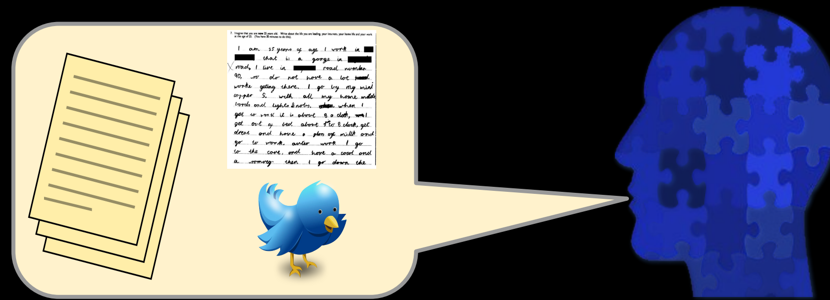
Syntactic Processing: Parts-of-Speech Tagging & Dependency Parsing

CSE354 - Spring 2021

Task

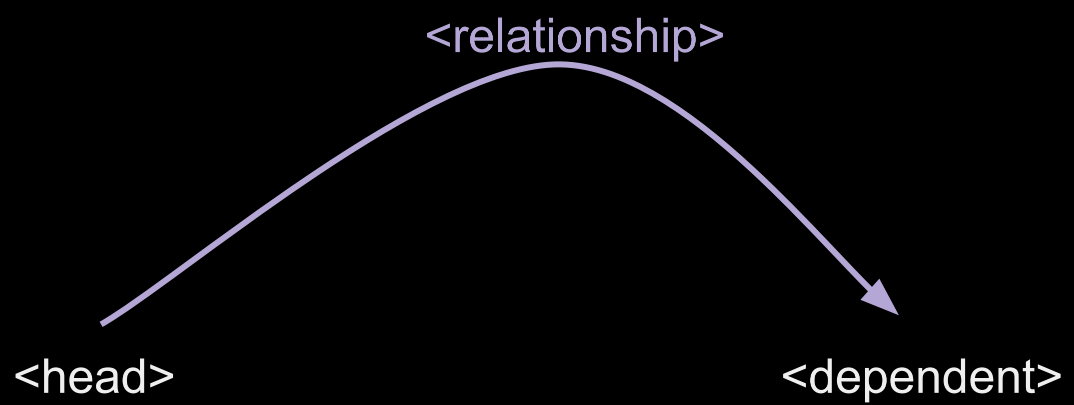


- Parts-of-Speech Tagging
- Dependency Parsing

how? →

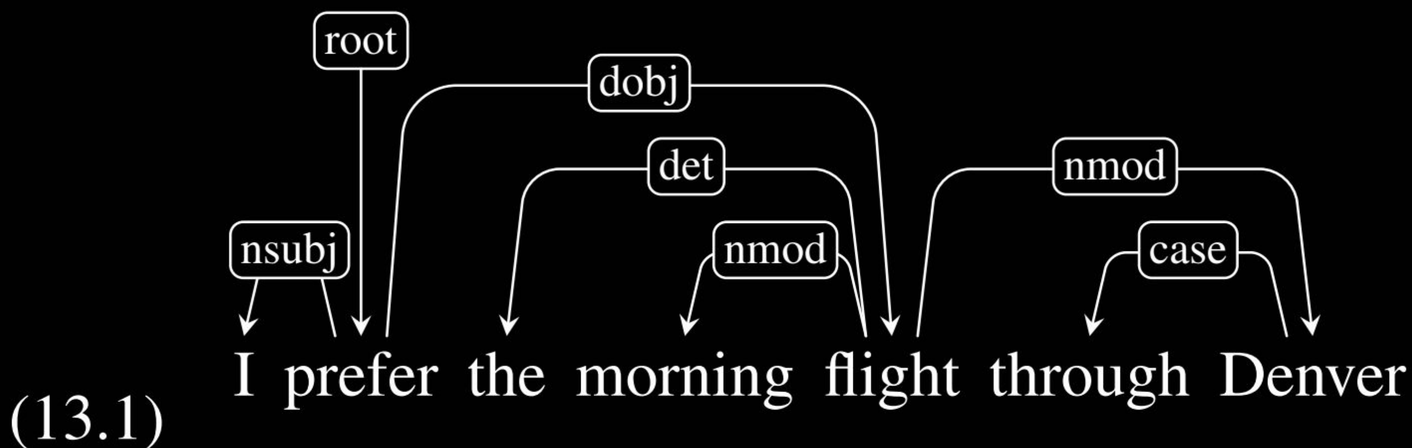
- **Machine learning:**
  - **Logistic regression**
  - **Conditional Random Fields**
- **Transition-Based Parsing**
- **Graph-based Parsing**

Dependency Parsing



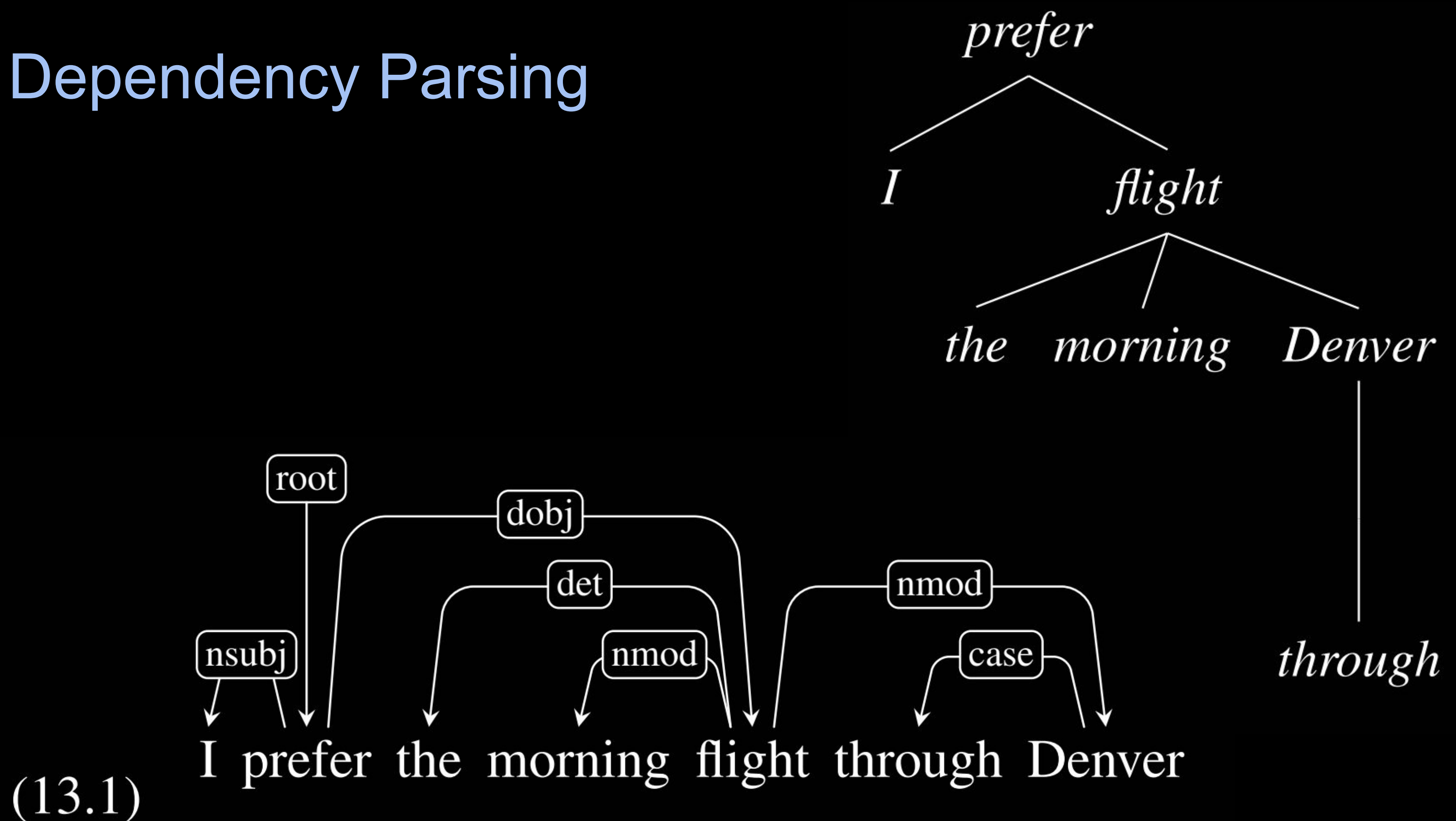
dependency -- binary asymmetrical relation between tokens

Dependency Parsing



(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing



(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing

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

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing

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 13.3 Examples of core Universal Dependency relations.

