Dies sind die in der Vorlesung

Sprachverstehen

Summer term 2015

Friedrich-Alexander-Universität Erlangen-Nürnberg

verwendeten Folien. Sie sind ausschließlich für den persönlichen Gebrauch zur Prüfungsvorbereitung bestimmt.

Eine Veröffentlichung, Vervielfältigung oder Weitergabe ist ohne meine schriftliche Zustimmung nicht gestattet.

Weitere Quellen sind die empfohlenen Lehrbücher.

Erlangen, 6. Juli 2015
Elmar Nöth

Sprachverstehen

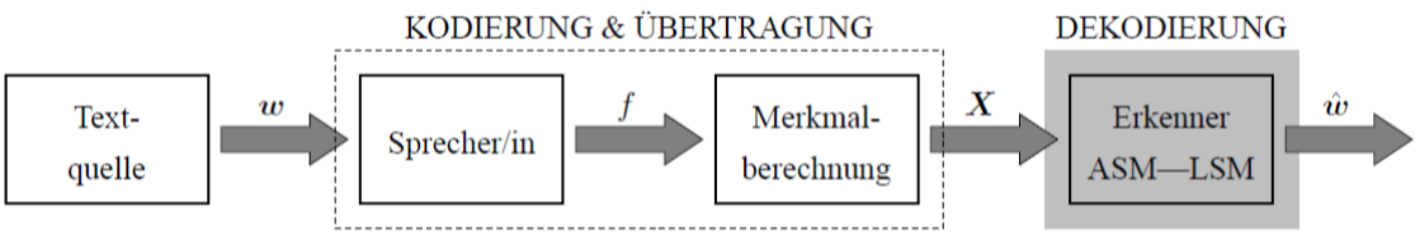
Summer term 2015

Elmar Nöth
Lehrstuhl für Informatik 5
(Mustererkennung)

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Dekodierung

- Dekodierung ist der eigentliche Spracherkennungsvorgang, bei dem mit akustischem und linguistischem Modell nach der gesprochenen Wortkette gesucht wird
- der Begriff kommt aus der Kommunikationstheorie
- optimal: Dekodierung mit der Bayes-Regel
- Problem: in der kontinuierlichen Spracherkennung müssen alle kombinatiorisch möglichen Wortketten und alle möglichen Wortdauern untersucht werden



Dekodierung

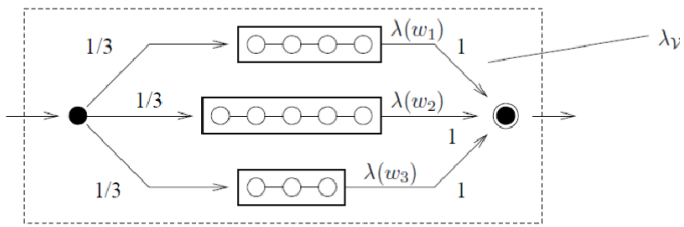
- selbst Einsatz von dynamischer Programmierung (Viterbi-Algorithmus) zur Suche ist bei langen Sprachsignalen und vielen Wörtern immer noch zu speicher- und rechenzeitaufwendig
- Abhilfe: Kombination verschiedener Verfahren zur Reduzierung des Aufwands
  - effiziente Suchalgorithmen wie DP und A*
  - Pruning: bestimmte Alternativen werden nach heuristischen Kriterien nicht berücksichtigt
  - schrittweise Verfeinerung: z.B. erst werden Bigramm-Modelle eingesetzt, dann wird genauer mit Trigrammen gesucht
  - Suchraumverkleinerung: geschickte Kombination unterschiedlicher Alternativen z.B. von Wörtern mit gleichen Lauten am Wortanfang
  - gute Implementierung
- Problem: Durch die suboptimale Dekodierung gibt es jetzt neben Modellierungsfehlern auch noch Dekodierungsfehler

Synchrone Suche

- für die synchrone Suche werden die Wort-HMM zu einem **kompilierten Netzwerk** verknüpft
- dieses Netzwerk ist selbst ein HMM und wird niemals komplett erzeugt sondern nur teilweise, je nachdem welche Wörter und Wortübergänge gerade verarbeitet werden
- **synchrone** Suche: die Suche erfolgt von links nach rechts in der Zeitrichtung
- es handelt sich dabei üblicherweise um Erweiterungen und Varianten des Viterbi-Algorithmus

