Parsing

CSE 5525: Foundations of Speech and Natural Language Processing

https://shocheen.github.io/courses/cse-5525-spring-2025

The Ohio State University

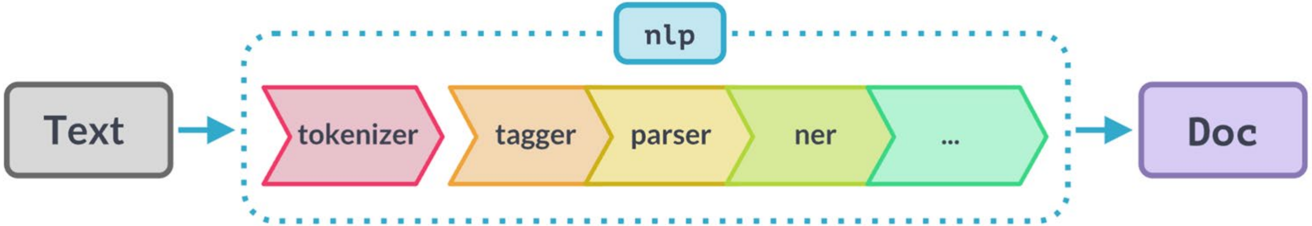
Slide Credits: Ana Marasović

Logistics

- Final Project Presentations
  - Dec 5, Dec 10
  - I'll randomly assign you to one of the days, if you can't make one of them please reach out to me
  - Each presentation will be 7-8 minutes depending on total number of teams with 1-2 time for QA (I will enforce hard time stops)

- Grades for the mid-project report will be released next week.

“Classical” NLP Pipeline



NAME	COMPONENT	CREATES	DESCRIPTION
tokenizer	`Tokenizer`	`Doc`	Segment text into tokens.
processing pipeline			
tagger	`Tagger`	`Token.tag`	Assign part-of-speech tags.
parser	`DependencyParser`	`Token.head`, `Token.dep`, `Doc.sents`, `Doc.noun_chunks`	Assign dependency labels.
ner	`EntityRecognizer`	`Doc.ents`, `Token.ent_iob`, `Token.ent_type`	Detect and label named entities.
lemmatizer	`Lemmatizer`	`Token.lemma`	Assign base forms.
textcat	`TextCategorizer`	`Doc.cats`	Assign document labels.
custom	custom components	`Doc._.xxx`, `Token._.xxx`, `Span._.xxx`	Assign custom attributes, methods or properties.

Source: https://spacy.io/usage/processing-pipelines

Hidden Markov Model for Tagging

$w_1, w_2, \ldots, w_N$ ... a sequence of observed words

$$\hat{t}_{1:N} = \text{argmax}_{t_{1:N}} \mathbb{P}(t_1, \ldots, t_N | w_1, \ldots, w_N)$$

Bayes rule to turn to the process we've just seen with the toy example

$$= \text{argmax}_{t_{1:N}} \frac{\mathbb{P}(w_1, \ldots, w_N | t_1, \ldots, t_N)\mathbb{P}(t_1, \ldots, t_N)}{\mathbb{P}(w_1, \ldots, w_N)}$$

The denominator independent of tags

$$= \text{argmax}_{t_{1:N}} \mathbb{P}(w_1, \ldots, w_N | t_1, \ldots, t_N)\mathbb{P}(t_1, \ldots, t_N)$$

Markov and output independence assumptions

$$\approx \text{argmax}_{t_{1:N}} \prod_{i=1}^{N} \mathbb{P}(w_i | t_i) \cdot \mathbb{P}(t_1) \prod_{i=2}^{N} \mathbb{P}(t_i | t_{i-1})$$

Let's use the notation we introduced

$$\approx \text{argmax}_{t_{1:N}} \prod_{i=1}^{N} B_{t_i, w_i} \cdot \pi_{t_1} \prod_{i=2}^{N} A_{t_{i-1}, t_i}$$

Once we have estimated emission, initial, and transition probabilities, all we need to do to get the most probable sequence of tags for a given sequence of words is to plug the probabilities into this equation for every possible sequence of tags and return the sequence that maximizes the equation's value

How many possible sequences?

The	Fed	raises	interest	rates

Suppose each word allows only the following tags

Determiner	Verb	Verb	Verb	Verb
	Noun	Noun	Noun	Noun
1	2	2	2	2

Slide credit: Vivek Srikumar

How many possible sequences?

The	Fed	raises	interest	rates

Suppose each word allows only the following tags

Determiner	Verb	Verb	Verb	Verb
	Noun	Noun	Noun	Noun
1	2	2	2	2

In this simple case, $1 \times 2 \times 2 \times 2 \times 2 = 16$ possible sequences exist

Slide credit: Vivek Srikumar

Given an observed sequence, and a model (π, A, B), how to efficiently calculate the most probable state sequence?

$t_1, t_2, \ldots, t_N$ ... a sequence of tags

$w_1, w_2, \ldots, w_N$ ... a sequence of observed words

$$\hat{t}_{1:N} = \text{argmax}_{t_{1:N}} \mathbb{P}(t_1, \ldots, t_N | w_1, \ldots, w_N) \approx \text{argmax}_{t_{1:N}} \prod_{i=1}^{N} B_{t_i, w_i} \cdot \pi_{t_1} \prod_{i=2}^{N} A_{t_{i-1}, t_i}$$

Naïve approaches

1. Try out every sequence and return the highest scoring one
   - Correct, but slow, $O(\text{num-possible-states}^{\text{sequence-length}})$
2. Greedy search
   - The best tag given the previously chosen tag and observed word
   - Incorrect, but fast $O(\text{sequence-length})$ $t_i = \arg\max_t \mathbb{P}(t \mid t_{i-1}) P(w_i \mid t)$

Viterbi algorithm: $O(\text{sequence-length} \times \text{num-possible-states}^2)$

Solution: Use the independence assumptions

$t_1, t_2, \ldots, t_N$… a sequence of tags

$w_1, w_2, \ldots, w_N$… a sequence of observed words

$$\hat{t}_{1:N} = \mathrm{argmax}_{t_{1:N}} \mathbb{P}(t_1, \ldots, t_N | w_1, \ldots, w_N) \approx \mathrm{argmax}_{t_{1:N}} \prod_{i=1}^{N} B_{t_i, w_i} \cdot \pi_{t_1} \prod_{i=2}^{N} A_{t_{i-1}, t_i}$$

Take advantage of the first order Markov assumption

The state for any observation is only influenced by the previous state, the next state and the observation itself

Given the adjacent labels, the others do not matter

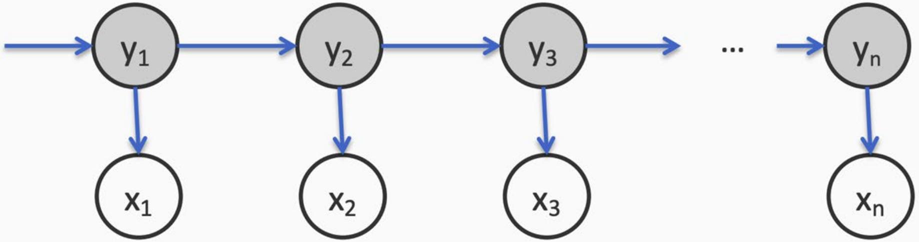
Suggests a recursive algorithm

Slide credit: Vivek Srikumar

Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

What we want: An assignment to all the y_i's that maximizes this product

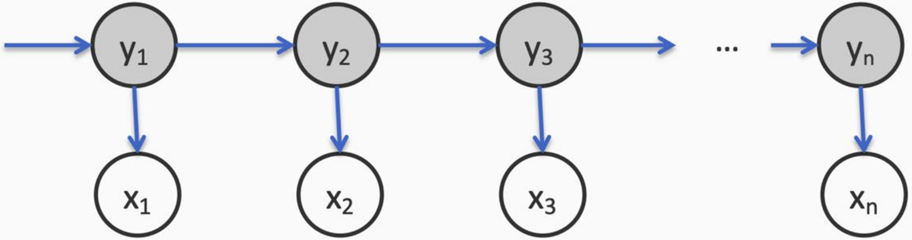


Slide credit: Vivek Srikumar

Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = \boxed{P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i)} \boxed{\prod_{i=1}^{n} P(x_i|y_i)}$$

$$\max_{y_1, y_2, \cdots, y_n} \boxed{P(y_n|y_{n-1})} \boxed{P(x_n|y_n)} \cdots \boxed{P(y_2|y_1)} \boxed{P(x_2|y_2)} \boxed{P(y_1)} \boxed{P(x_1|y_1)}$$

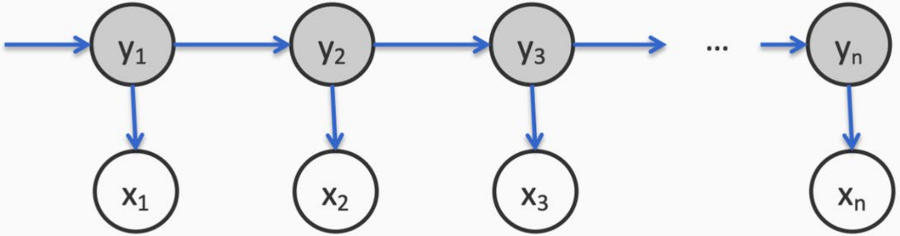


Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

Initial probability

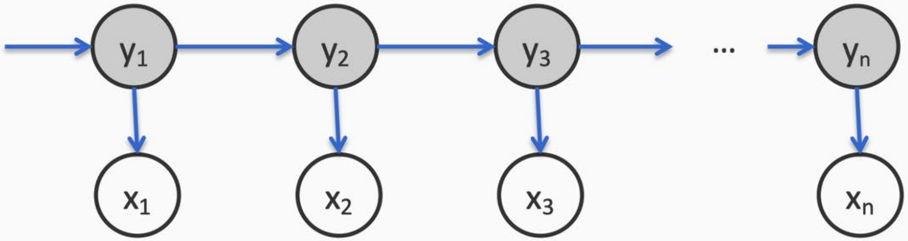


Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

Emission probabilities   Initial probability

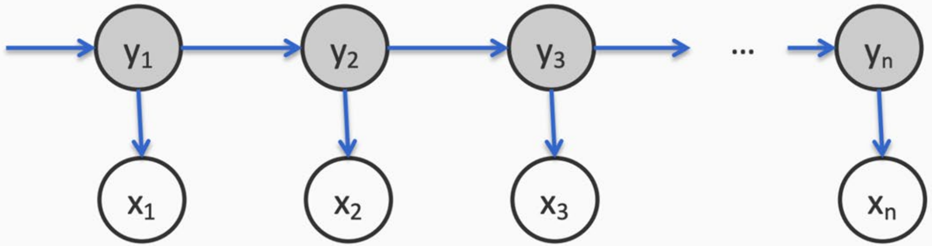


Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1}) P(x_n|y_n) \cdots P(y_2|y_1) P(x_2|y_2) P(y_1) P(x_1|y_1)$$

