Deep Neural Networks for Natural Language Processing (AI6127)

JUNG-JAE KIM

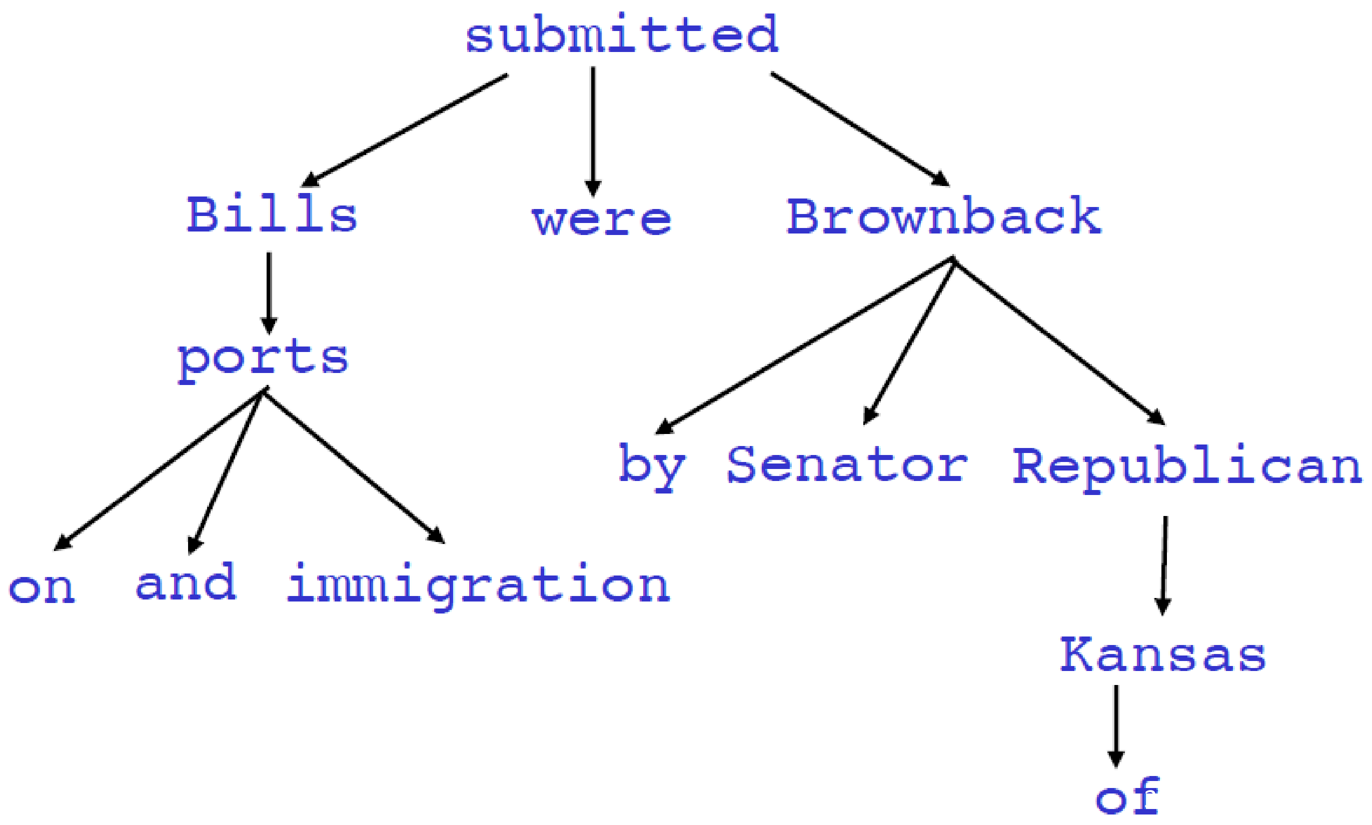
LECTURE 13: TRANSITION-BASED DEPENDENCY PARSING

Lecture Plan

- Recap: Dependency Grammar and Treebanks
- Transition-based dependency parsing
- Neural dependency parsing

