How to undestand the formula quickly

Viterbi-Algorithmus (Bigramm-Tagger)

1. Initialisierung: $\delta_t(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$

2. Berechnung: (für $1 \leq k \leq n+1$)

$$\begin{aligned} \delta_t(k) &= \max_{t'} \delta_{t'}(k-1)\, p(t|t')\, p(w_k|t) \\ \psi_t(k) &= \arg\max_{t'} \delta_{t'}(k-1)\, p(t|t')\, p(w_k|t) \end{aligned}$$

3. Ausgabe: (für $0 < k \leq n$)

$$\begin{aligned} t_{n+1} &= \langle /s \rangle \\ t_k &= \psi_{t_{k+1}}(k+1) \quad \text{für } 1 \leq k \leq n \end{aligned}$$

t, t' sind Tags, k sind Wortpositionen, $\delta_t(k)$ sind die Viterbiwahrscheinlichkeiten und $\psi_t(k)$ sind die besten Vorgängertags

- In this formula, t‘ is the tag at the previous position and t is the tag at the current position, but it can happen that you get a formula that uses other variables. So, before you compute anything, make sure you understand the formula correctly.
- Here are some steps that will help you with that.

- First, write a sample sentence such as „ I can can“ and add some sample tags for each word.
- add the starting and ending symbols according to the n-gram order of your HMM model
  - bigram (add one starting and one ending symbol)
  - trigram (add 2 starting and one or two ending symbols, depending on the given formula)
- then, write down the positions
- pick a position to be an example of position k, mark it and also mark t
- mark position k-1 and mark t‘
- look at the context prob p(tag|tag) to check the order of each variable. E.g. p(t| t‘) means t‘ followed by t.
- then try computing the Viterbi prob for position k
- and maybe try computing psi too

Example: I want to understand this Trigram formula

Viterbi-Algorithmus (Trigramm-Tagger)

- Beim Trigramm-Tagger entspricht jeder Zustand des Hidden-Markow-Modelles nicht einem einzelnen Tag, sondern einem Tagpaar.
- Übergänge gibt es nur zwischen Zuständen (t, t') und (t'', t''') mit $t' = t''$

1. Initialisierung: $\delta_{t',t}(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \text{ und } t' = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$

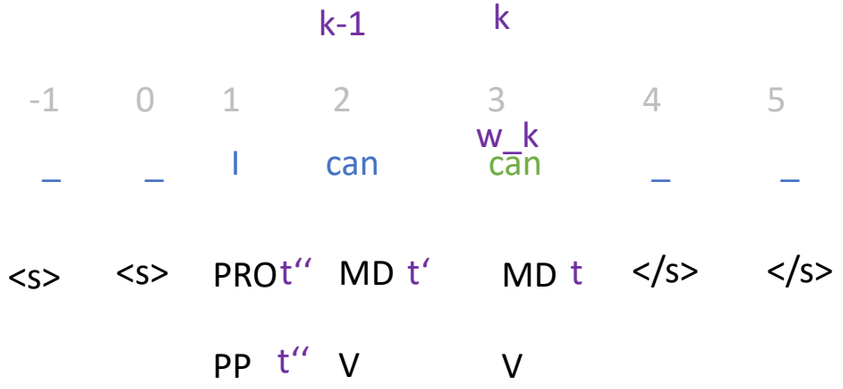
2. Berechnung: (für $0 < k \leq n+2$)

$$\delta_{t',t}(k) = \max_{t''} \delta_{t'',t'}(k-1)\, p(t|t'', t')\, p(w_k|t)$$

$$\psi_{t',t}(k) = \arg\max_{t''} \delta_{t'',t'}(k-1)\, p(t|t'', t')\, p(w_k|t)$$

Wir fügen hier 2 Endetags hinzu und iterieren bis n+2. Das hat den Vorteil, dass wir das letzte Tag direkt in $\psi_{\langle/s\rangle,\langle/s\rangle}(n+2)$ finden. Andernfalls müssten wir erst in der Spalte n+1 durch Maximierung über t nach dem Eintrag $\delta_{t,\langle/s\rangle}(n+1)$ mit der größten Viterbi-Wahrscheinlichkeit suchen, um von dort ausgehend die beste Tagfolge zu extrahieren.

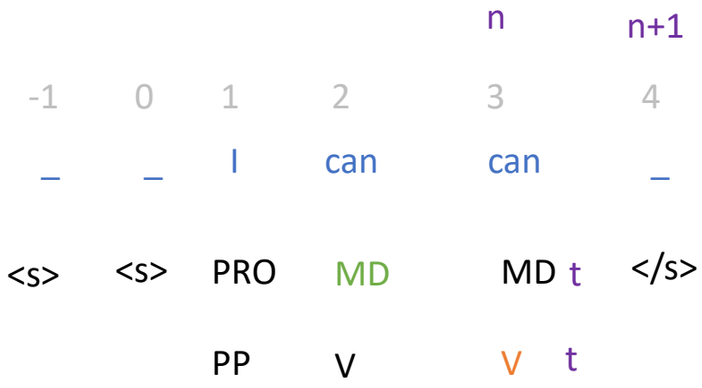
- The formula says we should compute the Viterbi prob until position n+2, meaning we will add 2 ending symbols.
- Here I choose position 3 as an example and want to compute vit_MD, MD(3)



vit_MD,MD(3) = max (vit_PRO,MD(2) p(MD|PRO, MD) p(can | MD),
vit_PP, MD (2) p(MD|PP,MD) p(can | MD))

psi_MD,MD(3) = either PRO or PP (tags from position 1) depending on which tag leads to a higher value of $\delta_{t'',t'}(k-1)\, p(t|t'', t')\, p(w_k|t)$

Alternative : compute Viterbi prob until n+1 (instead of n+2)



Viterbi-Algorithmus (Trigramm-Tagger)

- Beim Trigramm-Tagger entspricht jeder Zustand des Hidden-Markow-Modelles nicht einem einzelnen Tag, sondern einem Tagpaar.
- Übergänge gibt es nur zwischen Zuständen (t, t') und (t'', t''') mit $t' = t''$

1. Initialisierung: $\delta_{t',t}(0) = \begin{cases} 1 & \text{falls } t = \langle s \rangle \text{ und } t' = \langle s \rangle \\ 0 & \text{sonst} \end{cases}$

2. Berechnung: (für $0 < k \leq n+2$)

$$\delta_{t',t}(k) = \max_{t''} \delta_{t'',t'}(k-1)\, p(t|t'',t')\, p(w_k|t)$$
$$\psi_{t',t}(k) = \arg\max_{t''} \delta_{t'',t'}(k-1)\, p(t|t'',t')\, p(w_k|t)$$

Wir fügen hier 2 Endetags hinzu und iterieren bis n+2. Das hat den Vorteil, dass wir das letzte Tag direkt in $\psi_{\langle/s\rangle,\langle/s\rangle}(n+2)$ finden. Andernfalls müssten wir erst in der Spalte n+1 durch Maximierung über t nach dem Eintrag $\delta_{t,\langle/s\rangle}(n+1)$ mit der größten Viterbi-Wahrscheinlichkeit suchen, um von dort ausgehend die beste Tagfolge zu extrahieren.

