

# LECTURE 8: PROBABILISTIC CONTEXT-FREE GRAMMARS

"Triumph of Grammar", Pierre Brebiette, 17th century



# Different ways of speaking

- *"No more training do you require. Already know you, that which you need."*  
- Yoda, *Return of the Jedi*
- *"To a dark place, this line of thought will carry us. Hmmm. Great care, we must take."*  
- Yoda, *Revenge of the Sith*
- *This, he announced, was "a thing up with which I will not put".*  
- anecdote in *The West Sussex Gazette*, 1941, later misattributed to Winston Churchill
- The generative models of word sequences we have seen so far (N-grams, HMMs) assume sequences are generated beginning-to-end based on previous words/states.
- But what about more general ways of constructing sequences? For example, we know that question sentences tend to have particular structure - can we model how particular sentences arise from more general 'templates' of their parts? This can be done by **grammars**.

# Probabilistic context-free grammar

- A **probabilistic context-free grammar** (PCFG), also called a stochastic context-free grammar, is defined by:
    - A **start symbol**  $n_1$
    - A set of **nonterminal symbols**  $\{n_i\}, i=1, \dots, N$
    - A set of **terminal symbols** (specific words, punctuation, phrases):  $\{w_i\}, i=1, \dots, V$
    - A set of **rules**,  $\{n_{i_r} \rightarrow S_r\}, r=1, \dots, R$  where each rule  $r$  transforms a particular nonterminal symbol  $n_{i_r}$  into a sequence  $S_r$  of nonterminal and terminal symbols
    - A set of **conditional probabilities** of the rules given their left-hand-side nonterminal symbol, so that for each nonterminal symbol the probabilities of its rules sum to 1
- $$\{p_r = p(n_{i_r} \rightarrow S_r | n_{i_r})\}, r=1, \dots, R, \quad \forall i=1, \dots, N \quad \sum_{r=1}^R \delta(i_r=i) p_r = 1$$
- 1 if  $i_r=i$ ,  
0 otherwise

# Probabilistic context-free grammar

- The PCFG is called a "**context-free**" grammar because each rule can be applied to a nonterminal symbol regardless of the context where the nonterminal symbol appears, i.e. regardless of the surrounding symbols
- A PCFG can be used to generate sentences, or to parse existing observed sentences.
- There can be several ways to generate the same sentence (through different sequences of applying the rules), and thus there can be several parses of an observed sentence.
- Because the rules in a PCFG have probabilities, the PCFG can be used to find **the most likely parse** of an observed sentence

# Dogs versus Robots

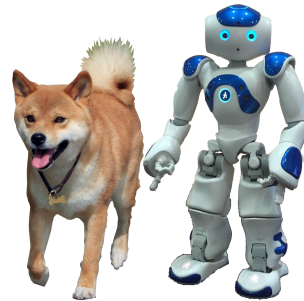


# Probabilistic context-free grammar

- Example grammar:

- Start symbol:  $n_1 = \textit{Sentence}$
- Set of nonterminal symbols:  
 $\{\textit{Sentence}, \textit{Noun}, \textit{Article}, \textit{Verb}\}$
- Terminal symbols:  $\{a, the, robot, dog, bit, fed, walked, petted, .\}$
- Rules and their probabilities:
 

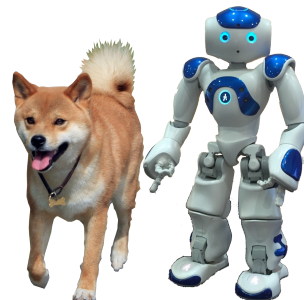
$\textit{Sentence} \rightarrow \textit{Article} \textit{Noun} \textit{Verb} \textit{Article} \textit{Noun} .$	1
$\textit{Article} \rightarrow a$	0.7
$\textit{Article} \rightarrow the$	0.3
$\textit{Noun} \rightarrow robot$	0.6
$\textit{Noun} \rightarrow dog$	0.4
$\textit{Verb} \rightarrow bit$	0.1
$\textit{Verb} \rightarrow fed$	0.2
$\textit{Verb} \rightarrow walked$	0.4
$\textit{Verb} \rightarrow petted$	0.3



# Probabilistic context-free grammar

- Example sentences and their probabilities:

– The robot walked a dog.  $1 \cdot 0.3 \cdot 0.6 \cdot 0.4 \cdot 0.7 \cdot 0.4 = 0.02016$



<i>Sentence</i>	<i>Sentence</i> $\rightarrow$ <i>Article Noun Verb Article Noun.</i>	<i>1</i>
<i>Article Noun Verb Article Noun .</i>	<i>Article</i> $\rightarrow$ <i>the</i>	0.3
<i>The Noun Verb Article Noun.</i>	<i>Noun</i> $\rightarrow$ <i>robot</i>	0.6
<i>The robot Verb Article Noun .</i>	<i>Verb</i> $\rightarrow$ <i>walked</i>	0.4
<i>The robot walked Article Noun .</i>	<i>Article</i> $\rightarrow$ <i>a</i>	0.7
<i>The man walked a Noun .</i>	<i>Noun</i> $\rightarrow$ <i>dog</i>	0.4
<i>The robot walked a dog .</i>		

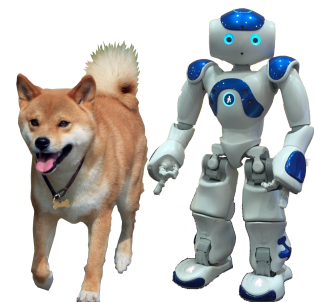
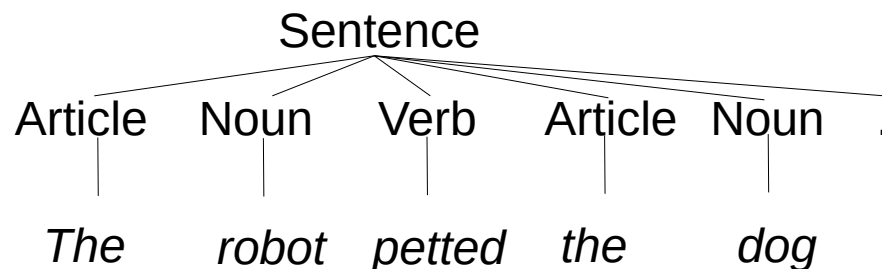
– A dog petted a dog .  $1 \cdot 0.7 \cdot 0.4 \cdot 0.3 \cdot 0.7 \cdot 0.4 = 0.02352$

