INF4820: Algorithms for Artificial Intelligence and Natural Language Processing

Context-Free Grammars

Stephan Oepen & Milen Kouylekov

Language Technology Group (LTG)

October 29, 2014

Overview

Last Time

- Sequence Labeling
- Dynamic programming
- Viterbi algorithm

Today

- Syntactic structure
  - Context-free grammar
  - Treebanks
- Basic parsing strategies
  - Bottom-up
  - Top-down

Dynamic Programming

- Dynamic programming algorithms
  - solve large problems by compounding answers from smaller sub-problems
  - record sub-problem solutions for repeated use
- They are used for complex problems that
  - can be described recursively
  - require the same calculations over and over again
- Examples:
  - Dijkstra's shortest path
  - minimum edit distance
  - longest common subsequence
  - Viterbi

Viterbi Algorithm

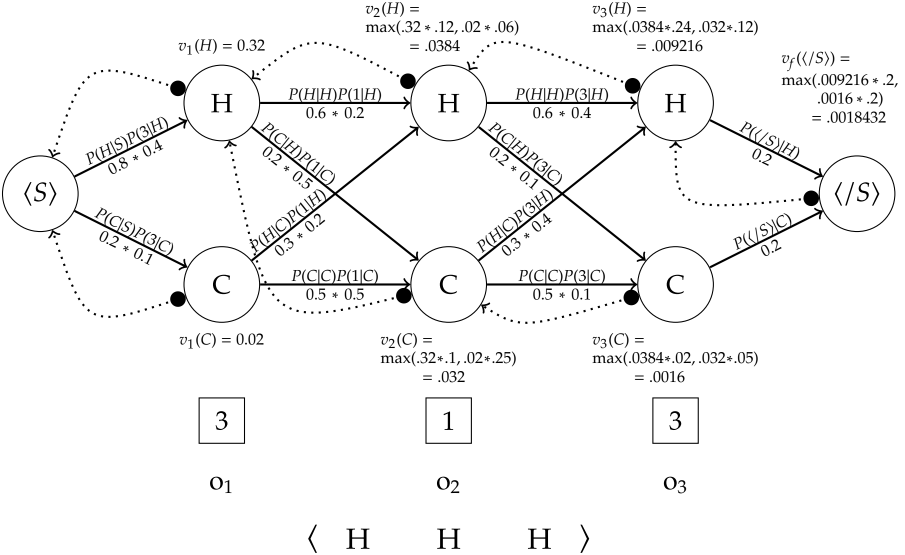
- To find the best state sequence, **maximize**:

  $P(s_1 \ldots s_n | o_1 \ldots o_n) = P(s_1|s_0)P(o_1|s_1)P(s_2|s_1)P(o_2|s_2) \ldots$
- The value we cache at each step:

$$v_i(x) = \max_{k=1}^{L} [v_{i-1}(k) \cdot P(x|k) \cdot P(o_i|x)]$$

- The variable $v_i(x)$ represents the maximum probability that the i-th state is x, given that we have seen O_1^i.
- At each step, we record backpointers showing which previous state led to the maximum probability.

An Example of the Viterbi Algorithmn



Using HMMs

The HMM models the process of generating the labelled sequence. We can use this model for a number of tasks:

- $P(S, O)$ given S and O
- **$P(O)$ given O**
- S that maximizes $P(S|O)$ given O
- $P(s_x|O)$ given O
- We can learn model parameters from a set of observations.

Computing Likelihoods

Task

Given an observation sequence O, determine the likelihood $P(O)$, according to the HMM.

