Behavioral Data Mining

Lecture 17
Natural Language Processing II

Outline

- Basics: Constituency and Dependency Parsing
- Constituency Parsing: CFGs and PCFGs
- CKY parsing
- Learning/optimizing grammars

Parsing

Parsing is the process of discovering the relationships between words in sentences: how they form larger meaning-forming units, and how those units are arranged.

Parsing is fundamental to text processing:

- Moving from word-level features to more ***semantically-natural units*** (Subject-Verb-Object) through noun-phrases, verb-phrases etc.
- ***Understanding modifiers*** (adjectives and adverbs) and their ***attachments***.
- Constraining the ***meaning of a sentence***, e.g. the type of speech act the sentence represents.

Parsing

Why not?

In practice parsing is rarely used in data mining. There seem to be several reasons:

- ***Performance:*** state-of-the-art constituency parsers process tens of sentences/second on fast computers. This is simply too slow for medium/large datasets.
- ***Parse quality/error rate.*** Most parsers are trained on Treebanks of high-quality data (Mostly news articles, some books). Application corpora are often very different (e.g. Weblogs or social media) in structure.

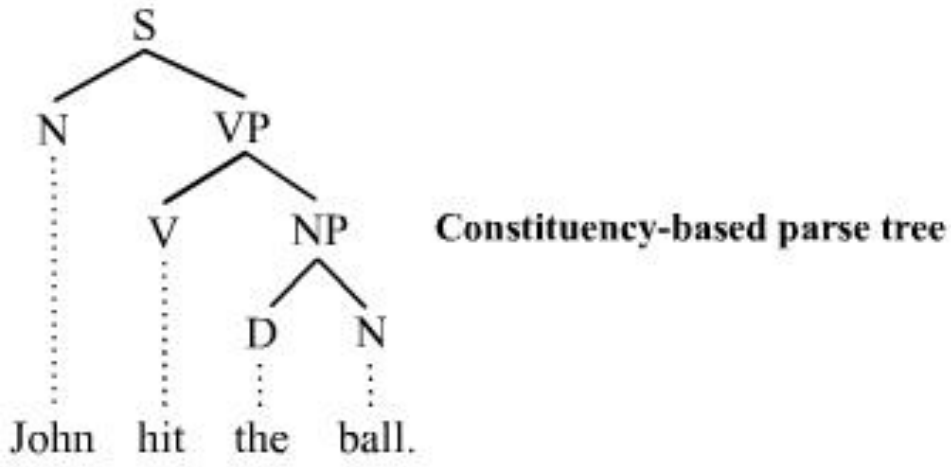
Mitigators:

- Dependency parsers are significantly faster
- New corpora such as the English Web Treebank better match informal text content.

Constituency Parsing

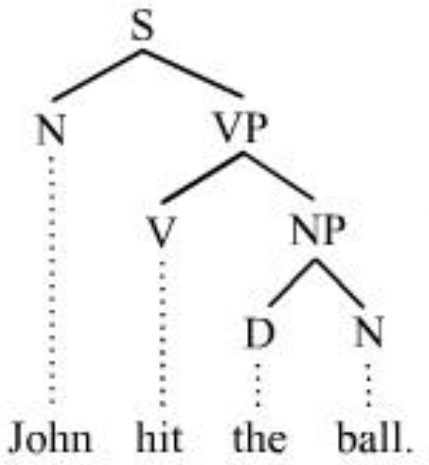
A constituency parse groups words into constituents, which are groups of (consecutive) words with a particular role.

A constituents are often defined recursively, and a constituency parse is naturally represented as a parse tree:



Because of the tree structure, symbols can be labeled as ***terminal*** (leaf nodes) or ***non-terminal*** (internal nodes).

Constituency Parsing



Constituency-based parse tree

S = sentence

N = noun

VP = verb phrase

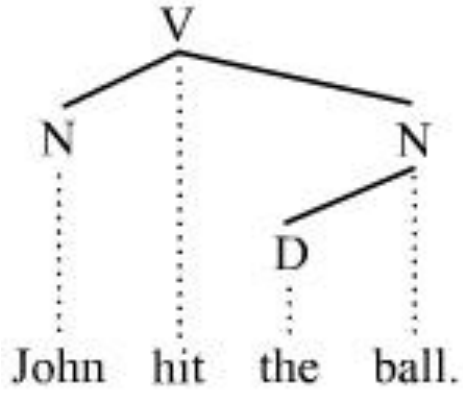
V = verb

NP = noun phrase

D = determiner

Dependency Parsing

Dependency structures are rooted in existing words, i.e. there are no non-terminal nodes, and a tree on n words has exactly n nodes.

