

How to understand 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 $1 \leq k \leq n + 1$)

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

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

3. Ausgabe: (für $0 < k \leq n$)

$$t_{n+1} = \langle /s \rangle$$

$$t_k = \psi_{t_{k+1}}(k+1) \quad \text{für } 1 \leq k \leq n$$

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 $p(\text{tag}|\text{tag})$ 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 ψ too

Example: I want to understand this Trigram formula

- 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)

Viterbi-Algorithmus (Trigramm-Tagger)

- Beim Trigramm-Tagger entspricht jeder Zustand des Hidden-Markow-Modells 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.

	-1	0	1	2	3	k-1	k	
	-	-	I	can	w k	cān	-	-
	< /s >	< /s >	PRO t''	MD t'	MD t	< /s >	< /s >	

$$\text{vit_MD,MD}(3) = \max (\text{vit_PRO,MD}(2) p(\text{MD} | \text{PRO,MD}) p(\text{can} | \text{MD}), \text{vit_PP,MD}(2) p(\text{MD} | \text{PP,MD}) p(\text{can} | \text{MD}))$$

$\text{psi_MD,MD}(3)$ = 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)

				n	n+1
-1	0	1	2	3	4
-	-	I	can	can	-
<S>	<S>	PRO	MD	MD t	</S>
		PP	V	V	t

Viterbi-Algorithmus (Trigramm-Tagger)

- Beim Trigramm-Tagger entspricht jeder Zustand des Hidden-Markow-Modells 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 Endtags 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

$$\arg \max \text{ over } t (\text{MD and V}) \text{ of } (\text{vit_MD}, \langle s \rangle (4), \text{vit_V}, \langle s \rangle (4))$$

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.

```

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)

	0	1	2	k-1	k	
	-		can	w_k	cān	-

<s> PRO MD t MD t' </s>
PP V V t'

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

backward_MD(2) = $p(\text{MD}|\text{MD}) p(\text{can}|\text{MD}) \text{backward_MD}(3) + p(\text{V}|\text{MD}) p(\text{can}|\text{V}) \text{backward_V}(3)$

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

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

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:

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

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

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

(5 Punkte)

Aufgabe 2)

$$\delta_{\langle s \rangle}(0) = 1$$

$$\delta_{PRO}(1) = \delta_{\langle s \rangle}(0) p(PRO|\langle s \rangle) p(we|PRO) = 1 \cdot 0.2 \cdot 0.2 = 0.04$$

$$\psi_{PRO}(1) = \langle s \rangle$$

$$\delta_{MD}(2) = \delta_{PRO}(1) p(MD|PRO) p(can|MD) = 0.04 \cdot 0.3 \cdot 0.3 = 0.0036$$

$$\psi_{MD}(2) = PRO$$

$$\delta_N(2) = \delta_{PRO}(1) p(N|PRO) p(can|N) = 0.04 \cdot 0.1 \cdot 0.1 = 0.0004$$

$$\psi_N(2) = PRO$$

$$\delta_{\langle s \rangle}(3) = \max(\delta_{MD}(2) p(\langle s \rangle|MD) p(\epsilon|\langle s \rangle), \delta_N(2) p(\langle s \rangle|N) p(\epsilon|\langle s \rangle))$$

$$= \max(0.0036 \cdot 0.1 \cdot 1, 0.0004 \cdot 0.2 \cdot 1)$$

$$= \max(0.00036, 0.00008) = 0.00036$$

$$\psi_{\langle s \rangle}(3) = MD$$

$$t_2 = \psi_{\langle s \rangle}(3) = MD$$

$$t_1 = \psi_{MD}(2) = PRO$$

Ergebnis-Tagfolge: PRO MD

Aufgabe 5) Der Forward-Backward-Algorithmus kann zum Training von Hidden-Markow-Modellen auf unannotierten Daten benutzt werden. Welche Daten brauchen Sie, um einen Wortart-Tagger auf diese Weise zu trainieren? Wie lauten die Formeln zur Berechnung der Forward- und Backward-Wahrscheinlichkeiten? Wozu werden die berechneten Forward- und Backward-Wahrscheinlichkeiten anschließend verwendet? (4 Punkte)

EM-Training

mention that it is about EM-training

Lösung: Expectation-Maximization-Training

(maximiert iterativ die Wahrscheinlichkeit der Trainingsdaten)

gegeben

- ▶ ein nicht annotiertes Trainingskorpus
- ▶ ein Lexikon mit möglichen Wortarten von Wörtern

- ① Uniforme Initialisierung aller $p(t|t')$
- ② Uniforme Initialisierung aller $p(w|t)$, die im Lexikon auftauchen
- ③ Berechnung der erwarteten Wort-Tag- und Tag-Tag-Häufigkeiten mit dem Forward-Backward-Algorithmus (E-Schritt) **WOZU**
- ④ Neuschätzung der HMM-Wahrscheinlichkeiten aus den erwarteten Häufigkeiten (M-Schritt)
- ⑤ weiter mit Schritt 3 bis das Stoppkriterium erfüllt ist

Beispiele von Stoppkriterien:

- N Trainings-Iterationen wurden durchlaufen.
- Die Genauigkeit auf Development-Daten hat abgenommen.

Formel

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

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

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

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

Formeln für die Berechnung der erwartete Häufigkeiten (extra Antwort)

$$f_{tw} = \sum_{w \in C} \sum_{1 \leq k \leq |w|: w_k = w} \gamma_t(k, w)$$

$$f_{tt'} = \sum_{w \in C} \sum_{k=1}^{n+1} \gamma_{tt'}(k, w)$$

$$\gamma_t(k) = \frac{\alpha_t(k) \beta_t(k)}{\alpha_{\langle /s \rangle}(n+1)}$$

$$\gamma_{tt'}(k) = \frac{\alpha_t(k-1) p(t'|t) p(w_k|t') \beta_{t'}(k)}{\alpha_{\langle /s \rangle}(n+1)}$$

Aufgabe 4) Erklären Sie das Prinzip des EM-Algorithmus am Beispiel des unüberwachten Trainings eines HMM-Taggers. (3 Punkte)

similar answer as the
previous question

Aufgabe 4) Nach welchen Formeln berechnen Sie die Forward- und Backward-Wahrscheinlichkeiten mit einem Hidden-Markow-Modell? (4 Punkte)

Wie berechnen Sie aus den Forward- und Backward-Wahrscheinlichkeiten die Wahrscheinlichkeit des Tags t an Position i bzw. die Wahrscheinlichkeit der Tags t und t' an den Positionen i-1 und i? (In der Vorlesung wurde diese Werte mit γ bezeichnet.) (3 Punkte)

Forward

$$\begin{aligned}\alpha_t(0) &= \begin{cases} 1 & \text{falls } t = \langle s \rangle \\ 0 & \text{sonst} \end{cases} \\ \alpha_t(k) &= \sum_{t' \in T} \alpha_{t'}(k-1) p(t|t') p(w_k|t) \quad \text{für } 0 < k \leq n+1\end{aligned}$$

Backward

$$\begin{aligned}\beta_t(n+1) &= \begin{cases} 1 & \text{falls } t = \langle /s \rangle \\ 0 & \text{sonst} \end{cases} \\ \beta_t(k-1) &= \sum_{t' \in T} p(t'|t) p(w_k|t') \beta_{t'}(k) \quad \text{für } 0 < k \leq n+1\end{aligned}$$

gamma_t (k)

die Aposteriori-Wahrscheinlichkeit $p(t_k = t | w_1^n) = \gamma_t(k)$

$$\gamma_t(k) = \frac{\alpha_t(k) \beta_t(k)}{\alpha_{\langle /s \rangle}(n+1)}$$

gamma_tt' (k)

die Aposteriori-Wahrscheinlichkeit $p(t_{k-1} = t, t_k = t' | w_1^n) = \gamma_{tt'}(k)$

$$\gamma_{tt'}(k) = \frac{\alpha_t(k-1) p(t'|t) p(w_k|t') \beta_{t'}(k)}{\alpha_{\langle /s \rangle}(n+1)}$$

Aufgabe 8) Angenommen Sie trainieren einen HMM-Tagger mit dem Forward-Backward-Algorithmus. Wie können Sie die erwartete Häufigkeit (= Aposteriori-Wahrscheinlichkeit) des Tags t an der Position des Wortes w_k aus den Forward- und Backward-Wahrscheinlichkeiten berechnen?

Wie berechnen Sie die erwartete Häufigkeit des Tagpaars t und t' an den Positionen der Wörter w_k und w_{k+1} . 2

This is not quite correct because „erwartete HF“ is not the same thing as Aposteriori-WK according to the lecture slide.

This question is actually asking about the gamma formula (Aposteriori).

1

$$\gamma_t(k) = \frac{\alpha_t(k) \beta_t(k)}{\alpha_{\langle /s \rangle}(n+1)}$$

$$\gamma_{tt'}(k) = \frac{\alpha_t(k-1) p(t'|t) p(w_k|t') \beta_{t'}(k)}{\alpha_{\langle /s \rangle}(n+1)}$$

2

$$\gamma_{tt'}(k) = \frac{\alpha_t(k-1) p(t'|t) p(w_k|t') \beta_{t'}(k)}{\alpha_{\langle /s \rangle}(n+1)}$$

Aufgabe 10) Mit dem **Forward-/Backward-Algorithmus** können Sie die Wahrscheinlichkeit jedes möglichen Tags bei jedem Eingabewort berechnen. Sie wissen dann beispielsweise, dass das Tag *NN* beim 3. Wort eines gegebenen Satzes die Wahrscheinlichkeit 0,47 hat.

Wie könnten Sie auf Basis dieser Wahrscheinlichkeiten (sinnvoll) eine **beste Tagfolge** für den Satz definieren? (Die Lösung kennen Sie nicht aus der Vorlesung.)

Bekommen Sie mit dieser Methode dieselbe Tagfolge wie mit dem Viterbi-Algorithmus?
Versuchen Sie, Ihre Antwort zu begründen. (3 Punkte)

This question is actually too difficult.
Normally we don't have such a question.
But just so you know that there is a
possibility that such questions could be
in the exam.

Antwort:

Man kann einen Satz taggen, indem man an jeder Wortposition das Tag mit der höchsten Aposteriori-Wahrscheinlichkeit auswählt.

Das Ergebnis ist nicht dasselbe wie beim Viterbi-Algorithmus, weil die Rechnungen nicht äquivalent sind.
Angenommen ein HMM hat die Wahrscheinlichkeiten:

	A	B	$\langle s \rangle$	a	ϵ
$p(. A)$	0.9	0	0.1	1	0
$p(. B)$	0	0.9	0.1	1	0
$p(. \langle s \rangle)$	0.5	0.5	0	0	1

Dieses HMM generiert eine Folge von a's (plus ein Endesymbol), die entweder alle mit A oder alle mit B getaggt sind. Der Viterbi-Algorithmus kann nur eine dieser beiden Tagfolgen ausgeben. Der Forward-Backward-Tagger könnte dagegen eine beliebige Tagfolge ausgeben, da die Aposteriori-Wahrscheinlichkeit des Tags A (analog B) an jeder Position 0.5 beträgt und damit maximal ist.

Schriftliche Wiederholungsprüfung zur Übung

Statistische Methoden in der maschinellen Sprachverarbeitung

SS 2020

Dozent: Helmut Schmid

Sie haben 60 Minuten Zeit plus 5 Minuten zum Absenden.

Allgemeiner Hinweis: Wenn Sie einen Fehler oder ein anderes Problem in einer der Aufgaben entdecken sollten, dann schicken Sie mir bitte eine Nachricht an schmid@cis.1mu.de.

Sie sollen den **Forward-Backward-Algorithmus** für einen **Trigramm-Tagger** implementieren. Die Verarbeitung erfolgt wie immer satzweise.

Aufgabe 1) Forward-Algorithmus

Definition der Forward-Wahrscheinlichkeiten $\alpha_{t',t}(k)$

$$\begin{aligned}\alpha_{t',t}(0) &= \begin{cases} 1 & \text{falls } t = t' = \langle s \rangle \\ 0 & \text{sonst} \end{cases} \\ \alpha_{t',t}(k) &= \sum_{t'' \in T} \alpha_{t'',t'}(k-1) p(t|t'', t') p(w_k|t) \quad \text{für } 1 \leq k \leq n+2\end{aligned}$$

t , t' und t'' sind hier Tags, k ist eine Wortposition, $\langle s \rangle$ ist das Satzgrenzen-Tag. w_1, \dots, w_n sind die Wörter des Satzes. w_{-1} , w_0 , w_{n+1} und w_{n+2} sind Satzgrenzentokens (bspw. der leere String). Die Wahrscheinlichkeit des Satzgrenzentokens gegeben das Satzgrenzentag ist 1 und sonst 0.

Achtung: Sie iterieren hier bis $n+2$ statt nur bis $n+1$. Auch die forward-Tabelle ist entsprechend größer.

Sie können den Forward-Algorithmus ähnlich wie den Viterbi-Algorithmus implementieren. Die Maximierung des Viterbi-Algorithmus muss aber durch eine Summe ersetzt werden, und es werden keine Rückwärtszeiger (best_prev) benötigt.

Schreiben Sie Code zur Berechnung der Forward-Wahrscheinlichkeiten. Folgende Variablen und Funktionen sind gegeben und müssen nicht implementiert werden:

- $words$: eine Liste mit den Wörtern des aktuellen Satzes
- $tagset$: eine Liste mit allen vorhandenen Tags
- $lexprob(w,t)$: eine Funktion, welche $p(w|t)$ zurückliefert.
- $contextprob(t'',t',t)$: eine Funktion, die $p(t|t'', t')$ zurückliefert

(10 Punkte)

Musterlösung zur Prüfungsaufgabe:

