Statistische Maschinelle Übersetzung

Teil IV - Phrasenbasierte Übersetzung

Modelle und Dekodierung

Thomas Schoenemann
Heinrich-Heine-Universität Düsseldorf

Sommersemester 2012

Wiederholung: Probabilistische Übersetzung

Erinnerung:

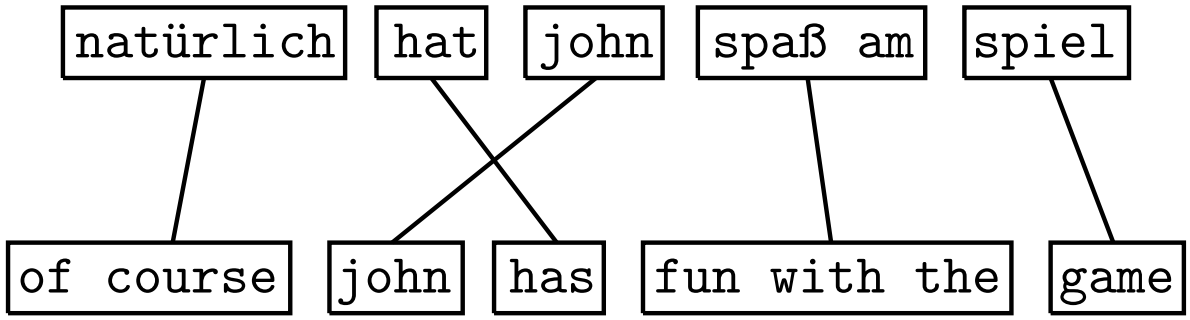
$$\begin{aligned} \text{finde} \quad & \arg\max_{I, e_1^I} \; p(e_1^I, I | f_1^J) \\ = \quad & \arg\max_{I, e_1^I} \; p(f_1^J | e_1^I) \cdot p(e_1^I, I) \end{aligned}$$

$p(e_1^I, I)$: Sprachmodell ($\rightarrow$ siehe Teil III)

$p(f_1^J | e_1^I)$: Übersetzungsmodell. Hier: phrasenbasiert.

Achtung: ursprünglich war f_1^J gegeben.
Durch Umformung der Wk: Übersetzungsmodell konditioniert auf e_1^I

Überblick



Phrasenbasierte Übersetzung: $p(\mathbf{f}|\mathbf{e})$

1. Unterteilung des Zielsatzes in zusammenhängende Segmente
   - d.h. Segmente haben keine Lücken
   - *nicht* angelehnt an linguistische Theorien
2. dann Übersetzung der einzelnen Segmente
3. Schließlich Neuordnung der Quellsatzsegmente

vgl. das Modell von Marcu & Wong (Teil II der Vorlesung)

Mathematisches Modell

Gegeben: Zielsatz $\mathbf{e}$ der Länge I.

1. Segmentierung (unter Einhaltung der Reihenfolge) in eine Sequenz von Phrasen $\bar{e}_1^K$, $K \leq I$:
   – jedes $\bar{e}_k$ ist eine Sequenz von mindestens einem Wort.
   – $\bar{e}_1$ fängt mit e_1 an.
   – $\bar{e}_K$ hört mit e_I auf.
   – hört $\bar{e}_k$ mit e_i auf $(k < K, i < I)$, so fängt $\bar{e}_{k+1}$ mit e_{i+1} an.
2. Der Sequenz $\bar{e}_1^K$ wird nun eine Sequenz $\bar{f}_1^K$ zugeordnet.
3. Letzter Schritt: bestimme die Ordnung der $\bar{f}_k$ d.h. die Reihenfolge, in der die $\bar{f}_k$ hintereinandergehängt werden, um den Zielsatz $\mathbf{f}$ zu bilden. $\rightarrow$ nächste Folie

Mathematisches Modell: Ordnung von $\bar{f}_1^K$

Ordnung mathematisch ausgedrückt durch eine Permutation π der Zahlen $\{1, \ldots, K\}$. $\pi(k) = j$ drückt aus, dass das Segment $\bar{f}_k$ das j-te Segment im **f**-Satz ist.

Mit Umkehrabbildung π^{-1} :

$$\mathbf{f} = \bar{f}_{\pi^{-1}(1)} \circ \ldots \circ \bar{f}_{\pi^{-1}(K)}$$

wobei $\circ$ für Konkatenation steht.

Im Folgenden (für gegebene $\pi, \bar{f}_1^K$):
– $\texttt{start}_k$ = Position des ersten Zeichens von $\bar{f}_k$ in **f**
– $\texttt{end}_k$ = Position des letzten Zeichens von $\bar{f}_k$ in **f**

Basismodell

Erinnerung:

$$\begin{aligned} \text{finde} \quad & \arg\max_{I, e_1^I} \; p(e_1^I, I | f_1^J) \\ = \quad & \arg\max_{I, e_1^I} \; p(f_1^J | e_1^I) \cdot p(e_1^I, I) \end{aligned}$$

$p(e_1^I, I)$: Sprachmodell ($\rightarrow$ siehe Teil III)

Für $p(f_1^J | e_1^I)$: zunächst Wahrscheinlichkeit für eine Folge von $\bar{f}$-Segmenten bei gegebener Segmentierung $\bar{e}_1^K$:

$$p(\bar{f}_1^K, \pi | \bar{e}_1^K) = \prod_{k=1}^{K} p(\bar{f}_k | \bar{e}_k) \cdot p(\mathtt{start_k} - \mathtt{end}_{k-1})$$

$p(\mathtt{start_k} - \mathtt{end}_{k-1})$ heißt *Reordering-Modell*

Gesamtwahrscheinlichkeit

Sei $\mathcal{S}_{\mathbf{f},\mathbf{e}}$ die Menge aller Tupel $(\bar{f}_1^K, \pi, \bar{e}_1^K)$, die ein bestimmtes Paar $\mathbf{f}, \mathbf{e}$ ergeben.

Dann:

$$p(\mathbf{f}|\mathbf{e}) = \sum_{(\bar{f}_1^K, \pi, \bar{e}_1^K) \in \mathcal{S}_{\mathbf{f},\mathbf{e}}} p(\bar{f}_1^K, \pi | \bar{e}_1^K)$$

hier: Gleichverteilung über Segmentierungen $\bar{e}_1^K$ (Term ignoriert)

In der Praxis:

– Summation über alle Tupel nicht effizient machbar.

– Statt dessen: Suche nach nur einem Tupel (mit maximaler Wk.).
(später mehr)

