Rheinisch Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik VI
Prof. Dr.-Ing. Hermann Ney

Seminar "Speech Recognition and Language Processing" im SS 2003

Multi-Pass Search

Arne Theres

Mat.-Nr. 221 931

30. 7. 2003

Betreuer: Stephan Kanthak

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Problem der maschinellen Erkennung gesprochener Sprache ist wie folgt definiert: ein digitalisiertes Sprachsignal soll mit mit einem Erkennungsalgorithmus in die entsprechende schriftliche Form übersetzt werden, wie in Abb. 1 dargestellt.

Der Erkenner steht dabei vor dem Problem, aus der schier unendlichen Menge der möglichen Wortsequenzen die korrekte Sequenz herauszufinden. Der damit verbundene Suchvorgang ist insbesondere bei sprecherunabhängiger Erkennung und großem Vokabular sehr rechenintensiv.

Erst seit kurzem lässt sich Spracherkennung in Realzeit auf handelsüblichen PC-Plattformen realisieren. Dieser Fortschritt ist nicht nur der zunehmenden Rechenleistung der PCs zu verdanken, sondern auch algorithmischen Verbesserungen, unter anderem der hier vorgestellten Multi-Pass-Suche.

Die Idee der Multi-Pass-Suche besteht in einer Aufteilung des gesamten Suchproblems in mehrere hintereinander ausgeführte einfachere Suchvorgänge, wie in Abb. 2 dargestellt.

Bei einem 3-Pass-Verfahren könnte die Aufteilung wie folgt aussehen:

- Der erste Pass trifft mit einem groben, aber effizienten Sprachmodell eine Vorauswahl der wahrscheinlichsten Wort-Hypothesen.

- Der zweite Pass untersucht mit einem detaillierten, aber in puncto Rechenzeit und Speicherverbrauch teuren Sprachmodell diese Vorauswahl genauer. Nur noch eine sehr kleine Zahl möglicher Satzhypothesen wird zum dritten Pass hindurch gelassen.

- Im dritten Pass wird aus den übriggebliebenen Hypothesen mit einem noch aufwändigeren Modell der auszugebende Satz ermittelt.

Man könnte diese Verfahren mit einem mehrstufigen Filtersystem vergleichen, wie in Abb. 3 dargestellt.

Der Vorteil dieser Vorgehensweise ist eine Reduktion der Gesamtkosten der Suche im Vergleich zu einem 1-Pass-System.

Als positiver Nebeneffekt ergibt sich bei der Multi-Pass-Suche außerdem eine bessere Strukturierung des Suchproblems.

Abbildung 1: Black-Box-Modell eines Spracherkenners

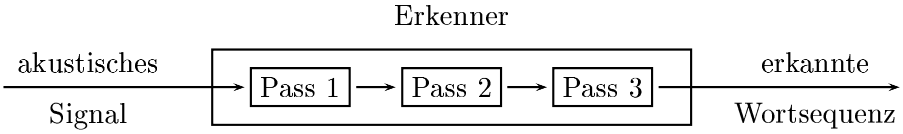


Abbildung 2: Organisation der Multi-Pass-Suche

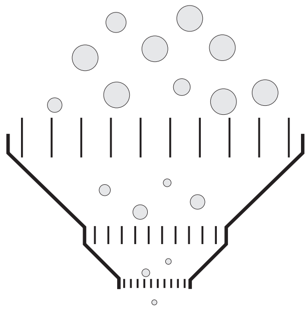


Abbildung 3: Darstellung der Multi-Pass-Suche als mehrstufiges Filtersystem

2 Grundkonzepte statistischer Spracherkennung

2.1 Bayes'sche Entscheidungsregel

Der Grundgedanke eines statistischen Erkenners ist es, den möglichen Wortsequenzen $w_1^N = w_1 \ldots w_N$ eine Wahrscheinlichkeit bezüglich des eingehenden, aufbereiteten Sprachsignals $x_1^T = x_1 \ldots x_T$ zuzuordnen. Unter allen Wortsequenzen gilt es diejenige herauszufinden, deren Wahrscheinlichkeit $Pr(w_1^N|x_1^T)$ maximal ist. Diese sogenannte a-posteriori-Wahrscheinlichkeit lässt sich wie folgt umschreiben:

$$Pr(w_1^N|x_1^T) \;=\; \frac{Pr(w_1^N, x_1^T)}{Pr(x_1^T)} \;=\; \frac{Pr(w_1^T) \cdot Pr(x_1^T|w_1^N)}{Pr(x_1^T)} \tag{1}$$

Die Wahrscheinlichkeit $Pr(x_1^T)$ ist dabei für alle Worthypothesen gleich und somit für einen relativen Vergleich der Hypothesen untereinander irrelevant. Deshalb erhält man:

$$\arg\max_{w_1^N} Pr(w_1^N|x_1^T) \;=\; \arg\max_{w_1^N} \{Pr(w_1^N) \cdot Pr(x_1^T|w_1^n)\} \tag{2}$$

Diese Formel wird als die Bayes'sche Entscheidungsregel bezeichnet und ist von zentraler Bedeutung für die gesamte statistische Mustererkennung. Die Wahrscheinlichkeit $Pr(w_1^N)$ wird als a-priori-Wahrscheinlichkeit bezeichnet, $Pr(x_1^T|w_1^N)$ als klassenbedingte Wahrscheinlichkeit.