1. Dependency Grammar and Dependency Structure

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



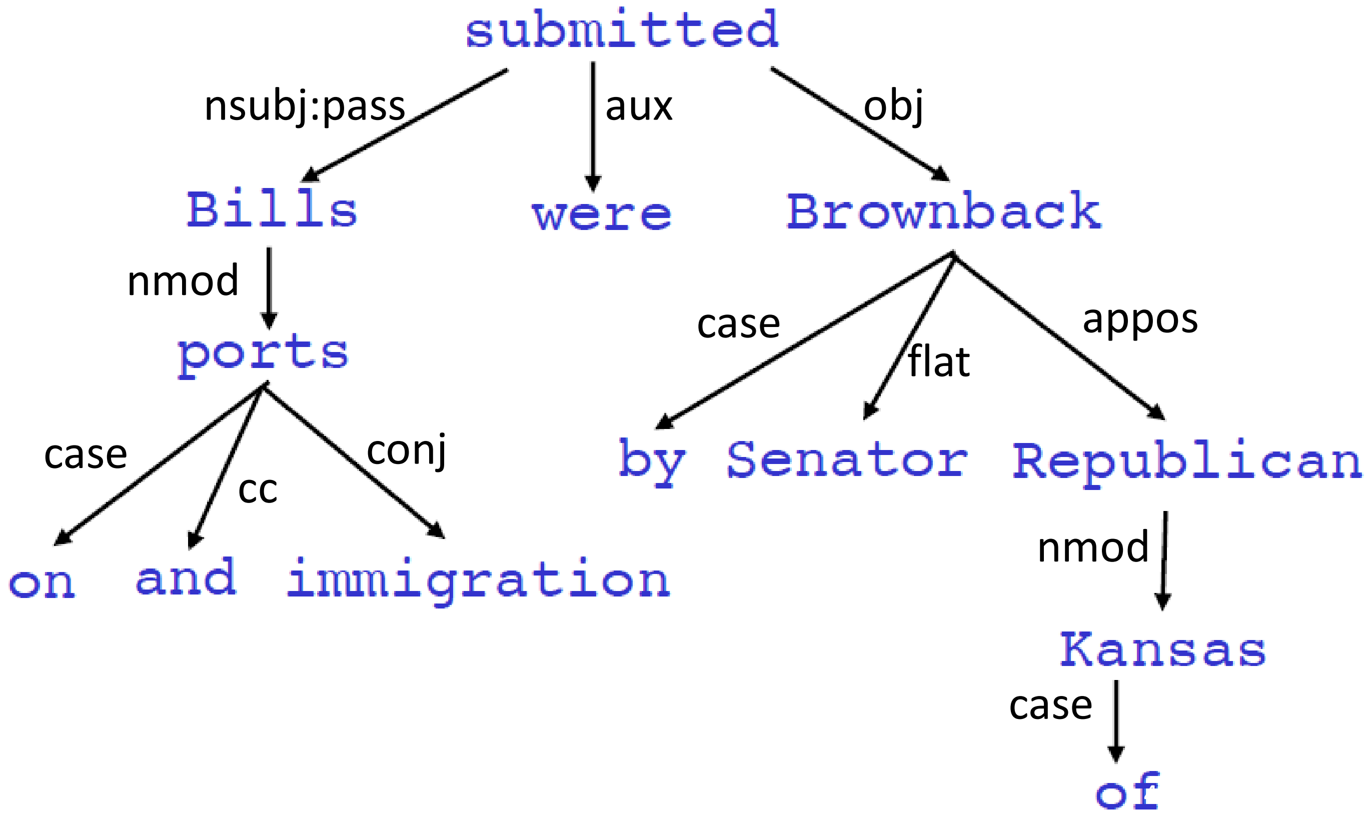
Dependency Grammar and Dependency Structure

- 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.)

- appos: apposition
- aux: auxiliary
- case: case marking
- cc: coordinator
- conj: conjunct
- flat: name
- nmod: nominal modifier
- nsubj:pass: nominal subject (passive)
- obj: object