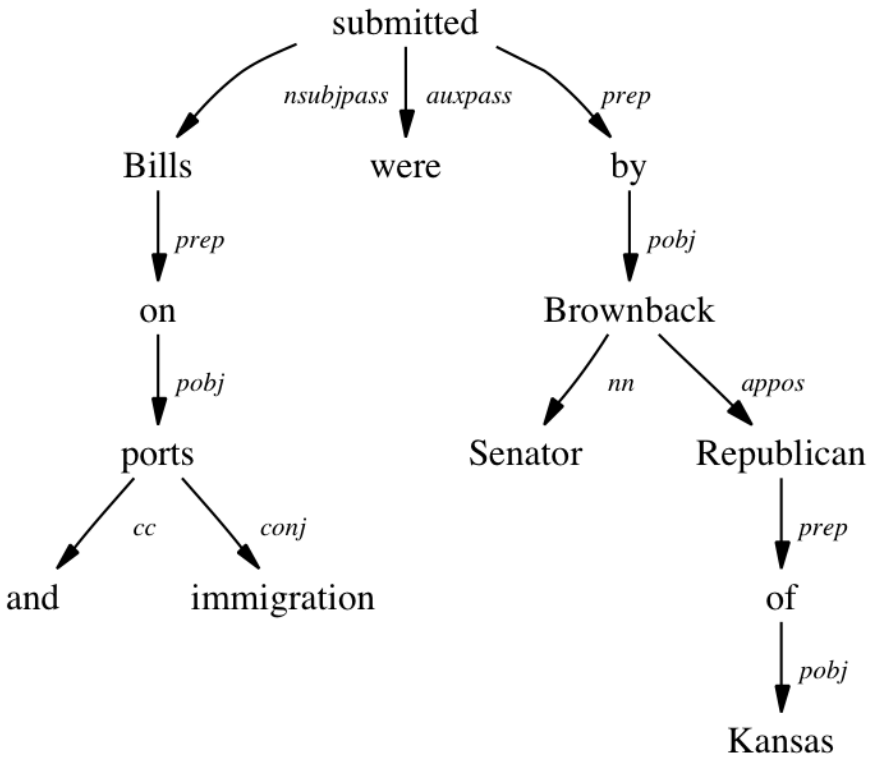


Dependency-based parse tree

A dependency tree has about half as many total nodes as a binary (Chomsky Normal form) constituency tree.

Dependency Parsing

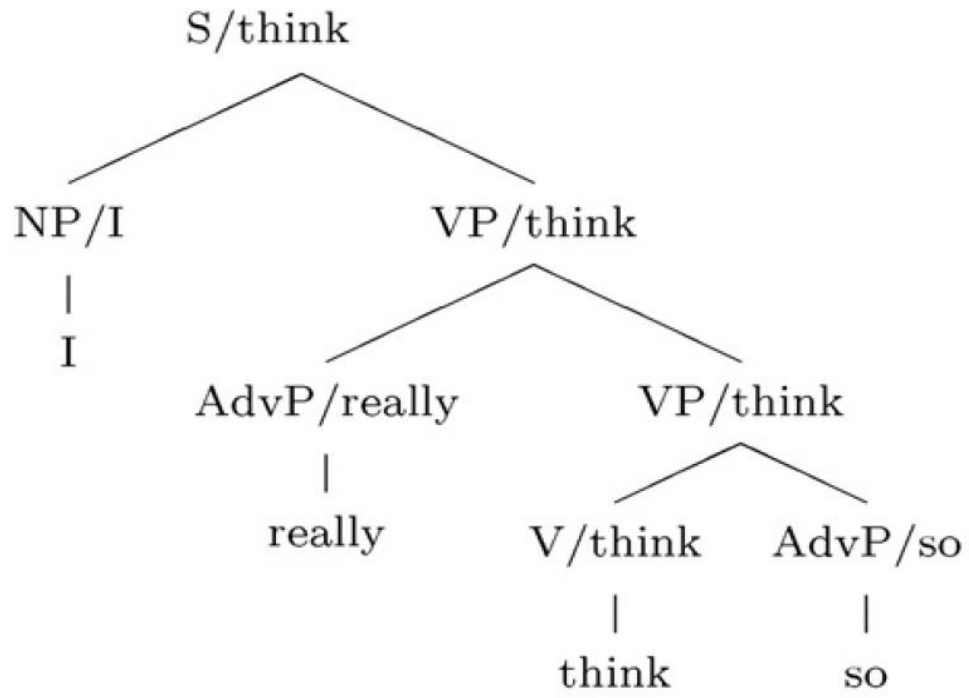
Dependency parses may be non-binary, and structure type is encoded in links rather than nodes:



Constituency vs. Dependency Grammars

Dependency parses are faster and more compact descriptions. There are not simplifications of constituency parses, although they can be derived from ***lexicalized*** constituency grammars.

Lexicalized grammars assign a ***head*** word to each non-terminal node.



Constituency vs. Dependency Grammars

- Dependency and constituency parses capture ***attachment information*** (e.g. sentiments directed at particular targets).
- Constituency parses are probably better for ***abstracting semantic features***, i.e. constituency units often correspond to semantic units.

Constituency grammars have a much large space to search and take longer, especially as sentence length grows $O(N^3)$.

However, if the goal is to find semantic groups its possible to proceed bottom-up with (small) bounded N.

Outline

- Basics: Constituency and Dependency Parsing
- Constituency Parsing: CFGs and PCFGs
- CKY parsing
- Learning/optimizing grammars

Context-Free Grammars

Context-free grammars are based on ***rules*** of the form:

$$A \rightarrow B\ C\ D$$

Where A is a non-terminal symbol, and B, C, D are non-terminals or terminal symbols (words).

The context-free descriptor implies that the rule is valid whatever words precede or follow the rule.

Binary or Chomsky Normal Form grammars include only binary productions:

$$A \rightarrow B\ C$$

And hence generate binary parse trees.

Context-Free Grammars

A context-free grammar (CFG) comprises these elements:

N: a set of ***non-terminal symbols***

Σ: a set of ***terminal symbols***

R: a set of ***rules*** or productions, of the form A $\rightarrow$ β [p]

where A is a non-terminal,

β is a string of symbols from the set $(\Sigma \cup N)^*$

S: a designated ***start symbol***.

An English Context-Free Grammar

Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

Figure 13.1 The $\mathcal{L}_1$ miniature English grammar and lexicon.

From Jurafsky and Martin "Speech and Language Processing" 2nd Ed. Prentice-Hall 2009

Probabilistic Context-Free Grammars

Context-free grammars are based on ***rules*** of the form:

$$A \rightarrow B\ C\ D$$

Where A is a non-terminal symbol, and B, C, D are non-terminals or terminal symbols (words).

The context-free descriptor implies that the rule is valid whatever words precede or follow the rule.

Binary or Chomsky Normal Form grammars include only binary productions:

$$A \rightarrow B\ C$$

And hence generate binary parse trees.

Probabilistic Context-Free Grammars

A context-free grammar (CFG) comprises these elements:

N: a set of ***non-terminal symbols***

Σ: a set of ***terminal symbols***

R: a set of ***rules*** or productions, of the form A $\rightarrow$ β [p]

where A is a non-terminal,

where β is a string of symbols from the set $(\Sigma \cup N)^*$,

and p is a number between 0 and 1 expressing p(β |A)

S: a designated ***start symbol***.