Man beachte, dass es sich bei der klassenbedingten Wahrscheinlichkeit in einem Spracherkenner um eine Wahrscheinlichkeitsdichte handelt. Somit wendet man auf der rechten Seite der Gleichung den *argmax*-Operator im Gegensatz zu der linken Gleichungsseite auf eine Wahrscheinlichkeitsdichte an.

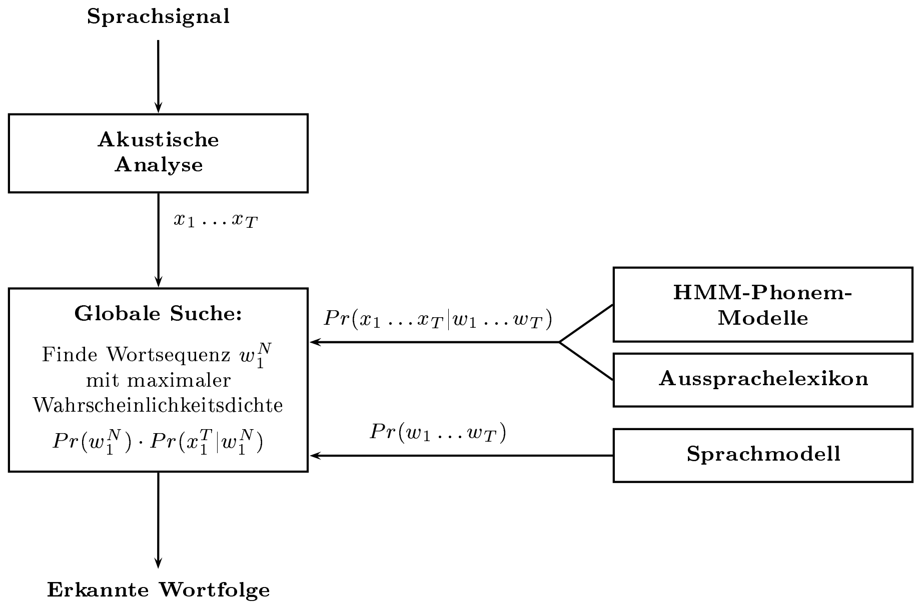


Abbildung 4: Aufbau eines statistischen Spracherkenners

2.2 Aufbau eines Spracherkenners

Der grundsätzliche Aufbau ist bei fast allen statistischen Spracherkennern derselbe, siehe hierzu Abb. 4.

Das aufgenommene Sprachsignal wird zuerst einer akustischen Analyse unterzogen. Dort werden die für eine Erkennung der Sprache wichtigen Merkmale des Signals extrahiert und in akustische Vektoren $x_1 \dots x_T$ gepackt. Diese werden daraufhin in einer globalen Suche mit den im Training gewonnenen Referenzdaten für einzelne Phoneme verglichen. Der Vergleich basiert auf der Definition eines Abstandsmaßes zwischen den akustischen Vektoren x zu den im Trainingsverfahren gewonnenen Verteilungen s dieser Vektoren. Dabei wird folgender Zusammenhang zwischen Wahrscheinlichkeiten $p(x|s)$ und den Distanzen $d(x;s)$ angenommen:

$$d(x;s) \; = \; -logp(x|s) \; = \; -const \cdot \ln p(x|s) \qquad , \qquad const \in \mathbf{R}^{\geq 0} \qquad (3)$$

Bei der Zuordnung von Testvektoren $x_1 \dots x_T$ zu den Referenzvektoren $s_1 \dots s_T$ eines Phonems muss eine *Zeitanpassung* vorgenommen werden, weil sich die Sprechgeschwindigkeiten von Test- und Referenzäußerungen unterscheiden können. Diese Zeitanpassung wird mittels Dynamischer Programmierung erreicht.