Compute the **sum over all possible state sequences**:

$$P(O) = \sum_{S} P(O, S)$$

For example, the ice cream sequence 3 1 3:

$$\begin{aligned} P(3\ 1\ 3) = \quad & P(3\ 1\ 3, \text{cold cold cold}) + \\ & P(3\ 1\ 3, \text{cold cold hot}) + \\ & P(3\ 1\ 3, \text{hot hot cold}) + \ldots \end{aligned}$$

The Forward Algorithm

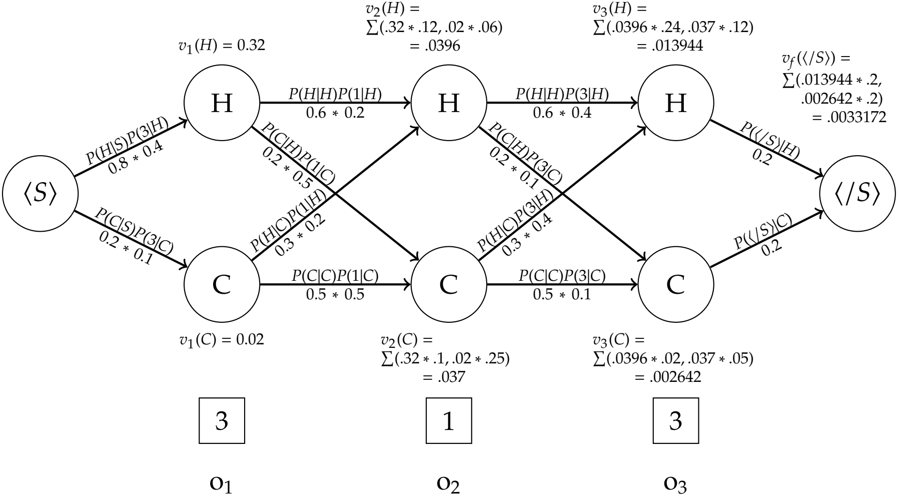
Again, we use **dynamic programming**—storing and reusing the results of partial computations in a **trellis** α.

Each cell in the trellis stores the probability of being in state x after seeing the first i observations:

$$\begin{aligned}
\alpha_i(x) &= P(o_1 \ldots o_i, s_i = x) \\
&= \sum_{k=1}^{L} \alpha_{i-1}(k) \cdot P(x|k) \cdot P(o_i|x)
\end{aligned}$$

Note $\sum$, instead of the max in Viterbi.

An Example of the Forward Algorithmn

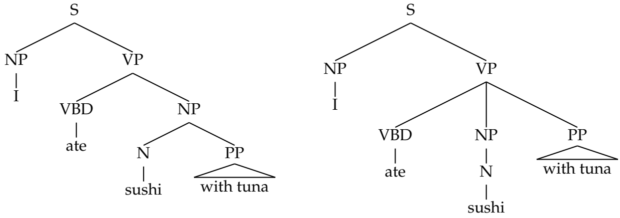


$$P(3\,1\,3) = 0.0033172$$

Today

Determining

- which string is most likely: ✓
  - *How to recognize speech* vs. *How to wreck a nice beach*
- which tag sequence is most likely for *flies like flowers*: ✓
  - **NNS VB NNS** vs. **VBZ P NNS**
- which syntactic structure is most likely:



From Linear Order to Hierarchical Structure

- The models we have looked at so far:
  - *n*-gram models (Markov chains).
    - Purely linear (sequential) and surface oriented.
  - sequence labeling: HMMs.
    - Adds one layer of abstraction: PoS as hidden variables.
    - Still only sequential in nature.
- **Formal grammar** adds hierarchical structure.
  - In NLP, being a sub-discipline of AI, we want our programs to *'understand'* natural language (on some level).
  - Finding the grammatical structure of sentences is an important step towards 'understanding'.
  - Shift focus from *sequences* to *syntactic structures*.

Why We Need Structure (1/3)

Constituency

- Words tends to lump together into groups that behave like single units: we call them *constituents*.
- *Constituency tests* give evidence for constituent structure:
  - interchangeable in similar syntactic environments.
  - can be co-ordinated
  - can be moved within a sentence as a unit

(1) Kim read [a very interesting book about grammar]$_{NP}$.
Kim read [it]$_{NP}$.

