content

- Viterbi for PCFG
- Inside-Outside-Algorithmus
- Berkeley-Parser

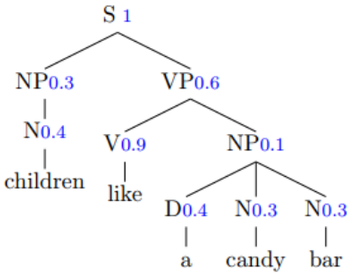
Viterbi for PCFG

What does the Viterbi algorithm compute and where and when do we use it?

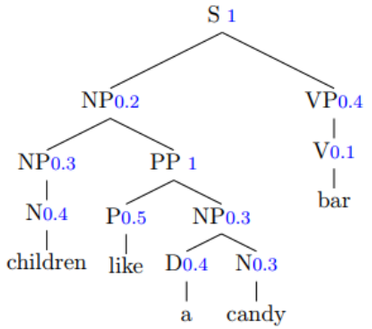
- we use it to find the best analysis (best parse tree) and the probability of it
- when the probability of each grammar rule is already given/computed.

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



$p_1 = 0.0002333$



$p_2 = 0.0000173$

recap

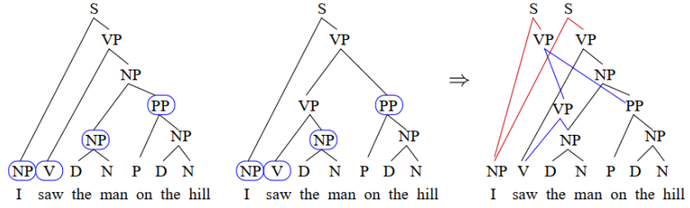
- The initial problem that we want to solve is to find the best syntactic analysis of a sentence that can be parsed in more than one way by a symbolic syntactic parser (symbolische Parser)
- a symbolic parser (in this lecture) is a parser that uses some context-free grammar (without probability) to compute syntactic analyses of a sentence.
- In the example (left), we already have a PCFG (grammar rules with probabilities), so we can compute the probability of each tree for the sentence "children like a candy bar". The best tree is the one with the highest probability.
- we have learned how to estimate the prob for each rule (using Baumbank training or EM-training) and also how to compute the prob of a tree (by multiplying the prob of all nodes in the tree together)

- another way to determine the best tree is to use the **Viterbi algorithm**. To do that, we first have to merge both trees together to create a **Parsewald.**

Parsewald

Beispiel

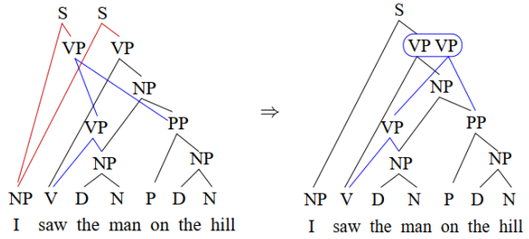
Zusammenfassen (maximaler) gemeinsamer Teilbäume



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

Beispiel

Zusammenfassen von Parsebäumen, die sich nur in einem Teilbaum unters

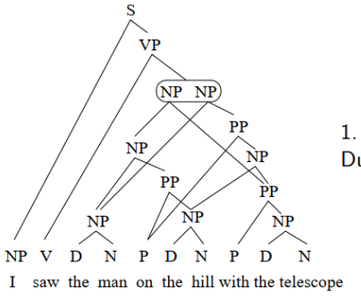


Hier werden die beiden roten Teilbäume zusammengefasst.
Die beiden VP-Teilbäume werden in einem ambigen VP-Knoten vereinigt.
Die blauen Kanten gehören zum 2. Parsebaum.

See the lecture video (Vorlesung8) to see how to use Viterbi

Parsewälder als spezialisierte Grammatiken

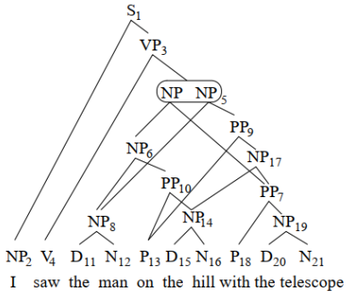
Wir definieren nun für jeden Parsewald eine Grammatik, die genau die darin enthaltenen Parsebäume generiert.



1. Schritt:
Durchnumerierung der Knoten

Parsewald als spezialisierte Grammatik

2. Schritt: Extraktion der Grammatikregeln



$S_1 \rightarrow NP_2\ VP_3$
$NP_2 \rightarrow I$
$VP_3 \rightarrow V_4\ NP_5$
$V_4 \rightarrow saw$
$NP_5 \rightarrow NP_6\ PP_7$
$NP_5 \rightarrow NP_8\ PP_9$
$NP_6 \rightarrow NP_8\ PP_{10}$
$NP_8 \rightarrow D_{11}\ N_{12}$
$D_{11} \rightarrow the$
$N_{12} \rightarrow man$
$PP_{10} \rightarrow P_{13}\ NP_{14}$
$NP_{14} \rightarrow \ldots$

Diese Grammatik generiert genau die Parsebäume, die zu dem Parsewald zusammengefasst wurden (wenn man ignoriert, dass die Symbole noch Indizes tragen).

- Der **Viterbi**-Algorithmus berechnet die Wahrscheinlichkeit der **besten Analyse** jedes Parsewaldknotens

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

Berechnung der Viterbi-Wahrscheinlichkeiten

Viterbi-Algorithmus

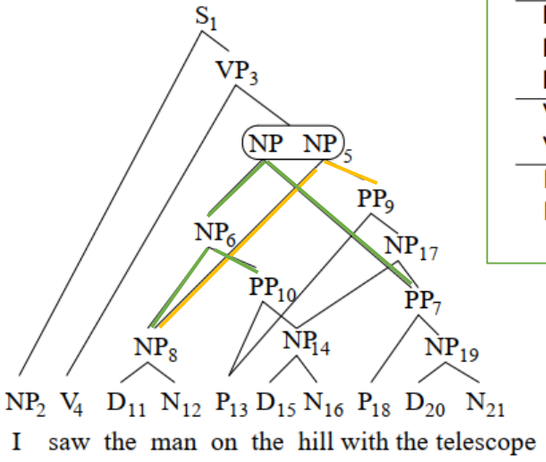
$$
\begin{aligned}
\delta(a) &= 1 \quad \text{für jedes Terminalsymbol a} \\
\delta(A \to X_1 \ldots X_n) &= p(A \to X_1 \ldots X_n) \prod_{i=1}^{n} \delta(X_i) \quad \text{für Parsewald-Regeln} \\
\delta(A) &= \max_{\alpha} \delta(A \to \alpha) \quad \text{für Nichtterminale A} \\
\psi(A) &= \arg\max_{\alpha} \delta(A \to \alpha) \quad \text{beste Analyse von A}
\end{aligned}
$$

$p(S_1 \to NP_2\ VP_3)$ ist die Wahrscheinlichkeit $p(S \to NP\ VP)$ der entsprechenden PCFG-Regel.