2.2.1 Sprachmodell

Die a-priori-Wahrscheinlichkeiten $Pr(w_1^N)$ sind unabhängig vom eingehenden Sprachsignal und hängen nur von der Plausibilität des Satzes w_1^N ab. Diese

Plausibilität ergibt sich eigentlich von der Syntax, Semantik und dem umgebenden Kontext des Satzes. Sprachmodelle, die in der Lage wären, den Sinngehalt eines Satzes zu überprüfen, wären jedoch „überqualifiziert" für eine reine Transkription der Sprache.

Daher begnügt man sich in der Regel mit einem *m-gramm-Modell*, das die Wahrscheinlichkeit eines Wortes in Abhängigkeit von den $m-1$ vorangehenden Worten schätzt. Die Wahrscheinlichkeiten werden üblicherweise aus einem Textcorpus von mehreren 100 Millionen Wörtern in einer *Trainingsphase* gewonnen.

2.2.2 HMM-Phonem-Modelle

Die Aufgabe des Phonemmodells ist es, den im Aussprachelexikon auftretenden Phonemen eine Folge von möglichen Zuständen zuzuordnen. Dieses Ziel wird durch die Verwendung eines probabilistischen Endlichen Automaten, auch Hidden Markov Modell (HMM) genannt, erreicht.

Die Zustände des Automaten modellieren dabei eine Wahrscheinlichkeitsverteilung für die zugelassenen akustischen Vektoren und vergleichen auf diese Weise immer einen akustischen Vektor mit einer Referenz-Vektorverteilung.

Den Transitionen des Automaten ist auch jeweils eine Wahrscheinlichkeit zugeordnet, die regelt, wie wahrscheinlich der Übergang zwischen zwei Zuständen ist. Mit der Übergangswahrscheinlichkeit wird eine Zeitanpassung zwischen den Zuständen und den Testvektoren erreicht.

Eine Hintereinanderschaltung von HMM-Phonem-Automaten ergibt dann einen Automaten für ein komplettes Wort. Welche Phonem-Automaten für ein bestimmtes Wort zu verwenden sind, steht im Aussprachelexikon.

2.2.3 Verwaltung der Hypothesen

Um eine Menge von besten Satz-Hypothesen zu konstruieren, wird zu jedem Zeitpunkt t eine Menge der bis dorthin besten Teil-Hypothesen gespeichert.

Eine Hypothese wird z.B. als Tupel $(h, s, t_A, q; t)$ gespeichert, wobei h ein Zeiger auf die Historie (die vorangehenden Wörter dieser Satzhypothese) darstellt, s den Zustand im aktuellen HMM-Automaten und t_A die Anfangszeit des aktuellen Wortes bezeichnet.

Die Qualität jeder einzelnen Hypothese ergibt sich aus der ihr zugeordneten Wahrscheinlichkeit q. Um nicht alle sich ergebenden Hypothesen speichern zu müssen, setzt man eine untere Wahrscheinlichkeits-Schranke (engl. *Threshold*). Hypothesen, deren Wahrscheinlichkeit unterhalb dieses Thresholds liegt, werden nicht weiterverfolgt in der globalen Suche. Dieser Vorgang wird als *Beam-Pruning* bezeichnet.

2.2.4 Baumsuche

In einer linearen Suche müssen am Satzanfang oder nach einem endenden Wort die HMM-Anfangszustände aller möglicher Folgewörter aktiviert werden. Üblicherweise sind die möglichen Folgewörter das gesamte Vokabular.

Nach einigen Timeframes kristallisiert sich dann eine nur sehr kleine Menge von Hypothesen heraus, die den Pruning-Prozess überlebt haben.

Weil das parallele Starten der HMM-Automaten für alle Wörter sehr rechenintensiv ist und ein hohes Maß an Redundanz birgt, werden die HMM-Automaten der einzelnen Wörter in einem Baum organisiert.

Jeder Weg von der Wurzel des Baums bis zu einem Blatt stellt dabei einen Phonemautomaten eines bestimmten Wortes dar. Dabei durchlaufen alle Wörter, die mit der gleichen Phonemsquenz anfangen, am Anfang dieselben ersten Phonem-Kanten im Baum.

Durch diese Zusammenlegung müssen z.B. am Wortanfang statt der ca. 20000 Wortautomaten nur noch etwa 100 Phonemautomaten gestartet werden. Insgesamt ergibt sich in etwa eine Reduktion des Suchraums um den Faktor 4.

3 Multi-Pass Search

Die Multi-Pass-Suche ist vor allem aus technischen Gründen für die Spracherkennung interessant:

- Für eine technische Umsetzung von Spracherkennung z.B. in einem Dialogsystem wird eine Erkennung in Echtzeit verlangt. Mit einer Multi-Pass-Suche läßt sich die Effizienz der Erkennung bis hin zur Erkennung in Echtzeit erhöhen.

