

Dependency Parsing

Bonan Min

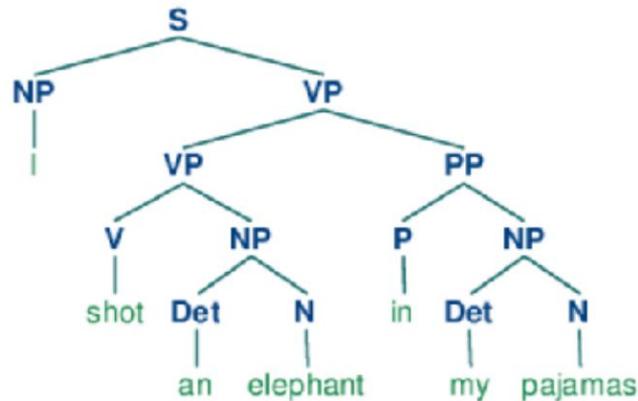
bonanmin@gmail.com

Some slides are based on class materials from Thien Huu Nguyen, Ralph Grishman, David Bamman, Dan Jurasky, Chris Manning and others

Dependency Syntax

“Between the word and its neighbors, the mind perceives connections, the totality of which forms the structure of the sentence. The structural connections establish dependency relations between the words. Each connection in principle unites a superior and an inferior term.”

Dependency Syntax



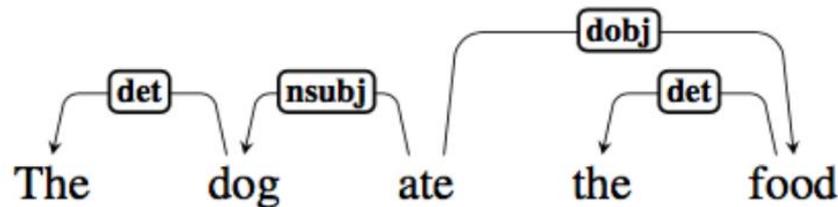
Dependency syntax doesn't have non-terminal structure like a CFG; words are directly linked to each other.

Syntactic structure = asymmetric, binary relations between words.

Dependency Structures/Trees

A dependency structure is a directed graph $G = (V, A)$ consisting of a set of vertices V and arcs A between them. Typically constrained to form a tree:

- Each vertex corresponds to a word in the sentence
- Single root vertex with no incoming arcs
- Every vertex has exactly one incoming arc except root (single head constraint), defining the **parent/children** or **governor/dependent** or **head/tail** relations
- Each arc is associated with one label to indicate the dependency relation between the two ends (e.g., nsubj, dobj, det) (i.e., typed arcs)
- There is a unique path from the root to each vertex in V (acyclic constraint)



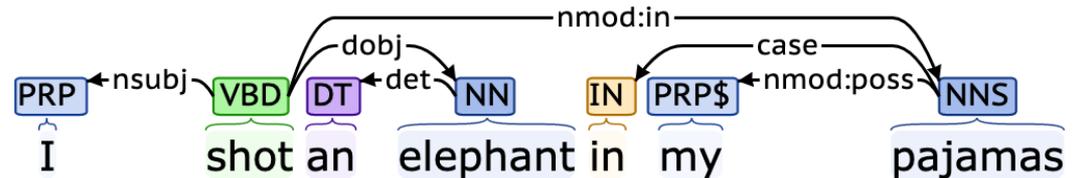
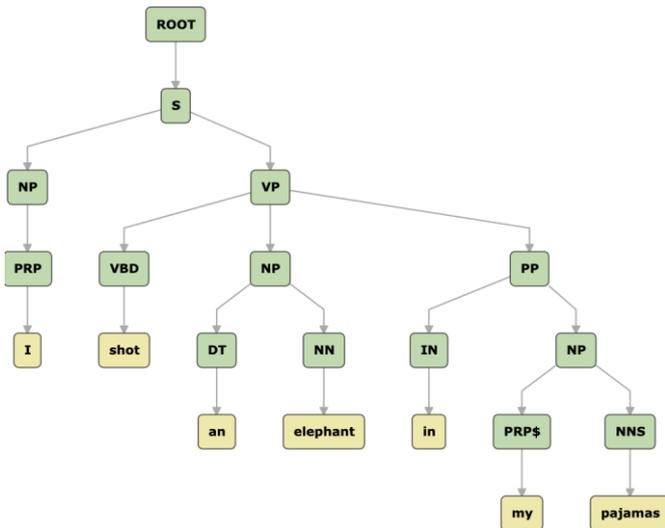
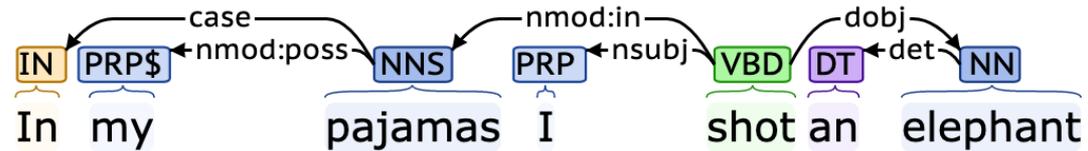
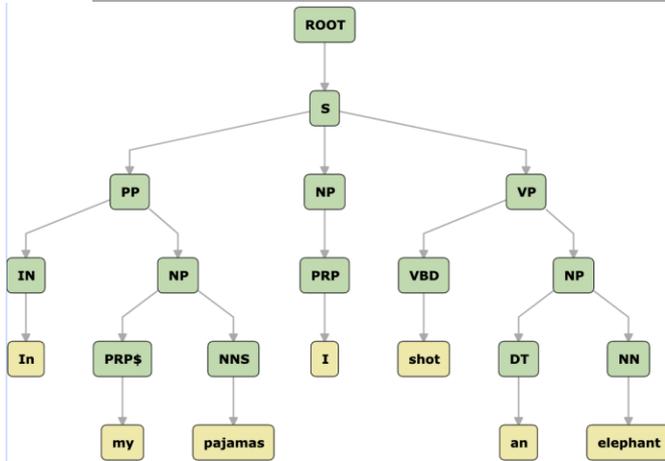
Dependency Trees

Unlike phrase-structure trees, dependency trees aren't tied to the linear order of the words in a sentence.

Dependency relations belong to the structural order of a sentence, **not the linear order**.

- This is different from a phrase-structure tree, where the syntax is constrained by the linear order of the sentence (a different linear order yields a different parse tree).