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing

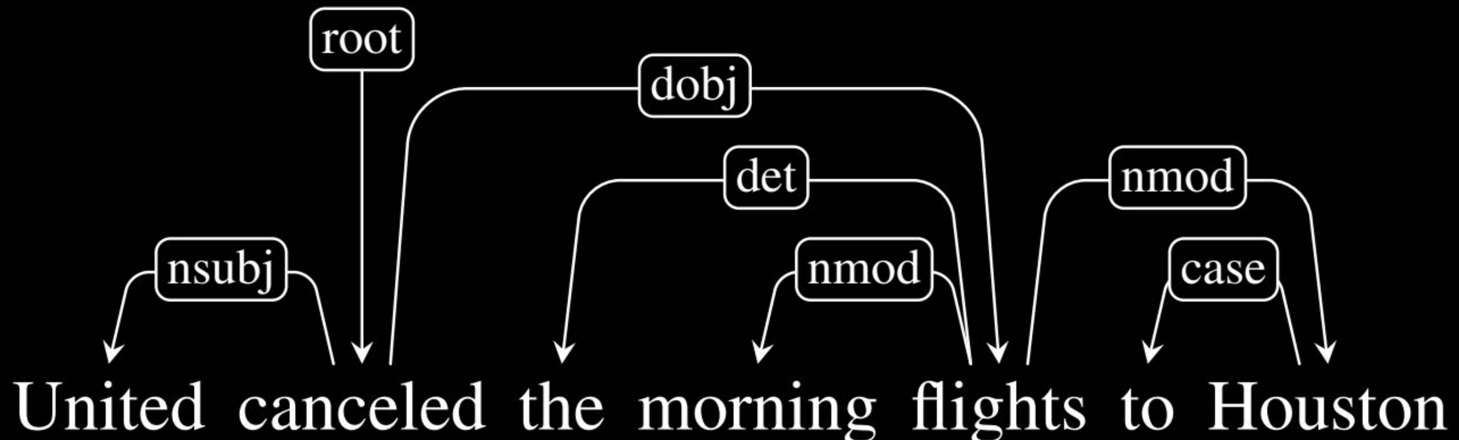
Verbal Predicate -- like a function, takes arguments: "United" and "the flight" in this case.

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 13.3 Examples of core Universal Dependency relations.

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- Verbal Predicates

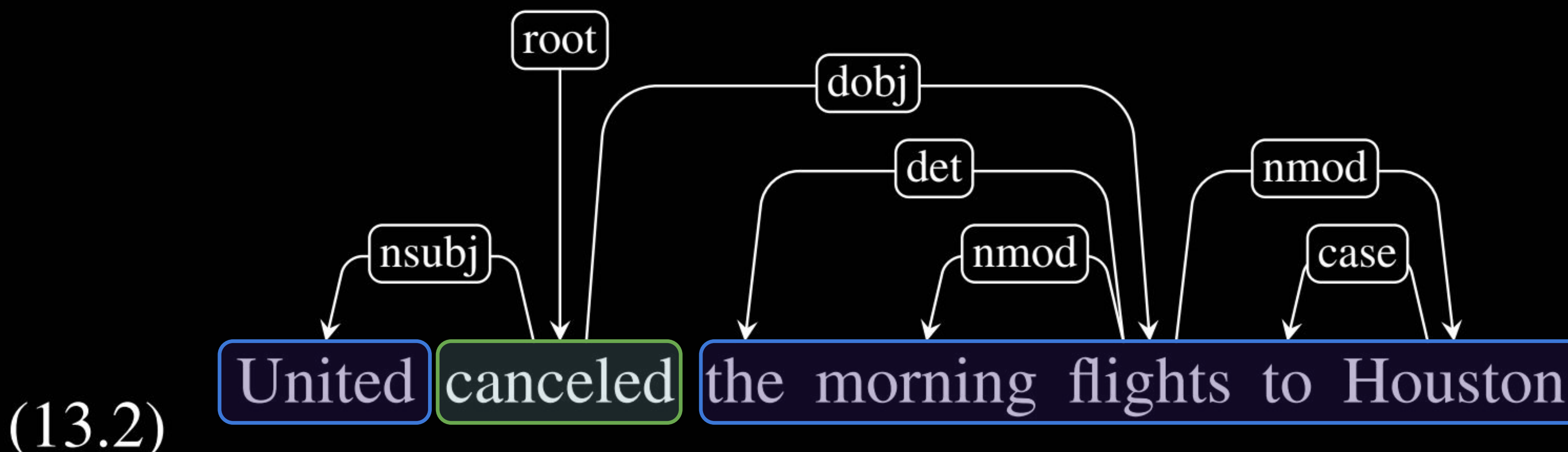


(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- Verbal Predicates

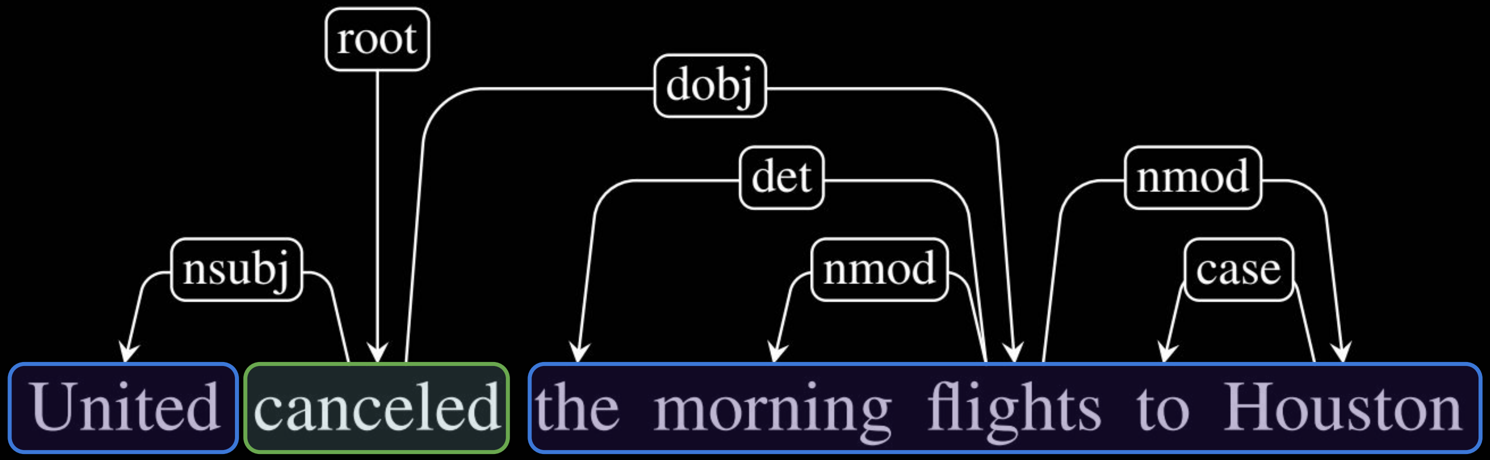
cancel("United", "the morning flights to Houston")



(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- Verbal Predicates

to_call_off("United", "the morning flights to Houston")

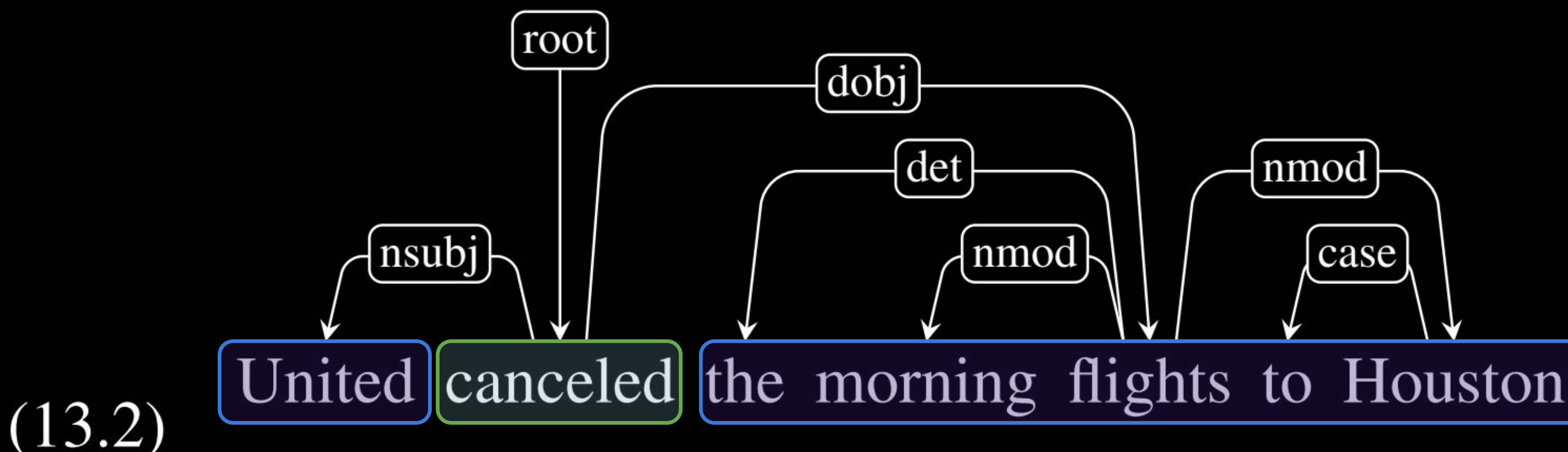


(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- Verbal Predicates
Semantic Roles

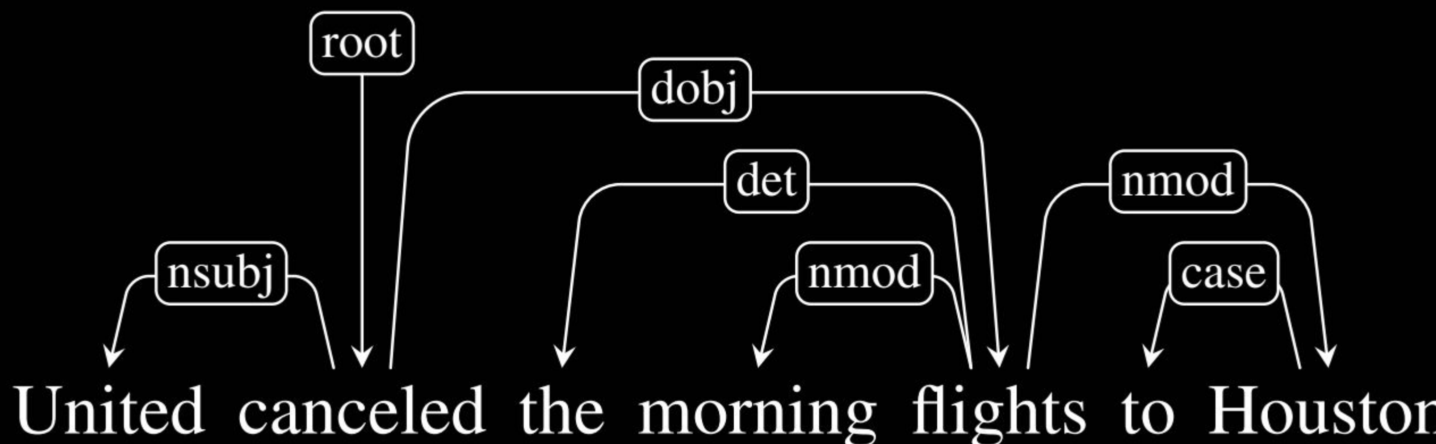
to_call_off(agent="United", event="the morning flights to Houston")



(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V2, A2), …] (vertices and arcs)
Restrictions:
?