Transition probabilities   Emission probabilities   Initial probability



Deriving the recursive algorithm

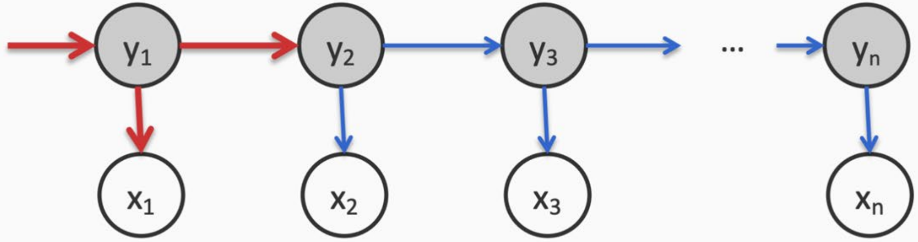
$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

Only a few factors depend on y_1 so we rearrange the product such that we place all those factors to the right

We can move max_{y_1} to the right too because other terms do not depend on y_1



Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

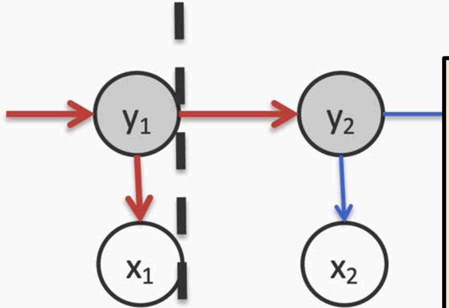
$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\mathrm{score}_1(y_1)$$

Abstract away the score for all decisions till here into score_1

$$\mathrm{score}_1(s) = P(s)P(x_1|s)$$



Abstract away the last two terms into something that we will give a special name `score_1`

`s` is a symbol for any state

Deriving the recursive algorithm

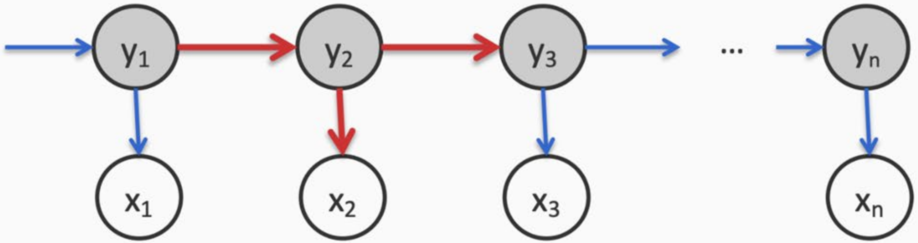
$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\mathrm{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3) \max_{y_1} P(y_2|y_1)P(x_2|y_2)\mathrm{score}_1(y_1)$$



Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

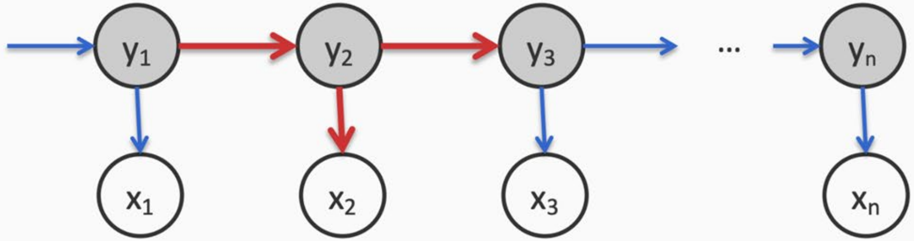
$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3) \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)$$

Only terms that depend on y_2

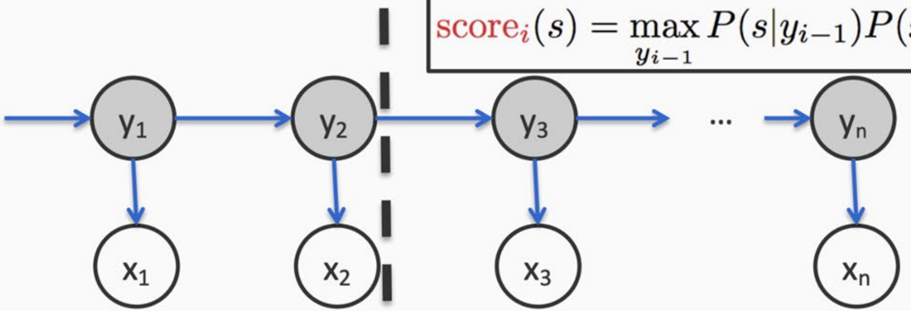


Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1)\prod_{i=1}^{n-1} P(y_{i+1}|y_i)\prod_{i=1}^{n} P(x_i|y_i)$$

$$\begin{aligned}
&\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3)\max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n)\cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3)\text{score}_2(y_2)
\end{aligned}$$

$$\boxed{\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})}$$



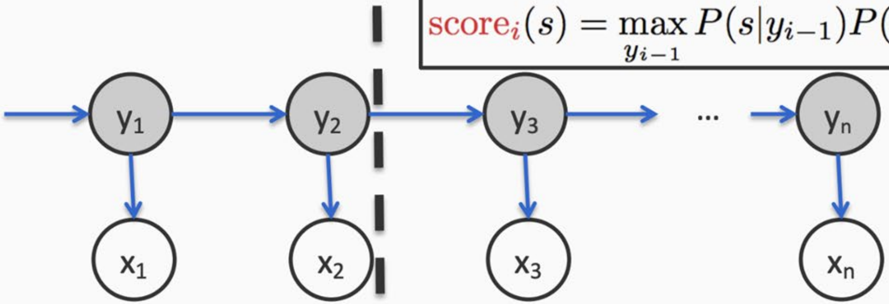
Abstract away the score for all decisions till here into score

Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\begin{aligned}
&\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1) \\
&= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\mathrm{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3) \max_{y_1} P(y_2|y_1)P(x_2|y_2)\mathrm{score}_1(y_1) \\
&= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3)\mathrm{score}_2(y_2)
\end{aligned}$$

$$\boxed{\mathrm{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\mathrm{score}_{i-1}(y_{i-1})}$$



Abstract away the score for all decisions till here into score

68

Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

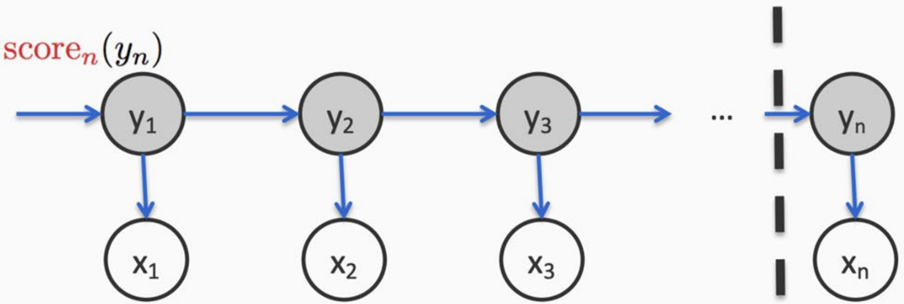
$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3) \max_{y_1} P(y_2|y_1)P(x_2|y_2)\text{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_2} P(y_3|y_2)P(x_3|y_3)\text{score}_2(y_2)$$

$$\vdots$$

$$= \max_{y_n} \text{score}_n(y_n)$$



Abstract away the score for all decisions till here into score

Deriving the recursive algorithm

$$P(x_1, x_2, \cdots, x_n, y_1, y_2, \cdots y_n) = P(y_1) \prod_{i=1}^{n-1} P(y_{i+1}|y_i) \prod_{i=1}^{n} P(x_i|y_i)$$

$$\max_{y_1, y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n|y_{n-1})P(x_n|y_n) \cdots \max_{y_1} P(y_2|y_1)P(x_2|y_2)P(y_1)P(x_1|y_1)$$

$$= \max_{y_2, \cdots, y_n} P(y_n \cdots$$

$$= \max_{y_3, \cdots, y_n} P(y_n \cdots x_2|y_2)\text{score}_1(y_1)$$

$$= \max_{y_3, \cdots, y_n} P(y_n \cdots$$

$$\vdots$$

$$= \max_{y_n} \text{score}_n(y_n)$$

$$\text{score}_1(s) = P(s)P(x_1|s)$$

$$\text{score}_i(s) = \max_{y_{i-1}} P(s|y_{i-1})P(x_i|s)\text{score}_{i-1}(y_{i-1})$$

Viterbi algorithm

Max-product algorithm for first order sequences

1. Initial: For each state s, calculate

$$score_1(s) = P(s)P(x_1 \mid s)$$

2. Recurrence: For i = 2 to n, for every state s, calculate

$$score_i(s) = \max_{y_{i-1}} P(s \mid y_{i-1})P(x_i \mid s)score_{i-1}(y_{i-1})$$