(2) Kim [read a book]$_{VP}$, [gave it to Sandy]$_{VP}$, and [left]$_{VP}$.

(3) You said I should read the book and [read it]$_{VP}$ I did.

Examples from *Linguistic Fundamentals for NLP: 100 Essentials from Morphology and Syntax.* Bender (2013)

Why We Need Structure (2/3)

Constituency

- Constituents are theory-dependent, and are not absolute or language-independent.
- Language word order is often described in terms of constituents, and word order may be more or less free within constituents or between them.
- A constituent usually has a *head* element, and is often named according to the type of its head:
  - A noun phrase (NP) has a nominal (noun-type) head:

    (4) [a very interesting <u>book</u> about grammar]$_{NP}$

  - A verb phrase (VP) has a verbal head:

    (5) [<u>gives</u> books to students]$_{VP}$

Why We Need Structure (3/3)

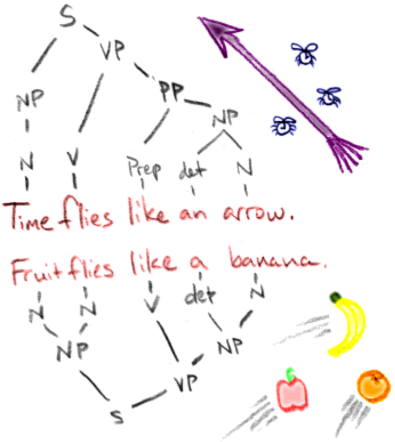
Grammatical functions

- Terms such as subject and object describe the grammatical function of a constituent in a sentence.
- *Agreement* is generally feature of the relationship between grammatical features.

  The *decision* of the Nobel committee members surprises most of us.

- Why would a purely linear model have problems predicting this phenomenon?
- Verb agreement reflects the grammatical structure of the sentence, not just the sequential order of words.

Syntactic Ambiguity



(Courtesy of the *Speculative Grammarian, –the journal of satirical linguistics*.)

Grammars: A Tool to Aid Understanding

Formal grammars describe a language, giving us a way to:

- judge or predict well-formedness

  Kim was happy because ________ passed the exam.

  Kim was happy because ________ final grade was an A.

- make explicit structural ambiguities

  Have her report on my desk by Friday!

  I like to eat sushi with { chopsticks | tuna }.

- derive abstract representations of meaning

  Kim gave Sandy a book.

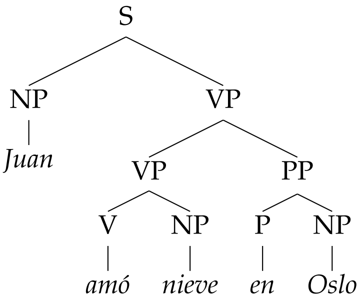
  Kim gave a book to Sandy.

  Sandy was given a book by Kim.