See https://universaldependencies.org/en/dep/

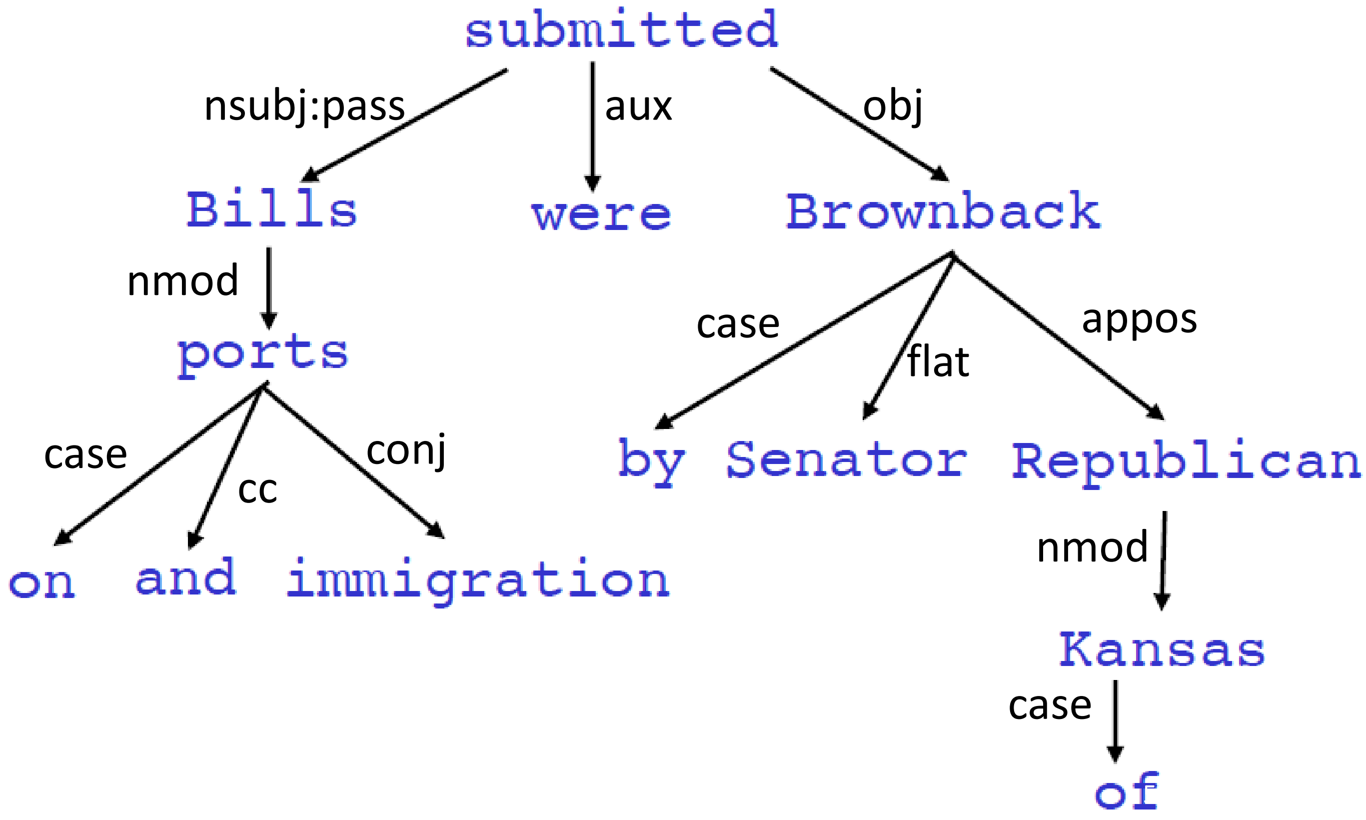


Dependency Grammar and Dependency Structure

- 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)



Pāṇini's grammar (c. 5th century BCE)

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

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)

Dependency Grammar and Dependency Structure

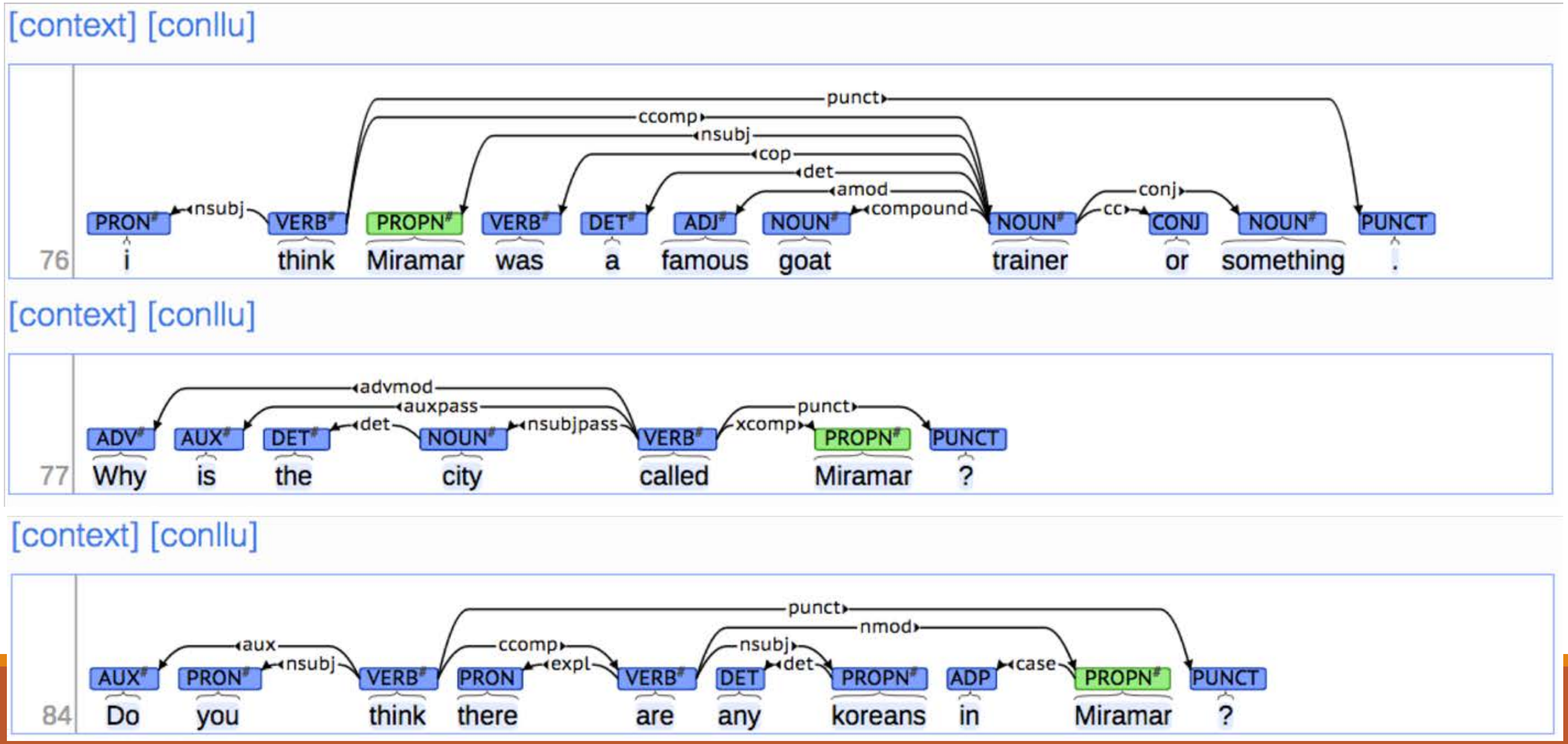
- Some people draw the arrows one way; some the other way!
  - Tesnière had them point from head to dependent…
- Usually add a fake ROOT so every word is a dependent of precisely 1 other node

