

Probabilistic Context Free Grammars

Many slides from Michael Collins

Overview

- ▶ Probabilistic Context-Free Grammars (PCFGs)
- ▶ The CKY Algorithm for parsing with PCFGs

A Probabilistic Context-Free Grammar (PCFG)

S	\Rightarrow	NP	VP	1.0
VP	\Rightarrow	Vi		0.4
VP	\Rightarrow	Vt	NP	0.4
VP	\Rightarrow	VP	PP	0.2
NP	\Rightarrow	DT	NN	0.3
NP	\Rightarrow	NP	PP	0.7
PP	\Rightarrow	P	NP	1.0

Vi	\Rightarrow	sleeps	1.0
Vt	\Rightarrow	saw	1.0
NN	\Rightarrow	man	0.7
NN	\Rightarrow	woman	0.2
NN	\Rightarrow	telescope	0.1
DT	\Rightarrow	the	1.0
IN	\Rightarrow	with	0.5
IN	\Rightarrow	in	0.5

- Probability of a tree t with rules

$$\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \dots, \alpha_n \rightarrow \beta_n$$

is $p(t) = \prod_{i=1}^n q(\alpha_i \rightarrow \beta_i)$ where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$.

DERIVATION

RULES USED

PROBABILITY

S

DERIVATION

S

NP VP

RULES USED

$S \rightarrow NP VP$

PROBABILITY

1.0

DERIVATION

S

NP VP

DT NN VP

RULES USED

$S \rightarrow NP VP$

$NP \rightarrow DT NN$

PROBABILITY

1.0

0.3

DERIVATION

S

NP VP

DT NN VP

the NN VP

RULES USED

$S \rightarrow NP VP$

$NP \rightarrow DT NN$

$DT \rightarrow \text{the}$

PROBABILITY

1.0

0.3

1.0

DERIVATION

S

NP VP

DT NN VP

the NN VP

the dog VP

RULES USED

$S \rightarrow NP VP$

$NP \rightarrow DT NN$

$DT \rightarrow \text{the}$

$NN \rightarrow \text{dog}$

PROBABILITY

1.0

0.3

1.0

0.1

DERIVATION

S

NP VP

DT NN VP

the NN VP

the dog VP

the dog Vi

RULES USED

$S \rightarrow NP VP$

$NP \rightarrow DT NN$

$DT \rightarrow \text{the}$

$NN \rightarrow \text{dog}$

$VP \rightarrow V_i$

PROBABILITY

1.0

0.3

1.0

0.1

0.4

DERIVATION

S

NP VP

DT NN VP

the NN VP

the dog VP

the dog Vi

the dog laughs

RULES USED

$S \rightarrow NP VP$

$NP \rightarrow DT NN$

$DT \rightarrow \text{the}$

$NN \rightarrow \text{dog}$

$VP \rightarrow V_i$

$V_i \rightarrow \text{laughs}$

PROBABILITY

1.0

0.3

1.0

0.1

0.4

0.5

Properties of PCFGs

- ▶ Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG

Properties of PCFGs

- ▶ Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG
- ▶ Say we have a sentence s , set of derivations for that sentence is $\mathcal{T}(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $\mathcal{T}(s)$. i.e., *we now have a ranking in order of probability*.