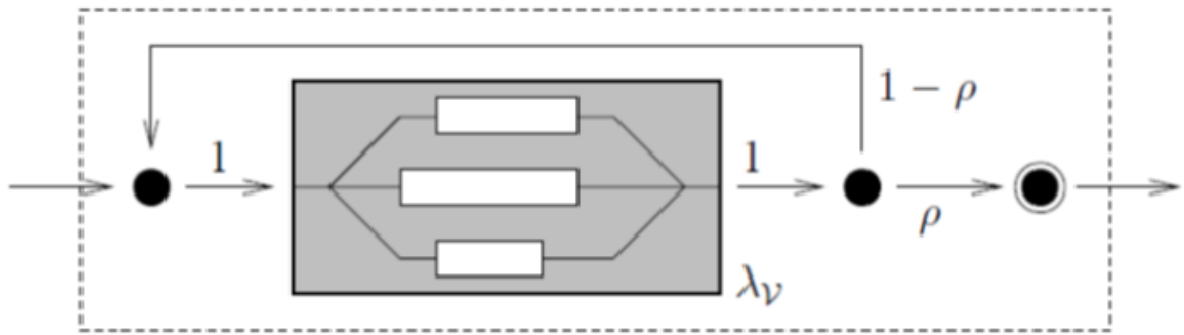
Netzwerke von HMM

- die Wortmodelle $\lambda(w_i)$ für alle Wörter des Erkennungslexikons ($w_i \in \mathcal{V}$) werden zu einem Netzwerk $\lambda_\mathcal{V}$ verknüpft
- Erkennung kann dann z.B. durch den Viterbi-Algorithmus erfolgen: für eine Beobachtung wird die wahrscheinlichste Folge der Wortmodelle im Netzwerk-HMM $\lambda_\mathcal{V}$ gesucht
- die HMM $\lambda(w_i)$ werden durch **konfluente Zustände** verknüpft, das sind Zustände, die keine Ausgabeverteilung haben
- im Suchalgorithmus muss dafür gesorgt werden, dass die konfluenten Zustände keinen Merkmalvektoren zugeordnet werden



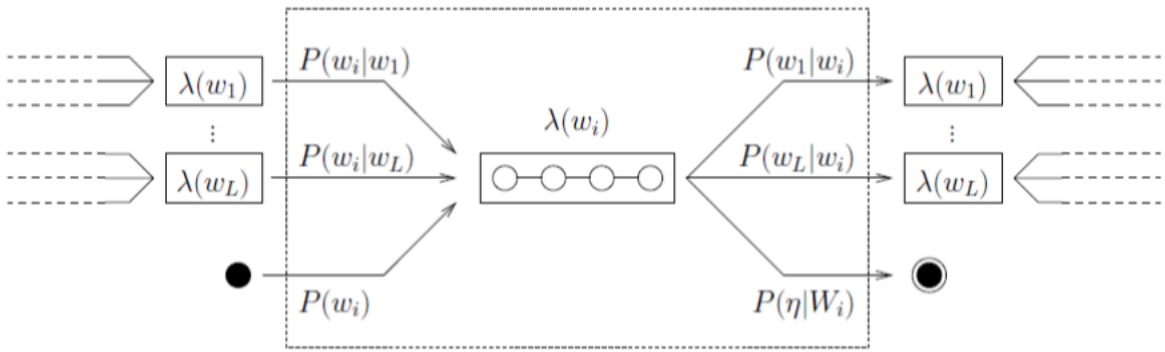
Netzwerke von HMM: Sätze beliebiger Länge

Durch die konfluenten Zustände wird die Komplexität des Netzwerks reduziert, da Wortendezustände zusammengeführt werden:



Netzwerke von HMM: Sätze beliebiger Länge

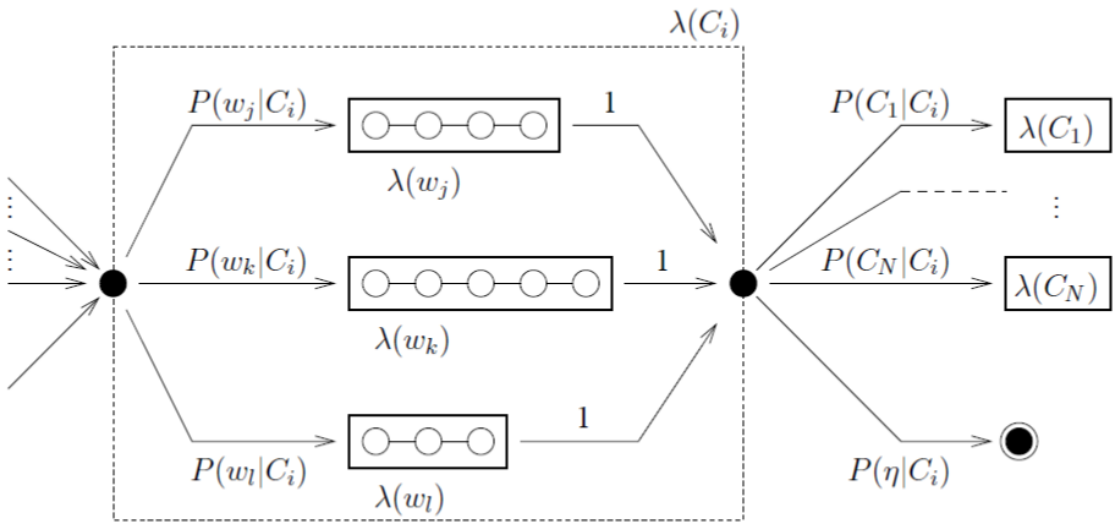
Bei einem Bigramm-Netz müssen alle Wort-Wort Übergänge erzeugt werden:



Wörter einer Kategorie können jedoch in Unternetzwerken zusammengefasst werden $\rightarrow$ Komplexitätsreduktion

Netzwerke von HMM: Kategoriebigramme

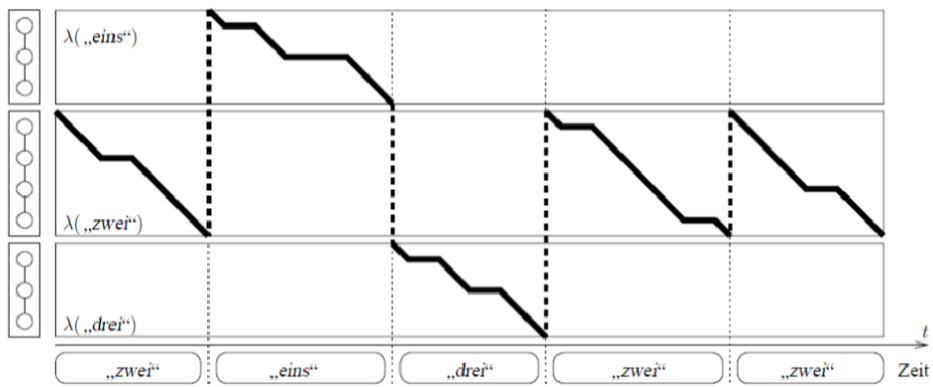
Unternetze für jede Kategorie



Zeitsynchrone Suche: Viterbi-Algorithmus

- einfachstes Verfahren zur zeitsynchronen Dekodierung
- bestimme wahrscheinlichste Zustandsfolge $\boldsymbol{q}^*$ für die Beobachtung $\boldsymbol{X}$
- Erkennungsergebnis: Wortfolge $\boldsymbol{w}^* = w(\boldsymbol{q}^*)$

Optimale Zustandsfolge für eine Äusserung: Sprünge an beliebige Folgewörter an den Wortendeknoten, nicht innerhalb von Wörtern



Zeitsynchrone Suche: Vorwärts-Dekodierung

- Erweiterung des Viterbi-Algorithmus: innerhalb eines Wortes wird nicht die beste Zustandsfolge gesucht, sondern die Vorwärtswahrscheinlichkeit berechnet
- d.h. Maximierung nur an Wortübergängen, Summation im Wortinneren

Zeitsynchrone Suche: Vorwärts-Dekodierung

1. Initialisierung: $\vartheta_1(j) = \pi_j b_j(\boldsymbol{x}_1)$, $\psi_1(j) = 0$ für alle $j = 1, \ldots, N$
2. Iteriere für $t = 0..T-1$ und alle Zustände $j = 1, \ldots, N$ setze

$$\psi_t(j) = \operatorname*{argmax}_i \vartheta_{t-1}(i) a_{ij}$$

$$\vartheta_t(j) = \begin{cases} \max\limits_i & (\vartheta_{t-1}(i) a_{ij}) \cdot b_j(\boldsymbol{x}_t) \quad \text{falls } s_j \text{ Wortanfangszustand ist} \\ \sum\limits_i & (\vartheta_{t-1}(i) a_{ij}) \cdot b_j(\boldsymbol{x}_t) \quad \text{für alle sonstigen } s_j \end{cases}$$

3. Terminierung: Setze $P^*(\boldsymbol{X} \mid \boldsymbol{\lambda}) = \vartheta_T(N)$ und $q_T^* = \vartheta_T(N)$
4. Rückverfolgung der optimalen Wortkette: Für $t = t-1, \ldots, 1$ setze $q_t^* = \psi_{t+1}(q_{t+1}^*)$
5. Lösungswortkette: $\boldsymbol{w}^* = w(\boldsymbol{q}^*)$

Zeitsynchrone Suche: Beam Search

- Problem: Bei der Viterbi- oder der Vorwärtssuche müssen zu viele Pfade berücksichtigt werden $\rightarrow$ langsam
- für $t = 0..T-1$ und alle Zustände j

$$\vartheta_t(j) = \max_i(\vartheta_{t-1}(i)a_{ij}) \cdot b_j(\boldsymbol{x}_t)$$