A Grossly Simplified Example

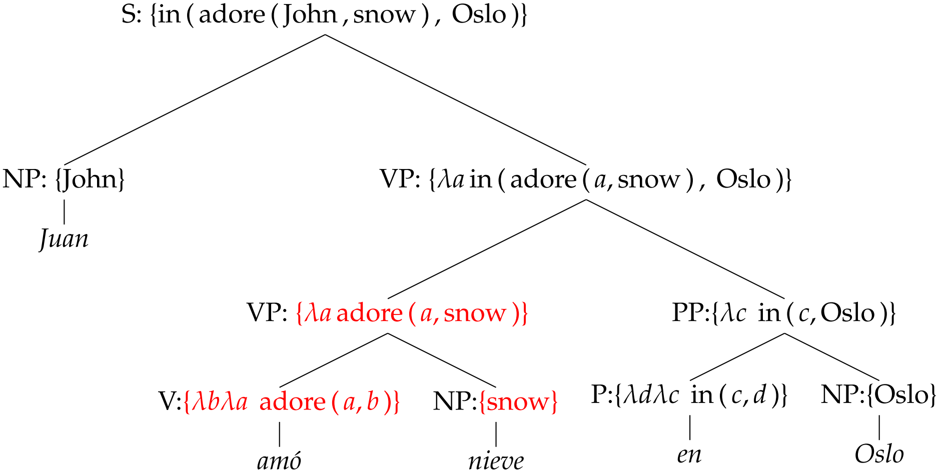
The Grammar of Spanish

S → NP VP	$\{\,\mathrm{VP}\,(\,\mathrm{NP}\,)\,\}$
VP → V NP	$\{\,\mathrm{V}\,(\,\mathrm{NP}\,)\,\}$
VP → VP PP	$\{\,\mathrm{PP}\,(\,\mathrm{VP}\,)\,\}$
PP → P NP	$\{\,\mathrm{P}\,(\,\mathrm{NP}\,)\,\}$
NP → “nieve”	{ snow }
NP → “Juan”	{ John }
NP → “Oslo”	{ Oslo }
V → “amó”	$\{\,\lambda b \lambda a\ \mathrm{adore}\,(\,a, b\,)\,\}$
P → “en”	$\{\,\lambda d \lambda c\ \mathrm{in}\,(\,c, d\,)\,\}$



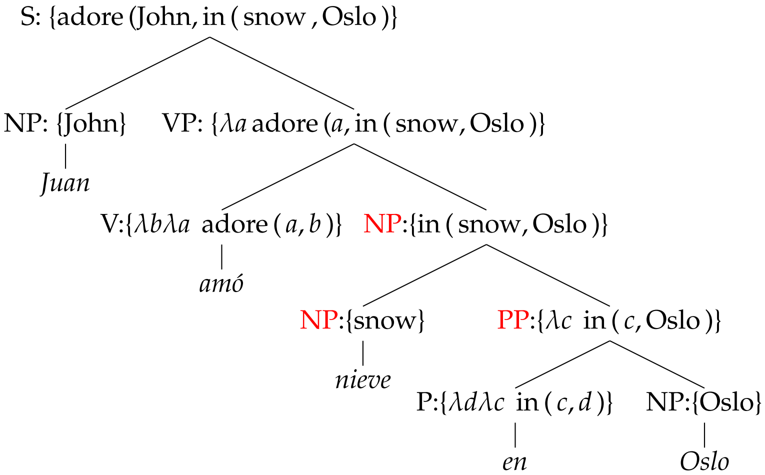
Juan amó nieve en Oslo

Meaning Composition (Still Grossly Simplified)



$$\text{VP} \rightarrow \text{V NP} \qquad \{\, \text{V}\,(\,\text{NP}\,)\,\}$$

Another Interpretation



$$\text{NP} \rightarrow \text{NP PP} \;\; \{ \text{PP}(\text{NP}) \}$$

Context Free Grammars (CFGs)

- Formal system for modeling constituent structure.
- Defined in terms of a lexicon and a set of rules
- Formal models of 'language' in a broad sense
  - natural languages, programming languages, communication protocols, . . .
- Can be expressed in the 'meta-syntax' of the Backus-Naur Form (BNF) formalism.
  - When looking up concepts and syntax in the Common Lisp HyperSpec, you have been reading (extended) BNF.
- Powerful enough to express sophisticated relations among words, yet in a computationally tractable way.

CFGs (Formally, this Time)

Formally, a CFG is a quadruple: $G = \langle C, \Sigma, P, S \rangle$

- C is the set of categories (aka *non-terminals*),
  - $\{\mathrm{S}, \mathrm{NP}, \mathrm{VP}, \mathrm{V}\}$
- Σ is the vocabulary (aka *terminals*),
  - $\{\mathrm{Kim}, \mathrm{snow}, \mathrm{adores}, \mathrm{in}\}$
- P is a set of category rewrite rules (aka *productions*)

$$
\begin{array}{ll}
\mathrm{S} \rightarrow \mathrm{NP\ VP} & \mathrm{NP} \rightarrow \mathrm{Kim} \\
\mathrm{VP} \rightarrow \mathrm{V\ NP} & \mathrm{NP} \rightarrow \mathrm{snow} \\
 & \mathrm{V} \rightarrow \mathrm{adores}
\end{array}
$$

