CS388: Natural Language Processing

Lecture 11: Syntax I

credit: Imgflip

Greg Durrett
TEXAS
The University of Texas at Austin

Some slides adapted from Dan Klein, UC Berkeley

Administrivia

- Mini 2 due Tuesday
- Project 1 back tomorrow
- Final project spec posted

Final Project

- Done in pairs or alone
- Compute: allocation on TACC (Maverick2). 4 1080 Ti / 2 V100 / 2 P100 per machine
- Topic: see spec for suggestions
- Proposal due October 15, in-class presentations December 3/5, final report due December 13

This Lecture

- Constituency formalism
- Context-free grammars and the CKY algorithm
- Refining grammars
- Discriminative parsers

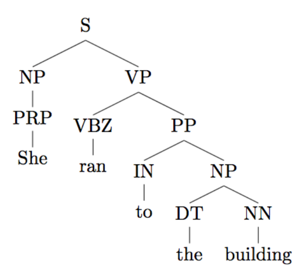
Constituency

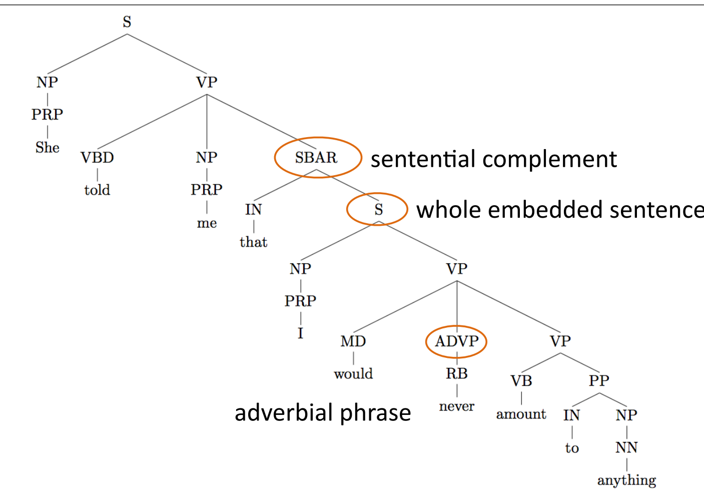
Syntax

- Study of word order and how words form sentences
- Why do we care about syntax?
  - Multiple interpretations of words (noun or verb?)
  - Recognize verb-argument structures (who is doing what to whom?)
  - Higher level of abstraction beyond words: some languages are SVO, some are VSO, some are SOV, parsing can canonicalize

Constituency Parsing

- Tree-structured syntactic analyses of sentences
- Common things: noun phrases, verb phrases, prepositional phrases
- Bottom layer is POS tags
- Examples will be in English. Constituency makes sense for a lot of languages but not all





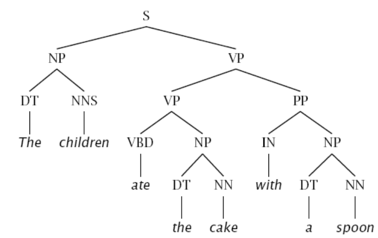
Constituency Parsing

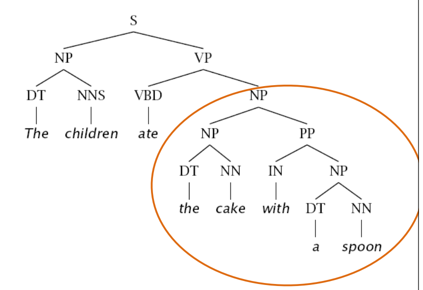
The rat the cat chased squeaked

I raced to Indianapolis , unimpeded by traffic

Challenges

- PP attachment

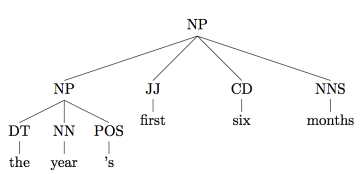


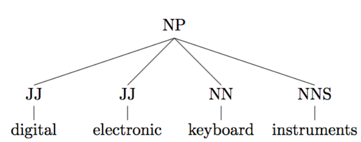


same parse as "the cake with some icing"

Challenges

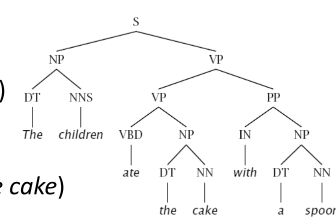
- NP internal structure: tags + depth of analysis