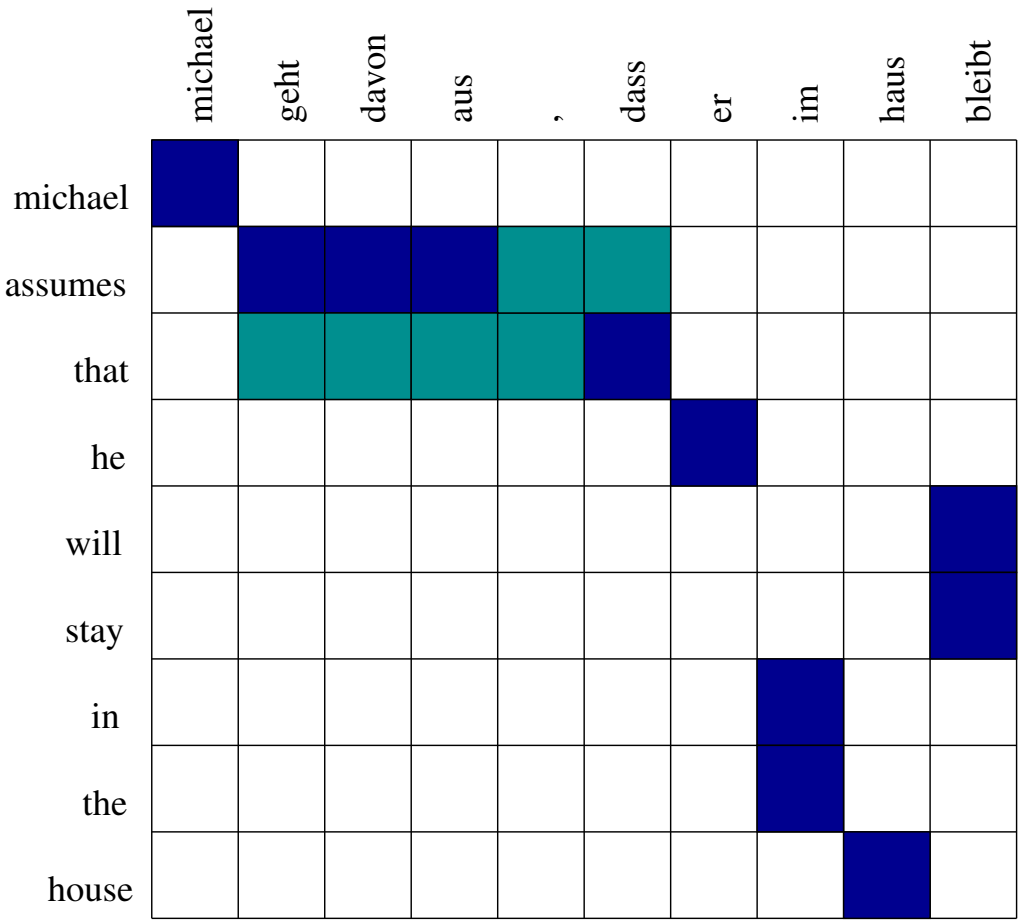
Lernen von Phrasenpaaren

Die Basiswahrscheinlichkeiten für phrasenbasierte Modelle werden (meist) aus Trainingskorpora geschätzt.
Aber: es gibt extrem viele mögliche Phrasenpaare.

In Teil II behandelt: EM-Training für phrasenbasierte Modelle
Aber: sehr rechenzeitaufwändig, nur auf kleinere Corpora anwendbar, da alle möglichen Phrasenpaare gespeichert werden.

In der Praxis: Extraktion aus Word Alignments
- Alignments meist mit einzelwort-basierten Modellen (GIZA++) generiert, dann symmetrisiert mit diag-grow-final-and
- Ergebnis: große Menge an Phrasenpaaren, aber nur ein kleiner Teil aller möglichen.
- resultierende Übersetzungen mindestens genauso gut wie mit EM-Training

Extraktion von Phrasen aus Word Alignments



Extrahiere Boxen, sodass:

- min. ein Alignmentpunkt innerhalb der Box
- kein an der Box beteiligtes Wort aligniert zu einem nicht beteiligten Wort

Extraktion von Phrasen aus Word Alignments - formal

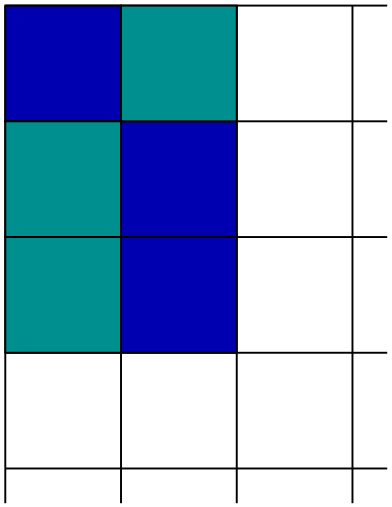
Gegeben:
(i) zwei Sätze $\mathbf{f}, \mathbf{e}$ der Längen J und I.
(ii) ein Word Alignment $\mathbf{a} \subseteq \{1, \ldots J\} \times \{1, \ldots, I\}$

Das Phrasenpaar $f_{j_1}^{j_2}, e_{i_1}^{i_2}$ mit $j_1 \leq j_2, i_1 \leq i_2$ kann extrahiert werden, wenn diese drei Bedingungen gelten:

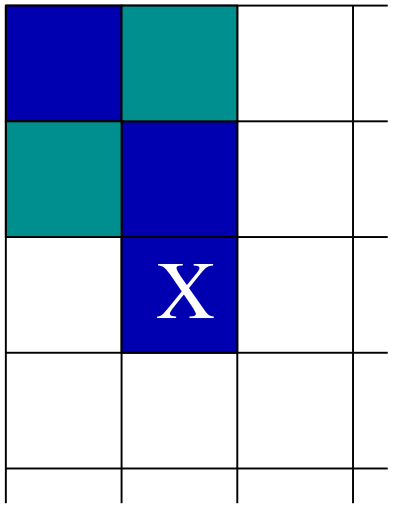
- $\exists\, i, j$ mit $i_1 \leq i \leq i_2$, $j_1 \leq j \leq j_2$, sodass $(j, i) \in A$
- $\forall i \in \{i_1, \ldots, i_2\} : \nexists j$ sodass $j \notin \{j_1, \ldots, j_2\}$ und $(j, i) \in A$
- $\forall j \in \{j_1, \ldots, j_2\} : \nexists i$ sodass $i \notin \{i_1, \ldots, i_2\}$ und $(j, i) \in A$

Zusätzlich: Limit für $|j_2 - j_1|$ und $|i_2 - i_1|$ (meist: nicht größer als 7).

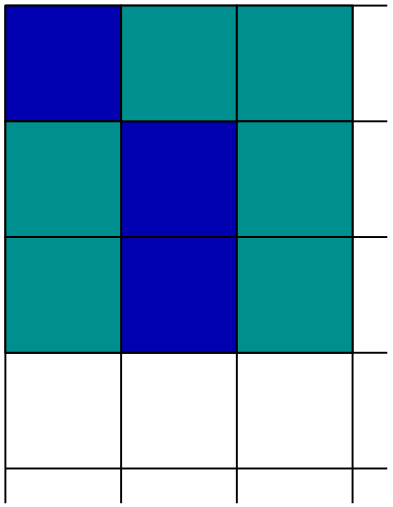
Illustration: extrahierbare und nicht extrahierbare Alignments



extrahierbar



nicht extrahierbar



extrahierbar

Schätzung von Phrasenwahrscheinlichkeiten

Nach der Extraktion:

– Menge von möglichen Phrasenpaaren

– *nicht* bestimmt: Segmentierung für die einzelnen Sätze
(somit bleiben sehr viele mögliche Mengen von Phrasenpaaren, die ein Satzpaar erklären.)

⇒ wir können keine relativen Häufigkeiten nutzen

Statt dessen: $\texttt{count}(\bar{e}, \bar{f}) =$ # Satzpaare, in denen $(\bar{e}, \bar{f})$ extrahiert werden konnte.

Dann:

$$p(\bar{f}|\bar{e}) = \frac{\texttt{count}(\bar{e}, \bar{f})}{\sum_{\bar{f}'} \texttt{count}(\bar{e}, \bar{f}')}$$

Extraktion und Schätzung von Wk.s sind extrem speicheraufwändig,

⇒ Zwischen- und Endergebnisse meist nicht komplett im Speicher

Log-lineare Modelle

Bisher:

– Modelle durch Kettenregel und Unabhängigkeitsannahmen

– erlaubt nur eine gewisse Klasse von Modellen

z.B. bei Word Alignment: kann nicht gleichzeitig auf die Pos. des vorhergehenden f-Wortes konditionieren *und* die Fertilität eines e-Wortes mitberücksichtigen.

– alle Terme werden ungewichtet multipliziert.

Bei phrasenbasierten Modellen: Annahmen zu restriktiv.