English PCFG

Grammar		Lexicon
$S \rightarrow NP\ VP$	[.80]	$Det \rightarrow that$ [.10] | a [.30] | the [.60]
$S \rightarrow Aux\ NP\ VP$	[.15]	$Noun \rightarrow book$ [.10] | $flight$ [.30]
$S \rightarrow VP$	[.05]	| $meal$ [.15] | $money$ [.05]
$NP \rightarrow Pronoun$	[.35]	| $flights$ [.40] | $dinner$ [.10]
$NP \rightarrow Proper\text{-}Noun$	[.30]	$Verb \rightarrow book$ [.30] | $include$ [.30]
$NP \rightarrow Det\ Nominal$	[.20]	| $prefer$; [.40]
$NP \rightarrow Nominal$	[.15]	$Pronoun \rightarrow I$ [.40] | she [.05]
$Nominal \rightarrow Noun$	[.75]	| me [.15] | you [.40]
$Nominal \rightarrow Nominal\ Noun$	[.20]	$Proper\text{-}Noun \rightarrow Houston$ [.60]
$Nominal \rightarrow Nominal\ PP$	[.05]	| NWA [.40]
$VP \rightarrow Verb$	[.35]	$Aux \rightarrow does$ [.60] | can [40]
$VP \rightarrow Verb\ NP$	[.20]	$Preposition \rightarrow from$ [.30] | to [.30]
$VP \rightarrow Verb\ NP\ PP$	[.10]	| on [.20] | $near$ [.15]
$VP \rightarrow Verb\ PP$	[.15]	| $through$ [.05]
$VP \rightarrow Verb\ NP\ NP$	[.05]	
$VP \rightarrow VP\ PP$	[.15]	
$PP \rightarrow Preposition\ NP$	[1.0]	

Figure 14.1 A PCFG that is a probabilistic augmentation of the $\mathcal{L}_1$ miniature English CFG grammar and lexicon of Fig. 13.1. These probabilities were made up for pedagogical purposes and are not based on a corpus (since any real corpus would have many more rules, so the true probabilities of each rule would be much smaller).

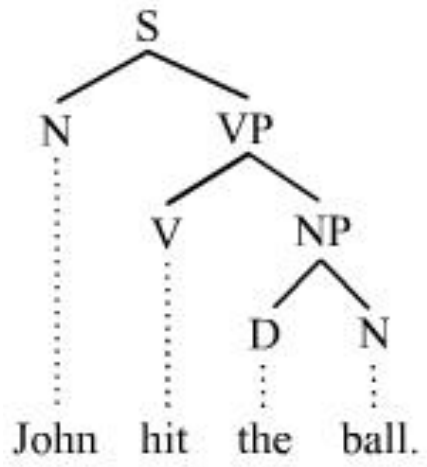
From Jurafsky and Martin "Speech and Language Processing" 2nd Ed. Prentice-Hall 2009

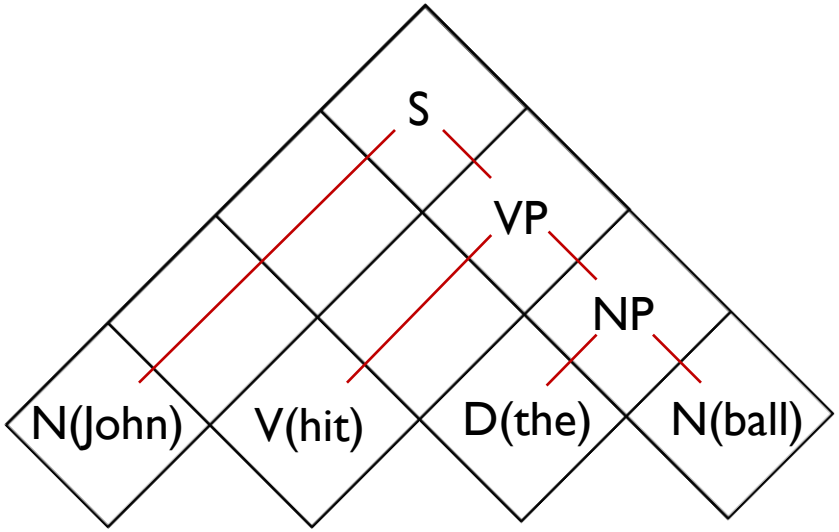
Outline

- Basics: Constituency and Dependency Parsing
- Constituency Parsing: CFGs and PCFGs
- CKY parsing
- Learning/optimizing grammars

CKY parsing

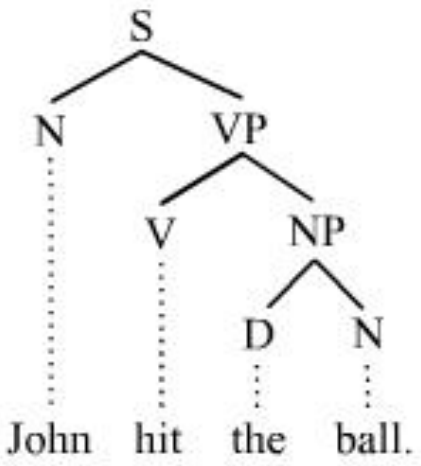
A CKY (Cocke Younger Kasami) table is a dynamic programming parser:

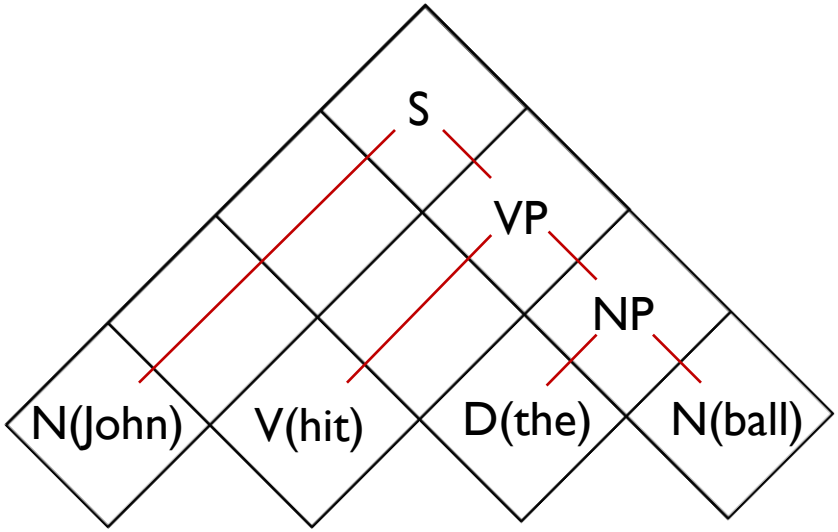




CKY parsing

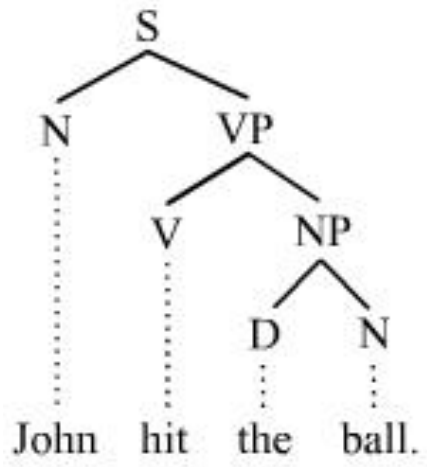
Not every internal node encodes a symbol (there are N-1 internal nodes in the tree, but $O(N^2)$ cells in the table).

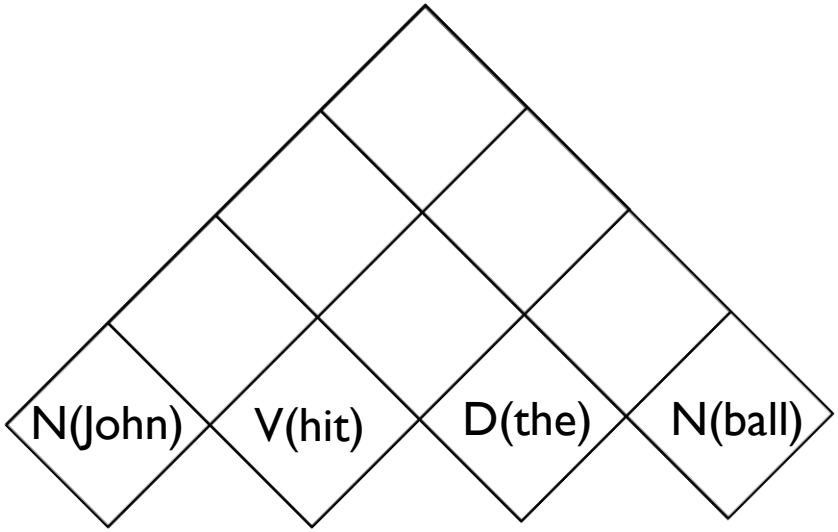




CKY parsing

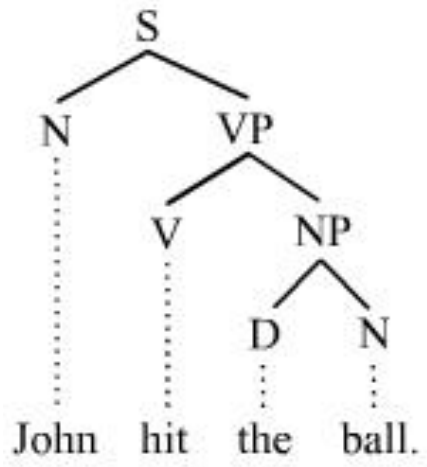
Not every internal node encodes a symbol (there are N-1 internal nodes in the tree, but $O(N^2)$ cells in the table).

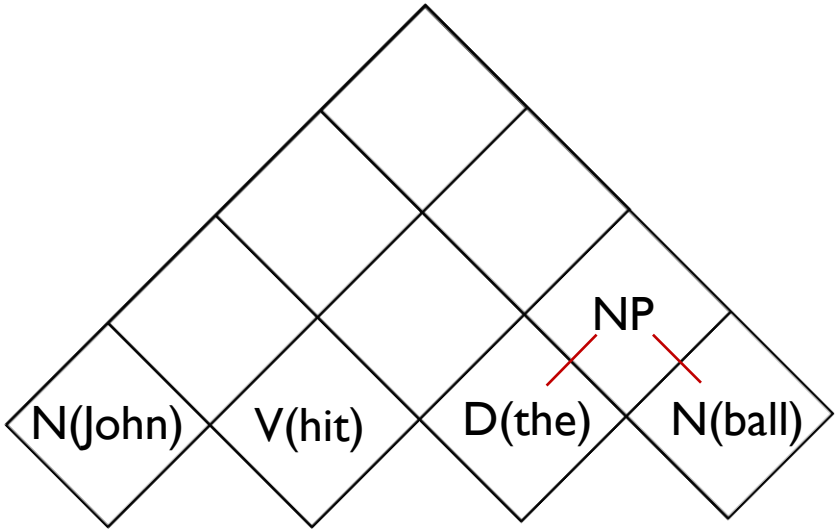




CKY parsing

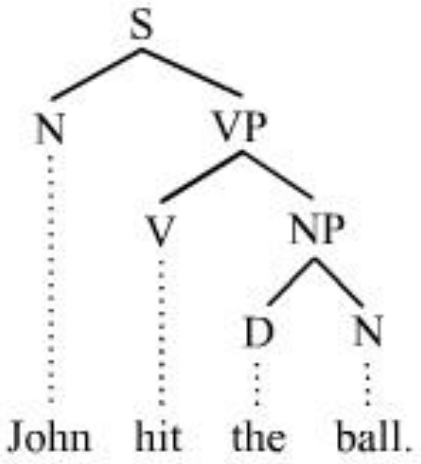
Not every internal node encodes a symbol (there are N-1 internal nodes in the tree, but $O(N^2)$ cells in the table).