Properties of PCFGs

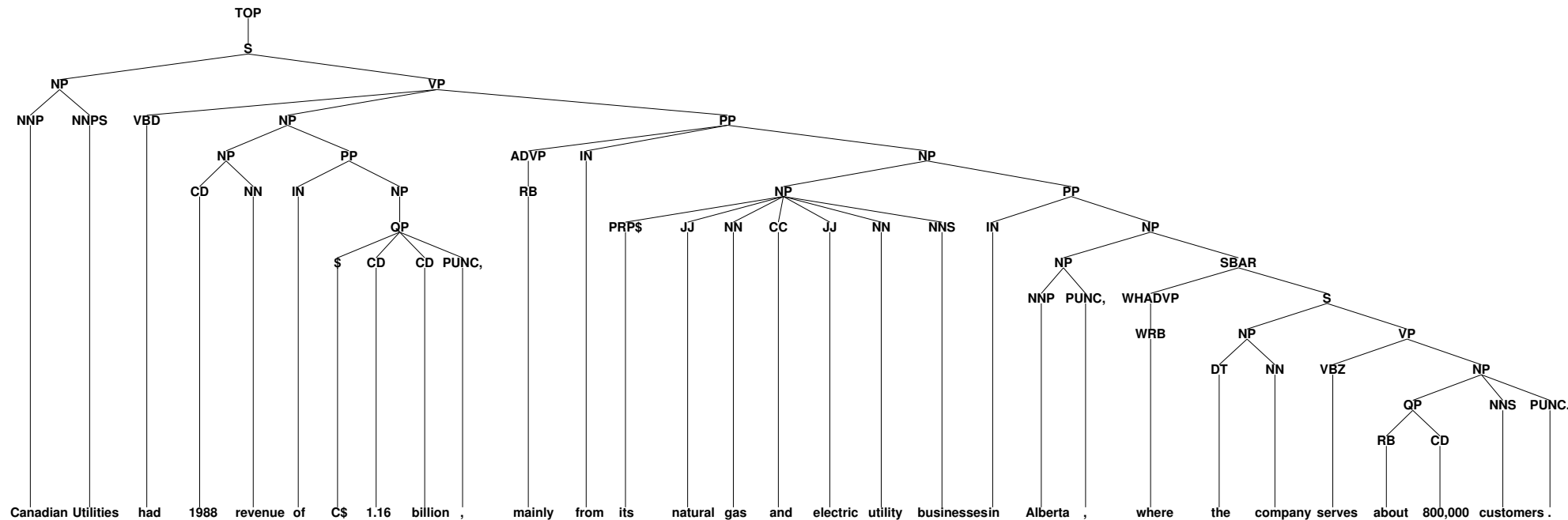
- ▶ Assigns a probability to each *left-most derivation*, or parse-tree, allowed by the underlying CFG
- ▶ Say we have a sentence s , set of derivations for that sentence is $\mathcal{T}(s)$. Then a PCFG assigns a probability $p(t)$ to each member of $\mathcal{T}(s)$. i.e., *we now have a ranking in order of probability*.
- ▶ The most likely parse tree for a sentence s is

$$\arg \max_{t \in \mathcal{T}(s)} p(t)$$

Data for Parsing Experiments: Treebanks

- ▶ Penn WSJ Treebank = 50,000 sentences with associated trees
- ▶ Usual set-up: 40,000 training sentences, 2400 test sentences

An example tree:



Deriving a PCFG from a Treebank

- ▶ Given a set of example trees (a treebank), the underlying CFG can simply be **all rules seen in the corpus**
- ▶ Maximum Likelihood estimates:

$$q_{ML}(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

where the counts are taken from a training set of example trees.

- ▶ **If the training data is generated by a PCFG**, then as the training data size goes to infinity, the maximum-likelihood PCFG will converge to the same distribution as the “true” PCFG.

PCFGs

Booth and Thompson (1973) showed that a CFG with rule probabilities correctly defines a distribution over the set of derivations provided that:

1. The rule probabilities define conditional distributions over the different ways of rewriting each non-terminal.
2. A technical condition on the rule probabilities ensuring that the probability of the derivation terminating in a finite number of steps is 1. (This condition is not really a practical concern.)

Parsing with a PCFG

- ▶ Given a PCFG and a sentence s , define $\mathcal{T}(s)$ to be the set of trees with s as the yield.
- ▶ Given a PCFG and a sentence s , how do we find

$$\arg \max_{t \in \mathcal{T}(s)} p(t)$$

Chomsky Normal Form

A context free grammar $G = (N, \Sigma, R, S)$ in Chomsky Normal Form is as follows

- ▶ N is a set of non-terminal symbols
- ▶ Σ is a set of terminal symbols
- ▶ R is a set of rules which take one of two forms:
 - ▶ $X \rightarrow Y_1 Y_2$ for $X \in N$, and $Y_1, Y_2 \in N$
 - ▶ $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$
- ▶ $S \in N$ is a distinguished start symbol

A Dynamic Programming Algorithm

- ▶ Given a PCFG and a sentence s , how do we find

$$\max_{t \in \mathcal{T}(s)} p(t)$$

- ▶ Notation:

n = number of words in the sentence

w_i = i 'th word in the sentence

N = the set of non-terminals in the grammar

S = the start symbol in the grammar

- ▶ Define a dynamic programming table

$\pi[i, j, X]$ = maximum probability of a constituent with non-terminal X
spanning words $i \dots j$ inclusive