Besser: log-lineare Modelle, Grundform:

$$p(\bar{f}_1^K, \pi | \bar{e}_1^K) \propto e^{\sum_i \lambda_i \, g_i(\bar{f}_1^K, \pi, \bar{e}_1^K)}$$

Hier: λ_i Gewichtungsparameter, $g_i(\cdot)$ beliebig wählbare Featurefunktionen (mit Werten in $\mathbb{R}$)

Basismodell als log-lineares Modell

$$p(I, e_1^I | f_1^J, \pi) \quad \propto \quad \exp \left[\lambda_{\mathtt{f|e}} \sum_{k=1}^{K} \log \left(p(\bar{f}_k | \bar{e}_k) \right) \right.$$
$$+ \lambda_{\mathtt{D}} \sum_{k=1}^{K} \log \left(p(\mathtt{start_k} - \mathtt{end}_{k-1}) \right)$$
$$\left. + \lambda_{\mathtt{LM}} \sum_{i=1}^{I+1} \log \left(p(e_i | e_1^{i-1}) \right) \right]$$

Neu hier: Gewichtungsparameter $\lambda_{\mathtt{f|e}}, \lambda_{\mathtt{D}}, \lambda_{\mathtt{LM}}$

für $\lambda_{\mathtt{f|e}} = \lambda_{\mathtt{D}} = \lambda_{\mathtt{LM}} = 1$ ergibt sich das bisherige Basismodell

- Bestimmung dieser Parameter in Teil VI der Vorlesung

Im Folgenden: Erweiterung der Features

Features für Übersetzung

Jetzt problemlos möglich: Integration der umgekehrten Übersetzungsrichtung:

$$\lambda_{\mathtt{e|f}} \sum_{k=1}^{K} \log\left(p(\bar{e}_k|\bar{f}_k)\right)$$

– hilfreich wenn $\bar{e}_k$ selten ist, aber $\bar{f}_k$ häufig.

Lexikalische Gewichtung

Weiterhin: Lexikalische Gewichtung
verhindert, dass lange Phrasen übermäßig bevorzugt werden

Gegeben hier: Alignment $\mathbf{a}_k$ für jedes Phrasenpaar,
statt unalignierten Wörtern: Alignierung zu `NULL`

Lex-Funktion:

$$\texttt{lex}(\bar{e}|\bar{f},\mathbf{a}) = \prod_{i=1}^{|\bar{e}|}\left[\frac{1}{|\{j|(j,i)\in\mathbf{a}\}|}\sum_{j:(j,i)\in\mathbf{a}} p(e_i|f_j)\right]$$

Hier: $p(e_i|f_j)$ meist aus Word Alignments geschätzt (relative Häufigkeiten)

Alternativ: $p(e_i|f_j)$ aus EM-Training, etwa von einzelwortbasierten Modellen.

Features für Übersetzung (2)

Mit lexikalischer Gewichtung: zwei neue Features

$$\lambda_{\texttt{lex-f|e}} \sum_{k=1}^{K} \log\left(\texttt{lex}(\bar{f}_k|\bar{e}_k, \mathbf{a}_k)\right)$$

$$+ \lambda_{\texttt{lex-e|f}} \sum_{k=1}^{K} \log\left(\texttt{lex}(\bar{e}_k|\bar{f}_k, \mathbf{a}_k)\right)$$

Strafterm-Features

- für die Länge des Zielsatzes $\mathbf{e} = e_1^I$: Wortpenalty

  resultierender Featureterm: $\lambda_{\mathtt{wp}}\, I$

  – Sprachmodell bevorzugt kurze Sätze (weniger Faktoren)
  – manche Übersetzungsfeatures können aber längere Ziel- als Quellphrasen bevorzugen
  – je nach Sprachpaar: gutes $\lambda_{\mathtt{wp}}$ kann positiv oder negativ sein.

- für die Anzahl der verwendeten Phrasen: Phrasenpenalty

  resultierender Featureterm: $\lambda_{\mathtt{pp}}\, K$

  – lange Phrasen sind manchmal notwendig, um den Kontext zu erfassen
  – aber: bei langen Phrasen Gefahr des Overfitting

Reordering: Details

Reordering sollte unwahrscheinlicher sein als monotone Übersetzung
$\Rightarrow$ wird nur stattfinden, wenn die Sprachmodell-Wk. dadurch steigt

Phrasenbasierte Modelle: bessere Performance, wenn nur begrenzt Reordering erlaubt ist, z.B. $|\texttt{start}_k - \texttt{end}_{k-1}| \leq 10$.

Teilgrund:
– Ohne Reordering: Dekodierungsproblem in Polynomzeit lösbar
– Mit Reordering: NP-hart (Knight, 1999), meist Approximationen
(aber: Chang & Collins EMNLP 2011)

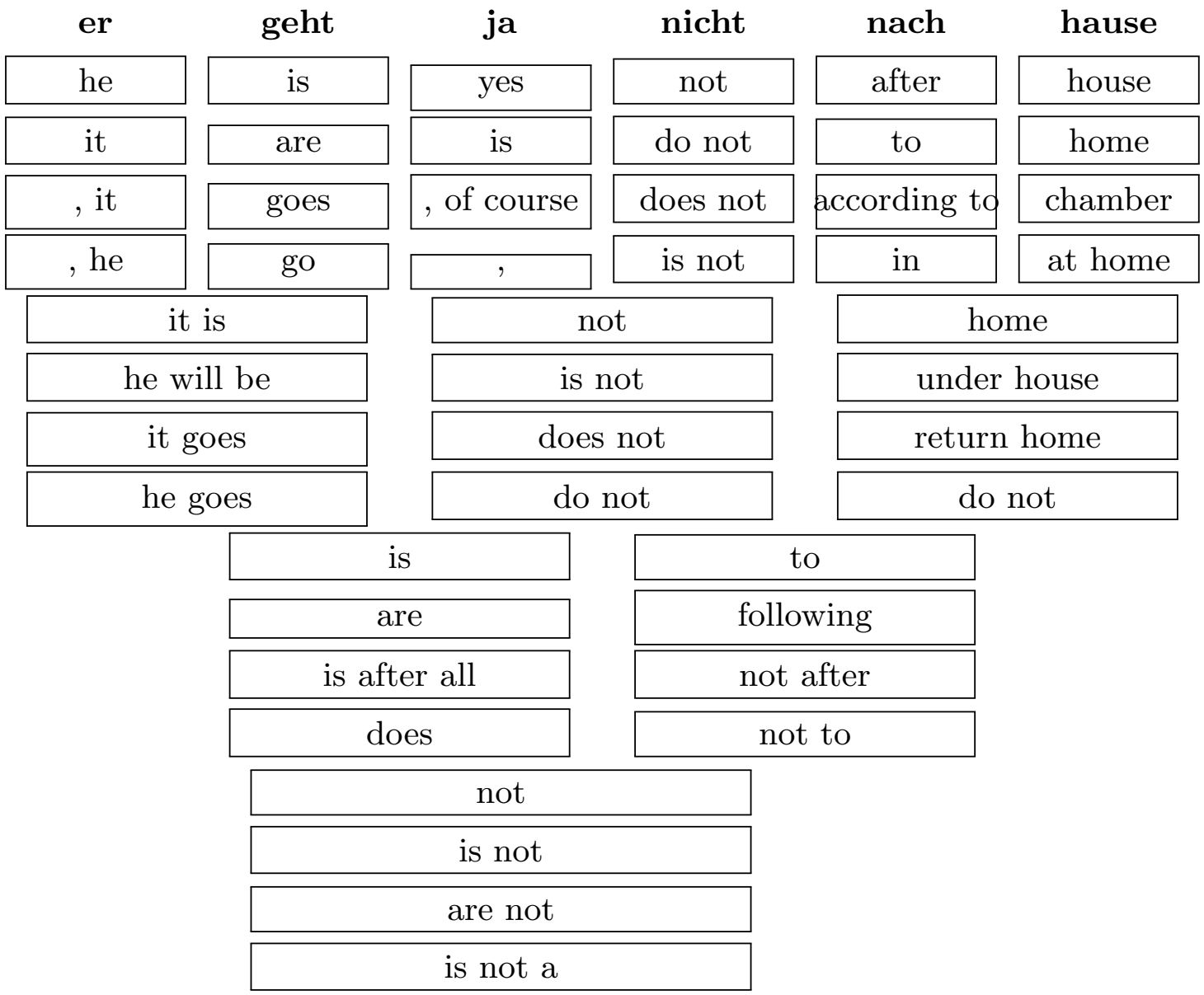
Reordering: Erweiterungen

Statt $p(\texttt{start}_k - \texttt{end}_{k-1})$: kann auch auf das vorherige Phrasenpaar konditionieren

$\Rightarrow p(\texttt{start}_k - \texttt{end}_{k-1} | \bar{f}_k, \bar{e}_k)$

Wegen Datenknappheit: meist nur Unterscheidung zwischen monoton und zwei Klassen von nicht-monoton (swap / diskontinuierlich).