- Um flexibel auf die verschiedenen Anforderungen der Kunden bezüglich des Spracherkennungssystems reagieren zu können, ist es wichtig, wiederverwendbare Programmkomponenten zu besitzen, um nicht jedesmal „das Rad neu zu erfinden“. Die Aufteilung des Suchproblems in mehrere Phasen hilft dabei, die Modularität des Systems zu erhöhen.

3.1 Steigerung der Effizienz

Das Suchproblem der Spracherkennung besteht darin, die wahrscheinlichste Wortsequenz unter Ausnutzung aller Wissensquellen zu finden.

Bei einfachen Problemstellungen ist es möglich, alle Wortautomaten für die einzelnen Wörter im Vokabular zu jedem Zeitpunkt parallel auszuwerten unter Verwendung des Viterbi- Suchalgorithmus.

Bei größeren Suchproblemen kann der Suchraum durch Verbesserungen wie der in Kap. 2.2.4 vorgestellten Baumsuche reduziert werden.

Wenn jedoch in puncto Vokabulargröße, Fehlerrate und akustischer Robustheit noch größere Anforderungen an den Erkenner gestellt werden, reichen selbst diese verbesserten Techniken nicht aus, um Spracherkennung in Realzeit durchzuführen.

Das Problem ist, dass man zur Verbesserung der Fehlerrate oftmals komplexere Modelle einsetzen muss, die dann ihrerseits wieder einen höheren Rechenaufwand bedeuten.

Bei einer 1-Pass-Suche bringt eine Verdoppelung der Rechenzeit pro untersuchter Hypothese immer eine Verdopplung der gesamten Rechenzeit mit sich.

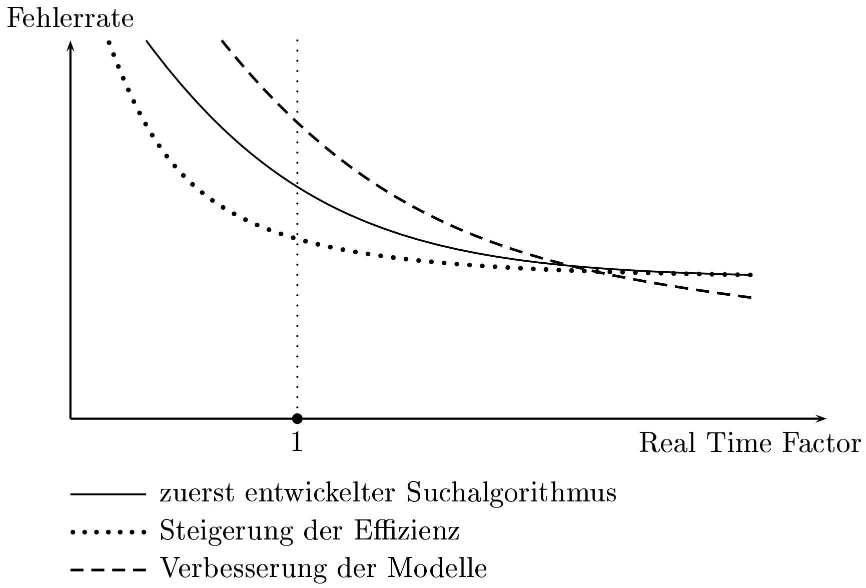


Abbildung 5: Zusammenhang von Fehlerrate und Effizienz

Das dazugehörige kompliziertere Modell, dass die Rechenzeit verdoppelt, drückt die Fehlerrate aber normalerweise nur um einen kleinen Prozentsatz nach unten. Dieser Tradeoff zwischen Effizienz und guter Erkennungsrate ist in Abb. 3.1 dargestellt.

Man erkennt dort, dass sich eine Steigerung der Effizienz oftmals eine große Verbesserung der Erkennungsrate mit sich bringt, wenn die Erkennung in Echtzeit vorgenommen werden muß. Der Grund hierfür ist, dass sich durch die gesteigerte Effizienz der Pruning-Schwellwert herabsetzen lässt. Auf diese Weise lässt sich die Anzahl der untersuchten Hypothesen erhöhen, ohne dabei den Echtzeit-Bereich zu verlassen.

Eine Verbesserung der verwendeten Modelle bringt jedoch meistens nur eine bessere Erkennung bei größeren Realzeit-Faktoren mit sich, weil die Modelle komplexer und rechenaufwändiger werden.

Mit einer Multi-Pass-Suche lässt sich der Tradeoff zwischen Effizienz und Fehlerrate weitgehend beseitigen. Durch eine sequentielle Zerlegung des Dekodierungsvorgangs in eine Hypothesenbildungsphase und eine anschließende Testphase werden die Vorteile einer effizienten Erkennung mit den Vorteilen eines besseren Sprachmodells kombiniert.