(13.2) 

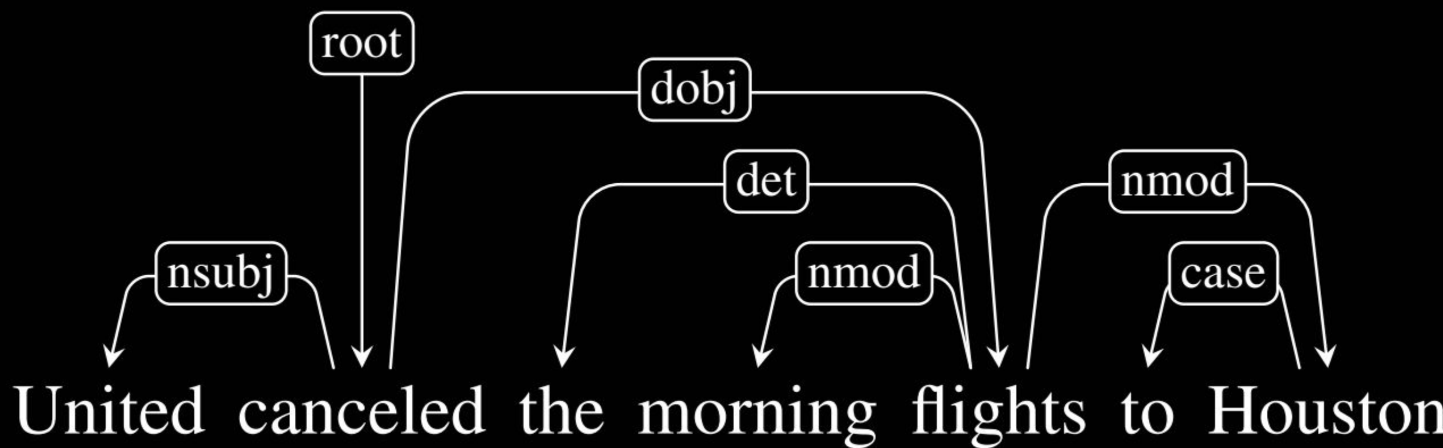
(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V2, A2), …]   (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex



(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

Inspired by “Shift-reduce parsing” -- process one word at a time, using a stack to keep some sort of memory.

Elements:

- *S*: stack, initialized with “ROOT”
- *B*: input buffer, initialized with tokens (w1, w2, ….) of sentence
- *A:* set of dependency arcs, initialized empty
- *T:* Actions, given *wi* (next token in stack)

Transition-based Dependency Parsing

Inspired by “Shift-reduce parsing” -- process one word at a time, using a stack to keep some sort of memory.

Elements:

- *S*: stack, initialized with “ROOT”
- *B*: input buffer, initialized with tokens (w1, w2, ….) of sentence
- *A:* set of dependency arcs, initialized empty
- *T:* Actions, given *wi* (next token in stack)
  - *shift*(*B*,*S*): move w from *B* to *S*
  - *left-arc(S,A):* make top of stack **head** of next item: add to A; remove dependent from stack
  - *right-arc(S,A):* make top of stack **dependent** of next item: add to A; remove dep from stack

Using discriminative classifiers (i.e. logistic regression) to make decisions.

Transition-based Dependency Parsing

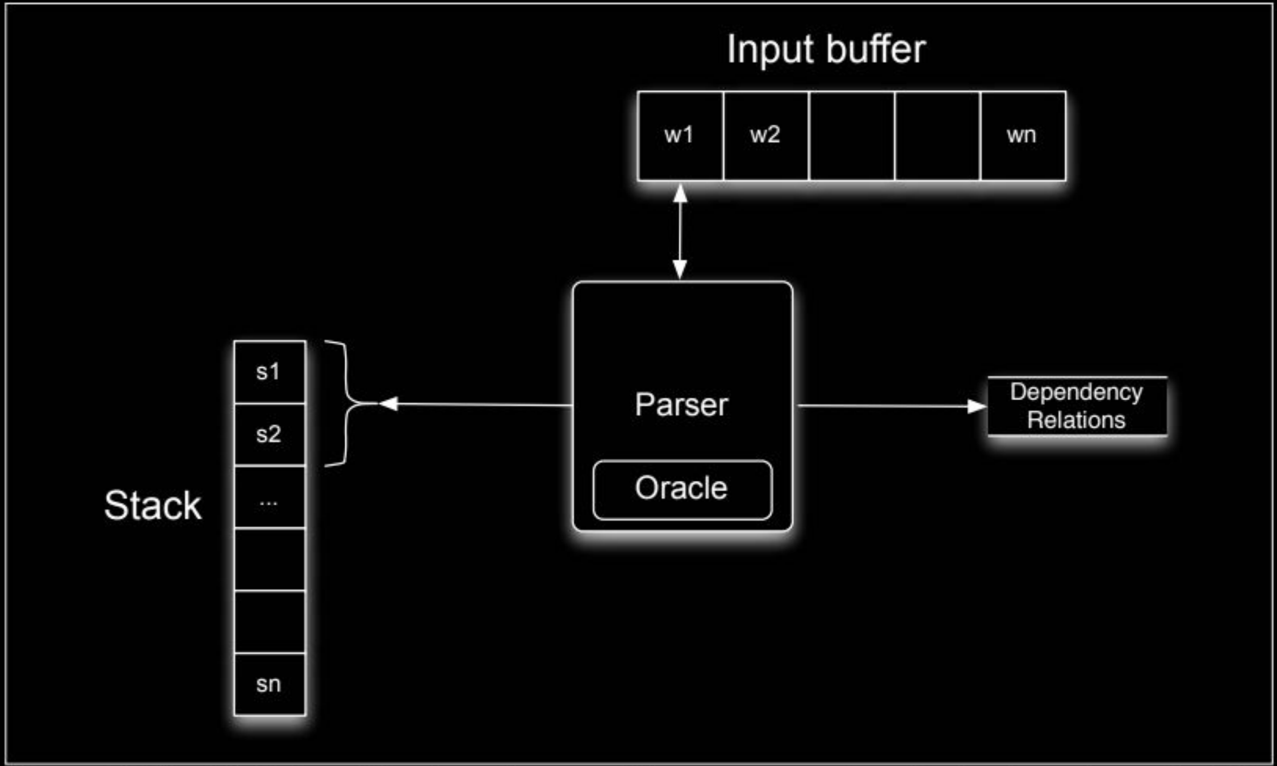


Figure 13.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

function DEPENDENCYPARSE(*words*) **returns** dependency tree

state ← {[root], [*words*], [] } ; initial configuration
while *state* **not final**
t ← ORACLE(*state*) ; choose a transition operator to apply
state ← APPLY(*t*, *state*) ; apply it, creating a new state
return *state*