Training: für jede Phrasen-Box $(j_1, \ldots, j_2; i_1, \ldots, i_2)$, überprüfe:
– wenn $(j_1 - 1, i_1 - 1) \in \mathbf{a}$: monoton
– sonst, wenn $(j_2 + 1, i_1 - 1) \in \mathbf{a}$: swap
– sonst diskontinuierlich


er
geht
ja
nicht
nach
hause
he
it
, it
, he
is
are
goes
go
yes
is
, of course
,
not
do not
does not
is not
after
to
according to
in
house
home
chamber
at home
it is
he will be
it goes
he goes
not
is not
does not
do not
home
under house
return home
do not
is
are
is after all
does
to
following
not after
not to
not
is not
are not
is not a

Dekodierung / Übersetzung

Übersetzungsproblem für das Basismodell mit Bigram: finde

$$\arg\max_{I,e_1^I} \max_{(K,\bar{f}_1^K,\bar{e}_1^K,\pi)\in\mathcal{S}[e_1^I,f_1^J]} \prod_{i=1}^{I+1} p(e_i|e_{i-1}) \cdot \prod_{k=1}^{K} \left[p(\bar{f}_k|\bar{e}_k)\, p(\mathtt{start}_k - \mathtt{end}_{k-1})\right]$$

wobei $\mathcal{S}[e_1^I, f_1^J]$ die Menge aller $(K, \bar{f}_1^K, \bar{e}_1^K, \pi)$, die e_1^I ergeben, ist.

Ideal: eigentlich $\sum_{(K,\bar{f}_1^K,\bar{e}_1^K,\pi)\in\mathcal{S}[e_1^I,f_1^J]}$ anstatt $\max_{(K,\bar{f}_1^K,\bar{e}_1^K,\pi)\in\mathcal{S}[e_1^I,f_1^J]}$.
Jedoch nicht effizient handhabbar.

Anmerkung: die log-linearen Modelle lassen sich ganz ähnlich behandeln, da auch sie aus einem Produkt über i und einem Produkt über k bestehen.

Dekodierung: Grundprinzip

Kostenfunktion (wird maximiert):

$$\prod_{i=1}^{I+1} p(e_i|e_{i-1}) \cdot \prod_{k=1}^{K} \left[p(\bar{f}_k|\bar{e}_k)\, p(\texttt{start}_k - \texttt{end}_{k-1}) \right]$$

Grundprinzip:

– baue den Zielsatz von links nach rechts auf

– für alle Zwischenergebnisse, merke welche Quellwörter bereits übersetzt wurden. Jedes Quellwort muss genau einmal übersetzt werden.

– der Zielsatz ist vollständig wenn alle Quellwörter übersetzt wurden.

Aber: wir wollen **alle** möglichen Zielsätze betrachten, und den mit den besten Kosten ermitteln.

Exploration aller Zielsätze (Basisprinzip)

Grundelement: Hypothese (= Zwischenzustand)

Verbundelement (`struct/class`) mit folgenden Einträgen:

– der schon übersetzte Teil des Zielsatzes e_1^i

– pro Quellwort j ein Flag $t_j \in \{\texttt{av.}, \texttt{proc.}\}$, ob schon übersetzt

– $\texttt{start}_k$ und $\texttt{end}_k$ für alle bisherigen Phrasen k

– bisherige (Teil-)Wk. p

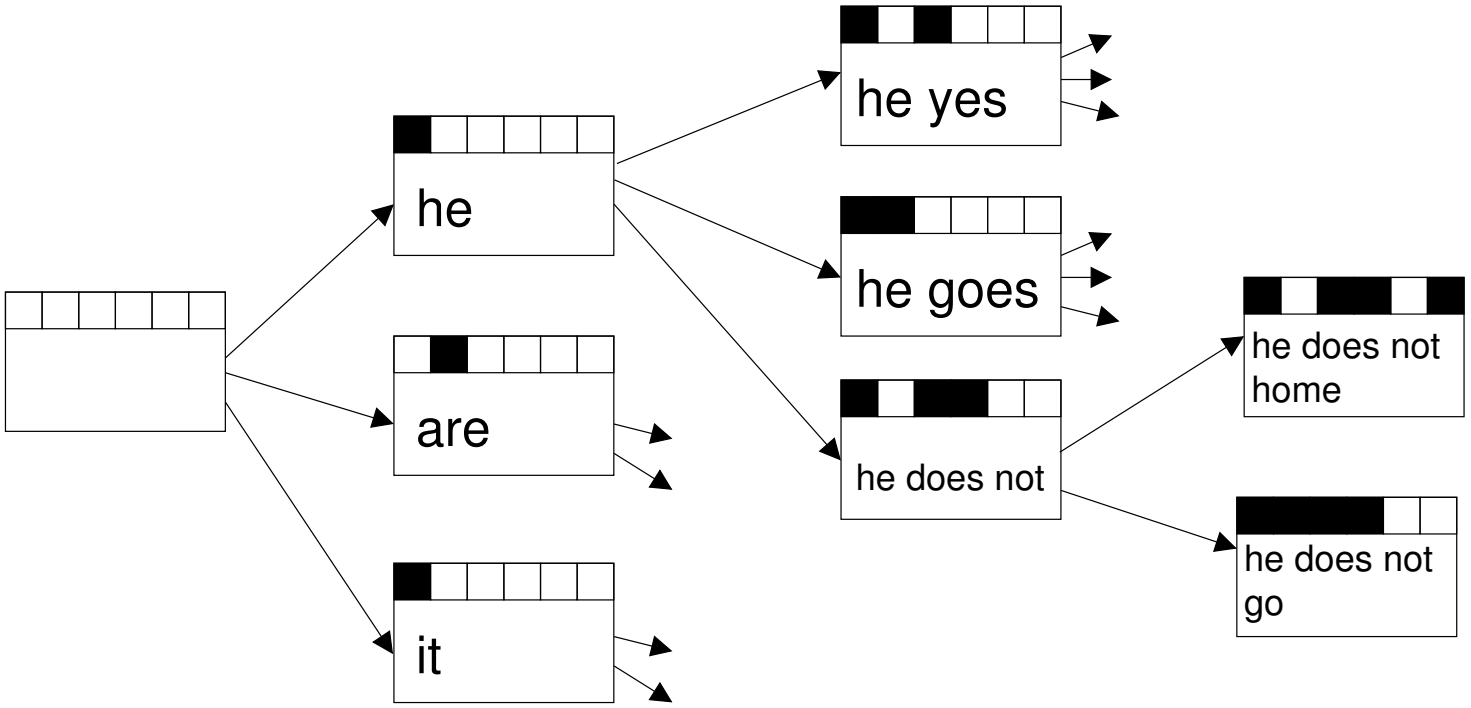
Anmerkung: t_1^J ergibt sich aus der Gesamtheit der $\texttt{start}_k$ und $\texttt{end}_k$, müsste also nicht explizit gespeichert werden. Bei den späteren effizienten Ansätzen wird es aber benötigt.