Zur Hypothesenbildung bedient man sich einer Modellierung, die Trennschärfe für Effizienz opfert und erzeugt damit eine Zwischenrepräsentation alternativer Lösungskandidaten. Während der Testphase werden die Alternativen erneut bewertet, diesmal aber auf der Grundlage eines sehr viel genaueren Modells. Bei geeigneter Wahl der Modelle für Vorauswahl und Test liefert dieses Prinzip der schrittweisen Verfeinerung einen sehr schnellen Dekodierer, der auch meistens die bezüglich des Testmodells optimale Lösung findet.

Voraussetzung ist hierfür, dass die optimale Lösung bezüglich des Testmodells mit einer sehr hohen Wahrscheinlichkeit in der Hypothesenmenge des ersten Durchlaufs zu finden ist. Einen Algorithmus, der *immer* garantiert, dass die Lösungsschar der 1. Phase die beste Lösung der 2. Phase enthält, nennt man einen *admissiblen Algorithmus*. Um Admissibilität zu garantieren, müßte man in der Multi-Pass-Suche jedoch zu viele Einschränkungen in Kauf nehmen, so dass man versucht, die zusätzlichen Erkennungsfehler empirisch zu beheben.

Für die Fehlerrate des Gesamtsystems läßt sich leider nur eine obere und untere Schranke angeben. Es sei:

$p(r1)$	Wahrscheinlichkeit, dass ein Wort in der Lösungsschar der 1. Phase enthalten ist
$p(r2)$	Wahrscheinlichkeit, dass das Wort in der Lösungsschar der 2. Phase enthalten ist
$p(r1, r2)$	Wahrscheinlichkeit, dass ein Wort nach Durchlaufen der 1. und danach der 2. Phase in der Lösungsschar enthalten ist

Weil aufgrund der Korrelation zwischen den verwendeten Modellen im 1. und 2. Pass immer die Beziehung $p(r2|r1) \geq p(r2)$ gilt, erhält man

$$p(r1) \cdot p(r2) \;\leq\; p(r1, r2) \;\leq\; \min\{p(r1), p(r2)\} \tag{4}$$

Für die Berechnungszeiten lassen sich hingegen exakte Angaben machen. Dazu nehmen wir ein System mit folgenden Eckdaten an:

V	Vokabulargröße
t_2	Berechnungszeit pro Wort in der 2. Phase (Mittelwert)
v	durchschnittliche Größe der Hypothesenliste nach dem 1. Pass
t_1	Berechnungszeit pro Wort im der 1. Phase (Mittelwert)

Dann ergibt sich für die gesamte Berechnungszeit

- bei einem 1-Pass-System, das nur aus einer Testphase besteht: $V \cdot t_2$
- bei einem 2-Pass-System: $V \cdot t \;+\; v \cdot t_2$
- ein Verringerung der Rechenzeit um den Faktor $t_1/t_2 \;+\; v/V$

3.2 Modularisierung des Suchproblems

Ein weiterer Vorteil der Multi-Pass-Suche ist die bessere Strukturierung des Systems. An der Entwicklung eines so umfangreichen und komplexen Projekts wie dem eines Spracherkenners sind meistens mehrere Personen beteiligt. Für eine sinnvolle Aufgabenteilung ist es vorteilhaft, wenn man einzelne Programmkomponenten unabhängig voneinander programmieren, debuggen, testen und verbessern kann.

Die Multi-Pass-Suche ermöglicht eine solche Modularisierung des Programms auf eine sehr einfache Weise, indem jede Suchphase in ein eigenes Modul eingebettet wird.

Wie in Abb. 6 dargestellt, wird auch eine weitgehende Trennung der Trainingsdaten zwischen den Modulen erstrebt. Wissensquellen werden nur dann

