DSC291: Advanced Statistical Natural Language Processing

Sequence Tagging
Parsing

Zhiting Hu

Lecture 11, May 3, 2022

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

Outline

- Sequence Tagging/Labeling
- Parsing

Sequence Labeling

[Slides adapted from UW CSE 447 by Noah Smith]

Recap: Sequence Labeling

Problem statement: given a sequence of n words $\boldsymbol{x}$, assign each a label from $\mathcal{L}$. Let $L = |\mathcal{L}|$.

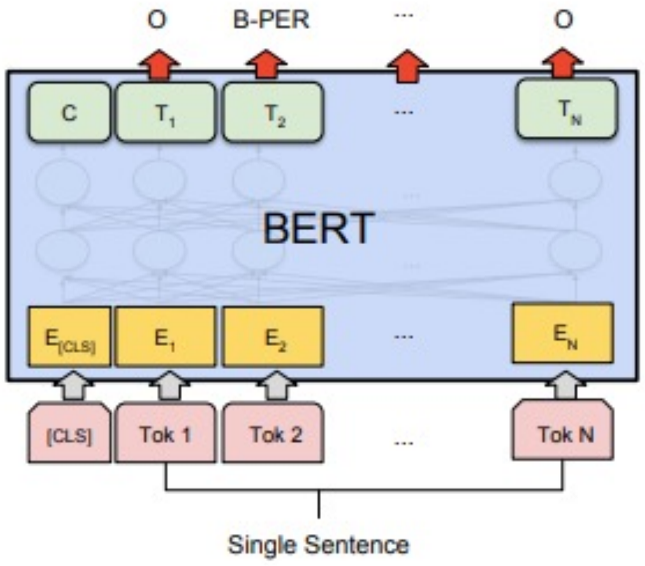
Every approach we see today will cast the problem as:

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^n} \mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$$

Naïvely, that's a classification problem where the number of possible 'labels" (output sequences) depends on the input and is $O(L^n)$ in size!

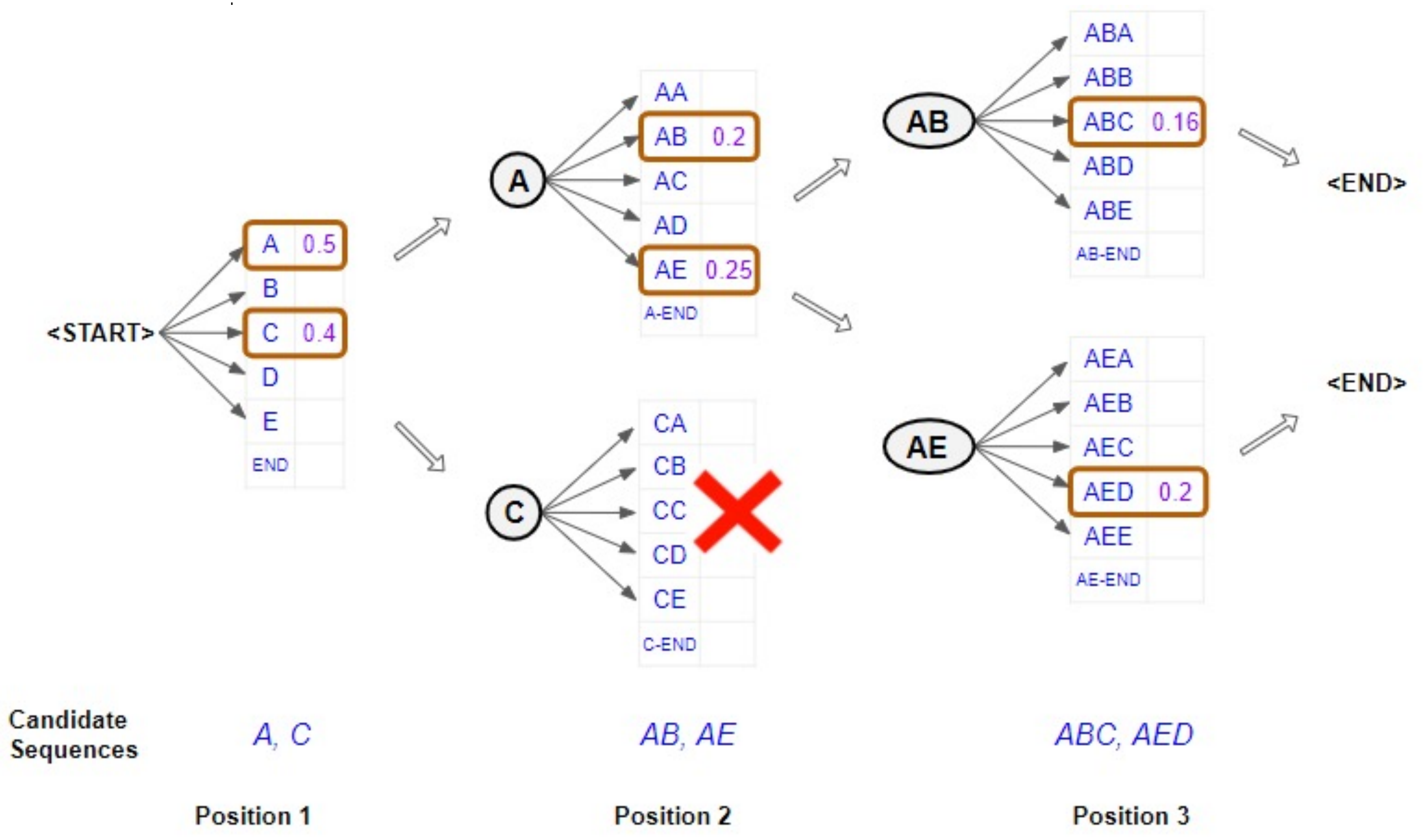
Recap: Sequence Labeling

Model	0	1	2	3	4
Score decomp.	$s(\boldsymbol{x}, i, y_i)$	$s(\boldsymbol{x}, i, \boldsymbol{y}_{1:i})$	emission/ transition	$s(x_i, y_i)+$ $s(y_i, y_{i+1})$	$s(\boldsymbol{x}, i, y_i, y_{i+1})$
learn	SGD	?	count & normalize	?	?
decode	local	beam search	Viterbi	Viterbi	Viterbi



model v.0

Recap: Sequence Labeling



model v.1

Recap: Sequence Labeling

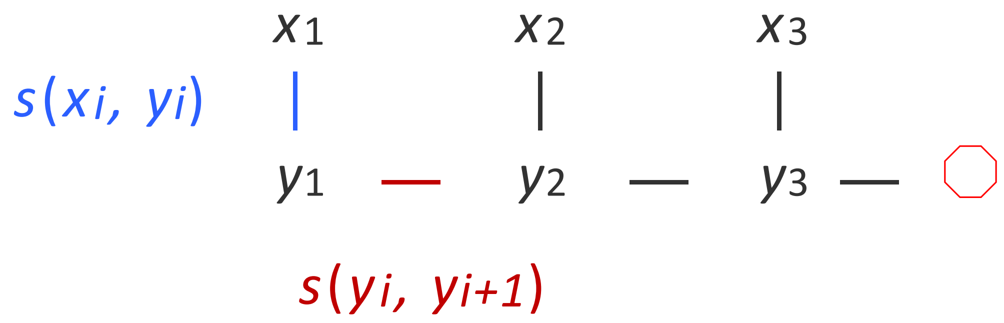
Model	0	1	2	3	4
Score decomp.	$s(\boldsymbol{x}, i, y_i)$	$s(\boldsymbol{x}, i, \boldsymbol{y}_{1:i})$	emission/ transition	$s(x_i, y_i)+$ $s(y_i, y_{i+1})$	$s(\boldsymbol{x}, i, y_i, y_{i+1})$
learn	SGD	?	count & normalize	?	?
decode	local	beam search	Viterbi	Viterbi	Viterbi

$$\begin{array}{ccccccc} & & x_1 & & x_2 & & x_3 \\ x_i \sim p_{emission}(X \mid y_i) & & \uparrow & & \uparrow & & \uparrow \\ & & y_1 & \rightarrow & y_2 & \rightarrow & y_3 & \rightarrow \end{array}$$

$$y_{i+1} \sim p_{transition}(Y \mid y_i)$$

Recap: Sequence Labeling

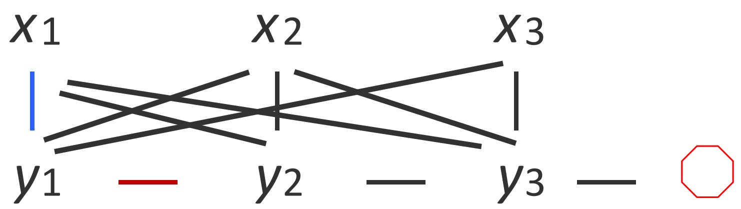
Model	0	1	2	3	4
Score decomp.	$s(\boldsymbol{x}, i, y_i)$	$s(\boldsymbol{x}, i, \boldsymbol{y}_{1:i})$	emission/ transition	$s(x_i, y_i)+$ $s(y_i, y_{i+1})$	$s(\boldsymbol{x}, i, y_i, y_{i+1})$
learn	SGD	?	count & normalize	?	?
decode	local	beam search	Viterbi	Viterbi	Viterbi



model v.3

Recap: Sequence Labeling

Model	0	1	2	3	4
Score decomp.	$s(\boldsymbol{x}, i, y_i)$	$s(\boldsymbol{x}, i, \boldsymbol{y}_{1:i})$	emission/ transition	$s(x_i, y_i) +$ $s(y_i, y_{i+1})$	$s(\boldsymbol{x}, i, y_i, y_{i+1})$
learn	SGD	?	count & normalize	?	?
decode	local	beam search	Viterbi	Viterbi	Viterbi

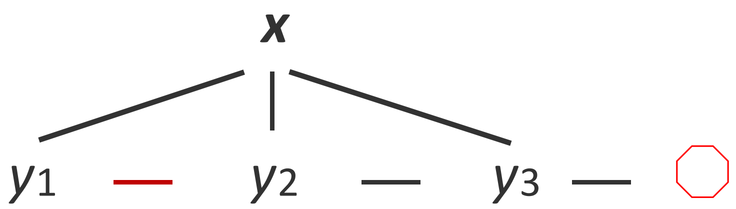


$s(\mathbf{x}, y_i, y_{i+1})$

model v.4

Recap: Sequence Labeling

Model	0	1	2	3	4
Score decomp.	$s(\boldsymbol{x}, i, y_i)$	$s(\boldsymbol{x}, i, \boldsymbol{y}_{1:i})$	emission/ transition	$s(x_i, y_i) + s(y_i, y_{i+1})$	$s(\boldsymbol{x}, i, y_i, y_{i+1})$
learn	SGD	?	count & normalize	?	?
decode	local	beam search	Viterbi	Viterbi	Viterbi



$s(\mathbf{x}, y_i, y_{i+1})$

model v.4

Recap: Inference (Decoding) -- Viterbi Algorithm

Let $\heartsuit_i(y)$ be the score of the best label sequence for $\boldsymbol{x}_{1:i}$ that ends in y. It is defined recursively:

$$
\begin{aligned}
\heartsuit_{n+1}(\octagon) &= \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \octagon) + \boxed{\heartsuit_n(y_n)} \\
\heartsuit_n(y) &= \max_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})} \\
\heartsuit_{n-1}(y) &= \max_{y_{n-2} \in \mathcal{L}} s(\boldsymbol{x}, n-2, y_{n-2}, y) + \boxed{\heartsuit_{n-2}(y_{n-2})} \\
&\vdots \\
\heartsuit_i(y) &= \max_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \boxed{\heartsuit_{i-1}(y_{i-1})} \\
&\vdots \\
\heartsuit_1(y) &= s(\boldsymbol{x}, 0, \bigcirc, y)
\end{aligned}
$$