3. At the final state: calculate

$$\max_{y_{i-1}} P(y, x \mid \pi, A, B) = \max_{s} score_n(s)$$

Viterbi algorithm

Max-product algorithm for first order sequences

π: Initial probabilities
A: Transitions
B: Emissions

1. **Initial**: For each state s, calculate

$$score_1(s) = P(s)P(x_1 \mid s) = \pi_s B_{x_1,s}$$

2. **Recurrence**: For i = 2 to n, for every state s, calculate

$$score_i(s) = \max_{y_{i-1}} P(s \mid y_{i-1})P(x_i \mid s)score_{i-1}(y_{i-1})$$

$$\boxed{= \max_{y_{i-1}} A_{y_{i-1},s} B_{s,x_i} score_{i-1}(y_{i-1})}$$

3. **At the final state**: calculate

$$\max_{y_{i-1}} P(y, x \mid \pi, A, B) = \max_s score_n(s)$$

Viterbi algorithm

Max-product algorithm for first order sequences

π: Initial probabilities
A: Transitions
B: Emissions

1. Initial: For each state s, calculate

$$score_1(s) = P(s)P(x_1 \mid s) = \pi_s B_{x_1,s}$$

2. Recurrence: For i = 2 to n, for every state s, calculate

$$score_i(s) = \max_{y_{i-1}} P(s \mid y_{i-1})P(x_i \mid s)score_{i-1}(y_{i-1})$$
$$= \boxed{\max_{y_{i-1}}} A_{y_{i-1},s} B_{s,x_i} score_{i-1}(y_{i-1})$$

3. At the final state: calculate

$$\max_{y_{i-1}} P(y, x \mid \pi, A, B) = \max_s score_n(s)$$

Runtime complexity:
O(sequence length x #possible states^2)

This only calculates the max. To get final answer (*argmax*):

- keep track of which state corresponds to the max at each step
- build the answer using these back pointers

From sequence labeling to syntactic parsing

Syntax: The set of principles under which sequences of words are judged to be grammatically acceptable

- We've already learned one of the most basic syntactic concepts: the syntactic role of each word
🔜 Turning to thinking about word/phrase order

Grammar (informally) is the broader term that encompasses all implicit rules by which speakers intuitively judge which strings are well-formed and what they mean; including syntax, morphology, phonetics (sounds), semantics, and sometimes pragmatics (contextual use of language)

- Different from a **grammar formalism** that provides a set of mathematical rules or algorithms that can be used to generate the syntactic structures of a language

Syntactic parsing: The task of assigning a *syntactic structure* to a sequence of text

- Different theories of grammar propose different formalisms for describing the syntactic structure of sentences:
  ➔ constituency grammars & dependency grammars

Why do we care?

Getting the **right interpretations of words**:

- *<u>Visiting</u> relatives can be annoying.*
- *<u>Visiting relatives</u> can be annoying.*

Gateway to thinking about recognizing **who is doing what to whom**:

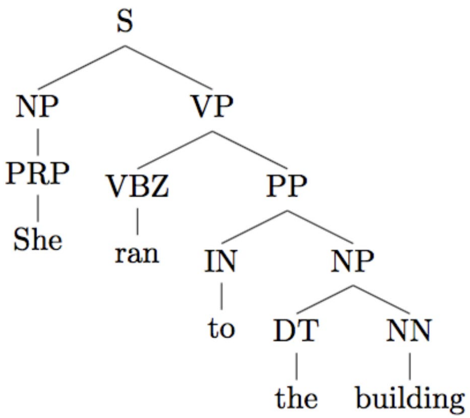
- *The **cat** <u>chased</u> the dog.*

Machine translation from subject-verb-object (SVO) languages like English to verb-subject-object (VSO) languages like Welsh [https://en.wikipedia.org/wiki/Verb%E2%80%93subject%E2%80%93object_word_order]

Grammar checking: Sentences that cannot be parsed may have grammatical errors (or at least be hard to read)

Always useful for chunking text into phrases

Constituency parsing: Intro



Constituency parsing is a method that breaks a sentence down into its constituent parts

Constituents are words or groups of words that function as a single unit within a hierarchical structure

- **S**entence, **N**oun **P**hrase, **V**erb **P**hrase, **P**repositional **P**hrases
- Bottom layers in POS tags

Constituents are represented in a **parse tree**

- Not a binary tree
- Right branching in English

Constituency makes sense for a lot of languages but not all, e.g., those where the word order is free such as Latin

Overview

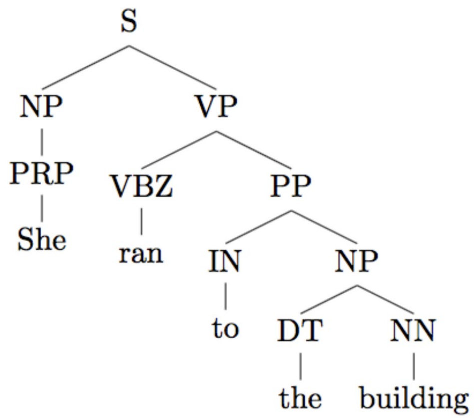
Learn how to produce a constituency parse using an non-neural algorithm

- **Context-Free Grammars (CFGs)**
- Probabilistic CFGs
- CKY Algorithm
- Evaluation

Dependency Parsing (if time)

Semantics and Discourse (if time)

Definition



Context-free grammars (CFGs) are tuples (N, Σ, R, S) consisting of:

- A finite set of **non-terminals** N
  - S, NP, VP, PP, ... , POS tags (**pre-terminals**)
- A finite alphabet/lexicon Σ of **terminal symbols**
  - Words
- A set of **productions** or **rules** R, each of the form $A \to \beta$, where $A \in N$ (so, a non-terminal) and β is a sequence of symbols in $\Sigma \cup N$ (so, a sequence of terminals or non-terminals)
  - NP $\to$ ProperNoun
- A designated start S

CFG: Toy example

Non-terminals, N = {S, NP, VP, DET, N, V}

Terminals, Σ = {"the", "a", "cat", "dog", "sleeps", "eats"}

or

Productions/rules, R = {S → NP VP, NP → Det N, VP → V NP | V, Det → "the" | "a", N → "cat" | "dog", V → "sleeps" | "eats"}

binary rules

unary rules

Start symbol, S = S (Sentence)

With this CFG, we can generate simple sentences like: "The cat sleeps"

1. Start with S
2. Replace S with NP VP (S → NP VP)
3. Replace NP with Det N (NP → Det N)
4. Replace Det with "the" (Det → "the")
5. Replace N with "cat" (N → "cat")
6. Replace VP with V (VP → V)
7. Replace V with "sleeps" (V → "sleeps")

A few more good-to-know terms

Derivation: A sequence of steps from the start symbol S to a surface string of non-terminals, which is the **yield** of the derivation

A **string is in a context-free language** if there is some derivation from S yielding this string

Parsing: The problem of finding a derivation for a string in a grammar

Informally…

Probabilistic context-free grammars (PCFGs) are CFGs, but **rules have probabilities** that represent the likelihood of a particular production being used in the derivation of a sentence; by now we know that probabilities can be estimated from data and this **helps with ambiguities**

S → NP VP	p=1.0
NP → Det N	p=1.0
VP → V NP	p=0.2
VP → V	p=0.8
Det → “the”	p=0.4
Det → “a”	p=0.6
N → “cat”	p=0.45
N → “dog”	p=0.55
V → “sleeps”	p=0.7
V → “eats”	p=0.3

Informally…

The probabilities for all rules expanding the same non-terminal [the left-hand side, LHS] should sum to 1

S $\rightarrow$ NP VP	p=1.0
NP $\rightarrow$ Det N	p=1.0
VP $\rightarrow$ V NP	p=0.2
VP $\rightarrow$ V	p=0.8
Det $\rightarrow$ "the"	p=0.4
Det $\rightarrow$ "a"	p=0.6
N $\rightarrow$ "cat"	p=0.45
N $\rightarrow$ "dog"	p=0.55
V $\rightarrow$ "sleeps"	p=0.7
V $\rightarrow$ "eats"	p=0.3

For all n in N:

$$\sum_{r \in R \text{ s.t. } n = \mathrm{LHS}(r)} \mathbb{P}(r|n) = 1$$

$$\mathbb{P}(\text{tree}) = \prod_{r \text{ in derivation}} \mathbb{P}(r|\mathrm{LHS}(r))$$

How to estimate these probabilities?

Supervised approach

Treebanks: Corpora that have been annotated with syntactic structure

- Penn Treebank project, which includes various treebanks in English, Arabic, and Chinese

As with HMM, the probabilities that maximize the likelihood of data can be estimated by counting and normalizing:

- For each non-terminal, divide the frequency of each rule that terminal is the left-hand side of by the total number of occurrences of that non-terminal's expansions
- $P(S \rightarrow NP\ VP) = 100 / 150 = 2/3$
- $P(S \rightarrow VP) = 50 / 150 = 1/3$
- Smoothing

Let's parse!

Given a sentence, how do we find the highest scoring parse tree for it?

We'll apply the **CKY algorithm** to *Probabilistic* **Context-Free Grammars**

CKY (Cocke-Kasami-Younger) algorithm

A **bottom-up parser**:

- Starts by recognizing the smallest components (like individual words) and gradually builds up to larger structures (like phrases or entire sentences)

Dynamic programming to parse <u>efficiently</u>:

- Once a substring is analyzed and its possible derivations are stored, these results are reused whenever that substring is part of a larger segment being analyzed

Ambiguity handling:

- CKY allows multiple entries for each substring in the table where it stores intermediate results, reflecting the different possible derivations
- Finds the most likely parse when applied to PCFGs

CKY Step 1: Convert the PCFG to Chomsky Normal Form (CNF)

Also known as **binarization**

In CNF**,** the right-hand side of every production includes either two non-terminals, or a single terminal symbol

The CKY algorithm we present applies to a restricted type of PCFG: a PCFG where which is in **Chomsky normal form (CNF)**

- Turns out this is not a very strong assumption
- We won't go into details but there are ways to remove all unary rules and transform n-ary rules

CKY Step 2: Initialize the parsing table

Create a triangular matrix/table where the rows and columns correspond to the words in the sentence

Each cell `(i,j)`, `i<j`, represents the substring from the `i`-th to `j`-th word, so we start counting columns by 1 and rows from 0 cells

Each cell in the matrix will store the most probable non-terminal(s) that can generate the corresponding substring of the sentence, along with the probability of the most likely derivation

$S \rightarrow NP\ VP\ (0.9)$
$S \rightarrow VP\ (0.1)$
$VP \rightarrow V\ NP\ (0.5)$
$VP \rightarrow V\ (0.5)$
$NP \rightarrow$ "she" (0.5)
$NP \rightarrow$ "fish" (0.5)
$V \rightarrow$ "eats" (1.0)

		she	eats	fish
she		(0,1)	(0,2)	(0,3)
eats			(1,2)	(1,3)
fish				(2,3)

CKY Step 3: Populate the parsing table

Fill in the diagonal of the matrix with the non-terminal(s) that can produce that word, along with the probability of that production

S → NP VP (0.9)
S → VP (0.1)
VP → V NP (0.5)
VP → V (0.5)
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

		she	eats	fish
she		NP (0.5)		
eats			V (1.0)	
fish				NP (0.5)

CKY Step 3: Populate the parsing table **(continued)**

Populate the rest of the table a column at a time working from left to right, with each column filled from bottom to top

- A bottom-up fashion so that at the point where we are filling any cell, the cells containing the parts that could contribute to this entry [the cells to the left and the cells below] have already been filled

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word, compute the most probable non-terminals that can generate this string:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` in cell `(i,j)`

CKY Step 3: Populate the parsing table **(continued)**

$S \rightarrow NP\ VP\ (0.9)$
$S \rightarrow VP\ (0.1)$
$VP \rightarrow V\ NP\ (0.5)$
$VP \rightarrow V\ (0.5)$
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i,j)`

Non-terminals: NP, V
Rule with NP V on RHS?