Exploration aller Zielsätze: Basisalgorithmus

Basisalgorithmus (für Quellsatz f_1^J):

1. Initialisiere Menge der aktiven Hypothesen $\mathcal{H}$ mit "leerer" Hypothese (Zielsatz leer, keine Quellwörter übersetzt, Teil-Wk. $= 1$)
2. Solange $\mathcal{H}$ nicht leer
   (a) entferne ein Element h aus $\mathcal{H}$
   (b) **falls** $h.t_j = \texttt{proc}$. für alle $1 \leq j \leq J$: speichere h in einer Menge $\mathcal{C}$ von vollständigen Zielsätzen
   sonst: für alle j_1, $j_2 \geq j_1$ sodass $h.t_j = \texttt{av}$. $\forall j_1 \leq j \leq j_2$ und für alle $\bar{e}$ mit $p(f_{j_1}^{j_2}|\bar{e}) > 0$:
   – erzeuge neue Hypothese $\bar{h} = h[j_1, j_2, \bar{e}]$ (s. übernächste Folie)
   – füge $\bar{h}$ zu $\mathcal{H}$ hinzu.
   (c) Unter allen vollständigen Sätzen (gespeichert in $\mathcal{C}$) ermittele nun den mit den besten Kosten

Hypothesen und Nachfolger (Basisalgorithmus)



Erweiterung einer Hypothese

$\bar{h} = h[j_1, j_2, \bar{e}]$ ist wie folgt definiert:

- $\bar{h}.e_1^{h.i+|\bar{e}|} = h.e_1^i \circ \bar{e}$
  bisheriger Zielsatz wird um die Phrase $\bar{e}$ erweitert.
- $\bar{h}.t_j = \texttt{proc}.$ für $j_1 \leq j \leq j_2$, sonst $\bar{h}.t_j = h.t_j$.
- wenn K Phrasenpaare in h, so hat $\bar{h}$ genau $K+1$ Paare, mit $\texttt{start}_{K+1} = j_1, \texttt{end}_{K+1} = j_2$ (alle anderen $\texttt{start}_k$ und $\texttt{end}_k$ wie in h).
- $\bar{h}.p = h.p \ \cdot p(f_{j_1}^{j_2}|\bar{e}) \cdot p(j_1 - h.\texttt{end}_K) \cdot \prod_{i'=h.i+1}^{h.i+|\bar{e}|} p(e_{i'}|e_{i'-1})$

Rekombination

Erinnerung: Update der Kosten

$$\bar{h}.p = h.p \cdot p(f_{j_1}^{j_2}|\bar{e}) \cdot p(j_1 - h.\mathtt{end}_K) \cdot \prod_{i'=h.i+1}^{h.i+|\bar{e}|} p(e_{i'}|e_{i'-1})$$

Beobachtung: von der Hypothese h gehen hier lediglich e_i und $\mathtt{end}_K$ ein.
$\Rightarrow$ viele Berechnungen werden sehr oft wiederholt (für unterschiedliche e_1^{i-1} und unterschiedliche Segmentierungen).

$\Downarrow$

Rekombination / Dynamische Programmierung (= DP):
– speichere nur den relevanten Kontext in jeder Hypothese
– aber: jetzt behalte alle Zwischenhypothesen im Speicher
und merke die beste Vorgängerhypothese

Hypothesen für Dynamische Programmierung

Hypothesen für ein Bigram-Sprachmodell:

Verbundelement mit folgenden Einträgen:
– das letzte Wort $\tilde{e}$ des schon übersetzten Teils des Zielsatzes
– pro Quellwort j ein Flag $t_j \in \{\texttt{av.}, \texttt{proc.}\}$, ob schon übersetzt
– Ein Eintrag j für das Ende $\texttt{end}_K$ der letzten bisherigen Phrase K
– bisherige (Teil-)Wk. p (max. über alle eingehenden Hypothesen)
– Zeiger auf beste Vorgängerhypothese

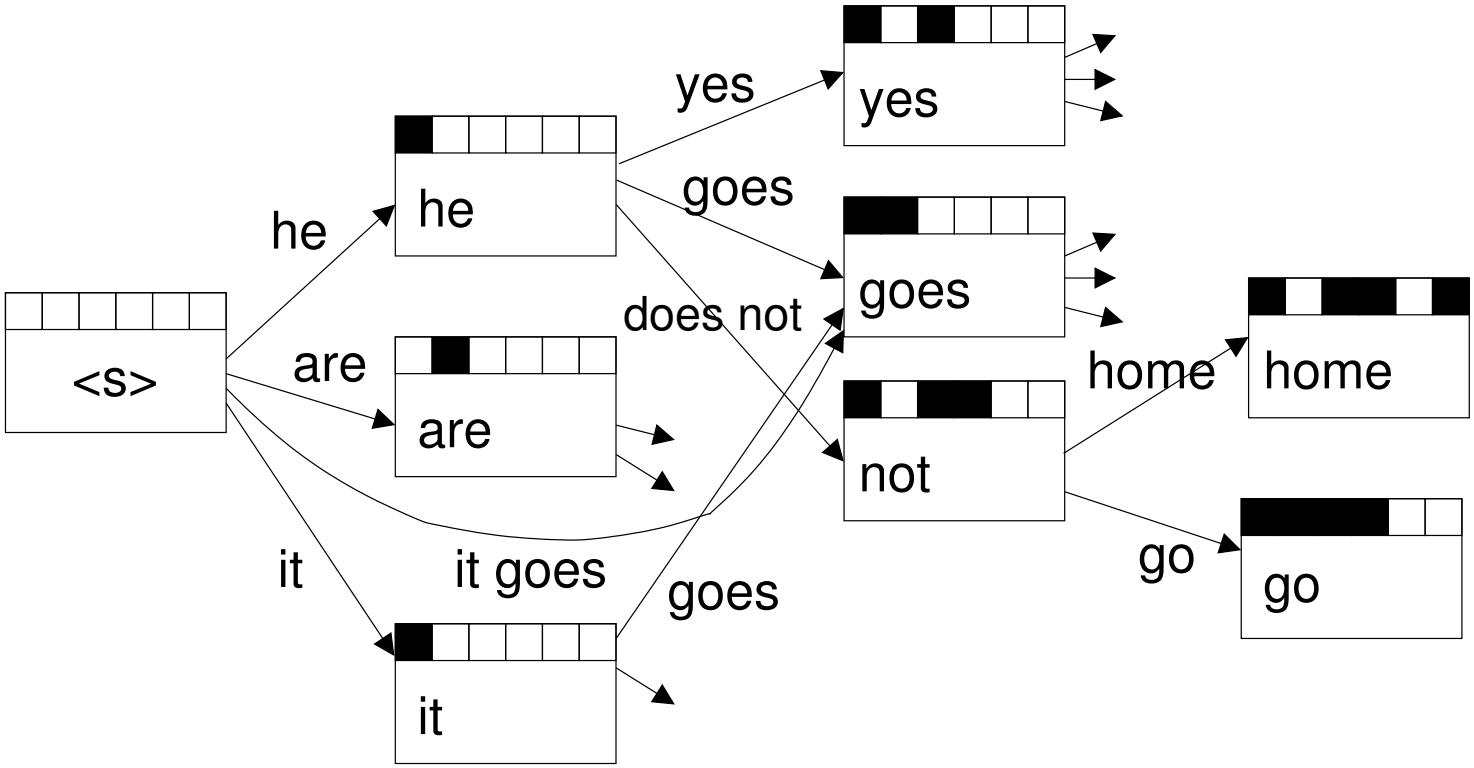
$$\Downarrow$$

jetzt deutlich weniger Zwischenhypothesen

falls monotone Übersetzung: polynomiell viele Hypothesen

mit Reordering: exponentiell viele

Dynamische Programmierung: Zustandsraum



Dynamische Programmierung für Monotone Übersetzung

monotone Kostenfunktion: (ggb. Quellsatz f_1^J)

$$\tilde{p}(e_1^I|f_1^J) = \max_{(K,\bar{f}_1^K,\bar{e}_1^K)\in\mathcal{S}[e_1^I,f_1^J]} \prod_{i=1}^{I} p(e_i|e_{i-1}) \cdot \prod_{k=1}^{K} p(\bar{f}_k|\bar{e}_k)$$

Beachte: jetzt kein explizites π (bzw. $\pi = \mathtt{id}$)