Viterbi Procedure (Part I: Prefix Scores)

input sequence

$\mathcal{L}$		x_1	x_2	$\ldots$	x_n	
	ℓ_1					
	ℓ_2					
	$\vdots$					
	ℓ_L					

Viterbi Procedure (Part I: Prefix Scores)

input sequence

		x_1	x_2	$\ldots$	x_n	
	ℓ_1	$\heartsuit_1(\ell_1)$				
	ℓ_2	$\heartsuit_1(\ell_2)$				
$\mathcal{L}$	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$				
	$\bigcirc$					

$$\heartsuit_1(y) = s(\boldsymbol{x}, 0, \bigcirc, y)$$

Viterbi Procedure (Part I: Prefix Scores)

input sequence

		x_1	x_2	$\ldots$	x_n	
	ℓ_1	$\heartsuit_1(\ell_1)$	$\heartsuit_2(\ell_1)$			
$\mathcal{L}$	ℓ_2	$\heartsuit_1(\ell_2)$	$\heartsuit_2(\ell_2)$			
	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$	$\heartsuit_2(\ell_L)$			

$$\heartsuit_i(y) = \max_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \boxed{\heartsuit_{i-1}(y_{i-1})}$$

Viterbi Procedure (Part I: Prefix Scores)

input sequence

		x_1	x_2	$\ldots$	x_n	
	ℓ_1	$\heartsuit_1(\ell_1)$	$\heartsuit_2(\ell_1)$		$\heartsuit_n(\ell_1)$	
$\mathcal{L}$	ℓ_2	$\heartsuit_1(\ell_2)$	$\heartsuit_2(\ell_2)$		$\heartsuit_n(\ell_2)$	
	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$	$\heartsuit_2(\ell_L)$		$\heartsuit_n(\ell_L)$	
	$\octagon$					

$$\heartsuit_n(y) = \max_{y_{n-1} \in \mathcal{L}} s(\boldsymbol{x}, n-1, y_{n-1}, y) + \boxed{\heartsuit_{n-1}(y_{n-1})}$$

Viterbi Procedure (Part I: Prefix Scores)

input sequence

		x_1	x_2	$\ldots$	x_n	
$\mathcal{L}$	ℓ_1	$\heartsuit_1(\ell_1)$	$\heartsuit_2(\ell_1)$		$\heartsuit_n(\ell_1)$	
	ℓ_2	$\heartsuit_1(\ell_2)$	$\heartsuit_2(\ell_2)$		$\heartsuit_n(\ell_2)$	
	$\vdots$					
	ℓ_L	$\heartsuit_1(\ell_L)$	$\heartsuit_2(\ell_L)$		$\heartsuit_n(\ell_L)$	
	$\bigcirc$					$\heartsuit_{n+1}(\bigcirc)$

$$\heartsuit_{n+1}(\bigcirc) = \max_{y_n \in \mathcal{L}} s(\boldsymbol{x}, n, y_n, \bigcirc) + \boxed{\heartsuit_n(y_n)}$$

Recap: Inference (Decoding) -- Viterbi Algorithm

Input: scores $s(\boldsymbol{x}, i, y, y')$, for all $i \in \{0, \ldots, n\}$, $y, y' \in \mathcal{L}$

Output: $\hat{\boldsymbol{y}}$

1. Base case: $\heartsuit_1(y) = s(\boldsymbol{x}, 0, \bigcirc, y)$
2. For $i \in \{2, \ldots, n+1\}$:
   - Solve for $\heartsuit_i(*)$ and $\mathrm{bp}_i(*)$.

$$\heartsuit_i(y) = \max_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \heartsuit_{i-1}(y_{i-1}),$$

$$\mathrm{bp}_i(y) = \operatorname*{argmax}_{y_{i-1} \in \mathcal{L}} s(\boldsymbol{x}, i-1, y_{i-1}, y) + \heartsuit_{i-1}(y_{i-1})$$

   (At $n+1$ we're only interested in $y = \octagon$.)

3. $\hat{y}_{i+1} \leftarrow \octagon$
4. For $i \in \{n, \ldots, 1\}$:
   - $\hat{y}_i \leftarrow \mathrm{bp}_{i+1}(\hat{y}_{i+1})$

Recap: Inference (Decoding) -- Viterbi Algorithm

- Viterbi Asymptotics

input sequence

labels in $\mathcal{L}$

	x_1	x_2	$\ldots$	x_n
ℓ_1				
ℓ_2				
$\vdots$				
ℓ_L				

Space: need to store s, and fill in the cells above. $O(nL^2)$ for s (in the most general case, often less), $O(nL)$ for cells

Runtime: each cell requires an "argmax." $O(nL^2)$

Back to *s*

We haven't said much about the function that scores candidate label pairs at different positions, $s(\boldsymbol{x}, i, y, y')$.

This function is very important; two common choices are:

- Expert-designed, task-specific features $\mathbf{f}(\boldsymbol{x}, i, y, y')$ and weights $\boldsymbol{\theta}$
- A neural network that encodes x_i in context, y_i, and y_{i+1} and gives back a goodness score

Either way, let $\boldsymbol{\theta}$ denote the parameters of s. From now on, we'll use $s(\boldsymbol{x}, i, y, y'; \boldsymbol{\theta})$ and $\mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})$ to emphasize that "s" is a function of parameters $\boldsymbol{\theta}$ we need to estimate.

Probabilistic View of Learning

As we've done before, we start with the principle of maximum likelihood to estimate $\boldsymbol{\theta}$:

$$\begin{aligned}
\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}\in\mathbb{R}^d} \prod_{t=1}^{T} p(\boldsymbol{Y}_t = \boldsymbol{y} \mid \boldsymbol{X}_t = \boldsymbol{x}\,; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}\in\mathbb{R}^d} \sum_{t=1}^{T} \log p(\boldsymbol{Y}_t = \boldsymbol{y} \mid \boldsymbol{X}_t = \boldsymbol{x}\,; \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \sum_{t=1}^{T} \underbrace{-\log p(\boldsymbol{Y}_t = \boldsymbol{y} \mid \boldsymbol{X}_t = \boldsymbol{x}\,; \boldsymbol{\theta})}_{\text{sometimes called “log loss” or “cross entropy”}}
\end{aligned}$$

Next, we'll drill down into "$p(\boldsymbol{Y} = \boldsymbol{y}_t \mid \boldsymbol{X} = \boldsymbol{x}_t; \boldsymbol{\theta})$."

Conditional Random Fields (CRFs)

Lafferty et al. (2001)

CRFs are a tremendously influential model that generalizes multinomial logistic regression to structured outputs like sequences.

$$
\begin{aligned}
p_{\mathrm{CRF}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) &= \frac{\exp \mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}{Z(\boldsymbol{x}; \boldsymbol{\theta})} \\
Z(\boldsymbol{x}; \boldsymbol{\theta}) &= \sum_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} \exp \mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}'; \boldsymbol{\theta}) \\
-\log p_{\mathrm{CRF}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) &= -\underbrace{\mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}_{\text{“hope”}} + \underbrace{\log Z(\boldsymbol{x}; \boldsymbol{\theta})}_{\text{“fear”}}
\end{aligned}
$$

So, our “CRF”:

- Uses Viterbi for decoding (our v. 4 sequence labeler)
- Trains parameters to maximize likelihood (like MLR and NNs)

Sequence-Level Log Loss

Here's the maximum likelihood learning problem (equivalently, sequence-level log loss):

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta} \in \mathbb{R}^d} \sum_{t=1}^{T} -\mathrm{Score}(\boldsymbol{x}_t, \boldsymbol{y}_t; \boldsymbol{\theta}) + \log Z(\boldsymbol{x}_t; \boldsymbol{\theta})$$

If we can calculate and differentiate (w.r.t. $\boldsymbol{\theta}$) the Score and Z functions, we can use SGD to learn.

Calculating $Z(\boldsymbol{x}; \boldsymbol{\theta})$

Good news! The algorithm that gives us Z is *almost exactly like* the Viterbi algorithm.

Forward algorithm: sums the $\exp \text{Score}$ values for all label sequences, given $\boldsymbol{x}$, in the same asymptotic time and space as Viterbi.

Let $\alpha_i(y)$ be the sum of all (exponentiated) scores of label prefixes of length i, ending in y.

Some Algebra

Given the decomposition

$$\mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \sum_{i=0}^{n} s(\boldsymbol{x}, i, y_i, y_{i+1}; \boldsymbol{\theta}),$$

it holds that

$$\exp \mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \prod_{i=0}^{n} e^{s(\boldsymbol{x}, i, y_i, y_{i+1}; \boldsymbol{\theta})},$$

and therefore

$$Z(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{\boldsymbol{y'} \in \mathcal{Y}(\boldsymbol{x})} \prod_{i=0}^{n} e^{s(\boldsymbol{x}, i, y'_i, y'_{i+1}; \boldsymbol{\theta})}$$

Forward Algorithm

Input: scores $s(\boldsymbol{x}, i, y, y'; \boldsymbol{\theta})$, for all $i \in \{0, \ldots, n\}$, $y, y' \in \mathcal{L}$

Output: $Z(\boldsymbol{x}; \boldsymbol{\theta})$

1. Base case: $\alpha_1(y) = e^{s(\boldsymbol{x}, 0, \bigcirc, y; \boldsymbol{\theta})}$
2. For $i \in \{2, \ldots, n+1\}$:
   - Solve for $\alpha_i(*)$.

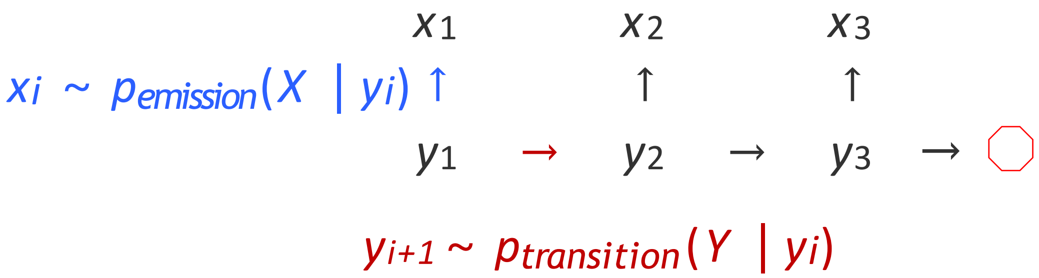
$$\alpha_i(y) = \sum_{y_{i-1} \in \mathcal{L}} e^{s(\boldsymbol{x}, i-1, y_{i-1}, y; \boldsymbol{\theta})} \times \alpha_{i-1}(y_{i-1})$$

   (At $n+1$ we're only interested in $y = \octagon$.)

3. Return $\alpha_{n+1}(\octagon)$, which is equal to $Z(\boldsymbol{x}; \boldsymbol{\theta})$.

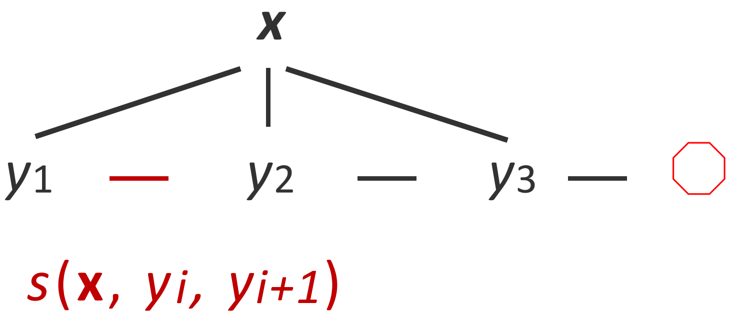
Key Takeaways

- HMM (generative model) - models joint distribution



$$p(\boldsymbol{x}, \boldsymbol{y}) = \prod_{i=1}^{N} p(y_i | y_{i-1}) p(x_i | y_i)$$

- CRF (discriminative model) - directly models conditional distribution



$$p_{\mathrm{CRF}}(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta}) = \frac{\exp \mathrm{Score}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta})}{Z(\boldsymbol{x}; \boldsymbol{\theta})}$$

$$\mathrm{Score}(\mathbf{x},\ \mathbf{y}) = \sum_{i=0}^{n} s(\mathbf{x}, i, y_i, y_{i+1})$$

Key Takeaways

- HMM (generative model) - models joint distribution
- CRF (discriminative model) - directly models conditional distribution
- Inference: Viterbi algorithm
- Learning

Parsing

[Slides adapted from UW CSE 447 by Noah Smith; UCB Info 159/259 by David Bamman]

Motivation

As data, we tend to view natural language text as sequences (of words, wordpieces, or characters, depending on the NLP application).