Dependency Trees



From: <http://corenlp.run/> (Stanford CoreNLP)

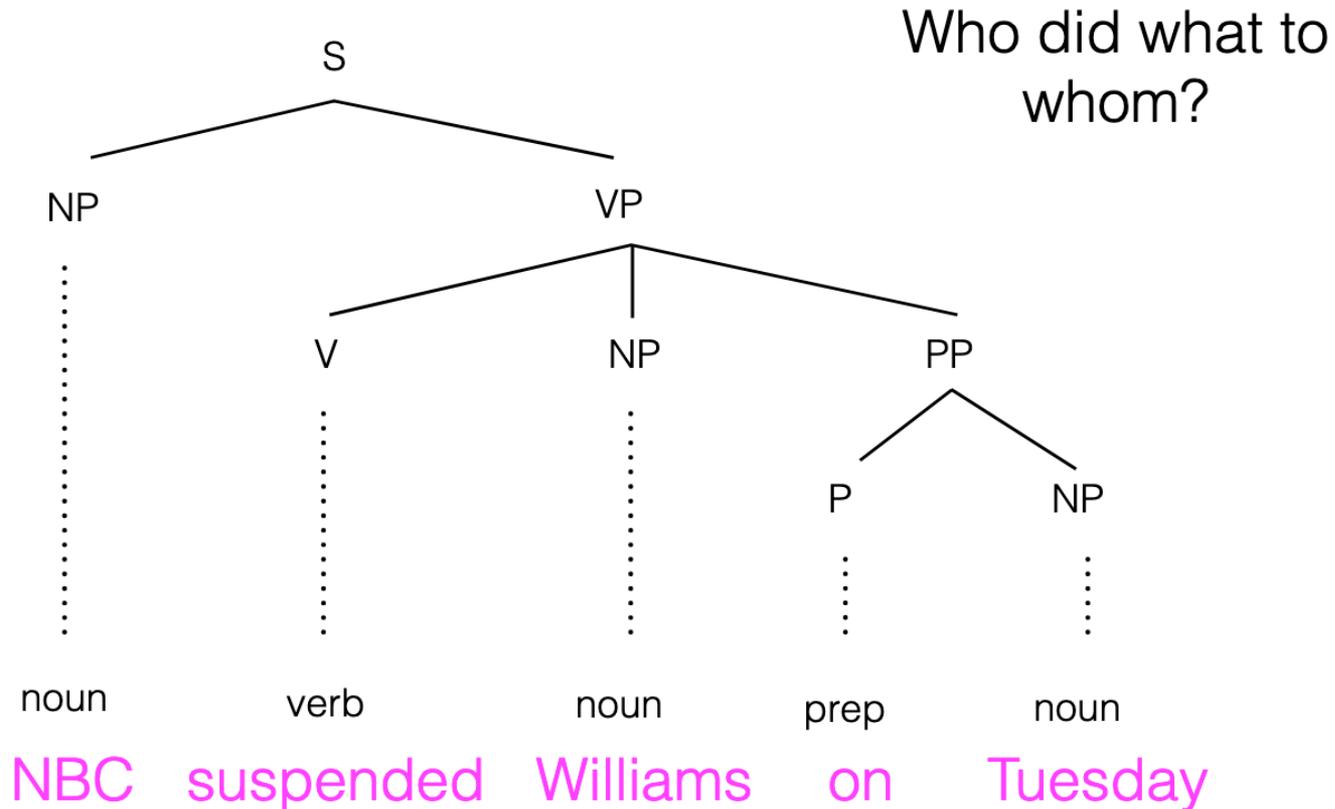
Dependencies Vs Constituents

Dependency links are closer to semantic relationships; no need to infer the relationships from the structure of a tree

A dependency tree contains one edge for each word, no intermediate hidden structures that also need to be learned for parsing.

Easier to represent languages with free word order.

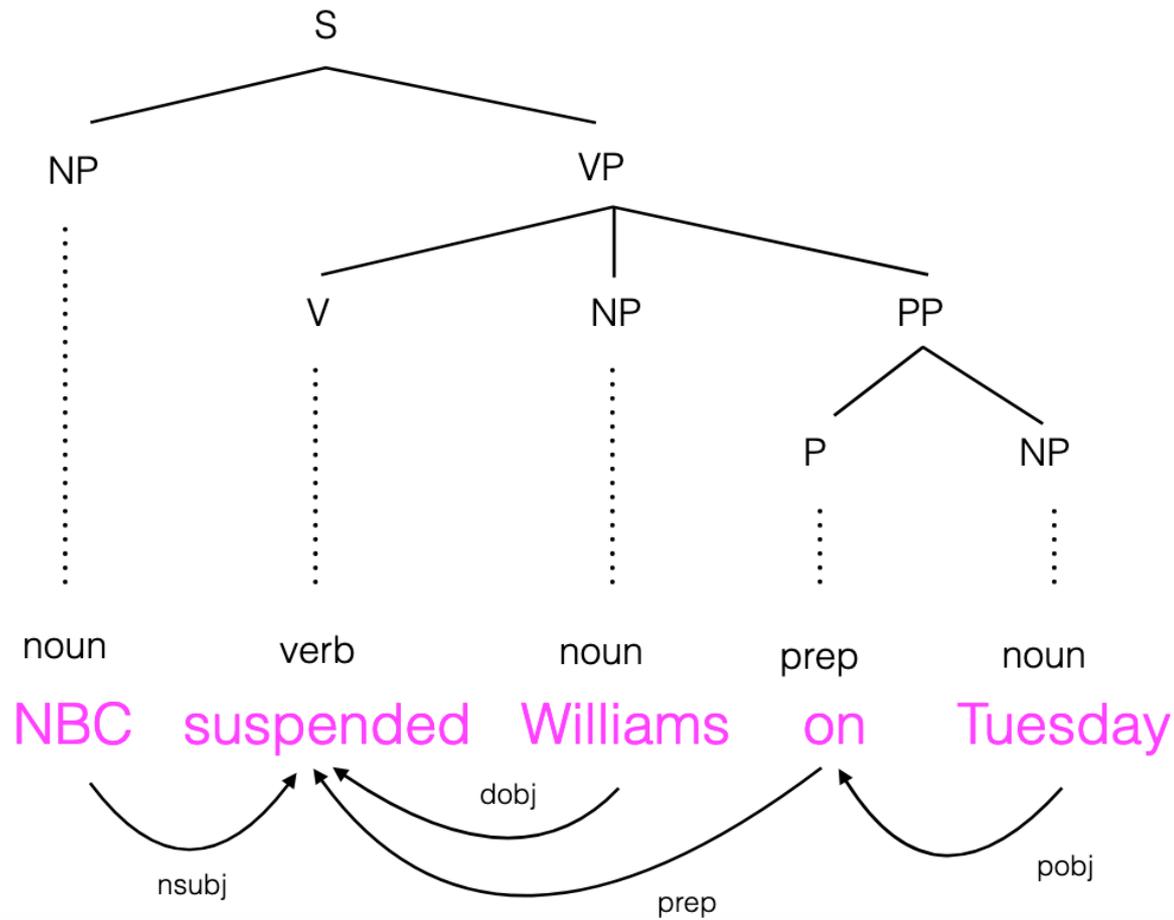
Dependencies vs Constituents



subject: S → NP VP

direct object: S → NP (VP → ... NP ...)