```
def forward(words):
    words = [''] + words + ['', ''] # Grenztokens hinzufügen

    # Initialisierung der Forward-Tabelle
    fwd_prob = [defaultdict(float) for _ in words]
    fwd_prob[0][('<s>', '<s>')] = 1.0
    for i in range(1, len(words)):
        tags = tagset if i < len(words)-2 else ['<s>']
        for tag3 in tags:
            for (tag1, tag2), prevp in fwd_prob[i-1].items():
                p = prevp * contextprob(tag1, tag2, tag3) * lexprob(word[i], tag3)
                fwd_prob[i][(tag2, tag3)] += p
    return fwd_prob
```

You can download this solution from the lecture website
(Altklausuren + Lösung)

Aufgabe 2) Backward-Algorithmus

Definition der Backward-Wahrscheinlichkeiten $\beta_{t'',t'}(k)$:

$$\begin{aligned}\beta_{t'',t'}(n+2) &= \begin{cases} 1 & \text{falls } t'' = t' = \langle s \rangle \\ 0 & \text{sonst} \end{cases} \\ \beta_{t'',t'}(k) &= \sum_{t \in T} p(t|t'', t') p(w_{k+1}|t) \beta_{t',t}(k+1) \quad \text{für } 0 \leq k \leq n+1\end{aligned}$$

Schreiben Sie Code zur Berechnung der Backward-Wahrscheinlichkeiten. Die Berechnung der Backward-Wahrscheinlichkeiten geht ähnlich wie die Berechnung der Forward-Wahrscheinlichkeiten, aber Sie iterieren in umgekehrter Reihenfolge $n+1, \dots, 0$ und initialisieren die letzte Spalte der Tabelle. (10 Punkte)

```
def backward(words):
    words = [''] + words + ['', ''] # Grenztokens hinzufügen

    # Initialisierung der Forward-Tabelle
    bwd_prob = [defaultdict(float) for _ in words]
    bwd_prob[-1][('<s>', '<s>')] = 1.0
    for i in range(len(words)-1, 0, -1):
        tags = tagset if i > 2 else ['<s>']
        for tag1 in tags:
            for (tag2, tag3), nextp in bwd_prob[i].items():
                p = nextp * contextprob(tag1, tag2, tag3) * lexprob(word[i], tag3)
                bwd_prob[i-1][(tag1, tag2)] += p
    return bwd_prob
```

Aufgabe 3) Berechnung der geschätzten Häufigkeiten

Nun berechnen Sie noch die geschätzten Häufigkeiten von Tag-Trigrammen:

$$\gamma_{t'', t', t}(k) = \frac{\alpha_{t'', t'}(k-1) p(t|t'', t') p(w_k|t) \beta_{t', t}(k)}{\alpha_{\langle s \rangle, \langle s \rangle}(n+2)} \quad \text{für } 1 \leq k \leq n+2$$

Sie iterieren erneut über alle Elemente der Forward-Matrix und berechnen die γ -Werte für die Tag-Tripel. Summieren Sie diese Werte für die einzelnen Tag-Trigramme in einem (bereits vorhandenen) Dictionary *freq*, welches Tag-Tripel auf ihre erwartete Häufigkeit abbildet.

(10 Punkte)

(30 Punkte insgesamt)