But language obeys implicit rules of grammar, and it carries meaning.

- It's helpful to consider an analogy to programming languages, which have *syntax* and *semantics*; well-formed programs can be compiled and executed to carry out a task.
- Well-formed natural language strings can be understood by others.

Computational models that analyze natural language syntax and semantics typically map into **structures** like trees, graphs, and more.

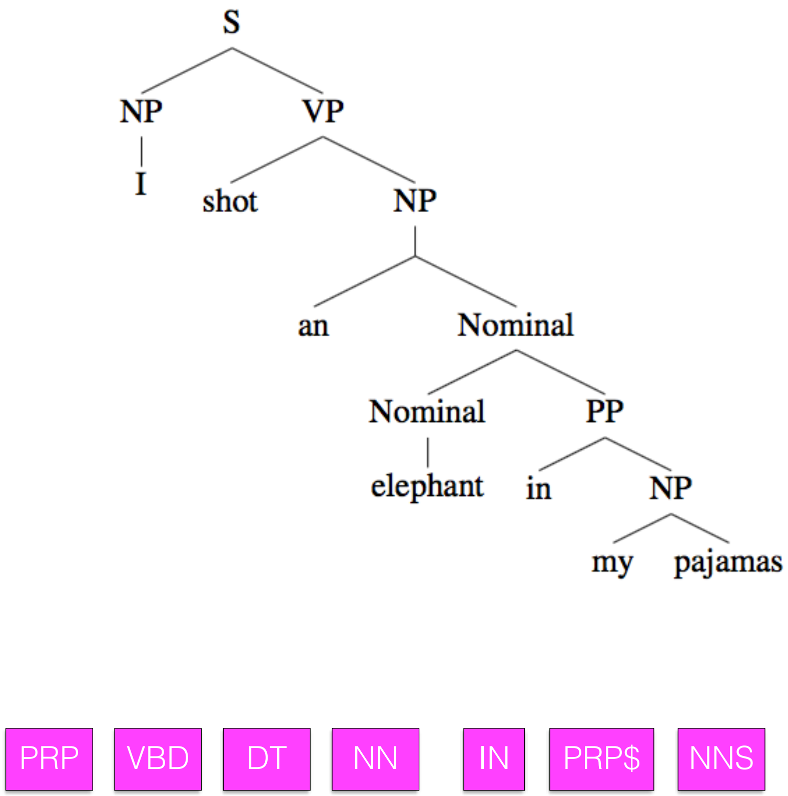
Linguistic Analysis

- **Syntax**: rules governing grammaticality or well-formedness of strings, relative to a language
- **Semantics**: how the meaning of an utterance is constructed, grounded in "the world" (or a proxy to the world)
- **Pragmatics**: the intended meaning by a speaker, in a given social context

Each has many theories, and none of them is complete!

Syntax

- With syntax, we're moving from labels for discrete items — documents (sentiment analysis), tokens (POS tagging, NER) — to the structure between items.



I shot an elephant in my pajamas

Syntax

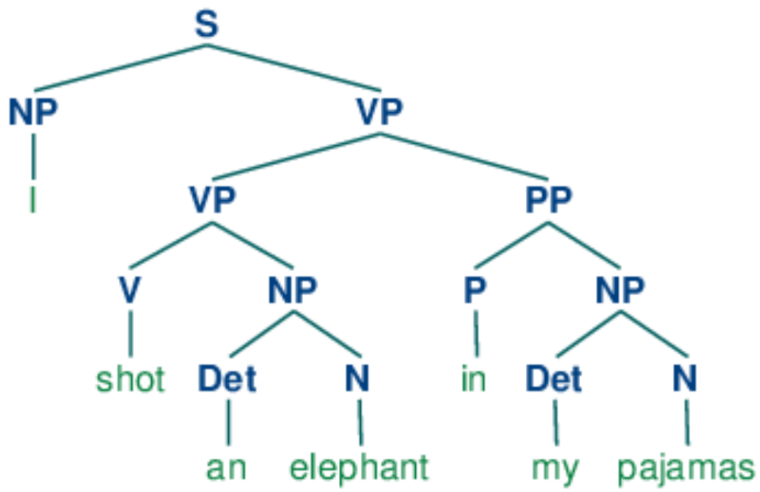
- With syntax, we're moving from labels for discrete items — documents (sentiment analysis), tokens (POS tagging, NER) — to the structure between items.

- Syntax is fundamentally about the hierarchical structure of language and (in some theories) which sentences are grammatical in a language

words → phrases → clauses → sentences

Formalisms

Phrase structure grammar
(Chomsky 1957)



Dependency grammar
(Mel’čuk 1988; Tesnière 1959; Pāṇini)

Constituency

- Groups of words ("constituents") behave as single units
  - Noun Phrases: groups of tokens that act Like nouns

▶ Harry the Horse

▶ the Broadway coppers

▶ they

▶ a high-class spot such as Mindy's

▶ the reason he comes into the Hot Box

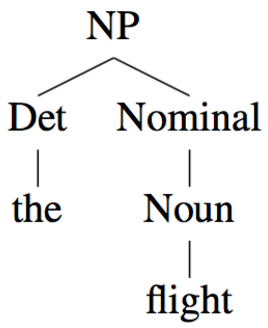
▶ three parties from Brooklyn

Constituency

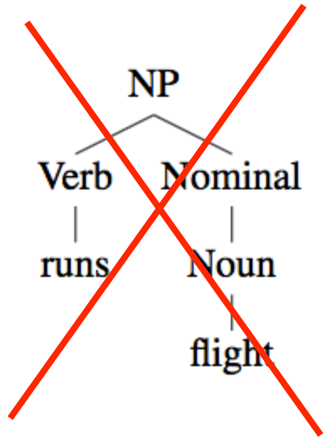
- Groups of words ("constituents") behave as single units
  - Noun Phrases: groups of tokens that act Like nouns
- Linguists characterize constituents in a number of ways, including:
  - where they occur (e.g., "NPs can occur before verbs")
  - where they can *move* in variations of a sentence
    - On September 17th, I'd like to fly from Atlanta to Denver
    - I'd like to fly on September 17th from Atlanta to Denver
    - I'd like to fly from Atlanta to Denver on September 17th
  - what parts can move and what parts can't
    - *On September I'd like to fly 17th from Atlanta to Denver
  - what they can be conjoined with
    - I'd like to fly from Atlanta to Denver on September 17th and in the morning

Context-Free Grammar (CFG)

- Take constituents to be the main building block of natural language syntax, we can attempt to formalize what makes a string grammatical in a language.
- A CFG gives a formal way to define what meaningful constituents are and exactly how a constituent is formed out of other constituents (or words). It defines valid structure in a language.



NP → Det Nominal



NP → Verb Nominal

Context-Free Grammar (CFG)

A **context-free grammar** consists of:

- A finite set of nonterminal symbols $\mathcal{N}$ (sometimes called "categories")
  - A start symbol $S \in \mathcal{N}$
- A finite alphabet Σ, called "terminal" symbols, distinct from $\mathcal{N}$
- Production rule set $\mathcal{R}$, each of the form "$N \rightarrow \boldsymbol{\alpha}$" where
  - The lefthand side N is a nonterminal from $\mathcal{N}$
  - The righthand side $\boldsymbol{\alpha}$ is a sequence of zero or more terminals and/or nonterminals: $\boldsymbol{\alpha} \in (\mathcal{N} \cup \Sigma)^*$
    - Special case: **Chomsky normal form** constrains $\boldsymbol{\alpha}$ to be either a single terminal symbol or two nonterminals

An Example CFG for a Tiny Bit of English

From Jurafsky and Martin (forthcoming)

S $\rightarrow$ NP VP
S $\rightarrow$ Aux NP VP
S $\rightarrow$ VP
NP $\rightarrow$ Pronoun
NP $\rightarrow$ Proper-Noun
NP $\rightarrow$ Det Nominal
Nominal $\rightarrow$ Noun
Nominal $\rightarrow$ Nominal Noun
Nominal $\rightarrow$ Nominal PP
VP $\rightarrow$ Verb
VP $\rightarrow$ Verb NP
VP $\rightarrow$ Verb NP PP
VP $\rightarrow$ Verb PP
VP $\rightarrow$ VP PP
PP $\rightarrow$ Preposition NP

Det $\rightarrow$ that $|$ this $|$ a
Noun $\rightarrow$ book $|$ flight $|$ meal $|$ money
Verb $\rightarrow$ book $|$ include $|$ prefer
Pronoun $\rightarrow$ I $|$ she $|$ me
Proper-Noun $\rightarrow$ Houston $|$ NWA
Aux $\rightarrow$ does
Preposition $\rightarrow$ from $|$ to $|$ on $|$ near
$|$ through

"Lexicon"

This term is used in NLP to refer to an object that associates information with words.

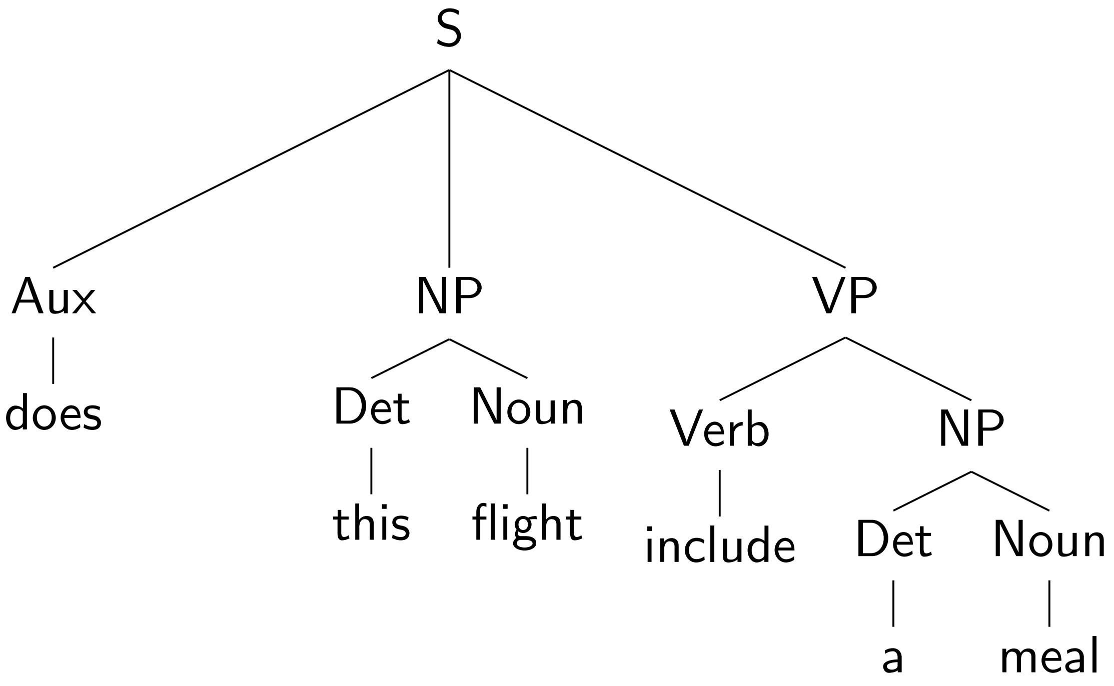
In a CFG, the "lexicon rules" are the rules that map a nonterminal (usually a part of speech) to a single word.