Hilfsgröße:

$$\begin{aligned} Q(j,\tilde{e}) &= \max_{i,e_1^i:e_i=e} \tilde{p}(e_1^i|f_1^j) \\ &= \max\Big\{ \max_{\tilde{e}',j'<j} \Big(Q(j',\tilde{e}') \max_{\bar{e}:\bar{e}_{[|\bar{e}|]}=\tilde{e}} p(f_{j'+1}^j|\bar{e})\, p(\bar{e}_{[1]}|\tilde{e}') \prod_{k=2}^{|\bar{e}|} p(\bar{e}_{[k]}|\bar{e}_{[k-1]}) \Big), \\ &\qquad \max_{\bar{e}:\bar{e}_{[|\bar{e}|]}=\tilde{e}} \Big(p(f_1^j|\bar{e})\, p(\bar{e}_{[1]}|<\mathtt{s}>) \prod_{k=2}^{|\bar{e}|} p(\bar{e}_{[k]}|\bar{e}_{[k-1]}) \Big) \Big\} \end{aligned}$$

wobei $\bar{e}_{[k]}$ das k-te Wort der Phrase $\bar{e}$ ist.

höchste Wk. aller Übersetzungen: $\max_{\tilde{e}} Q(J,\tilde{e}) \cdot p(</\mathtt{s}>|\tilde{e})$.

Dynamische Programmierung mit Reordering

mit Reordering:

Hilfsgröße $Q(t_1^J, j, \tilde{e})$, j = Ende der letzten Phrase.

Im allgemeinen zu viele Zustände (es gibt 2^J Sequenzen t_1^J).

$\Rightarrow$ Ausweg: approximative Suche durch Nicht-Weiterverfolgung einiger Zwischenhypothesen (*Pruning*, später mehr).

Für effiziente dynamische Programmierung:
bearbeite Hypothesen in aufsteigender Ordnung bzgl. der abgedeckten Wörter
$\Rightarrow$ statt einer Menge $\mathcal{H}$, nun Mengen $\mathcal{H}[n]$ (genannt *Stack*),
$n = 0, \ldots J$ von aktiven Hypothesen. n = Anzahl der übersetzten Quellwörter.

Dekodierungsalgorithmus mit Dynamischer Programmierung (für ein Bigram)

1. $\mathcal{H}[0]$ enthält Starthyp. (Wort <s>, alle $t_j = \texttt{av.}, \texttt{end}_0 = 0$, $p = 1$), sonst alle $\mathcal{H}[n]$ leer.
2. **für** n=0,. . . , J-1
   solange $\mathcal{H}[n]$ nicht leer
   - entferne ein h aus $\mathcal{H}[n]$ (aber behalte h im Speicher)
   - für alle j_1, $j_2 \geq j_1$ sodass $h.t_j = \texttt{av.}$ $\forall j_1 \leq j \leq j_2$ und für alle $\bar{e}$ mit $p(f_{j_1}^{j_2}|\bar{e}) > 0$, e sei letztes Wort von $\bar{e}$:
     - ermittele die Nachfolgehyp. $\bar{h} = h[j_1, j_2, \bar{e}]$
     - ermittele $n' = |\{j|\bar{h}.t_j = \texttt{proc.}\}|$ (Nachfolgestack)
     - **falls** schon ein Element für e, j_2 und $\bar{h}.t_1^J$ in $\mathcal{H}[n']$: Update bei geringeren Kosten (incl. Zeiger) **sonst** füge $\bar{h}$ mit entspr. Zeiger zu $\mathcal{H}[n']$ hinzu
     - Optional: Pruning, d.h. reduziere $\mathcal{H}[n']$ auf einige vielversprechende Hypothesen.
3. $\mathcal{H}[J]$ enthält nun alle gefundenen vollständigen Zielsätze.

Ermittlung der Nachfolgerhypothese bei DP mit Bigram

Für DP mit Bigram ist $\bar{h} = h[j_1, j_2, \bar{e}]$ ist wie folgt definiert (hier mit gleichzeitigem Update der Kosten und Vorgängerhyp.):

- $\bar{h}.\tilde{e} = \bar{e}_{[|\bar{e}|]}$
- $\bar{h}.j = j_2$
- $\bar{h}.t_j = \texttt{proc.}$ für $j_1 \leq j \leq j_2$, sonst $\bar{h}.t_j = h.t_j$.
- Kosten und Zeiger auf Vorgängerhypothese:
  ermittele
  $$p = h.p \ \cdot p(f_{j_1}^{j_2}|\bar{e}) \cdot p(j_1 - h.j) \cdot p(\bar{e}_{[1]}|h.\tilde{e}) \prod_{k=2}^{|\bar{e}|} p(\bar{e}_{[k]}|\bar{e}_{[k-1]})$$
  Falls noch keine Hypothese mit gleichem $\bar{h}.\tilde{e}, \bar{h}.j, \bar{h}.t_1^J$ existiert *oder* diese Hypothese eine Wahrscheinlichkeit kleiner als p hat:
  – setze $\bar{h}.p = p$, merke h als Vorgängerhypothese von $\bar{h}$

Dekodierung mit n-gram Sprachmodellen für n > 2

Allgemein für n-gram Sprachmodell:

- Jede Hypothese muss die letzten $n-1$ e-Wörter speichern (d.h. die Historie für das erste Wort einer neuen Phrase).
- Für monotone Übersetzung mit Trigram: Hilfsgröße
  $Q(j, \tilde{e}', \tilde{e}) = \max_{i, e_1^i : e_i = \tilde{e}, e_{i-1} = \tilde{e}'} \tilde{p}(e_1^i | f_1^j)$

  wobei $\tilde{p}(e_1^i | f_1^j)$ jetzt ein Trigram-Sprachmodell beinhaltet.

Pruning (simple)

An mehreren Stellen: Option, $\mathcal{H}[n]$ auf ein paar Hypothesen zu reduzieren.

Basis zunächst: Wk. $h.p$ einer Hypothese.

Definition: $p_{\texttt{max}}(\mathcal{H}[n]) = \max_{h \in \mathcal{H}[n]} h.p$

Threshold Pruning: entferne h wenn $h.p < \alpha\, p_{\texttt{max}}(\mathcal{H}[n])$ $(0 < \alpha < 1)$
– recht zuverlässig, um nahe an das Optimum zu kommen
– Achtung: bei log-linearen Modellen ohne Normalisierung:
logarithmische Scores können positiv und negativ sein $\rightarrow$ exponentieren

Histogram Pruning: behalte die T Hypothesen mit den besten Wk.s
– macht Laufzeiten und Speicherverbrauch vorhersagbar.

In der Praxis: meist Kombination von beiden Techniken

Anmerkung: DP mit Pruning wird als **Beam-Search** bezeichnet.

Pruning mit Restwahrscheinlichkeiten

Problem beim Pruning (bisher):

– unterschiedliche Hypothesen haben unterschiedliche Quellwörter übersetzt.

– unterschiedliche Teile des Quellsatzes sind unterschiedlich leicht (bzgl. der Teilwahrscheinlichkeiten) zu übersetzen.

⇒ bisheriges Pruning wird verstärkt Hypothesen entfernen, die schwierige Teile zuerst übersetzen.

Besser:

– Schätzung der Restwahrscheinlichkeiten.

– Anstatt Vergleich der Teilwk.s $h.p$ nun Vergleich von $h.p \cdot \texttt{rc}(h)$, wobei $\texttt{rc}(h)$ = Restwk.schätzung für h.

Restwahrscheinlichkeitsschätzung

Basis: $h.t_1^J$

Hier nur: Schätzung der Rest-Übersetzungswk.s

Prinzip: eine Schätzung für jede ununterbrochene Lücke in t_1^J.