```
def estimated_freq(words, fwd_prob, bwd_prob, freq):
    words = [''] + words + ['', ''] # Grenztokens hinzufügen
    total_prob = fwd_prob[-1][('<s>', '<s>')]
    for i in range(1, len(words)):
        tags = tagset if i < len(words)-2 else ['<s>']
        for tag3 in tags:
            for (tag1, tag2), prevp in fwd_prob[i-1].items():
                p = prevp * contextprob(tag1, tag2, tag3) * lexprob(word[i], tag3)
                freq[(tag1, tag2, tag3)] += p * bwd_prob[i][(tag2, tag3)] / total_prob
```

Aufgabe 4) Erklären Sie ausführlich, wie der Viterbi-Algorithmus beim Wortart-Taggen mit Hidden-Markow-Modellen die wahrscheinlichste Tagfolge berechnet. Geben Sie die Formeln zur Berechnung der Viterbiwahrscheinlichkeiten an. (4 Punkte)

Aufgabe 3) Erläutern Sie, wie der Viterbi-Algorithmus bei Hidden-Markow-Modellen 2. Ordnung (=Trigramm-Modellen) arbeitet und geben Sie an, wie die Viterbi-Wahrscheinlichkeiten definiert sind. (3 Punkte)

Aufgabe 4) Begründen Sie, warum der Viterbi-Algorithmus beim Wortart-Taggen mehr Zeit braucht, wenn die Zahl der möglichen Tags eines Wortes oder die Ordnung des HMMs (= Zahl der relevanten vorhergehenden Tags) erhöht wird.

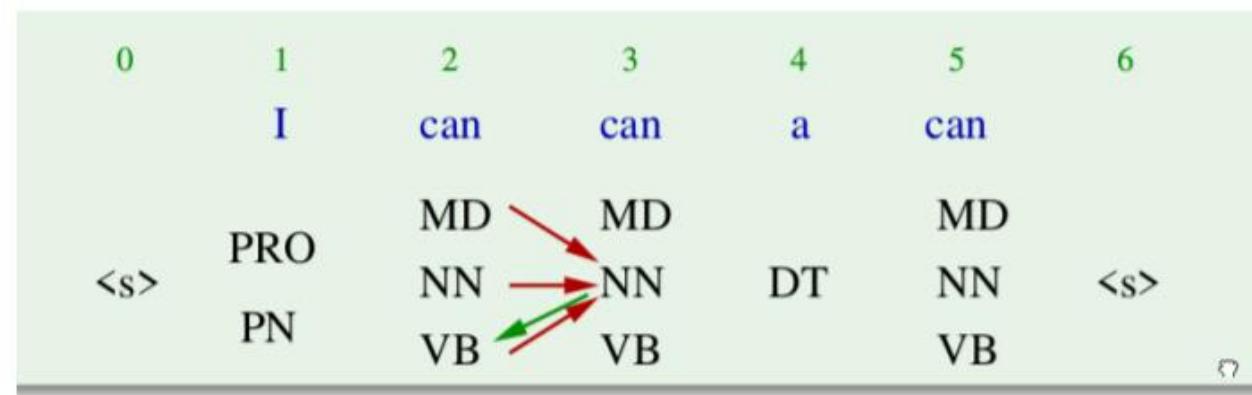
Steigt die benötigte Rechenzeit schneller bei Erhöhung der Zahl der Tags oder bei Erhöhung der Ordnung des HMMs? (3 Punkte)

This question is quite difficult. Don't worry if you can not solve it. Normally, we don't have such difficult questions in the exam.

Aufgabe 4) Begründen Sie, warum der Viterbi-Algorithmus beim Wortart-Taggen mehr Zeit braucht, wenn die Zahl der möglichen Tags eines Wortes oder die Ordnung des HMMs (= Zahl der relevanten vorhergehenden Tags) erhöht wird.

Steigt die benötigte Rechenzeit schneller bei Erhöhung der Zahl der Tags oder bei Erhöhung der Ordnung des HMMs?

□ (3 Punkte)



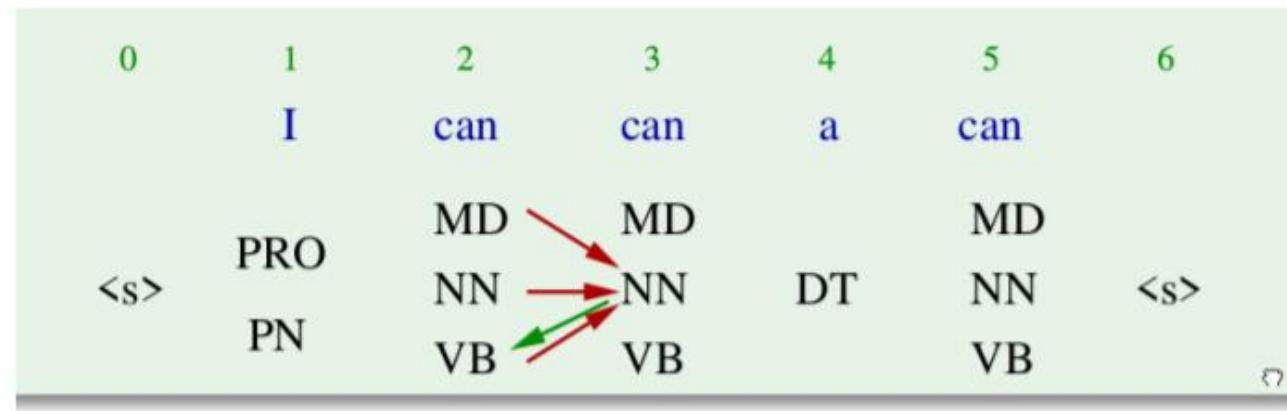
Wenn #Tag erhöht wird, erhöht sich die Anzahl von $\delta_t(k)$, die berechnet werden müssen. In jede Position k, berechnet der Algorithmus für jeden Tag t eine viterbi prob δ .

Z.B. Wenn "can" noch einen Tag hat, dann werden für position 2,3,5 jeweils vier viterbi-prob (statt 3 viterbi-prob) berechnet.

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

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

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



Wenn die Ordnung erhöht wird, dann erhöht sich auch die Anzahl von $\delta_t(k)$, die berechnet werden müssen.

In jeder Position k, berechnet der Algorithmus für jedes Tagpaar t',t eine viterbi prob δ .

Z.B. bigram vs trigram

Bei 2-gram

Für Position 3, berechnet der Algorithmus
 $\text{vit_MD}(3)$
 $\text{vit_NN}(3)$
 $\text{vit_VB}(3)$

Bei 3-gram

Für Position 3, der Algorithmus
 $\text{vit_MD,MD}(3), \text{vit_NN,MD}(3), \text{vit_VB,MD}(3)$
 $\text{vit_MD,NN}(3), \text{vit_NN,NN}(3), \text{vit_VB,NN}(3)$
 $\text{vit_MD,VB}(3), \text{vit_NN,VB}(3), \text{vit_VB,VB}(3)$

Aufgabe 4) Begründen Sie, warum der Viterbi-Algorithmus beim Wortart-Taggen mehr Zeit braucht, wenn die Zahl der möglichen Tags eines Wortes oder die Ordnung des HMMs (= Zahl der relevanten vorhergehenden Tags) erhöht wird.

Steigt die benötigte Rechenzeit schneller bei Erhöhung der Zahl der Tags oder bei Erhöhung der Ordnung des HMMs?

o

(3 Punkte)

Antwort: Bei Erhöhung der Ordnung des HMMs

Wie prüfen wir das?

Wir vergleichen die Rechenzeit von beiden Fällen.

	0	1	2	3	4	5	6
	<s>	I	can	can	a	can	
PRO			MD	MD		MD	
PN			NN	NN		NN	
			VB	VB		VB	

This graphic shows that the algorithm already knows the possible tags for each word.