Derivation

- Given a CFG, a derivation is the sequence of productions used to generate a string of words (e.g., a sentence), often visualized as a parse tree.

- Language: the formal language defined by a CFG is the set of strings derivable from *S* (start symbol)

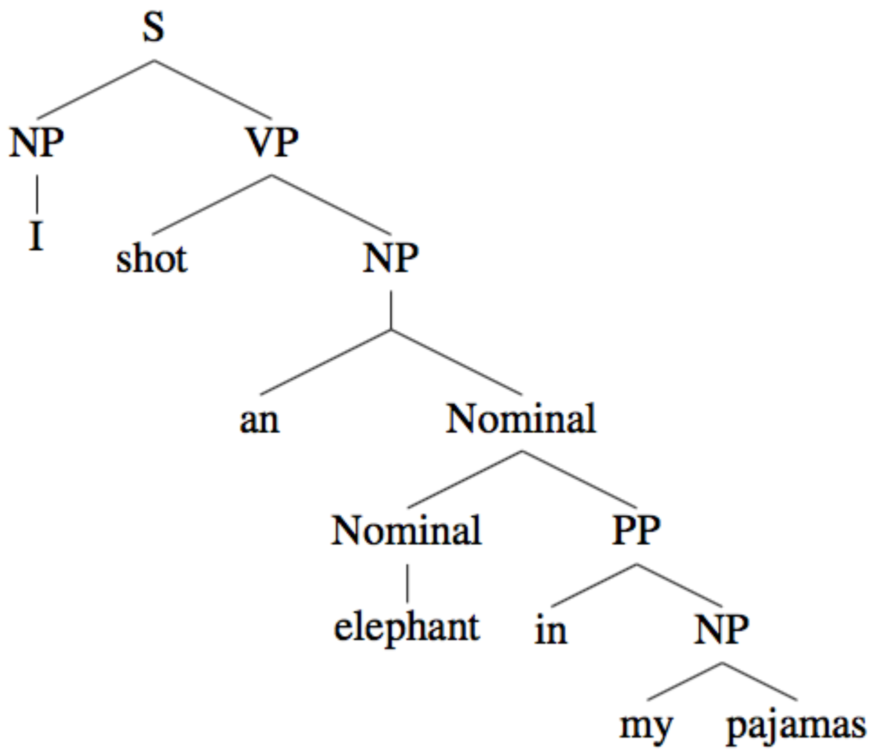
Example Derivation



The phrase-structure tree represents both the syntactic structure of the sentence and the **derivation** of the sentence under the grammar. E.g., VP corresponds to the rule VP $\rightarrow$ Verb NP.

Verb NP

Example Derivation



Every internal node is a phrase

- my pajamas
- in my pajamas
- elephant in my pajamas
- an elephant in my pajamas
- shot an elephant in my pajamas
- I shot an elephant in my pajamas

Each phrase could be replaced by another of the same type of constituent

Where do natural language CFGs come from?

Building a CFG for a natural language by hand is really hard (Jurafsky and Martin, forthcoming, chapter 10).

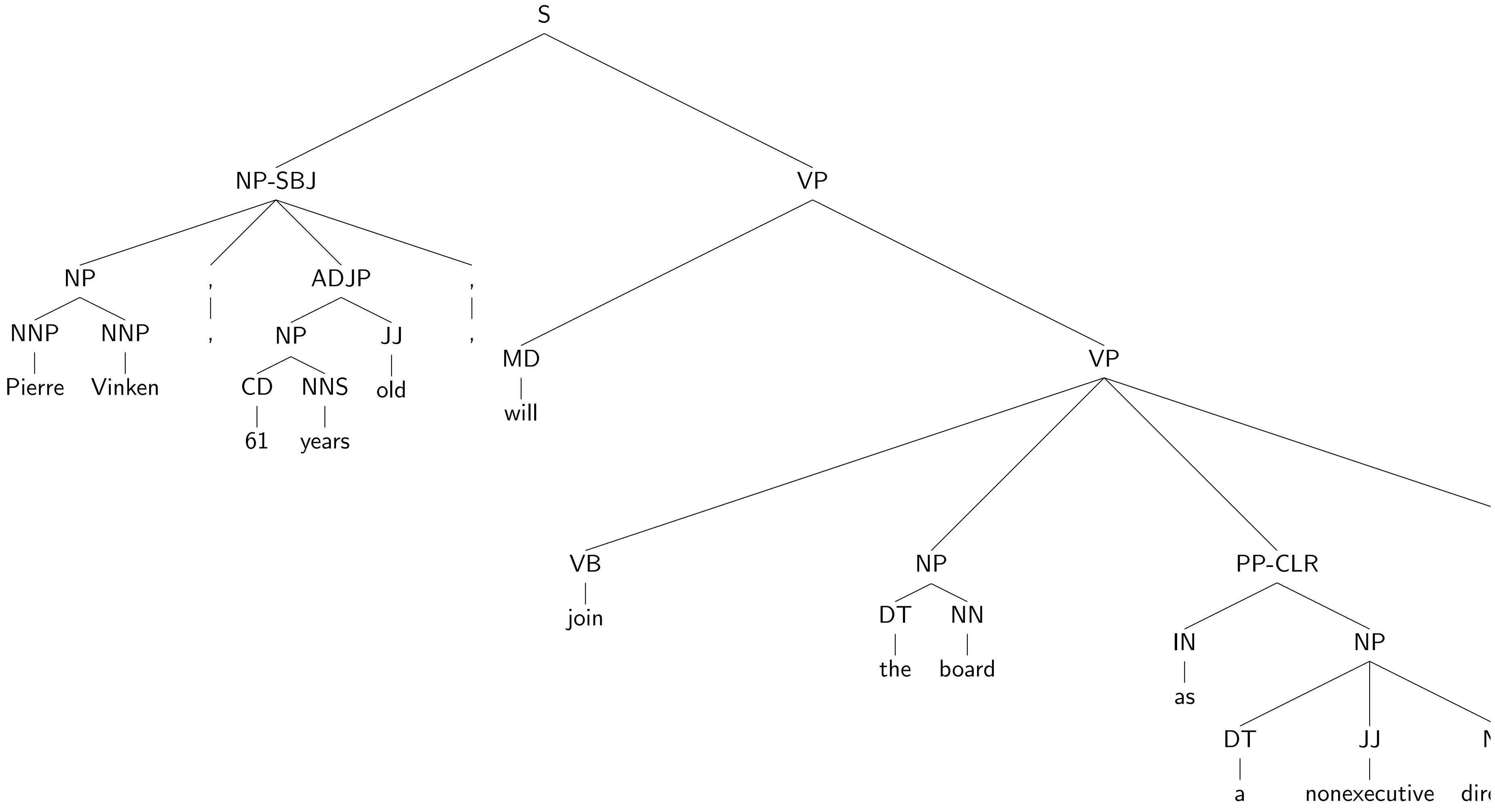
- Need lots of categories to make sure all and only grammatical sentences are included.
- Categories tend to start exploding combinatorially.
- Alternative grammar formalisms are typically used for manual grammar construction; these are often based on constraints and a powerful algorithmic tool called *unification*.

A data-driven approach:

1. Build a corpus of annotated sentences, called a **treebank**. (e.g., the Penn Treebank, Marcus et al., 1993.)
2. Extract rules from the treebank.
3. Optionally, use statistical models to generalize the rules.

collections of sentences annotated with syntactic structure

Example from the Penn Treebank



Some Penn Treebank Rules with Counts

40717 PP $\rightarrow$ IN NP
33803 S $\rightarrow$ NP-SBJ VP
22513 NP-SBJ $\rightarrow$ -NONE-
21877 NP $\rightarrow$ NP PP
20740 NP $\rightarrow$ DT NN
14153 S $\rightarrow$ NP-SBJ VP .
12922 VP $\rightarrow$ TO VP
11881 PP-LOC $\rightarrow$ IN NP
11467 NP-SBJ $\rightarrow$ PRP
11378 NP $\rightarrow$ -NONE-
11291 NP $\rightarrow$ NN
...
989 VP $\rightarrow$ VBG S
985 NP-SBJ $\rightarrow$ NN
983 PP-MNR $\rightarrow$ IN NP
983 NP-SBJ $\rightarrow$ DT
969 VP $\rightarrow$ VBN VP

100 VP $\rightarrow$ VBD PP-PRD
100 PRN $\rightarrow$: NP :
100 NP $\rightarrow$ DT JJS
100 NP-CLR $\rightarrow$ NN
99 NP-SBJ-1 $\rightarrow$ DT NNP
98 VP $\rightarrow$ VBN NP PP-DIR
98 VP $\rightarrow$ VBD PP-TMP
98 PP-TMP $\rightarrow$ VBG NP
97 VP $\rightarrow$ VBD ADVP-TMP VP
...
10 WHNP-1 $\rightarrow$ WRB JJ
10 VP $\rightarrow$ VP CC VP PP-TMP
10 VP $\rightarrow$ VP CC VP
ADVP-MNR
10 VP $\rightarrow$ VBZ S , SBAR-ADV
10 VP $\rightarrow$ VBZ S ADVP-TMP

Penn Treebank Rules: Statistics

32,728 rules in the training section (not including 52,257 lexicon rules)

4,021 rules in the development section

overlap: 3,128

(Phrase-Structure) Recognition and Parsing

Given a CFG $(\mathcal{N}, S, \Sigma, \mathcal{R})$ and a sentence $\boldsymbol{x}$, the **recognition** problem is:

Is $\boldsymbol{x}$ in the language of the CFG?

The proof is a derivation of $\boldsymbol{x}$ using the rules $\mathcal{R}$.