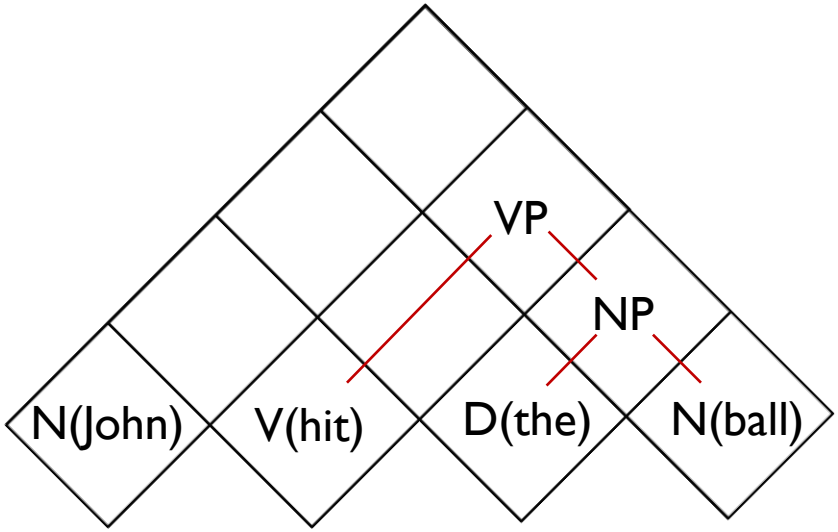




CKY parsing

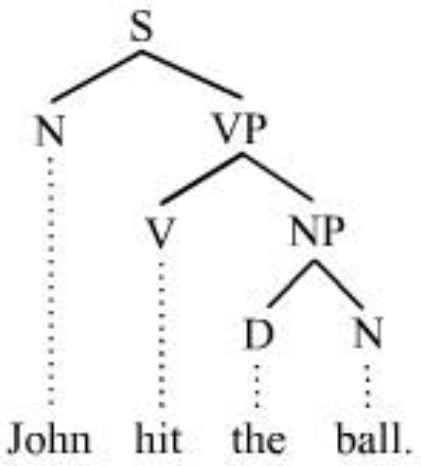
Not every internal node encodes a symbol (there are N-1 internal nodes in the tree, but $O(N^2)$ cells in the table).

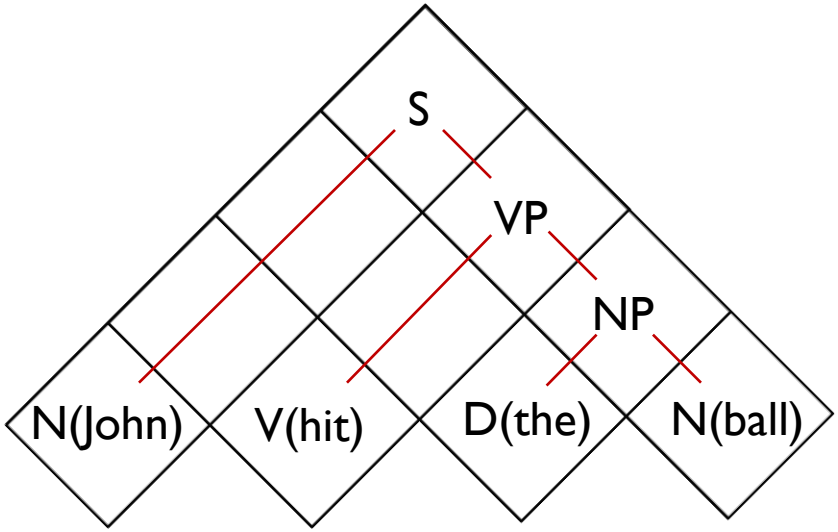




CKY parsing

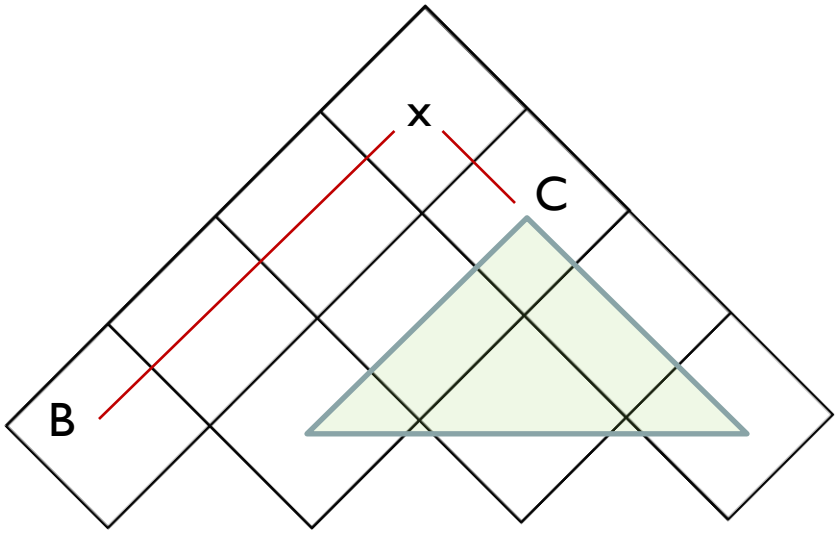
Not every internal node encodes a symbol (there are N-1 internal nodes in the tree, but $O(N^2)$ cells in the table).