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

The rise of annotated data: Universal Dependencies treebanks

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

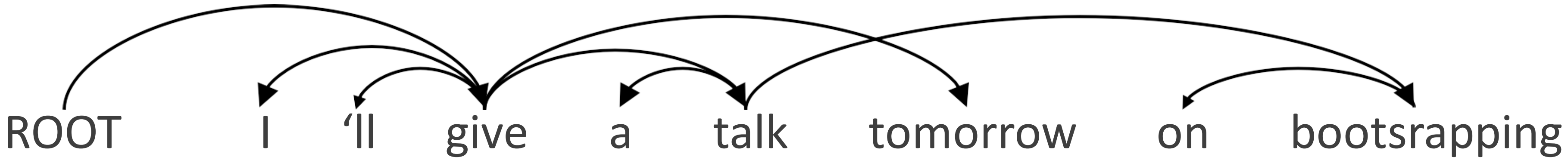


Dependency Conditioning Preferences

- What are the sources of information for dependency parsing?
  - Bilexical affinities: [discussion → issues] is plausible
  - Dependency distance: Mostly with nearby words
  - Valency of heads: How many dependents on which side are usual for a head?

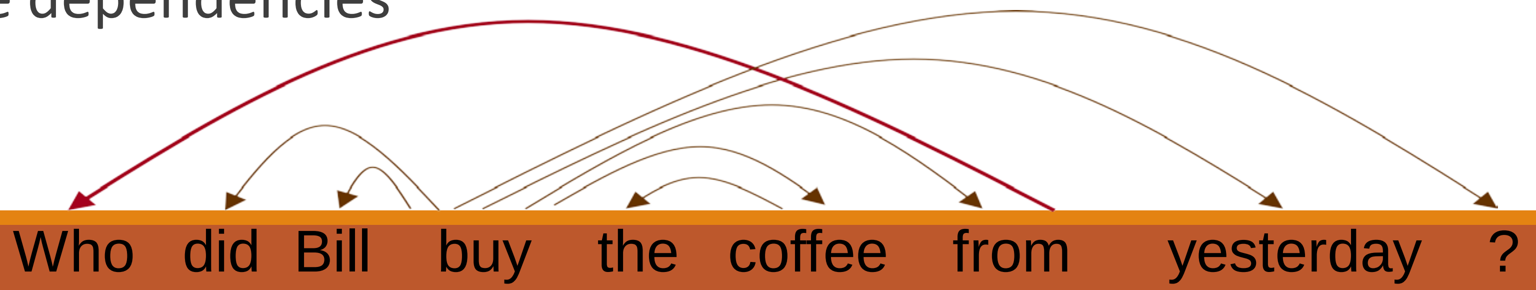
Dependency Parsing

- A sentence is parsed by choosing for each word what other word (including ROOT) is it a dependent of
- 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



Projectivity

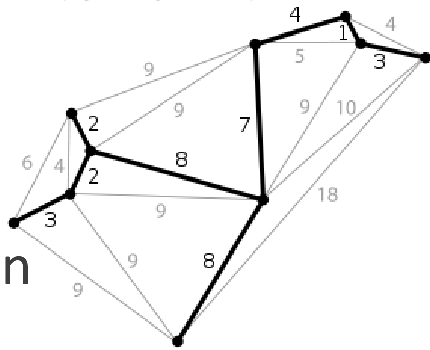
- Defn: There are no crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words
- Dependencies parallel to a CFG tree must be **projective**
  - Forming dependencies by taking 1 child of each category as head
- But dependency theory normally does allow non-projective structures to account for displaced constituents
  - You can't easily get the semantics of certain constructions right without these non-projective dependencies