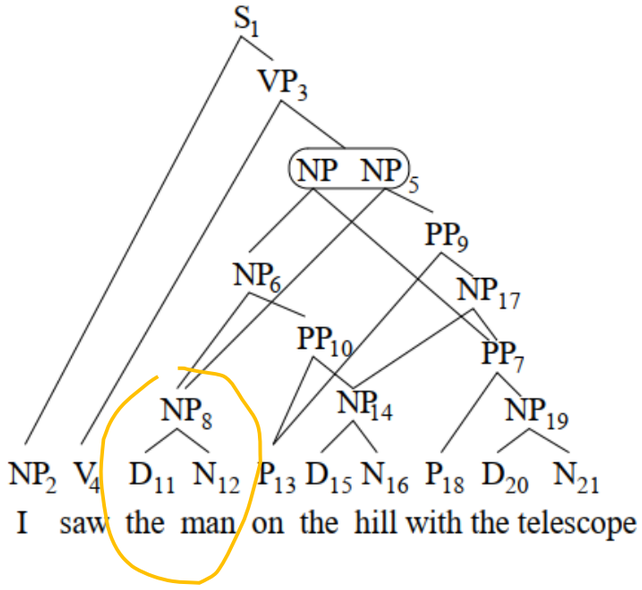
Die ψ-Variable speichert die beste Parsewald-Regel für jedes Nichtterminal.

Viterbi-Beispiel

$$
\begin{aligned}
\delta(I) &= 1 \\
\delta(saw) &= 1 \\
&\ldots \\
\delta(NP_2) &= \delta(NP_2 \to I) \\
&= p(NP \to I)\ \delta(I) \\
\delta(V_4) &= p(V \to saw)\ \delta(saw) \\
&\ldots \\
\delta(NP_8) &= p(NP \to D\ N)\ \delta(D_{11})\ \delta(N_{12}) \\
&\ldots \\
\delta(NP_6) &= p(NP \to NP\ PP)\ \delta(NP_8)\ \delta(PP_{10}) \\
&\ldots \\
\delta(NP_5) &= \max(p(NP \to NP\ PP)\ \delta(NP_6)\ \delta(PP_7), p(NP \to NP\ PP)\ \delta(NP_8)\ \delta(PP_9)) \\
&\ldots \\
\delta(S_1) &= p(S \to NP\ VP)\ \delta(NP_2)\ \delta(VP_3)
\end{aligned}
$$



Exercise1 : compute vit(NP8)



grammar for this Parsewald

$S_1 \rightarrow NP_2\ VP_3$
$NP_2 \rightarrow I$
$VP_3 \rightarrow V_4\ NP_5$
$V_4 \rightarrow saw$
$NP_5 \rightarrow NP_6\ PP_7$
$NP_5 \rightarrow NP_8\ PP_9$
$NP_6 \rightarrow NP_8\ PP_{10}$
$NP_8 \rightarrow D_{11}\ N_{12}$
$D_{11} \rightarrow the$
$N_{12} \rightarrow man$
$PP_{10} \rightarrow P_{13}\ NP_{14}$
$NP_{14} \rightarrow \ldots$

Viterbi-Algorithmus

(1) $\delta(a) = 1$ für jedes Terminalsymbol a

(3) $\delta(A \rightarrow X_1 \ldots X_n) = p(A \rightarrow X_1 \ldots X_n) \prod_{i=1}^{n} \delta(X_i)$ für Parsewald-Regeln

(2) $\delta(A) = \max_{\alpha} \delta(A \rightarrow \alpha)$ für Nichtterminale A

$\psi(A) = \arg\max_{\alpha} \delta(A \rightarrow \alpha)$ beste Analyse von A

global grammar

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 → candv		0.3
N → man		0.3
V → bar		0.1
V → like		0.9
P → like		0.5
P → for		0.5

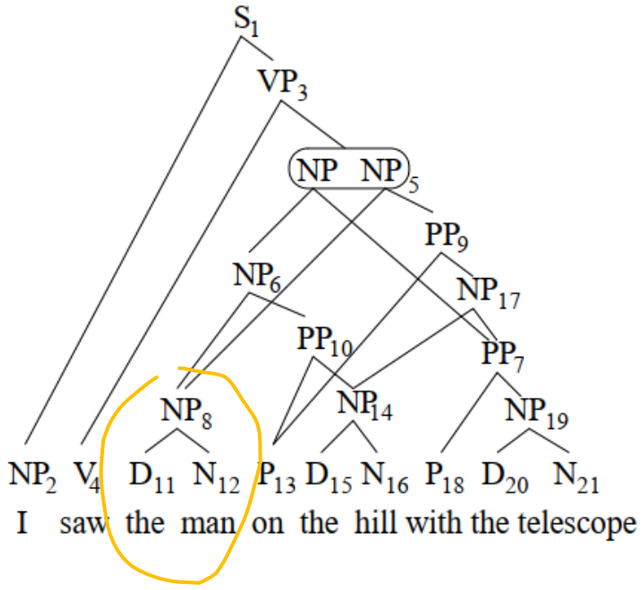
steps
1. compute vit(the), vit(man)
2. then vit(D11), vit(N12)
3. then vit(NP8)

explain

1. compute vit(the), vit(man) using (1)
   vit(the) = 1
   vit(man) = 1

2. then vit(D11), vit(N12) using (2)
for D11, look at the grammar for Parsewald and find every rule that starts with D11, here we found only one (D11-> the), so we have
   vit(D11) = max [vit (D11-> the)]
   = vit (D11 -> the) # then use (3)
   = p(D -> the) vit(the)
   # look at the global grammar for p(D -> the)
   = 0.6 * 1
   = 0.6