Constituency

- How do we know what the constituents are?
- Constituency tests:
  - Substitution by *proform* (e.g., pronoun)
  - Clefting (*It was with a spoon that...*)
  - Answer ellipsis (What did they eat? *the cake*)
    (How? *with a spoon*)



- Sometimes constituency is not clear, e.g., coordination: *she went to and bought food at the store*

Context-Free Grammars, CKY

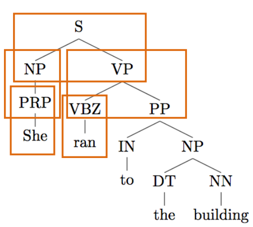
CFGs and PCFGs

Grammar (CFG)				Lexicon	
ROOT → S	1.0	NP → NP PP	0.3	NN → interest	1.0
S → NP VP	1.0	VP → VBP NP	0.7	NNS → raises	1.0
NP → DT NN	0.2	VP → VBP NP PP	0.3	VBP → interest	1.0
NP → NN NNS	0.5	PP → IN NP	1.0	VBZ → raises	1.0

- Context-free grammar: symbols which rewrite as one or more symbols
- Lexicon consists of "preterminals" (POS tags) rewriting as terminals (words)
- CFG is a tuple (N, T, S, R): N = nonterminals, T = terminals, S = start symbol (generally a special ROOT symbol), R = rules
- PCFG: probabilities associated with rewrites, normalize by source symbol

Estimating PCFGs

- Tree *T* is a series of rule applications *r*. $P(T) = \prod_{r \in T} P(r|\text{parent}(r))$



S → NP VP 1.0

NP → PRP 0.5

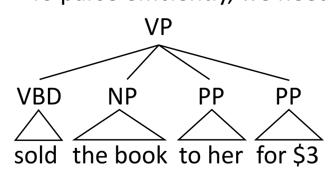
NP → DT NN 0.5

...

- Maximum likelihood PCFG: count and normalize! Same as HMMs / Naive Bayes

Binarization

- To parse efficiently, we need our PCFGs to be at most binary (not CNF)

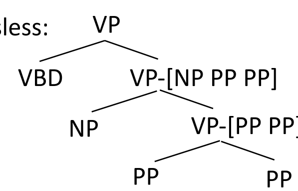


P(VP → VBD NP PP PP) = 0.2

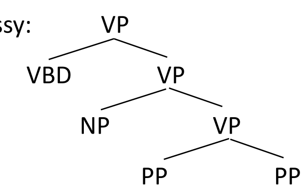
P(VP → VBZ PP) = 0.1

...

- Lossless:

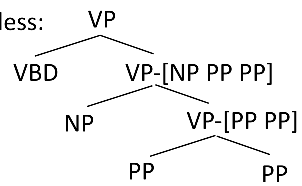


- Lossy:



Binarization

- Lossless:



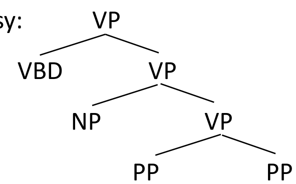
$P(VP \rightarrow VBD\ VP\text{-}[NP\ PP\ PP]) = 0.2$

$P(VP\text{-}[NP\ PP\ PP] \rightarrow NP\ VP\text{-}[PP\ PP]) = 1.0$

$P(VP\text{-}[PP\ PP] \rightarrow PP\ PP) = 1.0$

- Deterministic symbols make this the same as before

- Lossy:



$P(VP \rightarrow VBD\ VP) = 0.2$

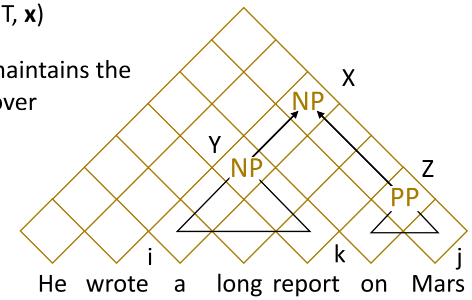
$P(VP \rightarrow NP\ VP) = 0.03$

$P(VP \rightarrow PP\ PP) = 0.001$

- Makes different independent assumptions, not the same PCFG

CKY

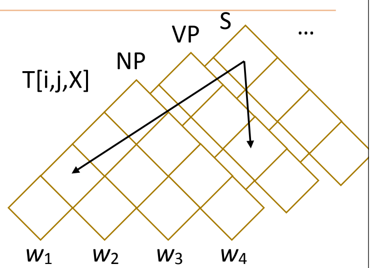
- Find argmax $P(T|\mathbf{x})$ = argmax $P(T, \mathbf{x})$
- Dynamic programming: chart maintains the best way of building symbol X over span (i, j)
- CKY = Viterbi, there is also an algorithm called inside-outside = forward-backward