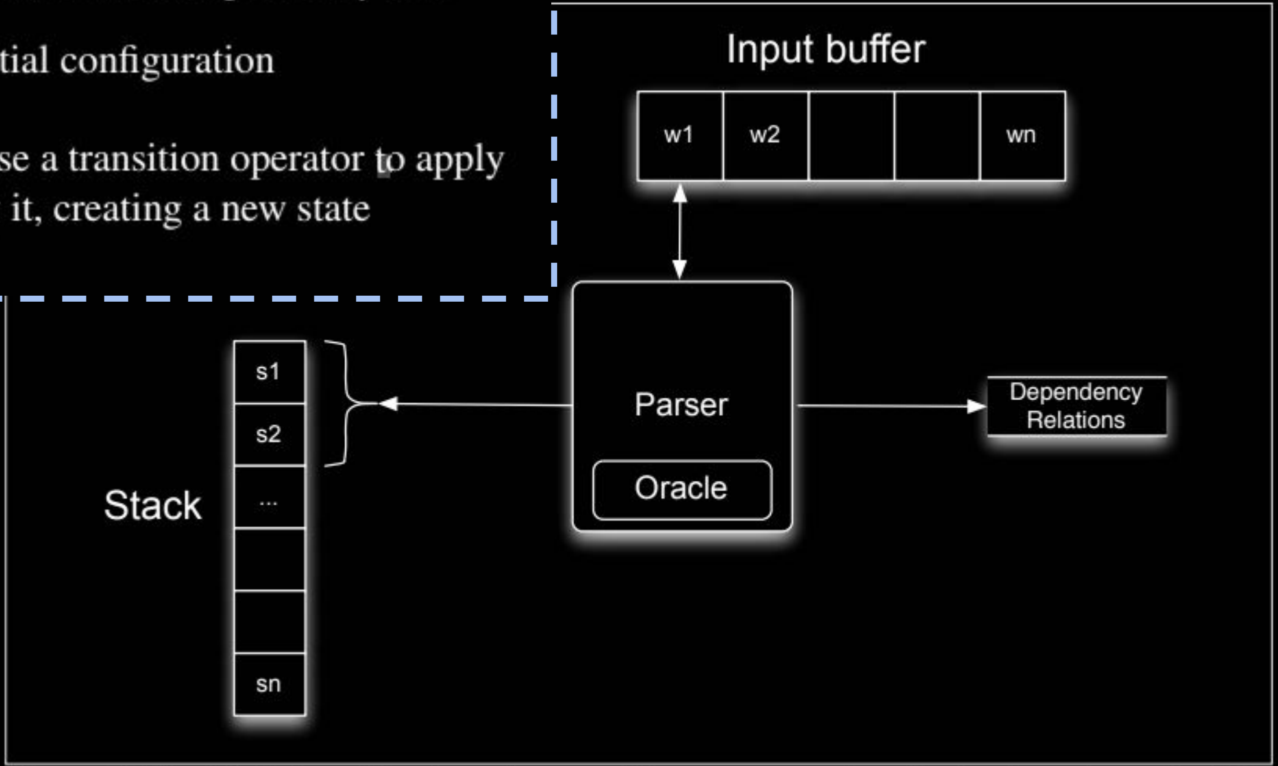


Figure 13.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)             ; choose a transition operator to apply
    state ← APPLY(t, state)       ; apply it, creating a new state
  return state
```

(13.5) Book me the morning flight

Let's consider the state of the configuration at Step 2, after the word *me* has been pushed onto the stack.

Stack	Word List	Relations
[root, book, me]	[the, morning, flight]	

The correct operator to apply here is RIGHTARC which assigns *book* as the head of *me* and pops *me* from the stack resulting in the following configuration.

Stack	Word List	Relations
[root, book]	[the, morning, flight]	(book → me)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	

shift(*B*,*S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	

shift(*B*,*S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

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 $\rightarrow$ me)

shift(*B*,*S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

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 $\rightarrow$ me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	

shift(*B*,*S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

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 $\rightarrow$ 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 $\leftarrow$ flight)

shift(*B*,*S*): move w from *B* to *S*

left-arc(S,A): make top of stack **head** of next item: add to A;
remove dependent from stack

right-arc(S,A): make top of stack **dependent** of next item: add to A;
remove dep from stack

(From SLP 3rd ed., Jurafsky and Martin 2018)

Transition-based Dependency Parsing

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 13.7 Trace of a transition-based parse.

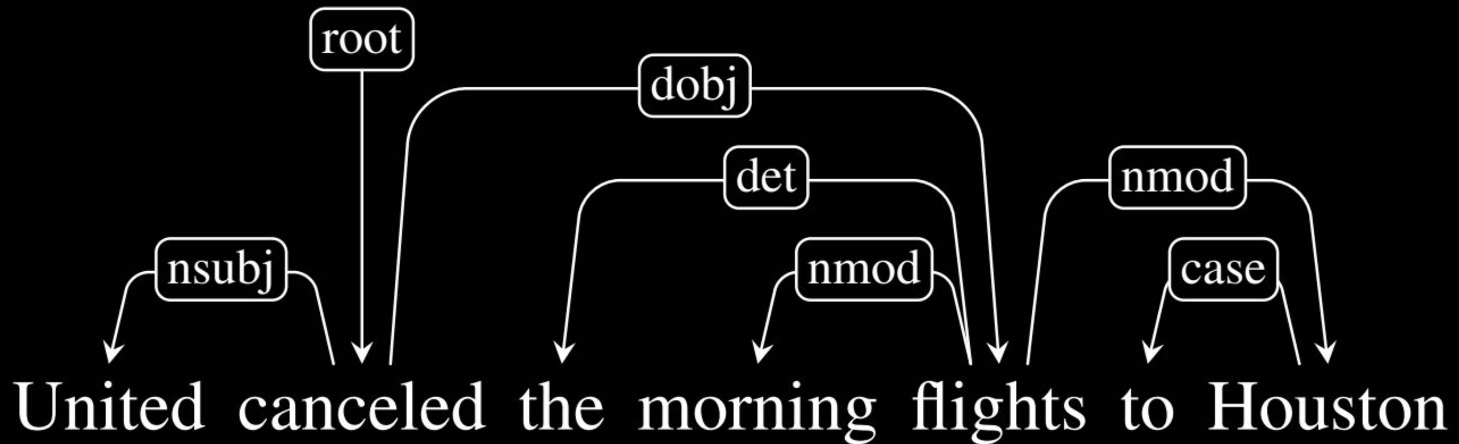
(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex



(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

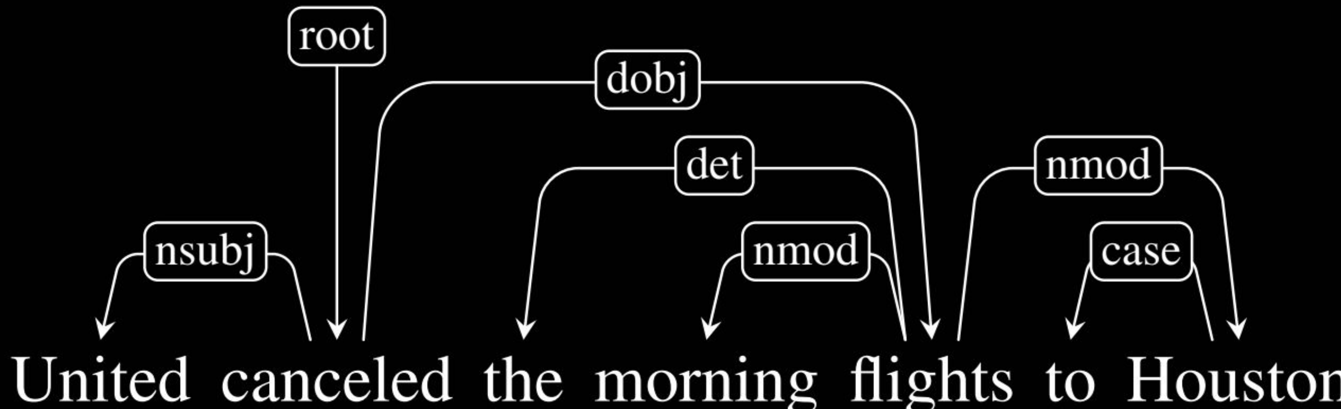
Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word



(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

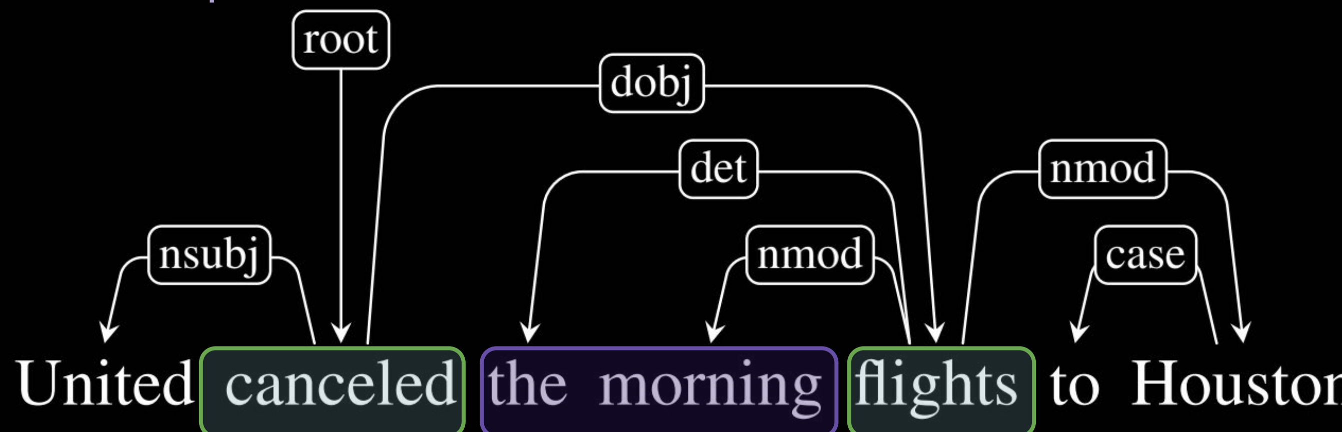
Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …]   (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word



(13.2)

(From SLP 3rd ed., Jurafsky and Martin 2018)

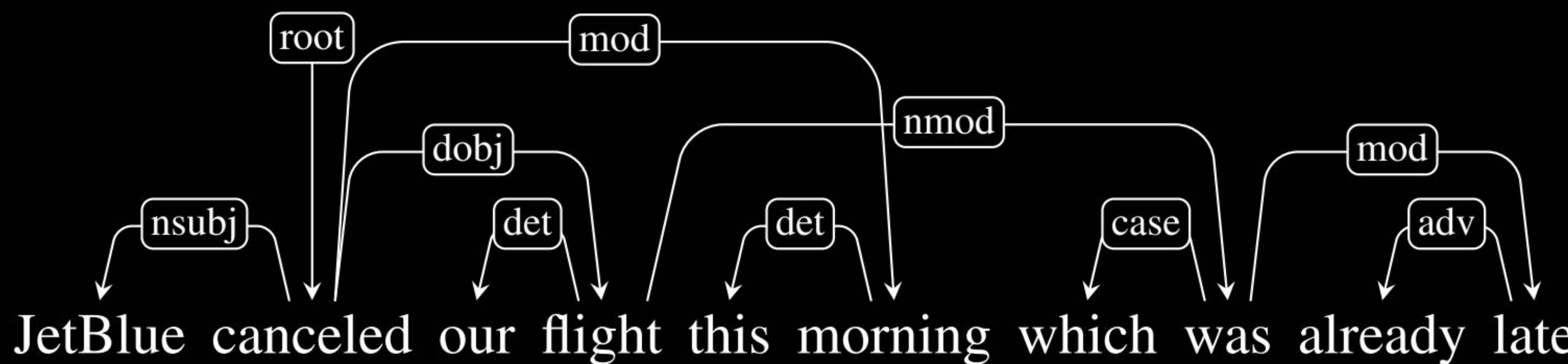
Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word



(13.3)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

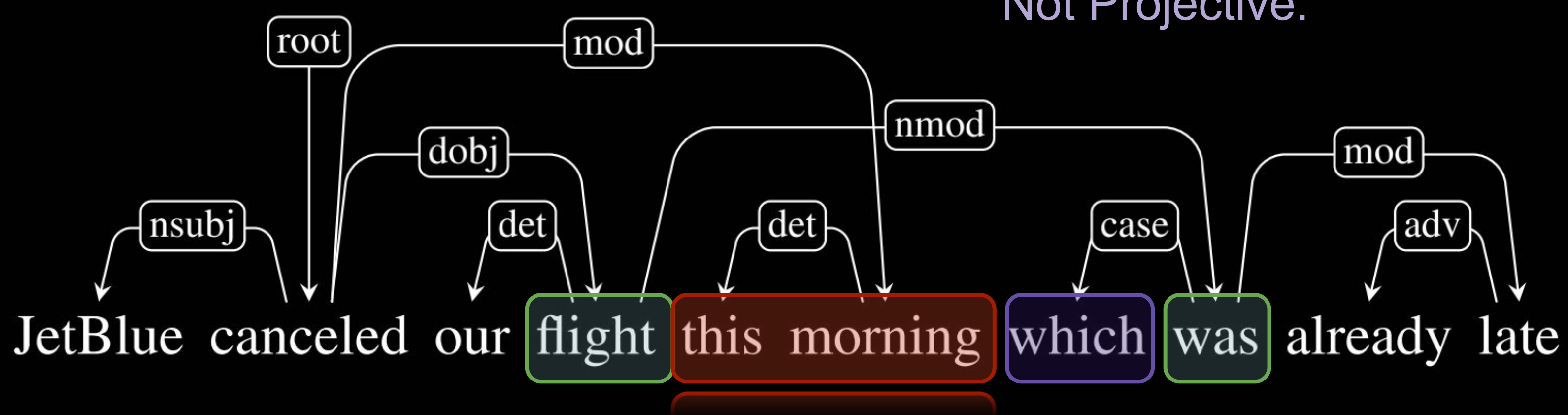
A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word.

Not Projective:



(13.3)

(From SLP 3rd ed., Jurafsky and Martin 2018)

Dependency Parsing -- How to Represent?