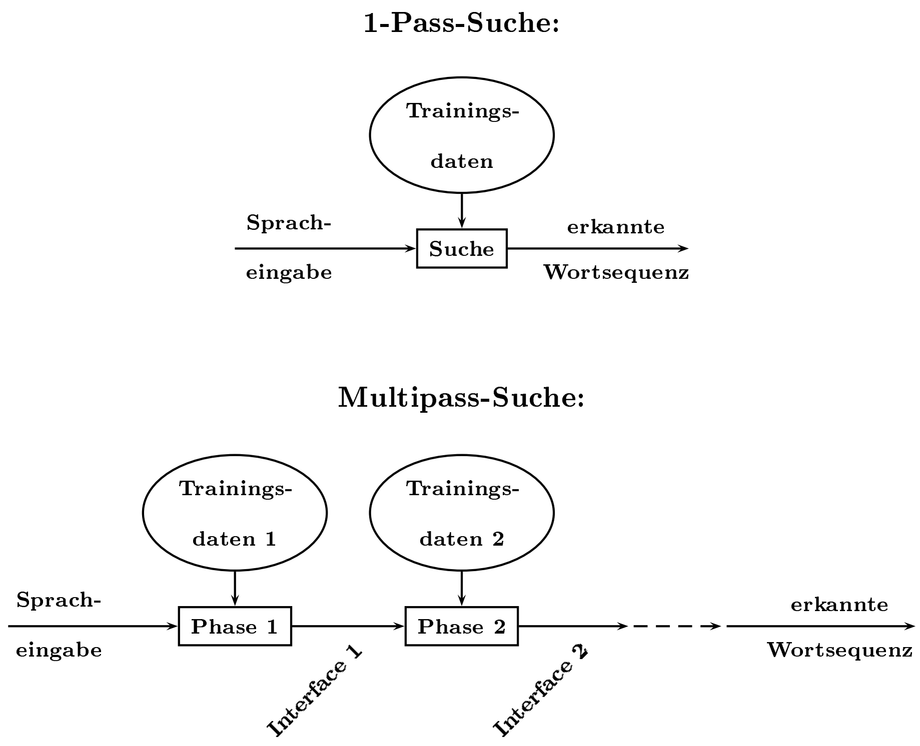


Abbildung 6: 1-Pass- und Multi-Pass-Suche im Vergleich

gemeinsam benutzt, wenn diese nicht zu speziell auf eine einzelne Suchphase zugeschnitten sind. Durch separates Training handelt man sich außerdem keine unnötigen Abhängigkeiten zwischen den Modulen ein.

4 Zwischenrepräsentationen

Es gibt mehrere Möglichkeiten zur Aufteilung der Suchphasen und der Wahl der Zwischenrepräsentation:

- Die erste Möglichkeit ist, dass man Phonemhypothesen wiederbewertet. Die Aufteilung kann z.B. so aussehen, dass man eine Erkennung mit kontextunabhängigen Phonemen im 1. Pass vornimmt und die wahrscheinlichsten Phonemhypothesen im 2. Pass mit einem Triphon-Modell wiederbewertet

- Die zweite Möglichkeit ist ein Rescoring von Worthypothesen. Dies kann vorgenommen werden, indem zum Satzanfang und nach jedem endenden Wort mit einem approximativen akustischen Modell eine Liste der wahrscheinlichsten nachfolgenden Wörter generiert und im 2. Pass nur diejenigen Wörter wiederbewertet, die in dieser Liste stehen.

- Die dritte Möglichkeit schlielich ist eine Neubewertung kompletter Sätze z.B. mit einem besseren Sprachmodell wie einer kontextfreien probabilistischen Grammatik.

In diesem Kapitel werden wir uns auf das Rescoring von kompletten Sätzen beschränken und uns die Möglichkeiten für die Zwischenspeicherung von Satzhypothesen anschauen.

Es gibt im wesentlichen zwei unterschiedliche Zwischenrepräsentationen (Abb. 7), deren mögliche Implementierungen sowie Vor- und Nachteile wir im Folgenden genauer unter die Lupe nehmen werden:

1. bei der *N-Best-Liste* wird eine Liste der N besten Satzhypothesen an den nächsten Pass weitergegeben. Die Liste wird nach der Wahrscheinlichkeit der Satzhypothesen sortiert.

   In vielen Implementierungen orientiert sich die Länge der Liste nicht an einem fixen N, sondern wächst dynamisch mit der Anzahl der Satzhypothesen, der zeitlichen Länge der Äußerung oder der Länge der Wortsequenz. In diesem Fall spricht man allgemein von einer *Bestenliste* oder auch *Kandidatenliste.*

2. ein *Wortgitter / Wortgraph / word lattice* besteht im Allgemeinen aus einer Menge von Wortkanten (w, t_a, t_b, q), wobei q die Wahrscheinlichkeit ist, dass das Wort w in der Zeit $t_a \dots t_b - 1$ gesprochen wurde. Eine Untermenge dieser Wortkanten formt genau dann eine Satzhypothese, wenn die Kanten zeitlich hintereinander liegen. Die eventuell dazwischen liegenden Silence-Abschnitte wurden nicht in die Darstellung von Abb. 7 aufgenommen. Entweder werden diese Abschnitte als Silence-Kanten im Wortgitter abgespeichert oder man ordnet jedem Timeframe in einem separaten Array eine Silence-Wahrscheinlichkeit zu.