Cocke-Kasami-Younger

CKY

- Chart: T[i,j,X] = best score
- Base: $T[i,i+1,X] = \log P(X \rightarrow w_i)$
- Loop over all split points k, apply rules X -> Y Z to build X in every possible way



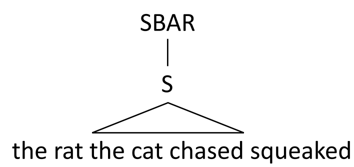
S[0,4] => NP[0,2] VP[2,4]

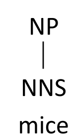
- Recurrence:

$$T[i,j,X] = \max_{k} \max_{r: X \rightarrow X1\ X2} T[i,k,X1] + T[k,j,X2] + \log P(X \rightarrow X1\ X2)$$

- Runtime: $O(n^3G)$ G = grammar constant

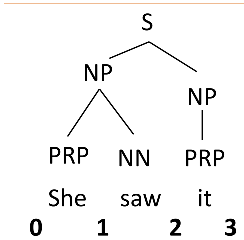
Unary Rules



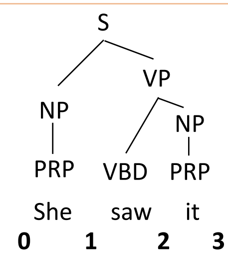


- Unary productions in treebank need to be dealt with by parsers
- Binary trees over n words have at most n-1 nodes, but you can have unlimited numbers of nodes with unaries ($S \rightarrow SBAR \rightarrow NP \rightarrow S \rightarrow \ldots$)
- In practice: enforce at most one unary over each span, modify CKY accordingly

Parser Evaluation



S(0,3),
NP(0,2),
NP(2,3),
~~PRP(0,1),~~
~~NN(1,2),~~
~~PRP(2,3)~~



S(0,3),
NP(0,1),
VP(1,3),
NP(2,3),
~~PRP(0,1),~~
~~VBD(1,2),~~
~~PRP(2,3)~~

- Precision: number of correct brackets / num pred brackets = 2/3
- Recall: number of correct brackets / num of gold brackets = 2/4
- F1: harmonic mean of precision and recall = $(1/2 * ((2/4)^{-1} + (2/3)^{-1}))^{-1}$
  = 0.57

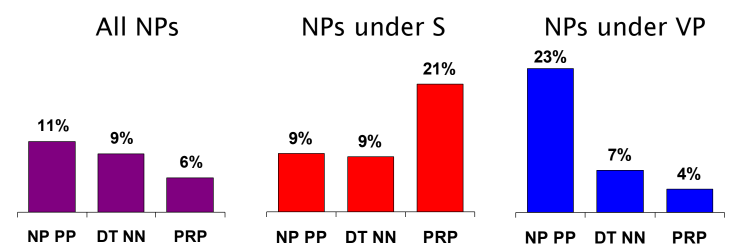
Results

- Standard dataset for English: Penn Treebank (Marcus et al., 1993)
  - Evaluation: F1 over labeled constituents of the sentence
- Vanilla PCFG: ~75 F1
- Best PCFGs for English: ~90 F1
- SOTA (discriminative models): 95 F1
- Other languages: results vary widely depending on annotation + complexity of the grammar

Klein and Manning (2003)

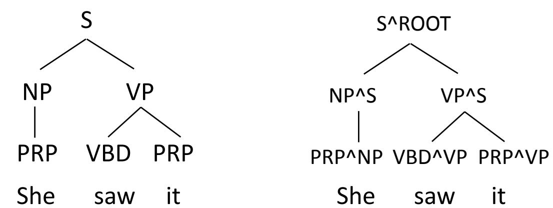
Refining Generative Grammars

PCFG Independence Assumptions



- Language is not context-free: NPs in different contexts rewrite differently
- Can we make the grammar "less context-free"?