Methods of Dependency Parsing

a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight

- 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



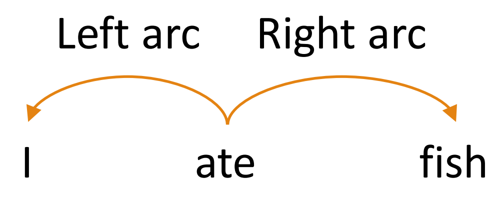
- Graph algorithms
  - You create a Minimum Spanning Tree for a sentence
  - McDonald et al.'s (2005) MSTParser scores dependencies independently using an ML classifier (he uses MIRA, for online learning, but it can be something else)
- Constraint Satisfaction
  - Edges are eliminated that don't satisfy hard constraints. Karlsson (1990), etc.
- "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.

2. 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: $\sigma = [\text{ROOT}]$, $\beta = w_1, \ldots, w_n$, $A = \emptyset$
- Shift $\quad \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
- Left-Arc$_r$ $\quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j,w_i)\}$
- Right-Arc$_r$ $\quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i,w_j)\}$
- Finish: $\sigma = [w]$, $\beta = \emptyset$



Arc-standard transition-based parser

- Analysis of “I ate fish”

Start

[root] | I ate fish

Shift

[root] I | ate fish

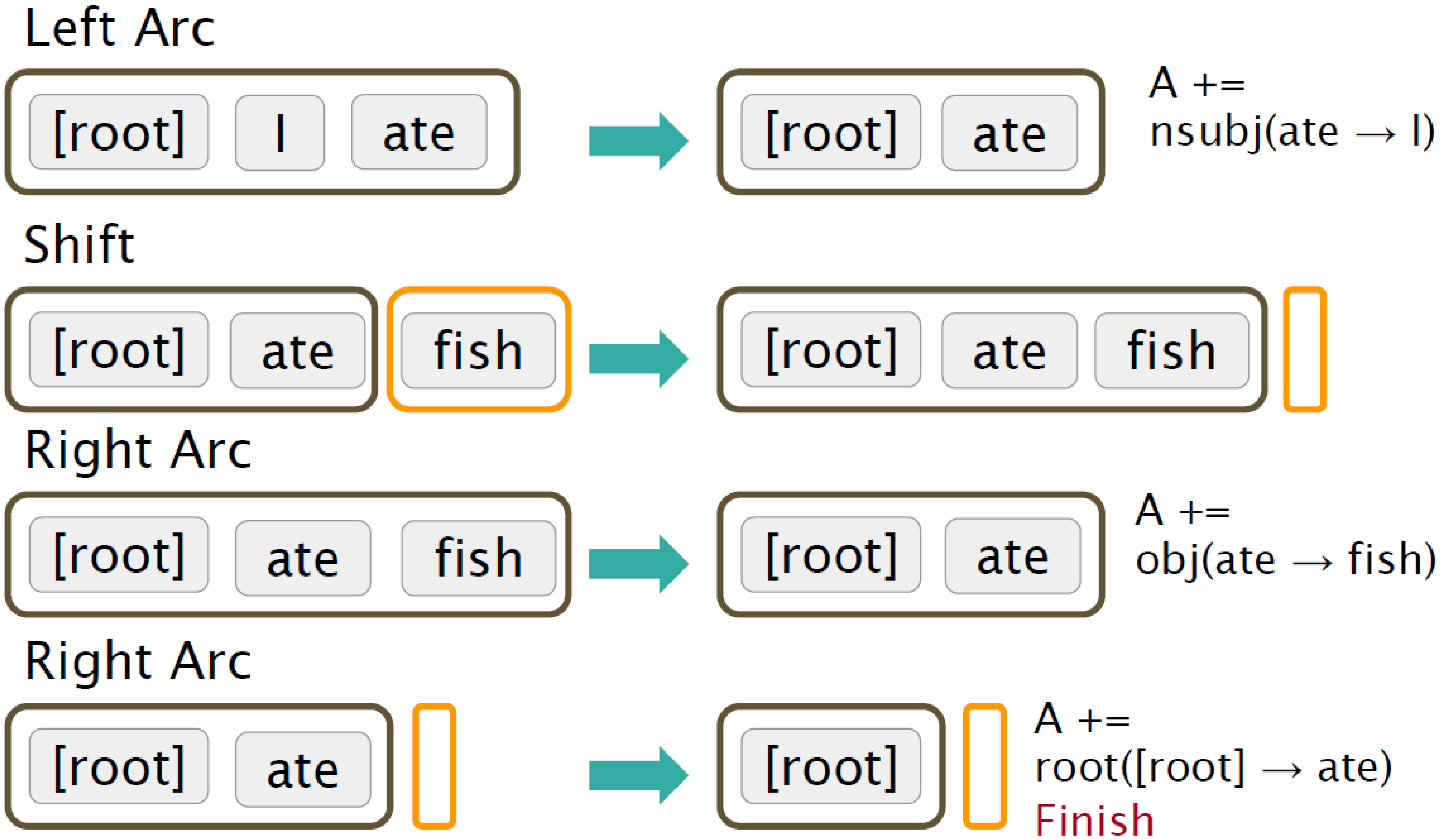
Shift

[root] I ate | fish

- Start: σ = [ROOT], $\beta = w_1, \ldots, w_n$, A = $\emptyset$
- Shift $\quad \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
- Left-Arc$_r$ $\quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j,w_i)\}$
- Right-Arc$_r$ $\quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i,w_j)\}$