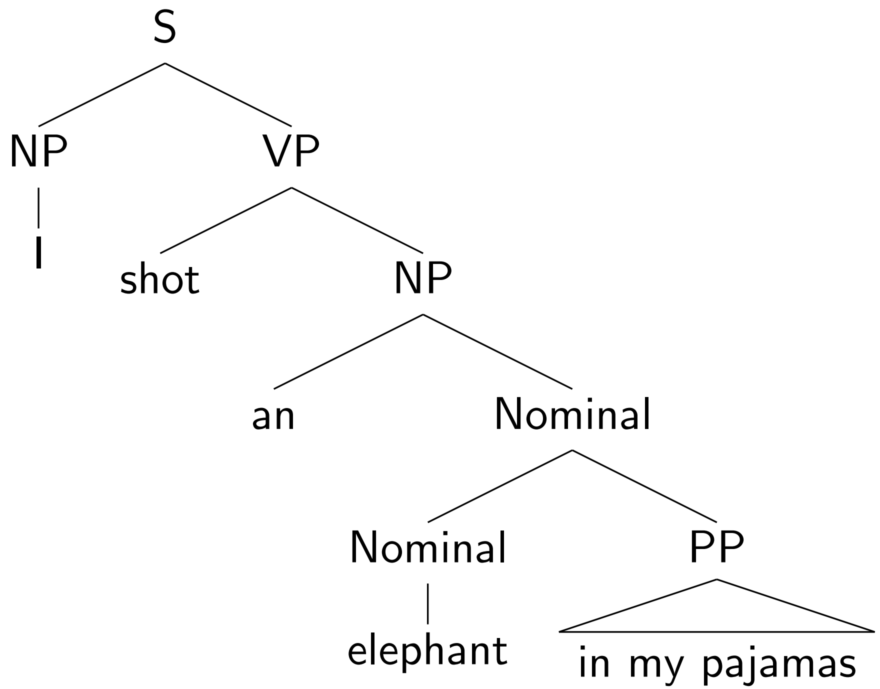
Related problem: **parsing**:

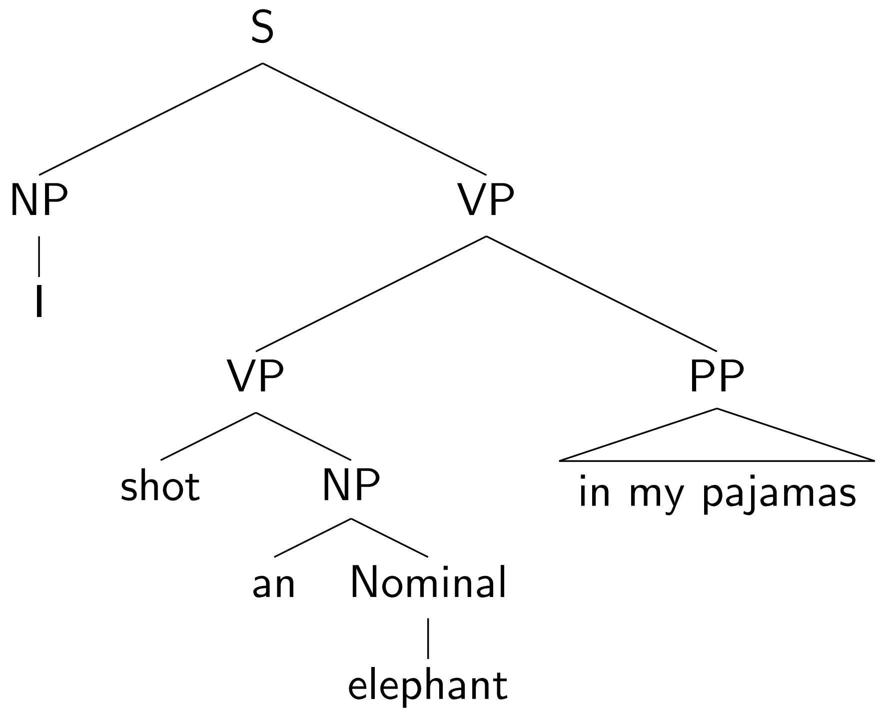
Show one or more derivations for $\boldsymbol{x}$, using $\mathcal{R}$.

With reasonable grammars, the number of parses is exponential in $|\boldsymbol{x}|$.

Syntactic Ambiguity

Ambiguity is the most serious problem faced by syntactic parsers



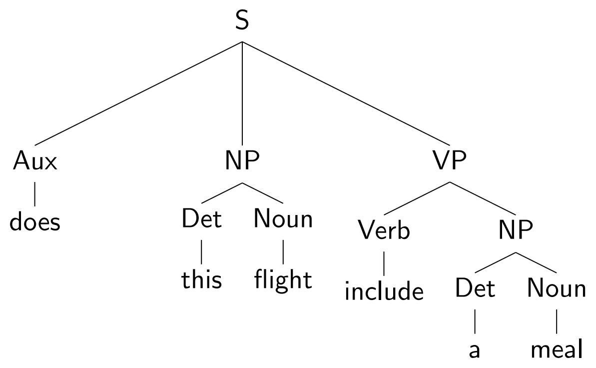


Parser Evaluation

Represent a parse tree as a collection of tuples $\langle\langle \ell_1, i_1, j_1\rangle, \langle \ell_2, i_2, j_2\rangle, \ldots, \langle \ell_n, i_n, j_n\rangle\rangle$, where

- ℓ_k is the nonterminal labeling the kth phrase
- i_k is the index of the first word in the kth phrase
- j_k is the index of the last word in the kth phrase

Example:

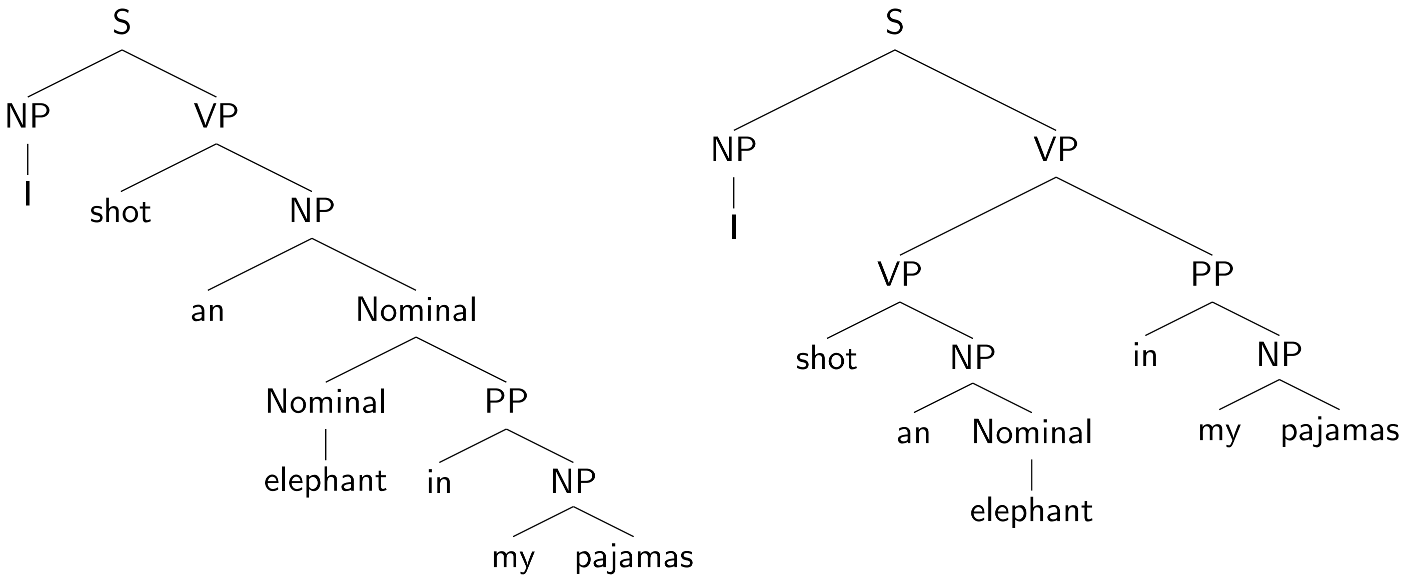


$$\longrightarrow \left\langle \begin{array}{ll} \langle \mathsf{S}, 1, 6\rangle, & \langle \mathsf{NP}, 2, 3\rangle, \\ \langle \mathsf{VP}, 4, 6\rangle, & \langle \mathsf{NP}, 5, 6\rangle \end{array} \right\rangle$$

Convert gold-standard tree and system hypothesized tree into this representation, then estimate precision, recall, and F_1.

Parser Evaluation

Tree Comparison Example



$$\underbrace{\left\langle \begin{array}{c} \langle \mathsf{NP}, 3, 7\rangle, \\ \langle \mathsf{Nominal}, 4, 7\rangle \end{array} \right\rangle}_{\text{only in left tree}} \underbrace{\left\langle \begin{array}{c} \langle \mathsf{NP}, 1, 1\rangle \\ \langle \mathsf{S}, 1, 7\rangle, \langle \mathsf{VP}, 2, 7\rangle, \\ \langle \mathsf{PP}, 5, 7\rangle, \langle \mathsf{NP}, 6, 7\rangle \\ \langle \mathsf{Nominal}, 4, 4\rangle \end{array} \right\rangle}_{\text{in both trees}} \underbrace{\left\langle \begin{array}{c} \langle \mathsf{VP}, 2, 4\rangle, \\ \langle \mathsf{NP}, 3, 4\rangle \end{array} \right\rangle}_{\text{only in right tree}}$$

Two Views of Parsing

1. Incremental search: the state of the search is the partial structure built so far; each action incrementally extends the tree.
   - Often **greedy**, with a statistical classifier deciding what action to take in every state.
2. Discrete optimization: define a scoring function and seek the tree with the highest score.

Probabilistic Context-Free Grammar (PCFG)

- A basic CFG allows us to check whether a sentence is grammatical in the language it defines
- Binary decision: a sentence is either in the language (a series of productions yields the words we see) or it is not.

- PCFG: each production is also associated with a probability.
- This lets us calculate the probability of a parse for a given sentence
  - For a given parse tree T for sentence S comprised of n rules (each $A \rightarrow \beta$):

$$P(T, S) = \prod_{i}^{n} P(\beta \mid A)$$

Probabilistic Context-Free Grammar (PCFG)

A **probabilistic context-free grammar** consists of:

- A finite set of nonterminal symbols $\mathcal{N}$
  - A start symbol $S \in \mathcal{N}$
- A finite alphabet Σ, called "terminal" symbols, distinct from $\mathcal{N}$
- Production rule set $\mathcal{R}$, each of the form "$N \rightarrow \boldsymbol{\alpha}$" where
  - The lefthand side N is a nonterminal from $\mathcal{N}$
  - The righthand side $\boldsymbol{\alpha}$ is a sequence of zero or more terminals and/or nonterminals: $\boldsymbol{\alpha} \in (\mathcal{N} \cup \Sigma)^*$
    - Special case: **Chomsky normal form** constrains $\boldsymbol{\alpha}$ to be either a single terminal symbol or two nonterminals
- For each $N \in \mathcal{N}$, a probability distribution over the rules where N is the lefthand side, $p(* \mid N)$.

PCFG Scores Trees

We can write the parsing problem as finding the best-scoring tree:

$$\hat{\boldsymbol{t}} = \operatorname*{argmax}_{\boldsymbol{t} \in \mathcal{T}_{\boldsymbol{x}}} \mathrm{Score}(\boldsymbol{t})$$

PCFGs view each tree $\boldsymbol{t}$ as a "bag of rules" (from $\mathcal{R}$), and define:

$$\begin{aligned} \mathrm{Score}(\boldsymbol{t}) &= p(\boldsymbol{t}) \\ &= \prod_{(N \to \alpha) \in \mathcal{R}} p(\alpha \mid N)^{\mathrm{count}(N \to \alpha; \boldsymbol{t})} \end{aligned}$$

PCFG Example

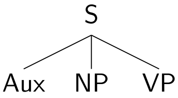
S

Write down the start symbol. Here: S

Probability:

$$1$$

PCFG Example

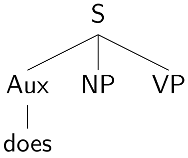


Choose a rule from the “S” distribution. Here: S $\rightarrow$ Aux NP VP

Probability:

$$p(\text{Aux NP VP} \mid \text{S})$$

PCFG Example

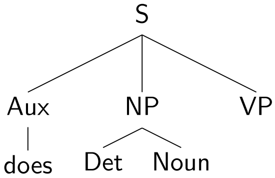


Choose a rule from the "Aux" distribution. Here: Aux $\rightarrow$ does

Probability:

$$p(\text{Aux NP VP} \mid \text{S}) \cdot p(\text{does} \mid \text{Aux})$$

PCFG Example

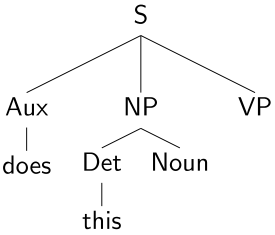


Choose a rule from the “NP” distribution. Here: NP $\rightarrow$ Det Noun

Probability:

$$p(\text{Aux NP VP} \mid \text{S}) \cdot p(\text{does} \mid \text{Aux}) \cdot p(\text{Det Noun} \mid \text{NP})$$