(i,k)=
(0,1)

		she	eats	fish
she		NP (0.5)	(0,2),k=1	
eats			V (1.0)	
fish				NP (0.5)

(k,j)=
(1,2)

CKY Step 3: Populate the parsing table **(continued)**

S → NP VP (0.9)
S → VP (0.1)
VP → V NP (0.5)
VP → V (0.5)
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i,j)`

Non-terminals: NP, V
Rule with NP V on RHS?
<u>None!</u>

		she	eats	fish
she		NP (0.5)	Ø	
eats			V (1.0)	
fish				NP (0.5)

CKY Step 3: Populate the parsing table **(continued)**

S → NP VP (0.9)
S → VP (0.1)
VP → V NP (0.5)
VP → V (0.5)
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i,j)`

Non-terminals: V, NP
Rule with V NP on RHS?

		she	eats	fish
she		NP (0.5)	Ø	(i,k)= (1,2)
eats			V (1.0)	(1,3), k=2
fish				(k,j)= (2,3)

CKY Step 3: Populate the parsing table (continued)

$S \rightarrow NP\ VP\ (0.9)$
$S \rightarrow VP\ (0.1)$
$VP \rightarrow V\ NP\ (0.5)$
$VP \rightarrow V\ (0.5)$
$NP \rightarrow$ "she" (0.5)
$NP \rightarrow$ "fish" (0.5)
$V \rightarrow$ "eats" (1.0)

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i,j)`

NP VP? Yes!

$$\mathbb{P}(VP \rightarrow V\ NP) \cdot \mathbb{P}(V \rightarrow \text{eats}) \cdot \mathbb{P}(NP \rightarrow \text{fish}) = 0.5 \cdot 1.0 \cdot 0.5 = 0.25$$

		she	eats	fish
she		NP (0.5)	Ø	
eats			V (1.0)	VP (0.25)
fish				NP (0.5)

CKY Step 3: Populate the parsing table (continued)

$S \rightarrow NP\ VP\ (0.9)$
$S \rightarrow VP\ (0.1)$
$VP \rightarrow V\ NP\ (0.5)$
$VP \rightarrow V\ (0.5)$
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i,j)`

Non-terminals NP VP
Rule with NP VP on RHS?

		she	eats	fish
she		NP (0.5)	Ø	(0,3),k=1,2
eats			V (1.0)	VP (0.25)
fish				NP (0.5)

CKY Step 3: Populate the parsing table **(continued)**

S → NP VP (0.9)
S → VP (0.1)
VP → V NP (0.5)
VP → V (0.5)
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i,j)`

		she	eats	fish
she		NP (0.5)	Ø	S (0.1125)
eats			V (1.0)	VP (0.5)
fish				NP (0.5)

CKY Step 3: Populate the parsing table (continued)

S → NP VP (0.9)
S → VP (0.1)
VP → V NP (0.5)
VP → V (0.5)
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i,j)`

Non-terminals NP VP
Rule with NP VP on RHS?

		she	eats	fish
she		NP (0.5)	Ø	(0,3),k=1,2
eats			V (1.0)	VP (0.25)
fish				NP (0.5)

CKY Step 3: Populate the parsing table (continued)

S → NP VP (0.9)
S → VP (0.1)
VP → V NP (0.5)
VP → V (0.5)
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i,j)`

Non-terminals: NP
Rules with NP on RHS?

		she	eats	fish
she		NP (0.5)	Ø	S (0.1125) (0,3),k=1,2
eats			V (1.0)	VP (0.5)
fish				NP (0.5)

CKY Step 3: Populate the parsing table (continued)

S → NP VP (0.9)
S → VP (0.1)
VP → V NP (0.5)
VP → V (0.5)
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

For each cell `(i, j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B, C)` in the cells `(i, k)` and `(k, j)`
- For each pair `(B, C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i, k)` and `(k, j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i, j)`

		she	eats	fish
she		NP (0.5)	Ø	S (0.1125)
eats			V (1.0)	VP (0.5)
fish				NP (0.5)

CKY Step 3: Populate the parsing table (continued)

$S \rightarrow NP\ VP\ (0.9)$
$S \rightarrow VP\ (0.1)$
$VP \rightarrow V\ NP\ (0.5)$
$VP \rightarrow V\ (0.5)$
NP → "she" (0.5)
NP → "fish" (0.5)
V → "eats" (1.0)

For each cell `(i,j)`, `i<j`, representing the substring from the `i`-th to `j`-th word:

- Split the substring into two parts at every possible point `k`, where `i<k<j`
- Check every pair of non-terminals `(B,C)` in the cells `(i,k)` and `(k,j)`
- For each pair `(B,C)`, look for a rule `A -> BC` and calculate the probability of this rule multiplied by the probabilities storied in `(i,k)` and `(k,j)`
- Keep the max. probability and the corresponding non-terminal `A` along with the split point `k` in cell `(i,j)`

**(S
(NP she)
(VP
(V eats)
(NP fish)
))**

		she	eats	fish
she		NP (0.5)	Ø	S (0.1125)
eats			V (1.0)	VP (0.25)
fish				NP (0.5)

Constituency parsing: Evaluation

Given a **treebank**: How much the constituents in the **hypothesis parse** tree look like the constituents in a hand-labeled, **reference parse**?

A constituent in a hypothesis parse of a sentence s is labeled correct if there is a constituent in the reference parse with the same starting point, ending point, and non-terminal symbol.

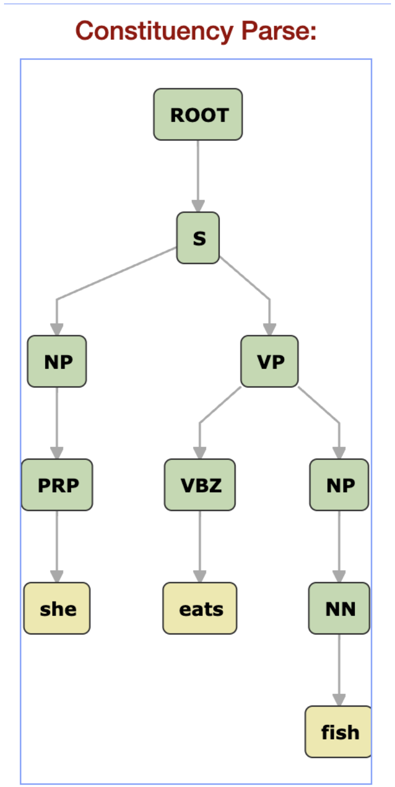
$$\textbf{labeled recall:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of total constituents in reference parse of } s}$$

$$\textbf{labeled precision:} = \frac{\text{\# of correct constituents in hypothesis parse of } s}{\text{\# of total constituents in hypothesis parse of } s}$$

As always, calculate F1!

CKY: Bottom-up parser

$S \rightarrow NP\ VP\ (0.9)$
$S \rightarrow VP\ (0.1)$
$VP \rightarrow V\ NP\ (0.5)$
$VP \rightarrow V\ (0.5)$
$NP \rightarrow$ "she" (0.5)
$NP \rightarrow$ "fish" (0.5)
$V \rightarrow$ "eats" (1.0)



		she	eats	fish
she		NP (0.5)	Ø	S (0.1125)
eats			V (1.0)	VP (0.25)
fish				NP (0.5)

Source: https://corenlp.run/ (not using the CKY and the same grammar!)

Use spaCy

Berkeley Neural Parser

Constituency Parsing with a Self-Attentive Encoder (ACL 2018)

release models | license MIT | Stars 873

A Python implementation of the parsers described in *"Constituency Parsing with a Self-Attentive Encoder"* from ACL 2018.