- $S \in C$ is the *start symbol*, a filter on complete results;
- for each rule $\alpha \rightarrow \beta_1, \beta_2, ..., \beta_n \in P$: $\alpha \in C$ and $\beta_i \in C \cup \Sigma$

Generative Grammar

Top-down view of generative grammars:

- For a grammar G, the language $\mathcal{L}_G$ is defined as the set of strings that can be derived from S.
- To derive w_1^n from S, we use the rules in P to recursively rewrite S into the sequence w_1^n where each $w_i \in \Sigma$
- The grammar is seen as **generating** strings.
- *Grammatical* strings are defined as strings that can be generated by the grammar.
- The 'context-freeness' of CFGs refers to the fact that we rewrite non-terminals without regard to the overall context in which they occur.

Treebanks

Generally

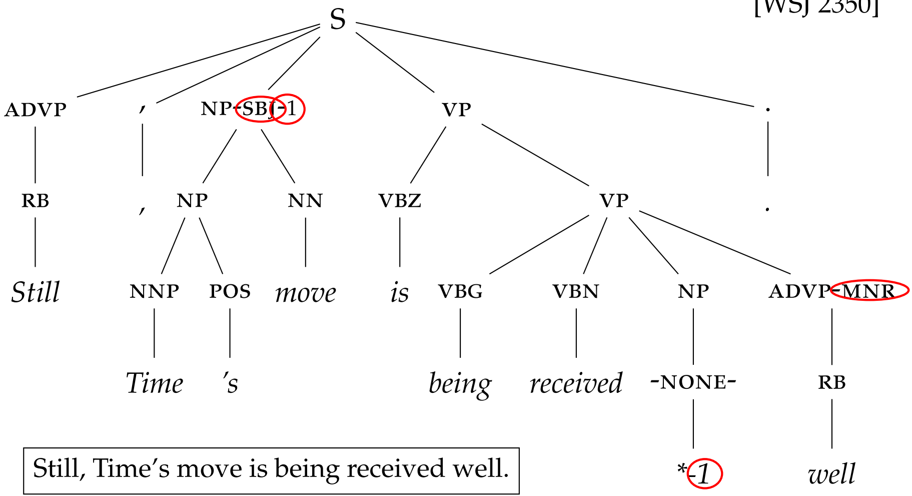
- A *treebank* is a corpus paired with 'gold-standard' (syntactic) analyses
- Can be created by manual annotation or selection among outputs from automated processing (plus correction).

Penn Treebank (Marcus et al., 1993)

- About one million tokens of Wall Street Journal text
- Hand-corrected PoS annotation using 45 word classes
- Manual annotation with (somewhat) coarse constituent structure

One Example from the Penn Treebank

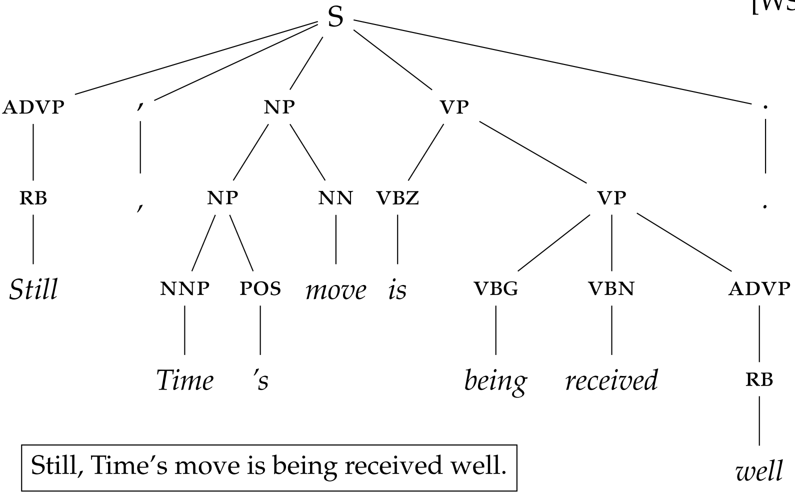
[WSJ 2350]



Still, Time's move is being received well.