To get the best tag sequence, we compute as follows.
- start with position n+1, compute
  arg max over t (MD and V) of (vit_MD, </s> (4), vit_V , </s> (4))

Suppose vit_V, </s> (4) is higher than *vit_MD, </s> (4)*.
-Then, we would then assign the best tag for position n (or 3) to be V (this step is not in the formula)
-then we will compute the best tag for position 2 by looking at psi_V, </s>(4). Suppose it is MD.
-then compute the best tag for position 1 by looking at psi_MD,V(3), and we will get either PRO or PP.
-then we stop.

```python
def viterbi(self, words):
    ''' berechnet die beste Tagfolge für eine gegebene Wortfolge '''
    words =  [''] + words + ['']  # Grenztokens hinzufügen

    # Initialisierung der Viterbi-Tabelle
    vitscore = [dict() for _ in range(len(words))]  # speichert logarithmierte Werte
    bestprev = [dict() for _ in range(len(words))]  # speichert die besten Vorgänger
    vitscore[0][('<s>','<s>')] = 0.0  # =log(1)
    for i in range(1, len(words)):
        lexprobs = self._lex_probs(words[i])  # die möglichen Tags nachschlagen
        for tag, lexprob in lexprobs:
            for tagpair in vitscore[i-1]:
                tag1, tag2 = tagpair  # Kontext-Tags
                p = self._context_prob(tagpair, tag) * lexprob / self._apriori_tag_prob[tag]
                p = vitscore[i-1][tagpair] + log(p)
                newtagpair = (tag2, tag)
                if newtagpair not in vitscore[i] or vitscore[i][newtagpair] < p:
                    vitscore[i][newtagpair] = p
                    bestprev[i][newtagpair] = tagpair
    # in der letzten Spalte das Tagpaar mit der höchsten Bewertung suchen
    tagpair = max(vitscore[-1], key=vitscore[-1].get)
    # beste Tagfolge extrahieren
    result_tags = []
    for i in range(len(words)-1, 1, -1):
        result_tags.append(tagpair[0])
        tagpair = bestprev[i][tagpair]
    return reversed(result_tags)  # Tagfolge umdrehen
```

The alternative version is the method used in the solution code

backward prob can be a bit tricky

Die Backward-Wahrscheinlichkeit $\beta_t(k)$ des Tags t an Position k ist die Summe der Wahrscheinlichkeiten aller Tagfolgen t_k^{n+1} für die Teil-Wortfolge w_{k+1}^n, die mit dem Tag t beginnen und dem Tag $\langle/s\rangle$ enden.

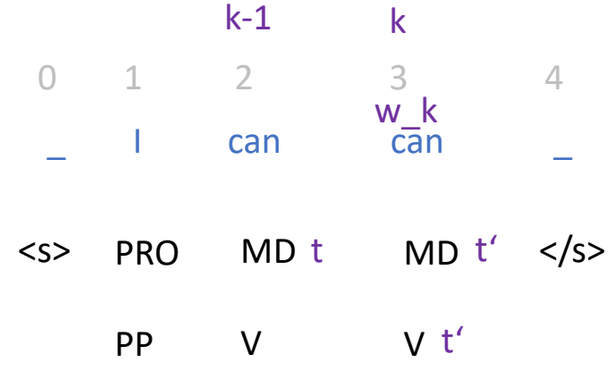
Die lexikalische Wk. $p(w_k|t_k)$ ist in $\beta_t(k)$ nicht enthalten!

Formeln für die rekursive Berechnung im Fall des Bigramm-Taggers:

$$\beta_t(n+1) = \begin{cases} 1 & \text{falls } t = \langle/s\rangle \\ 0 & \text{sonst} \end{cases}$$

$$\beta_t(k-1) = \sum_{t' \in T} p(t'|t)\, p(w_k|t')\, \beta_{t'}(k) \quad \text{für } 0 < k \leq n+1$$

compute from k= n+1 to k=1



initial step, set backward_</s> (4) = 1
Then compute the following.

k=4 , compute backward_t(4-1) or backward_t(3)
k=3, compute backward_t(2)
k=2, compute backward_t(1)
k=1, compute backward _t(0)

I choose as an example k-1 = 2 (meaning k=3)

backward_MD (2) = p(MD|MD) p(can |MD) backward_MD(3) +
p(V|MD) p(can | V) backward_V(3)

Aufgabe 2) Ein HMM ist gegeben durch die (unvollständige) Tabelle:

	PRO	MD	N	$\langle s \rangle$	we	can	ϵ
PRO	0.1	0.3	0.1	0.1	0.2	0	0
MD	0.1	0.0	0.1	0.1	0	0.3	0
N	0.1	0.2	0.2	0.2	0	0.1	0
$\langle s \rangle$	0.2	0.1	0.1	0	0	0	1

mit $p(\langle s \rangle | PRO) = 0.1$ und $p(I|PRO) = 0.2$.

Berechnen Sie für die Tokenfolge *"we can"* und das obige HMM die **Viterbi**-Wahrscheinlichkeiten $\delta_t(i)$ und die besten Vorgänger-Tags $\psi_t(i)$ nach den Formeln:

$$
\begin{aligned}
\delta_t(0) &= \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases} \\
\delta_t(k) &= \max_{t'} \; \delta_{t'}(k-1)\; p(t|t')\; p(w_k|t) \qquad \text{für } 0 < k \leq n+1 \\
\psi_t(k) &= \arg\max_{t'} \; \delta_{t'}(k-1)\; p(t|t')\; p(w_k|t) \qquad \text{für } 0 < k \leq n+1
\end{aligned}
$$

Schreiben Sie nicht nur das Ergebnis hin, sondern zeigen Sie auch den Rechenweg.
Extrahieren Sie dann die beste **Tagfolge** nach den Formeln

$$
\begin{aligned}
t_n &= \psi_{\langle s \rangle}(n+1) \\
t_k &= \psi_{t_{k+1}}(k+1) \qquad \text{für } n > k > 0
\end{aligned}
$$

(5 Punkte)

Lösung

Aufgabe 2)

$$
\begin{aligned}
\delta_{\langle s\rangle}(0) &= 1 \\
\delta_{PRO}(1) &= \delta_{\langle s\rangle}(0)\, p(PRO|\langle s\rangle)\, p(we|PRO) = 1 \cdot 0.2 \cdot 0.2 = 0.04 \\
\psi_{PRO}(1) &= \langle s\rangle \\
\delta_{MD}(2) &= \delta_{PRO}(1)\, p(MD|PRO)\, p(can|MD) = 0.04 \cdot 0.3 \cdot 0.3 = 0.0036 \\
\psi_{MD}(2) &= PRO \\
\delta_{N}(2) &= \delta_{PRO}(1)\, p(N|PRO)\, p(can|N) = 0.04 \cdot 0.1 \cdot 0.1 = 0.0004
\end{aligned}
$$

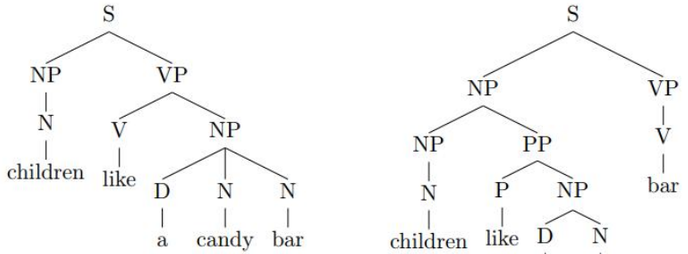
$$
\begin{aligned}
\psi_{N}(2) &= PRO \\
\delta_{\langle s\rangle}(3) &= max(\delta_{MD}(2)\, p(\langle s\rangle|MD)\, p(\epsilon|\langle s\rangle), \delta_{N}(2)\, p(\langle s\rangle|N)\, p(\epsilon|\langle s\rangle)) \\
&= max(0.0036 \cdot 0.1 \cdot 1, 0.0004 \cdot 0.2 \cdot 1) \\
&= max(0.00036, 0.00008) = 0.00036 \\
\psi_{\langle s\rangle}(3) &= MD \\
t_2 &= \psi_{\langle s\rangle}(3) = MD \\
t_1 &= \psi_{MD}(2) = PRO
\end{aligned}
$$

Ergebnis-Tagfolge: PRO MD

PCFGs: Statistisches Parsen

Syntaktische Ambiguitäten

Problem: (Symbolische) Parser liefern für viele Sätze mehrere mögliche syntaktische Analysen (= Parsebäume).