- The probability of the sentence is the probability that out of all valid sentences in the grammar, it will generate this one.
- All possible sentences (**for this example grammar** their probabilities sum to 1): A robot bit a robot. A robot bit a dog. A robot bit the robot. A robot bit the dog. A robot fed a robot. A robot fed a dog. A robot fed the robot. A robot fed the dog. A robot petted a robot. A robot petted a dog. A robot petted the robot. A robot petted the dog. A robot walked a robot. A robot walked a dog. A robot walked the robot. A robot walked the dog. A dog bit a robot. A dog bit a dog. A dog bit the robot. A dog bit the dog. A dog fed a robot. A dog fed a dog. A dog fed the robot. A dog fed the dog. A dog petted a robot. A dog petted a dog. A dog petted the robot. A dog petted the dog. A dog walked a dog. A dog walked a dog. A dog walked the robot. A dog walked the dog. The robot bit a robot. The robot bit a dog. The robot bit the robot. The robot bit the dog. The robot fed a robot. The robot fed a dog. The robot fed the robot. The robot fed the dog. The robot petted a robot. The robot petted a dog. The robot petted the robot. The robot petted the dog. The robot walked a robot. The robot walked a dog. The robot walked the robot. The robot walked the dog. The dog bit a robot. The dog bit a dog. The dog bit the robot. The dog bit the dog. The dog fed a robot. The dog fed a dog. The dog fed the robot. The dog fed the dog. The dog petted a robot. The dog petted a dog. The dog petted the robot. The dog petted the dog. The dog walked a dog. The dog walked a dog. The dog walked the robot. The dog walked the dog.



# Probabilistic context-free grammar

- Parse tree of a sentence:



- (In this simple example grammar each valid sentence has only one possible **parse tree**: robot, dog can only come from Noun; a, the from Article, and so on. Generally there can be many possible parses for the same sentence.)
- PCFGs have useful independence properties:
  - **Place invariance**: probability of a subtree, which yields some substring of words in the text, is independent of where that subtree occurs in a longer text
  - **Context-free**: probability of a subtree is independent of any words that are not part of the subtree
  - **Ancestor-free**: probability of a subtree is independent of any nonterminal symbols (parse tree nodes) that are not part of the subtree



# Probabilistic context-free grammar

- A PCFG may define a proper language model (proper distribution) but not always.

Example:

This kind of grammar is actually called  
a probabilistic regular grammar

$Sentence \rightarrow blah$  0.5

$Sentence \rightarrow blah \ Sentence$  0.5

- Possible sentences and their probabilities:  
blah 0.5, blah blah 0.25, blah blah blah 0.125, ...

- Sum of probabilities:  $0.5 + 0.25 + 0.125 + \dots = 1$ , **ok (proper distribution)!**

- But consider this alternative PCFG:

$Sentence \rightarrow blah$  0.3

$Sentence \rightarrow Sentence \ Sentence$  0.7

- Now:
  - blah 0.3    1 way to generate = 0th Catalan number
  - blah blah  $0.7 \cdot 0.3 \cdot 0.3$     1 way to generate = 1st Catalan number
  - blah blah blah  $2 \cdot (0.7 \cdot 0.7 \cdot 0.3 \cdot 0.3 \cdot 0.3)$     2 ways  $S \rightarrow SS \rightarrow Sb \rightarrow SSb \rightarrow \dots$ ,  $S \rightarrow SS \rightarrow bS \rightarrow bSS \rightarrow \dots$
  - blah blah blah blah  $5 \cdot (0.7 \cdot 0.7 \cdot 0.7 \cdot 0.3 \cdot 0.3 \cdot 0.3 \cdot 0.3)$     5 ways = 3rd Catalan number
  - blah blah blah blah blah  $14 \cdot (0.7 \cdot 0.7 \cdot 0.7 \cdot 0.7 \cdot 0.3 \cdot 0.3 \cdot 0.3 \cdot 0.3 \cdot 0.3)$     14 ways = 4th C.n.
  - blah blah blah blah blah blah  $42 \cdot (0.7 \cdot 0.7 \cdot 0.7 \cdot 0.7 \cdot 0.7 \cdot 0.3 \cdot 0.3 \cdot 0.3 \cdot 0.3 \cdot 0.3 \cdot 0.3)$     42 ways = 5th C.n.
- Sum: about 0.43, **less than 1 - not a proper distribution!**
- Grammars whose parameters are learned from a parsed training corpus can be guaranteed to yield a proper distribution (Chi and German, 1998)

# Chomsky normal form PCFG

- A PCFG in **Chomsky normal form** only has **unary and binary rules** of this form:

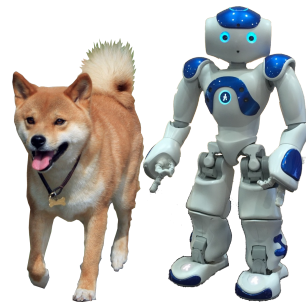
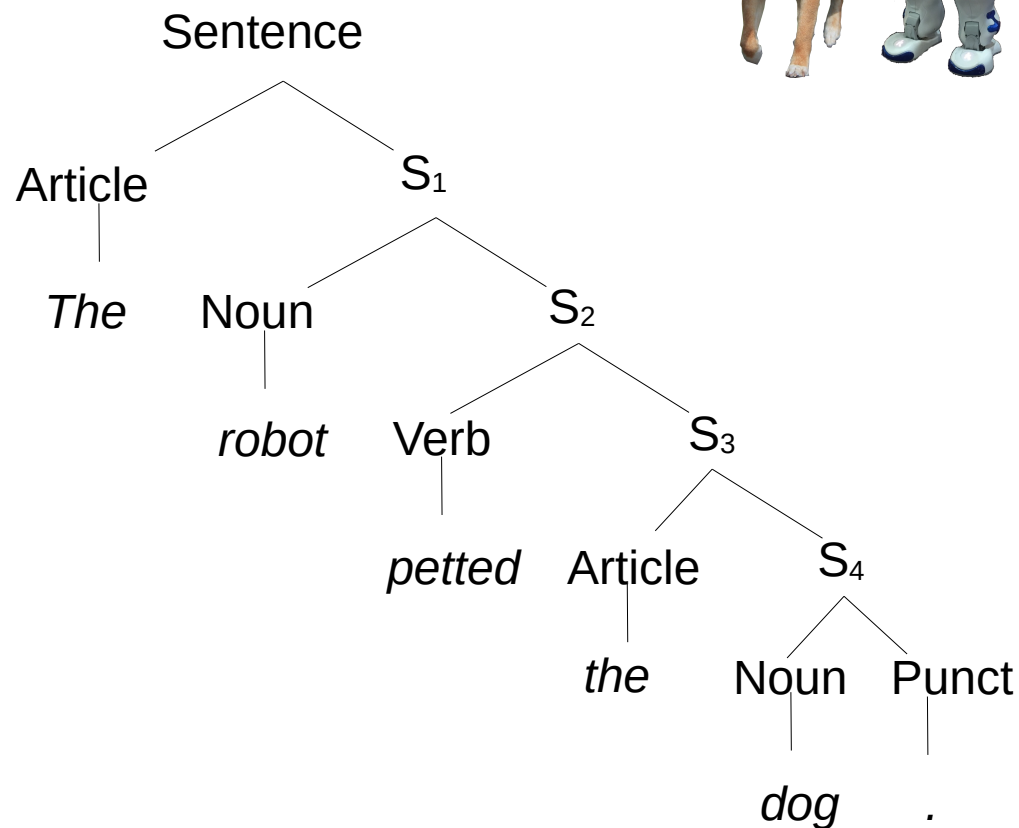
$$\begin{aligned} n_i \rightarrow n_j n_k & \quad (Nonterminal_i \rightarrow Nonterminal_j Nonterminal_k) \\ n_i \rightarrow w_j & \quad (Nonterminal_i \rightarrow word_j) \end{aligned}$$