Elimination of Traces and Functions

[WSJ 2350]



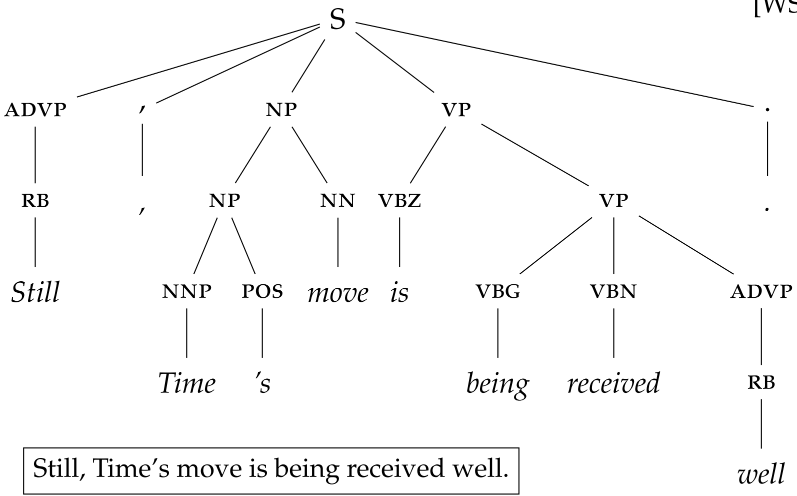
Still, Time's move is being received well.

Probabilitic Context-Free Grammars

- We are interested, not just in which trees apply to a sentence, but also to which tree is **most likely**.
- Probabilistic context-free grammars (PCFGs) augment CFGs by adding probabilities to each production, e.g.
  - S → NP VP 0.6
  - S → NP VP PP 0.4
- These are conditional probabilities — the probability of the right hand side (RHS) given the left hand side (LHS)
  - $P(S \rightarrow NP\ VP) = P(NP\ VP|S)$
- We can learn these probabilities from a treebank, again using Maximum Likelihood Estimation.

Estimating PCFGs (1/3)

[WSJ 2350]



Still, Time’s move is being received well.

Estimating PCFGs (2/3)

```
(S
   (ADVP (RB "Still"))
   (|,| ",")
   (NP
      (NP (NNP "Time") (POS "'s"))
      (NN "move"))
    (VP
      (VBZ "is")
      (VP
         (VBG "being")
         (VP
            (VBN "received")
            (ADVP (RB "well")))))
   (\. "."))
```

RB → Still	1
AVP → RB	2
|,| → ,	1
NNP → Time	1
POS → ’s	1
NP → NNP POS	1
NN → move	1
NP → NP NN	1
VBZ → is	1
VBG → being	1
VBN → received	1
RB → well	1
VP → VBN ADVP	1
VP → VBG VP	1
\\. → .	1
S → ADVP |,| NP VP \\.	1
START → S	1

Estimating PCFGs (3/3)

Once we have counts of all the rules, we turn them into probabilities.

S $\rightarrow$ ADVP |,| NP VP \\.	50	S $\rightarrow$ NP VP \\.	400
S $\rightarrow$ NP VP PP \\.	350	S $\rightarrow$ VP !	100
S $\rightarrow$ NP VP S \\.	200	S $\rightarrow$ NP VP	50

$$
\begin{aligned}
P(S \rightarrow ADVP\ |,|\ NP\ VP\ \backslash.) &\approx \frac{C(S \rightarrow ADVP\ |,|\ NP\ VP\ \backslash.)}{C(S)} \\
&= \frac{50}{1150} \\
&= 0.0435
\end{aligned}
$$