Dependencies vs Constituents

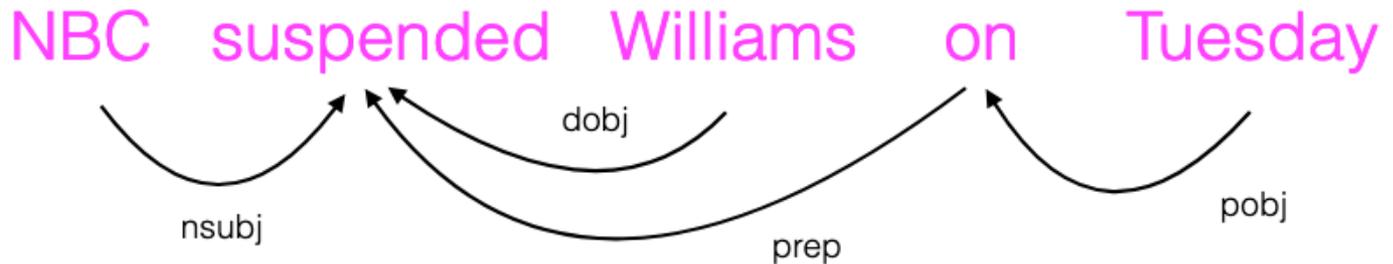


People can write arrows in this or the other way

Dependency Grammar

Captures binary relations between words

- nsubj(NBC, suspended)
- dobj(Williams, suspended)



Universal Dependencies

Developing cross-linguistically consistent treebank annotation for many languages

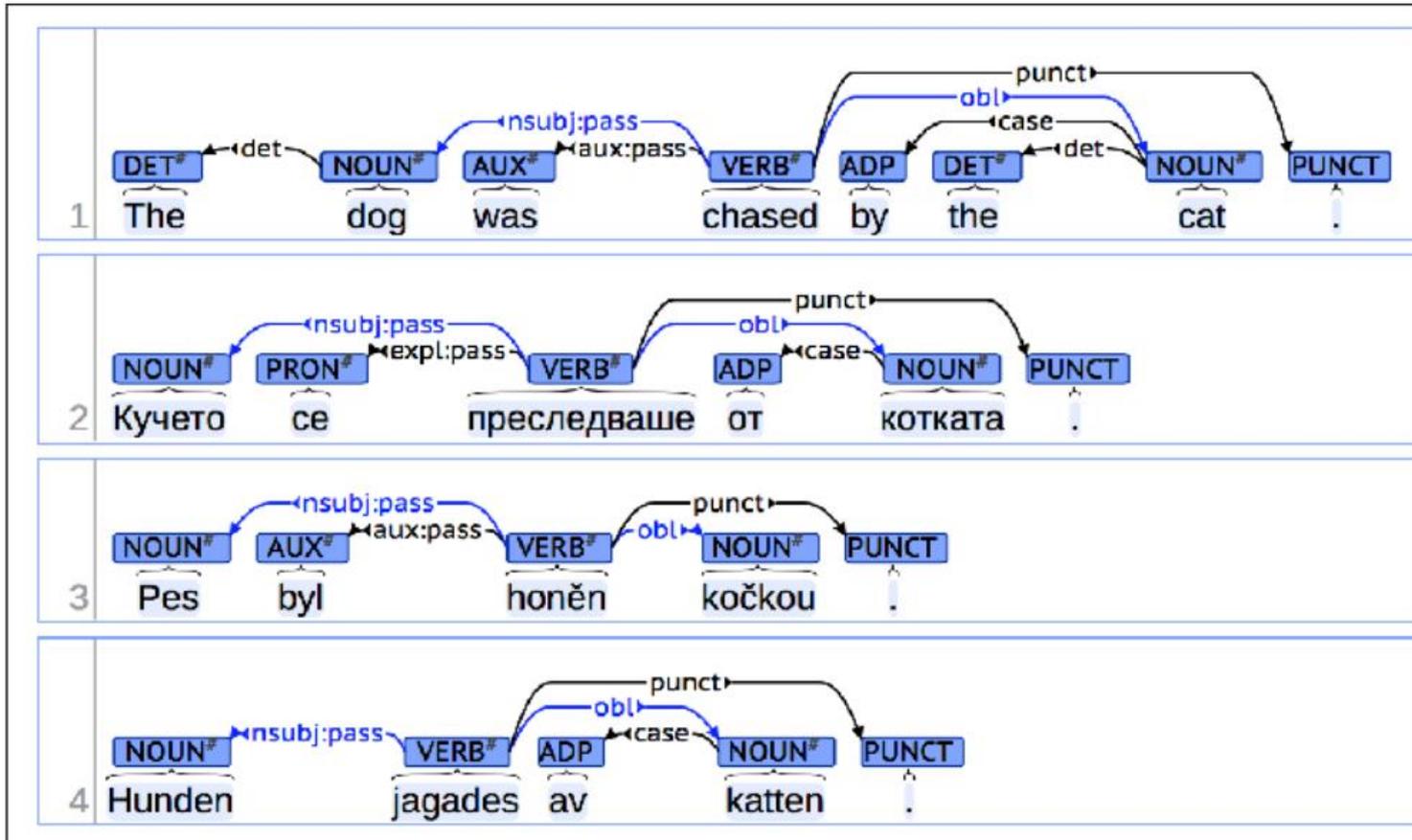
Goals:

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

	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		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		IE, Celtic
	Bulgarian	1	156K		IE, Slavic
	Buryat	1	10K		Mongolic
	Cantonese	1	13K		Sino-Tibetan
	Catalan	1	531K		IE, Romance
	Chinese	5	161K		Sino-Tibetan
	Classical Chinese	1	55K		Sino-Tibetan
	Coptic	1	25K		Afro-Asiatic, Egyptian
	Croatian	1	199K		IE, Slavic
	Czech	5	2,222K		IE, Slavic
	Danish	2	100K		IE, Germanic
	Dutch	2	307K		IE, Germanic
	English	6	603K		IE, Germanic
	Erzya	1	15K		Uralic, Mordvin
	Estonian	2	461K		Uralic, Finnic
	Faroese	1	10K		IE, Germanic
	Finnish	3	377K		Uralic, Finnic
	French	8	1,156K		IE, Romance
	Galician	2	164K		IE, Romance
	German	4	3,409K		IE, Germanic
	Gothic	1	55K		IE, Germanic