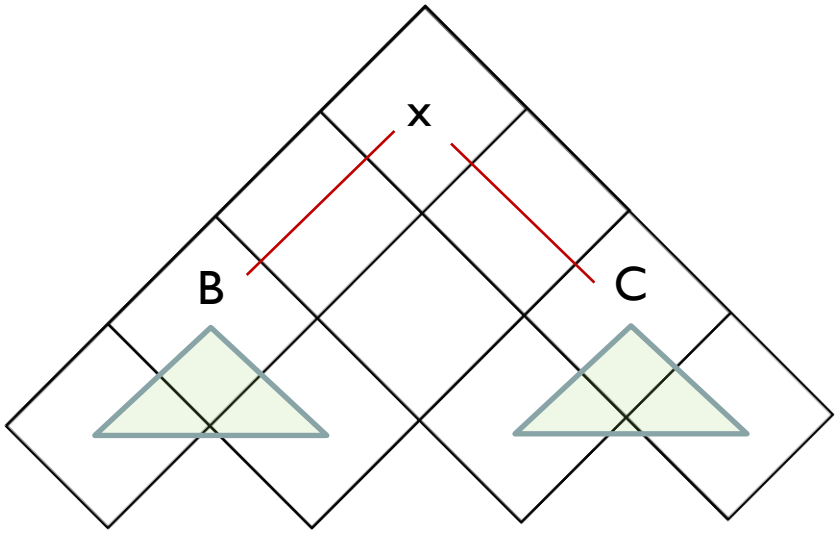
CKY parsing

For an internal node x height k, there are k possible pairs of symbols which x can decompose into:



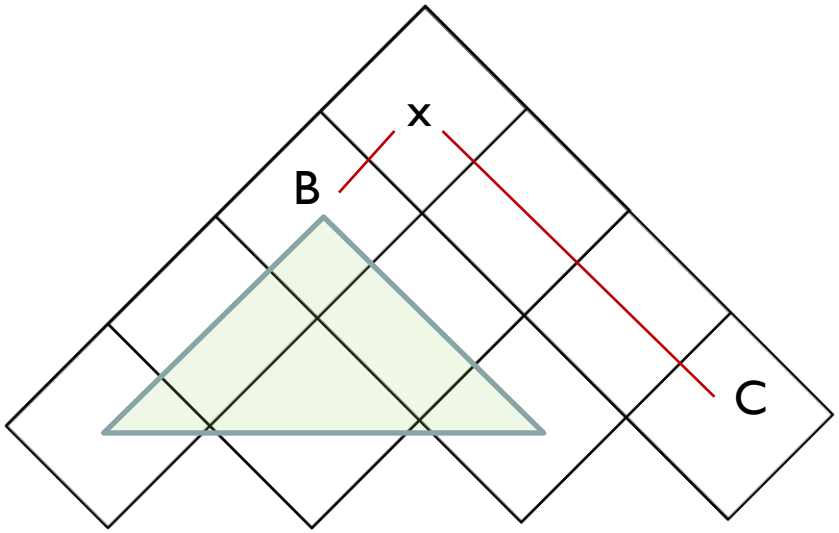
CKY parsing

For an internal node x height k, there are k possible pairs of symbols which x can decompose into:



CKY parsing

For an internal node x height k, there are k possible pairs of symbols which x can decompose into:

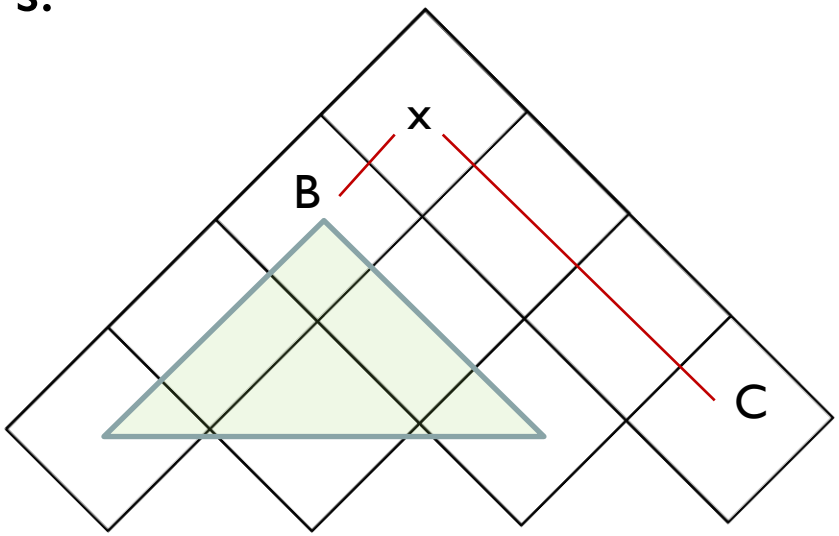


Viterbi Parse

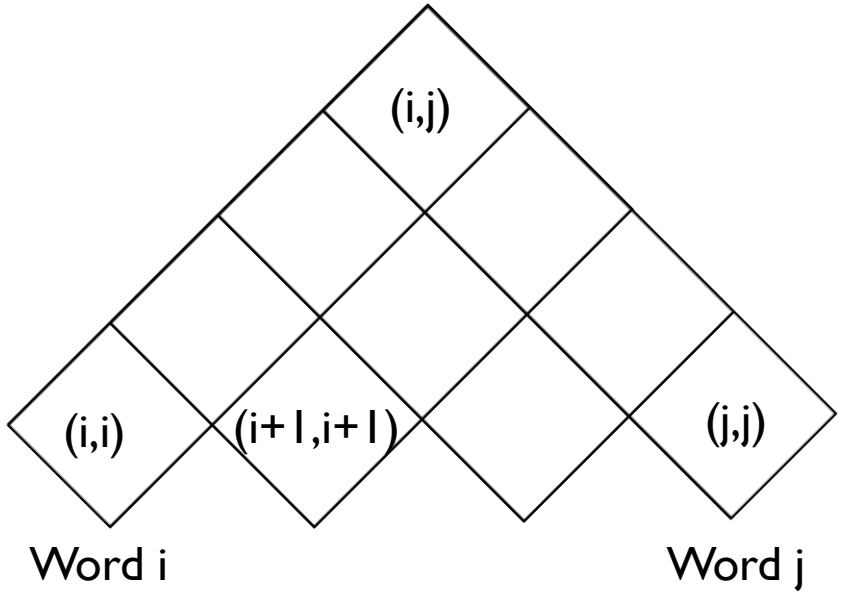
Given the probabilities of cells in the table, we can find the most probable parse using a generalized Viterbi scan.

i.e. find the most likely parse at the root, then return it, and the left and right nodes which generated it.