A Graph: G = [(V1, A1), (V1, A2), …]   (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Projectivity: Given head, dependent; for every word between head and dependent there exists a path from head to that word.

Not Projective:

Why do we care? Dependency trees from Context-Free Grammars are guaranteed to be projective; Thus, transition based techniques are certain to have errors occasionally on non-projective dependency graphs.

(From SLP 3rd ed., Jurafsky and Martin 2018)

From Syntax to Semantics

- We've already seen words have many meanings.
  - Context is key

- Verbs can been seen as functions (predicates) that take arguments.
  - **Syntactic** arguments fulfill **semantic** roles

- Words have implicit syntactic relationships
  with each other in given sentences.
  - Dependency Parsing: each word has one head
  - Easily constructed through 3 actions of shift-reduce parsing.

Takeaway: There is an interplay between word meaning and sentence structure!

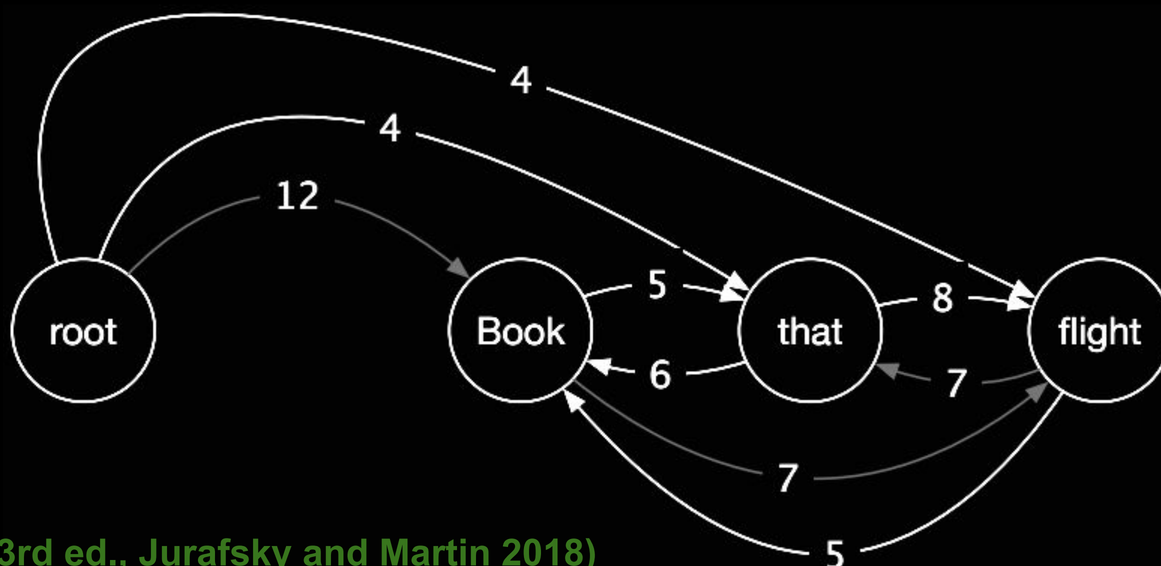
Graph-based Approaches

A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Idea: Search through all possible trees and pick best.



(From SLP 3rd ed., Jurafsky and Martin 2018)

Graph-based Approaches

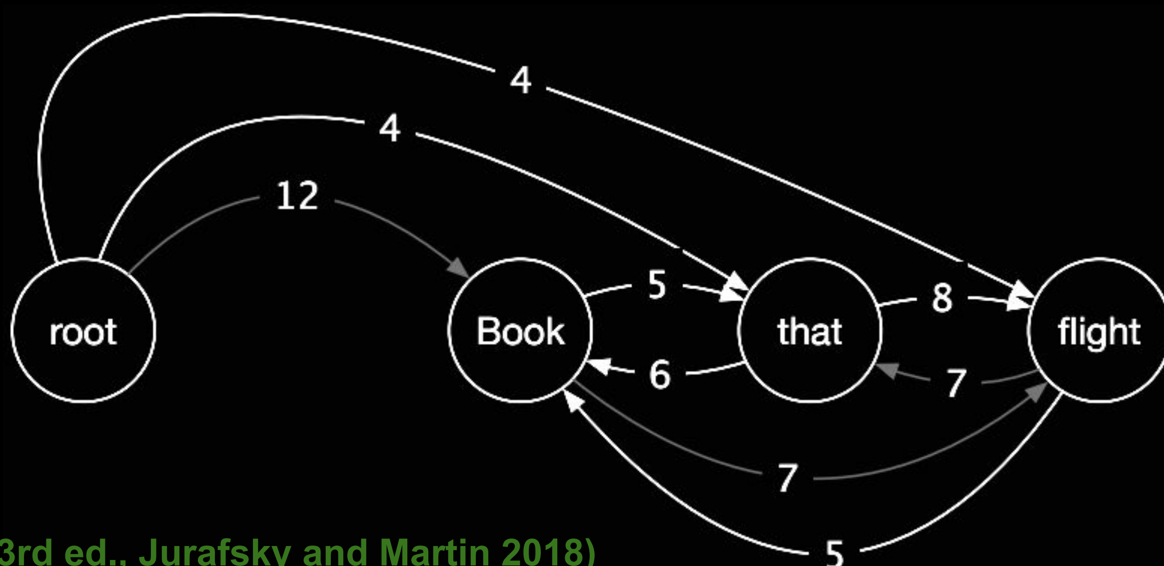
A Graph: G = [(V1, A1), (V1, A2), …] (vertices and arcs)

Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Idea: Search through all possible trees and pick best.

General approach: For each word, pick the most likely head. Then check if still a fully-connected tree, and adjust.



(From SLP 3rd ed., Jurafsky and Martin 2018)

Graph-based Approaches

A Graph: G = [(V1, A1), (V1, A2), …]   (vertices and arcs)

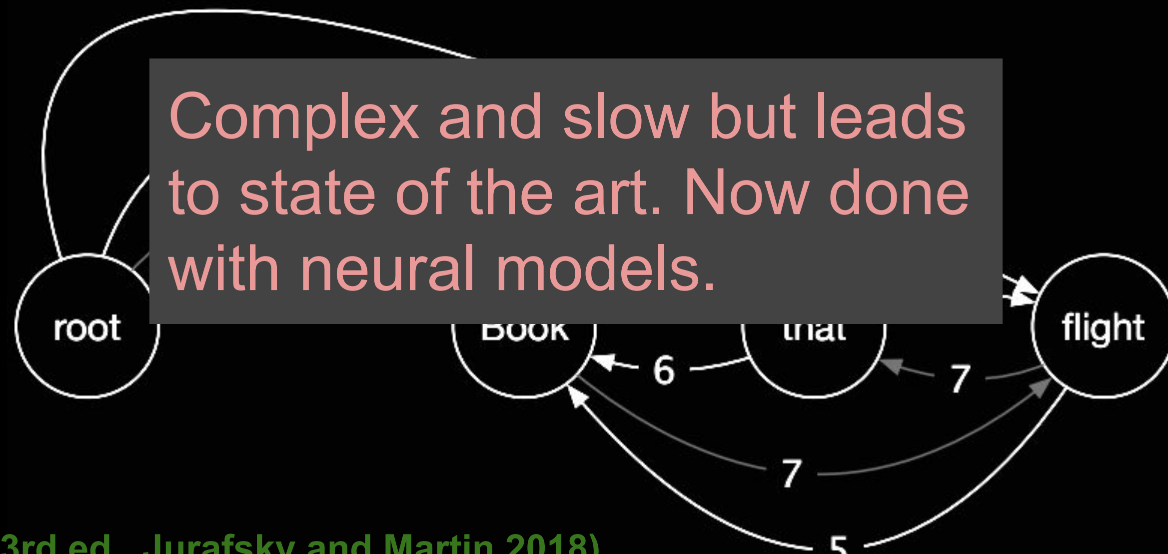
Restrictions:

1) Single designated ROOT with no incoming arcs
2) Every vertex only has one head (parent, governer); i.e. only one incoming arc
3) unique path from ROOT to every vertex

Idea: Search through all possible trees and pick best.

General approach: For each word, pick the most likely head. Then check if still a fully-connected tree, and adjust.

Complex and slow but leads to state of the art. Now done with neural models.

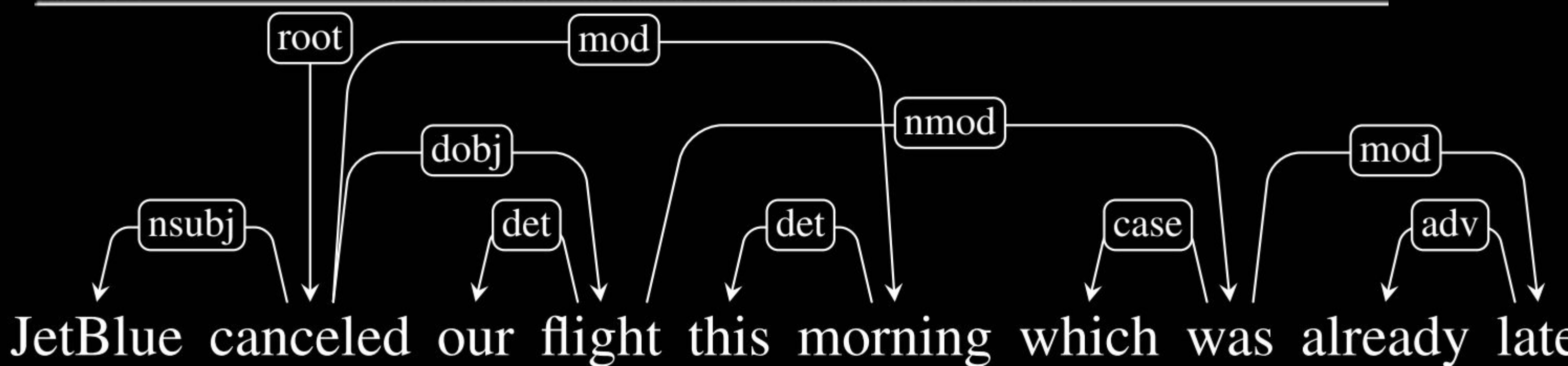


(From SLP 3rd ed., Jurafsky and Martin 2018)

Relation to Semantic Roles

Thematic Role	Definition
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
RESULT	The end product of an event
CONTENT	The proposition or content of a propositional event
INSTRUMENT	An instrument used in an event
BENEFICIARY	The beneficiary of an event
SOURCE	The origin of the object of a transfer event
GOAL	The destination of an object of a transfer event

(13.3)



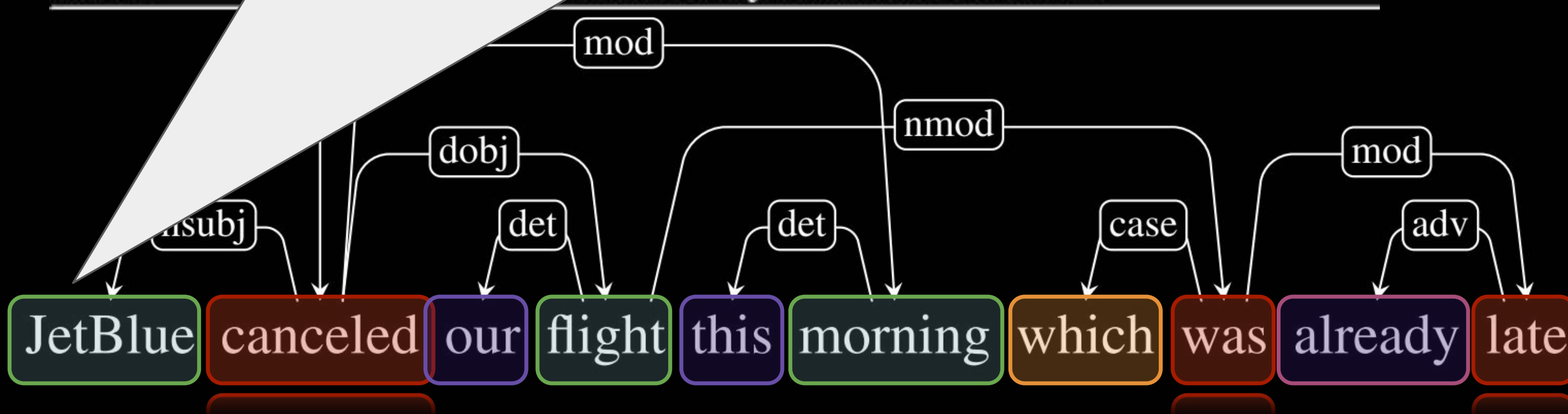
(From SLP 3rd ed., Jurafsky and Martin 2018)

Relation to Semantic Roles