Exercise1 : compute vit(NP8)



grammar for this Parsewald

$S_1 \rightarrow NP_2\ VP_3$
$NP_2 \rightarrow I$
$VP_3 \rightarrow V_4\ NP_5$
$V_4 \rightarrow saw$
$NP_5 \rightarrow NP_6\ PP_7$
$NP_5 \rightarrow NP_8\ PP_9$
$NP_6 \rightarrow NP_8\ PP_{10}$
$NP_8 \rightarrow D_{11}\ N_{12}$
$D_{11} \rightarrow the$
$N_{12} \rightarrow man$
$PP_{10} \rightarrow P_{13}\ NP_{14}$
$NP_{14} \rightarrow \ldots$

steps
1. compute vit(the), vit(man)
2. then vit(D11), vit(N12)
3. then vit(NP8)

explain (continue)
2. then vit(D11), vit(N12) using (2)
vit(N12) = max (vit(N12 -> man))
= vit(N12 -> man)
= p(N -> man) vit(man)
= 0.3 * 1
= 0.3

Viterbi-Algorithmus

(1) $\delta(a) = 1$ für jedes Terminalsymbol a

(3) $\delta(A \rightarrow X_1 \ldots X_n) = p(A \rightarrow X_1 \ldots X_n) \prod_{i=1}^{n} \delta(X_i)$ für Parsewald-Regeln

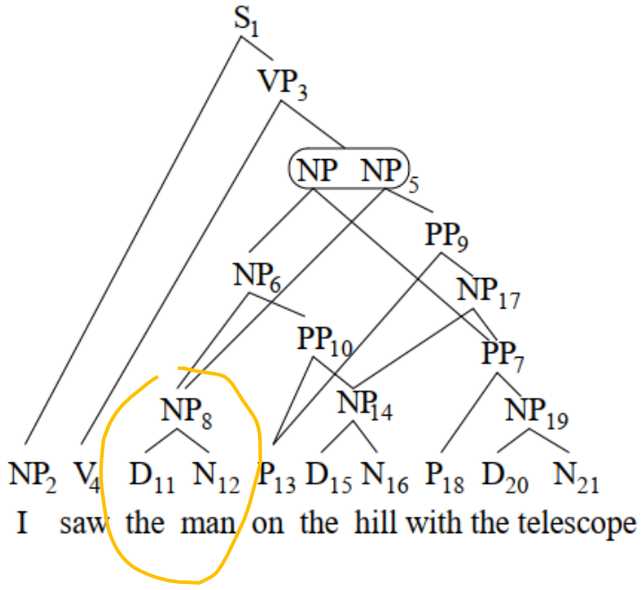
(2) $\delta(A) = \max_{\alpha} \delta(A \rightarrow \alpha)$ für Nichtterminale A

$\psi(A) = \arg\max_{\alpha} \delta(A \rightarrow \alpha)$ beste Analyse von A

global grammar

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	→ candv	0.3
N	→ man	0.3
V	→ bar	0.1
V	→ like	0.9
P	→ like	0.5
P	→ for	0.5

Exercise1 : compute vit(NP8)



grammar for this Parsewald

$S_1 \rightarrow NP_2\ VP_3$
$NP_2 \rightarrow I$
$VP_3 \rightarrow V_4\ NP_5$
$V_4 \rightarrow saw$
$NP_5 \rightarrow NP_6\ PP_7$
$NP_5 \rightarrow NP_8\ PP_9$
$NP_6 \rightarrow NP_8\ PP_{10}$
$NP_8 \rightarrow D_{11}\ N_{12}$
$D_{11} \rightarrow the$
$N_{12} \rightarrow man$
$PP_{10} \rightarrow P_{13}\ NP_{14}$
$NP_{14} \rightarrow \ldots$

Viterbi-Algorithmus

(1) $\delta(a) = 1$ für jedes Terminalsymbol a

(3) $\delta(A \rightarrow X_1 \ldots X_n) = p(A \rightarrow X_1 \ldots X_n) \prod_{i=1}^{n} \delta(X_i)$ für Parsewald-Regeln

(2) $\delta(A) = \max_{\alpha} \delta(A \rightarrow \alpha)$ für Nichtterminale A

$\psi(A) = \arg\max_{\alpha} \delta(A \rightarrow \alpha)$ beste Analyse von A

steps
1. compute vit(the), vit(man)
2. then vit(D11), vit(N12)
3. then vit(NP8)

explain (continue)
3. compute vit(NP8)

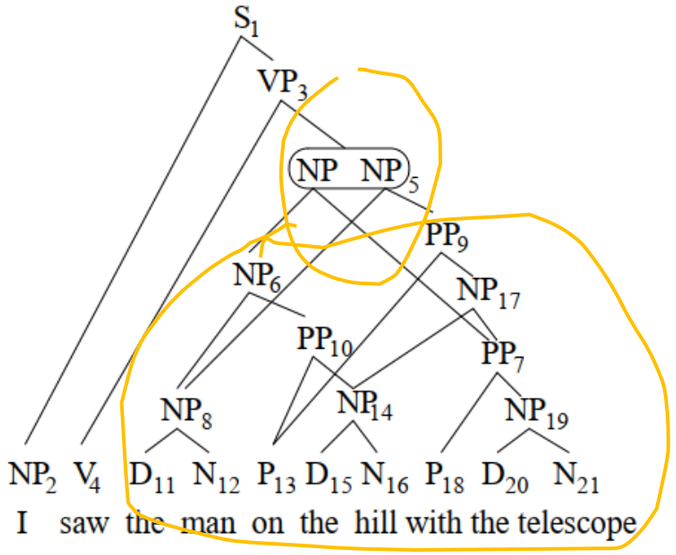
vit(NP8) = max [vit(NP8 -> D11 N12)]
= vit(NP8 -> D11 N12)
= p(NP -> D N) vit(D11) vit(N12)
= 0.3 * 0.6 * 0.3

finished

global grammar

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	→ candv	0.3
N	→ man	0.3
V	→ bar	0.1
V	→ like	0.9
P	→ like	0.5
P	→ for	0.5

Exercise2 : compute vit(NP5)



$S_1 \rightarrow NP_2\ VP_3$
$NP_2 \rightarrow I$
$VP_3 \rightarrow V_4\ NP_5$
$V_4 \rightarrow saw$
$NP_5 \rightarrow NP_6\ PP_7$
$NP_5 \rightarrow NP_8\ PP_9$
$NP_6 \rightarrow NP_8\ PP_{10}$
$NP_8 \rightarrow D_{11}\ N_{12}$
$D_{11} \rightarrow the$
$N_{12} \rightarrow man$
$PP_{10} \rightarrow P_{13}\ NP_{14}$
$NP_{14} \rightarrow \ldots$

steps (in the actual computation)
-compute vit prob of every node below NP5
-then compute vit(NP5)