- Finish: σ = [w], $\beta = \emptyset$

Arc-standard transition-based parser

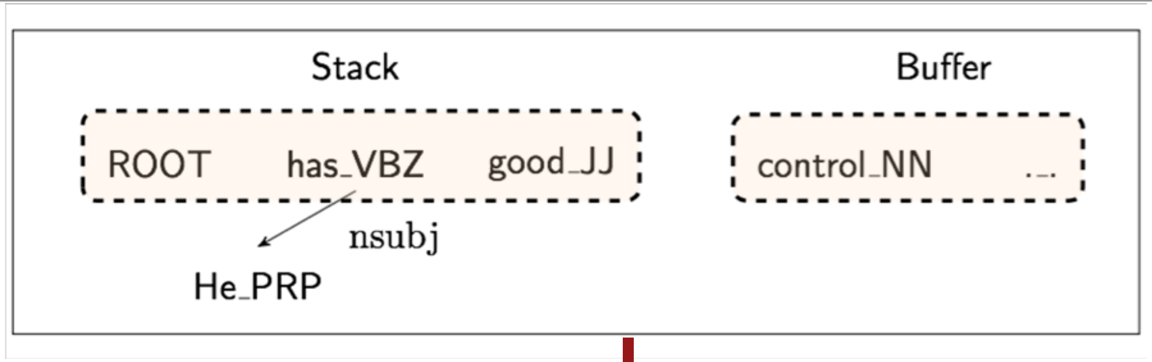
- Analysis of "I ate fish"



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 great performance

Conventional 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.

- JJ: Adjective
- NN: Noun
- PRP: Personal pronoun
- VBZ: Verb, 3rd person singular present

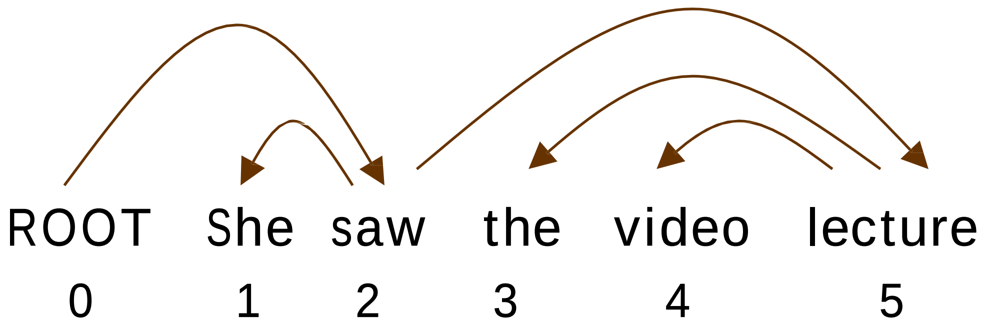
See https://cs.nyu.edu/grishman/jet/guide/PennPOS.html

Indicator features

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

lc: leftmost child

Evaluation of Dependency Parsing: (labeled) dependency accuracy



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

UAS = 4 / 5 = 80%
LAS = 2 / 5 = 40%

UAS: unlabeled attachment score
LAS: labeled attachment score

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 presented only builds projective dependency trees
- Possible directions to head:
  - Just declare defeat on non-projective arcs
  - Use dependency formalism which only has projective representations
    - CFG only allows projective structures
  - Use a postprocessor to a projective dependency parsing algorithm to identify and resolve non-projective links
  - Add extra transitions that can model at least most non-projective structures
    - e.g., add an extra SWAP transition, cf. bubble sort
  - Move to a parsing mechanism that does not use or require any constraints on projectivity
    - e.g., the graph-based MSTParser