DP-Algorithmus für die Schätzung der einzelnen Lücken
(generelle Vorberechnung, Ergebnis verwendbar für *alle* $h.t_1^J$):

for $\texttt{len} = 0, \ldots, J-1$

  for $j = 1, \ldots, J - \texttt{len}$

  – $\texttt{rc}_{j,j+\texttt{len}} := \max_{\bar{e}} p_{\texttt{trans}}(f_j^{j+\texttt{len}}|\bar{e})$ % **kann 0 sein**

  – **for** $j' = j+1, \ldots, j+\texttt{len}$

    $\texttt{rc}_{j,j+\texttt{len}} := \max\{\texttt{rc}_{j,j+\texttt{len}}, \texttt{rc}_{j,j'-1} \cdot \texttt{rc}_{j',j+\texttt{len}}\}$

Dekodierung mittels Relaxierung (Chang & Collins 2011)

Das Dekodierungsproblem kann wie folgt geschrieben werden:

– Tupel $\mathbf{y} = (K, \bar{e}_1^K, \texttt{start}_1^K, \texttt{end}_1^K)$

– *zulässige* Tupel: jedes f_j im gegeb. f_1^J genau einmal übersetzt

– Menge der zulässigen Tuple: $\mathcal{Y}$

– Somit:

$$\arg\max_{\mathbf{y} \in \mathcal{Y}} \; p(\mathbf{y})$$

Idee von Chang & Collins: ersetze einige Bedingungen an $\mathbf{y}$ durch schwächere.

– z.B: die $y_k = (\bar{e}_k, \texttt{start}_k, \texttt{end}_k)$ decken zusammen genau J Quellwörter ab (es dürfen aber einige doppelt übersetzt werden, andere gar nicht) [Chang & Collins nutzen eine leicht stärkere Bedingung]

– Menge der nun zulässigen Tupel: $\bar{\mathcal{Y}}$

– Wichtig: wir wollen $\mathcal{Y} \subset \bar{\mathcal{Y}}$

Lagrange Relaxierung

Wichtig: die ursprünglichen Bedingungen werden nicht verworfen, sondern in die Kostenfunktion integriert.

Math. Bedingung, dass $\mathbf{y}$ jedes f_j genau einmal übersetzt:

$$\sum_{k=1}^{K} \mathcal{T}[\mathbf{y}.\mathtt{start}_k \leq j \leq \mathbf{y}.\mathtt{end}_k] = 1 \qquad \forall j$$

wobei $\mathcal{T}[\mathtt{x}] = 1$ wenn der Ausdruck $\mathtt{x}$ wahr ist, sonst 0.

$$\Downarrow$$

entsprechende Kostenfunktion:

$$\sum_j \lambda_j \Big(\sum_{k=1}^{K} \mathcal{T}[\mathbf{y}.\mathtt{start}_k \leq j \leq \mathbf{y}.\mathtt{end}_k] - 1 \Big)$$

mit geeigneten $\lambda_j \in \mathbb{R}$

Reflektion

Ursprünglich:

$$\arg\max_{\mathbf{y}\in\mathcal{Y}} \; p(\mathbf{y}), \quad p(\mathbf{y}) = \prod_{i=1}^{I+1} p(e_i|e_{i-1}) \cdot \prod_{k=1}^{K} \left[p(\bar{f}_k|\bar{e}_k)\, p(\mathtt{start}_k - \mathtt{end}_{k-1}) \right]$$

Jetzt:

$$\arg\max_{\mathbf{y}\in\bar{\mathcal{Y}}} \; \log(p(\mathbf{y})) + \sum_j \lambda_j \Big(\sum_k \mathcal{T}[\mathbf{y}.\mathtt{start}_k \leq j \leq \mathbf{y}.\mathtt{end}_k] \; - \; 1 \Big)$$

Aber wie wählen wir λ_j ?

Beobachtungen:

– falls $\mathbf{y} \in \mathcal{Y}$: Kosten unverändert.

– somit (da $\mathcal{Y} \subset \bar{\mathcal{Y}}$): neuer optimaler Zielfunktionswert $\geq$ alter

$$\Rightarrow \quad \min_{\{\lambda_j\}} \max_{\mathbf{y}\in\bar{\mathcal{Y}}} \; \log(p(\mathbf{y})) + \sum_j \lambda_j \Big(\sum_k \mathcal{T}[\mathbf{y}.\mathtt{start}_k \leq j \leq \mathbf{y}.\mathtt{end}_k] \; - \; 1 \Big)$$

Maximierung über y

Zunächst: Maximierung für gegebene λ_j

Ziel: Lösung von

$$\arg\max_{\mathbf{y}\in\bar{\mathcal{Y}}} \; q(\mathbf{y}) = \log(p(\mathbf{y})) + \sum_j \lambda_j \Big(\sum_k \mathcal{T}[\mathbf{y}.\mathtt{start}_k \leq j \leq \mathbf{y}.\mathtt{end}_k] \; - \; 1 \Big)$$

Mittel (für Bigram): dynamische Programmierung.
– Hilfsgröße (hier nur der Fall $k \geq 2$):

$$\begin{aligned} Q(n,j,\tilde{e}) \; &= \max_{\mathbf{y}:\sum_k(\mathbf{y}.\mathtt{end}_k - \mathbf{y}.\mathtt{start}_k + 1)=n, \mathbf{y}.\mathtt{end}_K = j} q(\mathbf{y}) \\ &= \max_{j'<j, \bar{e}:\bar{e}_{[|\bar{e}|]}=\tilde{e}} \Big(\log(p(f_{j'+1}^{j}|\bar{e})) \; + \sum_{k=2}^{|\bar{e}|} \log(p(\bar{e}_{[k]}|\bar{e}_{[k-1]})) \; + \sum_{j''=j'+1}^{j} \lambda_{j''} \\ &\qquad + \max_{\tilde{e}'} \big[\log(p(\bar{e}_{[1]}|\tilde{e}')) \max_{j''} \log(p(j'+1-j'')) \, Q(n-(j-j'), j'', \tilde{e}') \big] \Big) \end{aligned}$$

Vgl. Beam-Search: Hilfsgröße $Q(t_1^J, j, \tilde{e})$

Minimierung über λ_j

Iteriere:

1. bestimme ein $\hat{\mathbf{y}} \in \arg\max_{\mathbf{y} \in \bar{\mathcal{Y}}} \; q(\mathbf{y}) =$
   $\log(p(\mathbf{y})) + \sum_j \lambda_j \Big(\sum_k \mathcal{T}[\mathbf{y}.\mathtt{start}_k \leq j \leq \mathbf{y}.\mathtt{end}_k] \; - \; 1 \Big)$
   für die bisherigen λ_j
2. Update (i = Nummer der aktuellen Iteration) :

$$\lambda_j := \lambda_j - \frac{\alpha}{i} \Big(\sum_k \mathcal{T}[\hat{\mathbf{y}}.\mathtt{start}_k \leq j \leq \hat{\mathbf{y}}.\mathtt{end}_k] \; - \; 1 \Big) \qquad \forall j$$

Für den mathematischen Hintergrund siehe [Bertsekas, 1999].
Stichwort: Subgradienten

Vergleich zu Beam-Search

Chang & Collins: Zusatzbedingung, dass sich die Quellphrasen für aufeinanderfolgende Zielphrasen nicht überlappen dürfen.

Dann: für 78% der Sätze (beweisbar) das globale Optimum gefunden.

Mit schrittweise stärkeren Relaxierungen: 99.6 %

MOSES Beam-Search: 85%
(natürlich abhängig von den Pruning-Parametern)

N-best Dekodierung

Bisher: Ermittlung genau einer Übersetzung
– ideal: Ermittlung der Übersetzung mit maximaler Wk.
– durch Pruning: hoffentlich wenigstens eine hohe Wk.