Lösung: Statistische Desambiguierung durch Auswahl des wahrscheinlichsten Parsebaumes.

Grammatik

S	→ NP VP	
VP	→ V NP	
VP	→ V	
NP	→ D N1	
N1	→ A N1	
N1	→ N	
D	→ the	
N	→ bell	
V	→ rings ...	

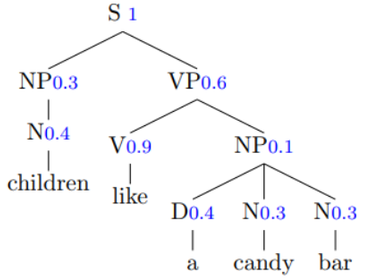
When we parse a sentence (to get a syntactic analysis) using context-free grammar, there can be more than one analysis.

To find the best unique parse tree (analysis), we will use PCFG.
= parse using context-free grammar where each rule has a probability assigned.

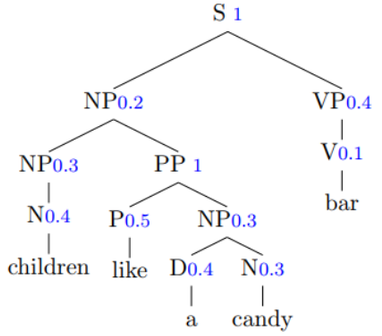
PCGF

Beispiele für Parsebaum-Wahrscheinlichkeiten

S	→ NP VP	1
NP	→ NP PP	0.2
NP	→ D N	0.3
NP	→ N	0.3
NP	→ D N N	0.1
NP	→ N N	0.1
VP	→ V NP	0.6
VP	→ V	0.4
PP	→ P NP	1
D	→ the	0.6
D	→ a	0.4
N	→ children	0.4
N	→ candy	0.3
N	→ bar	0.3
V	→ bar	0.1
V	→ like	0.9
P	→ like	0.5
P	→ for	0.5



$p_1 = 0.0002333$



$p_2 = 0.0000173$

Probabilistische kontextfreie Grammatiken

Wahrscheinlichkeit eines Parsebaumes T mit Linksableitung $r_1, ..., r_n$

$$p(T) = p(r_1, .., r_n) = \prod_{i=1}^{n} p(r_i | r_1, ..., r_{i-1})$$

$$= \prod_{i=1}^{n} p(r_i | nextcat(r_1, ..., r_{i-1}))$$

$nextcat(r_1, ..., r_k)$: Kategorie des Nichtterminalknotens, der als nächster expandiert wird im partiellen Baum von $r_1, ..., r_{i-1}$

Es gilt: $p(A \to \alpha | B) = 0$ falls $A \neq B$ für $A, B \in V$ und $\alpha \in (V \cup \Sigma)^*$

Daraus folgt: $p(r_1, ..., r_k) = 0$ falls $r_1, ..., r_k$ keine (partielle) Linksableitung ist.

Meistens wird einfach $p(A \to \alpha)$ statt $p(A \to \alpha | A)$ geschrieben. Damit vereinfacht sich die obige Definition zu:

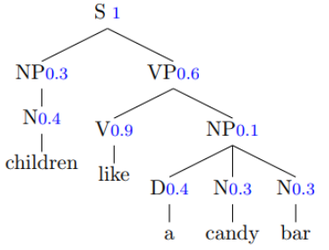
$$p(T) = p(r_1, ..., r_n) = \prod_{i=1}^{n} p(r_i)$$

Diese Formel gilt aber nur, wenn $r_1, ..., r_n$ tatsächlich eine Linksableitung ist!

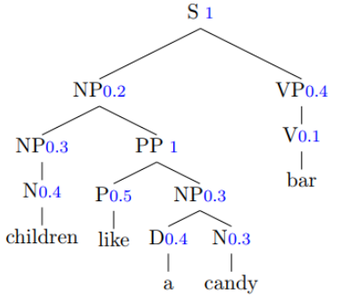
We can represent a parse tree with a set of ordered grammar rules (Linksableitung)
-with these rules, we will keep expanding the leftmost non-terminal symbol.
-In the PCFG approach, each rule is assigned a probability.
-When we multiply the probabilities of the rules together, we get p(T), the probability of the tree.
-If one sentence has 2 parse trees, the best parse is the one with the highest probability.

Beispiele für Parsebaum-Wahrscheinlichkeiten

S	→ NP VP	1
NP	→ NP PP	0.2
NP	→ D N	0.3
NP	→ N	0.3
NP	→ D N N	0.1
NP	→ N N	0.1
VP	→ V NP	0.6
VP	→ V	0.4
PP	→ P NP	1
D	→ the	0.6
D	→ a	0.4
N	→ children	0.4
N	→ candy	0.3
N	→ bar	0.3
V	→ bar	0.1
V	→ like	0.9
P	→ like	0.5
P	→ for	0.5



$p_1 = 0.0002333$



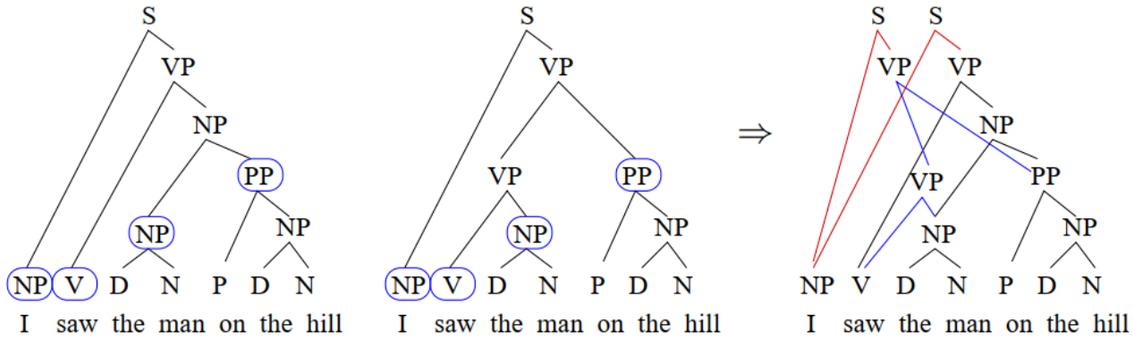
$p_2 = 0.0000173$

Grammatik	Satzform	Aktion
S → NP VP	S	S → NP VP
VP → V NP	NP VP	NP → D N1
VP → V	D N1 VP	D → the
NP → D N1	the N1 VP	N1 → N
N1 → A N1	the N VP	N → bell
N1 → N	the bell VP	VP → V
D → the	the bell V	V → rings
N → bell	the bell rings	
V → rings ...		

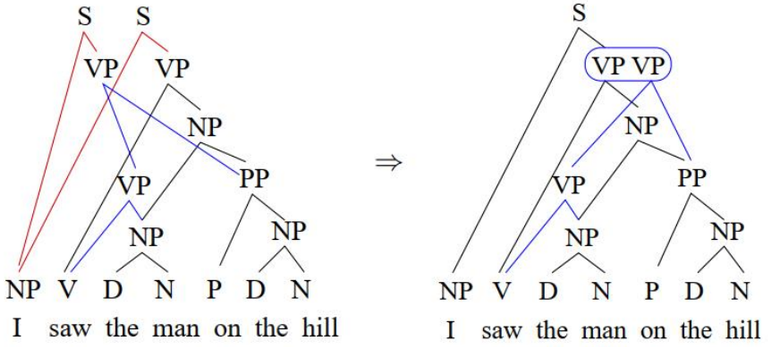
Parsewald

- eine kompakte Repräsentation der Menge aller Analysen eines Satzes
- ergibt sich aus der Menge der Parsebäume durch 2 Operationen:
  1. Zusammenfassen gemeinsamer Teilbäume
  2. Zusammenfassen von Parsebäumen, die sich nur in einem Teilbaum unterscheiden.