- Parameters: conditional probabilities of all  $N^3$  binary rules (all combinations  $i,j,k$ ) and all  $N \cdot V$  unary rules (all combinations  $i,j$ )
- As before, the conditional probabilities of all rules with the same left-hand-side must sum to 1:
 
$$\forall i=1,\dots,N \quad \sum_{j=1}^N \sum_{k=1}^N p(n_i \rightarrow n_j n_k | n_i) + \sum_{j=1}^V p(n_i \rightarrow w_j | n_i) = 1$$
- Any context-free grammar (CFG) can be represented by a CFG in Chomsky normal form that is **weakly equivalent** to the original CFG:
  - Two grammars are said to be weakly equivalent if they generate the same language (same set of possible sentences).
  - Two grammars are **strongly equivalent** if they also give sentences the same tree structure.
  - In the probabilistic case also the probabilities of the sentences must be the same in the two grammars

# Chomsky normal form PCFG

- Previous example in Chomsky normal form:

<i>Sentence</i>	$\rightarrow$	<i>Article</i> $S_1$	1
$S_1$	$\rightarrow$	<i>Noun</i> $S_2$	1
$S_2$	$\rightarrow$	<i>Verb</i> $S_3$	1
$S_3$	$\rightarrow$	<i>Article</i> $S_4$	1
$S_4$	$\rightarrow$	<i>Noun Punct</i>	1
<i>Article</i>	$\rightarrow$	<i>a</i>	0.7
<i>Article</i>	$\rightarrow$	<i>the</i>	0.3
<i>Noun</i>	$\rightarrow$	<i>robot</i>	0.6
<i>Noun</i>	$\rightarrow$	<i>dog</i>	0.4
<i>Verb</i>	$\rightarrow$	<i>bit</i>	0.1
<i>Verb</i>	$\rightarrow$	<i>fed</i>	0.2
<i>Verb</i>	$\rightarrow$	<i>walked</i>	0.4
<i>Verb</i>	$\rightarrow$	<i>petted</i>	0.3
<i>Punct</i>	$\rightarrow$	<i>.</i>	1





# Probability of a string

- There can be exponentially many possible parse trees for a string of  $M$  words, probability of the string needs to be summed over them all **efficiently**.
- We define **inside probabilities** and **outside probabilities**.
- Inside probability: given a specific nonterminal  $n_i$ , probability that it generates the substring of words  $l - m$  in the text

$$p_i^{Inside}(l, m) = p(w_l, \dots, w_m | n_i)$$

- Outside probability: given the starting symbol, probability to generate all words outside  $l-m$  and a nonterminal  $n_i$  responsible for the contents of  $l-m$  which we denote as  $n_i(l, m)$

$$p_i^{Outside}(l, m) = p(w_1, \dots, w_{l-1}, n_i(l, m), w_{m+1}, \dots, w_M | n_1)$$

- The probability of a string can be computed by a dynamic programming algorithm: the **inside algorithm**

# Inside algorithm

- Probability of a whole string is an inside probability:

$$p(w_1, \dots, w_M) = p(w_1, \dots, w_M | n_1) = p_1^{Inside}(1, M)$$

- We need to break down the inside probability into terms we know.

- If  $l = m$ , the string is a single word which must result from a unary rule  $n_i \rightarrow w_m$  so

$$p_i^{Inside}(m, m) = p(n_i \rightarrow w_m | n_i)$$

- If  $l < m$  the string must result from some binary rule and its follow-up rules:  $n_i \rightarrow n_j n_k \rightarrow \dots \rightarrow w_l, \dots, w_m$

- In such a rule, the nonterminals  $n_j$  and  $n_k$  must each be responsible for a substring:  $w_l, \dots, w_u$  and  $w_{u+1}, \dots, w_m$  where  $u$  is the breakpoint

# Inside algorithm

- The inside probability can then be computed inductively, as a sum over the possible intermediate nonterminals and breakpoints:

$$\begin{aligned}
 p_i^{\text{Inside}}(l, m) &= p(w_l, \dots, w_m | n_i) \\
 &= p(w_l, \dots, w_u, w_{u+1}, \dots, w_m | n_i) \\
 &= \sum_{j=1}^N \sum_{k=1}^N \sum_{u=l}^{m-1} p(w_l, \dots, w_u, w_{u+1}, \dots, w_m, n_j(l, u), n_k(u+1, m) | n_i) \\
 &= \sum_{j=1}^N \sum_{k=1}^N \sum_{u=l}^{m-1} p(w_l, \dots, w_u | w_{u+1}, \dots, w_m, n_j(l, u), n_k(u+1, m), n_i) \\
 &\quad \cdot p(w_{u+1}, \dots, w_m | n_j(l, u), n_k(u+1, m), n_i) \cdot p(n_j(l, u), n_k(u+1, m) | n_i) \\
 &= \sum_{j=1}^N \sum_{k=1}^N \sum_{u=l}^{m-1} p(w_l, \dots, w_u | n_j) \cdot p(w_{u+1}, \dots, w_m | n_k) \cdot p(n_j, n_k | n_i) \\
 &= \sum_{j=1}^N \sum_{k=1}^N \sum_{u=l}^{m-1} p_j^{\text{Inside}}(l, u) p_k^{\text{Inside}}(u+1, m) p(n_i \rightarrow n_j n_k | n_i)
 \end{aligned}$$

- Context-free: probability of the l-u subtree does not depend on words not in the l-u subtree
- Ancestor-free: probability of the l-u subtree does not depend on nodes not in the l-u subtree
- Same for the subtree of (u+1)-m

- Substrings l-u and (u+1)-m are shorter than l-m, so eventually they become single-word substrings whose probabilities come from the unary rules. Thus the above algorithm suffices to calculate the probability of a string of text.