We can do this as part of the scoring process by maintaining a set of backpointers.

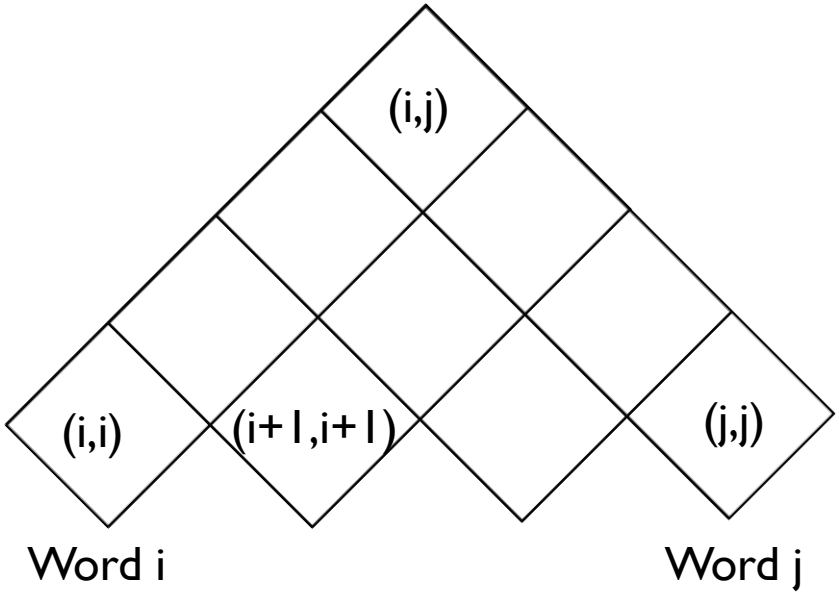


```
N = sentence length
for h = 1...N                  // height of (i,j)
  for i = 1...(N-h)            // left end of span
     j = i + h;                // right end of span
     for (A,B,C) in {rules}
       s_A(i,j) = min_score
       for k = i...(j-1)  // possible subspans
         s_A(i,j) = max(s_A(i,j),r_BC+s_B(i,k)+s_C(k+1,j))
       end
     end
  end
end
```



```
N = sentence length
for h = 1...N                  // height of (i,j)
  for i = 1...(N-h)            // left end of span
     j = i + h;                // right end of span
     for (A,B,C) in {rules}
       s_A(i,j) = min_score
       for k = i...(j-1)  // possible subspans
         s_A(i,j) = max(s_A(i,j),r_BC+s_B(i,k)+s_C(k+1,j))
       end
     end
  end
end
```

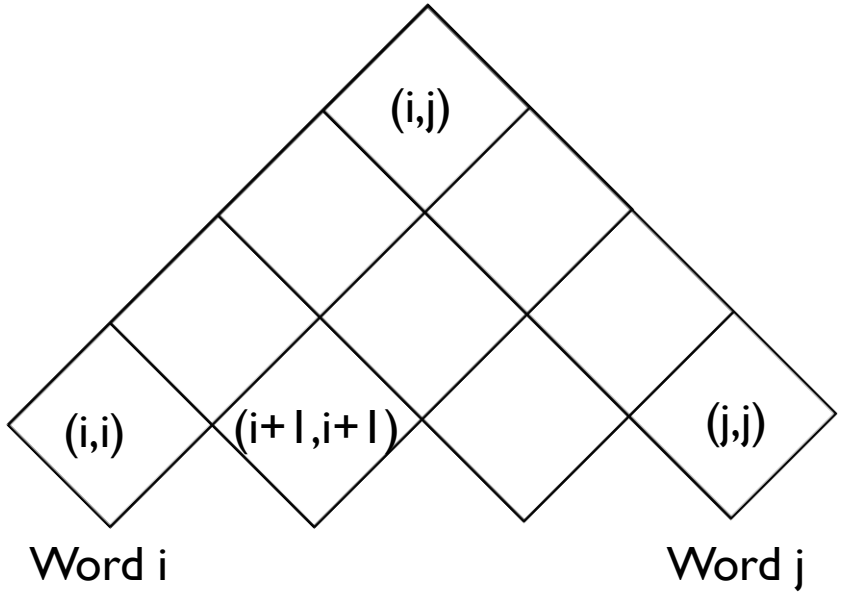
O(N³)



```
N = sentence length
for h = 1...N                  // height of (i,j)
  for i = 1...(N-h)            // left end of span
     j = i + h;                // right end of span
     for (A,B,C) in {rules}
       s_A(i,j) = min_score // score (probability)
       for k = i...(j-1)  // possible subspans
         s_A(i,j) = max(s_A(i,j),r_BC*s_B(i,k)*s_C(k+1,j))
       end
     end
  end
end
```

O(N³ R)

R = number of rules



Complexity

$O(N^3 R)$, for a typical sentence, N=30.

R for a high-quality grammar can be 1.5 million.

Actual cost of full evaluation is about $30^3 * 1.5m \approx 5\ 10^9$ flops.

i.e. a gigaflop implementation would achieve <1 sentence/sec!

There are some efficiencies that can be taken (most rule tests involve 3 non-terminal, and Berkeley grammar has only about 300k such rules), but we still need to perform about 10^9 flop/s.