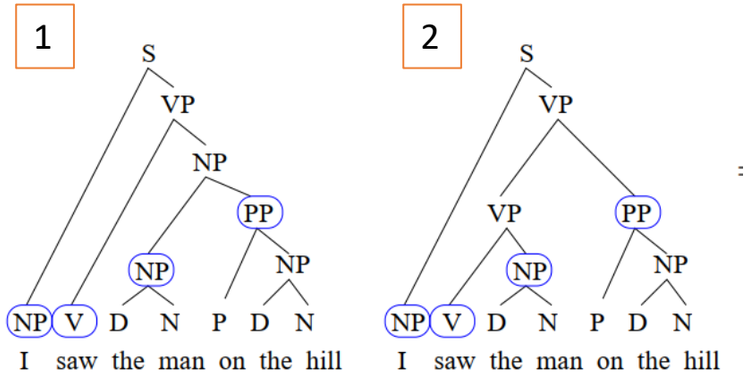
Zusammenfassen (maximaler) gemeinsamer Teilbäume



Die blauen Knoten der beiden Parsebäume werden jeweils zu einem Knoten zusammengefasst.

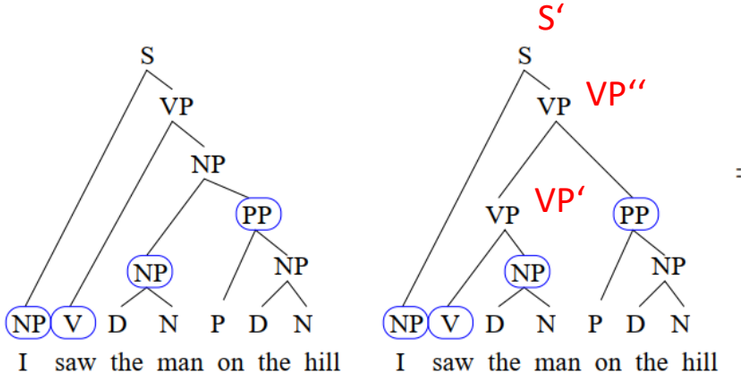


Hier werden die beiden roten Teilbäume zusammengefasst.
Die beiden VP-Teilbäume werden in einem ambigen VP-Knoten vereinigt.
Die blauen Kanten gehören zum 2. Parsebaum.

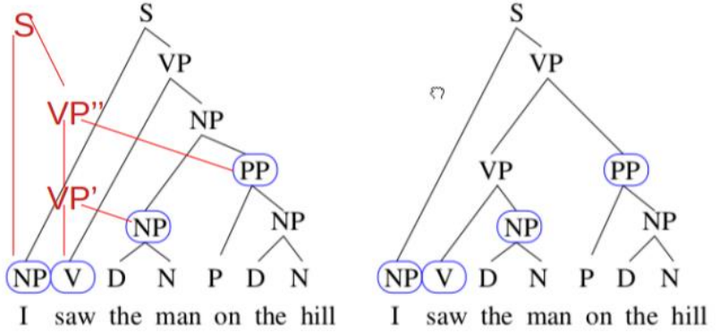


How to merge tree 1 with tree 2

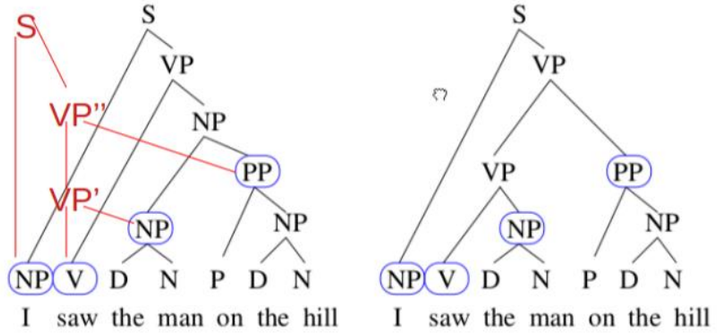
- mark nodes that are identical in both trees

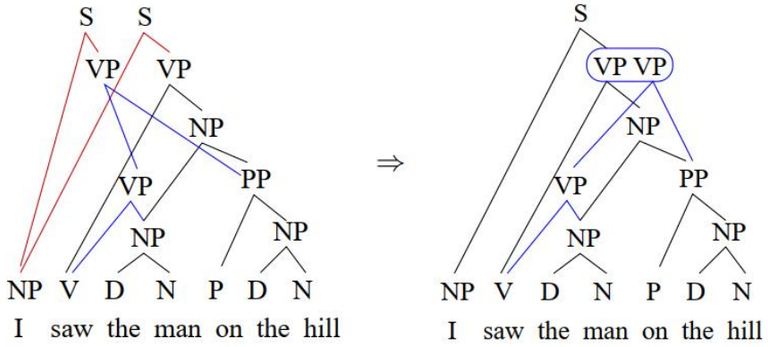


- In tree 2, write down the rest nodes. Starting with the bottommost node to the topmost node
  - VP‘
  - VP‘‘
  - S‘



- add these nodes to tree1 in the correct order
  - namely, VP‘, VP‘‘, S‘





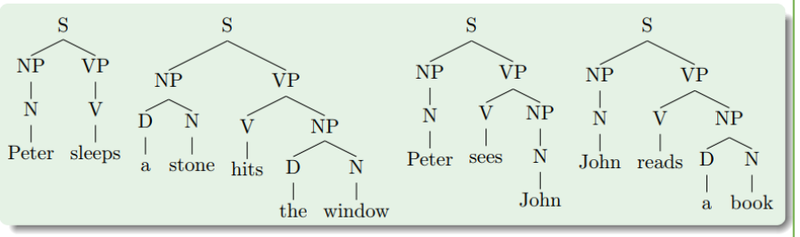
group the same symbol together
Here S and VP are grouped together

Hier werden die beiden roten Teilbäume zusammengefasst.
Die beiden VP-Teilbäume werden in einem ambigen VP-Knoten vereinigt.
Die blauen Kanten gehören zum 2. Parsebaum.

Parameterschätzung(= estimating p for each grammar rule)

- Baumbank-Traning (supervised)
- EM-Training (unsupervised)

Baumbank-Training



Extraktion der Grammatikregeln und ihrer Häufigkeiten:

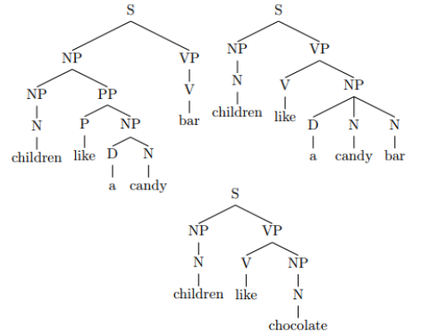
S → NP VP	4	1
VP → V NP	3	0.75
VP → V	1	0.25
NP → D N	3	0.43
NP → N	4	0.57

D → a	2	0.67
D → the	1	0.33
V → sleeps	1	0.25
V → hits	1	0.25
V → sees	1	0.25
V → reads	1	0.25

N → Peter	2	0.29
N → John	2	0.29
N → stone	1	0.14
N → window	1	0.14
N → book	1	0.14

Training auf unannotierten Daten

children like a candy bar
children like chocolate



Regel	*f*	*p*
S → NP VP	2	1
NP → D N	0.5	0.11
NP → D N N	0.5	0.11
NP → N	3	0.67
NP → NP PP	0.5	0.11
VP → V	0.5	0.25
VP → V NP	1.5	0.75
PP → P NP	0.5	1
D → a	1	1
D → the	0	0
N → children	2	0.44
N → bar	0.5	0.11
N → candy	1	0.22
N → chocolate	1	0.22
V → bar	0.5	0.25
V → like	1.5	0.75
P → like	0.5	1

\+ Die Wahrscheinlichkeiten von "candy" und "chocolate" stimmen jetzt.
– Die gute und die schlechte Analyse des ersten Satzes haben dasselbe Gewicht.