- Abhilfe: aussichtslose Alternativen werden mit einem heuristischen Verfahren so früh wie möglich gelöscht, nur aussichtsreiche Alternativen bleiben in dem **Suchstrahl** $\mathcal{O}_{t-1}$

$$\vartheta_t(j) = \max_{i \in \mathcal{O}_{t-1}} (\vartheta_{t-1}(i)a_{ij}) \cdot b_j(\boldsymbol{x}_t)$$

- dabei kann es natürlich passieren, dass ein Pfad, der am Anfang schlecht bewertet ist, sich aber später als optimal herausstellt, eliminiert wird
  $\rightarrow$ beste Lösung geht verloren $\rightarrow$ Erkennungsfehler

Zeitsynchrone Suche: Beam Search

- Problem: je mehr Alternativen untersucht werden, desto unwahrscheinlicher ist der Fall, dass ein optimaler Pfad fälschlicherweise gelöscht wird, aber desto langsamer ist die Suche
- benötigt wird ein Kompromiss zwischen Genauigkeit und Effizienz
- schlechte Kompromisse sind konstante Schwellwerte bezüglich der Menge der aktiven Zustände oder der minimalen Bewertung der Zustände

Zeitsynchrone Suche: Beam Search

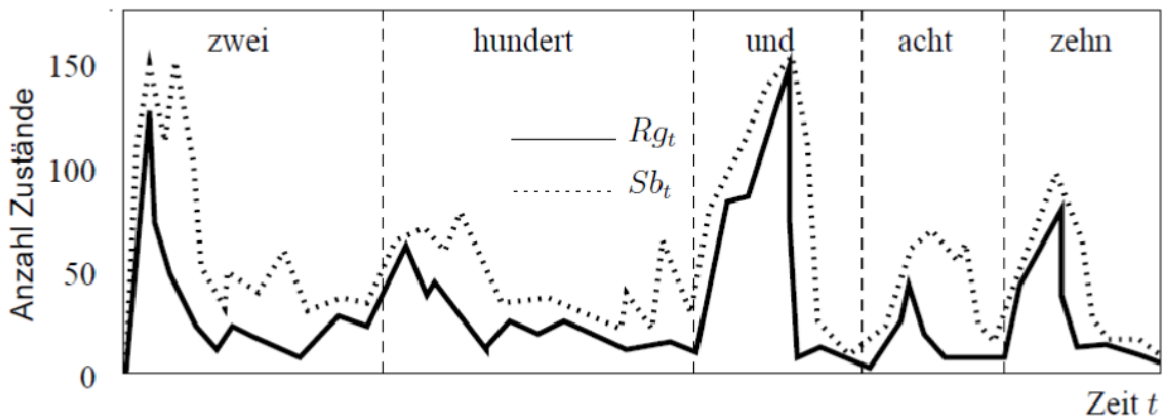
- Beam Search: zeitabhängiger Wahrscheinlichkeitsschwellwert bezogen auf die Bewertung des aktuell besten Pfades:

$$\mathcal{O}_t = \{i | \vartheta_t(i) \geq B_0 \cdot \Lambda_t\} \quad \text{mit} \quad \Lambda_t = \max_k \vartheta_t(k)$$

- B_0 ist sehr kleine pos. Zahl (anwendungsabhängig!), z.B. $B_0 = 1e^{-22}$, je kleiner B_0 gewählt wird, desto langsamer (und besser) ist der resultierende Erkenner
- ist die Bewertung des besten Zustands im Suchstrahl gut, wird der Strahl automatisch enger (weniger Zustände darin)
- ist die Bewertung des besten Zustands im Suchstrahl schlecht, werden mehr Zustände in die Suche aufgenommen

Beam Search: Funktionsweise

- typisch: die Strahlbreite variiert sehr stark und ist typischerweise an Wortanfängen sehr gross
- Kenngrößen:
  - **Rang** Rb_t des Zustandes q_t^* der global besten Folge
  - **Strahlbreite** Sb_t: Anzahl verbleibender $\vartheta_t(j)$–Berechnungen
- Ideal: Breite Sb_t eng an Rangkurve ohne diese zu unterschreiten



Beam Search: Effiziente Realisierung

- Effizienzproblem: Maximumbestimmung kann erst erfolgen, wenn alle Bewertungen bekannt
- Abhilfe: Verwendung des partiellen Maximums $\tilde{\Lambda}_t \leq \Lambda_t$, das während der Berechnung der Bewertungen ständig neu ermittelt wird
  $\rightarrow$ Strahl ist geringfügig weiter als er sein müsste