   Das Wortgitter ermöglicht eine kompaktere Darstellung alternativer Satzhypothesen als die N-Best-Liste. Dabei wird die Tatsache ausgenutzt, dass sich die N besten Wortketten oft nur an einer Wortposition paarweise unterscheiden. Ein weiterer Vorteil ist, dass viele Rescoring-Algorithmen effizienter auf einem Wortgitter operieren können als auf einer N-Best-Liste. Insbesondere ist die explizite Angabe der Zeitgrenzen von Vorteil für ein akustisches Rescoring.

Man beachte, dass es auch Variationen und Mischformen der Ansätze gibt. Zum Beispiel wird manchmal wird die akustische Bewertung der Wörter oder des Satzes auch bei der N-Best-Liste mitgeliefert.

Die Schnittstellen müssen außerdem nicht zwangsläufig komplette Sätze an den nächsten Pass weitergeben, sondern können auch auf Wort- oder Phonemebene operieren.

Beim IBM-Spracherkenner wird z.B. im ersten Pass mit dem akustischen Fast-Match (Kap. 5) eine Bestenliste der wahrscheinlichsten Wörter erstellt, was dem Bestenliste-Ansatz auf Wortebene entspricht.

Im AT&T-System (Kap. 6.2) wird zunächst ein Phonemgitter erzeugt, welches danach in ein Wortgitter konvertiert wird.

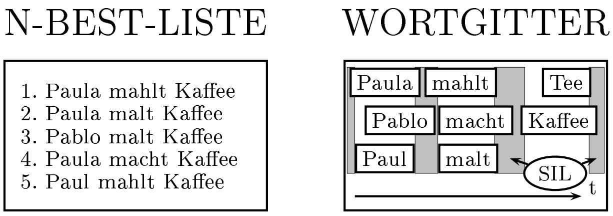


Abbildung 7: Potentielle Zwischenrepräsentationen (Bsp. aus [6])

4.1 N-Best-Listen

Wir werden in diesem Abschnitt einige mögliche Implementationen der N-Best-Liste auf Satzebene betrachten.

Der N-Best-Ansatz ist insbesondere dann die erste Wahl, wenn eine akustische oder linguistische Nachbearbeitung (*Rescoring*) verlangt wird, die nicht oder nur mit sehr großem Aufwand direkt in den Dekodierungsprozeß integriert werden kann.

Auch in der Spracherkennung mit kleinem Vokabular gibt es Beispiele, in denen der Einsatz einer N-Best-Liste Sinn macht. So können sprachgesteuerte Schachcomputer nach Vorlage der N besten Kommando-Hypothesen diejenige Hypothese als den auszuführenden Zug festlegen, die als erstbeste Hypothese mit Partiestellung und Spielregeln vereinbar ist. Ein anderes Beispiel ist ein System, das 16-stellige Kreditkartennummern annehmen soll. Diese Nummern seien jedoch nur dann gültig, wenn die Bedingung „Nummer durch 127 teilbar“ erfüllt ist. Auch hier kann nach Vorlage der N-Best-Liste bequem die erstbeste Nummer, die durch 127 teilbar ist, ausgewählt werden.

Ein großer Vorteil der N-Best-Liste ist, dass sie aufgrund ihres relativ einfachen Interfaces leicht in ein bestehendes System integriert werden kann.

Es gibt drei unterschiedliche Möglichkeiten, die zeitsynchrone Viterbi-Suche dahingehend zu modifizieren, dass nicht nur die beste Lösung am Satzende ausgegeben wird, sondern die N besten Lösungen. Dabei wird sich jedoch herausstellen, dass der exakte *Sentence-Dependent N-Best*-Algorithmus leider den Zeit- und Speicheraufwand zu sehr erhöhen würde und deshalb nicht praktikabel ist. Die anderen beiden Ansätze sind effizienter, haben aber den Nachteil, dass einige Lösungen „verloren“ gehen können. Es ist jedoch immer garantiert, dass sich die beste Lösung auf dem 1. Platz der N-Best-Liste befindet.

4.1.1 Sentence-Dependent N-Best

Die fundamentale Idee in diesem Ansatz besteht darin, alle aktiven Hypothesen $(w, s, t; h, q, b)$, wobei

w	momentan in der Erkennung befindliches Wort
s	ID für den Zustand im Phonemmodell
t	aktueller Zeitframe (wird i.d.R. bei der Viterbi-Suche nicht explizit gespeichert)

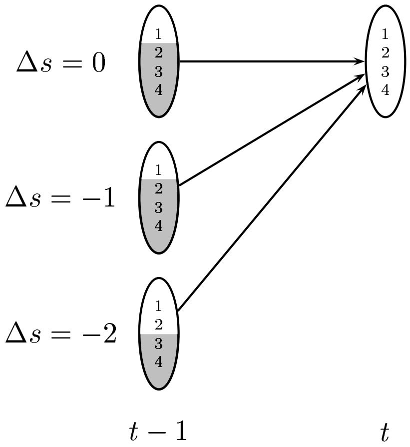


Abbildung 8: Zeitanpassung beim Sentence-Dependent 4-Best im (0,1,2)-Standardmodell

h	Historie der vorangehenden Wörter
q	Verbundwahrscheinlichkeit der gesamten Hypothese
b	Backpointer auf den Wortanfang von w

durch Hypothesen $(w, s, t; (h_1, q_1, b_1), \ldots (h_N, q_N, b_N))$ zu ersetzen und somit die Anzahl der berücksichtigten Historien von 1 auf N zu erhöhen.