# Outside algorithm

- Probability of a whole string can also be computed using outside probabilities. For any breakpoint  $l$ :

$$\begin{aligned}
 & p(w_1, \dots, w_M | n_1) \\
 &= p(w_1, \dots, w_{l-1}, w_l, w_{l+1}, \dots, w_M | n_1) \\
 &= \sum_{j=1}^N p(w_1, \dots, w_{l-1}, w_l, w_{l+1}, \dots, w_M, n_j(l, l) | n_1) \\
 &= \sum_{j=1}^N p(w_l, w_1, \dots, w_{l-1}, n_j(l, l), w_{l+1}, \dots, w_M | n_1) \\
 &= \sum_{j=1}^N p(w_l | w_1, \dots, w_{l-1}, n_j(l, l), w_{l+1}, \dots, w_M, n_1) p(w_1, \dots, w_{l-1}, n_j(l, l), w_{l+1}, \dots, w_M | n_1) \\
 &= \sum_{j=1}^N p(w_l | n_j(l, l)) p(w_1, \dots, w_{l-1}, n_j(l, l), w_{l+1}, \dots, w_M | n_1) \\
 &= \sum_{j=1}^N p(n_j \rightarrow w_l | n_j) p_j^{\text{Outside}}(l, l)
 \end{aligned}$$

# Outside algorithm

- We start top down from the top nonterminal:

$$p_1^{Outside}(1, M) = p(n_1(1, M) | n_1) = 1$$

$$\forall i \neq 1 \quad p_i^{Outside}(1, M) = p(n_i(1, M) | n_1) = 0$$

- Consider a nonterminal  $n_v$  that is not the top one and is responsible for words l-m.
  - The nonterminal must have resulted from a binary rule  $n_i \rightarrow n_j n_k$ , thus it must be the **left child** or **right child** of a parent nonterminal.
  - If  $n_v$  is the left child, there must be a right child  $n_k$  which is responsible for a substring (m+1)-q just after m, their parent  $n_i$  is then responsible for l-q.
  - If  $n_v$  is the right child, there must be a left child  $n_j$  which is responsible for a substring q-(l-1) just before l, their parent  $n_i$  is then responsible for q-m.

# Outside algorithm

- We can then write:

$$\begin{aligned}
 p_v^{\text{Outside}}(l, m) &= p(w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M | n_1) \\
 &= \sum_{i=1}^N \sum_{k=1}^N \sum_{q=m+1}^M p(w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_q, w_{q+1}, \dots, w_M, n_k(m+1, q), n_i(l, q) | n_1) \\
 &\quad + \sum_{i=1}^N \sum_{j=1}^N \sum_{q=1}^{l-1} p(w_1, \dots, w_{q-1}, w_q, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M, n_j(q, l-1), n_i(q, m) | n_1) \\
 &= \textcolor{red}{vIsLeftChild} + \textcolor{green}{vIsRightChild}
 \end{aligned}$$

- Note that in rules of the form  $n_i \rightarrow n_v n_v$  the nonterminal  $n_v$  appears as both a left child and a right child in the same rule. We must not count such rules twice, so we exclude them from one of the sums (e.g. from *vIsLeftChild*)



# Outside algorithm

- The case where  $v$  is the right child becomes:

*vIsRightChild*

$$\begin{aligned}
 &= \sum_{i=1}^N \sum_{j=1}^N \sum_{q=1}^{l-1} p(w_1, \dots, w_{q-1}, w_q, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M, n_j(q, l-1), n_i(q, m) | n_1) \\
 &= \sum_{i=1}^N \sum_{j=1}^N \sum_{q=1}^{l-1} p(w_1, \dots, w_{q-1}, n_i(q, m), w_{m+1}, \dots, w_M, w_q, \dots, w_{l-1}, n_j(q, l-1), n_v(l, m) | n_1) \\
 &= \sum_{i=1}^N \sum_{j=1}^N \sum_{q=1}^{l-1} p(w_q, \dots, w_{l-1} | w_1, \dots, w_{q-1}, n_i(q, m), w_{m+1}, \dots, w_M, n_j(q, l-1), n_v(l, m), n_1) \\
 &\quad \cdot p(n_j(q, l-1), n_v(l, m) | w_1, \dots, w_{q-1}, n_i(q, m), w_{m+1}, \dots, w_M, n_1) \\
 &\quad \cdot p(w_1, \dots, w_{q-1}, n_i(q, m), w_{m+1}, \dots, w_M | n_1) \\
 &= \sum_{i=1}^N \sum_{j=1}^N \sum_{q=1}^{l-1} p(w_q, \dots, w_{l-1} | n_j(q, l-1)) \\
 &\quad \cdot p(n_i(q, m) \rightarrow n_j(q, l-1), n_v(l, m) | n_i(q, m)) \\
 &\quad \cdot p(w_1, \dots, w_{q-1}, n_i(q, m), w_{m+1}, \dots, w_M | n_1) \\
 &= \sum_{i=1}^N \sum_{j=1}^N \sum_{q=1}^{l-1} p_j^{Inside}(q, l-1) \cdot p(n_i(q, m) \rightarrow n_j(q, l-1), n_v(l, m) | n_i(q, m)) \cdot p_i^{Outside}(q, m)
 \end{aligned}$$

# Outside algorithm

- The case where  $v$  is the left child becomes (here  $k \neq v$  to avoid counting rules  $n_i \rightarrow n_v n_v$  twice):