Thematic Role	Definition
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
GOAL	...n of an object of a transfer event

Roles are restricted to nouns, but signalled through the verb and other parts of speech.

(13.3)



(From SLP 3rd ed., Jurafsky and Martin 2018)

Parts-of-Speech

Open Class (also known as “*content words*”):

Nouns, Verbs, Adjectives, Adverbs

Parts-of-Speech

Open Class (also known as “*content words*”):

Nouns, Verbs, Adjectives, Adverbs

Function words:

Determiners, conjunctions, pronouns, prepositions

mostly specify syntactic structure; express broad semantics connecting content words

Parts-of-Speech: The Penn Treebank Tagset

Table 2
The Penn Treebank POS tagset.

1.	CC	Coordinating conjunction	25.	TO	*to*
2.	CD	Cardinal number	26.	UH	Interjection
3.	DT	Determiner	27.	VB	Verb, base form
4.	EX	Existential *there*	28.	VBD	Verb, past tense
5.	FW	Foreign word	29.	VBG	Verb, gerund/present participle
6.	IN	Preposition/subordinating conjunction	30.	VBN	Verb, past participle
7.	JJ	Adjective	31.	VBP	Verb, non-3rd ps. sing. present
8.	JJR	Adjective, comparative	32.	VBZ	Verb, 3rd ps. sing. present
9.	JJS	Adjective, superlative	33.	WDT	*wh*-determiner
10.	LS	List item marker	34.	WP	*wh*-pronoun
11.	MD	Modal	35.	WP$	Possessive *wh*-pronoun
12.	NN	Noun, singular or mass	36.	WRB	*wh*-adverb
13.	NNS	Noun, plural	37.	#	Pound sign
14.	NNP	Proper noun, singular	38.	$	Dollar sign
15.	NNPS	Proper noun, plural	39.	.	Sentence-final punctuation
16.	PDT	Predeterminer	40.	,	Comma
17.	POS	Possessive ending	41.	:	Colon, semi-colon
18.	PRP	Personal pronoun	42.	(	Left bracket character
19.	PP$	Possessive pronoun	43.	)	Right bracket character
20.	RB	Adverb	44.	"	Straight double quote
21.	RBR	Adverb, comparative	45.	‘	Left open single quote
22.	RBS	Adverb, superlative	46.	“	Left open double quote
23.	RP	Particle	47.	’	Right close single quote
24.	SYM	Symbol (mathematical or scientific)	48.	”	Right close double quote

Parts-of-Speech: Social Media Tagset

(Gimpel et al., 2010)

Tag	Description	Examples	%
Nominal, Nominal + Verbal			
N	common noun (NN, NNS)	books someone	13.7
O	pronoun (personal/WH; not possessive; PRP, WP)	it you u meeee	6.8
S	nominal + possessive	books' someone's	0.1
^	proper noun (NNP, NNPS)	lebron usa iPad	6.4
Z	proper noun + possessive	America's	0.2
L	nominal + verbal	he's book'll iono (= *I don't know*)	1.6
M	proper noun + verbal	Mark'll	0.0

Other open-class words			
V	verb incl. copula, auxiliaries (V*, MD)	might gonna ought couldn't is eats	15.1
A	adjective (J*)	good fav lil	5.1
R	adverb (R*, WRB)	2 (i.e., *too*)	4.6
!	interjection (UH)	lol haha FTW yea right	2.6
Other closed-class words			
D	determiner (WDT, DT, WP$, PRP$)	the teh its it's	6.5