Baumbank-Traning (supervised)

Parameterschätzung

Die Regelwahrscheinlichkeiten werden aus Regelhäufigkeiten geschätzt.

Die Regelhäufigkeiten können auf zwei Arten erhalten werden:

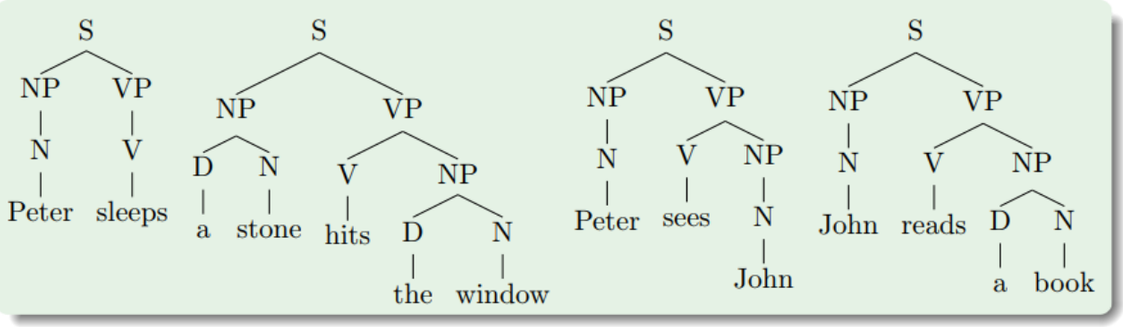
1. durch Zählen der Regeln in einem manuell mit Parsebäumen annotierten Korpus (Baumbank)
2. aus einem automatisch geparsten und desambiguierten Korpus (EM-Training)

Schätzung der Regelwahrscheinlichkeiten mit relativen Häufigkeiten

$$p(A \to \alpha) = \frac{f_{A\to\alpha}}{\sum_\beta f_{A\to\beta}}$$

Hier ist eine Parameterglättung nicht unbedingt notwendig, da meist alle Regeln beobachtet sind.

Baumbank-Training



Extraktion der Grammatikregeln und ihrer Häufigkeiten:

S → NP VP	4	1
VP → V NP	3	0.75
VP → V	1	0.25
NP → D N	3	0.43
NP → N	4	0.57

D → a	2	0.67
D → the	1	0.33
V → sleeps	1	0.25
V → hits	1	0.25
V → sees	1	0.25
V → reads	1	0.25

N → Peter	2	0.29
N → John	2	0.29
N → stone	1	0.14
N → window	1	0.14
N → book	1	0.14

EM-Training (unsupervised)

Training auf unannotierten Daten

Ansatz 1 (schlecht)

1. Die Sätze des Trainingskorpus werden (symbolisch) geparst.
2. Alle Analysen aller Sätze werden zusammen als "Baumbank" gespeichert.
3. Die Regelwahrscheinlichkeiten werden aus der Baumbank geschätzt.

Problem: Je mehr Analysen ein Satz besitzt, desto höher ist sein Einfluss auf die Grammatik.

Beispiel: Ein nicht ambiger Satz liefert nur einen Parsebaum für die Baumbank. Ein hoch-ambiger Satz kann dagegen Tausende Parsebäume liefern, die sich oft ähneln. Damit hat der ambige Satz einen tausendmal höheren Einfluss auf die Grammatikwahrscheinlichkeiten als der eindeutige Satz.

Training auf unannotierten Daten

Ansatz 2: ähnlich zu Ansatz 1 aber:

Die Regelhäufigkeiten, die aus den Analysen eines Satzes extrahiert wurden, werden durch die Zahl der Analysen geteilt.

⇒ Alle Sätze erhalten dasselbe Gewicht.

⇒ Alle Analysen eines Satzes erhalten dasselbe Gewicht.

Problem: Gute Analysen erhalten dasselbe Gewicht wie schlechte.

Training auf unannotierten Daten

Ansatz 3: Gewichtung der Analysen mit ihrer Güte.

Als Maß für die Güte der Analyse t dient ihre Wahrscheinlichkeit gegeben den Satz s:

$$p(t|s) = \frac{p(s|t)p(t)}{p(s)} = \frac{p(t)}{p(s)} = \frac{p(t)}{\sum_{t' \in T(s)} p(t')}$$

$p(s|t) = 1$, weil die Wortfolge s durch die Terminalsymbole des Parsebaum t gegeben ist.
$T(s)$ ist hier die Menge der Analysen des Satzes s.

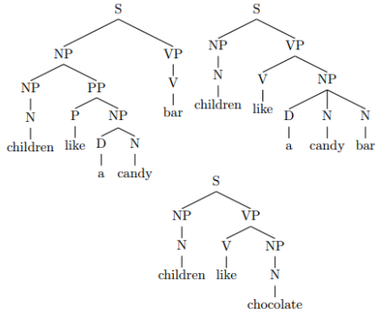
Die aus t extrahierten Regelhäufigkeiten werden mit $p(t|s)$ multipliziert.

Aber: Zur Berechnung der Regelwahrscheinlichkeiten $p(t)$ brauchen wir schon die Regelhäufigkeiten, die hier erst bestimmt werden sollen.

⇒ Henne-Ei-Problem
⇒ EM-Training

Training auf unannotierten Daten

children like a candy bar
children like chocolate

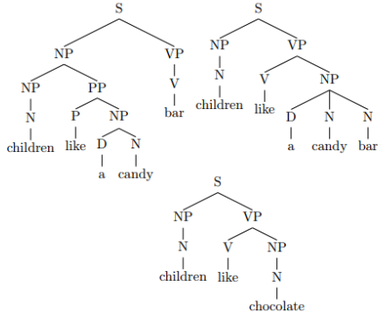


Regel	f	p
S → NP VP	3	1
NP → D N	1	0.14
NP → D N N	1	0.14
NP → N	4	0.57
NP → NP PP	1	0.14
VP → V	1	0.33
VP → V NP	2	0.67
PP → P NP	1	1
D → a	2	1
D → the	0	0
N → children	3	0.43
N → bar	1	0.14
N → candy	2	0.29
N → chocolate	1	0.14
V → bar	1	0.33
V → like	2	0.67
P → like	1	1

"candy" ist doppelt so wahrscheinlich wie "chocolate", obwohl beide in den Sätzen nur einmal auftauchten.

Training auf unannotierten Daten

children like a candy bar
children like chocolate



Regel	f	p
S → NP VP	2	1
NP → D N	0.5	0.11
NP → D N N	0.5	0.11
NP → N	3	0.67
NP → NP PP	0.5	0.11
VP → V	0.5	0.25
VP → V NP	1.5	0.75
PP → P NP	0.5	1
D → a	1	1
D → the	0	0
N → children	2	0.44
N → bar	0.5	0.11
N → candy	1	0.22
N → chocolate	1	0.22
V → bar	0.5	0.25
V → like	1.5	0.75
P → like	0.5	1

\+ Die Wahrscheinlichkeiten von "candy" und "chocolate" stimmen jetzt.
− Die gute und die schlechte Analyse des ersten Satzes haben dasselbe Gewicht.

Training auf unannotierten Daten

Ansatz 3: (Forts.)

EM-Training

1. Initialisierung der Regelwahrscheinlichkeiten
2. für alle Sätze (E-Schritt)
   - Berechnung der Parsebäume für den Satz
   - Berechnung der Parsebaumgewichte $p(t|s)$
   - Extraktion der gewichteten Regelhäufigkeiten
3. Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt)
4. Weiter mit Schritt 2 bis ein Endekriterium erfüllt ist

Problem: Wie können die gewichteten Regelhäufigkeiten effizient für hochambige Sätze berechnet werden?