PCFG Example

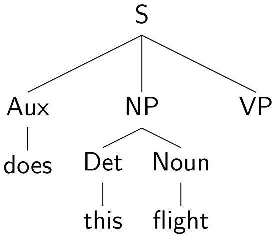


Choose a rule from the “Det” distribution. Here: Det $\rightarrow$ this

Probability:

$p(\text{Aux NP VP} \mid \text{S}) \cdot p(\text{does} \mid \text{Aux}) \cdot p(\text{Det Noun} \mid \text{NP}) \cdot p(\text{this} \mid \text{Det})$

PCFG Example

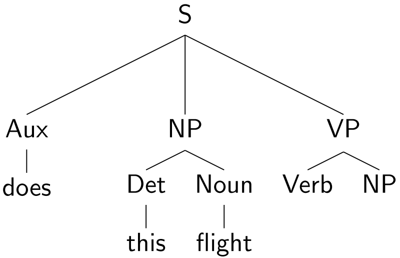


Choose a rule from the "Noun" distribution. Here: Noun $\rightarrow$ flight

Probability:

$p(\text{Aux NP VP} \mid \text{S}) \cdot p(\text{does} \mid \text{Aux}) \cdot p(\text{Det Noun} \mid \text{NP}) \cdot p(\text{this} \mid \text{Det})$
$\cdot\, p(\text{flight} \mid \text{Noun})$

PCFG Example

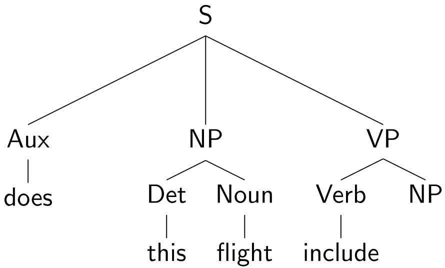


Choose a rule from the "VP" distribution. Here: VP $\rightarrow$ Verb NP

Probability:

$p(\text{Aux NP VP} \mid \text{S}) \cdot p(\text{does} \mid \text{Aux}) \cdot p(\text{Det Noun} \mid \text{NP}) \cdot p(\text{this} \mid \text{Det}) \cdot p(\text{flight} \mid \text{Noun}) \cdot p(\text{Verb NP} \mid \text{VP})$

PCFG Example

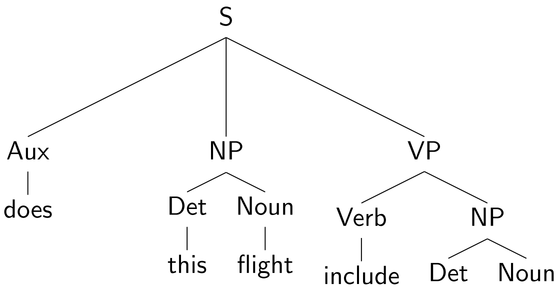


Choose a rule from the "Verb" distribution. Here: Verb $\rightarrow$ include

Probability:

p(Aux NP VP | S) $\cdot$ p(does | Aux) $\cdot$ p(Det Noun | NP) $\cdot$ p(this | Det) $\cdot$ p(flight | Noun) $\cdot$ p(Verb NP | VP) $\cdot$ p(include | Verb)

PCFG Example

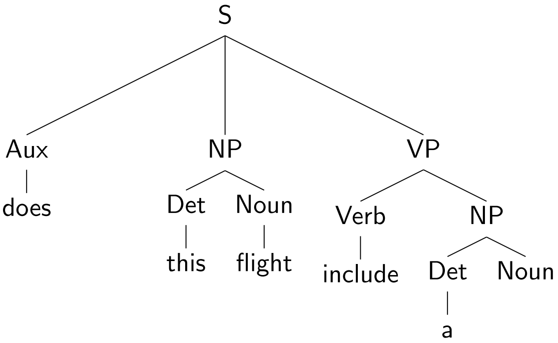


Choose a rule from the “NP” distribution. Here: NP $\rightarrow$ Det Noun

Probability:

$p(\text{Aux NP VP} \mid \text{S}) \cdot p(\text{does} \mid \text{Aux}) \cdot p(\text{Det Noun} \mid \text{NP}) \cdot p(\text{this} \mid \text{Det})$
$\cdot\, p(\text{flight} \mid \text{Noun}) \cdot p(\text{Verb NP} \mid \text{VP}) \cdot p(\text{include} \mid \text{Verb})$
$\cdot\, p(\text{Det Noun} \mid \text{NP})$

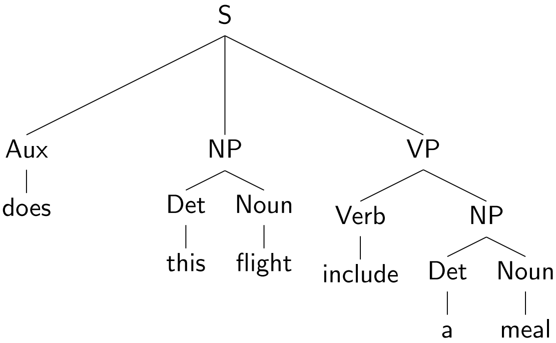
PCFG Example



Choose a rule from the "Det" distribution. Here: Det $\rightarrow$ a
Probability:

$$p(\text{Aux NP VP} \mid \text{S}) \cdot p(\text{does} \mid \text{Aux}) \cdot p(\text{Det Noun} \mid \text{NP}) \cdot p(\text{this} \mid \text{Det})$$
$$\cdot\, p(\text{flight} \mid \text{Noun}) \cdot p(\text{Verb NP} \mid \text{VP}) \cdot p(\text{include} \mid \text{Verb})$$
$$\cdot\, p(\text{Det Noun} \mid \text{NP}) \cdot p(\text{a} \mid \text{Det})$$

PCFG Example



Choose a rule from the "Noun" distribution. Here: Noun $\rightarrow$ meal
Probability:

$$p(\text{Aux NP VP} \mid \text{S}) \cdot p(\text{does} \mid \text{Aux}) \cdot p(\text{Det Noun} \mid \text{NP}) \cdot p(\text{this} \mid \text{Det})$$
$$\cdot\, p(\text{flight} \mid \text{Noun}) \cdot p(\text{Verb NP} \mid \text{VP}) \cdot p(\text{include} \mid \text{Verb})$$
$$\cdot\, p(\text{Det Noun} \mid \text{NP}) \cdot p(\text{a} \mid \text{Det}) \cdot p(\text{meal} \mid \text{Noun})$$

Parsing with PCFGs

- How to set the probabilities p(righthand side | lefthand side)?
  - Counting / Learning (we won't discuss in this class due to time limit)

- How to decode/parse?

Probabilistic CKY (Cocke-Kasami-Younger)

(Cocke and Schwartz, 1970; Kasami, 1965; Younger, 1967)

Input:

- a PCFG $(\mathcal{N}, S, \Sigma, \mathcal{R}, p(* \mid *))$, in **Chomsky normal form**
- a sentence $\boldsymbol{x}$ (let n be its length)

Output: If $\boldsymbol{x}$ is in the language of the grammar.

$$\operatorname*{argmax}_{\boldsymbol{t} \in \mathcal{T}_{\boldsymbol{x}}} \log p(\boldsymbol{t});$$

undefined if not.

Probabilistic CKY

Probabilistic CKY is closely related to the Viterbi algorithm; it is a dynamic programming algorithm.
The recurrence is defined around

$$\heartsuit_{i:j}(N),$$

which will store the best score (log probability) found (so far) for constructing an N-rooted constituent that spans $\langle x_i, \ldots, x_j \rangle$.

In Viterbi, we used conditional independence to collapse all prefix label sequences that ended in the same label into one stored item; here we collapse all trees spanning words i to j with the same root into a single item.

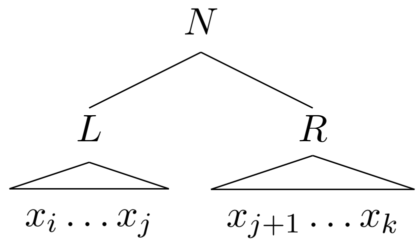
Probabilistic CKY

Base case: for $i \in \{1, \ldots, n\}$ and for each $N \in \mathcal{N}$:

$$\heartsuit_{i:i}(N) = \log p(x_i \mid N)$$

For each i, k such that $1 \leq i < k \leq n$ and each $N \in \mathcal{N}$:

$$\heartsuit_{i:k}(N) = \max_{L,R \in \mathcal{N}, j \in \{i,\ldots,k-1\}} \log p(L\ R \mid N) + \heartsuit_{i:j}(L) + \heartsuit_{(j+1):k}(R)$$

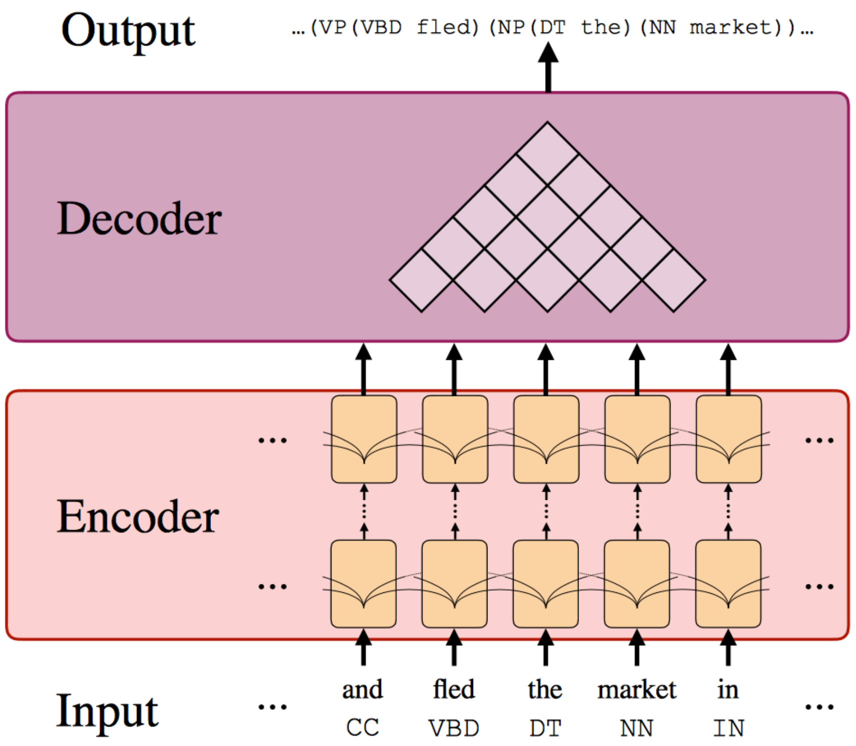


Solution:

$$\heartsuit_{1:n}(S) = \max_{\boldsymbol{t} \in \mathcal{T}_{\boldsymbol{x}}} \log p(\boldsymbol{t})$$

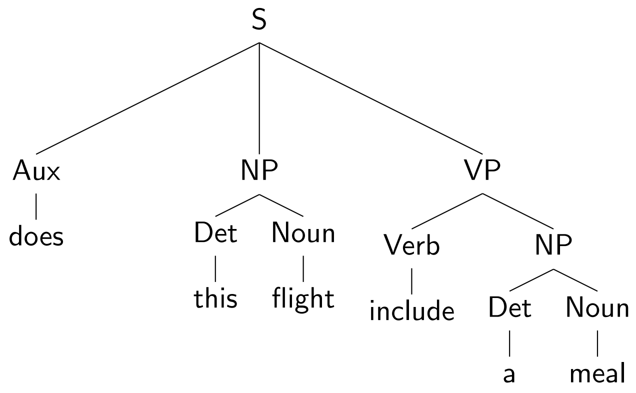
Neural Parsing

- Kitaev and Klein (2018), “Constituency Parsing with a Self-Attentive Encoder”
- Neural model (attention encoder) generates representations of each token in a sentence)
- Learned scoring s(i,j,k) function for each span from token i to token j with label k
- CKY for decoding to find the best tree through this space.