<http://universaldependencies.org>

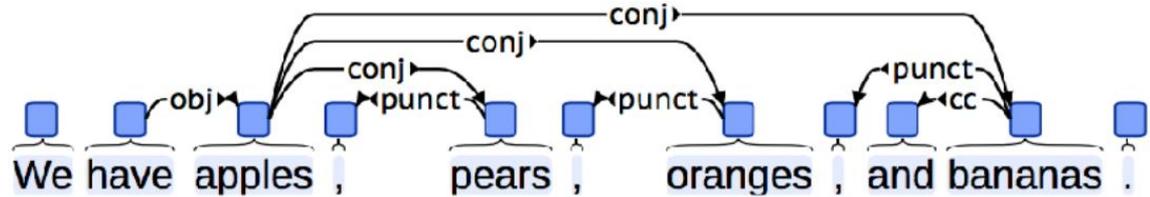
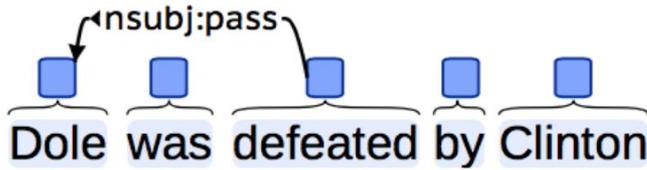
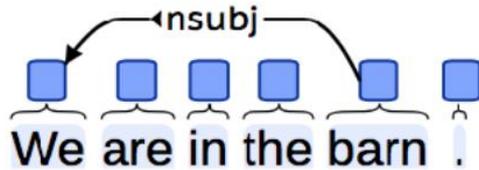
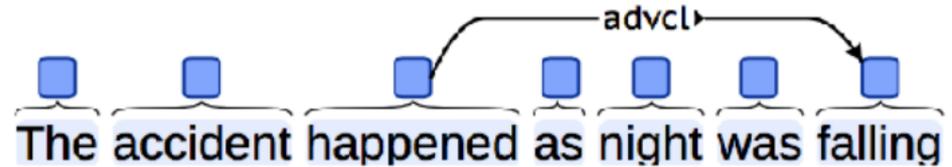
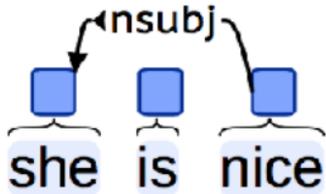
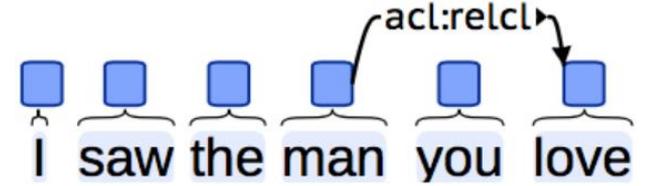
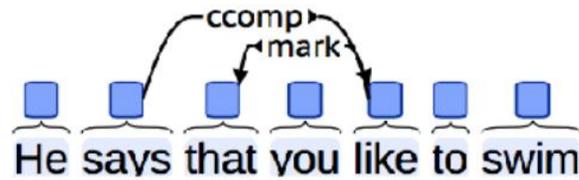
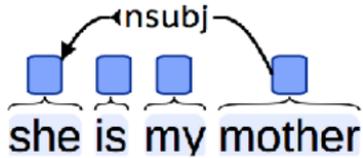
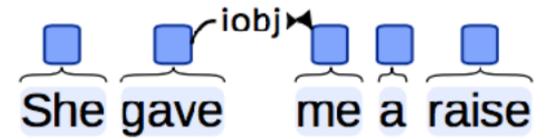
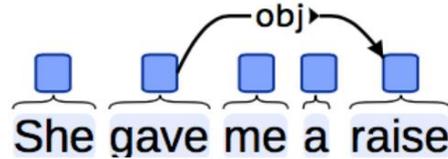
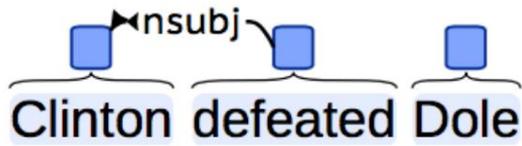
Universal Dependencies



Dependency Grammar

nsubj, nsubj:pass	subject of a verb
obj	direct object of a verb
iobj	indirect object of a verb
obl	oblique modifier of a verb
ccomp	clausal complement
advcl	adverbial clause modifier
acl, acl:rel	clausal modifier of noun
conj	conjunct in coordination

Some Examples

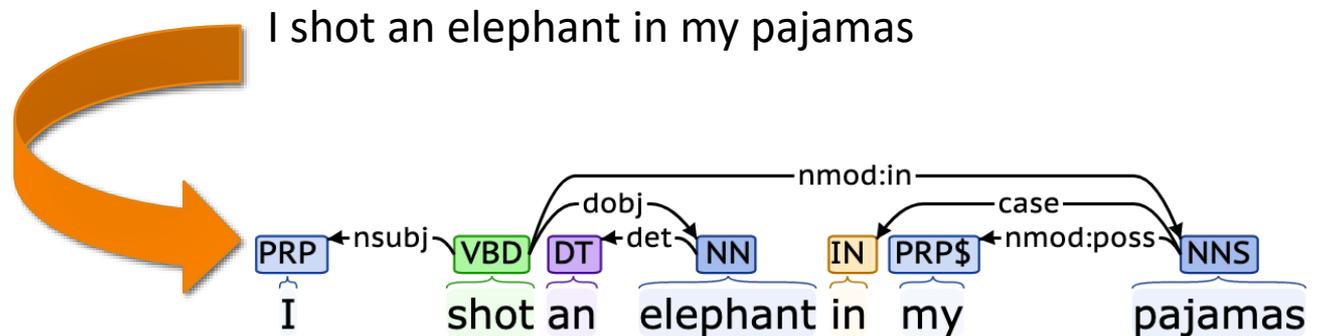


Dependency Parsing

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 $A \rightarrow B, B \rightarrow A$



Methods For Dependency Parsing

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. 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 if don't satisfy hard constraints (Karlsson 1990)

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

- Greedy choice of attachments guided by good machine learning classifiers (i.e., MaltParser (Nivre et al. 2008)). **Has proven highly effective.**

Greedy Transition-based Dependency Parsing

The parser starts in **an initial configuration**.

At each step, it asks **a guide** to choose between one of several **transitions (actions)** into new configurations.

Parsing stops if the parser reaches **a terminal configuration**.

The parser returns the dependency tree associated with the terminal configuration.

Greedy Transition-based Dependency Parsing

Eisner's algorithm runs in time $O(n^3)$.
This may be too much if a lot of data is involved.

Idea: Design a dumber but really fast algorithm and let the machine learning do the rest.

Eisner's algorithm searches over many different dependency trees at the same time.

A transition-based dependency parser **only builds one tree**, in **one left-to-right sweep over the input**.

Generic Parsing Algorithm

```
Configuration c = parser.getInitialConfiguration(sentence)

while c is not a terminal configuration do

    Transition t = guide.getNextTransition(c)

    c = c.makeTransition(t)

return c.getGraph()
```

Transition-based dependency parsers differ with respect to the configurations and the transitions that they use.

The Arc-standard Algorithm

The arc-standard algorithm is a simple algorithm for transition-based dependency parsing.

It is very similar to shift–reduce parsing as it is known for context-free grammars.

It is implemented in most practical transition-based dependency parsers, including MaltParser.

Configurations

A **configuration** for a sentence $w = w_1, \dots, w_n$ consists of three components:

- A **buffer** containing words of w
- A **stack** containing words of w
- The **dependency tree** constructed so far

Configurations

Initial configuration:

- All words are in the buffer
- The stack is empty
- The dependency tree is empty

Terminal configuration:

- The buffer is empty
- The stack contains a single word

Possible Transitions

`shift(sh)`:

push the next word in the buffer onto the stack

`left-arc(la)`:

add an arc from the topmost word on the stack, s_1 , to the second-topmost word, s_2 , and pop s_2

`right-arc(ra)`:

add an arc from the second-topmost word on the stack, s_2 , to the topmost word, s_1 , and pop s_1

Configurations And Transitions

Initial configuration: $([], [0, \dots, n], [])$

Terminal configuration: $([0], [], A)$

shift (sh):

$$(\sigma, [i|\beta], A) \Rightarrow ([\sigma|i], \beta, A)$$

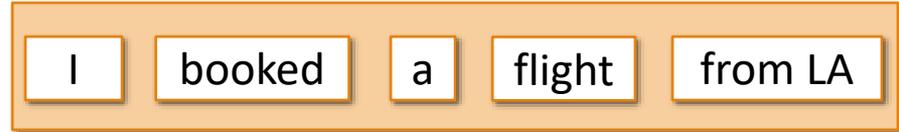
left-arc (la):

$$([\sigma|i|j], B, A) \Rightarrow ([\sigma|j], B, A \cup \{j, l, i\}) \text{ only if } i \neq 0$$

right-arc (ra):

$$([\sigma|i|j], B, A) \Rightarrow ([\sigma|i], B, A \cup \{i, l, j\})$$

Example



I

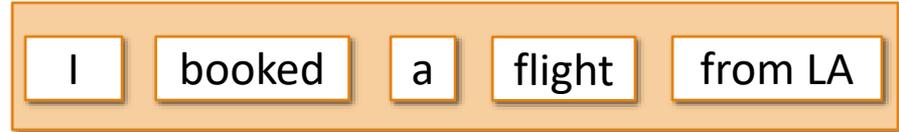
booked

a

flight

from LA

Example



I

booked

a

flight

from LA

A pink speech bubble with a thin orange border and a tail pointing towards the bottom left. Inside the bubble, the text "sh" is written in white.