In the exam, you may only have to write down the computation of vit(NP5) and do not need to actually compute it for other nodes below NP5 (see the answer example)

Answer
vit(NP5) = max [vit(NP5 -> NP6 PP7), vit(NP5 -> NP8 PP9)] (2)
where
vit(NP5 -> NP6 PP7) = p(NP-> NP PP) vit(NP6) vit(PP7)
vit(NP5 -> NP8 PP9) = p(NP -> NP PP) vit(NP8) vit(PP9) (3)

finished

Viterbi-Algorithmus

(1) $\delta(a) = 1$ für jedes Terminalsymbol a

(3) $\delta(A \rightarrow X_1 \ldots X_n) = p(A \rightarrow X_1 \ldots X_n) \prod_{i=1}^{n} \delta(X_i)$ für Parsewald-Regeln

(2) $\delta(A) = \max_{\alpha} \delta(A \rightarrow \alpha)$ für Nichtterminale A

$\psi(A) = \arg\max_{\alpha} \delta(A \rightarrow \alpha)$ beste Analyse von A

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	→ man	0.3
V	→ bar	0.1
V	→ like	0.9
P	→ like	0.5
P	→ for	0.5

Inside Outside

Inside-Outside-Algorithmus

Der **Inside-Outside-Algorithmus**

Anwendung

- berechnet effizient die **erwarteten Regelhäufigkeiten** beim EM-Training
- und entspricht damit dem Forward-Backward-Algorithmus bei den HMMs.
- Er berechnet bottom-up **Inside**-Wahrscheinlichkeiten (ähnlich dem Viterbi-Algorithmus)
- und top-down **Outside**-Wahrscheinlichkeiten.
- Aus den Inside- und Outside-Wahrscheinlichkeiten berechnet er die **erwarteten Häufigkeiten**.

To use the Inside-Outside-Algorithm to compute "die erwartete Häufigkeit" *γ(A -> B)* , we first compute the Inside-WK for every node (1) and rule (2) of the given Parsewald and grammar rule with prob. Then compute the Outside-WK. Then we can compute γ(A -> B) for every rule A-> B in the Parsewald. Do it for all sentences in the training set, then from γ(A -> B), we can compute "die erwartete Regelhäufigkeiten f(A -> B) that we can use in EM-Training.

Inside-Wahrscheinlichkeiten

- Der **Viterbi**-Algorithmus berechnet die Wahrscheinlichkeit der **besten Analyse** jedes Parsewaldknotens
- Der **Inside**-Algorithmus berechnet die Gesamtwahrscheinlichkeit **aller Analysen** für jeden Parsewaldknoten.

Inside-Algorithmus

$$(1)\quad \alpha(a) = 1 \quad \text{für Terminalsymbol a}$$

$$(2)\quad \alpha(A \to X_1 \ldots X_n) = p(A \to X_1 \ldots X_n) \prod_{i=1}^{n} \alpha(X_i) \quad \text{für Parsewaldregeln}$$

$$(1)\quad \alpha(A) = \sum_{A \to \gamma} \alpha(A \to \gamma) \quad \text{für Nichtterminale A}$$

Der Unterschied zum Viterbi-Algorithmus ist die Summen-Operation statt der max-Operation.

Old method (inefficient)

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

   compute f
3. Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt) compute p
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	

Old method (inefficient)

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

   compute f
3. Neuschätzung der Regelwahrscheinlichkeiten (M-Schritt) compute p
4. Weiter mit Schritt 2 bis ein Endekriterium erfüllt ist

More efficient with Inside-outside-Algorithm

Neuschätzung der Regel-Wahrscheinlichkeiten

EM-Algorithmus: wiederholte Ausführung der beiden Schritte

1. E-Schritt compute f
   - Jeder Satz des Trainingskorpus wird geparst und ein Parsewald erstellt.
   - Die erwartete Häufigkeit $\gamma(A \to \delta)$ jeder Parsewaldregel $A \to \delta$ wird berechnet.
   - Die erwarteten Häufigkeiten $\gamma(A \to \delta)$ werden für jede CFG-Regel über alle Trainingssätze zu $f(A' \to \delta')$ 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:

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

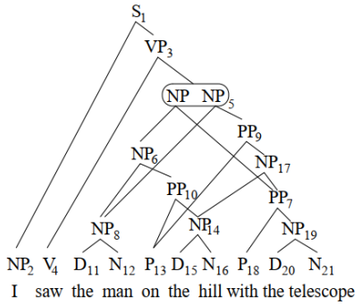
Beispiel

Neuschätzung der Regel-Wahrscheinlichkeiten

EM-Algorithmus: wiederholte Ausführung der beiden Schritte

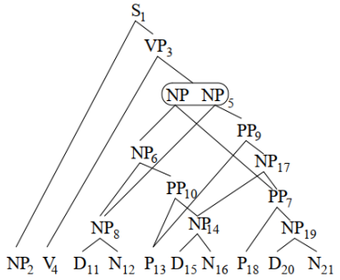
1. E-Schritt — compute f — + initialize p uniformly (1)
   - Jeder Satz des Trainingskorpus wird geparst und ein Parsewald erstellt.
   - Die erwartete Häufigkeit $\gamma(A \to \delta)$ jeder Parsewaldregel $A \to \delta$ wird berechnet. (4)
   - Die erwarteten Häufigkeiten $\gamma(A \to \delta)$ werden für jede CFG-Regel über alle Trainingssätze zu $f(A' \to \delta')$ 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:

(3) $$p(A \to \delta) = \frac{f(A \to \delta)}{\sum_{\delta'} f(A \to \delta')}$$ (2)



S1 → NP2 VP3
NP2 → I
VP3 → V4 NP5
V4 → saw
NP5 → NP6 PP7 gamma= 0.3
NP5 → NP8 PP9 gamma= 0.5
NP6 → NP8 PP10 gamma= 0.5
NP8 → D11 N12
D11 → the
N12 → man
PP10 → P13 NP14
NP14 → …

sentence1



S1 → NP2 VP3
NP2 → I
VP3 → V4 NP5
NP8 -> NP1 PP2 gamma= 0.12
NP6 -> NP2 PP2 gamma= 0.4