Lösung: Inside-Outside-Algorithmus (wird später vorgestellt)

Training auf unannotierten Daten

Ansatz 3: (Forts.)

EM-Training

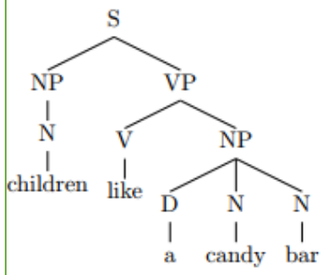
1. Initialisierung der Regelwahrscheinlichkeiten
2. für alle Sätze (E-Schritt)
   - Berechnung der Parsebäume für den Satz
   - Berechnung der Parsebaumgewichte $p(t|s)$
   - Extraktion der gewichteten Regelhäufigkeiten
3. Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt)
4. Weiter mit Schritt 2 bis ein Endekriterium erfüllt ist

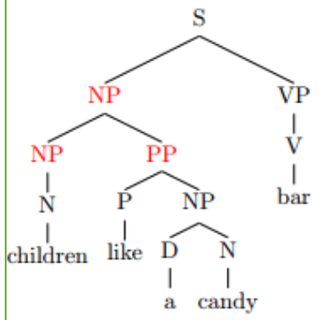
Problem: Wie können die gewichteten Regelhäufigkeiten effizient für hochambige Sätze berechnet werden?

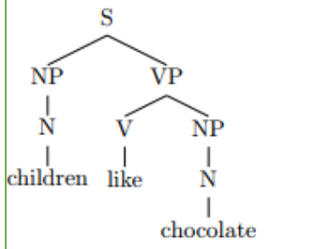
Lösung: Inside-Outside-Algorithmus (wird später vorgestellt)

Suppose our unannotated corpus contains 2 sentences
-children like a candy bar (has 2 analyses)
-children like chocolate (has 1 analysis)

we first extract the rules and initialize the probabilities of the rules with a uniform distribution







Regel	p_0
S → NP VP	1.00
NP → D N	0.25
NP → D N N	0.25
NP → N	0.25
NP → NP PP	0.25
VP → V	0.50
VP → V NP	0.50
PP → P NP	1.00
D → a	0.50
D → the	0.50
N → bar	0.25
N → candy	0.25
N → children	0.25
N → chocolate	0.25
V → bar	0.50
V → like	0.50
P → like	1.00
-logprob	

p = 1/ the number of rules that have the form *NP -> something*

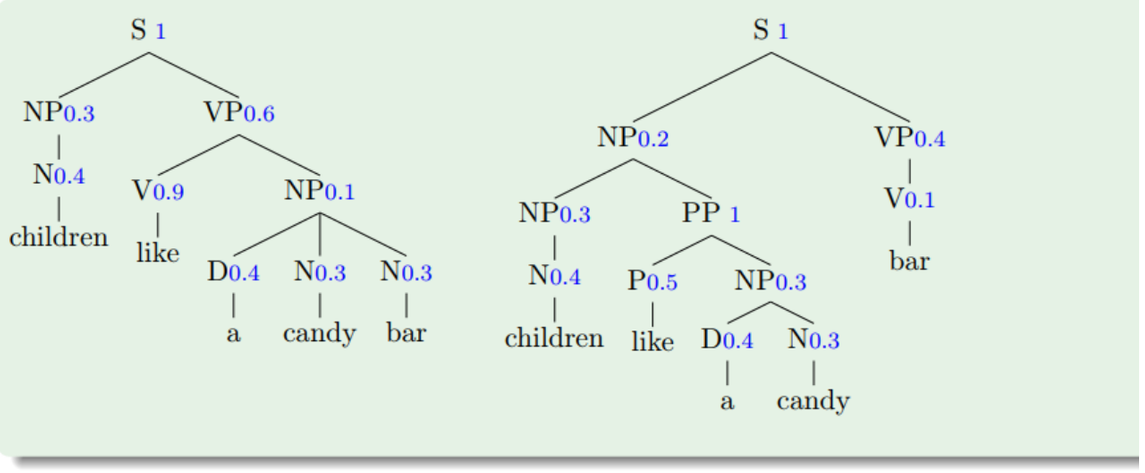
Training auf unannotierten Daten

Ansatz 3: (Forts.)

EM-Training

1. Initialisierung der Regelwahrscheinlichkeiten
2. für alle Sätze (E-Schritt)
   - Berechnung der Parsebäume für den Satz
   - Berechnung der Parsebaumgewichte $p(t|s)$
   - Extraktion der gewichteten Regelhäufigkeiten
3. Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt)
4. Weiter mit Schritt 2 bis ein Endekriterium erfüllt ist

= use p0 (that we calculated from the previous page) to assign the prob to every rule in the trees, then compute the weight for every tree.
note: the example shows how to compute the weight but it uses some random p here (not p0).



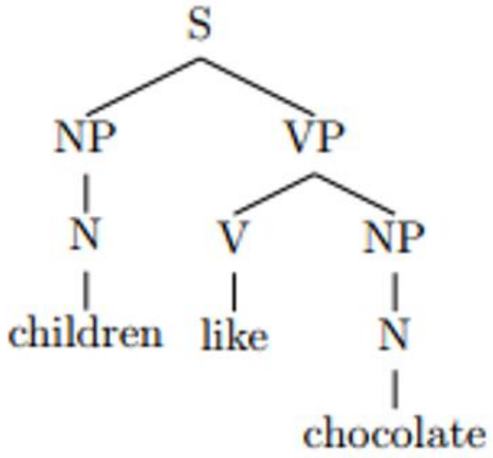
Zur Berechnung von $p(t_1)$ und $p(t_2)$ siehe Folie 182.

$$p(t_1) = 0.0002333 \qquad p(t_2) = 0.0000173$$
$$p(t_1|s) = \frac{p(t_1)}{p(t_1)+p(t_2)} = 0.93 \qquad p(t_2|s) = \frac{p(t_2)}{p(t_1)+p(t_2)} = 0.07$$

p(t1) is simply the sum of every probability in t1.

- Compute a weight for every tree.
- Here we only show an example of one sentence „children like a candy bar“ which has 2 analyses.
- We also have to do the same for the other sentence.
  - you would have p(t3|s2) = p(t3) / p(t3) = 1

compute the weight for sentence 2



$p(t3|s2) = p(t3) / p(t3)$

$= 1$

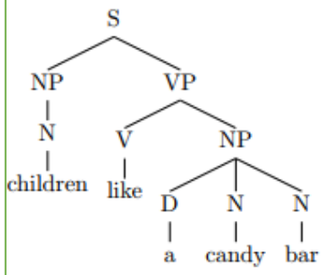
note: it has one analysis, that is why the prob is 1

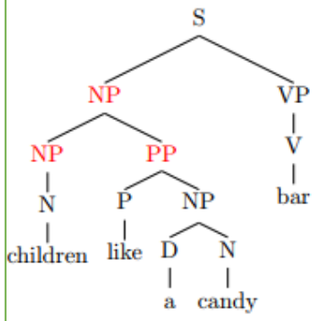
Training auf unannotierten Daten

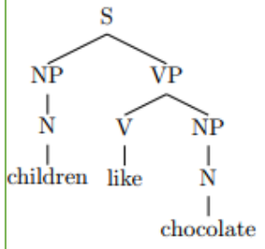
Ansatz 3: (Forts.)