Example

I

booked a flight from LA

I

booked

a

flight

from LA

Example

I

booked a flight from LA

I

booked

a

flight

from LA

sh

Example

I

booked

a

flight

from LA

I

booked

a

flight

from LA

Example

I booked

a flight from LA

I

booked

a

flight

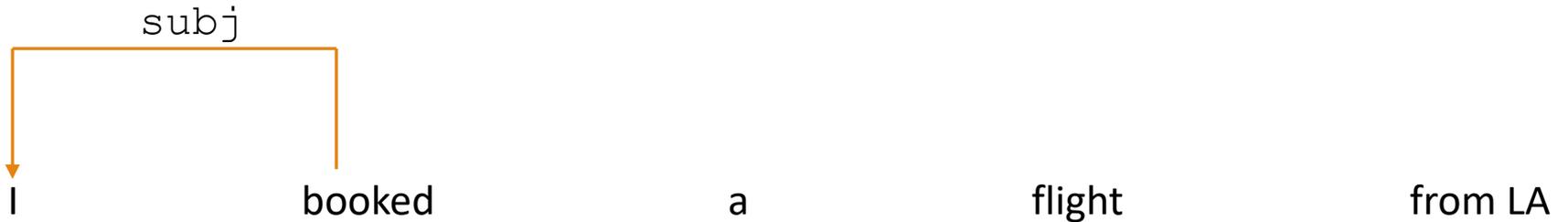
from LA

la-subj

Example

booked

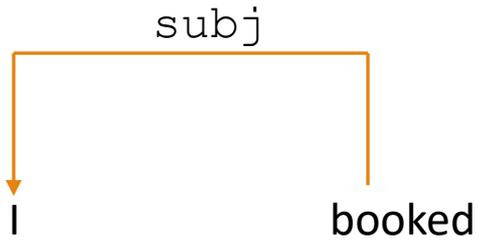
a flight from LA



Example

booked

a flight from LA



a

flight

from LA

sh

Example

booked

a

flight

from LA

subj



a

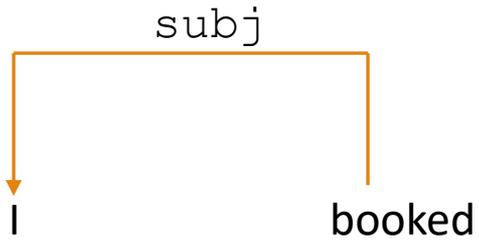
flight

from LA

Example

booked a

flight from LA



a

flight

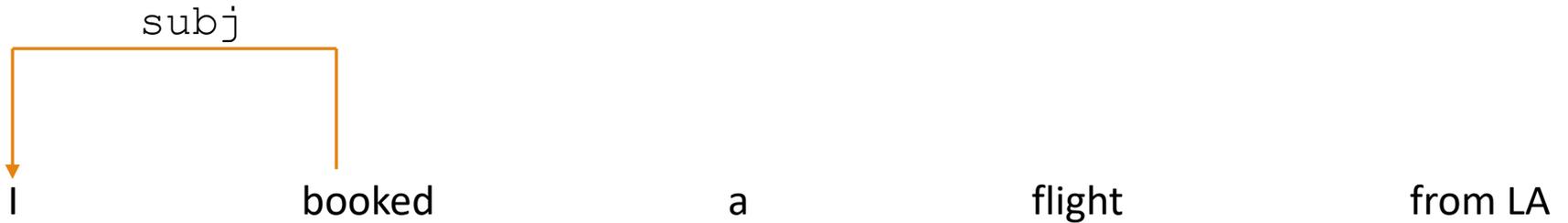
from LA

sh

Example

booked a flight

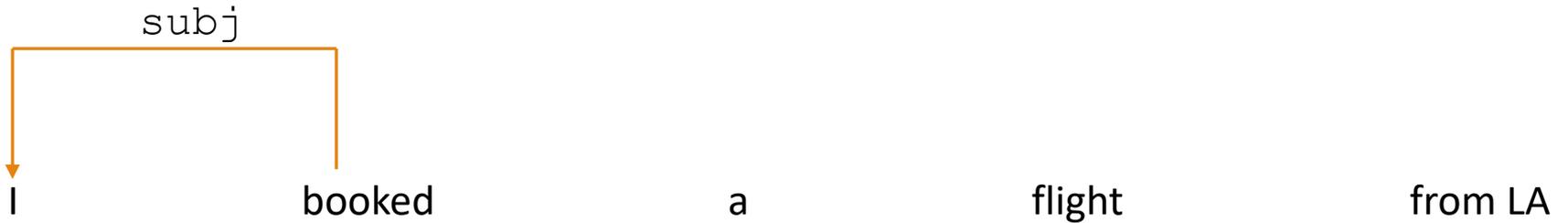
from LA



Example

booked a flight

from LA

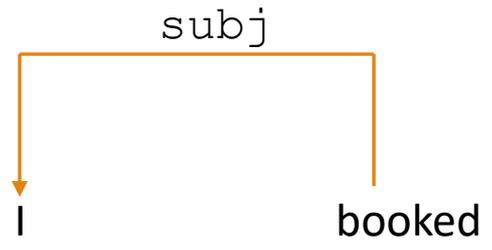


la-det

Example

booked flight

from LA

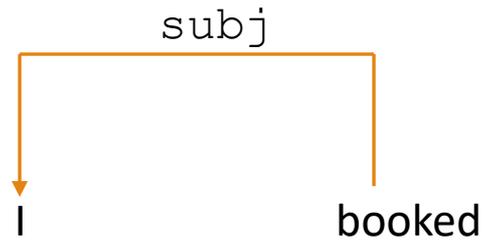


from LA

Example

booked flight

from LA



from LA

sh

Example

booked

flight

from LA

subj



det



from LA

Example

booked

flight

from LA

subj



det

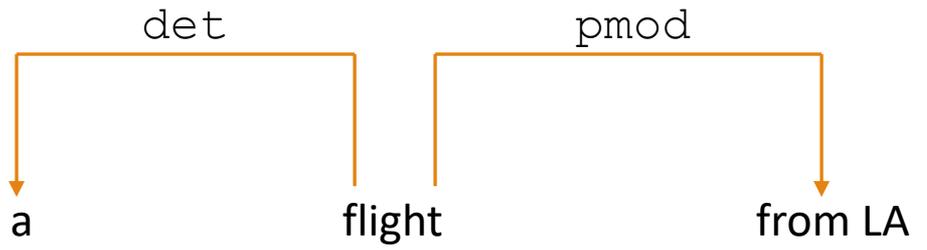
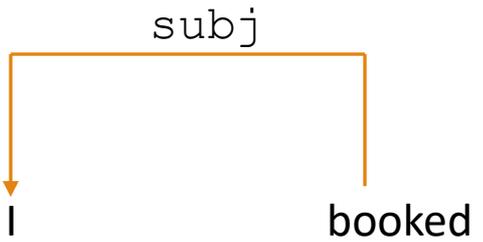