NP8 → D11 N12
D11 → the
N12 → man
PP10 → P13 NP14
NP14 → …

sentence2

Suppose we have 2 sentences in our training set, want to compute f(A -> δ) (2) so that we can compute p (A -> δ) (3) and perform the EM-Training. Create a Parsewald for every sentence to merge all possible analyses of the sentence together. Then compute the Inside and Outside WK for all Parsewald. Using the Inside and Outside Wk, we can compute **gamma(A -> *δ)*** *(4)* for every rule in each Parsewald. Then we sum up gamma(A -> δ) from all Parsewald that has the same underlying rule together. The result is f(A -> δ) (2) .
For example, to compute f(NP -> NP PP) we look in Parsewald1 for all rules of this form, sum the gamma value of them. Then look at the Parsewald2, do the same. Sum the results together.

f(NP -> NP PP) = gamma(NP5 -> NP6 PP7) + gamma(NP5 -> NP8 PP9) +gamma(NP6 -> NP8 PP9) + from Parsewald1
gamma(NP8 -> NP1 PP2) + gamma(NP6 -> NP2 PP2) from Parsewald2

(1) (2) (3)

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

Inside-Wahrscheinlichkeiten

- Der **Viterbi**-Algorithmus berechnet die Wahrscheinlichkeit der **besten Analyse** jedes Parsewaldknotens
- Der **Inside**-Algorithmus berechnet die Gesamtwahrscheinlichkeit **aller Analysen** für jeden Parsewaldknoten.

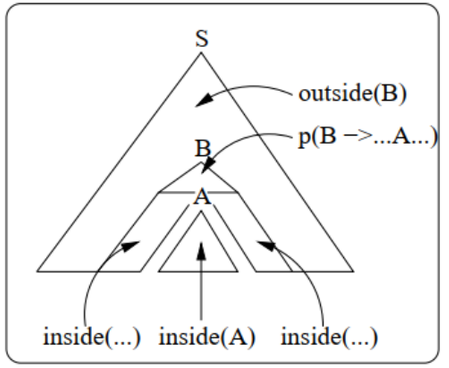
Inside-Algorithmus

$$\begin{aligned} \alpha(a) &= 1 \quad \text{für Terminalsymbol a} \\ \alpha(A \rightarrow X_1 ... X_n) &= p(A \rightarrow X_1 ... X_n) \prod_{i=1}^{n} \alpha(X_i) \quad \text{für Parsewaldregeln} \\ \alpha(A) &= \sum_{A \rightarrow \gamma} \alpha(A \rightarrow \gamma) \quad \text{für Nichtterminale A} \end{aligned}$$

Der Unterschied zum Viterbi-Algorithmus ist die Summen-Operation statt der max-Operation.

Berechnung der Outside-Wahrscheinlichkeiten

Outside-Algorithmus



$$\begin{aligned} \beta(S) &= 1 \quad \text{für Startsymbol S} \\ \beta(A) &= \sum_{B \rightarrow \gamma A \delta} \beta(B \rightarrow \gamma \underline{A} \delta) \end{aligned}$$

$$\beta(B \rightarrow X_1 ... X_m \underline{A} X_{m+1} ... X_n) = \beta(B) p(B \rightarrow X_1 ... X_m A X_{m+1} ... X_n) \prod_{i=1}^{n} \alpha(X_i)$$

$$\beta(B \rightarrow \gamma \underline{A} \delta) = \beta(B) \frac{\alpha(B \rightarrow \gamma A \delta)}{\alpha(A)} \quad \text{mit } \gamma = X_1 ... X_m \text{ und } \delta = X_{m+1},,,X_n$$

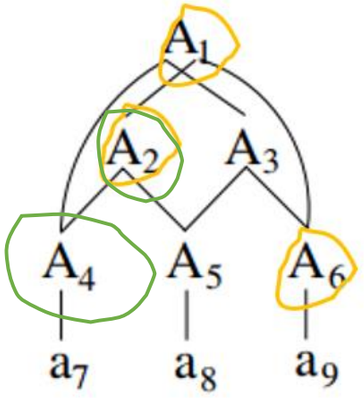
$\beta(B \rightarrow \gamma \underline{A} \delta)$: Beitrag der Regel $B \rightarrow \gamma A \delta$ zur Outside-Wahrscheinlichkeit von A

Beachte, dass $p(B \rightarrow X_1 ... X_m A X_{m+1} ... X_n) \prod_{i=1}^{n} \alpha(X_i) = \alpha(B \rightarrow X_1 ... X_m A X_{m+1} ... X_n) / \alpha(A)$

Outside-Algorithmus

b(A2) = b(A1 -> A2 A3)
= b(A1) p(A -> A A) a(A3)

b(A4) = Sum (b (A1->A4 A3), b(A2 -> A4 A5))
where
b (A1->A4 A3) = b(A1) p(A->A A) a(A3) # method1
b(A2 -> A4 A5) = b(A2) p(A -> A A) a(A5)



$$\beta(S) = 1 \quad \text{für Startsymbol S}$$

$$\beta(A) = \sum_{B \to \gamma A \delta} \beta(B \to \gamma \underline{A} \delta)$$

p

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

Methode 1) $\beta(B \to \gamma \underline{A} \delta) = \beta(B) \frac{\alpha(B \to \gamma A \delta)}{\alpha(A)}$ mit $\gamma = X_1 \ldots X_m$ und $\delta = X_{m+1}, , , X_n$

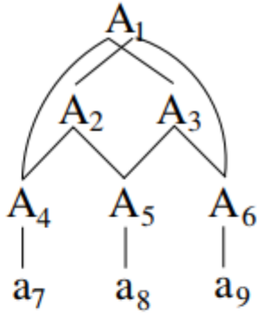
Methode 2) $\beta(B \to X_1 \ldots X_m \underline{A} X_{m+1} \ldots X_n) = \beta(B) p(B \to X_1 \ldots X_m A X_{m+1} \ldots X_n) \prod_{i=1}^{n} \alpha(X_i)$

inside WK

Inside-Outside-Beispiel

PCFG:

$A \rightarrow A\ A$	0.6
$A \rightarrow a$	0.4



$$
\begin{aligned}
\alpha(a_7) &= \mathbf{1} = \alpha(a_8) = \alpha(a_9) \\
\alpha(A_4) &= p(A \rightarrow a)\ \alpha(a_7) = 0.4 * 1 = \mathbf{0.4} = \alpha(A_5) = \alpha(A_6) \\
\alpha(A_2) &= p(A \rightarrow A\ A)\ \alpha(A_4)\ \alpha(A_5) = 0.6 * 0.4 * 0.4 = \mathbf{0.096} \\
\alpha(A_3) &= \alpha(A_2) \\
\alpha(A_1) &= p(A \rightarrow A\ A)\ \alpha(A_4)\ \alpha(A_3) + p(A \rightarrow A\ A)\ \alpha(A_2)\ \alpha(A_6) \\
&= 0.6 * 0.4 * 0.096 * 2 = \mathbf{0.04608} \\
\beta(A_1) &= 1 \\
\beta(A_2) &= \beta(A_1)\ p(A \rightarrow A\ A)\ \alpha(A_6) = 1 * 0.6 * 0.4 = \mathbf{0.24} = \beta(A_3) \\
\beta(A_4) &= \beta(A_1)\ p(A \rightarrow A\ A)\ \alpha(A_3) + \beta(A_2)\ p(A \rightarrow A\ A)\ \alpha(A_5) \\
&= 1 * 0.6 * 0.096 + 0.24 * 0.6 * 0.4 = \mathbf{0.1152} = \beta(A_6) \\
\beta(A_5) &= \beta(A_2)\ p(A \rightarrow A\ A)\ \alpha(A_4) + \beta(A_3)\ p(A \rightarrow A\ A)\ \alpha(A_6) \\
&= 0.24 * 0.6 * 0.4 * 2 = \mathbf{0.1152} \\
\beta(a_7) &= \beta(A_4)\ p(A \rightarrow a) = 0.1152 * 0.4 = \mathbf{0.04608}
\end{aligned}
$$

$$
\begin{aligned}
\gamma(A_2 \rightarrow A_4\ A_5) &= \beta(A_2) p(A \rightarrow A\ A)\ \alpha(A_4)\ \alpha(A_5)/\alpha(A_1) \\
&= 0.24 * 0.6 * 0.4 * 0.4/0.04608 = 0.5
\end{aligned}
$$

- erwartete Regelhäufigkeit

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

Aufgabe 7) Erläutern Sie den EM-Algorithmus am Beispiel des unüberwachten Trainings von PCFGs (also Training auf Rohtexten). Welche Daten benötigen Sie? Welche Berechnungsschritte führt der EM-Algorithmus aus? (4 Punkte)

- Training auf automatisch geparsten Texten (von einem Parser erzeugte Parsewälder)

EM-Algorithmus: wiederholte Ausführung der beiden Schritte

1. E-Schritt
   - Jeder Satz des Trainingskorpus wird geparst und ein Parsewald ausgegeben.
   - Die erwartete Häufigkeit jeder Parsewaldregel wird berechnet.
   - Die erwarteten Häufigkeiten werden für jede CFG-Regel über alle Trainingssätze summiert.
2. M-Schritt
   Die Regel-Wahrscheinlichkeiten werden aus den erwarteten Häufigkeiten neu geschätzt:

$$p(A \to \delta) = \frac{\gamma(A \to \delta)}{\sum_{\delta'} \gamma(A \to \delta')}$$

Parser-Evaluierung

Precision

Es werden korrekte Konstituenten statt korrekter Parsebäume gezählt

Eine Konstituente ist korrekt, wenn der Goldstandard-Parse eine Konstituente mit derselben Start- und Endposition und derselben Kategorie enthält.

$$\text{Precision} = \frac{\text{Anzahl korrekte Konstituenten}}{\text{Anzahl ausgegebene Konstituenten}}$$

Berechnung von Precision und Recall

TP (True Positives): Zahl der ausgegebenen Konstituenten, die korrekt sind

FP (False Positives): Zahl der ausgegebenen Konstituenten, die falsch sind

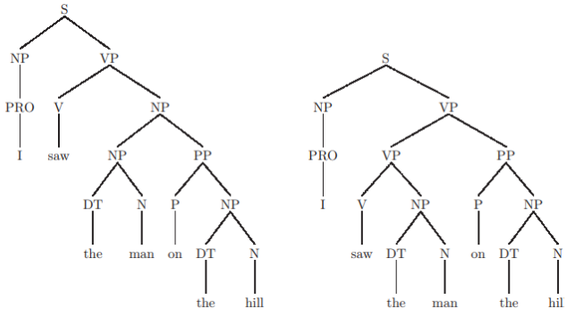
FN (False Negatives): Zahl der Goldstandard-Konstituenten, die fehlten

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \qquad \text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

Mit dem **Recall** messen wir zusätzlich, wieviel Prozent der Goldstandard-Konstituenten im Parse enthalten waren.

Zusammen ergeben Precision und Recall ein genaues Bild der Parsing-Genauigkeit.

Beispiel



Goldstandard Parserausgabe

Goldstandard-Konstituenten:
(S,0,7) (NP,0,1) (VP,1,7) (NP,2,7) (NP,2,4) (PP,4,7) (NP,5,7)

Konstituenten in Parserausgabe:
(S,0,7) (NP,0,1) (VP,1,7) (VP,1,4) (NP,2,4) (PP,4,7) (NP,5,7)

True Positives = 6
False Positives = 1
False Negatives = 1

Precision = TP/(TP+FP) = 6/7
Recall = TP/(TP+FN) = 6/7
F-Score = 2 P R / (P+R) = 2*6/7*6/7/(6/7+6/7)

F-Score

F-Score: harmonisches Mittel aus Precision und Recall

$$F_1 = \frac{1}{\frac{1}{2}(\frac{1}{P} + \frac{1}{R})} = \frac{2}{\frac{R}{PR} + \frac{P}{PR}} = \frac{2PR}{P+R}$$

Der gewichtete F-Score gibt Precision β-mal mehr Gewicht als Recall:

$$F_\beta = \frac{(1+\beta^2)PR}{\beta^2 P + R}$$

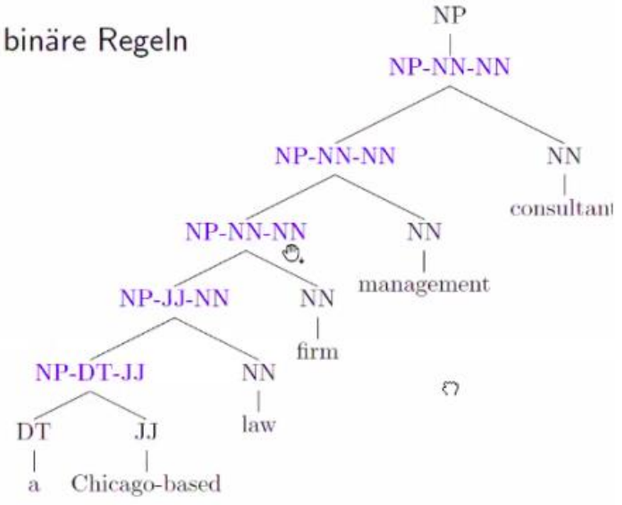
See the lecture video for an explanation