Vertical Markovization

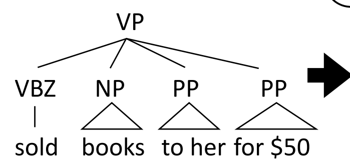


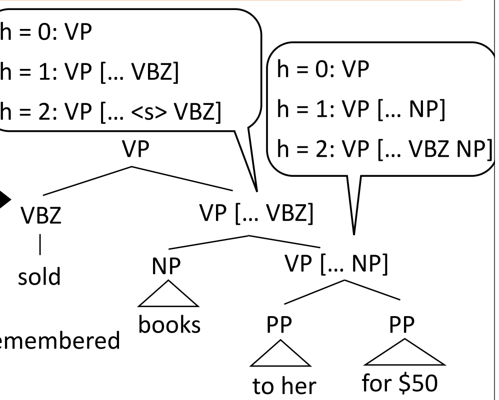
Basic tree (v = 0)

v = 1 Markovization

- Why is this a good idea?

Horizontal Markovization

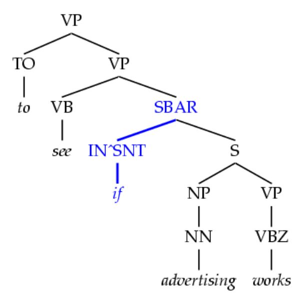




- Changes amount of context remembered in binarization process

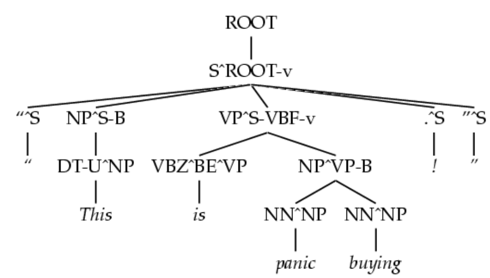
Tag Splits

- Can do some other ad hoc tag splits
- Sentential prepositions behave differently from other prepositions
- 75 F1 with basic PCFG => 86.3 F1 with a highly customized PCFG (v = 2, h = 2, other hacks like this)



Klein and Manning (2003)

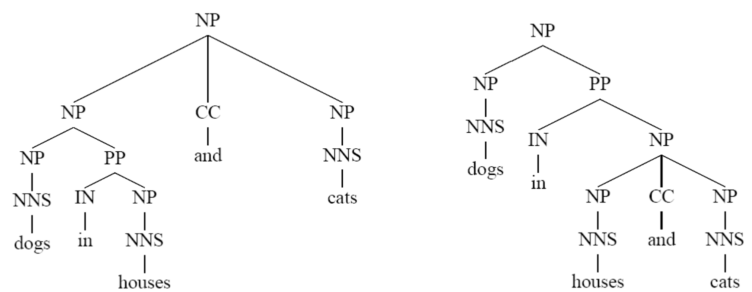
Annotated Tree



- 75 F1 with basic PCFG => 86.3 F1 with this highly customized PCFG (SOTA was 90 F1 at the time, but with more complex methods)

Klein and Manning (2003)

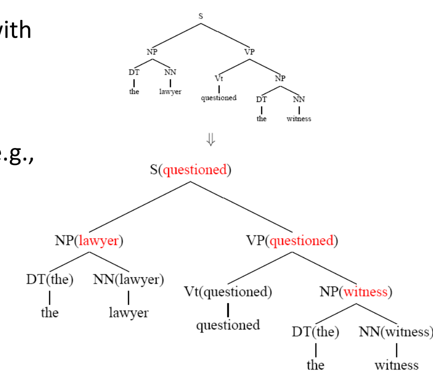
Lexicalized Parsers



- Even with parent annotation, these trees have the same rules. Need to use the words

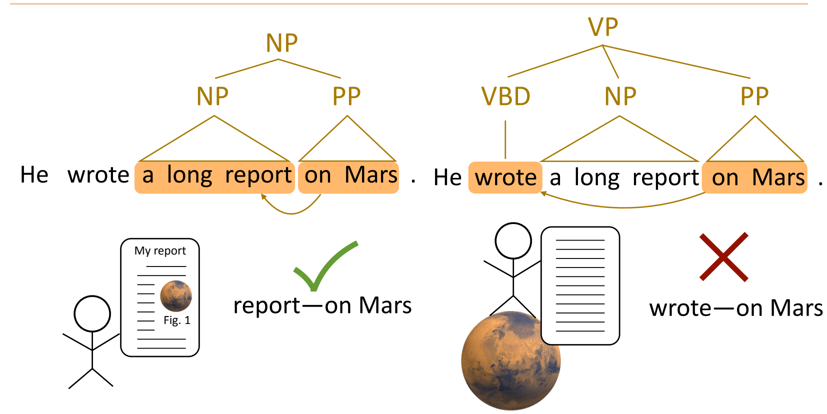
Lexicalized Parsers

- Annotate each grammar symbol with its "head word": most important word of that constituent
- Rules for identifying headwords (e.g., the last word of an NP before a preposition is typically the head)
- Collins and Charniak (late 90s): ~89 F1 with these