*vIsLeftChild*

$$\begin{aligned}
 &= \sum_{i=1}^N \sum_{k=1, k \neq v}^N \sum_{q=m+1}^M p(w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_q, w_{q+1}, \dots, w_M, n_k(m+1, q), n_i(l, q) | n_1) \\
 &= \sum_{i=1}^N \sum_{k=1, k \neq v}^N \sum_{q=m+1}^M p(w_{m+1}, \dots, w_q, n_v(l, m), n_k(m+1, q), w_1, \dots, w_{l-1}, n_i(l, q), w_{q+1}, \dots, w_M | n_1) \\
 &= \sum_{i=1}^N \sum_{k=1, k \neq v}^N \sum_{q=m+1}^M p(w_{m+1}, \dots, w_q | n_v(l, m), n_k(m+1, q), w_1, \dots, w_{l-1}, n_i(l, q), w_{q+1}, \dots, w_M, n_1) \\
 &\quad \cdot p(n_v(l, m), n_k(m+1, q) | w_1, \dots, w_{l-1}, n_i(l, q), w_{q+1}, \dots, w_M, n_1) \\
 &\quad \cdot p(w_1, \dots, w_{l-1}, n_i(l, q), w_{q+1}, \dots, w_M | n_1) \\
 &= \sum_{i=1}^N \sum_{k=1, k \neq v}^N \sum_{q=m+1}^M p(w_{m+1}, \dots, w_q | n_k(m+1, q)) \\
 &\quad \cdot p(n_v(l, m), n_k(m+1, q) | n_i(l, q)) \\
 &\quad \cdot p(w_1, \dots, w_{l-1}, n_i(l, q), w_{q+1}, \dots, w_M | n_1) \\
 &= \sum_{i=1}^N \sum_{k=1, k \neq v}^N \sum_{q=m+1}^M p_k^{\text{Inside}}(m+1, q) \cdot p(n_i(l, q) \rightarrow n_v(l, m), n_k(m+1, q) | n_i(l, q)) \cdot p_i^{\text{Outside}}(l, q)
 \end{aligned}$$

- Because the outside probabilities are of ever larger substrings, at some point this reaches the top level whose probability we know.

# Outside algorithm

- Using both the inside and outside probabilities we can write the probability that a particular nonterminal  $n_v$  will be generated and it will generate substring l-m:

$$\begin{aligned}
 & p(w_1, \dots, w_{l-1}, w_l, \dots, w_m, w_{m+1}, \dots, w_M, n_v(l, m) | n_1) \\
 &= p(w_l, \dots, w_m | w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M, n_1) \cdot p(w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M | n_1) \\
 &= p(w_l, \dots, w_m | n_v(l, m)) \cdot p(w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M | n_1) \\
 &= p_v^{Inside}(l, m) \cdot p_v^{Outside}(l, m)
 \end{aligned}$$

- Or the probability that substring l-m arises from a single nonterminal (no matter which one):

$$\begin{aligned}
 & \sum_{v=1}^N p(w_1, \dots, w_{l-1}, w_l, \dots, w_m, w_{m+1}, \dots, w_M, n_v(l, m) | n_1) \\
 &= \sum_{v=1}^N p_v^{Inside}(l, m) \cdot p_v^{Outside}(l, m)
 \end{aligned}$$



# Most likely parse of a string

- Remember that in the Inside algorithm we had

$$\begin{aligned}
 p_i^{\text{Inside}}(l, m) &= p(w_l, \dots, w_m | n_i) = \sum_{j=1}^N \sum_{k=1}^N \sum_{u=l}^{m-1} p_j^{\text{Inside}}(l, u) p_k^{\text{Inside}}(u+1, m) p(n_i \rightarrow n_j n_k | n_i) \\
 &= \sum_{j=1}^N \sum_{k=1}^N \sum_{u=l}^{m-1} p(n_i \rightarrow (n_j \rightarrow \dots \rightarrow w_l, \dots, w_u, n_k \rightarrow \dots \rightarrow w_{u+1}, \dots, w_m) | n_i) \\
 &= \sum_{j=1}^N \sum_{k=1}^N \sum_{u=l}^{m-1} p(w_l, \dots, w_m, n_j(l, u), n_k(u+1, m) | n_i)
 \end{aligned}$$

- Each sum term is a probability of l-m and a rule  $n_i \rightarrow n_j n_k$ . Thus the maximum a posteriori parse of a string can be computed by dynamic programming.

# Most likely parse of a string

- Computation phase: For each  $l$ - $m$  and each  $n_i$  we can record what continuation rule (and breakpoint  $u$ )  $r_i^{MaxInside}(l, m)$  will give the highest probability  $p_i^{MaxInside}(l, m)$ 
  - Start from bottom up (unary rules): for each  $i$  and  $l$ ,  

$$p_i^{MaxInside}(l, l) = p(n_i \rightarrow w_l)$$
  - Then for each  $i, l, m$ :

$$p_i^{MaxInside}(l, m) = \max_{j=1, \dots, N, \ k=1, \dots, N, \ u=1, \dots, m-1} p_j^{Inside}(l, u) p_k^{Inside}(u+1, m) p(n_i \rightarrow n_j n_k | n_i)$$

$$r_i^{MaxInside}(l, m) = \arg \max_{j=1, \dots, N, \ k=1, \dots, N, \ u=1, \dots, m-1} p_j^{Inside}(l, u) p_k^{Inside}(u+1, m) p(n_i \rightarrow n_j n_k | n_i)$$

# Most likely parse of a string

- The highest-probability parse of the whole string 1-M is then written starting from  $r_1^{MaxInside}(1, M)$  corresponding to the probability  $p_1^{MaxInside}(1, M)$  :
  - Suppose  $r_1^{MaxInside}(1, M) = (j, k, u)$  , then the first rule of the parse is  $n_1(1, M) \rightarrow n_j(1, u) n_k(u+1, M)$
  - Then find the next rules. For the left symbol  $n_j(1, u)$  , if  $r_j^{MaxInside}(1, u) = (j', k', u')$  the rule is  $n_j(1, u) \rightarrow n_{j'}(1, u') n_{k'}(u'+1, u)$
  - For the right symbol  $n_k(u+1, M)$  , if  $r_k^{MaxInside}(u+1, M) = (\tilde{j}, \tilde{k}, \tilde{u})$  the rule is  $n_k(u+1, M) \rightarrow n_{\tilde{j}}(u+1, \tilde{u}) n_{\tilde{k}}(\tilde{u}+1, M)$
  - Continue like this until the substrings are single words l: then  $r_i^{MaxInside}(l, l)$  corresponds to the unary rule  $n_i(l, l) \rightarrow w_l$  at the bottom of the parse

# Properties of PCFGs

- PCFGs can compute the probabilities of different possible parses of a sentence
- PCFGs can be learned from positive data alone (examples of data generated from the grammar) without negative examples (data that does not follow the grammar)
- The probabilistic nature of PCFGs makes it somewhat robust to mistakes and errors in data
- Predictive power can be greater than of a hidden Markov model with the same number of parameters
- However, might still be a worse model than an N-gram which takes local lexical context (nearby words) into account.
- Bias: probability of a smaller parse tree tends to be larger than of a long tree: too much probability for small sentences, whereas real sentences tend to be middle-length.