P	pre- or postposition, or subordinating conjunction (IN, TO)	while to for 2 (i.e., *to*) 4 (i.e., *for*)	8.7
&	coordinating conjunction (CC)	and n & + BUT	1.7
T	verb particle (RP)	out off Up UP	0.6
X	existential *there*, predeterminers (EX, PDT)	both	0.1
Y	X + verbal	there's all's	0.0

Twitter/online-specific			
#	hashtag (indicates topic/category for tweet)	#acl	1.0
@	at-mention (indicates another user as a recipient of a tweet)	@BarackObama	4.9
~	discourse marker, indications of continuation of a message across multiple tweets	RT and : in retweet construction RT @user : hello	3.4
U	URL or email address	http://bit.ly/xyz	1.6
E	emoticon	:-) :b (: <3 o_O	1.0
Miscellaneous			
$	numeral (CD)	2010 four 9:30	1.5
,	punctuation (#, $, '', (,), ,, ., :, ``)	!!! .... ?!?	11.6
G	other abbreviations, foreign words, possessive endings, symbols, garbage (FW, POS, SYM, LS)	ily (*I love you*) wby (*what about you*) 's ♫ --> awesome...I'm	1.1

POS Tagging: Applications

- Resolving ambiguity (speech: “lead”)
- Shallow searching: find noun phrases
- Speed up parsing
- Use as feature (or in place of word)

POS Tagging: Applications

- Resolving ambiguity (speech: “lead”)
- Shallow searching: find noun phrases
- Speed up parsing
- Use as feature (or in place of word)

- Understand what modern deep learning methods are dealing with implicitly.

Window-based POS Tagging

Approach like we did word sense disambiguation...

The book looks brief so I am happy .

?

Window-based POS Tagging

The book looks brief so I am happy .

D

Window-based POS Tagging

The book looks brief so I am happy .

D *N*

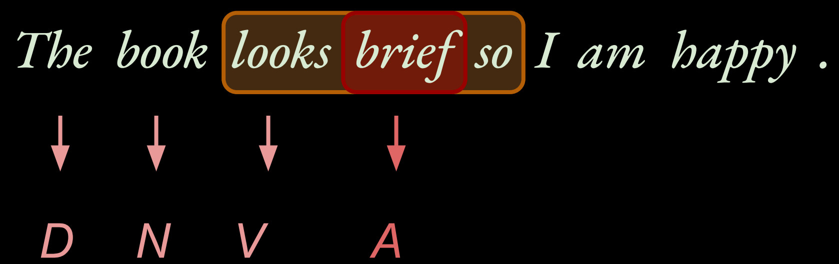
Window-based POS Tagging

The book looks brief so I am happy .

D N ?

Window-based POS Tagging

Window-based POS Tagging

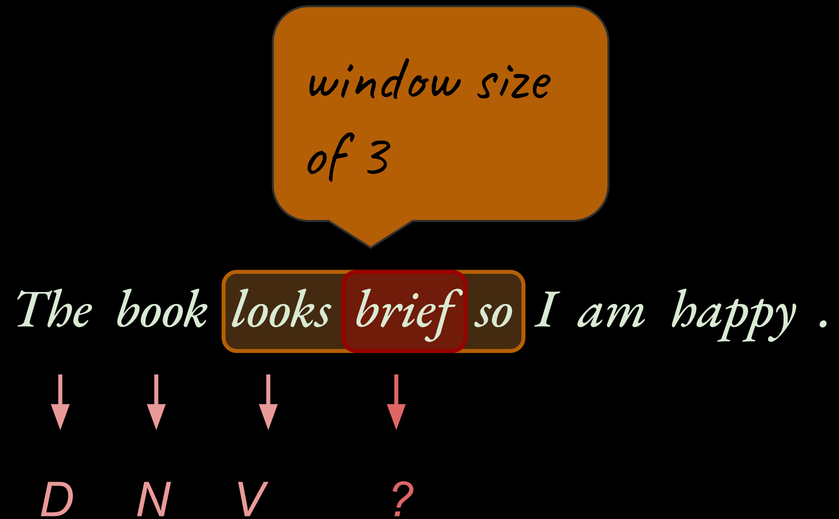


Window-based POS Tagging

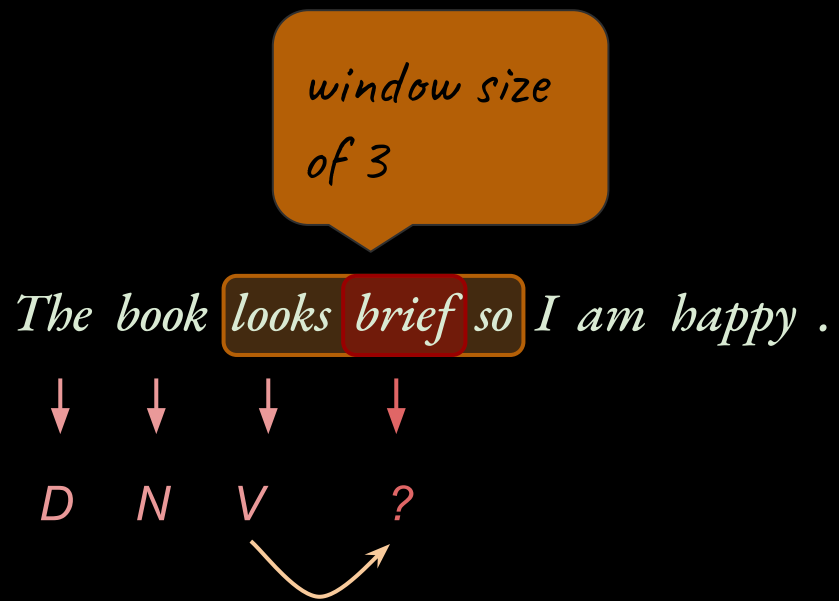
The book looks brief so I am happy .

D N V ?

Window-based POS Tagging



Window-based POS Tagging



Window-based POS Tagging

window size of 3

The book looks brief so I am happy .

D N V ?

$P(pos_i = \text{'N'} \mid word_i = \text{"brief"}) = 0.3$

Window-based POS Tagging

window size of 3

The book $\boxed{\textit{looks } \boxed{\textit{brief}} \textit{ so}}$ *I am happy .*

D N V ?

$$P(pos_i = \text{'N'} \,|\, word_i = \text{"brief"}) = 0.3$$
$$P(pos_i = \text{'V'} \,|\, word_i = \text{"brief"}) = 0.4$$
$$P(pos_i = \text{'A'} \,|\, word_i = \text{"brief"}) = 0.3$$

Window-based POS Tagging

window size of 3

The book looks brief so I am happy .

D N V ?

$$P(p_i = \text{'N'} \mid w_i = \text{brief}) = .30$$
$$P(p_i = \text{'V'} \mid w_i = \text{brief}) = .40$$
$$P(p_i = \text{'A'} \mid w_i = \text{brief}) = .30$$

Window-based POS Tagging