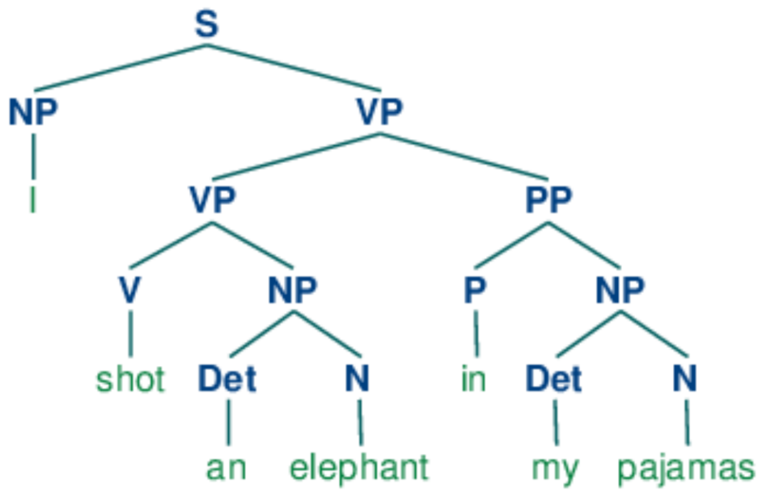
Summary so far

- Constituents: groups of words behave as single units
- Context-Free Grammar (CFG)
  - A CFG gives a formal way to define a valid structure in a language
- Probabilistic Context-Free Grammar (PCFG)
  - Each production is also associated with a probability
- Parsing:
  - Show one or more derivations for a sentence, using the grammar
  - (Probabilistic) CKY



Formalisms

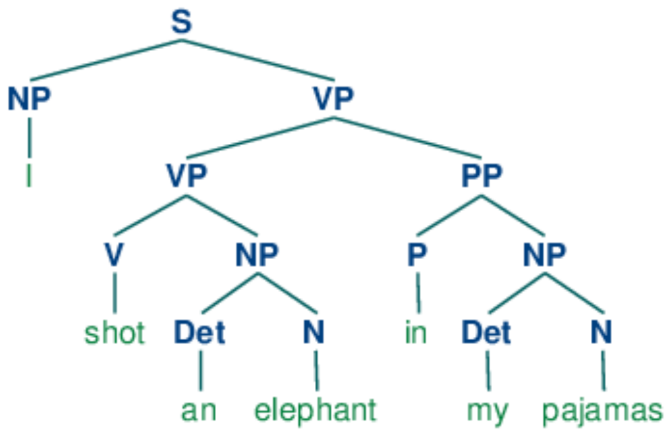
Phrase structure grammar
(Chomsky 1957)



Dependency grammar
(Mel’čuk 1988; Tesnière 1959; Pāṇini)

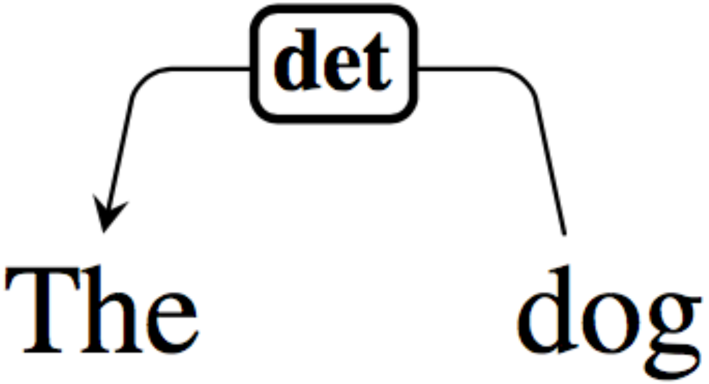
Dependency syntax

A different family of theories of syntax focuses on dependencies between words



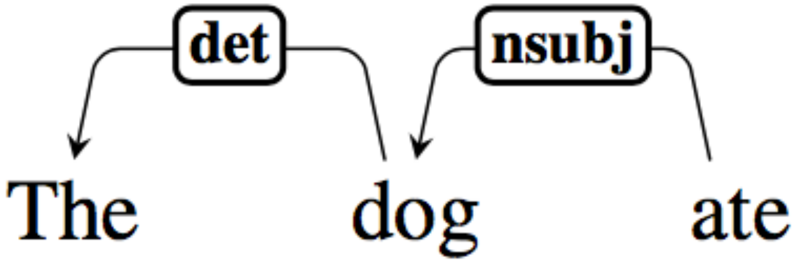
- Dependency syntax doesn’t have non-terminal structure like a CFG; words are directly linked to each other.

Dependency syntax

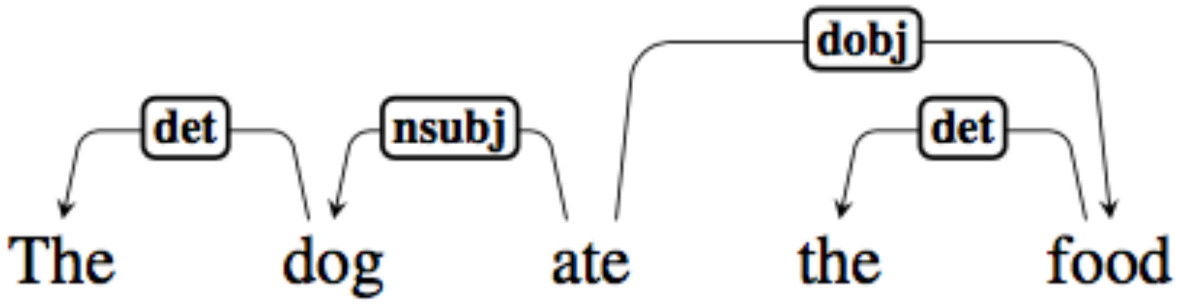


dependent head

Dependency syntax

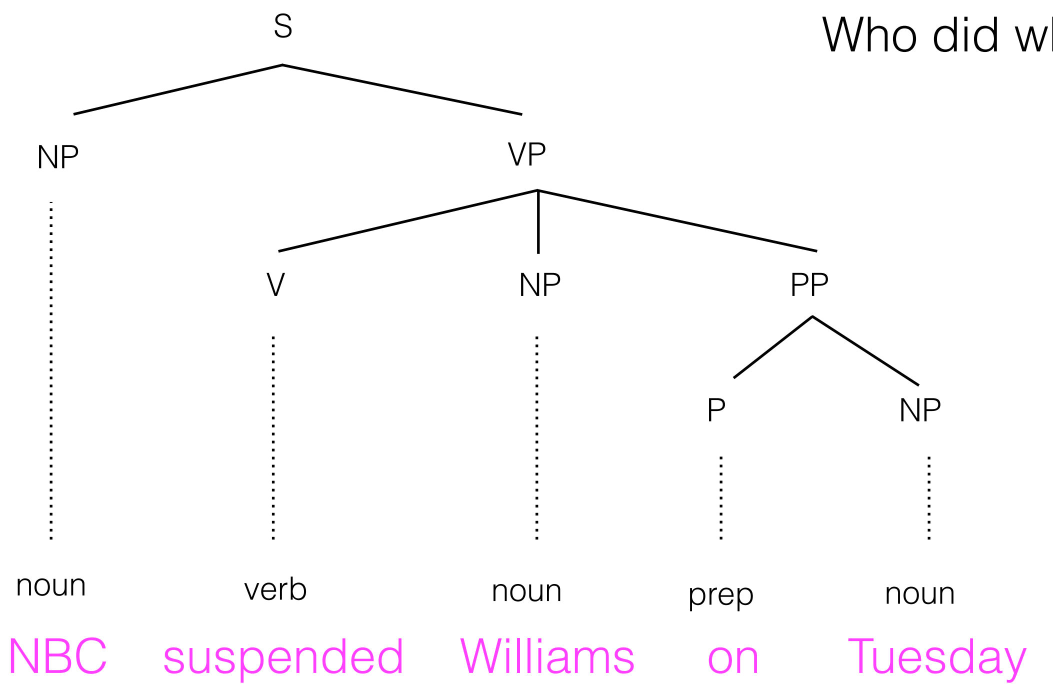


Dependency syntax



Dependencies vs constituents

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



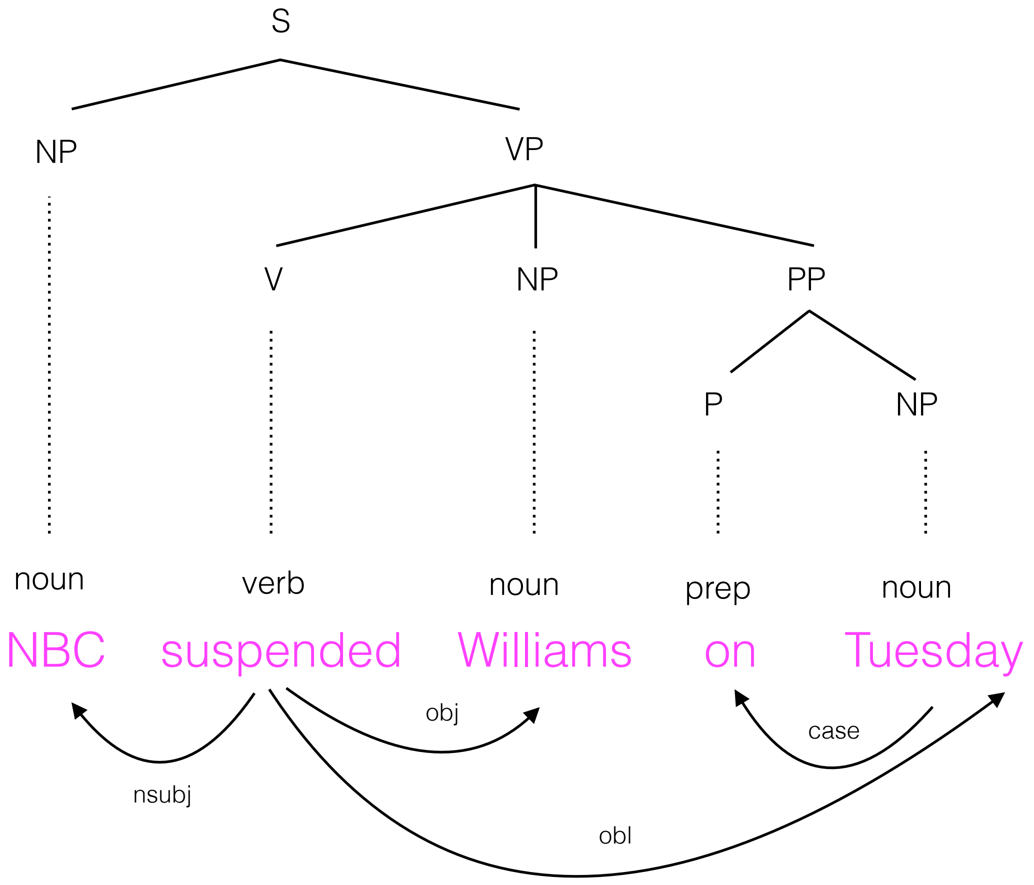
Who did what to whom?

subject: S → NP VP

direct object: S → NP (VP → … NP …)