# Learning PCFG probabilities

- The parameters of a PCFG can be learned from data by maximum likelihood, to maximize the probability of observing a set of training data.
- If we had already-parsed sentences, we could get a maximum-likelihood estimate simply by counting from all sentences the fraction of how many times each rule is taken starting from each nonterminal:

$$\underset{\text{general version}}{\hat{p}(n_i \rightarrow S | n_i)} = \frac{\text{count}(n_i \rightarrow S | n_i)}{\sum_{S_r} \text{count}(n_i \rightarrow S_r | n_i)} = \frac{\text{count}(n_i \rightarrow S, n_i)}{\sum_{S_r} \text{count}(n_i \rightarrow S_r, n_i)} = \frac{\text{count}(n_i \rightarrow S, n_i)}{\text{count}(n_i)}$$

- When ready-made parses of the training-data sentences are not available, they are **latent variables** that must be estimated during the parameter optimization
- The algorithm that does this is a variant of expectation maximization, and is called the **Inside-Outside algorithm**



# Learning PCFG probabilities

- In Chomsky normal form: first, choose the vocabulary, and the number of nonterminals. This defines the set of possible unary and binary rules. Then optimize their parameters (rule probabilities) by maximum likelihood.
- Start by setting the rule probabilities to some initial values
- Idea: we can compute **expected counts of times** that a particular nonterminal, or a particular rule following a nonterminal, is used. Then the maximum likelihood estimate is

$$\hat{p}(n_i \rightarrow n_j n_k | n_i) = \frac{E[\text{count}(n_i \rightarrow n_j n_k, n_i)]}{E[\text{count}(n_i)]}$$

Chomsky normal form  
version, binary rules

$$\hat{p}(n_i \rightarrow w_j | n_i) = \frac{E[\text{count}(n_i \rightarrow w_j, n_i)]}{E[\text{count}(n_i)]}$$

Chomsky normal form  
version, unary rules

- We need to compute these expected counts given the observed string of words.
- We will first consider just one string, then a data set of multiple strings.

# Learning PCFG probabilities

- **Part 1: The denominator.** Recall that

$$p(w_1, \dots, w_{l-1}, \mathbf{w}_l, \dots, \mathbf{w}_m, w_{m+1}, \dots, w_M, \mathbf{n}_v(l, m) | n_1) = p_v^{Inside}(l, m) \cdot p_v^{Outside}(l, m)$$

$$p(w_1, \dots, w_M | n_1) = p_1^{Inside}(1, M)$$

- Then the probability nonterminal  $n_v$  is used when it would lead to l-m is

$$\begin{aligned} & p(\mathbf{n}_v(l, m) | w_1, \dots, w_{l-1}, \mathbf{w}_l, \dots, \mathbf{w}_m, w_{m+1}, \dots, w_M, n_1) \\ &= \frac{p(w_1, \dots, w_{l-1}, \mathbf{w}_l, \dots, \mathbf{w}_m, w_{m+1}, \dots, w_M, \mathbf{n}_v(l, m) | n_1)}{p(w_1, \dots, w_{l-1}, \mathbf{w}_l, \dots, \mathbf{w}_m, w_{m+1}, \dots, w_M | n_1)} = \frac{p_v^{Inside}(l, m) \cdot p_v^{Outside}(l, m)}{p_1^{Inside}(1, M)} \end{aligned}$$

- and the expected count of the nonterminal  $n_v$  is the sum over all positions where it could appear (all substrings it could lead to):

$$E[\text{count}(n_v | w_1, \dots, w_M, n_1)] = \sum_{l=1}^M \sum_{m=l}^M \frac{p_v^{Inside}(l, m) \cdot p_v^{Outside}(l, m)}{p_1^{Inside}(1, M)} = \frac{\sum_{l=1}^M \sum_{m=l}^M p_v^{Inside}(l, m) \cdot p_v^{Outside}(l, m)}{p_1^{Inside}(1, M)}$$

# Learning PCFG probabilities

- **Part 2: Numerator for binary rules.** The probability of all the words, and that the nonterminal  $n_v$  is used in a binary rule leading to nonterminals  $n_j, n_k$  and those in turn to words  $l$ - $u$ ,  $u+1$ - $m$  is:

$$\begin{aligned}
 & p(w_1, \dots, w_{l-1}, \mathbf{w}_l, \dots, \mathbf{w}_u, \mathbf{w}_{u+1}, \dots, \mathbf{w}_m, w_{m+1}, \dots, w_M, \mathbf{n}_j(l, u), \mathbf{n}_k(u+1, m), n_v(l, m) | n_1) \\
 &= p(\mathbf{w}_l, \dots, \mathbf{w}_u, \mathbf{w}_{u+1}, \dots, \mathbf{w}_m, \mathbf{n}_j(l, u), \mathbf{n}_k(u+1, m), w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M | n_1) \\
 &= p(\mathbf{w}_l, \dots, \mathbf{w}_u | \mathbf{w}_{u+1}, \dots, \mathbf{w}_m, \mathbf{n}_j(l, u), \mathbf{n}_k(u+1, m), w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M, n_1) \\
 &\quad \cdot p(\mathbf{w}_{u+1}, \dots, \mathbf{w}_m | \mathbf{n}_j(l, u), \mathbf{n}_k(u+1, m), w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M, n_1) \\
 &\quad \cdot p(\mathbf{n}_j(l, u), \mathbf{n}_k(u+1, m) | w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M, n_1) \\
 &\quad \cdot p(w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M | n_1) \\
 &= p(\mathbf{w}_l, \dots, \mathbf{w}_u | \mathbf{n}_j(l, u)) \\
 &\quad \cdot p(\mathbf{w}_{u+1}, \dots, \mathbf{w}_m | \mathbf{n}_k(u+1, m)) \\
 &\quad \cdot p(\mathbf{n}_j(l, u), \mathbf{n}_k(u+1, m) | n_v(l, m)) \\
 &\quad \cdot p(w_1, \dots, w_{l-1}, n_v(l, m), w_{m+1}, \dots, w_M | n_1) \\
 &= p_j^{\text{Inside}}(l, u) \cdot p_k^{\text{Inside}}(u+1, w_m) \cdot p(n_v(l, m) \rightarrow \mathbf{n}_j(l, u), \mathbf{n}_k(u+1, m) | n_v(l, m)) \cdot p_v^{\text{Outside}}(l, m)
 \end{aligned}$$

(treatment of binary rule continues on next slides)