Markowisierung

- Aufspaltung von langen Regeln in binäre Regeln
- Neue Hilfskategorien mit z.B.
  - der Elternkategorie und
  - den Kategorien der beiden letzten Tochterknoten
- Das Entfernen der Hilfsknoten liefert wieder den Originalparse

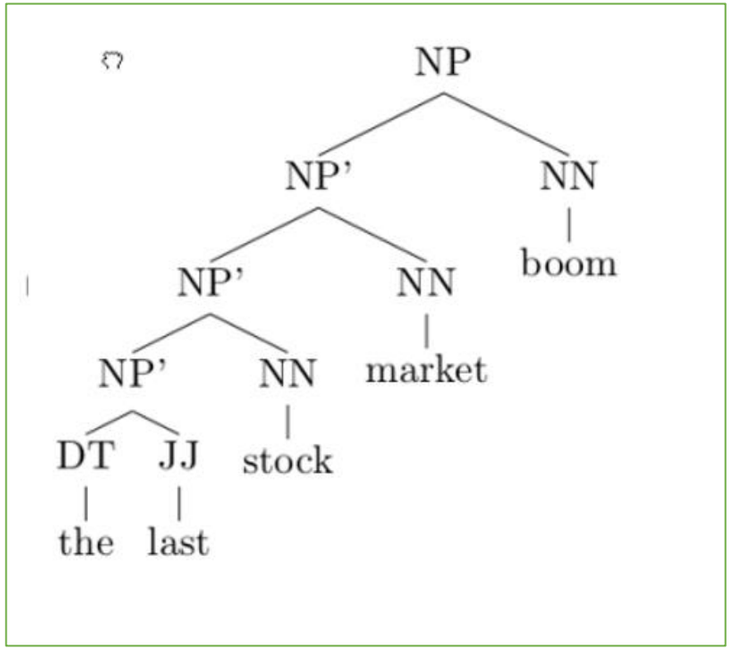
⇒ Reduktion der Grammatikgröße

⇒ Verbesserung ihrer Abdeckung



Die gezeigte Markowisierung ist äquivalent dazu, dass wir die rechten Seiten (und Wahrscheinlichkeiten) von bspw. NP-Regeln mit einem Markow-Modell 2. Ordnung erzeugen.

Binarisierung



See the lecture video for an explanation

Berkeley-Parser

Berkeley Parser : Grundidee

Wir wollen die Parse-Kategorien(die nicht-terminalen Symbole wie S, NP, PRO,..) in SubKategorien verfeinern.

Die Methode von Petrov und Klein soll die Subkategorisierung automatisch lernt.

Jede Kategorie wird in zwei neue Kategorien aufgespaltet. Z.B. NP wird NP/0 und NP/1.

Damit bekommen wir auch neue Regel z.B. NP/0 → PRO/0 oder NP/0 → PRO/1 und viele Parsebäume aus der neuen Regeln. Die neue Parsebäume soll mit EM-Training trainiert werden (EM schätzt die WK der neuen Regeln).
Nach dem Training: Wenn wir die Regeln nach dem Kategorie auf der linken Seite grouppieren und die Regeln nach der Regel-WK sortieren, sehen wir die gelernte Subkagerierung.

Beispiel: wahrscheinlichste Expansionen der DT-(Unter-)Kategorien

DT	the (0.50)	a (0.24)	The (0.08)	...
DT/0	that (0.15)	this (0.14)	some (0.11)	...
DT/1	the (0.54)	a (0.25)	The (0.09)	...

DT/0 → that (0.15)
DT/0 → this (0.14)
DT/0 → some (0.11)

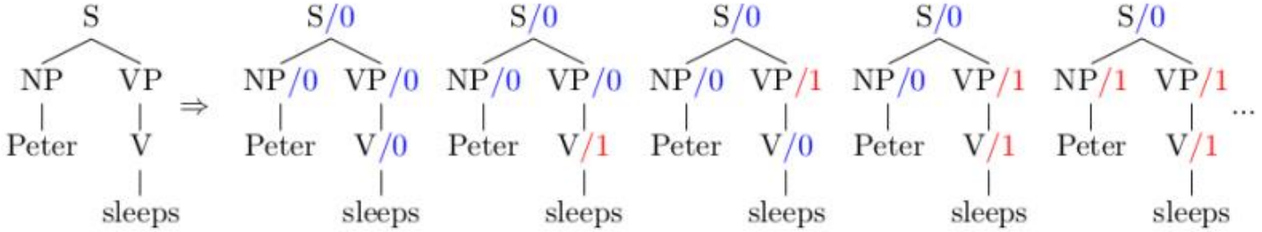
DT/1 → the (0.54)
DT/1 → a (0.25)
DT/1 → The (0.09)

Der orignale Parsebaum wird dann mit den neuen Subkategorien annotiert. Dadurch dass die Kategorien spezifischer werden, wird das Performanz der Parser auch erhöht.

Synthetische Merkmale

Grundidee (von Petrov/Klein)

- Alle Kategorien werden durch ein synthetisches Merkmal mit den Werten 0 bzw. 1 aufgespalten.
- Jeder Parse der Baumbank kann von der neuen Grammatik auf viele unterschiedliche Arten generiert werden.
- Durch EM-Training wird die neue Grammatik an die Baumbank angepasst.



...