However, in practice, all words have the same set of possible tags. In the implementation we use a function that filters out the unlikely tags before running the viterbi algorithm. Thus we have to count this step in the time complexity as well.



	0	1	2	3	4	5	6
	<s>	I	can	can	a	can	
	MD	MD	MD	MD	MD		
	NN	NN	NN	NN	NN		
	VB	VB	VB	VB	VB		

With this setting, we want to compare the runtime of bigram HMM after increasing a tag versus after increasing the HMM order.

To get a better understanding of the question, we consider a bigram model, what is the runtime complexity of it?

Next we ask, what is the runtime after increasing 1 tag (or more) ?

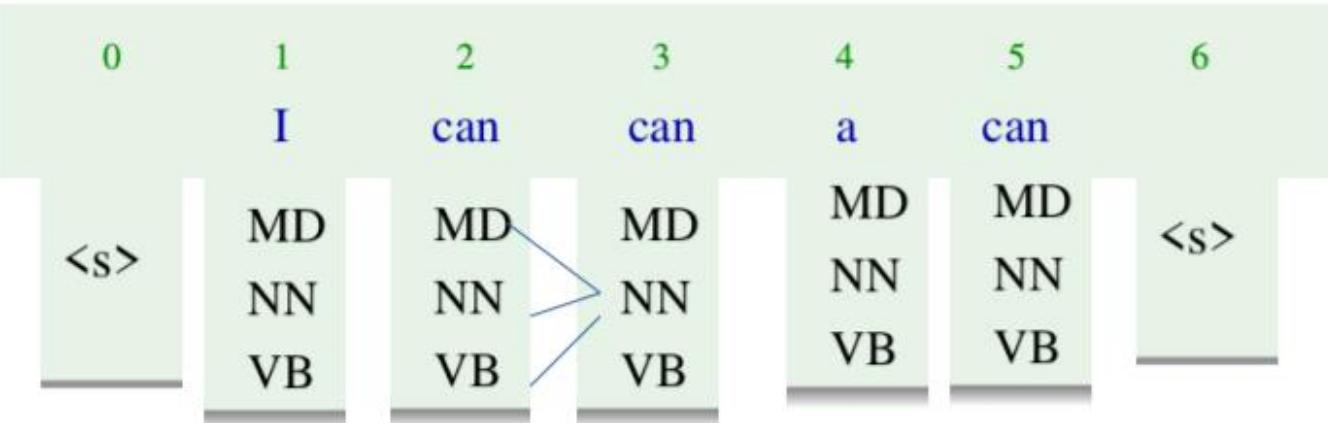
Next, what is the runtime after increasing the order of the bigram HMM ? (= trigram model)

This is just to show that normally all tags are taken into consideration.

v	w ₁ =the	w ₂ =doctor	w ₃ =is	w ₄ =in	</s>
Noun	0	.0756	.001512	0	
Verb	0	.00021	.027216	0	
Det	.21	0	0	0	.000027 2
Prep	0	0	0	.005443	
Adv	0	0	0	.000272	

Det Noun Verb Prep

ref: http://people.cs.georgetown.edu/nschneid/cosc572/s18/12_viterbi_slides.pdf



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

$$\delta_{NN}(3) = \max \left(\begin{array}{l} \delta_{MD}(2) p(NN|MD) p(can|NN), \\ \delta_{NN}(2) p(NN|NN) p(can|NN), \\ \delta_{VB}(2) p(NN|VB) p(can|NN) \end{array} \right)$$

What is the runtime complexity of a bigram model with 3 tags?

To calculate the runtime we count the number of operations done in the algorithm.

Let's look at how many calculations the algorithm has to do to compute one viterbi prob.

Um vit_NN(3) zuberechnen, braucht der Algorithmus 3 Berechnungen/Operationen (Wenn wir die Max-Operation nicht mitberechnen).

An Position 3 werden Vit-probs für all tags berechnen.

D.h. Die Rechenzeit für Position 3 ist Anzahl der Tags(Position 3) * Anzahl der Tags (Position 2).

Da alle Wörter den gleichen Tagset haben ist die Rechenzeit für Position 3 dann T^2 (wobei T für die Anzahl der Tags steht).

Also, die Rechenzeit für jede Position T^2. Für die gesamte Wortfolge ist es dann T^2 * N (wenn N die Anzahl der Wörter ist).

Nächste Frage: wie ändert sich die Komplexität (Rechenzeit), wenn die Anzahl der Tags erhöht wird?

Wie ändert sich die Komplexität (Rechenzeit), wenn die Anzahl der Tags erhöht wird?

	0	1	2	3	4	5	6
<S>	I	can	can	a	can		
MD	MD	MD	MD	MD	MD		<S>
NN	NN	NN	NN	NN	NN		
VB	VB	VB	VB	VB	VB		

Bigram-HMM mit Anzahl der Tag = 3

Die Rechenzeit = $O(T^2 * N)$

	0	1	2	3	4	5	6
<S>	I	can	can	a	can		
MD	MD	MD	MD	MD	MD		<S>
NN	NN	NN	NN	NN	NN		
VB	VB	VB	VB	VB	VB		
DT	DT	DT	DT	DT	DT		

Jetzt erhöhen wir die Anzahl der Tags um 1.
Wir haben dann Bigram-HMM mit Anzahl der Tag = 4.

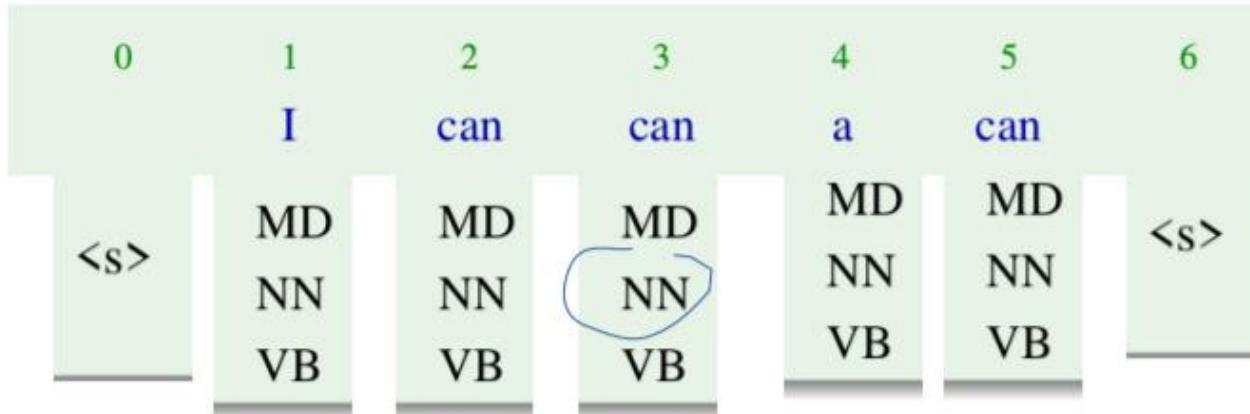
Für eine Position, braucht der Algorithmus $4 * 4$ Operationen(Berechnungen) um alle Vit-Prob für alle Tags (an der Position) zu berechnen.

Die Rechenzeit = $O((T+1)^2 * N)$
Wenn wir die Anzahl der Tags um m erhöht,
schreiben wir dann $O((T+m)^2 * N)$

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

Wir wollen als nächstes diese Rechenzeit mit der von einem 3-gram-HMM (mit 3 tags) vergleichen.

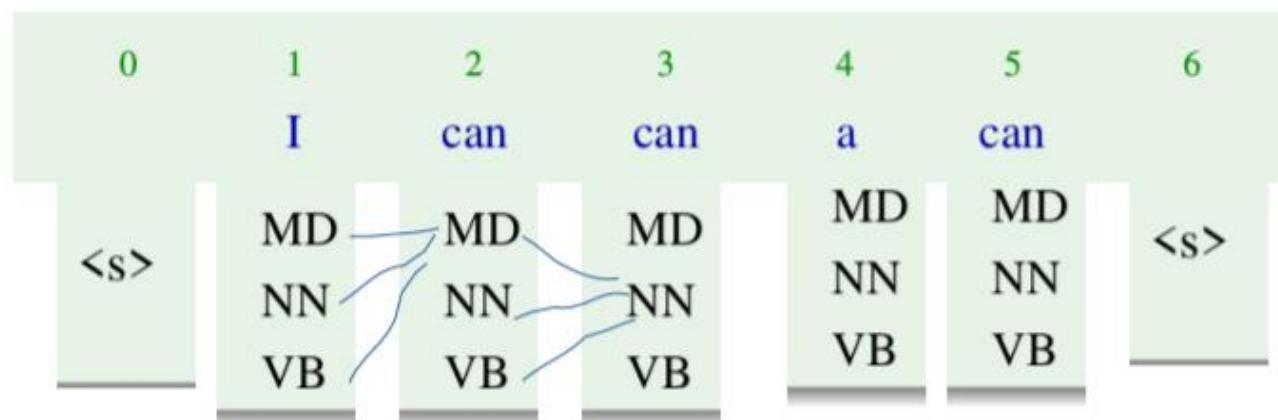
bigram HMM mit 3 tags



$$\delta_{NN}(3) = \max \left(\begin{array}{l} \delta_{MD}(2) p(NN|MD) p(can|NN), \\ \delta_{NN}(2) p(NN|NN) p(can|NN), \\ \delta_{VB}(2) p(NN|VB) p(can|NN) \end{array} \right)$$

Komplexität = $O(T^2 * N)$

trigram HMM mit 3 tags



Komplexität von Trigram = $O(T^3 * N)$

In Trigramm HMM berechnet der Algorithmus

$$\delta_{MD,MD}(3), \delta_{NN,MD}(3), \delta_{VB,MD}(3)$$

$$\delta_{MD,NN}(3), \delta_{NN,NN}(3), \delta_{VB,NN}(3)$$

$$\delta_{MD,VB}(3), \delta_{NN,VB}(3), \delta_{VB,VB}(3)$$