# Learning PCFG probabilities

- (repeat from previous slide) The probability of all words, and that nonterminal  $n_v$  is used in a binary rule leading to nonterminals  $n_j, n_k$  and those in turn to words  $l-u, u+1-m$  is:

$$p(w_1, \dots, w_{l-1}, \mathbf{w_l}, \dots, \mathbf{w_u}, \mathbf{w_{u+1}}, \dots, \mathbf{w_m}, w_{m+1}, \dots, w_M, \mathbf{n_j(l, u)}, \mathbf{n_k(u+1, m)}, n_v(l, m) | n_1)$$

$$= p_j^{Inside}(l, u) \cdot p_k^{Inside}(u+1, w_m) \cdot p(n_v(l, m) \rightarrow \mathbf{n_j(l, u)}, \mathbf{n_k(u+1, m)} | n_v(l, m)) \cdot p_v^{Outside}(l, m)$$

- Thus the probability  $n_v$  is used leading to  $l-m$ , with  $n_v \rightarrow n_j n_k$  used as a rule, is a sum over possible breakpoints  $u$ :

$$p(w_1, \dots, w_{l-1}, w_l, \dots, w_m, w_{m+1}, \dots, w_M, \mathbf{n_j}, \mathbf{n_k}, n_v(l, m) | n_1)$$

$$= \sum_{u=l}^m p_j^{Inside}(l, u) \cdot p_k^{Inside}(u+1, w_m) \cdot p(n_v(l, m) \rightarrow \mathbf{n_j(l, u)}, \mathbf{n_k(u+1, m)} | n_v(l, m)) \cdot p_v^{Outside}(l, m)$$

(treatment of binary rule continues on next slides)

# Learning PCFG probabilities

- Then the conditional probability, given the text, to use  $n_v$  leading to l-m with  $n_v \rightarrow n_j n_k$  is:

$$\begin{aligned}
 & p(n_v(l, m), \mathbf{n}_j, \mathbf{n}_k | w_1, \dots, w_{l-1}, w_l, \dots, w_m, w_{m+1}, \dots, w_M, n_1) \\
 &= \frac{p(w_1, \dots, w_{l-1}, w_l, \dots, w_m, w_{m+1}, \dots, w_M, n_v(l, m), \mathbf{n}_j, \mathbf{n}_k | n_1)}{p(w_1, \dots, w_{l-1}, w_l, \dots, w_m, w_{m+1}, \dots, w_M | n_1)} \\
 &= \frac{\sum_{u=l}^m p_j^{\text{Inside}}(l, u) \cdot p_k^{\text{Inside}}(u+1, w_m) \cdot p(n_v(l, m) \rightarrow \mathbf{n}_j(l, u), \mathbf{n}_k(u+1, m) | n_v(l, m)) \cdot p_v^{\text{Outside}}(l, m)}{p_1^{\text{Inside}}(1, M)}
 \end{aligned}$$

- Then the expected number of times  $n_v$  is used with  $n_v \rightarrow n_j n_k$  is a sum over possible positions:

$$\begin{aligned}
 & E[\text{count}(n_v, \mathbf{n}_j, \mathbf{n}_k | w_1, \dots, w_M, n_1)] \\
 &= \frac{\sum_{l=1}^M \sum_{m=l+1}^M \sum_{u=l}^m p_j^{\text{Inside}}(l, u) \cdot p_k^{\text{Inside}}(u+1, w_m) \cdot p(n_v(l, m) \rightarrow \mathbf{n}_j(l, u), \mathbf{n}_k(u+1, m) | n_v(l, m)) \cdot p_v^{\text{Outside}}(l, m)}{p_1^{\text{Inside}}(1, M)}
 \end{aligned}$$

(treatment of binary rule continues on next slides)



# Learning PCFG probabilities

- Now that we have the two kinds of expected counts, we can compute the maximum likelihood estimate for the probability of a binary rule:

$$\begin{aligned}
 & \hat{p}(n_v \rightarrow n_j n_k | n_v) \\
 &= \frac{E[\text{count}(n_v, n_j, n_k | w_1, \dots, w_M, n_1)]}{E[\text{count}(n_v | w_1, \dots, w_M, n_1)]} \\
 &= \frac{\sum_{l=1}^M \sum_{m=l+1}^M \sum_{u=l}^m p_j^{\text{Inside}}(l, u) \cdot p_k^{\text{Inside}}(u+1, w_m) \cdot p(n_v(l, m) \rightarrow n_j(l, u), n_k(u+1, m) | n_v(l, m)) \cdot p_v^{\text{Outside}}(l, m)}{\sum_{l=1}^M \sum_{m=l}^M p_v^{\text{Inside}}(l, m) \cdot p_v^{\text{Outside}}(l, m)}
 \end{aligned}$$

- Next we will do a similar computation for unary rules.

(treatment of unary rule continues on next slides)

# Learning PCFG probabilities

- **Part 3: numerator for unary rules.** The probability of all the words, and  $n_v$  is used in a unary rule leading to a word whose dictionary index is  $w$ , in position  $m$ , is:

$$\begin{aligned}
 & p(w_1, \dots, w_{m-1}, w_m, w_{m+1}, \dots, w_M, n_v(m, m) \rightarrow w, n_v(m, m) | n_1) \\
 &= p(w_m, n_v(m, m) \rightarrow w, w_1, \dots, w_{m-1}, n_v(m, m), w_{m+1}, \dots, w_M | n_1) \\
 &= p(w_m | n_v(m, m) \rightarrow w, w_1, \dots, w_{m-1}, n_v(m, m), w_{m+1}, \dots, w_M, n_1) \\
 &\cdot p(n_v(m, m) \rightarrow w | w_1, \dots, w_{m-1}, n_v(m, m), w_{m+1}, \dots, w_M, n_1) \\
 &\cdot p(w_1, \dots, w_{m-1}, n_v(m, m), w_{m+1}, \dots, w_M | n_1) \\
 &= p(w_m | n_v(m, m) \rightarrow w) \cdot p(n_v(m, m) \rightarrow w | n_v(m, m)) \cdot p(w_1, \dots, w_{m-1}, n_v(m, m), w_{m+1}, \dots, w_M | n_1) \\
 &= \delta(w_m, w) \cdot p(n_v \rightarrow w | n_v) \cdot p_v^{Outside}(m, m) \\
 &= \delta(w_m, w) \cdot p_v^{Inside}(m, m) \cdot p_v^{Outside}(m, m)
 \end{aligned}$$

- The conditional probability is then:

$$\begin{aligned}
 & p(n_v(m, m) \rightarrow w, n_v(m, m) | w_1, \dots, w_{m-1}, w_m, w_{m+1}, \dots, w_M, n_1) \\
 &= \frac{\delta(w_m, w) \cdot p_v^{Inside}(m, m) \cdot p_v^{Outside}(m, m)}{p_1^{Inside}(1, M)}
 \end{aligned}$$