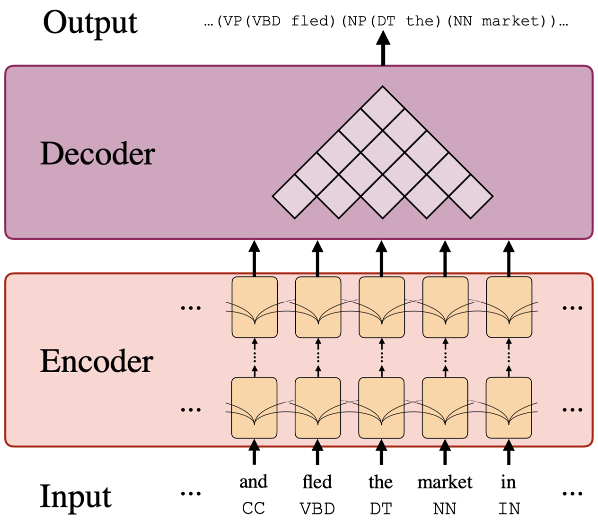
EXAMPLE

```
import benepar, spacy
nlp = spacy.load('en_core_web_md')
nlp.add_pipe('benepar', config={'model': 'benepar_en3'})
doc = nlp('The time for action is now. It is never too late to do something.')
sent = list(doc.sents)[0]
print(sent._.parse_string)
# (S (NP (NP (DT The) (NN time)) (PP (IN for) (NP (NN action)))) (VP (VBZ is) (
print(sent._.labels)
# ('S',)
print(list(sent._.children)[0])
# The time for action
```

INSTALLATION

```
pip install benepar
```

[Constituency Parsing with a Self-Attentive Encoder]



https://spacy.io/universe/project/self-attentive-parser

Dependency grammars

Constituency formalism based on phrasal constituents and phrase-structure rules

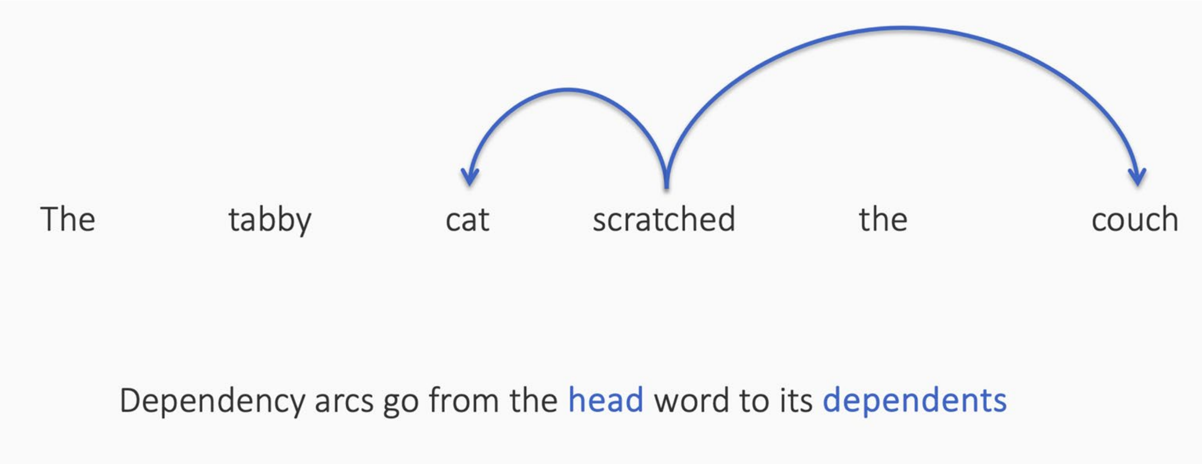
In **dependency formalism**: The syntactic structure of a sentence is described solely in terms of directed binary grammatical relations between the words

Figure source: Vivek Srikumar

Dependency grammars

Constituency formalism based on phrasal constituents and phrase-structure rules

In **dependency formalism**: The syntactic structure of a sentence is described solely in terms of directed binary grammatical relations between the words



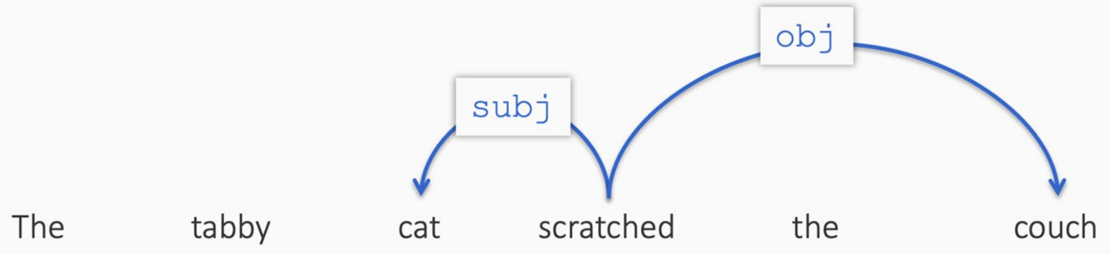
Head: (informally) the central organizing word
Dependent: (informally) modifier

Figure source: Vivek Srikumar

Dependency grammars

Constituency formalism based on phrasal constituents and phrase-structure rules

In **dependency formalism**: The syntactic structure of a sentence is described solely in terms of directed binary grammatical relations between the words



Dependency arcs are labeled with grammatical relations such as subject, object, noun modifier, determiner, etc.
The label describes what grammatical function the dependent plays with respect to the head

Figure source: Vivek Srikumar

Dependency grammars

Constituency formalism based on phrasal constituents and phrase-structure rules

In **dependency formalism**: The syntactic structure of a sentence is described solely in terms of directed binary grammatical relations between the words

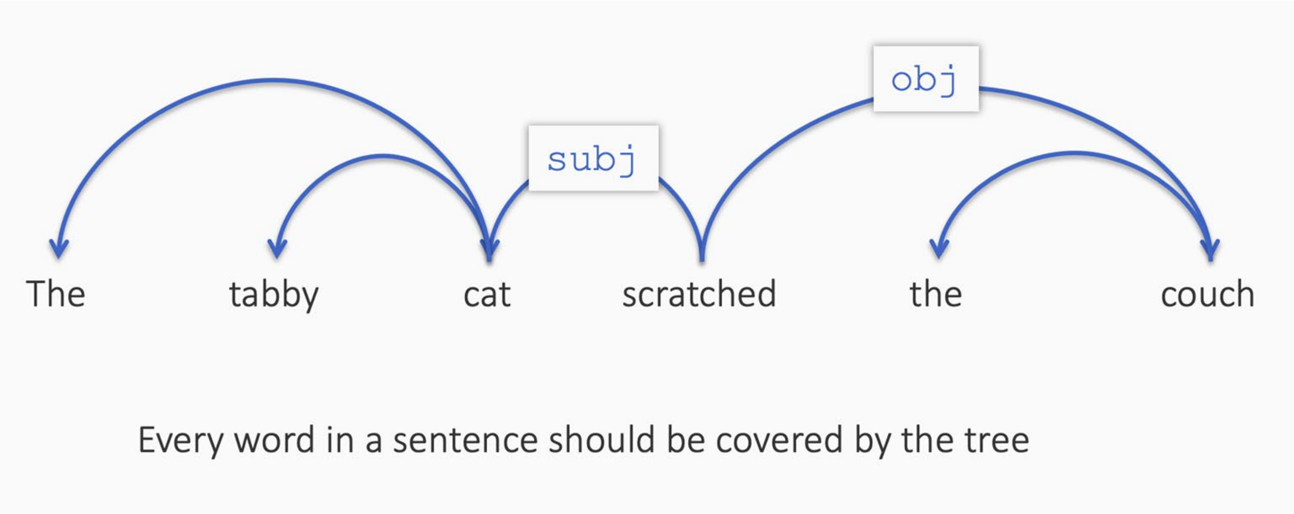


Figure source: Vivek Srikumar

Dependency grammars

Constituency formalism based on phrasal constituents and phrase-structure rules

In **dependency formalism**: The syntactic structure of a sentence is described solely in terms of directed binary grammatical relations between the words

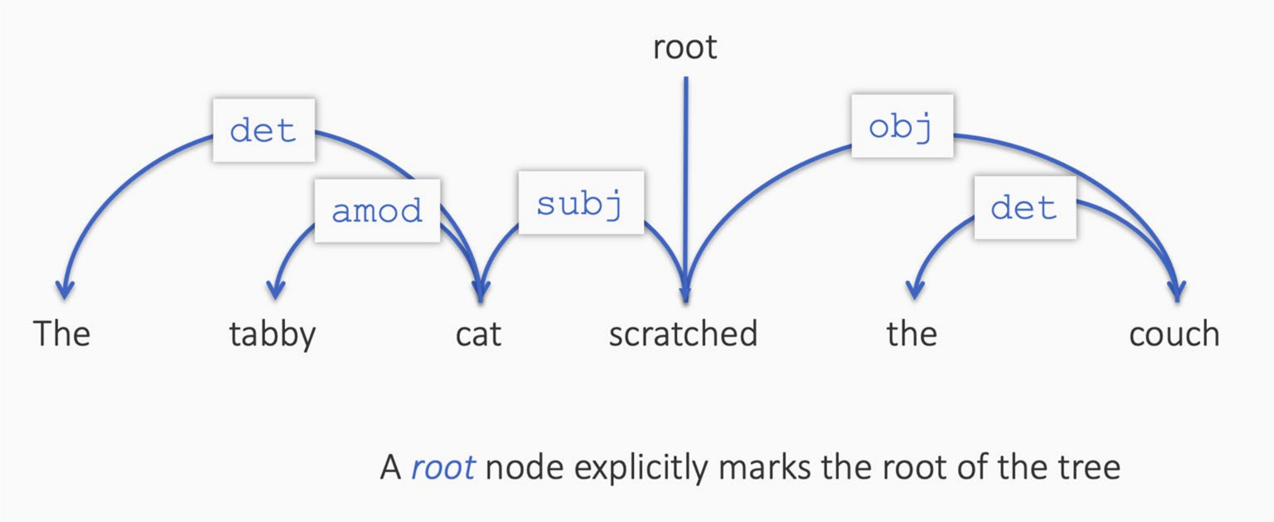


Figure source: Vivek Srikumar

Dependency vs. constituency

Not illustrated here:
Dependencies handle languages that have free word order more elegantly

no nodes corresponding to phrasal constituents (NPs, VPs, …)