- ▶ Our goal is to calculate $\max_{t \in \mathcal{T}(s)} p(t) = \pi[1, n, S]$

An Example

the dog saw the man with the telescope

A Dynamic Programming Algorithm

- ▶ Base case definition: for all $i = 1 \dots n$, for $X \in N$

$$\pi[i, i, X] = q(X \rightarrow w_i)$$

(note: define $q(X \rightarrow w_i) = 0$ if $X \rightarrow w_i$ is not in the grammar)

- ▶ Recursive definition: for all $i = 1 \dots n$, $j = (i + 1) \dots n$, $X \in N$,

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

An Example

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s+1, j, Z))$$

the dog saw the man with the telescope

The Full Dynamic Programming Algorithm

Input: a sentence $s = x_1 \dots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.

Initialization:

For all $i \in \{1 \dots n\}$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

- ▶ For $l = 1 \dots (n - 1)$
 - ▶ For $i = 1 \dots (n - l)$
 - ▶ Set $j = i + l$
 - ▶ For all $X \in N$, calculate

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

and

$$bp(i, j, X) = \arg \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \dots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s + 1, j, Z))$$

A Dynamic Programming Algorithm for the Sum

- ▶ Given a PCFG and a sentence s , how do we find

$$\sum_{t \in \mathcal{T}(s)} p(t)$$

- ▶ Notation:

n = number of words in the sentence

w_i = i 'th word in the sentence

N = the set of non-terminals in the grammar

S = the start symbol in the grammar

- ▶ Define a dynamic programming table

$\pi[i, j, X]$ = sum of probabilities for constituent with non-terminal X
spanning words $i \dots j$ inclusive

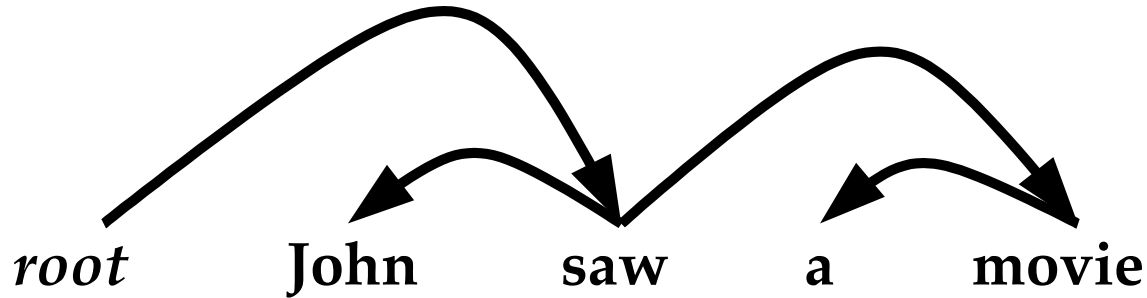
- ▶ Our goal is to calculate $\sum_{t \in \mathcal{T}(s)} p(t) = \pi[1, n, S]$

Summary

- ▶ PCFGs augments CFGs by including a probability for each rule in the grammar.
- ▶ The probability for a parse tree is the product of probabilities for the rules in the tree
- ▶ To build a PCFG-parsed parser:
 1. Learn a PCFG from a treebank
 2. Given a test data sentence, use the CKY algorithm to compute the highest probability tree for the sentence under the PCFG