Beam Search: Effiziente Realisierung

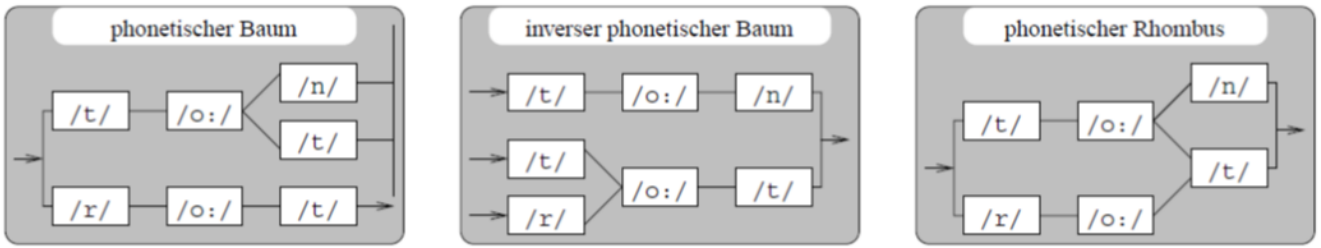
1. Initialisierung: Setze $\mathcal{O}_0 = \{1\}$ und für alle t: $\tilde{\Lambda}_t = 0$
2. Iteration: Für alle $t = 0, \ldots, T-1$ und für alle $i \in \mathcal{O}_t$ (in Listensortierung):
   ($\star$) Für alle j mit $a_{ij} > 0$:
   - $\vartheta = \vartheta_t(i) \cdot a_{ij} \cdot b_j(\boldsymbol{x}_{t+1})$
   - Wenn $\vartheta < B_0 \cdot \tilde{\Lambda}_{t+1} \rightarrow$ weiter bei ($\star$)
   - Wenn $j \in \mathcal{O}_{t+1}$ und $\vartheta \leq \vartheta_{t+1}(j) \rightarrow$ weiter bei ($\star$)
   - $\vartheta_{t+1}(j) = \vartheta$ und $\psi_{t+1}(j) = i$
   - Wenn $j \notin \mathcal{O}_{t+1}$ dann füge j am Kopf (falls $\vartheta > \tilde{\Lambda}_{t+1}$) bzw. am Ende der Liste $\mathcal{O}_{t+1}$ an
3. Terminierung und Rückverfolgung: Wie beim Viterbi-Algorithmus

Wortschatzorganisation

- Ziel: Beschleunigung der Suche durch die Vermeidung unnötiger Mehrfachberechnungen
- dazu wird die Anzahl der Zustände im kompilierten HMM-Netzwerk reduziert
- Beobachtung: **Präfixäquivalenz**, d.h. viele Wörter haben die gleichen Anfänge, z.B. ‘Ton’ (`/to:n/`) und ‘Tot’ (`/to:t/`)
- wenn die Wortanfänge im HMM-Netzwerk nur durch gemeinsame Knoten repräsentiert sind, dann müssen die HMM für die Laute `/to:/` nur einmal evaluiert werden
- **Phonetischer Baum**: jeder Pfad steht für die phonetische Grundform eines Wortes, an den Blättern stehen dann die akustischen Bewertungen der Wörter
- auch für kontextabhängige Modelle verallgemeinerbar

Phonetischer Baum

Strukturen des kompilierten HMM-Netzwerks:

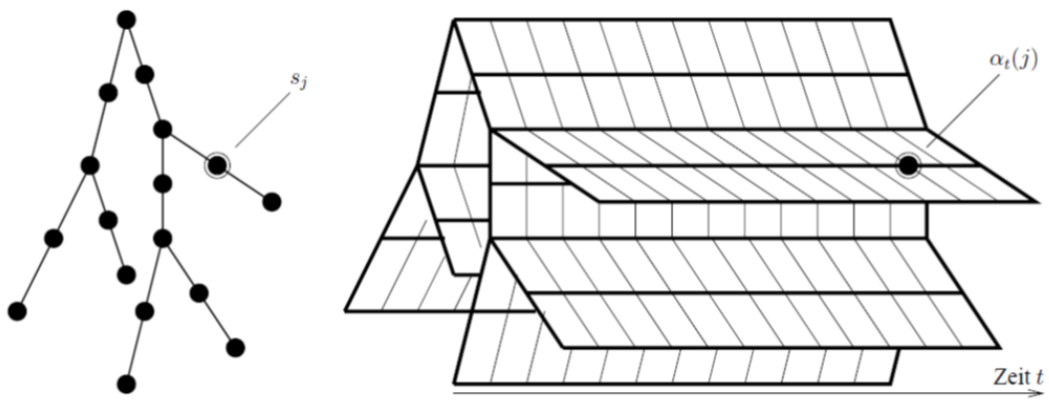


weitere Strukturen neben dem phonetischer Baum sind wenig verbreitet:

- inverser phonetischer Baum: Ausnutzung von Postfixäquivalenzen am Wortende
- phonetischer Rhombus: Ausnutzung von Prä- und Postfixäquivalenzen

Phonetischer Baum: Einzelworterkennung

- Baumknoten werden in Lautmodelle expandiert
- Matrix der Vorwärts- oder Viterbibewertungen ist ein ‘Baumflächler’
- bei großen Vokabularen erfolgt eine erhebliche Einschränkung des Suchraums



Phonetischer Baum: Erkennung kont. Sprache

- Problem: die Bewertungen an den Knoten und Blättern des phonetischen Baums enthalten auch die Sprachmodellbewertung
- die Sprachmodellbewertung hängt vom Vorgängerwort ab
- für jedes Vorgängerwort muss eine Kopie des Phonetischen Baums erzeugt werden $\rightarrow$ aufwendig
- bei einem Bigramm werden so $|\mathcal{V}|$ Kopien benötigt
- die Anzahl der Kopien reduziert sich, sobald Pfade aus dem Suchstrahl entfernt werden
- durch bestimmte Heuristiken ist es so möglich im Mittel mit nur zwei Instanzen des phonetischen Baumes pro Zeittakt auszukommen

Kombination von $P(\boldsymbol{X} \mid \boldsymbol{W})$ und $P(\boldsymbol{W})$

- Theoretisch optimal gemäß Bayes-Regel: Kombination der Wahrscheinlichkeiten von akustischen Modell und Sprachmodell durch Multiplikation
- Praxis zeigt jedoch, dass die geschätzten Werte von $P(\boldsymbol{X} \mid \boldsymbol{W})$ und $P(\boldsymbol{W})$ grobe Abweichungen von der Realität aufweisen
  - n-Gramme berücksichtigen nur einen Teil des Kontexts
  - Merkmalvektoren x_t und x_{t-1} sind nicht unabhängig
- die Abweichungen bewirken Verfälschung des Erkennungsergebnisses
- heuristische Abhilfe: Wortstrafparameter (insertion penalty) und Sprachmodellgewicht (language (model) weight oder linguistic matching factor oder linguistic weight)

Insertion Penalty

- das Sprachmodell $P(\boldsymbol{W})$ hat auch die Funktion, die Einfügung neuer Wörter zu bestrafen
- ist z.B. das Sprachmodell uniform, d.h. $P(w_i) = \frac{1}{L}$, dann wird durch das Sprachmodell lediglich dafür gesorgt, dass ein Pfad, bei dem ein neues Wort eingefügt wird, etwas schlechter bewertet ist als ein anderer Pfad ohne das neue Wort
- wenn die Strafe für die Einfügung eines Worts gross ist, wird der Erkenner dazu tendieren, weniger und dafür längere Wörter zu generieren
- ist die Strafe klein, wird der Erkenner viele kurze Wörter bevorzugen
- der Wortstrafparameter $0 < \rho \leq 1$ ist ein Gewicht, mit dem die Sprachmodellbewertung multipliziert wird, so kann man heuristisch zwischen den beiden Fällen ausbalancieren
- $\hat{P}(w_i|w_{i-n+1}, ..w_{i-1}) = P(w_i|w_{i-n+1}, ..w_{i-1}) \cdot \rho$
- $\hat{P}(w_1, w_2, ...w_N) = P(w_1, ...w_N) \cdot \rho^N$