(treatment of unary rule continues on next slides)

# Learning PCFG probabilities

- The expected count is then a sum over positions  $m$ :

$$\begin{aligned}
 & E[\text{count}(n_v \rightarrow w, n_v | w_1, \dots, w_M, n_1)] \\
 & \quad \sum_{m=1}^M \delta(w_m, w) \cdot p_v^{\text{Inside}}(m, m) \cdot p_v^{\text{Outside}}(m, m) \\
 & = \frac{\sum_{m=1}^M \delta(w_m, w) \cdot p_v^{\text{Inside}}(m, m) \cdot p_v^{\text{Outside}}(m, m)}{p_1^{\text{Inside}}(1, M)}
 \end{aligned}$$

- The probability estimate of the unary rule is then

$$\begin{aligned}
 & \hat{p}(n_v \rightarrow w | n_v) \\
 & = \frac{E[\text{count}(n_v \rightarrow w, n_v | w_1, \dots, w_M, n_1)]}{E[\text{count}(n_v | w_1, \dots, w_M, n_1)]} \\
 & \quad \sum_{m=1}^M \delta(w_m, w) \cdot p_v^{\text{Inside}}(m, m) \cdot p_v^{\text{Outside}}(m, m) \\
 & = \frac{\sum_{l=1}^M \sum_{m=l}^M p_v^{\text{Inside}}(l, m) \cdot p_v^{\text{Outside}}(l, m)}{\sum_{l=1}^M \sum_{m=l}^M p_v^{\text{Inside}}(l, m) \cdot p_v^{\text{Outside}}(l, m)}
 \end{aligned}$$

(treatment of unary rule continues on next slides)

# Learning PCFG probabilities

- When we have multiple strings  $w^t = [w_1^{(t)}, \dots, w_{M(t)}^{(t)}]$ ,  $t = 1, \dots, T$ , and we assume they are independently generated, we can take expected counts from each:

$$E_v^{(t)} = E[\text{count}(n_v | w_1^{(t)}, \dots, w_{M(t)}^{(t)}, n_1)] = \frac{\sum_{l=1}^{M(t)} \sum_{m=l}^{M(t)} p_v^{\text{Inside}(t)}(l, m) \cdot p_v^{\text{Outside}(t)}(l, m)}{p_1^{\text{Inside}(t)}(1, M)}$$

$$\begin{aligned} E_{v,j,k}^{(t)} &= E[\text{count}(n_v, \textcolor{red}{n}_j, \textcolor{green}{n}_k | w_1^{(t)}, \dots, w_{M(t)}^{(t)}, n_1)] \\ &= \frac{\sum_{l=1}^{M(t)} \sum_{m=l+1}^{M(t)} \sum_{u=l}^m \textcolor{red}{p}_j^{\text{Inside}(t)}(l, u) \cdot \textcolor{green}{p}_k^{\text{Inside}(t)}(u+1, w_m) \cdot p(n_v(l, m) \rightarrow \textcolor{red}{n}_j(l, u), \textcolor{green}{n}_k(u+1, m) | n_v(l, m)) \cdot p_v^{\text{Outside}(t)}(l, m)}{p_1^{\text{Inside}(t)}(1, M)} \end{aligned}$$

$$\begin{aligned} E_{v,w}^{(t)} &= E[\text{count}(\textcolor{red}{n}_v \rightarrow \textcolor{red}{w}, n_v | w_1^{(t)}, \dots, w_{M(t)}^{(t)}, n_1)] \\ &= \frac{\sum_{m=1}^{M(t)} \delta(w_m^{(t)}, w) \cdot p_v^{\text{Inside}(t)}(m, m) \cdot p_v^{\text{Outside}(t)}(m, m)}{p_1^{\text{Inside}(t)}(1, M)} \end{aligned}$$

where  $p_v^{\text{Inside}(t)}(l, m)$  and  $p_v^{\text{Outside}(t)}(l, m)$  are inside and outside probabilities computed for string  $t$ .

# Learning PCFG probabilities

- Then the probability estimates for binary and unary rules are again ratios of expected counts, but the numerator and denominator are summed over all the strings:

$$\hat{p}(n_v \rightarrow w | n_v) = \frac{\sum_{t=1}^T E[\text{count}(n_v \rightarrow w, n_v | w_1^{(t)}, \dots, w_{M(t)}^{(t)}, n_1)]}{\sum_{t=1}^T E[\text{count}(n_v | w_1^{(t)}, \dots, w_{M(t)}^{(t)}, n_1)]} = \frac{\sum_{t=1}^T E_{v,w}^{(t)}}{\sum_{t=1}^T E_v^{(t)}}$$

$$\hat{p}(n_v \rightarrow n_j n_k | n_v) = \frac{\sum_{t=1}^T E[\text{count}(n_v, n_j, n_k | w_1^{(t)}, \dots, w_{M(t)}^{(t)}, n_1)]}{\sum_{t=1}^T E[\text{count}(n_v | w_1^{(t)}, \dots, w_{M(t)}^{(t)}, n_1)]} = \frac{\sum_{t=1}^T E_{v,j,k}^{(t)}}{\sum_{t=1}^T E_v^{(t)}}$$

- The right-hand side of these rules depend on previous estimates of the rule probabilities, to give a new estimate.
- The updates must be repeated iteratively many times (recomputing the expectations each time), until the change in the rule probabilities is small enough. This is the **inside-outside algorithm**.



# Learning PCFG probabilities

- Properties of the inside-outside algorithm:
  - It is guaranteed that the algorithm **never decreases the total likelihood** (product of probabilities of the observed strings 1-T): each iteration either increases the likelihood or keeps it the same.
  - It is **not guaranteed to converge to a global maximum** of the likelihood, only to a local maximum which depends on the initialization of the probabilities.
    - There are some general approaches in machine learning, such as simulated annealing, that aim to more comprehensively search for a good local maximum.
  - It has a **high computational cost**: for each sentence  $t$  the cost of each iteration is  $O(M^3(t)N^3)$  where  $M(t)$  is the length of the sentence.
  - It is not guaranteed that resulting nonterminals and rules resemble those typically used to describe the language (e.g. that there would be nonterminals corresponding to noun phrases, verb phrases etc.)
- Some observations:
  - Charniak, 1993: in an experiment with artificial data, each of 300 trials found a different local optimum (sensitive to initialization)
  - Lari and Young, 1990: good results may require more nonterminals than are theoretically needed to describe the language. In artificial data generated with  $N$  nonterminals they needed  $3*N$  nonterminals to learn the grammar.