from LA

ra-pmod

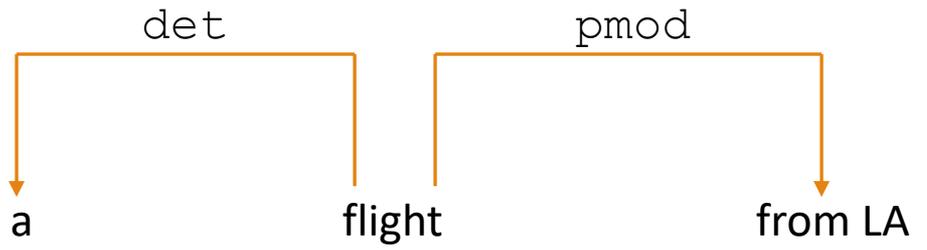
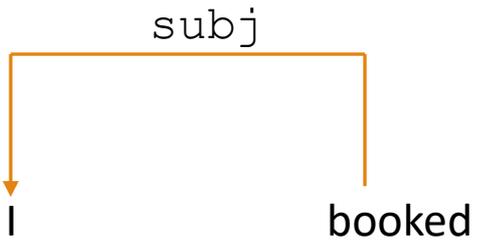
Example

booked flight



Example

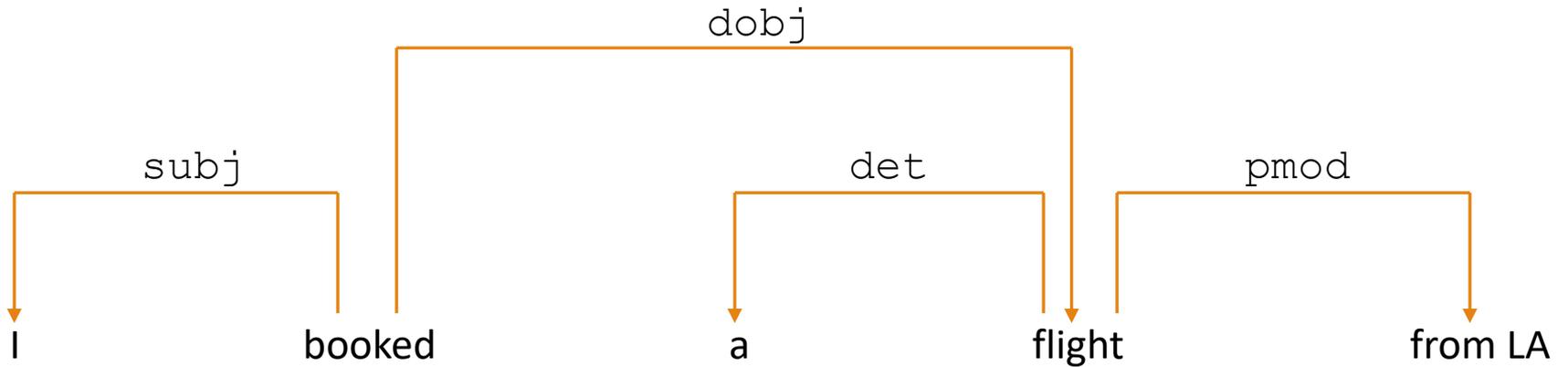
booked flight



ra-dobj

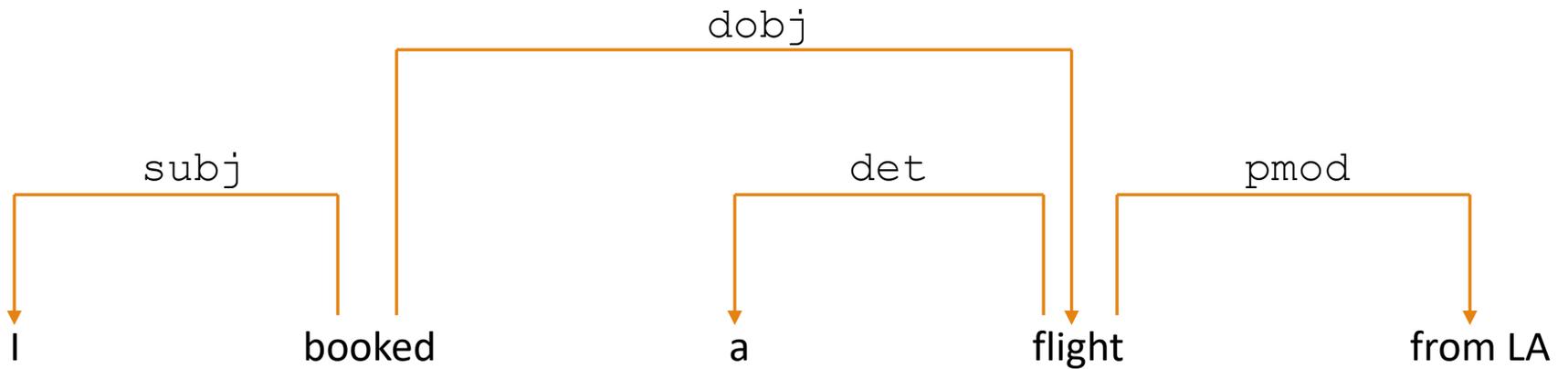
Example

booked



Example

booked



done!

Complexity And Optimality

Time complexity is linear, $O(n)$, since we only have to treat each word once

This can be achieved since the algorithm is greedy, and only builds one tree, in contrast to Eisner's algorithm, where all trees are explored

There is no guarantee that we will even find the best tree given the model, with the arc-standard model

There is a risk of error propagation

An advantage is that we can use very informative features, for the ML algorithm

Guides

We need a guide that tells us what the next transition should be.

The task of the guide can be understood as **classification**:
Predict the next transition (class), given the current configuration.

Training A Guide

We let the parser run on gold-standard trees.

We collect all (**configuration**, **transition**) pairs and train a classifier on them.

When parsing unseen sentences, we use the trained classifier as a guide.

The number of (configuration, transition) pairs is far too large.

We define a set of features of configurations that we consider to be relevant for the task of predicting the next transition.

- *Example:* word forms of the topmost two words on the stack and the next two words in the buffer

We can then describe every configuration in terms of a feature vector (**feature engineering**).

Training A Guide

In practical systems, we have thousands of features and hundreds of transitions.