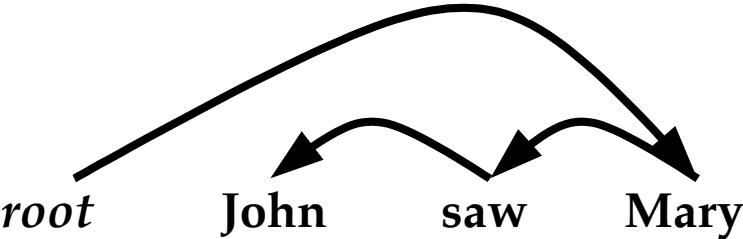
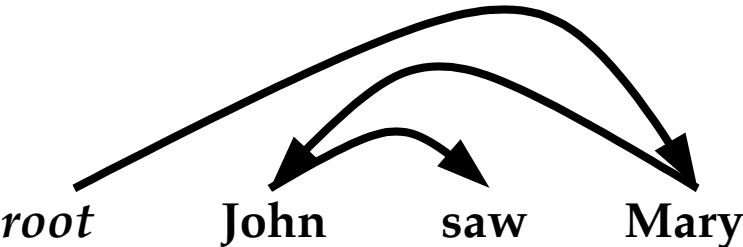
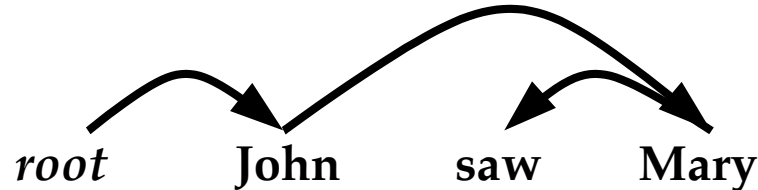
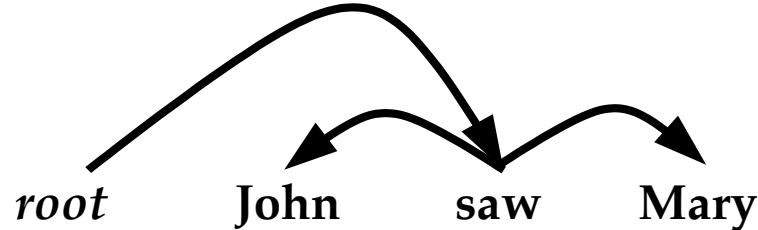
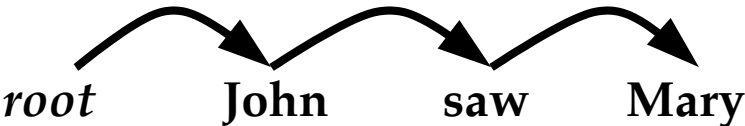
Dependency Parsing

Unlabeled Dependency Parses

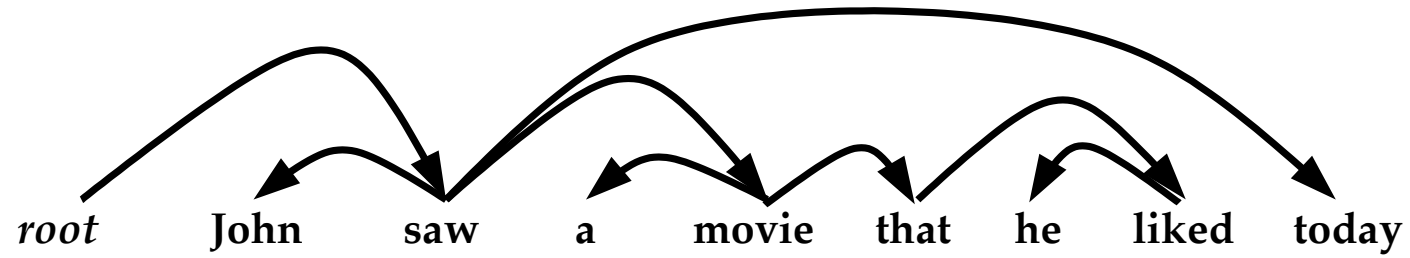


- ▶ *root* is a special *root* symbol
- ▶ Each dependency is a pair (h, m) where h is the index of a head word, m is the index of a modifier word. In the figures, we represent a dependency (h, m) by a directed edge from h to m .
- ▶ Dependencies in the above example are $(0, 2)$, $(2, 1)$, $(2, 4)$, and $(4, 3)$. (We take 0 to be the root symbol.)

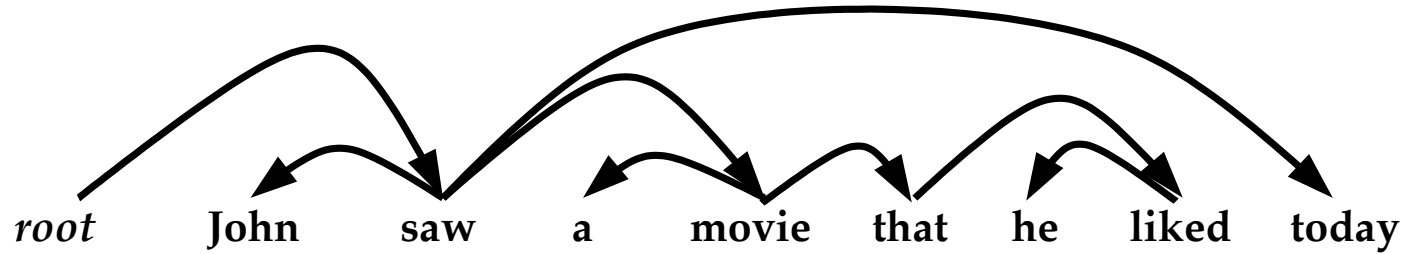
All Dependency Parses for *John saw Mary*