Das ist $3 * 3$ viterbi prob. (3 ist Anzahl der Tags)

Um eine Viterbiprob zu berechnen, braucht der Algo noch 3 Berechnungen. Z.B. für $\delta_{MD,NN}(3)$

$$\delta_{MD,NN}(3) = \max [$$

$$\delta_{MD,MD}(2)p(NN|MD,MD)p(can|NN),$$

$$\delta_{NN,MD}(2)p(NN|NN,MD)p(can|NN),$$

$$\delta_{VB,MD}(2)p(NN|VB,MD)p(can|NN)]$$

Für Position 3 werden also $3 * 3 * 3$ Berechnungen gemacht.

$O((T+m)^2 * N)$ vs $O(T^3 * N)$

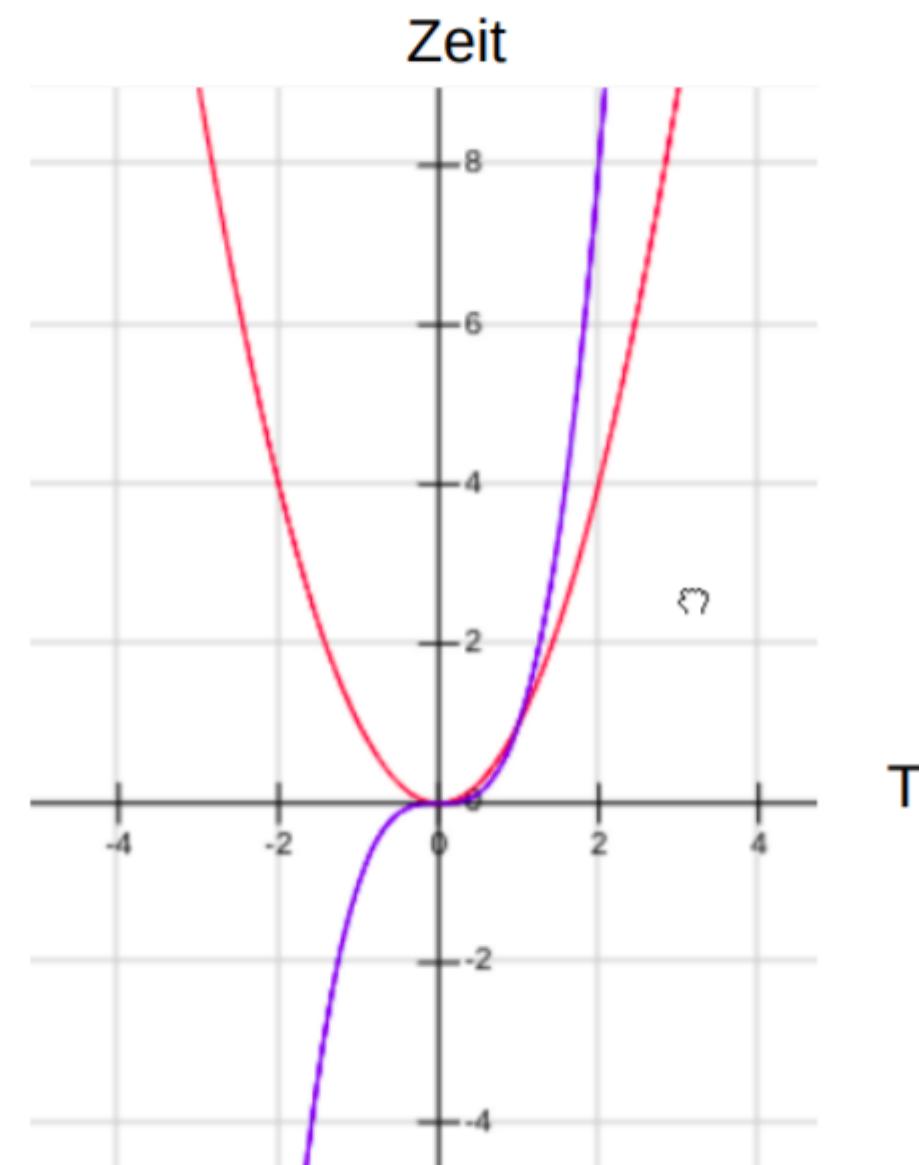
Welche Rechenzeit ist höher?

$O((T+m)^2 * N)$ vs $O(T^3 * N)$

N sind gleich bei beiden, kann man es weglassen.
In der Komplexität-Analyse ist m ein Konstant, kann man es auch weglassen.

$O(T^2)$ vs $O(T^3)$

Ab $T = 1$ erhöht sich $O(T^3)$ schneller als $O(T^2)$



Aufgabe 4) Warum benötigt man bei der Anwendung von statistischen Verfahren (und anderen Lernverfahren) in der Regel drei Teilmengen von Daten? Wie werden Sie üblicherweise benannt und wozu dient jede der drei Teilmengen? (2 Punkte)

Optimierung von Metaparametern

Der beschriebene HMM-Tagger hat eine Reihe von **Metaparametern** (Ordnung des Modells n , maximale Suffixlänge l), die beliebig gewählt werden können.

Um gute Werte dafür auszuwählen, kann der Tagger für verschiedene mögliche Werte der Metaparameter trainiert und dann jeweils auf Testdaten evaluiert werden. Diese Testdaten dürfen aber nicht mit den Testdaten für die eigentliche Evaluierung identisch sein.

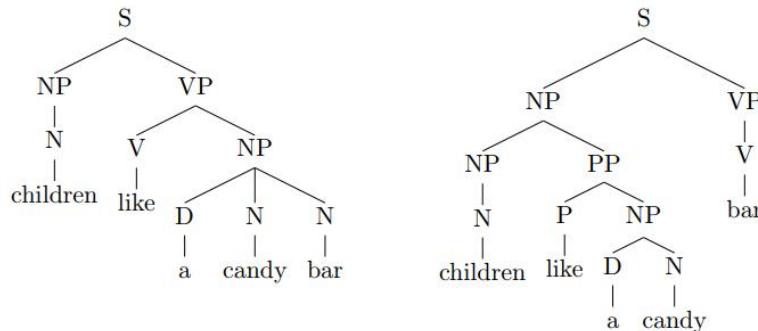
Solche Daten für die Parameter-Optimierung nennt man **Entwicklungsdaten** (Developmentdaten).

In der Regel braucht man für Experimente in der Computerlinguistik **Trainingsdaten**, **Developmentdaten** und **Testdaten**.

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

$S \rightarrow NP\ VP$
 $VP \rightarrow V\ NP$
 $VP \rightarrow V$
 $NP \rightarrow D\ N1$
 $N1 \rightarrow A\ N1$
 $N1 \rightarrow N$
 $D \rightarrow the$
 $N \rightarrow bell$
 $V \rightarrow rings \dots$

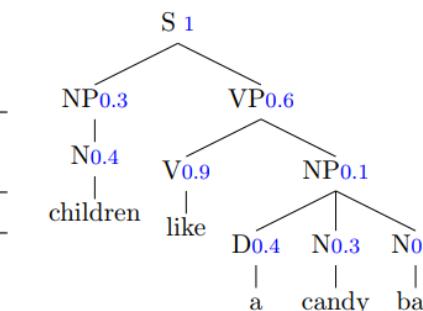
When we parse a sentence (to get a syntactic analysis) using context-free grammar, there can be more than one analysis.

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

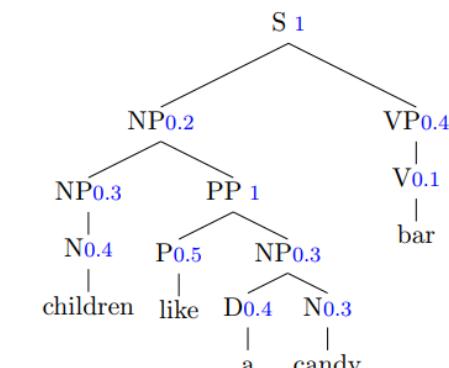
PCGF

Beispiele für Parsebaum-Wahrscheinlichkeiten

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



$$p_1 = 0.0002333$$



$$p_2 = 0.0000173$$

Probabilistische kontextfreie Grammatiken

Wahrscheinlichkeit eines Parsebaumes T mit Linksableitung r_1, \dots, r_n

$$\begin{aligned} p(T) = p(r_1, \dots, r_n) &= \prod_{i=1}^n p(r_i | r_1, \dots, r_{i-1}) \\ &= \prod_{i=1}^n p(r_i | \text{nextcat}(r_1, \dots, r_{i-1})) \end{aligned}$$

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

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

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

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

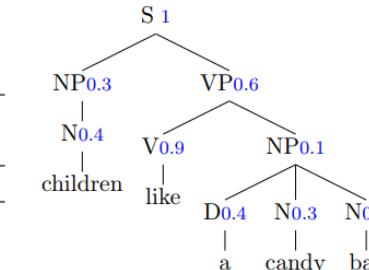
$$p(T) = p(r_1, \dots, r_n) = \prod_{i=1}^n p(r_i)$$

Diese Formel gilt aber nur, wenn r_1, \dots, r_n tatsächlich eine Linksableitung ist!

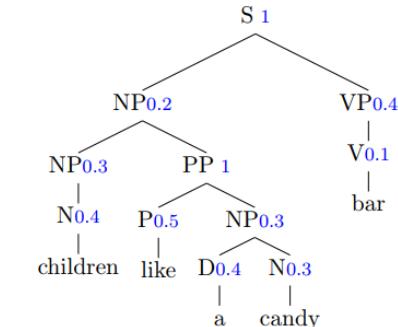
- We can represent a parse tree with a set of ordered grammar rules (Linksableitung)
- with these rules, we will keep expanding the leftmost non-terminal symbol.
- In the PCFG approach, each rule is assigned a probability.
- When we multiply the probabilities of the rules together, we get $p(T)$, the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

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



$$p_1 = 0.0002333$$



$$p_2 = 0.0000173$$

Grammatik	Satzform	Aktion
$S \rightarrow NP VP$	S	$S \rightarrow NP VP$
$VP \rightarrow V NP$	$NP VP$	$NP \rightarrow D N 1$