window size of 3

The book looks brief so I am happy .

D N V ?

$$P(p_i = \text{'N'} / w_i = \text{brief}, w_{i-1} = \text{looks}, w_{i+1} = \text{so}) = ??$$

$$P(p_i = \text{'V'} / w_i = \text{brief}, w_{i-1} = \text{looks}, w_{i+1} = \text{so}) = ??$$

$$P(p_i = \text{'A'} / w_i = \text{brief}, w_{i-1} = \text{looks}, w_{i+1} = \text{so}) = ??$$

Window-based POS Tagging

window size of 3

ideal result

The book looks brief so I am happy .

D N V ?

$P(p_i='N' | w_i=brief, w_{i-1}=looks, w_{i+1}=so) = .005$

$P(p_i='V' | w_i=brief, w_{i-1}=looks, w_{i+1}=so) = .005$

$P(p_i='A' | w_i=brief, w_{i-1}=looks, w_{i+1}=so) = .99$

Window-based POS Tagging

window size of 3

The book looks brief so I am happy .

D N V ?

$$P(p_i=\text{'N'} \mid w_i=brief, w_{i-1}=looks, w_{i+1}=so) = .3$$
$$P(p_i=\text{'V'} \mid w_i=brief, w_{i-1}=looks, w_{i+1}=so) = .4$$
$$P(p_i=\text{'A'} \mid w_i=brief, w_{i-1}=looks, w_{i+1}=so) = .3$$

More likely, because we haven't seen this context before.

Window-based POS Tagging

window size of 3

The book [*looks* **brief** *so*] *I am happy .*

The	book	looks	brief
D	N	V	?

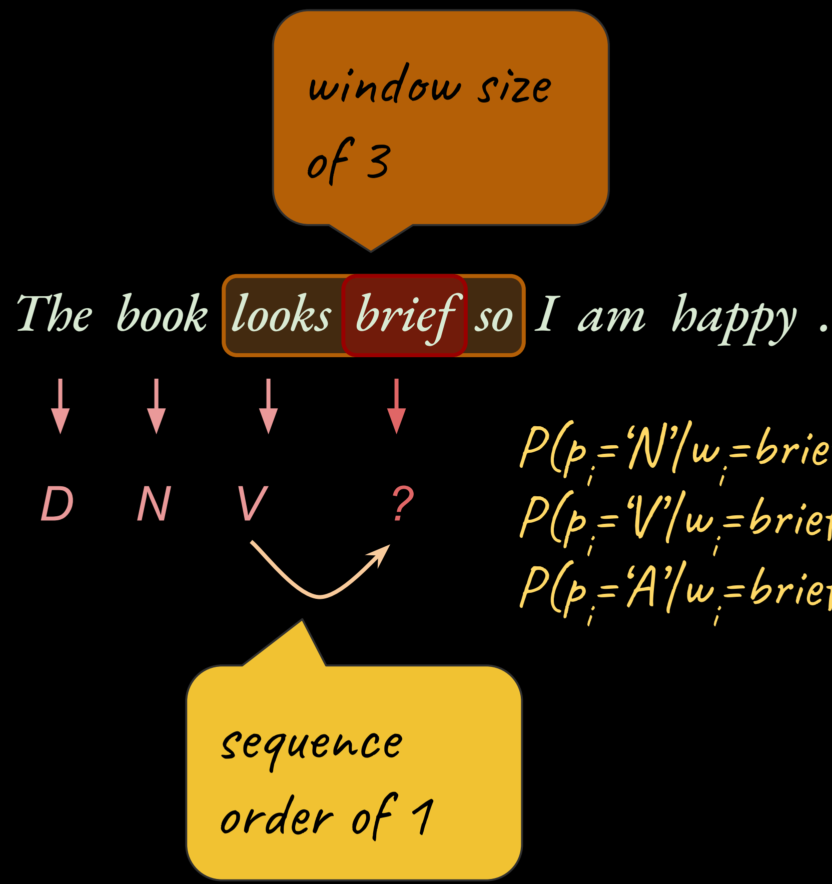
$P(p_i=\text{'N'} \mid w_i=brief, w_{i-1}=looks, w_{i+1}=so) = .3$

$P(p_i=\text{'V'} \mid w_i=brief, w_{i-1}=looks, w_{i+1}=so) = .4$

$P(p_i=\text{'A'} \mid w_i=brief, w_{i-1}=looks, w_{i+1}=so) = .3$

More likely, because we haven't seen this context before.

Sequential Model

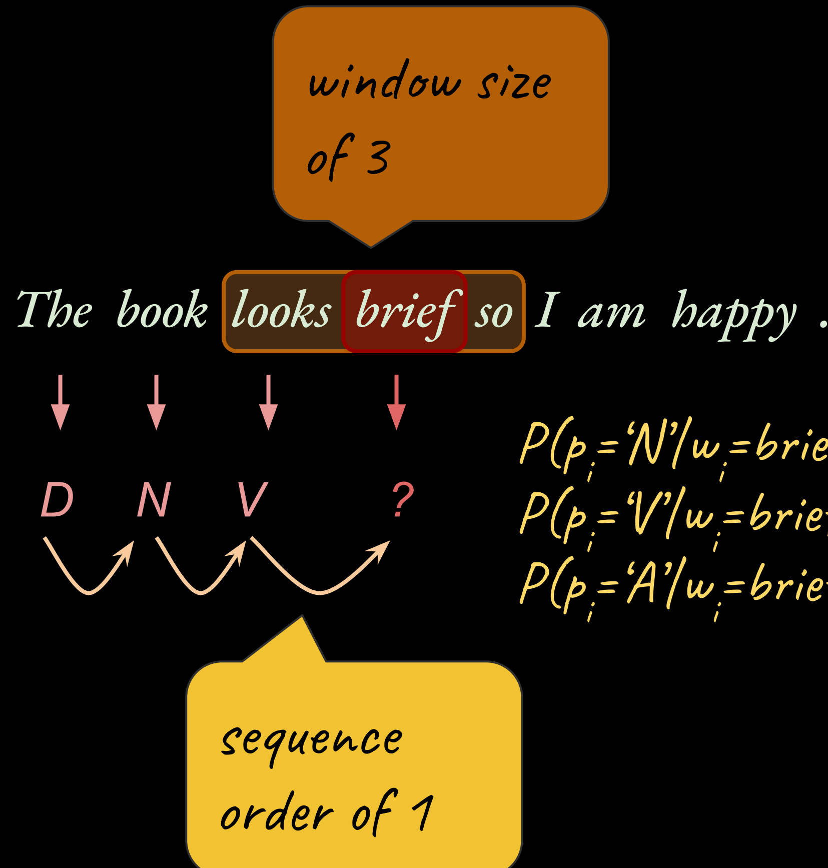


$$P(p_i = \text{'N'} \mid w_i = \text{brief}, w_{i-1} = \text{looks}, w_{i+1} = \text{so}) = .3$$

$$P(p_i = \text{'V'} \mid w_i = \text{brief}, w_{i-1} = \text{looks}, w_{i+1} = \text{so}) = .4$$

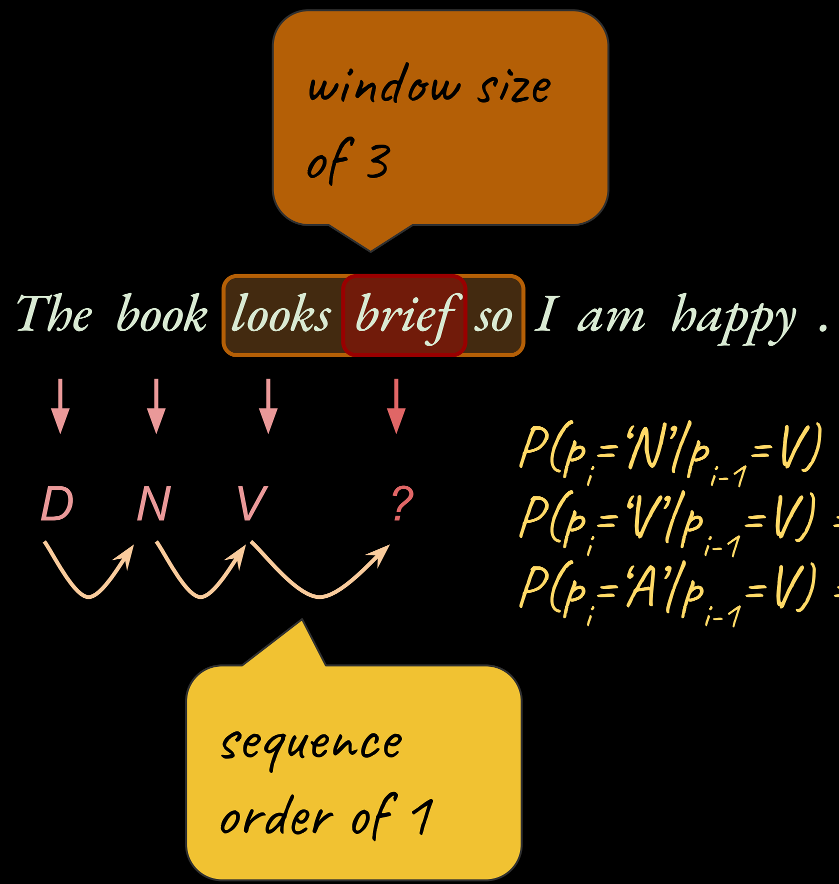
$$P(p_i = \text{'A'} \mid w_i = \text{brief}, w_{i-1} = \text{looks}, w_{i+1} = \text{so}) = .3$$

Sequential Model



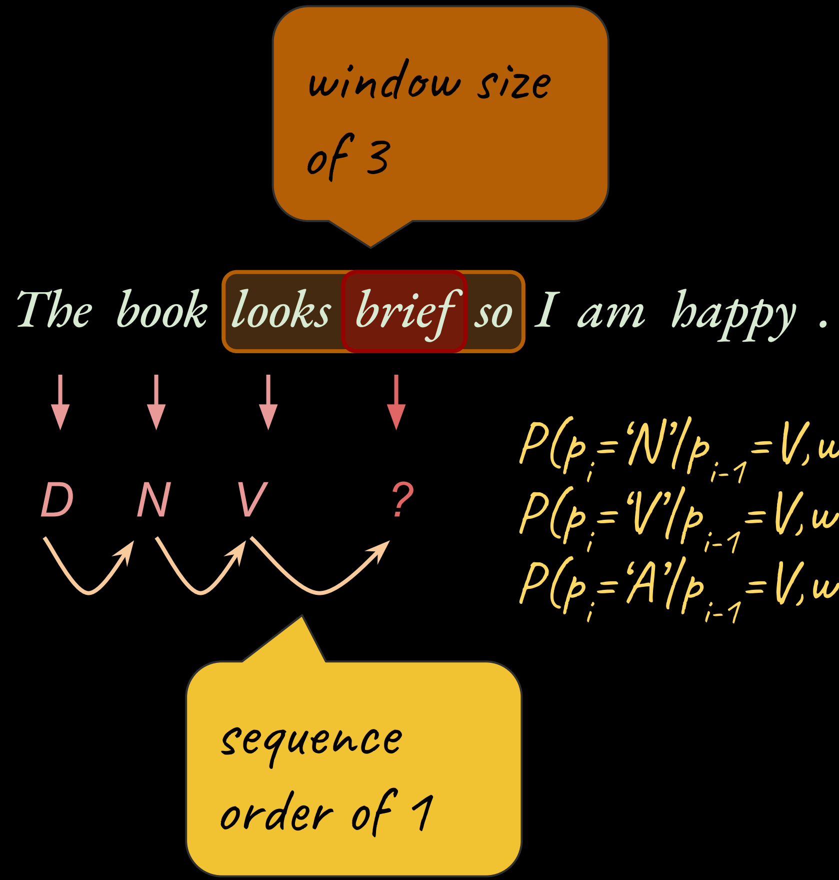
$$P(p_i = \text{'N'} | w_i = brief, w_{i-1} = looks, w_{i+1} = so) = .3$$
$$P(p_i = \text{'V'} | w_i = brief, w_{i-1} = looks, w_{i+1} = so) = .4$$
$$P(p_i = \text{'A'} | w_i = brief, w_{i-1} = looks, w_{i+1} = so) = .3$$

Sequential Model



$$P(p_i = \text{'N'} \mid p_{i-1} = V) = .4$$
$$P(p_i = \text{'V'} \mid p_{i-1} = V) = .10$$
$$P(p_i = \text{'A'} \mid p_{i-1} = V) = .4$$

Sequential Model



$$P(p_i = \text{'N'} | p_{i-1} = V, w_i = brief) = .3$$
$$P(p_i = \text{'V'} | p_{i-1} = V, w_i = brief) = .05$$
$$P(p_i = \text{'A'} | p_{i-1} = V, w_i = brief) = .65$$

Sequence modeling

-- Tasks that in which a current label is dependent on previous labels within a sequence.

More generally: tasks that can leverage the order of words.

Most basic example: *Language Modeling*

-- Predicting the next word given previous.