Dependencies vs constituents

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

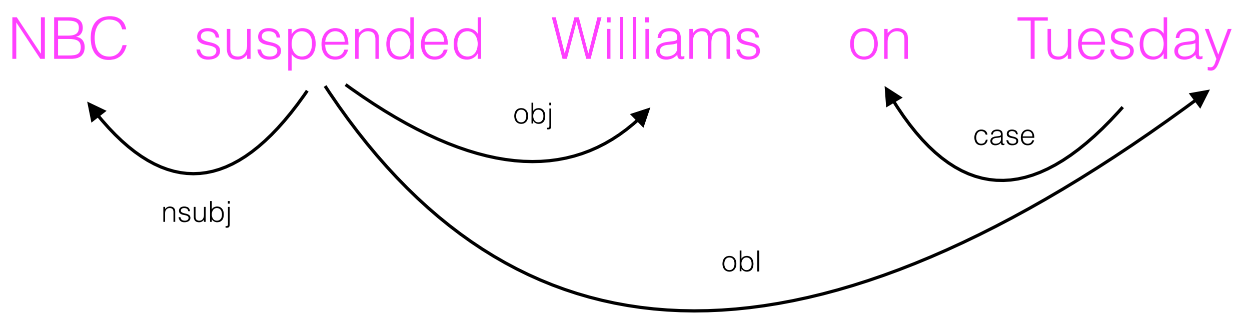


Dependencies vs constituents

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

Captures binary relations between words

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



Semantic Parsing

Semantic parsing comprises a wide range of tasks where strings are mapped into meaning representation languages. Examples:

- Programming languages, especially query languages that can be used to answer questions using a database (Zettlemoyer and Collins, 2005, e.g.,)

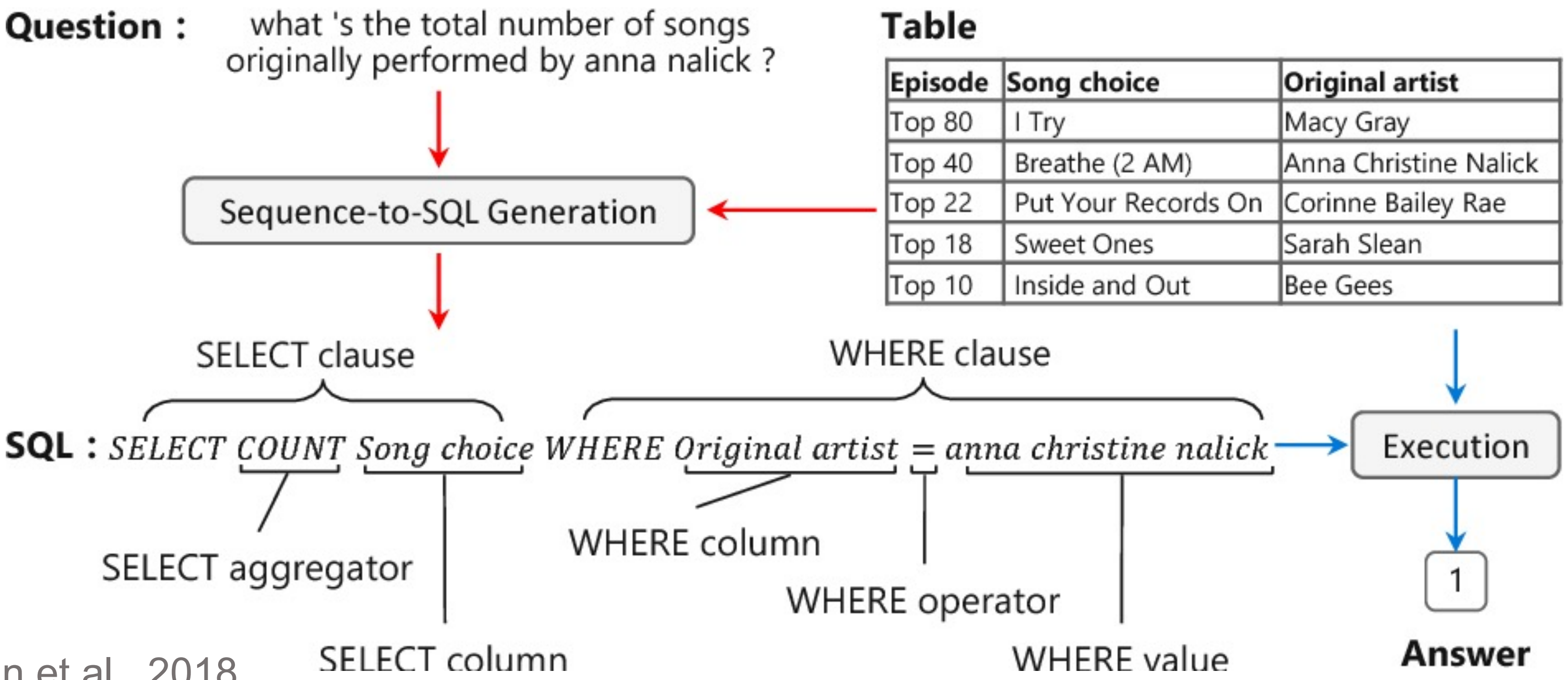


Figure Courtesy: Sun et al., 2018

Semantic Parsing

Semantic parsing comprises a wide range of tasks where strings are mapped into meaning representation languages. Examples:

- Programming languages, especially query languages that can be used to answer questions using a database (Zettlemoyer and Collins, 2005, e.g.,)
- Schemas designed around real-world event-types (called "frames"); trying to extract "who did what to whom?" (Baker et al., 1998; Palmer et al., 2005)

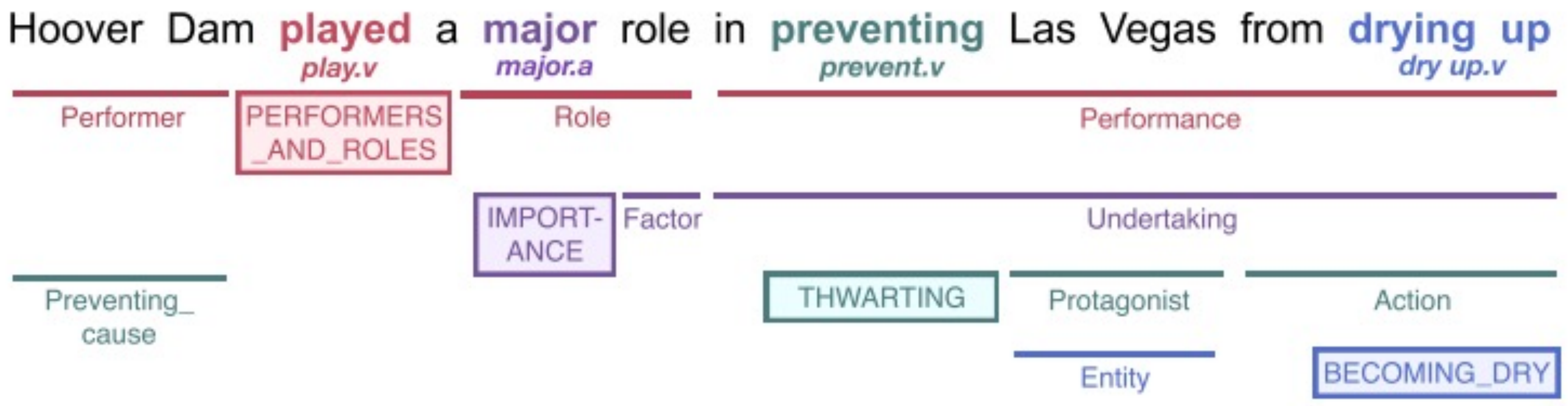


Figure Courtesy: Swayamdipta et al., 2017

Other Examples of Linguistic Structure Prediction

- Coreference resolution

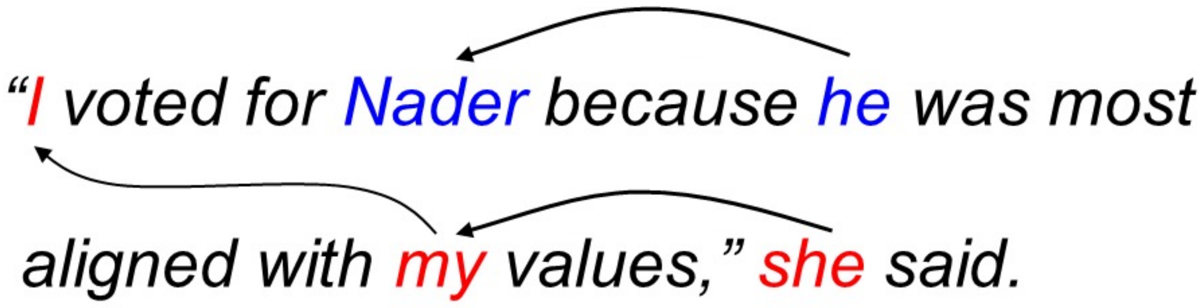


Figure Courtesy: https://nlp.stanford.edu/projects/coref.shtml

Other Examples of Linguistic Structure Prediction

- Coreference resolution
- Discourse parsing

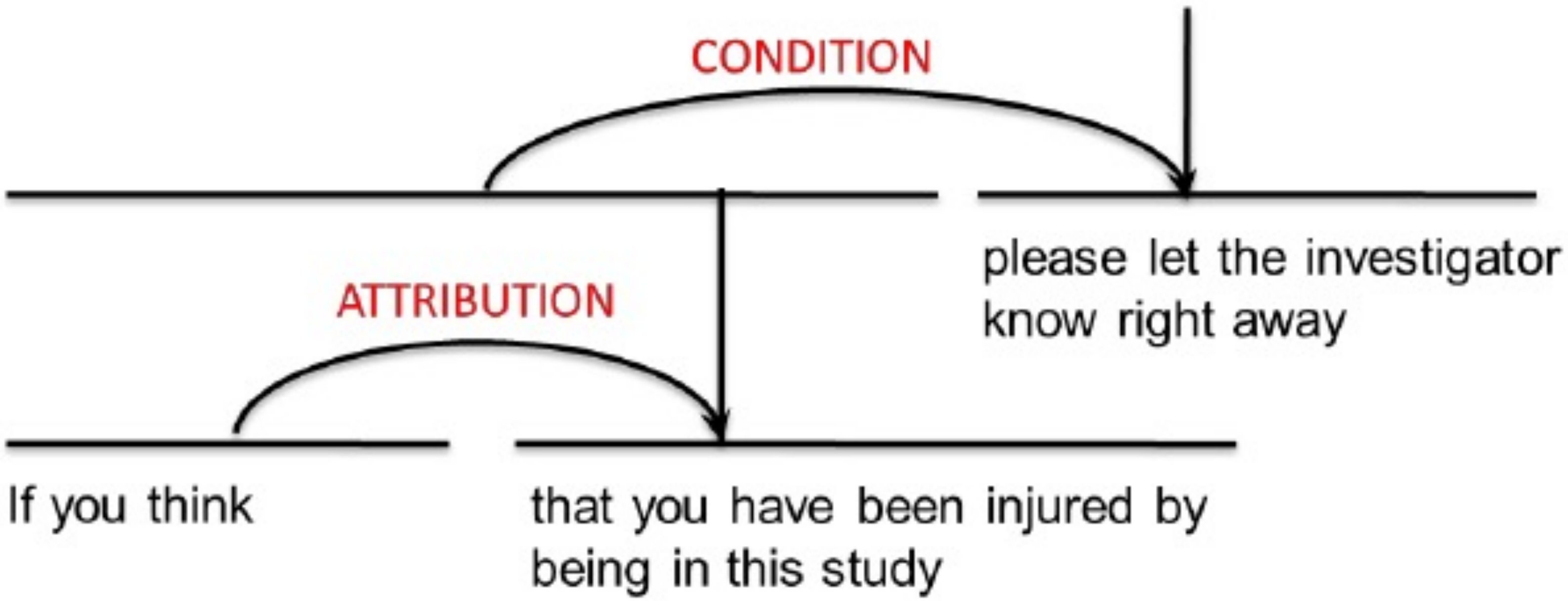


Figure Courtesy: Pascal Kuyten

Key Takeaways

- Syntax
  - Constituency parsing
    - CFG, PCFG
  - Dependency parsing
- Semantic Parsing
- Coreference Resolution
- Discourse Parsing

Questions?