3. Why train a neural dependency parser? Indicator Features Revisited

Problem #1: sparse
Problem #2: incomplete
Problem #3: expensive computation

Dense dim
=~1,000

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

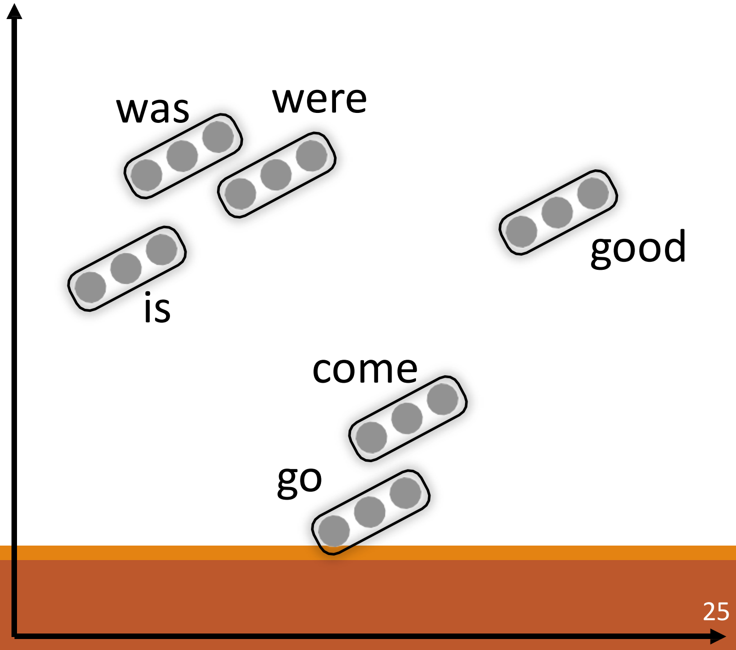
Parser	UAS	LAS	sent./s
MaltParser	89.8	87.2	469
MSTParser	91.4	88.1	10
TurboParser	**92.3**	89.6	8
C & M 2014	92.0	**89.7**	**654**

Distributed Representations

- Represent each word as a d-dimensional dense vector (i.e., word embedding)
  - Similar words are expected to have close vectors.

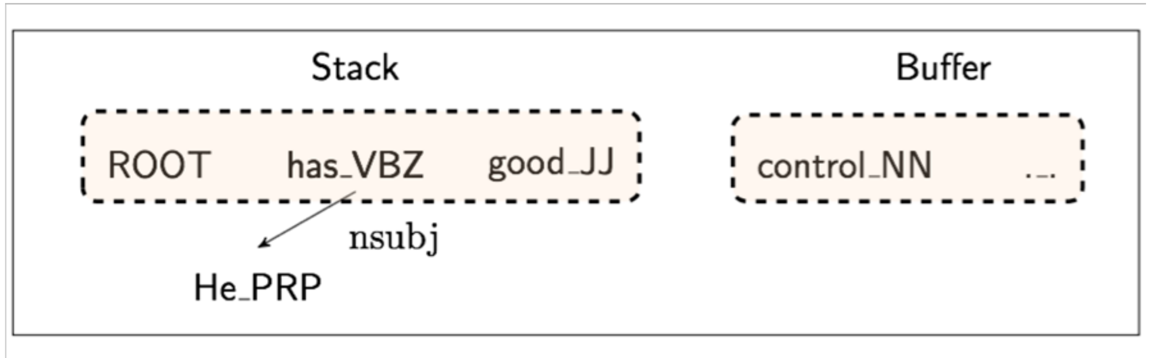
- Meanwhile, **part-of-speech tags** (POS) and **dependency labels** are also represented as d-dimensional vectors.
  - The smaller discrete sets also exhibit many semantical similarities.

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:



- We convert them to vector embeddings and concatenate them

	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
$rc(s_2)$	∅	∅	∅

Model Architecture

Softmax probabilities

Output layer y

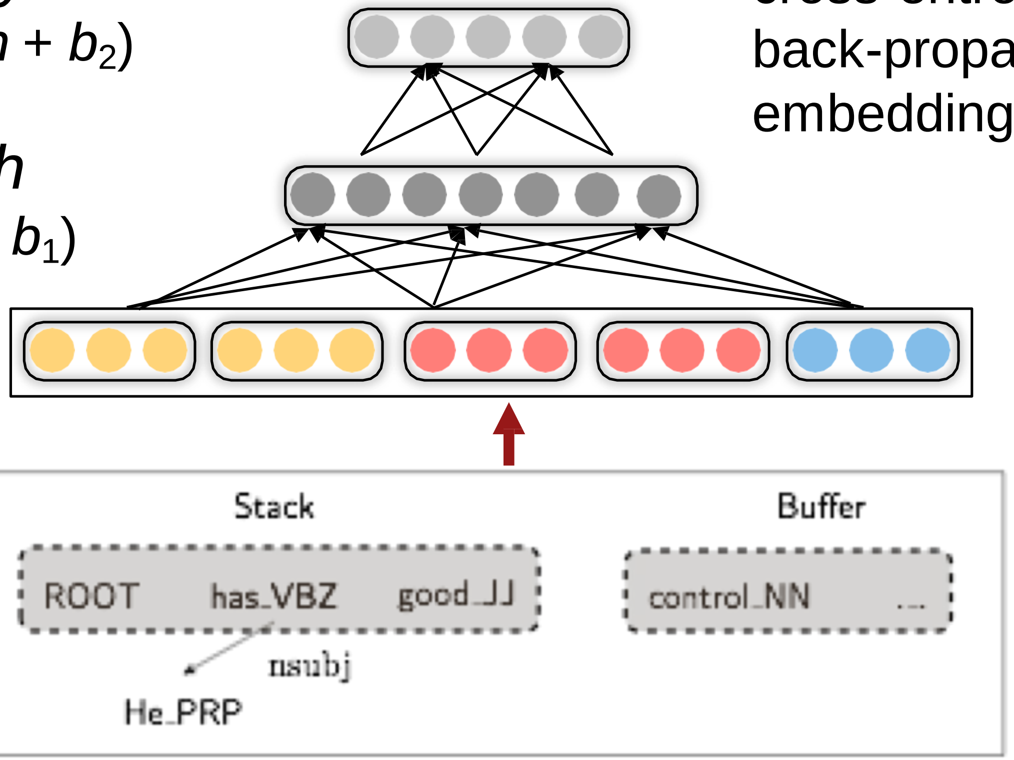
$y = \text{softmax}(Uh + b_2)$

Hidden layer h

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

Input layer x

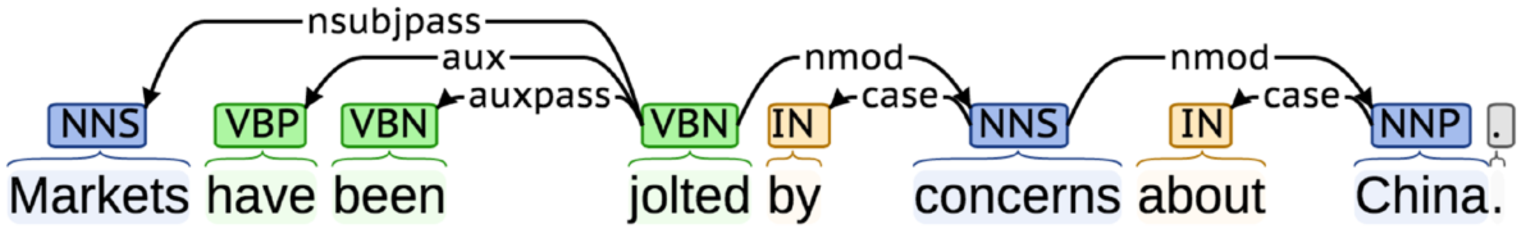
lookup + concat



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

Dependency parsing for sentence structure

- Neural networks can accurately determine the structure of sentences, supporting interpretation



- Chen and Manning (2014) was the first simple, successful neural dependency parser
- The dense representations let it outperform other greedy parsers in both accuracy and speed

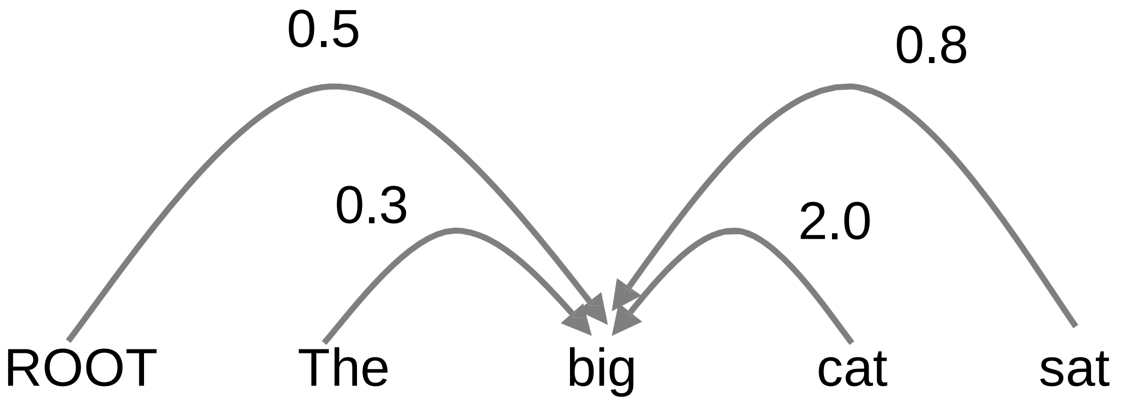
Further developments in transition-based neural dependency parsing

- This work was further developed and improved by others, including in particular at Google
  - Bigger, deeper networks with better tuned hyper-parameters
  - Beam search
  - Global, conditional random field (CRF)-style inference over the decision sequence
- Leading to SyntaxNet and the Parsey McParseFace model
  - 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 edge
  - Then add an edge from each word to its highest-scoring candidate head
  - And repeat the same process for each other word



e.g., picking the head for “big”

A Neural graph-based dependency parser

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

- Revived graph-based dependency parsing in a neural world
  - Design a bi-affine attention model for neural dependency parsing
    - score each possible head for each dependent
  - Really great results! 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**