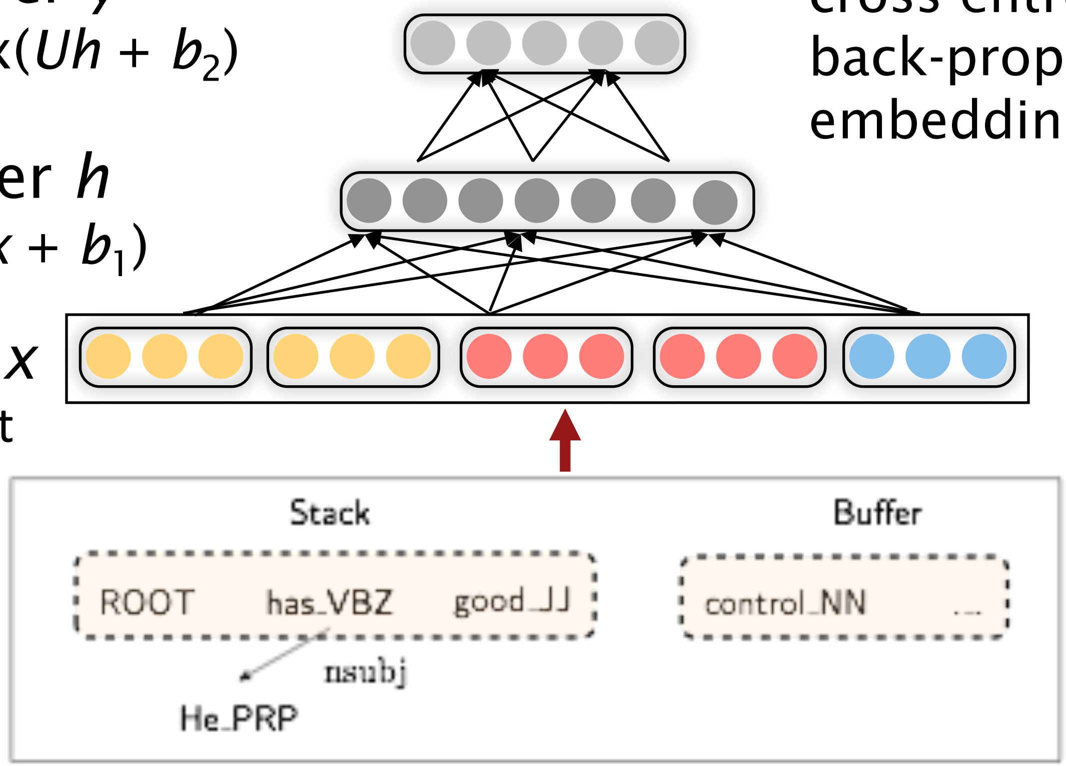
$y = \text{softmax}(Uh + b_2)$

Hidden layer h

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

Input layer x

lookup + concat



cross-entropy error will be back-propagated to the embeddings.