A More Complex Example



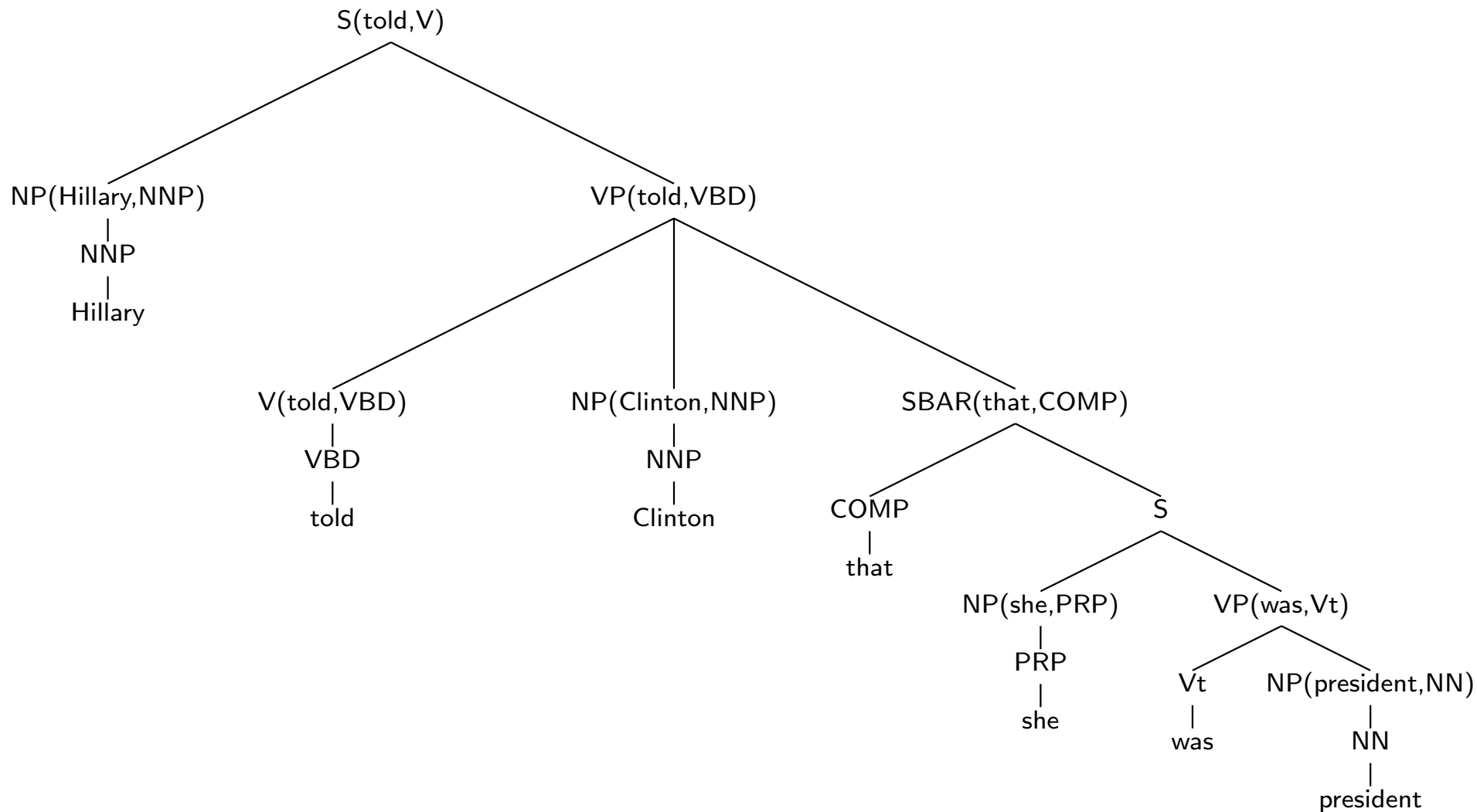
Conditions on Dependency Structures



- ▶ The dependency arcs form a *directed tree*, with the root symbol at the root of the tree.
(Definition: A directed tree rooted at *root* is a tree, where for every word *w* other than the root, there is a directed path from *root* to *w*.)
- ▶ There are no “crossing dependencies”.
Dependency structures with no crossing dependencies are sometimes referred to as **projective** structures.