Insertion Penalty: Beispiel

Parkettierung mit vielen kurzen, akustisch passenden Wörtern

Uhr in ja am ja am den morgen um am ja ein Zug nach ja am es Uhr

$(\rho = 1)$

Uhr dem fahren wann wie morgen vormittag ein Zug nach am Uhr

$(\rho = 10^{-2})$

Uhr den fahren wann geht morgen vormittag ein Zug nach Frankfurt

$(\rho = 10^{-4})$

guten Tag wann geht morgen vormittag ein Zug nach Frankfurt

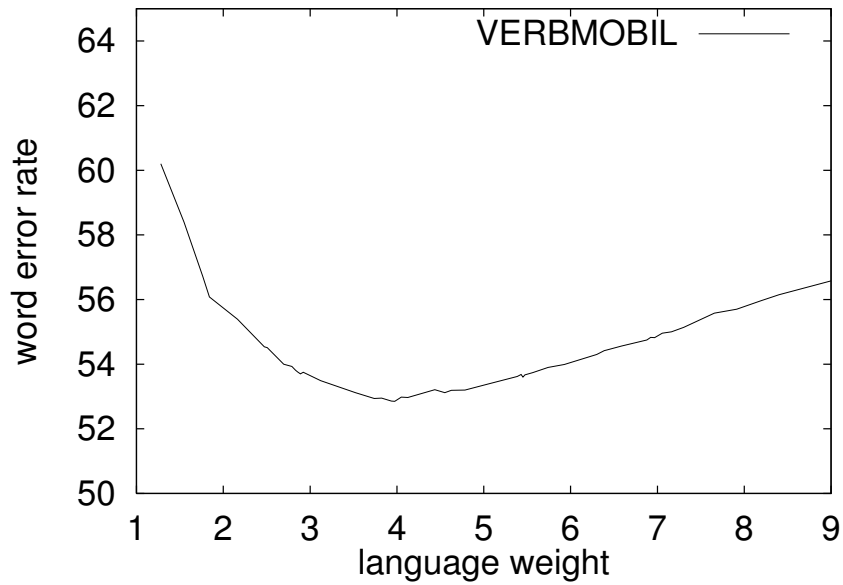
$(\rho = 10^{-6})$

Language Model Weight

- durch HMMs werden die akustischen Dichtewerte/Wahrscheinlichkeiten unterschätzt, d.h. typischerweise ist die akustische Bewertung um Größenordnungen kleiner als die Sprachmodellbewertung
- zusätzlich ist die Schwankungsbreite, d.h. der Wertebereich der akustischen und der Sprachmodellbewertung unterschiedlich
- Ausbalancierung der Bewertungen kann mit dem Language (Model) Weight *LW* erfolgen
- $\hat{P}(w_i|w_{i-n+1},..w_{i-1}) = P(w_i|w_{i-n+1},..w_{i-1})^{LW} \cdot \rho$
- *LW* ist anwendungsabhängig und typischerweise > 1 z.B. 4 oder 7
- da $P(w_i|w_{i-n+1},..w_{i-1}) << 1$ wird durch ein $LW > 1$ die Sprachmodellbewertung verkleinert und so an die akustischen Bewertungen angepasst
- es gibt natürlich Wechselwirkungen zwischen ρ und *LW*, beide müssen gemeinsam auf einer Validierungsstichprobe optimiert werden

Language Model Weight: Beispiel

Evaluation der Erkennungsleistung (große Word Error Rate (WER) → schlechter Erkenner) in Abhängigkeit des Language Weight:

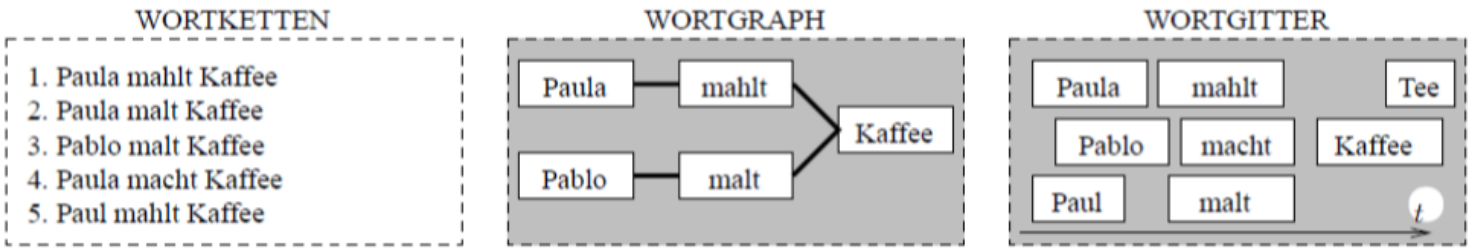


Schrittweise Verfeinerung: Mehrphasendekodierung

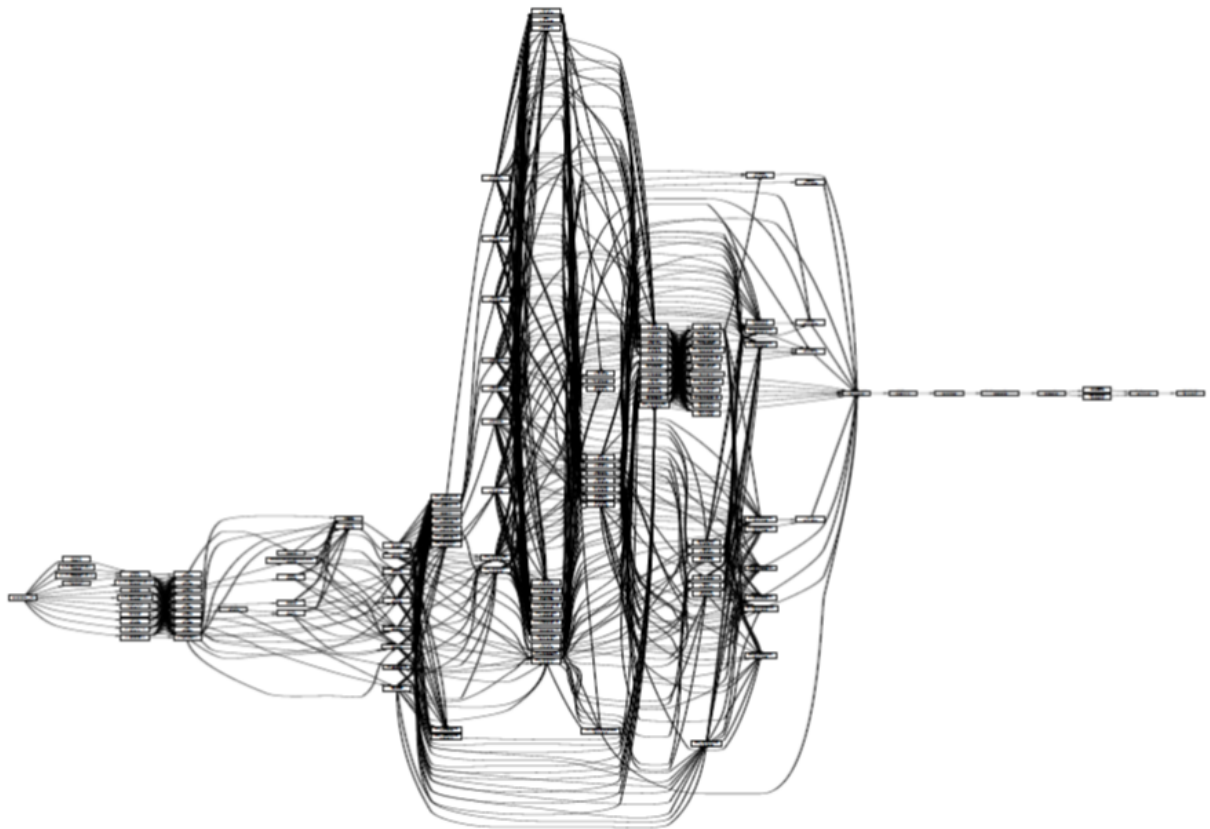
- in der Literatur auch als two-pass oder multi-pass decoding bekannt
- die Komplexität der Strahlsuche steigt bei komplexeren Informationsquellen, z.B. Trigrammen, 4-Grammen stark an
- Abhilfe: ein grober Erkennungslauf mit Bigrammen generiert eine Menge von Wort- oder Satzhypothesen
- die einfachste Möglichkeit sind dabei die sogenannten n-best-Listen, d.h. eine mit den Bewertungen sortierte Liste der N bestbewerteten Wortketten
- in einem Wortgitter oder word lattice sind verschiedene Worthypothesen mit ihren Anfangs- und Endzeiten eingetragen
- ein Wortgraph stellt die Worthypothesen als Knoten in einem Graphen dar
- Erzeugung eines Wortgitters oder n-best Liste z.B. durch eine Erweiterung des Viterbi-Algorithmus (zustands- oder satzbezogener NVA) : Statt des besten Vorgängers werden die k besten Vorgänger gespeichert

Schrittweise Verfeinerung: Mehrphasendekodierung

Beispiele für unterschiedliche Repräsentationsmöglichkeiten der Ergebnisse des ersten Erkennungslaufs:

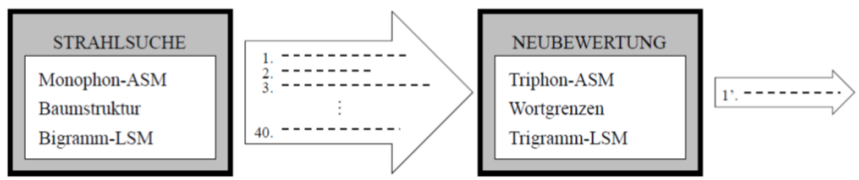


Wortgraph



Schrittweise Verfeinerung: Mehrphasendekodierung

- in einem zweiten Erkennungslauf werden dann die Satzketten in der n-best Liste mit Tri- oder 4-Grammen neu bewertet,
  z.B. Byblos-Erkenner (ca. 1992):



- Alternative: der Wortgraph wird mit dem A*-Algorithmus durchsucht (asynchrone Suche)
- Kosten werden mit akustischen Bewertungen und Trigramm Sprachmodellen berechnet
- die Restkosten können mit Bigramm-Bewertungen abgeschätzt werden
- diese Abschätzung ist aber im allgemeinen **nicht** optimistisch!