EM-Training

1. Initialisierung der Regelwahrscheinlichkeiten
2. für alle Sätze (E-Schritt)
   - Berechnung der Parsebäume für den Satz
   - Berechnung der Parsebaumgewichte $p(t|s)$
   - Extraktion der gewichteten Regelhäufigkeiten
3. Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt)
4. Weiter mit Schritt 2 bis ein Endekriterium erfüllt ist







Regel	p_0	f_1
S → NP VP	1.00	2.00
NP → D N	0.25	0.50
NP → D N N	0.25	0.50
NP → N	0.25	3.00
NP → NP PP	0.25	**0.50**
VP → V	0.50	0.50
VP → V NP	0.50	1.50
PP → P NP	1.00	0.50
D → a	0.50	1.00
D → the	0.50	0.00
N → bar	0.25	0.50
N → candy	0.25	1.00
N → children	0.25	2.00
N → chocolate	0.25	1.00
V → bar	0.50	0.50
V → like	0.50	1.50
P → like	1.00	0.50
-logprob		15.2

- Die aus jedem Parsebaum t extrahierten Regelhäufigkeiten werden gewichtet mit

$$p(t|s) = \frac{p(t)}{\sum_{t' \in T(s)} p(t')}$$

example is on the next page

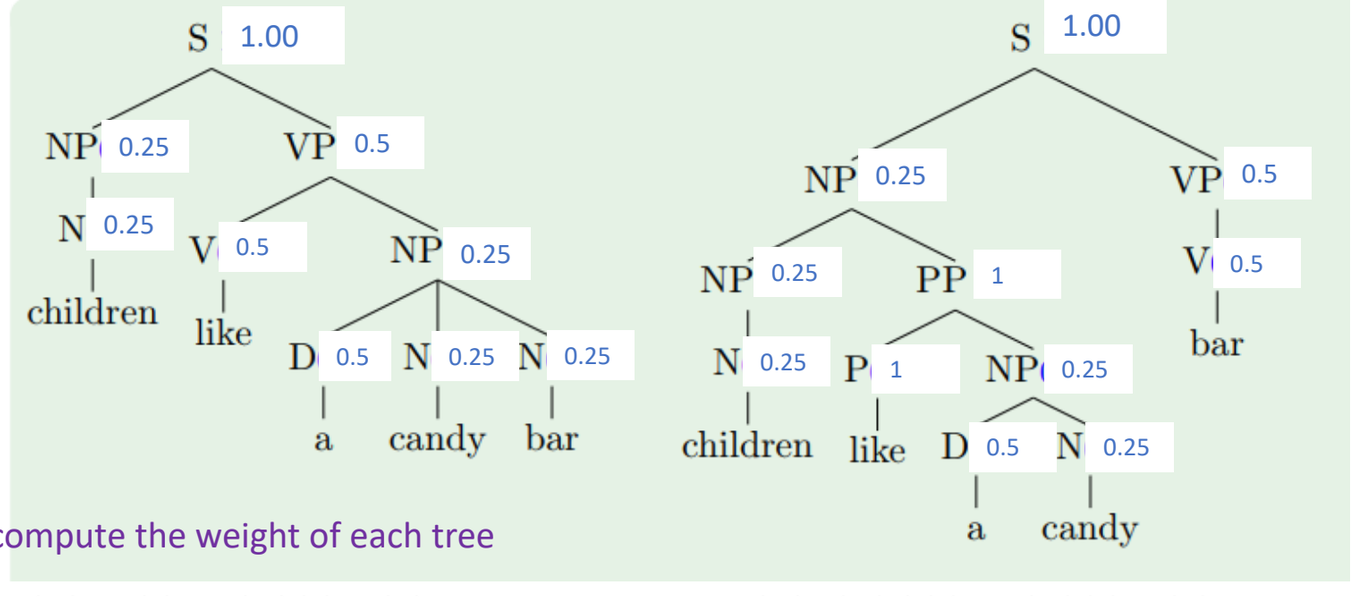
How to compute f1 (die gewichteten Häufigkeiten) 1/2

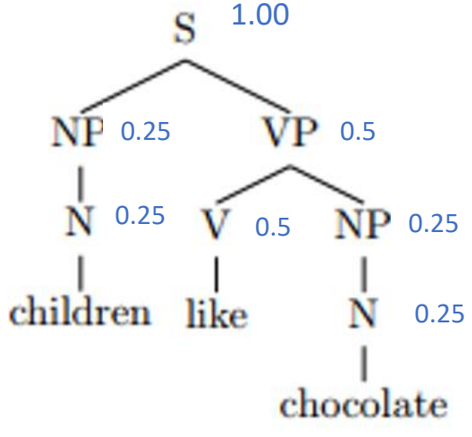
- Die aus jedem Parsebaum t extrahierten Regelhäufigkeiten werden gewichtet mit

$$p(t|s) = \frac{p(t)}{\sum_{t' \in T(s)} p(t')}$$

Regel	p_0	f_1
S → NP VP	1.00	
NP → D N	0.25	
NP → D N N	0.25	
NP → N	0.25	
NP → NP PP	0.25	
VP → V	0.50	
VP → V NP	0.50	
PP → P NP	1.00	
D → a	0.50	
D → the	0.50	
N → bar	0.25	
N → candy	0.25	
N → children	0.25	
N → chocolate	0.25	
V → bar	0.50	
V → like	0.50	
P → like	1.00	
-logprob		

1) assign p to each rule using p0





2) compute the weight of each tree

p(t1) = 1 * (0.25 *5) * (0.5 *3) = **1.875**

p(t2) = (1 *3) * (0.25 *5) * (0.5 *3) = **1.875**

$$p(t_1|s) = \frac{p(t_1)}{p(t_1)+p(t_2)} = 0.5$$

weight of t1

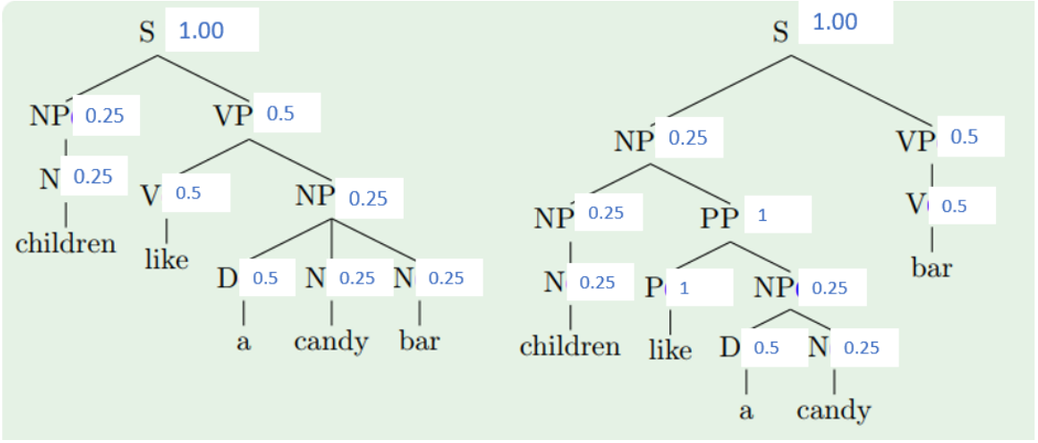
$$p(t_2|s) = \frac{p(t_2)}{p(t_1)+p(t_2)} = 0.5$$

weight of t2

p(t3| s2) = 1

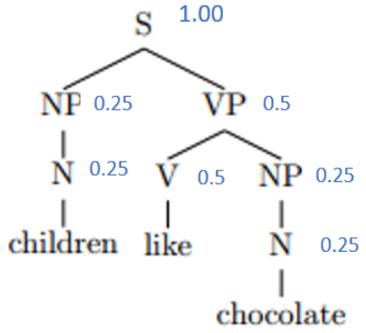
weight of t3

How to compute f1 (die gewichteten Häufigkeiten) 2/2



t1_weight = 0.5

t2_weight = 0.5



t3_weight = 1

Regel	p_0	f_1
S → NP VP	1.00	2.00
NP → D N	0.25	0.50
NP → D N N	0.25	0.50
NP → N	0.25	3.00
NP → NP PP	0.25	**0.50**
VP → V	0.50	0.50
VP → V NP	0.50	1.50
PP → P NP	1.00	0.50
D → a	0.50	1.00
D → the	0.50	0.00
N → bar	0.25	0.50
N → candy	0.25	1.00
N → children	0.25	2.00
N → chocolate	0.25	1.00
V → bar	0.50	0.50
V → like	0.50	1.50
P → like	1.00	0.50
-logprob		15.2

3) compute the expected frequency (f1) for every rule

For example, for S -> NP VP
-we have this rule in all trees, so we compute as follows.

p(S -> NP VP) = (1 * 0.5) + (1* 0.5) + (1*1) = 2

t2 t3

number of times we found S -> NP VP in t1

weight of t1

Example: compute f1 for NP -> NP PP
- We only found this rule in t2

p(NP -> NP PP) = 1* 0.5
= 0.5

Example: compute f1 for NP -> N
- we found 1 rule in t1, 1 rule in t2, and 2 rules in t3

p(NP -> NP PP) = (1* 0.5) + (1* 0.5) + (2*1)
= 3

Training auf unannotierten Daten

Ansatz 3: (Forts.)