$VP \rightarrow V$	$D N 1 VP$	$D \rightarrow the$
$NP \rightarrow D N 1$	$the N 1 VP$	$N 1 \rightarrow N$
$N 1 \rightarrow A N 1$	$the N VP$	$N \rightarrow bell$
$N 1 \rightarrow N$	$the bell VP$	$VP \rightarrow V$
$D \rightarrow the$	$the bell V$	$V \rightarrow rings$
$N \rightarrow bell$	$the bell rings$	
$V \rightarrow rings$...		

Aufgabe 8) Wie wird bei einer probabilistischen kontextfreien Grammatik (PCFG) die Wahrscheinlichkeit eines Parsebaumes/Satzes/Korpus definiert?

(Ein Parsebaum wird repräsentiert durch die Folge der Linksableitungs-Regeln r_1, \dots, r_n mit den Regelwahrscheinlichkeiten $p(r_i)$.)

(2 Punkte)

Korpuswahrscheinlichkeit $p(C) = \prod_{s \in C} p(s)$

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

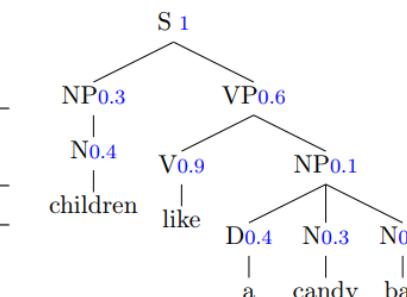
Parsewahrscheinlichkeit $p(t) = p(r_1, \dots, r_n) = \prod_{i=1}^n p(r_i)$

$T(s)$ sind die Analysen des Satzes s .

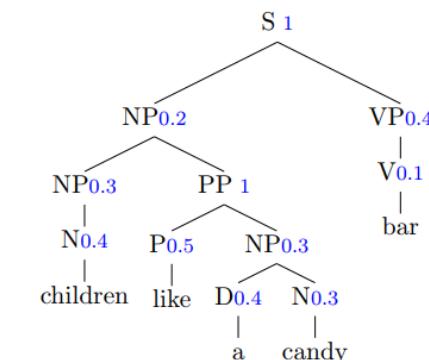
r_1, \dots, r_n sind die Regeln der Linksableitung von t .

Beispiele für Parsebaum-Wahrscheinlichkeiten

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



$$p_1 = 0.0002333$$



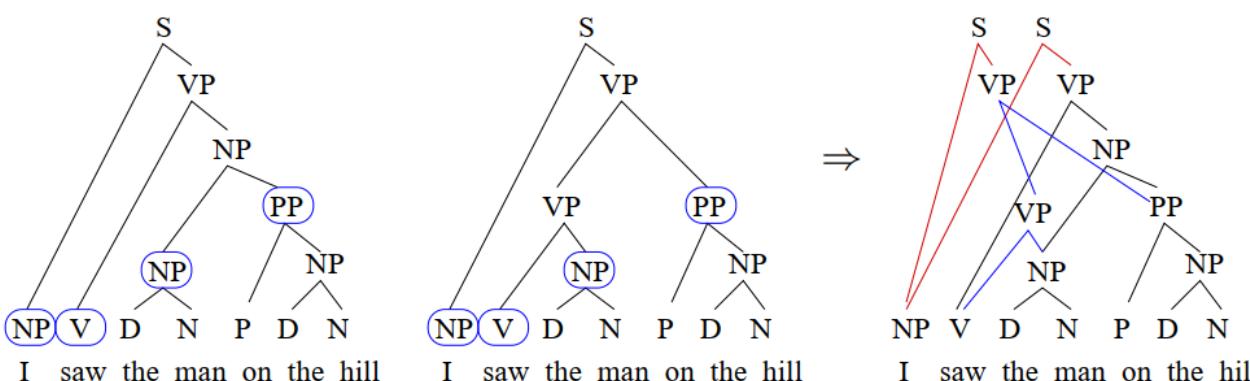
$$p_2 = 0.0000173$$

Aufgabe 6) Wie ist die Wahrscheinlichkeit eines Parsebaumes bei einer probabilistischen kontextfreien Grammatik (PCFG) definiert? (2 Punkte)

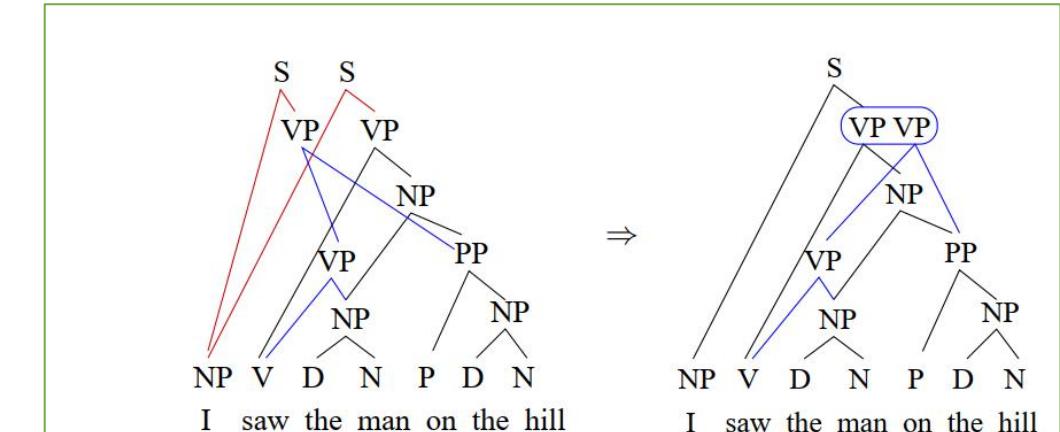
Aufgabe 8) Wie wird bei PCFGs die Wahrscheinlichkeit eines Parsebaumes, die Wahrscheinlichkeit eines Satzes und die Wahrscheinlichkeit einer Folge von Sätzen (=Korpus) definiert? (3 Punkte)

- 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.

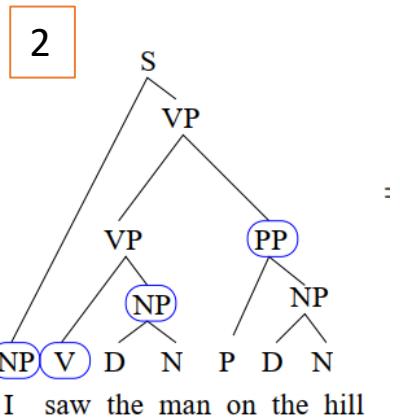
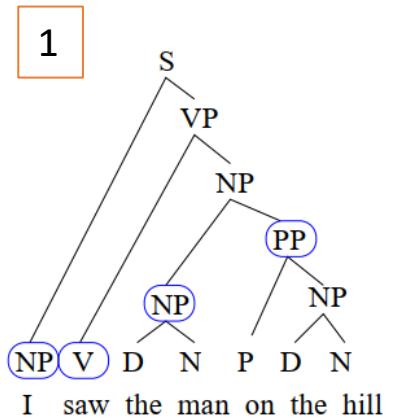
Zusammenfassen (maximaler) gemeinsamer Teilbäume



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



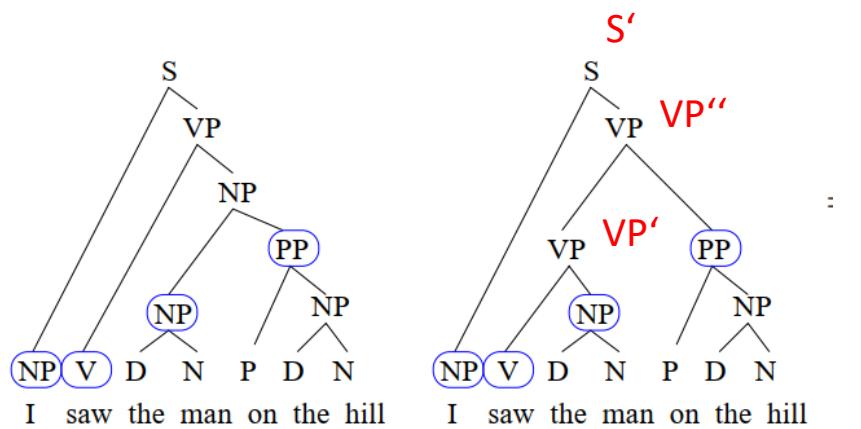
Hier werden die beiden roten Teilebäume zusammengefasst.
 Die beiden VP-Teilebäume werden in einem ambigen VP-Knoten vereinigt.
 Die blauen Kanten gehören zum 2. Parsebaum.



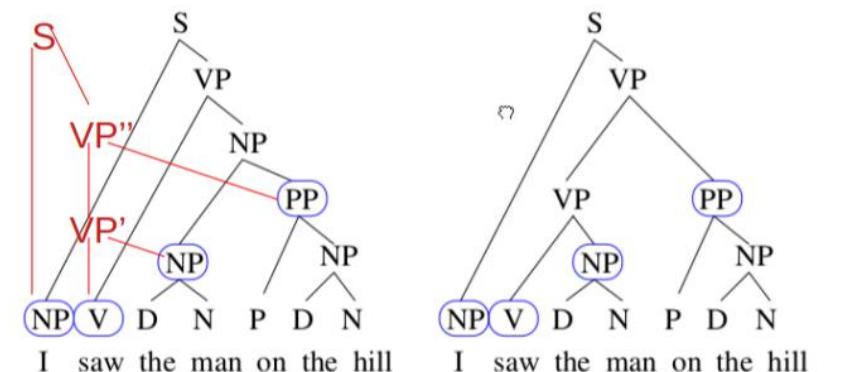
How to merge tree 1 with tree 2

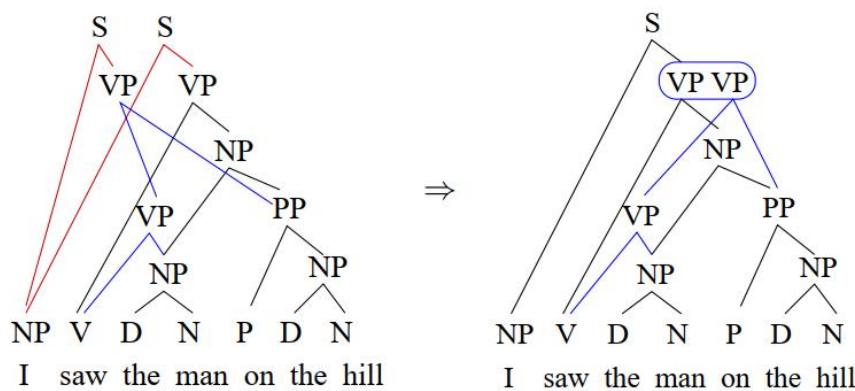
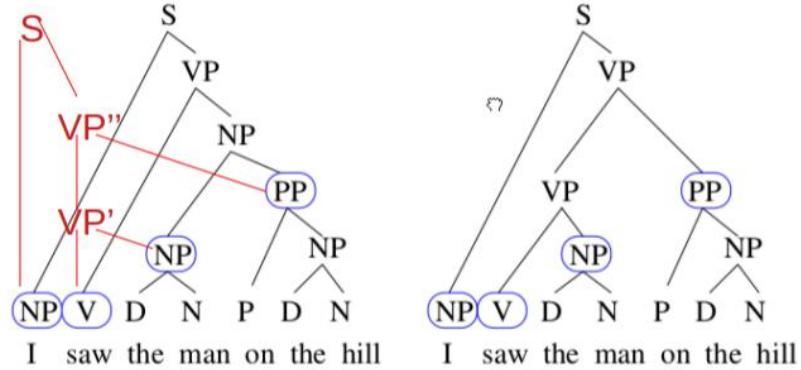
- mark nodes that are identical in both trees

- 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'





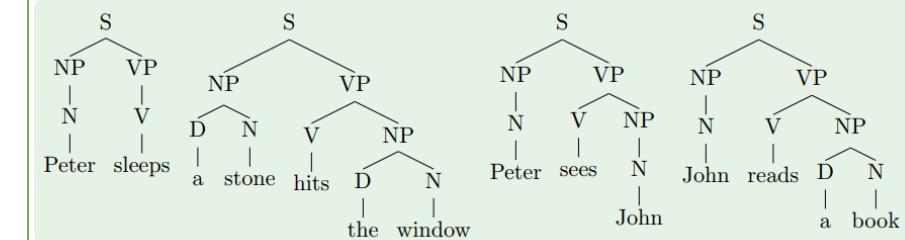
group the same symbol together
Here S and VP are grouped together

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:

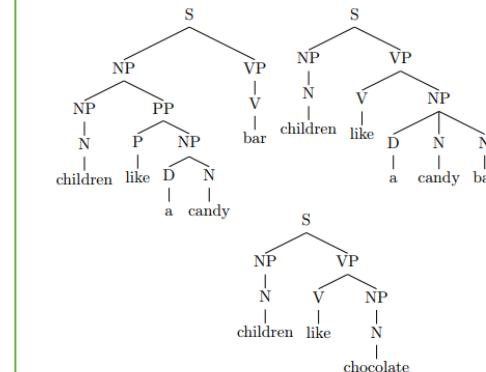
$S \rightarrow NP\ VP$	4	1
$VP \rightarrow V\ NP$	3	0.75
$VP \rightarrow V$	1	0.25
$NP \rightarrow D\ N$	3	0.43
$NP \rightarrow N$	4	0.57

$D \rightarrow a$	2	0.67
$D \rightarrow the$	1	0.33
$V \rightarrow sleeps$	1	0.25
$V \rightarrow hits$	1	0.25
$V \rightarrow sees$	1	0.25
$V \rightarrow reads$	1	0.25

$N \rightarrow Peter$	2	0.29
$N \rightarrow John$	2	0.29
$N \rightarrow stone$	1	0.14
$N \rightarrow window$	1	0.14
$N \rightarrow book$	1	0.14

Training auf unannotierten Daten

children like a candy bar
children like chocolate



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

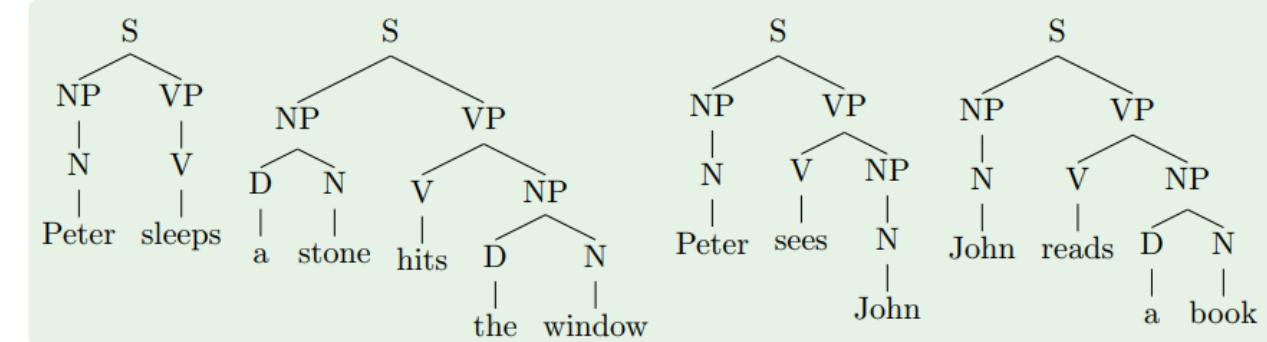
- ① durch Zählen der Regeln in einem manuell mit Parsebäumen annotierten Korpus (Baumbank)
- ② aus einem automatisch geparsten und desambiguierten Korpus (EM-Training)

Schätzung der Regelwahrscheinlichkeiten mit relativen Häufigkeiten

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

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

Baumbank-Training



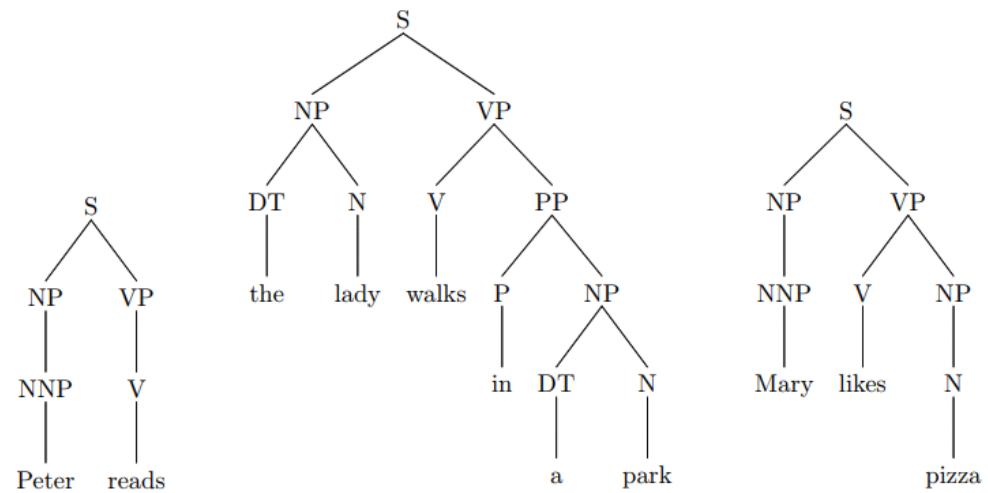
Extraktion der Grammatikregeln und ihrer Häufigkeiten:

$S \rightarrow NP\ VP$	4	1