the arguments to the verb are directly linked to it

the arguments's connection to the main verb is more distant

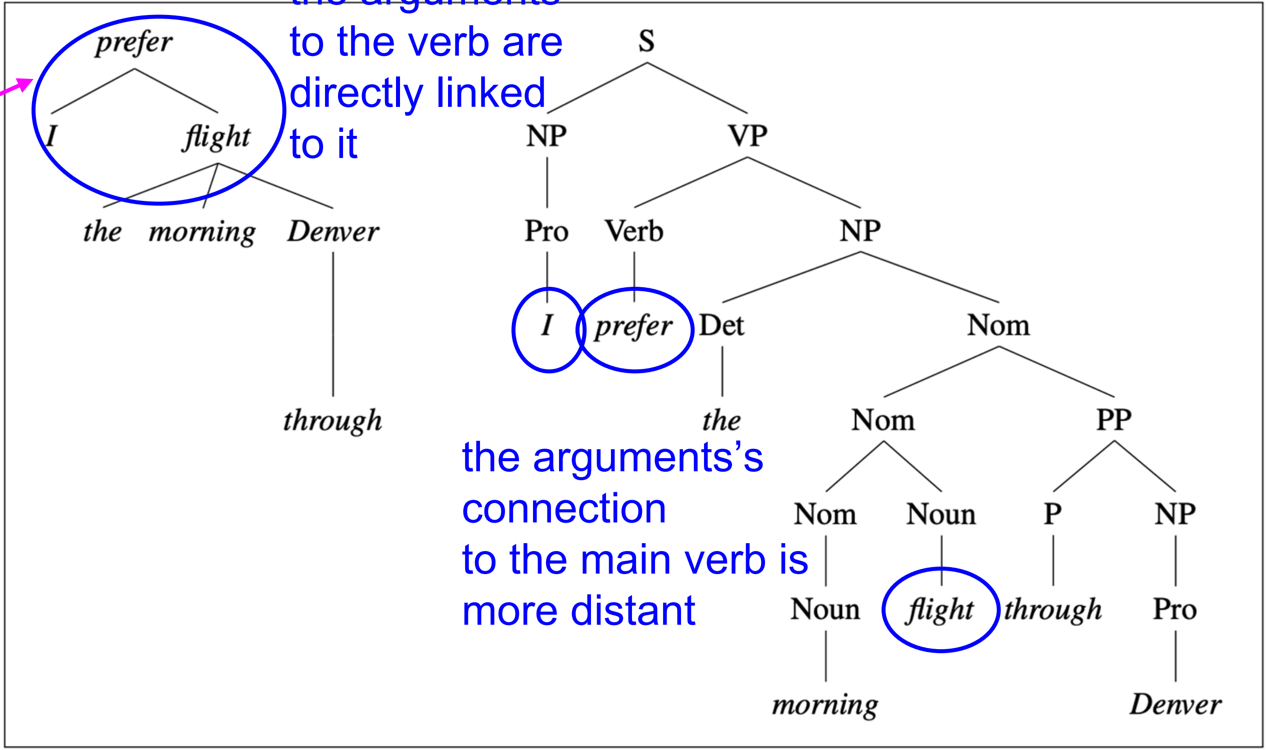


Figure 18.1 Dependency and constituent analyses for *I prefer the morning flight through Denver.*

Figure source: Jurafsky & Martin

Dependency Formalisms

$G = (V, A)$ ... a directed graph representing a dependency structure

V ... a set of vertices (words, but also punctuation & sometimes stems and affixes)

A ... a set of labeled arcs (ordered pairs of vertices)

A **dependency tree** is a directed graph that satisfies the following constraints:

1. There is a single designated root node that has no incoming arcs
2. With the exception of the root node, each vertex has exactly one incoming arc
3. There is a unique path from the root node to each vertex in V

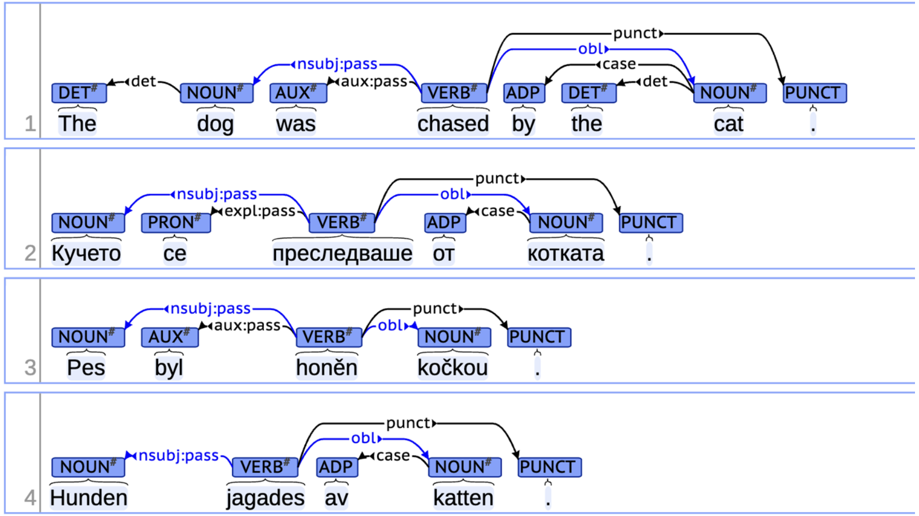
Each word has a single head, the dependency structure is connected, and there is a single root node from which one can follow a unique directed path to each of the words in the sentence.

The **Universal Dependencies** (UD) project

[de Marneffe et al., 2021]; https://universaldependencies.org/

An open community effort to annotate dependencies across more than 100 languages, provides an inventory of 37 dependency relations and 200+ treebanks

"The general philosophy is to provide a universal inventory of categories and guidelines to facilitate consistent annotation of similar constructions across languages, while allowing language-specific extensions when necessary."



FYI McDonald et al. "Universal Dependency Annotation for Multilingual Parsing" won the test-of-time award at ACL 2023!

analysis of the individual components of words like prefixes and suffixes

identifying and analyzing the structure of words and parts of speech

syntactic structure like a constituency or dependency parse tree

meaning of words (lexical semantics) but also entire expressions

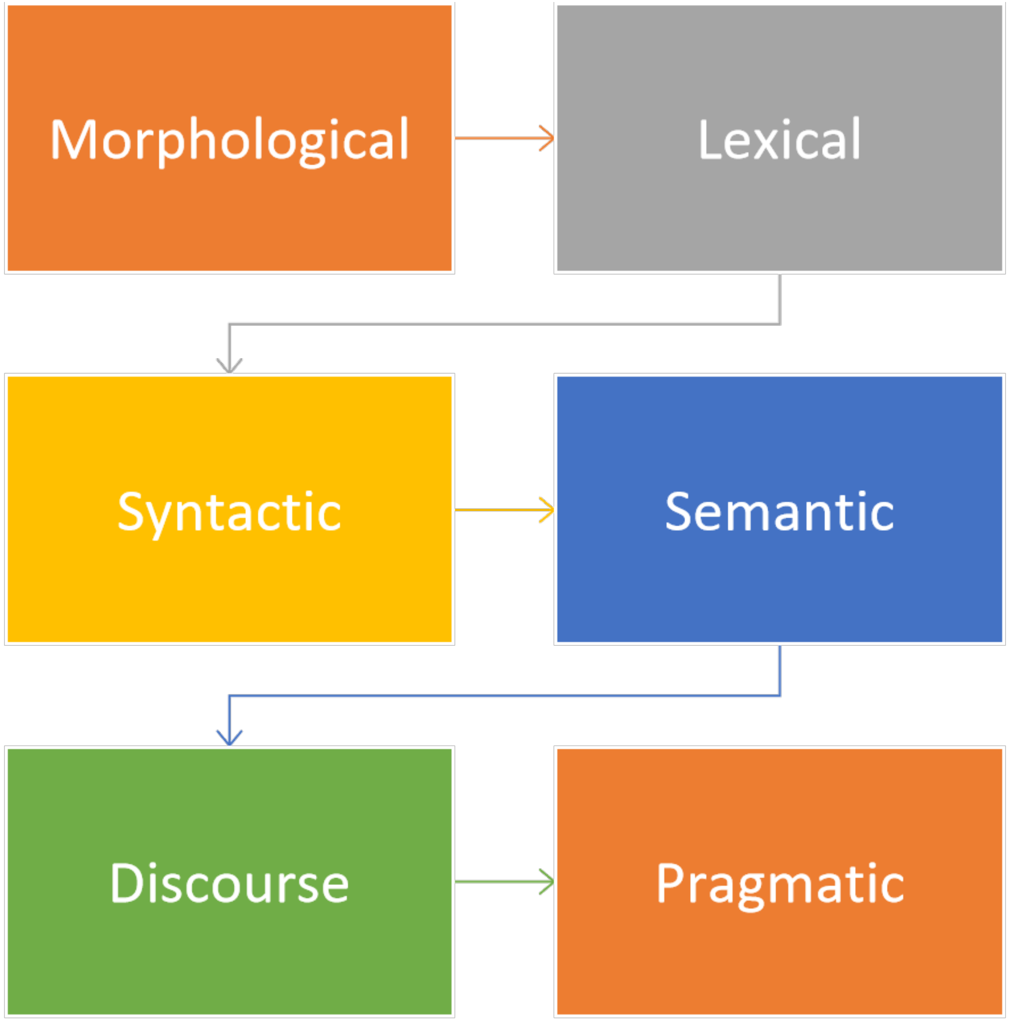


Figure source: https://www.datascienceprophet.com/different-levels-of-natural-language-processing/

Semantics

The study of linguistic **meaning**. It examines what meaning is, how words get their meaning, and how the meaning of a complex expression depends on its parts.

Reminder: Lexical semantics

Source: [Wiki]

Sense

The **sense** of an expression is the *idea*, *concept*, or *mental representation* associated with it

- It's about how we understand the meaning of the expression, independent of any specific context or object
- *Example:* Think about the word "cat"
  - The sense includes the idea of a small, furry, domesticated animal that purrs, has claws, and so on
  - This is the concept of a cat, which is stored in your mind

Source: Wiki

Reference

The **reference** of an expression is the actual object or entity in the real world that the expression refers to or points to in a specific context

- *Example*: If you say, "*My cat is sleeping*" the reference is your actual, specific cat.
  Another person's "cat" would have a different reference, even though the sense of the word is shared

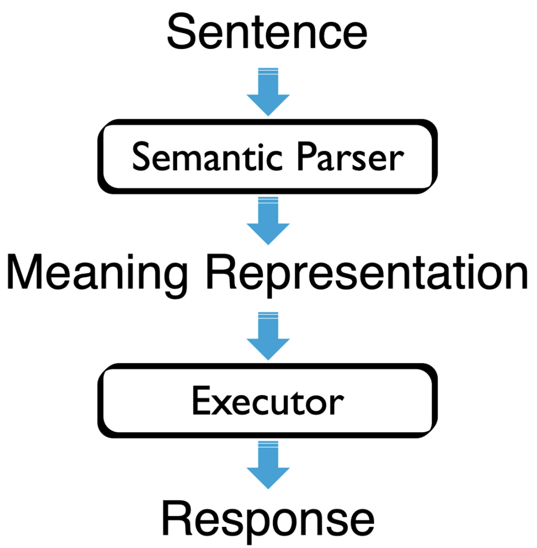
Source: [Wiki]

Semantic Parsing

Semantic parsing:
The task of converting a natural language utterance to a logical form or a program: a machine-understandable representation of its meaning

Meaning representations:
Formal structures that capture the "complete" meaning of linguistic expressions

What's complete? Debatable



[ACL 2018 tutorial on neural semantic parsing]

Semantic Parsing: QA

How many people live in Seattle?