EM-Training

1. Initialisierung der Regelwahrscheinlichkeiten
2. für alle Sätze (E-Schritt)
   - Berechnung der Parsebäume für den Satz
   - Berechnung der Parsebaumgewichte $p(t|s)$
   - Extraktion der gewichteten Regelhäufigkeiten
3. Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt)
4. Weiter mit Schritt 2 bis ein Endekriterium erfüllt ist

3) estimate p1 using f1

Schätzung der Regelwahrscheinlichkeiten mit relativen Häufigkeiten

$$p(A \to \alpha) = \frac{f_{A\to\alpha}}{\sum_\beta f_{A\to\beta}}$$

Regel	p_0	f_1	p_1
S → NP VP	1.00	2.00	1.00
NP → D N	0.25	0.50	0.11
NP → D N N	0.25	0.50	0.11
NP → N	0.25	3.00	0.67
NP → NP PP	0.25	**0.50**	0.11
VP → V	0.50	0.50	0.25
VP → V NP	0.50	1.50	0.75
PP → P NP	1.00	0.50	1.00
D → a	0.50	1.00	1.00
D → the	0.50	0.00	0.00
N → bar	0.25	0.50	0.11
N → candy	0.25	1.00	0.22
N → children	0.25	2.00	0.44
N → chocolate	0.25	1.00	0.22
V → bar	0.50	0.50	0.25
V → like	0.50	1.50	0.75
P → like	1.00	0.50	1.00
-logprob		15.2	

Example of how to estimate p1

Schätzung der Regelwahrscheinlichkeiten mit relativen Häufigkeiten

$$p(A \to \alpha) = \frac{f_{A\to\alpha}}{\sum_\beta f_{A\to\beta}}$$

Example: compute p1 for NP -> NP PP
-there are 4 rules with the form NP -> something with
- f1(NP -> D N) = 0.5
- f1(D -> N N) = 0.5
-f1(NP -> N) = 3
-f1(NP -> N PP) = 0.5

-so we compute 0.5 / 0.5 +0.5 +3 +0.5
= 0.5 / 4.5
= 0.11

Regel	p_0	f_1	p_1
S → NP VP	1.00	2.00	1.00
NP → D N	0.25	0.50	0.11
NP → D N N	0.25	0.50	0.11
NP → N	0.25	3.00	0.67
NP → NP PP	0.25	**0.50**	0.11
VP → V	0.50	0.50	0.25
VP → V NP	0.50	1.50	0.75
PP → P NP	1.00	0.50	1.00
D → a	0.50	1.00	1.00
D → the	0.50	0.00	0.00
N → bar	0.25	0.50	0.11
N → candy	0.25	1.00	0.22
N → children	0.25	2.00	0.44
N → chocolate	0.25	1.00	0.22
V → bar	0.50	0.50	0.25
V → like	0.50	1.50	0.75
P → like	1.00	0.50	1.00
-logprob		15.2	

Training auf unannotierten Daten

Ansatz 3: (Forts.)

EM-Training

1. Initialisierung der Regelwahrscheinlichkeiten
2. für alle Sätze (E-Schritt)
   - Berechnung der Parsebäume für den Satz
   - Berechnung der Parsebaumgewichte $p(t|s)$
   - Extraktion der gewichteten Regelhäufigkeiten
3. Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt)
4. Weiter mit Schritt 2 bis ein Endekriterium erfüllt ist

Regel	p_0	f_1	p_1	f_2	p_2	f_3	p_3
S → NP VP	1.00	2.00	1.00	2.00	1.00	2.00	1.00
NP → D N	0.25	0.50	0.11	0.10	0.02	0.00	0.00
NP → D N N	0.25	0.50	0.11	0.90	0.22	1.00	0.25
NP → N	0.25	3.00	0.67	3.00	0.73	3.00	0.75
NP → NP PP	0.25	**0.50**	0.11	**0.10**	0.02	**0.00**	0.00
VP → V	0.50	0.50	0.25	0.10	0.05	0.00	0.00
VP → V NP	0.50	1.50	0.75	1.90	0.95	2.00	1.00
PP → P NP	1.00	0.50	1.00	0.10	1.00	0.00	1.00
D → a	0.50	1.00	1.00	1.00	1.00	1.00	1.00
D → the	0.50	0.00	0.00	0.00	0.00	0.00	0.00
N → bar	0.25	0.50	0.11	0.90	0.18	1.00	0.20
N → candy	0.25	1.00	0.22	1.00	0.20	1.00	0.20
N → children	0.25	2.00	0.44	2.00	0.41	2.00	0.40
N → chocolate	0.25	1.00	0.22	1.00	0.20	1.00	0.20
V → bar	0.50	0.50	0.25	0.10	0.05	0.00	0.00
V → like	0.50	1.50	0.75	1.90	0.95	2.00	1.00
P → like	1.00	0.50	1.00	0.10	1.00	0.00	1.00
-logprob		15.2		11.3		9.3	

Problem: Wie können die gewichteten Regelhäufigkeiten effizient für hochambige Sätze berechnet werden?

Lösung: Inside-Outside-Algorithmus (wird später vorgestellt)

Neuschätzung der Regel-Wahrscheinlichkeiten

EM-Algorithmus: wiederholte Ausführung der beiden Schritte

1. E-Schritt
   - Jeder Satz des Trainingskorpus wird geparst und ein Parsewald erstellt.
   - Die erwartete Häufigkeit $\gamma(A \rightarrow \delta)$ jeder Parsewaldregel $A \rightarrow \delta$ wird berechnet.
   - Die erwarteten Häufigkeiten $\gamma(A \rightarrow \delta)$ werden für jede CFG-Regel über alle Trainingssätze zu $f(A' \rightarrow \delta')$ summiert, wobei sich A' und δ' aus A und δ durch Entfernen der Knotennummern ergeben.
2. M-Schritt
   Die Regel-Wahrscheinlichkeiten werden aus den erwarteten Häufigkeiten neu geschätzt:

$$p(A \rightarrow \delta) = \frac{f(A \rightarrow \delta)}{\sum_{\delta'} f(A \rightarrow \delta')}$$

Inside-Outside-Algorithmus

- Das Produkt der Inside- und Outside-Wahrscheinlichkeiten eines Parsewaldknotens ergibt die **Gesamtwahrscheinlichkeit aller Parsebäume mit diesem Knoten**.
- Durch Division dieses Produktes mit der Wahrscheinlichkeit aller Analysen erhält man die **erwartete Häufigkeit** des Parsewaldknotens.

  $\gamma(A) = \alpha(A)\beta(A)/\alpha(S)$
- erwartete Regelhäufigkeit

  $\gamma(A \rightarrow \delta) = \alpha(A \rightarrow \delta)\ \beta(A)/\alpha(S)$