$VP \rightarrow V\ NP$	3	0.75
$VP \rightarrow V$	1	0.25
$NP \rightarrow D\ N$	3	0.43
$NP \rightarrow N$	4	0.57

$D \rightarrow a$	2	0.67
$D \rightarrow the$	1	0.33
$V \rightarrow sleeps$	1	0.25
$V \rightarrow hits$	1	0.25
$V \rightarrow sees$	1	0.25
$V \rightarrow reads$	1	0.25

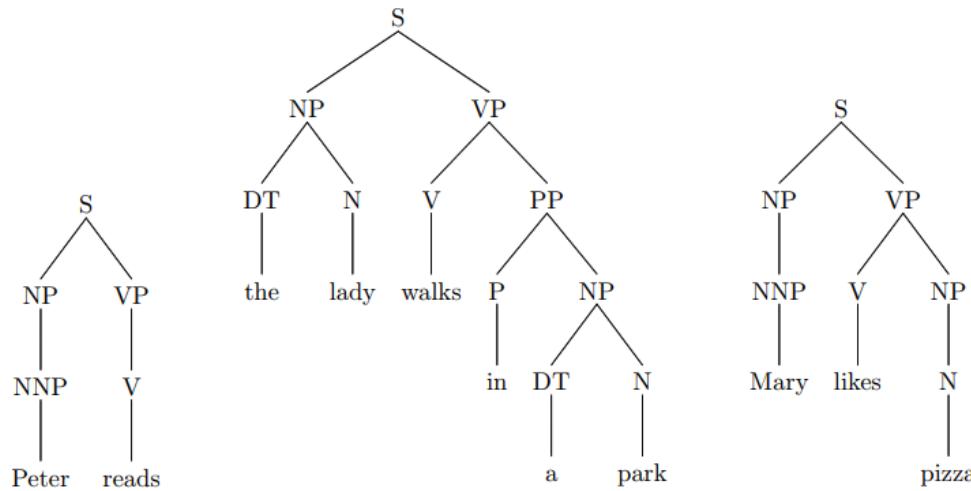
$N \rightarrow Peter$	2	0.29
$N \rightarrow John$	2	0.29
$N \rightarrow stone$	1	0.14
$N \rightarrow window$	1	0.14
$N \rightarrow book$	1	0.14

Aufgabe 2) Extrahieren Sie eine kontextfreie **Grammatik** inklusive Regelhäufigkeiten aus der folgenden Baumbank. Schätzen Sie dann die ungeglätteten **Regelwahrscheinlichkeiten**.



(4 Punkte)

Aufgabe 2) Extrahieren Sie eine kontextfreie **Grammatik** inklusive Regelhäufigkeiten aus der folgenden Baumbank. Schätzen Sie dann die ungeglätteten Regelwahrscheinlichkeiten.



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

(4 Punkte)

Lösung

Aufgabe 2)
Regel Häufigkeit Wahrscheinlichkeit

S --> NP VP	3	1	
VP --> V	1	1/3	
VP --> V NP	1	1/3	
VP --> V PP	1	1/3	
NP --> DT N	2	2/5	
NP --> N	1	1/5	
NP --> NNP	2	2/5	
PP --> P NP	1	1	
DT --> a	1	1/2	
DT --> the	1	1/2	
N --> lady	1	1/3	
N --> park	1	1/3	
N --> pizza	1	1/3	
NNP --> Mary	1	1/2	
NNP --> Peter	1	1/2	
P --> in	1	1	
V --> likes	1	1/3	
V --> reads	1	1/3	
V --> walks	1	1/3	

EM-Training (unsupervised)

Training auf unannotierten Daten

Ansatz 1 (schlecht)

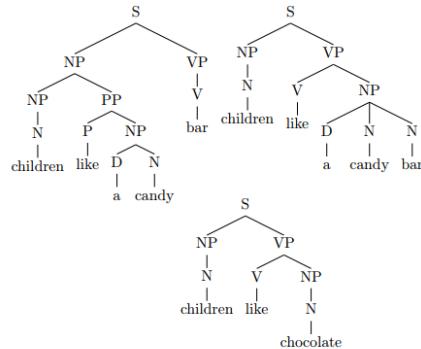
- ① Die Sätze des Trainingskorpus werden (symbolisch) geparsst.
- ② Alle Analysen aller Sätze werden zusammen als "Baumbank" gespeichert.
- ③ Die Regelwahrscheinlichkeiten werden aus der Baumbank geschätzt.

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

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 auftauchen.

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 gegebenen 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

Ansatz 3: (Forts.)

EM-Training

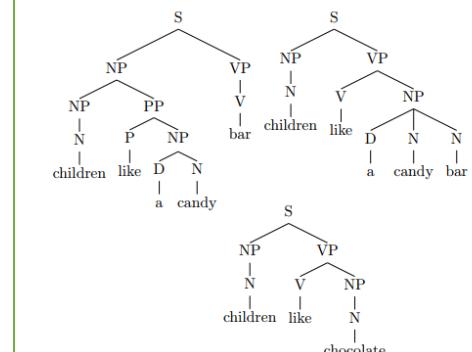
- ➊ Initialisierung der Regelwahrscheinlichkeiten
- ➋ für alle Sätze (E-Schritt)
 - ▶ Berechnung der Parsebäume für den Satz
 - ▶ Berechnung der Parsebaumgewichte $p(t|s)$
 - ▶ Extraktion der gewichteten Regelhäufigkeiten
- ➌ Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt)
- ➍ Weiter mit Schritt 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

children like a candy bar
children like chocolate



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

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

Training auf unannotierten Daten

Ansatz 3: (Forts.)

EM-Training

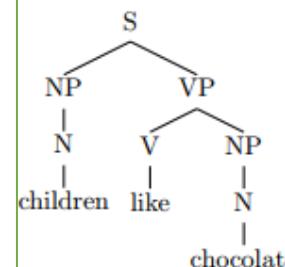
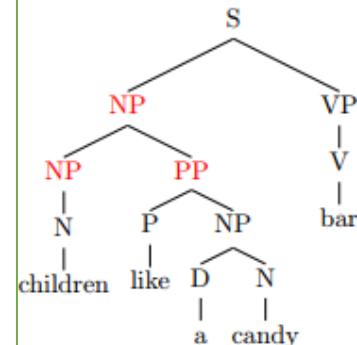
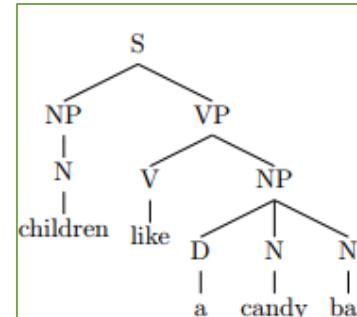
- ① Initialisierung der Regelwahrscheinlichkeiten
- ② für alle Sätze (E-Schritt)
 - ▶ Berechnung der Parsebäume für den Satz
 - ▶ Berechnung der Parsebaumgewichte $p(t|s)$
 - ▶ Extraktion der gewichteten Regelhäufigkeiten
- ③ Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt)
- ④ Weiter mit Schritt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

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



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

$p = 1 / \text{the number of rules that have the form } NP \rightarrow \text{something}$

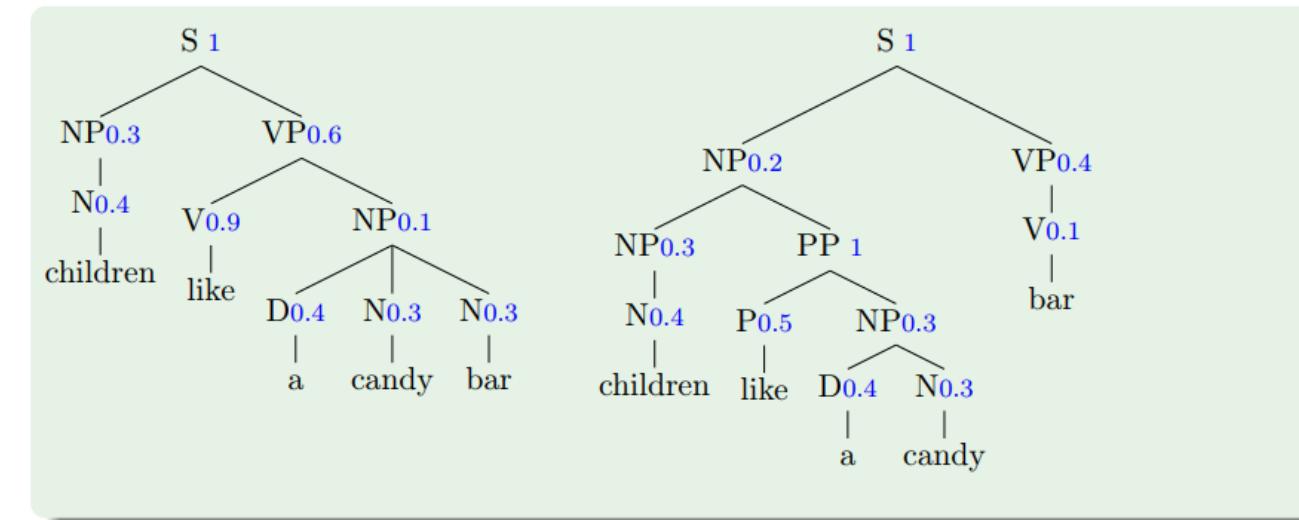
Training auf unannotierten Daten

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 p_0 (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 p_0).



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

$$p(t_1) = 0.0002333$$

$$p(t_1|s) = \frac{p(t_1)}{p(t_1)+p(t_2)} = 0.93$$

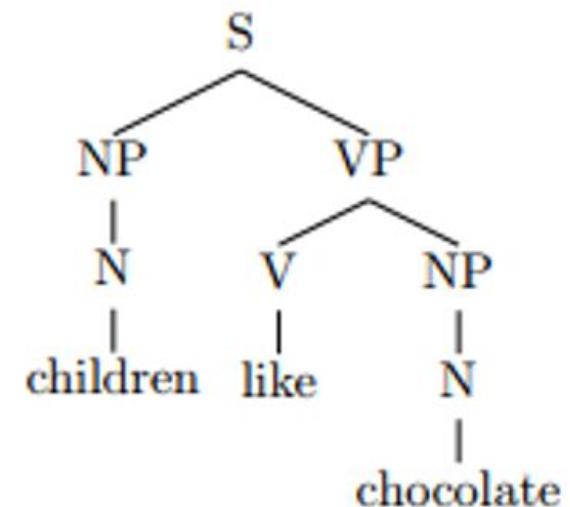
$$p(t_2) = 0.0000173$$