↓

Semantic Parser

↓

```
SELECT Population FROM CityData
where City=="Seattle";
```

↓

Executor

↓

620,778

[Wong & Mooney 2007], [Zettlemoyer & Collins 2005, 2007], [Kwiatkowski et.al 2010, 2011], [Liang et.al. 2011],[Berant et.al. 2013,2014],[Reddy et.al, 2014,2016], [Dong and Lapata, 2016] .....

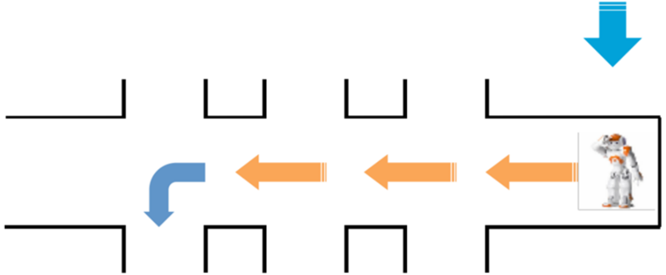
[ACL 2018 tutorial on neural semantic parsing]

Semantic Parsing: Instructions

Go to the third junction and take a left

Semantic Parser

```
(do-seq(do-n-times 3
  (move-to forward-loc
    (do-until
      (junction current-loc
        (move-to forward-loc))))
  (turn-right))
```



[Chen & Mooney 2011]
[Matuszek et al 2012]
[Artzi & Zettlemoyer 2013]
[Mei et.al. 2015][Andreas et al, 2015]
[Fried at al, 2018] ....

[ACL 2018 tutorial on neural semantic parsing]

Unlike syntax, where there are standard formalisms (e.g. UD, etc), there are no standard semantic formalisms

Semantics itself is not well defined because we have the following:

- Usually, predicate logic is used as the representation of choice
- Some (very restrictive) work involves quantified (i.e. first order) logic
- Some representations involve graphs (e.g. AMR)
- Some people argue that semantics should be represented by text (e.g. QA-SRL)
- It is usually English-specific

Semantic roles

For an event that is described in a verb, different noun phrases fulfill different semantic roles

- Think of noun phrases as representing typed arguments

Semantic roles

For an event that is described in a verb, different noun phrases fulfill different semantic roles

Think of noun phrases as representing typed arguments

John saw Mary eat the apple

Semantic roles

For an event that is described in a verb, different noun phrases fulfill different semantic roles

Think of noun phrases as representing typed arguments

The seeing event

John saw Mary eat the apple

Semantic roles

For an event that is described in a verb, different noun phrases fulfill different semantic roles

Think of noun phrases as representing typed arguments

The seeing event

John saw Mary eat the apple

Which entity is performing the "seeing" action? (i.e. initiating it)

What is being seen?

Semantic roles

For an event that is described in a verb, different noun phrases fulfill different semantic roles

Think of noun phrases as representing typed arguments

The eating event

John saw Mary eat the apple

Semantic roles

For an event that is described in a verb, different noun phrases fulfill different semantic roles

Think of noun phrases as representing typed arguments

The eating event

John saw Mary eat the apple

Which entity is performing the "eating"?

What is being eaten?

Semantic role labeling

Loosely speaking, the task of identifying *who does what to whom, when where and why*

Semantic role labeling

Loosely speaking, the task of identifying *who does what to whom, when where and why*

Input: A sentence and a verb

Output: A list of labeled spans

- Spans represent the arguments that participate in the event
- The labels represent the semantic role of each argument

- Optionally, also label the verb with a *frame type* that describes the

Semantic role labeling

Loosely speaking, the task of identifying *who does what to whom, when where and why*

Variants exist, but for simplicity we will use this setting

Input: A sentence and a verb

Output: A list of labeled spans

- Spans represent the arguments that participate in the event
- The labels represent the semantic role of each argument

- Optionally, also label the verb with a *frame type* that describes the

What is the set of labels?

We want the labels to be participants in event frames

- That is, the semantic arguments of events

Coming up with a closed set of labels can be daunting

What is the set of labels?

We want the labels to be participants in event frames

– That is, the semantic arguments of events

Coming up with a closed set of labels can be daunting

Some examples:

Semantic role	Description	Example
`Agent`	The entity who initiates an event	**John** cut an apple with a knife
`Patient`	The entity who undergoes a change of state	John cut an **apple** with a knife
`Instrument`	The means/intermediary used to perform the action	John cut an apple with a **knife**
`Location`	The location of the event	John placed an apple **on the table**

What is the set of labels?

We want the labels to be participants in event frames

- That is, the semantic arguments of events

Coming up with a closed set of labels can be daunting

Some examples (not nearly complete!):

Semantic role	Description	Example
`Agent`	The entity who initiates an event	**John** cut an apple with a knife
`Patient`	The entity who undergoes a change of state	John cut an **apple** with a knife
`Instrument`	The means/intermediary used to perform the action	John cut an apple with a **knife**
`Location`	The location of the event	John placed an apple **on the table**

Two styles of labels commonly seen

- FrameNet [Fillmore et al]
  - Labels are fine-grained semantic roles based on the theory of Frame Semantics
    - e.g. `Agent`, `Patient`, `Instrument`, `Location`, `Beneficiary`, etc
  - More a lexical resource than a corpus
    - Each semantic frame associated with exemplars
- PropBank [Palmer et al]
  - Labels are theory neutral but defined on a verb-by-verb basis
    - More abstract labels: e.g. `Arg0`, `Arg1`, `Arg2`, `Arg-Loc`, etc.
  - An annotated corpus
    - The Wall Street Journal part of the Penn Treebank

FrameNet and PropBank: Examples

Jack ***bought*** a glove from Mary.

Jack ***acquired*** a glove from Mary.

Jack ***returned*** a glove to Mary.

FrameNet and PropBank: Examples

Jack ***bought*** a glove from Mary.
Buyer Goods Seller

COMMERCE_GOODS_TRANSFER
frame

Jack ***acquired*** a glove from Mary.
Recipient Theme Source

ACQUIRE
frame

Jack ***returned*** a glove to Mary.
Agent Theme Recipient

FrameNet frame elements

FrameNet and PropBank: Examples

Jack ***bought*** a glove from Mary.

`Arg0` `Arg1` `Arg2`

Jack ***acquired*** a glove from Mary.

`Arg0` `Arg1` `Arg2`

Jack ***returned*** a glove to Mary.

`Arg0` `Arg1` `Arg2`

PropBank labels. The interpretation of these labels depends on the verb

Slides by Vivek Srikumar

Semantic Role Labeling

- Mostly based on PropBank [Palmer et. al. 05]
  - Large human-annotated corpus of verb semantic relations
- The task: To predict arguments of verbs

Given a sentence, identifies who does what to whom, where and when.

The bus was ***heading*** for Nairobi in Kenya

Semantic Role Labeling

- Mostly based on PropBank [Palmer et. al. 05]
  - Large human-annotated corpus of verb semantic relations
- The task: To predict arguments of verbs

Given a sentence, identifies who does what to whom, where and when.

The bus was ***heading*** for Nairobi in Kenya

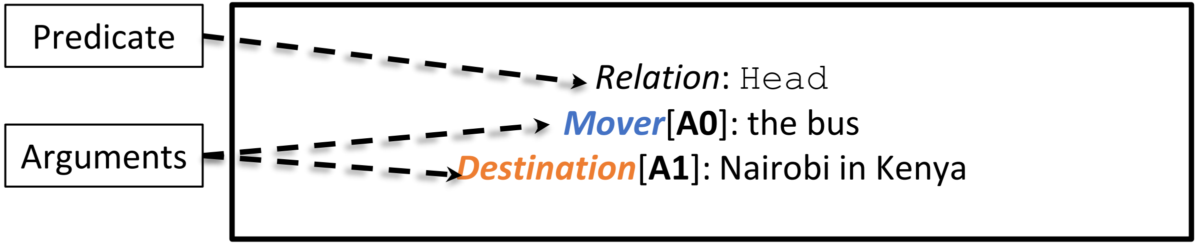
Relation: Head
Mover[**A0**]: the bus
Destination[**A1**]: Nairobi in Kenya

Semantic Role Labeling

- Mostly based on PropBank [Palmer et. al. 05]
  - Large human-annotated corpus of verb semantic relations
- The task: To predict arguments of verbs

Given a sentence, identifies who does what to whom, where and when.

The bus was ***heading*** for Nairobi in Kenya



Predicting verb arguments

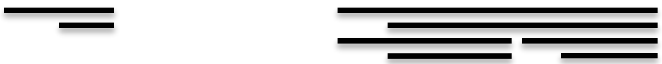
A state-of-the-art pre-neural network approach

The bus was **heading** for Nairobi in Kenya.

Predicting verb arguments

A state-of-the-art pre-neural network approach

The bus was **heading** for Nairobi in Kenya.



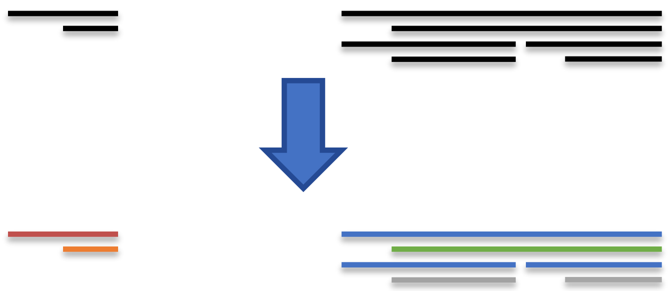
1. Identify candidate arguments for verb using parse tree
   – Filtered using a binary classifier

Predicting verb arguments

A state-of-the-art pre-neural network approach

1. **Identify** candidate arguments for verb using parse tree
   - Filtered using a binary classifier
2. **Classify** argument candidates
   - Multi-class classifier (one of multiple labels per candidate)

The bus was **heading** for Nairobi in Kenya.