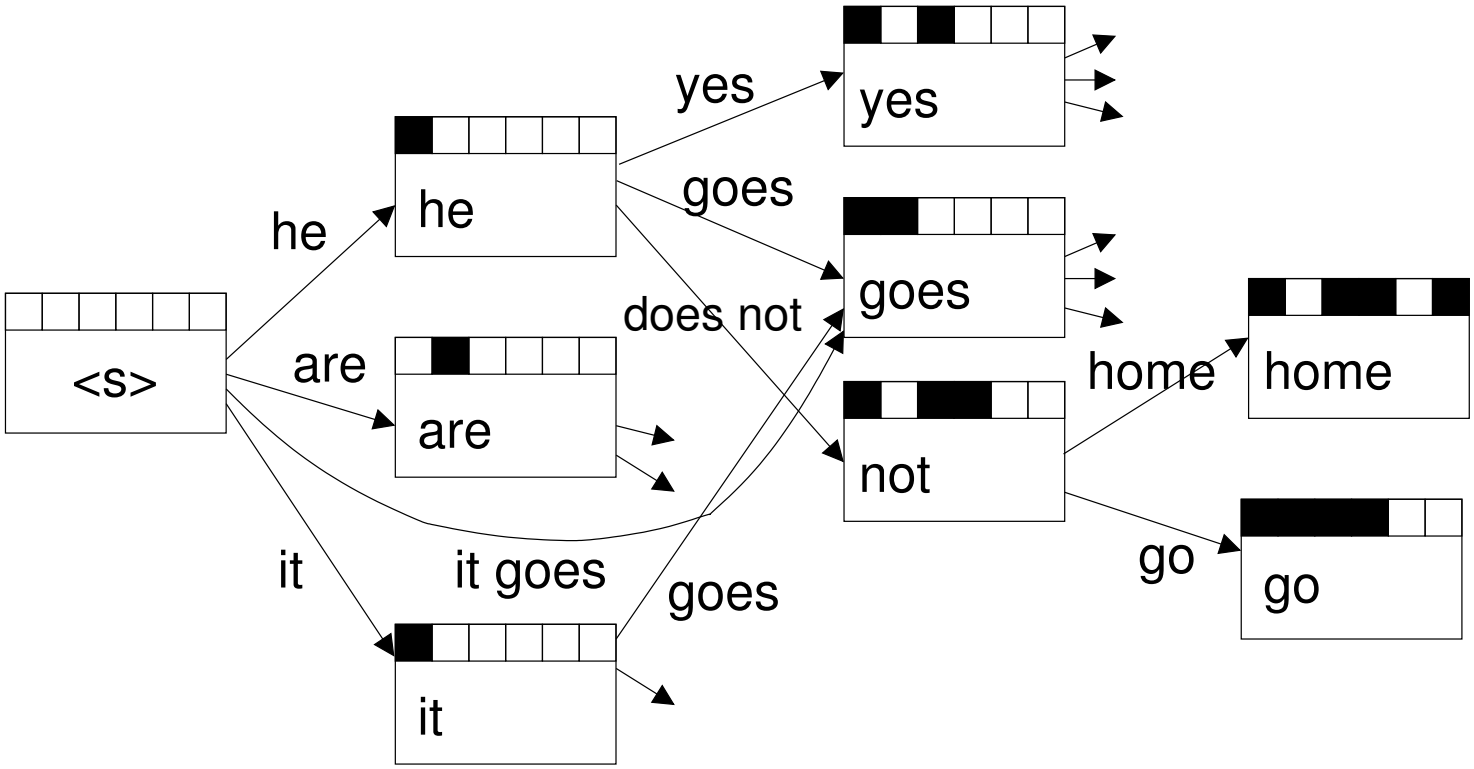
Oft von Interesse:
Ermittlung mehrerer Übersetzungen mit hohen Wk.s
– wichtig in Teil VI (Tuning)
– ideal: Ermittlung der N Übersetzungen mit den höchsten Wks.
– durch Pruning: die N besten der Übersetzungen, die vollendet wurden
– N vorgegeben, für Tuning oft $N = 100$.

Achtung: wir finden tatsächlich N **Paare** von Übersetzungen und Segmentierungen
$\Rightarrow$ manche Übersetzungen werden mehrfach gefunden.

Wortgraphen

Für N-best Dekodierung von phrasenbasierten Modellen:
Zunächst Erstellung von **Wortgraphen** zur Repräsentation aller (nicht geprunten) Hypothesen im Suchraum.

Tatsächlich bereits gesehen:



Wortgraphen (formal)

Ein Wortgraph (in unserem Kontext) enthält als Knoten alle Zustände (t_1^J, j, e) (hier für ein Bigram Sprachmodell), die nicht durch Pruning entfernt wurden.

Kanten: wann immer man von einem Zustand (t_1^J, j, e) mittels eines Phrasenpaares $(j_1, j_2, \bar{e})$ in einem Zustand $(\bar{t}_1^J, j', e')$ gelangt, so gibt es eine Kante $(t_1^J, j, e) \to (\bar{t}_1^J, j', e')$ im Wortgraphen. Diese Kante enthält als Label das Phrasenpaar $(j_1, j_2, \bar{e})$.

Wortgraphen sind gewichtet:
Knotengewichte:$\rho(n)$ = Kosten des besten Pfades, der von der Starthypothese nach n führt.
Kantengewichte:$\rho(l)$ für $l = (t_1^J, j, e) \to (\bar{t}_1^J, j', e')$: Faktor, um den sich die Wk. ändert, wenn eine beliebige Hyp., die in den Zustand (t_1^J, j, e) führt um das Phrasenpaar $(j_1, j_2, \bar{e})$ (Label der Kante) erweitert wird.

Wortgraphen und 1-best Suche

Generell: Wortgraphen werden parallel mit der Suche erstellt.

- Kantengewichte sind direkt durch die Wk.funktion definiert.
- Knotengewichte werden während der Suche ermittelt.
- beste Übersetzung (abzgl. Pruning): Pfad mit maximaler Wk. von der Starthyp. zu einem Endzustand $(\texttt{proc}._1^J, j, e)$ (j, e beliebig)

Wegen Rekombination führen meist mehrere Kanten in einen Knoten.

Die ausgehenden Kanten eines Knotens werden ermittelt, wenn der Knoten während der Suche vom Stack genommen wird.
$\Rightarrow$ Wenn Knoten geprunt werden, haben sie keine ausgehenden Kanten (bleiben aber meist im Graph).

N-best Suche in Wortgraphen: A*-Suche

Basis: Priority-Queue $\mathcal{Q}$ für Elemente (n, Ψ), n Knoten im Wortgraph $\Psi \in I\!R$: Kosten eines (gespeicherten) Pfades von einem Endzustand rückwärts nach n, Priorität von (n, Ψ): $w_{n,\Psi} = (\Psi \cdot \rho(n))$

1. Initialisiere $\mathcal{Q}$ mit allen Endzuständen $((\texttt{proc}._1^J, j, e), 1)$ $\forall j, e$
   (gespeicherte Pfade sind leer.)
2. **Solange** $\mathcal{Q}$ nicht leer:
   - entferne das Element (n, Ψ) mit dem höchsten $w_{n,\Psi}$ aus $\mathcal{Q}$
   - **falls** $n = (\texttt{av}._1^J, 0, <s>)$ (Startzustand):
     gib gespeicherten Pfad aus. Terminiere falls N Pfade gefunden.
   - **sonst**: für alle $l = n' \to n$ im Graphen:
     füge $(n', \Psi \cdot \rho(l))$ zu $\mathcal{Q}$ hinzu (und erweitere den gespeicherten Pfad für das neue Element um l).

Eigenschaften von A*-Suche

A*-Suche

- kann exponentielle Laufzeit haben.
- für generelle Prioritäten $w_{n,\Psi} = \gamma(n) \cdot \Psi$:
  die ausgegebenen Pfade sind (garantiert) die N besten gdw.
  $\gamma(n) \geq \rho(n) \quad \Rightarrow \rho(n)$ ist die optimale Wahl
  (Stichwort: optimistische Restkostenschätzung)
- zum Finden der N besten *unterschiedlichen* Übersetzungen:
  verwerfen schon bekannter Übersetzungen, Abbruch wenn N *unterschiedliche* ausgegeben.

siehe auch: `[Russel & Norvig: Artificial Intelligence - A Modern Approach, Prentice Hall, 2010]`