A GPU solution

```
N = sentence length
for h = 1...N                  // height of (i,j)
  for i = 1...(N-h)            // left end of span
     j = i + h;                // right end of span
     for (A,B,C) in {rules}
       s_A(i,j) = min_score
       for k = i...(j-1)  // possible subspans
         s_A(i,j) = max(s_A(i,j),r_BC*s_B(i,k)*s_C(k+1,j))
       end
     end
  end
end
```

A GPU solution

```
N = sentence length
for h = 1...N                  // height of (i,j)
  for i = 1...(N-h)            // left end of span
     j = i + h;                // right end of span
     for k = i...(j-1)         // possible subspans
       for (A,B,C) in {rules} // Partition, compile!
         s_A(i,j) = max(s_A(i,j),r_BC+s_B(i,k)+s_C(k+1,j))
       end
     end
  end
end
```

GPU implementation

Compile rules into kernel code.

Partition rules to minimize symbol cover.

Load a block of symbols into registers.

Reuse factor is very high.

Fully SIMD, 1024 threads execute same rules.

I machine instruction per rule!

Achieves about 1 Tflop per GPU, and around 1,000 sentences/sec.

Outline

- Basics: Constituency and Dependency Parsing
- Constituency Parsing: CFGs and PCFGs
- CKY parsing
- Learning/optimizing grammars

Grammar Learning

Given a fixed grammar structure, the problem of learning rule probabilities can be expressed as a likelihood maximization problem for the true parses from a Treebank.

The solution is an EM (Expectation Maximization) algorithm, which is rather similar to the forward-backward algorithm for HMMs.

The method is called "inside-outside," and it computes probabilities of cells bottom-up and then top-down.

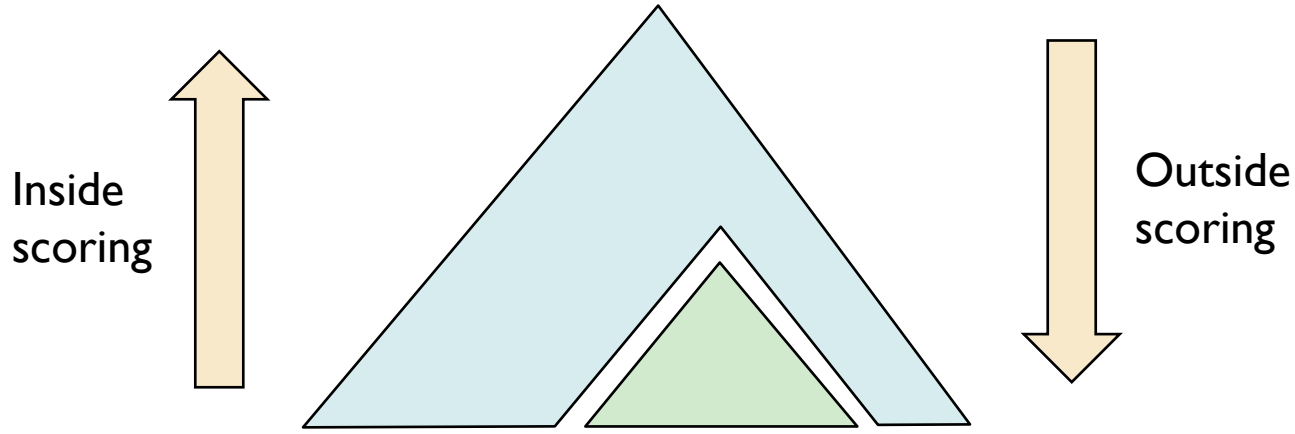
Grammar Learning

Inside score (green) is the score we defined so far, which depends only on the span of the current node.

Outside score (blue) depends on the rest of the sentence, excluding the current span.

We move up to the root computing inside scores, and then down from the root combining inside and outside scores.

Complexity is the same as full parsing.



Symbol Splitting/Merging

In reality, grammars allow many sentences that are nonsense:

"The span took the altitude out for a dog"

"Noun," "verb," etc. are very coarse categories to capture the set of "well-formed" English sentences.

These subcategories are not available as Treebank labels, but should be learnable as latent factors.

i.e. we can split a symbol into two subsymbols and optimize the likelihood of the new grammar on the training set.

If there is not enough likelihood improvement from doing this, we merge the symbols back into one.

Berkeley Grammar

Starting from a simple CFG grammar (98 symbols, 236 unary rules and 3840 binary rules), the grammar is progressively split and optimized to improve accuracy.

6 cycles are performed, which could lead to up to 64 splits of the base symbols.

The final grammar has about 1100 symbols and 1.7 million binary rules.

Berkeley Grammar

NNP	62	CC	7	WP$	2	NP	37	CONJP	2
JJ	58	JJR	5	WDT	2	VP	32	FRAG	2
NNS	57	JJS	5	-RRB-	2	PP	28	NAC	2
NN	56	:	5	”	1	ADVP	22	UCP	2
VBN	49	PRP	4	FW	1	S	21	WHADVP	2
RB	47	PRP$	4	RBS	1	ADJP	19	INTJ	1
VBG	40	MD	3	TO	1	SBAR	15	SBARQ	1
VB	37	RBR	3	$	1	QP	9	RRC	1
VBD	36	WP	2	UH	1	WHNP	5	WHADJP	1
CD	32	POS	2	,	1	PRN	4	X	1
IN	27	PDT	2	“	1	NX	4	ROOT	1
VBZ	25	WRB	2	SYM	1	SINV	3	LST	1
VBP	19	-LRB-	2	RP	1	PRT	2		
DT	17	.	2	LS	1	WHPP	2		
NNPS	11	EX	2	#	1	SQ	2		

Table 2: Number of latent annotations determined by our split-merge procedure after 6 SM cycles

From “Learning Accurate, Compact, and Interpretable Tree Annotation” Slav Petrov et al., Proc. of the 21st Int. Conf. on Computational Linguistics

Software

Stanford Parser:

http://nlp.stanford.edu/software/lex-parser.shtml

Constituency parses, lexicalized output, and dependencies.

Berkeley Parser:

http://code.google.com/p/berkeleyparser/

The split symbol grammar just described.

Summary

- Basics: Constituency and Dependency Parsing
- Constituency Parsing: CFGs and PCFGs
- CKY parsing
- Learning/optimizing grammars