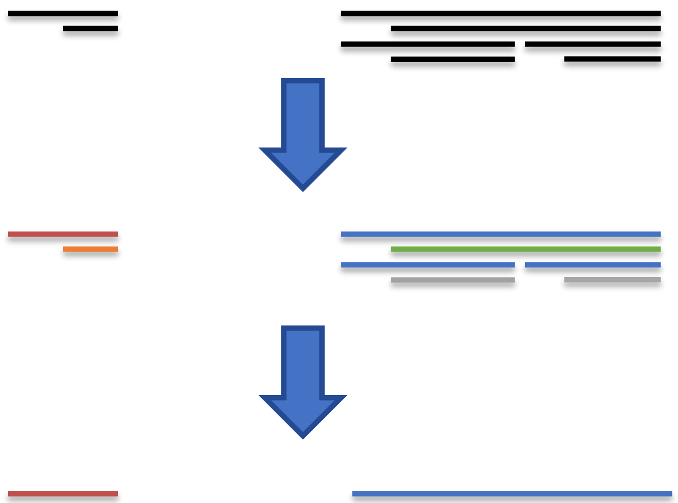


Predicting verb arguments

A state-of-the-art pre-neural network approach

1. Identify candidate arguments for verb using parse tree
   – Filtered using a binary classifier
2. Classify argument candidates
   – Multi-class classifier (one of multiple labels per candidate)
3. Inference
   – Using probability estimates from argument classifier
   – Must respect structural and linguistic constraints
      - Eg: No overlapping arguments

The bus was **heading** for Nairobi in Kenya.



Slides by Vivek Srikumar

How well did these perform?

- Shared tasks and evaluations based on PropBank
  - F1 scores across all labels
  - [Toutanova et al. 2005-2008]: 80.3
  - [Punyakanok et al. 2005-2008]: 79.4
  - [Täckström et al 2015]: 79.9

~10 years, nearly no change in numbers!!

How well did these perform?

- Shared tasks and evaluations based on PropBank
  - F1 scores across all labels
  - [Toutanova et al. 2005-2008]: 80.3
  - [Punyakanok et al. 2005-2008]: 79.4
  - [Täckström et al 2015]: 79.9

- Common characteristics of these approaches
  - Rich features
  - Used an ensemble of classifiers
  - Used some way to integrate multiple multi-class decisions
    - Either only at prediction time or at both training time and when the model is used

Why is this problem hard?

Encompasses a wide variety of linguistic phenomena

- Accounts for prepositional phrase attachment

John frightened *the raccoon with a big tail.*

Arg 0 **Arg 1**

John frightened *the raccoon* with a big stick.

Arg 0 **Arg 1**

Why is this problem hard?

Encompasses a wide variety of linguistic phenomena

- The dependencies can be very far away

John <u>frightened</u> *the raccoon*.

John walked quietly and <u>frightened</u> *the raccoon*.

John walked quietly into the garden and <u>frightened</u> *the raccoon*.

In all three cases, *John* is the **`Arg0`** of frightened….
…but it can be far away from the verb.

Slides by Vivek Srikumar

Why is this problem hard?

Encompasses a wide variety of linguistic phenomena

- Unifies syntactic alternations

John broke *the vase*

Subject position = **Arg0**

Object position = **Arg1**

The vase broke

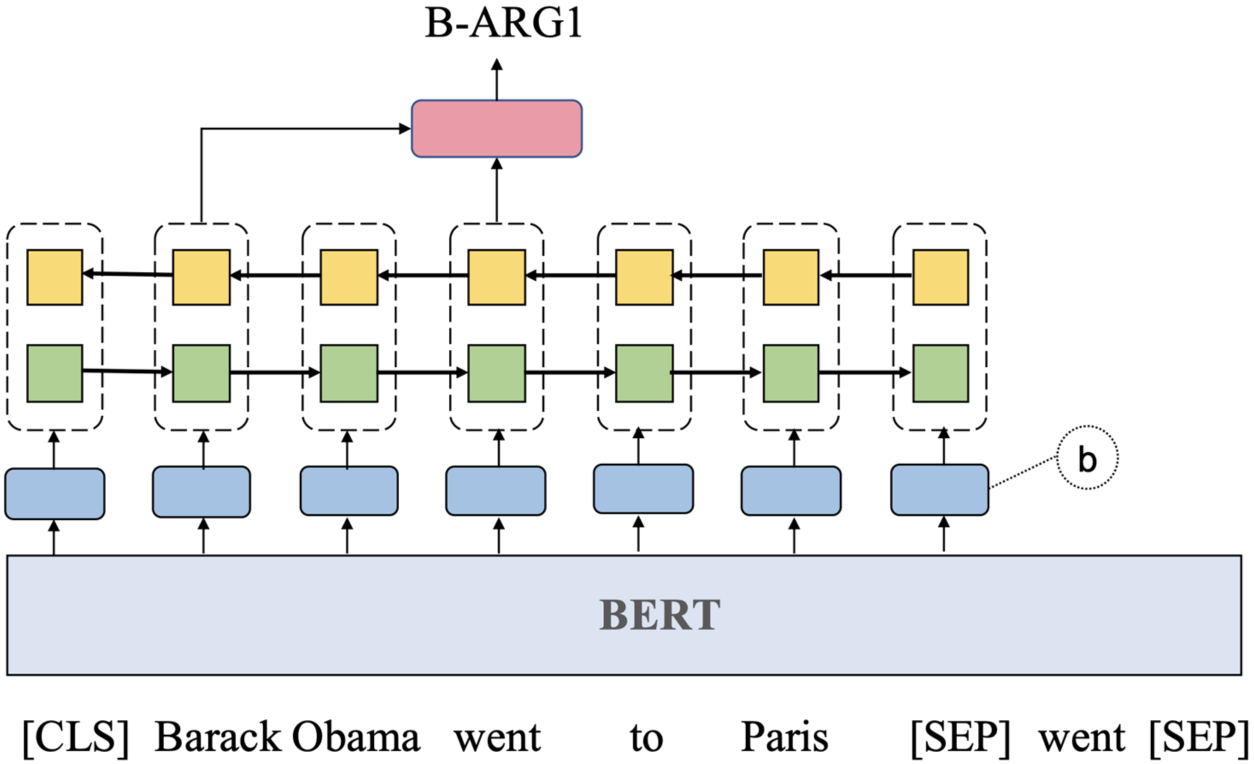
Subject position = **Arg1**

Performance

Shared tasks and evaluations based on PropBank

- F1 scores across all labels
- [Toutanova et al. 2005-2008]: 80.3
- [Punyakanok et al. 2005-2008]: 79.4
- [Täckström et al 2015]: 79.9
- [Fitzgerald et al 2015] (structured, product of experts): 80.3
- [He et al 2017](with product of experts): 84.6
  - No hand-designed features!

More recently in the SRL world; 88.8 F1



[Figure from: Shi and Lin, 2019]

Must in NLP: Knowing *who* is being talk about in a text

Taylor and Morgan went to a conference in Seattle. **Taylor** was excited to unveil **her** research on marine biology, while Morgan was keen on discussing her innovations in renewable energy. At the conference, **Taylor** impressed the audience with **her** presentation, and Morgan formed valuable connections with industry leaders. In the evening, ***Taylor** and Morgan* went downtown and they enjoyed a jazz concert.

Discourse

A **discourse model** [Karttunen et al., 1969] is a mental model that the understander builds incrementally when interpreting a text, containing:

- ↠ representations of the entities referred to in the text,
- ↠ properties of the entities and relations among them.

We use **discourse** to refer to a coherent structured group of sentences that make up language

Coherence refers to the relationship between sentences that makes real discourses different than just random assemblages of sentences

Terminology

Mentions:
Linguistic expressions like “her”, “Taylor”, “Morgan”, “Taylor and Morgan”, “they”

Referent:
The discourse entity that is referred (“Taylor”, “Morgan”, “Taylor and Morgan”)

Two or more referring expressions that are used to refer to the same discourse entity are said to **corefer**

- {Taylor, her}
- {Morgan, her}
- {Taylor and Morgan, they}

Taylor and Morgan went to a conference in Seattle. **Taylor** was excited to unveil **her** research on marine biology, while Morgan was keen on discussing her innovations in renewable energy. At the conference, **Taylor** impressed the audience with **her** presentation, and Morgan formed valuable connections with industry leaders. In the evening, ***Taylor*** *and* *Morgan* went downtown and *they* enjoyed a jazz concert.

Terminology (cont.)

Anaphora:
Reference in a text to an entity that has been previously introduced into the discourse

Antecedent:
A prior mention of the entity

Singleton:
An entity that has only a single mention in a text

Taylor and Morgan went to a conference in Seattle. **Taylor** was excited to unveil **her** research on marine biology, while Morgan was keen on discussing her innovations in renewable energy. At the conference, **Taylor** impressed the audience with **her** presentation, and Morgan formed valuable connections with industry leaders. In the evening, ***Taylor* *and Morgan*** went downtown and *they* enjoyed a jazz concert.

Coreference resolution

The task of determining whether two mentions corefer (refer to the same entity in the discourse model)

Coreference chain or **cluster**:
The set of coreferring expressions

- {Taylor, her, the 24-year-old}
- {Morgan, her}
- {Taylor and Morgan, they}

Coreference resolution comprises two sub-tasks:

1. Identifying the mentions (easier)
2. Clustering them into coreference chains

Taylor and Morgan went to a conference in Seattle. **Taylor** was excited to unveil **her** research on marine biology, while Morgan was keen on discussing her innovations in renewable energy. At the conference, **Taylor** impressed the audience with **her** presentation as **the 24-year-old**, and Morgan formed valuable connections with industry leaders. In the evening, *Taylor and Morgan* went downtown and *they* enjoyed a jazz concert.