$$p(t_2|s) = \frac{p(t_2)}{p(t_1)+p(t_2)} = 0.07$$

- 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 $p(t_3|s_2) = p(t_3) / p(t_3) = 1$

$p(t_1)$ is simply the sum of every probability in t1.

compute the weight for sentence 2



$$\begin{aligned} p(t_3 | s_2) &= p(t_3) / p(s_2) \\ &= 1 \end{aligned}$$

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

The figure shows three different parse trees for the sentence "children like a candy bar".

- Tree 1:** S → NP VP. NP → N [children] and VP → V [like] NP [NP → D N [a] NP [N [candy] N [bar]]].
- Tree 2:** S → NP VP. NP → NP PP. NP → N [children] and PP → P NP [PP → P [like] NP [D [a] N [candy]]] and VP → V [bar].
- Tree 3:** S → NP VP. NP → N [children] and VP → V NP [V → like] NP [N [chocolate]].

Regel p_0 f_1

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

- ▶ Die aus jedem Parsebaum t extrahierten Regelhäufigkeiten werden gewichtet mit

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

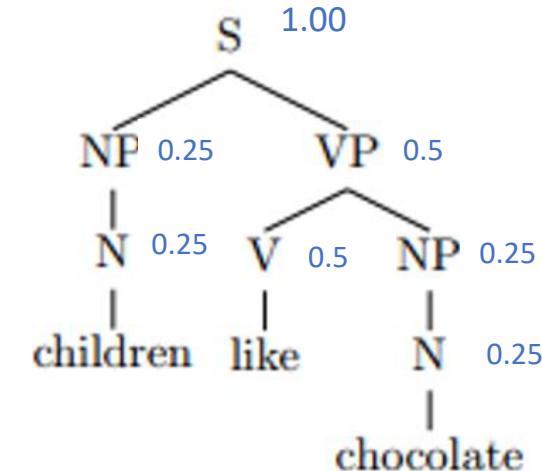
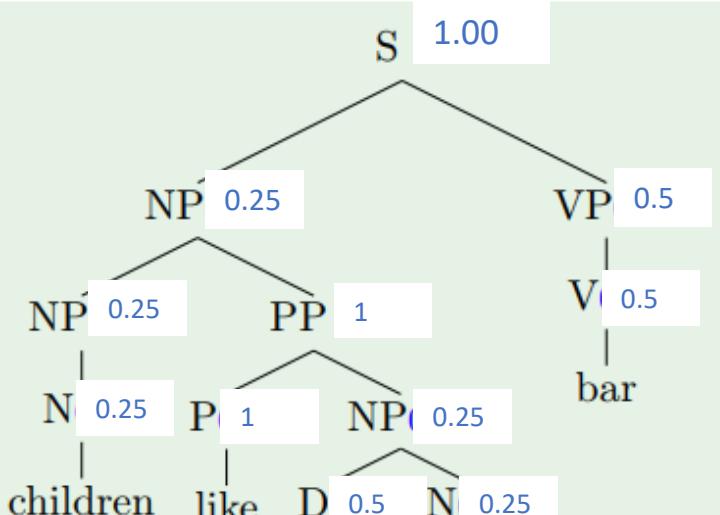
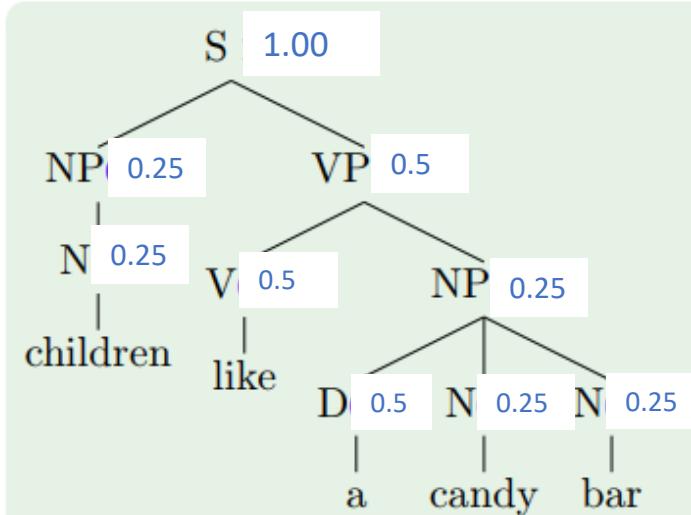
example is on the next page

How to compute f_1 (die gewichteten Häufigkeiten) 1/2

- Die aus jedem Parsebaum t extrahierten Regelhäufigkeiten werden gewichtet mit

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

1) assign p to each rule using p_0



2) compute the weight of each tree

$$p(t_1) = 1 * (0.25 * 5) * (0.5 * 3) = 1.875$$

$$p(t_2) = (1 * 3) * (0.25 * 5) * (0.5 * 3) = 1.875$$

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

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

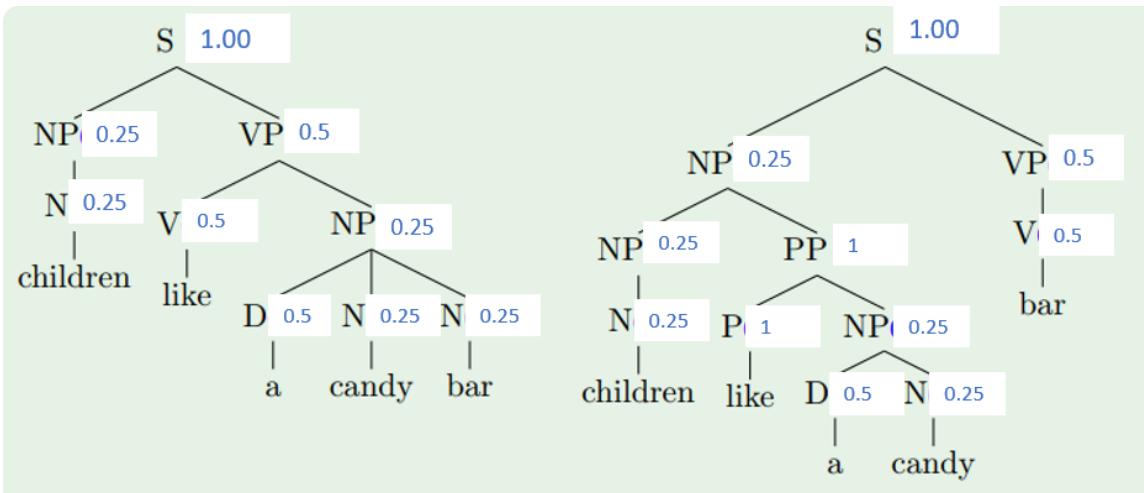
weight of t1

weight of t2

weight of t3

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

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



t1_weight = 0.5

t2_weight = 0.5

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

For example, for $S \rightarrow NP\ VP$

-we have this rule in all trees, so we compute as follows.

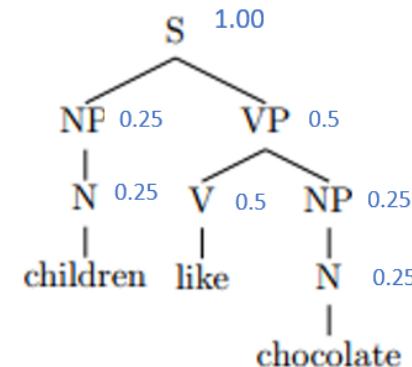
$$p(S \rightarrow NP\ VP) = (1 * 0.5) + (1 * 0.5) + (1 * 1) = 2$$

t2

t3

number of times we found $S \rightarrow NP\ VP$ in t1

weight of t1



t3_weight = 1

Example: compute f1 for $NP \rightarrow NP\ PP$

- We only found this rule in t2

$$p(NP \rightarrow NP\ PP) = 1 * 0.5 = 0.5$$

Example: compute f1 for $NP \rightarrow N$

- we found 1 rule in t1, 1 rule in t2, and 2 rules in t3

$$p(NP \rightarrow NP\ PP) = (1 * 0.5) + (1 * 0.5) + (2 * 1) = 3$$

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

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 geparsst 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:

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

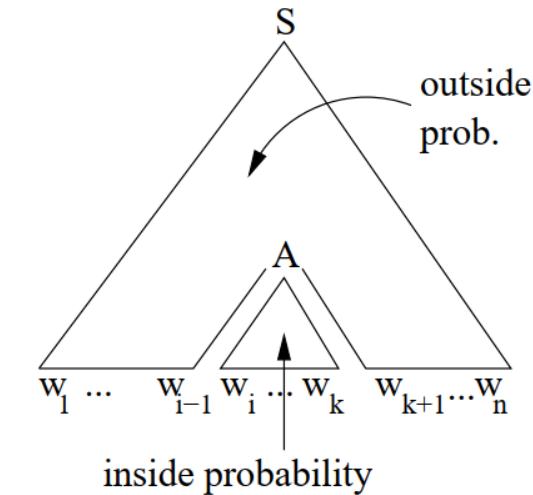
Inside-Outside-Algorithmus

- 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

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



Aufgabe 4) Der Inside-Outside-Algorithmus kann zum Training von probabilistischen kontextfreien Grammatiken (PCFG) auf unannotierten Daten benutzt werden.

Was benötigen Sie außer dem Trainingskorpus sonst noch für das Training? (1 Punkt)

Wie lauten die Formeln zur Berechnung der Inside- und Outside-Wahrscheinlichkeiten?
(3 Punkte)

Wie berechnen Sie die erwartete Häufigkeit der Parsewaldregel $A \rightarrow B_1 \dots B_m$, wenn die Inside-und Outside-Wahrscheinlichkeiten bereits berechnet wurden?
(3 Punkte)

Was wird im M-Schritt des EM-Algorithmus getan? (1 Punkt)

Aufgabe 9) Wie berechnet der Viterbi-Algorithmus den besten Parse für einen gegebenen Parsewald? Wie werden die Viterbi-Wahrscheinlichkeiten berechnet (mit Formeln)?
(5 Punkte)