There are several machine-learning paradigms that can be used to train a guide for such a task:

- SVM, Logistic Regression, Deep Neural Networks

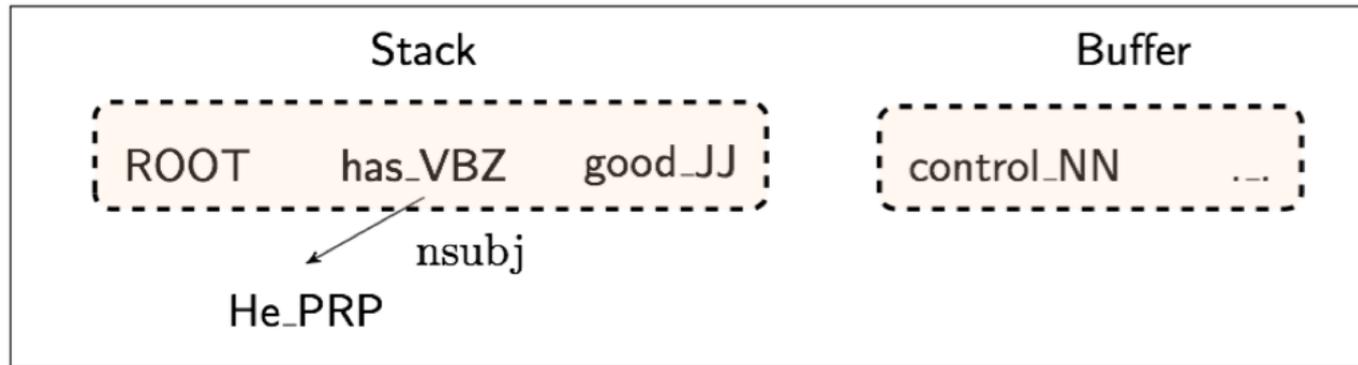
Example Features

	Attributes				
Adress	FORM	LEMMA	POS	FEATS	DEPREL
Stack[0]	X	X	X	X	
Stack[1]			X		
Ldep(Stack[0])					X
Rdep(Stack[0])					X
Buffer[0]	X	X	X	X	
Buffer[1]			X		
Ldep(Buffer[0])					X
Ldep(Buffer[0])					X
...					

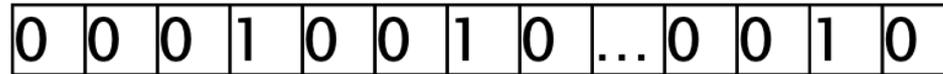
Combinations of addresses and attributes (e.g. those marked in the table)

Other features, such as distances, number of children, ...

Conventional Feature Representation



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



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

Indicator features

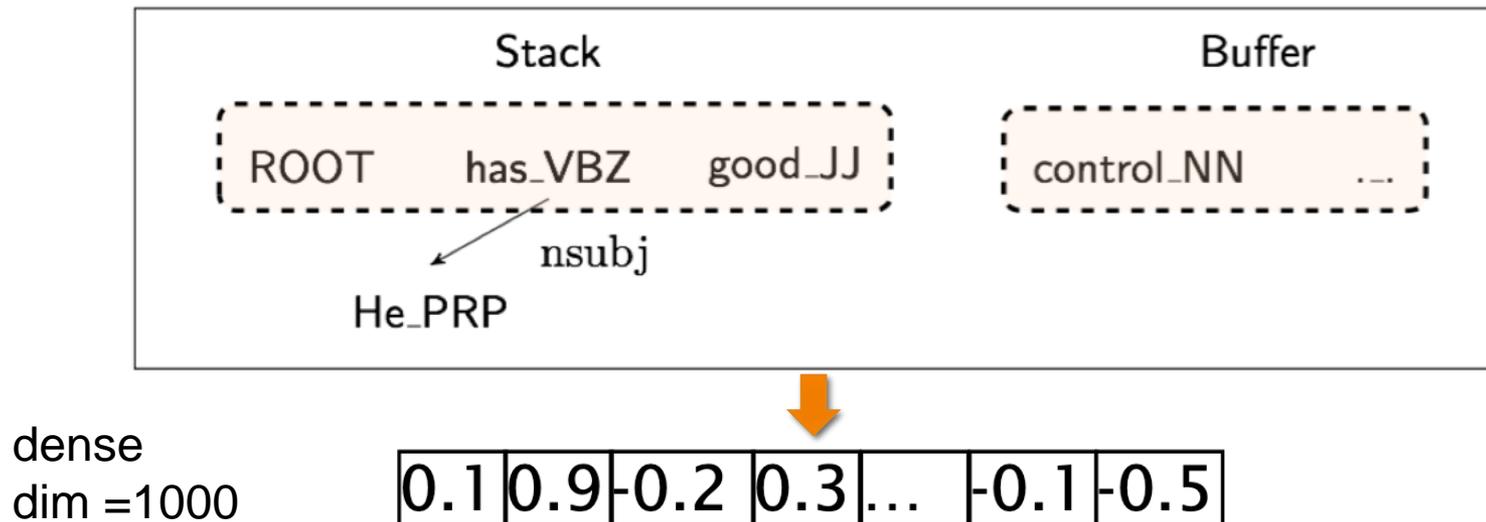
- $s1.w = \text{good} \wedge s1.t = \text{JJ}$
- $s2.w = \text{has} \wedge s2.t = \text{VBZ} \wedge s1.w = \text{good}$
- $lc(s_2).t = \text{PRP} \wedge s_2.t = \text{VBZ} \wedge s_1.t = \text{JJ}$
- $lc(s_2).w = \text{He} \wedge lc(s_2).l = \text{nsubj} \wedge s_2.w = \text{has}$

Problems With The Conventional Feature Representation

Sparse

Expensive (more than 95% of parsing time is consumed by feature computation)

So, use neural networks/deep learning to learn a dense and compact feature representation



A Neural Dependency Parser

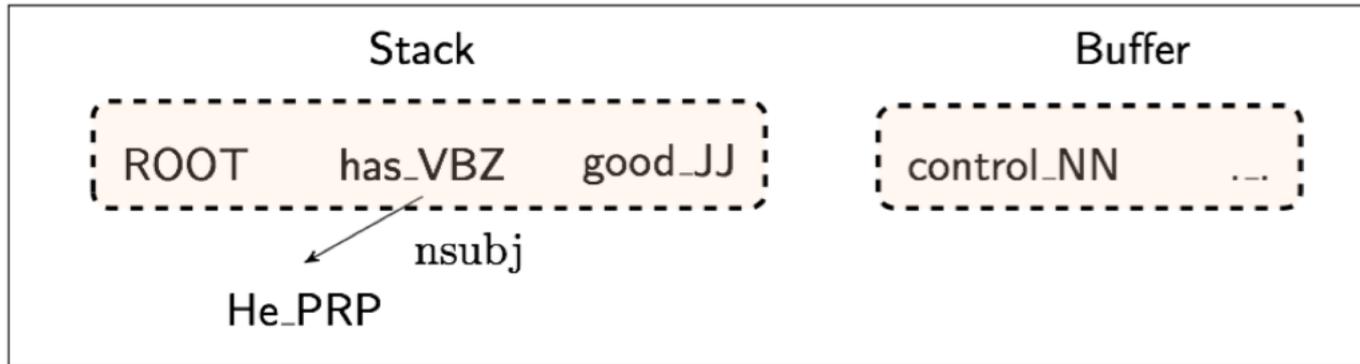
We represent each word as with its (dense) word embeddings.

Meanwhile, POS tags and dependency labels (relations) are also represented as d-dimensional dense vectors.

- The smaller discrete sets also exhibit many semantic similarities.
 - NNS (plural noun) should be close to NN (singular noun)
 - num (numerical modifier) should be close to amod (adjective modifier).