NP'/0 → NP'/0 NN/0 0.24

NP'/0 → NP'/0 NN/1 0.26

NP'/0 → NP'/1 NN/0 0.25

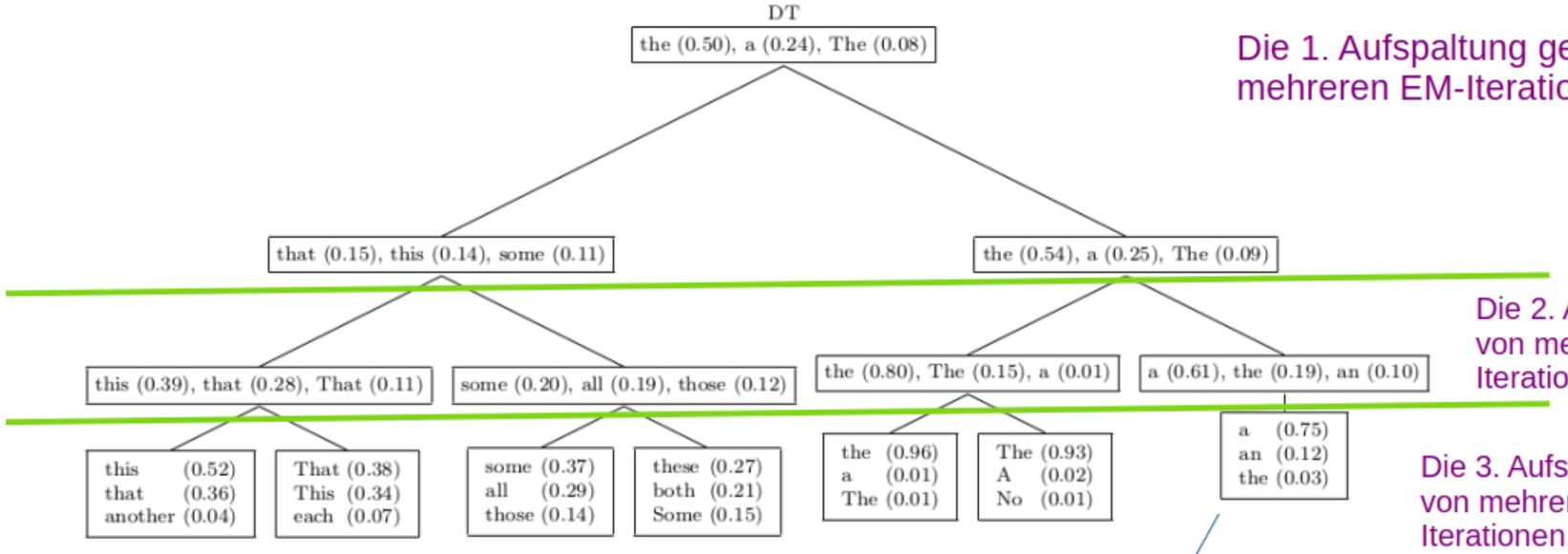
...

Die modifizierte Grammatik liefert für den alten Parsebaum $2^4 = 16$ neue Parsebäume.

Die Subkategorisieren können noch weiter verfeinert werden.

Rekursive Verfeinerung

Rekursives Aufspalten der Kategorien gefolgt von mehreren EM-Iterationen verfeinert die Kategorien immer mehr.



Die 1. Aufspaltung gefolgt von mehreren EM-Iterationen.

Die 2. Aufspaltung gefolgt von mehreren EM-Iterationen.

Die 3. Aufspaltung gefolgt von mehreren EM-Iterationen.

Es werden jeweils nur die 3 wahrscheinlichsten Expansionen gezeigt.

Wenn die Aufspaltung nicht mehr vorteilhaft ist, werden die Subkatogorien wieder zusammengefasst

Beispiel für andere Regeln : hier sehen wir, nach mehrmaligen Aufspaltungen und EM-Trainings wurde ADVP in 9 Subkagegorien unterteilt.

ADVP			
ADVP-0	RB-13 NP-2	RB-13 PP-3	IN-15 NP-2
ADVP-1	NP-3 RB-10	NP-3 RBR-2	NP-3 IN-14
ADVP-2	IN-5 JJS-1	RB-8 RB-6	RB-6 RBR-1
ADVP-3	RBR-0	RB-12 PP-0	RP-0
ADVP-4	RB-3 RB-6	ADVP-2 SBAR-8	ADVP-2 PP-5
ADVP-5	RB-5	NP-3 RB-10	RB-0
ADVP-6	RB-4	RB-0	RB-3 RB-6
ADVP-7	RB-7	IN-5 JJS-1	RB-6
ADVP-8	RB-0	RBS-0	RBR-1 IN-14
ADVP-9	RB-1	IN-15	RBR-0

Original paper: Learning Accurate, Compact, and Interpretable Tree Annotation
https://dl.acm.org/doi/pdf/10.3115/1220175.1220230

Vereinigung

- Ab einem bestimmten Punkt ist eine weitere Aufspaltung der Kategorien nicht mehr vorteilhaft, weil sie zu Sparse-Data-Problemen führt.

  Die Penn-Treebank verwendet bspw. eine eigene Kategorie "," für Kommas, deren Aufspaltung nichts nützt.

⇒ Unterkategorien, die sich zu ähnlich sind, werden daher nach dem EM-Training wieder zusammengefasst.

Strategie:

- Nach jedem EM-Training und vor dem Aufspalten der Kategorien, werden 50% der vorherigen Aufspaltungen rückgängig gemacht.
- Für jede Aufspaltung wird berechnet, wie stark sich die Wahrscheinlichkeit der Trainingsdaten verringert, wenn die Aufspaltung rückgängig gemacht wird. (Details im Originalartikel)
- Die Aufspaltungen mit der kleinsten Differenz werden rückgängig gemacht.

Experimentelle Ergebnisse

- "Berkeley"-Parser von Slav Petrov und Dan Klein
- Der Parser liefert für viele verschiedene Sprachen und Baumbanken gute Ergebnisse
- Deutsch (Precision=80.1%, Recall=80.1%)
- Englisch (Precision=90.2%, Recall=89.9%)
- Chinesisch (Precision=84.8%, Recall=81.9%)

Aber: Neuere Parser auf Basis neuronaler Netze sind noch besser.

Kurzzusammenfassung: Berkeley-Parser

1. gegeben: eine **Baumbank**
2. **Binarisierung** der Parsebäume durch Einfügen von Hilfsknoten
3. Extraktion einer Grammatik mit Regelhäufigkeiten
4. für mehrere Iterationen
   1. Aufspaltung jeder Kategorie X in zwei neue Kategorien X1 und X2
   2. annähernd uniforme Verteilung der Häufigkeit einer Regel auf die neuen Regeln
   3. für mehrere Iterationen
      1. Schätzung der Regelwahrscheinlichkeiten aus den Regelhäufigkeiten
      2. Für jeden Parsebaum der Baumbank wird der Parsewald mit Analysen entsprechend der verfeinerten Grammatik berechnet.
      3. Mit dem Inside-Outside-Algorithmus werden erwartete Häufigkeiten für die Parsewaldregeln berechnet.
      4. Die erwarteten Regelhäufigkeiten werden über alle Sätze summiert.
   4. Die schlechtere Hälfte der Aufspaltungen wird rückgängig gemacht.