Dependency Parsing Resources

- ▶ CoNLL 2006 conference had a “shared task” with dependency parsing of 12 languages (Arabic, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish, Turkish). 19 different groups developed dependency parsing systems. (See also CoNLL 2007).
- ▶ PhD thesis on the topic: Ryan McDonald, *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*, University of Pennsylvania.
- ▶ For some languages, e.g., Czech, there are “dependency banks” available which contain training data in the form of sentences paired with dependency structures
- ▶ For other languages, we can extract dependency structures from treebanks



Unlabeled Dependencies:

(0,2) (for root → told)
 (2,1) (for told → Hillary)
 (2,3) (for told → Clinton)
 (2,4) (for told → that)

(4,6) (for that → was)
 (6,5) (for was → she)
 (6,7) (for was → president)

Efficiency of Dependency Parsing

- ▶ PCFG parsing is $O(n^3G^3)$ where n is the length of the sentence, G is the number of non-terminals in the grammar
- ▶ Lexicalized PCFG parsing is $O(n^5G^3)$ where n is the length of the sentence, G is the number of non-terminals in the grammar.
- ▶ Unlabeled dependency parsing is $O(n^3)$.

GLMs for Dependency parsing

- ▶ x is a sentence
- ▶ **GEN**(x) is set of all dependency structures for x
- ▶ **f**(x, y) is a feature vector for a sentence x paired with a dependency parse y

GLMs for Dependency parsing

- ▶ To run the perceptron algorithm, we must be able to efficiently calculate

$$\arg \max_{y \in \mathbf{GEN}(x)} \mathbf{w} \cdot \mathbf{f}(x, y)$$

- ▶ Local feature vectors: define

$$\mathbf{f}(x, y) = \sum_{(h, m) \in y} \mathbf{g}(x, h, m)$$

where $\mathbf{g}(x, h, m)$ maps a sentence x and a dependency (h, m) to a local feature vector

- ▶ Can then use dynamic programming to calculate

$$\arg \max_{y \in \mathbf{GEN}(x)} \mathbf{w} \cdot \mathbf{f}(x, y) = \arg \max_{y \in \mathbf{GEN}(x)} \sum_{(h, m) \in y} \mathbf{w} \cdot \mathbf{g}(x, h, m)$$

Definition of Local Feature Vectors

- ▶ Features from McDonald et al. (2005):
 - ▶ Note: define w_i to be the i 'th word in the sentence, t_i to be the part-of-speech (POS) tag for the i 'th word.
 - ▶ *Unigram* features: Identity of w_h . Identity of w_m . Identity of t_h . Identity of t_m .
 - ▶ *Bigram* features: Identity of the 4-tuple $\langle w_h, w_m, t_h, t_m \rangle$. Identity of sub-sets of this 4-tuple, e.g., identity of the pair $\langle w_h, w_m \rangle$.
 - ▶ *Contextual features*: Identity of the 4-tuple $\langle t_h, t_{h+1}, t_{m-1}, t_m \rangle$. Similar features which consider t_{h-1} and t_{m+1} , giving 4 possible feature types.
 - ▶ *In-between features*: Identity of triples $\langle t_h, t, t_m \rangle$ for any tag t seen between words h and m .

Results from McDonald (2005)

Method	Accuracy
Collins (1997)	91.4%
1st order dependency	90.7%
2nd order dependency	91.5%

- ▶ Accuracy is percentage of correct unlabeled dependencies
- ▶ Collins (1997) is result from a lexicalized context-free parser, with dependencies extracted from the parser's output
- ▶ 1st order dependency is the method just described. 2nd order dependency is a model that uses richer representations.
- ▶ Advantages of the dependency parsing approaches: simplicity, efficiency ($O(n^3)$ parsing time).