We extract the tokens for the configurations based on the stack/buffer positions and use their vectors to obtain the representation for the configurations

A Neural Dependency Parser

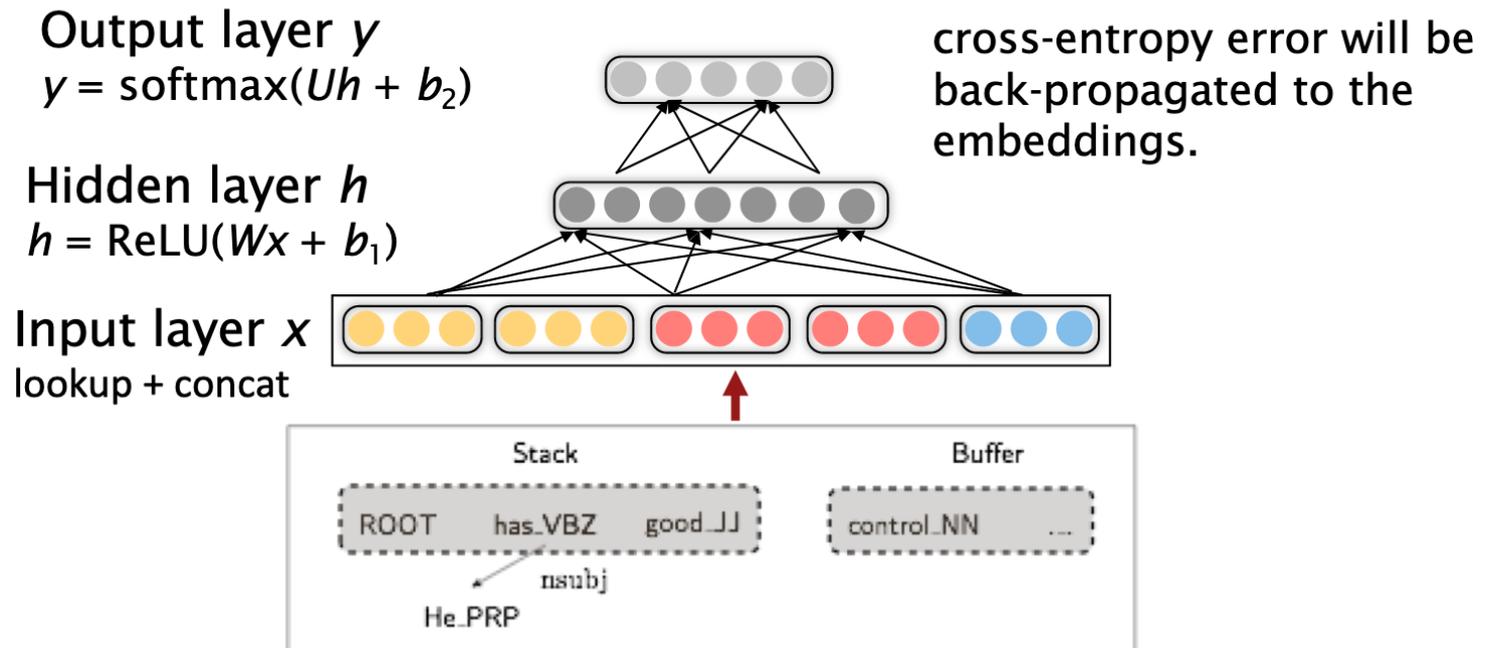


	word	POS	dep.
s1	good	JJ	∅
s2	has	VBZ	∅
b1	control	NN	∅
lc(s1)	∅	+ ∅	+ ∅
rc(s1)	∅	∅	∅
lc(s2)	He	PRP	nsubj
rc(s2)	∅	∅	∅

Extract a set of tokens for the configuration based on the positions on the stack and buffer, and then concatenate their representation vectors.

A Neural Dependency Parser

Softmax probabilities



Alternative Transition Models

There is another version of the arc-standard model, where arcs are added between the topmost word on the stack and the topmost word on the buffer

There are actually many other alternatives

Arc-eager model

- Contain four transitions:
 - Shift
 - Reduce
 - Left-arc
 - Right-arc
- Advantage: not strictly bottom-up, can create arcs earlier than in the arc-standard model

Evaluation Of Dependency Parsers

Labelled attachment score (LAS): percentage of correct arcs, relative to the gold standard

Labelled exact match (LEM): percentage of correct dependency trees, relative to the gold standard

Unlabeled attachment score/exact match (UAS/ UEM): the same, but ignoring arc labels

Word-vs sentence-level AS

Example: 2 sentence corpus

- sentence 1: 9/10 arcs correct
- sentence 2: 15/45 arcs correct

Word-level (micro-average):

- $(9+15) / (10+45) = 0.436$

Sentence-level (macro-average):

- $(9+10+15/45)/2 = 0.617$

Word-level evaluation is normally used

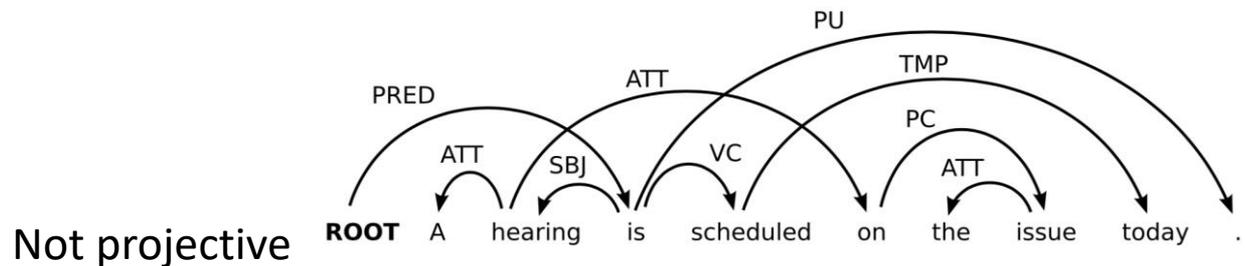
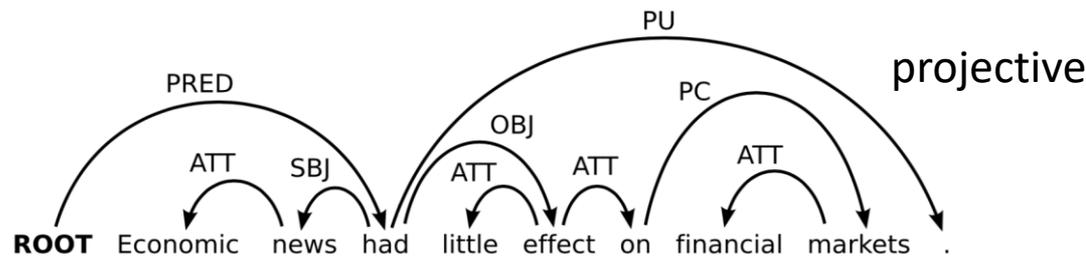
Evaluation Of Dependency Parsers

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

Projectivity

A dependency tree is projective if:

- For every arc in the tree, there is a directed path from the head of the arc to all words occurring between the head and the dependent (that is, the arc (i, l, j) implies that $i \rightarrow^* k$ for every k such that $\min(i, j) < k < \max(i, j)$).
- Or equivalently: There are no crossing dependency arcs when the words are laid out in their linear order, with all arcs above the words



Projectivity And Dependency Parsing

Many dependency parsing algorithms can only handle projective trees

- Including all algorithms we have discussed so far

Non-projective trees do occur in natural language

- How often depends on language (and treebank)

Non-projective Dependency Parsing

Variants of transition-based parsing

- Using a swap-transition to allow non-projective parsing
 - Contain four transitions: Shift, Swap, Left-arc, and Right-arc
 - Runtime is $O(n^2)$ in the worst case (but usually less in practice)
- Using more than one stack
- Pseudo-projective parsing

Graph-based parsing

- Minimum spanning tree algorithms