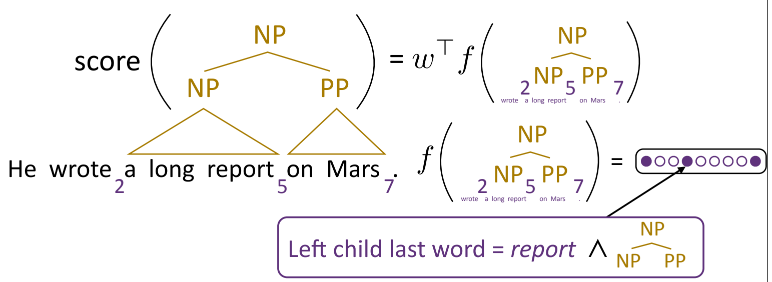


Discriminative Parsers

CRF Parsing



CRF Parsing

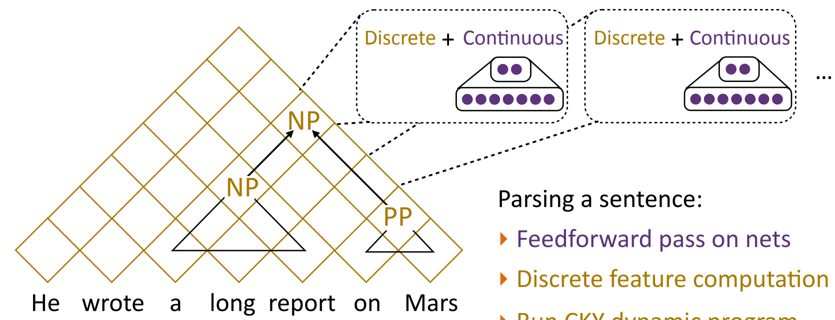


- Can learn that we *report* [PP], which is common due to *reporting on* things
- Can "neuralize" this as well like neural CRFs for NER

Taskar et al. (2004)
Hall, Durrett, and Klein (2014)
Durrett and Klein (2015)

Joint Discrete and Continuous Parsing

- Chart remains discrete!



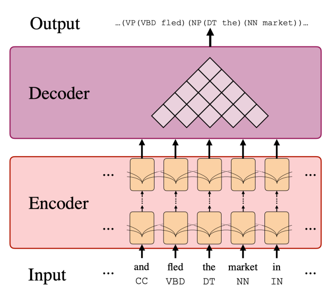
Parsing a sentence:

- Feedforward pass on nets
- Discrete feature computation
- Run CKY dynamic program

Durrett and Klein (ACL 2015)

Parsing with ELMo

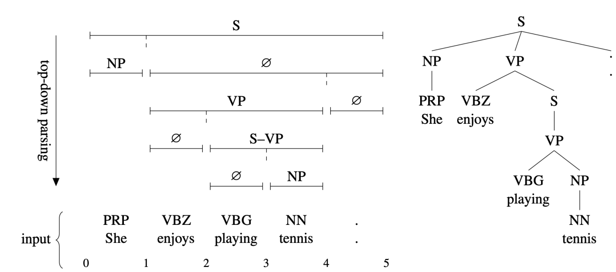
Encoder Architecture	F1 (dev)	Δ
LSTM (Gaddy et al., 2018)	92.24	-0.43
Self-attentive (Section 2)	92.67	0.00
+ Factored (Section 3)	93.15	0.48
+ CharLSTM (Section 5.1)	93.61	0.94
+ ELMo (Section 5.2)	95.21	2.54



- Improves the neural CRF by using a transformer layer (self-attentive), character-level modeling, and ELMo

Kitaev and Klein (2018)

Top-down Parsing



(a) Execution of the top-down parsing algorithm. (b) Output parse tree.

- Greedily predict bracketing at next stage of the tree. Like a neural CRF but with no dynamic program (CKY) pass

Stern et al. (2017)

Takeaways

- PCFGs estimated generatively can perform well if sufficiently engineered
- Neural CRFs work well for constituency parsing
- Next time: revisit lexicalized parsing as *dependency parsing*