Bei der Zeitanpassung müssen folgende Dinge geschehen:

1. Multipliziere zu den $3N$ Vorgänger-Hypothesen (bezogen auf das (0,1,2)-Standardmodell) die Transitionswahrscheinlichkeiten

2. Suche davon die N besten mit unterschiedlicher Historie heraus

3. Multipliziere jeweils die Emissionswahrscheinlichkeit und reiche die Historie und den Backpointer weiter

Wenn zwei oder mehrere Hypothesen zu einem Zeitpunkt in einem Zustand zusammenlaufen, wird also eine Fallunterscheidung getroffen: Falls die Hypothesen die gleiche Historie besitzen (d.h. dass die Hypothesen bis auf die Wortgrenzen gleich sind), wird nur versucht, die beste der beiden Hypothesen in die N-Best-Liste aufzunehmen. Ansonsten wird versucht, beide Hypothesen separat aufzunehmen.

Offensichtlich erfordert eine effiziente Implementierung der N-Best-Liste schnelle Algorithmen:

1. für den Vergleich zweier Historien: alle erkannte Historien werden zentral in einem Array gespeichert. Gleiche Historien bekommen im Array denselben Index. Der Vergleich der Historien kann somit auf den Vergleich der Indizes reduziert werden.

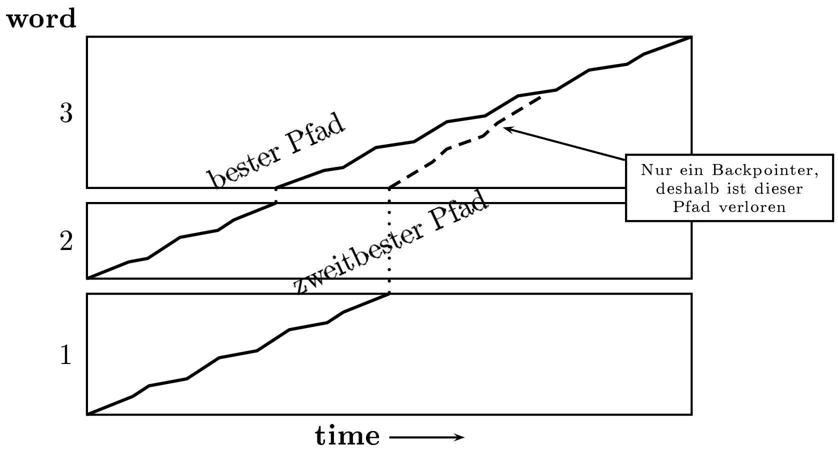


Abbildung 9: Schwachstelle des Traceback-abhängigen N-Best

2. für das Mergen dreier N-Best-Listen: weil die Listen jeweils sortiert sind, kann dies unter Verwendung dreier Zeiger mit $6N$ Vergleichsoperationen geschehen.

4.1.2 Traceback-Dependent N-Best

Bei der Untersuchung der N-Best-Liste stellt sich heraus, dass sich viele Satzhypothesen nur in jeweils einem Wort der Historie unterscheiden. Beim Sentence-Dependent N-Best wird deshalb viel Zeit und Speicher damit aufgewandt, die Kombinationen der ansonst fast gleichen Hypothesen parallel zu propagieren. Um auch bei längeren Sätzen eine ausreichende Anzahl von Alternativen vorliegen zu haben, benötigen wir einen N-Best-Algorithmus, dessen Zeit- und Speicheraufwand auch bei $N >> 1$ in der Größenordnung des 1-Best-Algorithmus liegt.

Deshalb liegt die Idee nahe, die N besten Hypothesen nicht auf der Zustands-Ebene mitzuführen, sondern auf Wort-Ebene. Dies wird erreicht, indem man die N besten Historien $H_1 \dots H_N$ mitsamt ihren Bewertungen $Q_1 \dots Q_N$ in die Traceback-Felder auslagert und innerhalb des Wortes einen gewöhnlichen Beam-Search mit Hypothesen $(w, s, t; q, b)$ durchführt.

Anstatt zu jedem Timeframe nur das beste endende Wort in dem Traceback-Array zu speichern, werden dort somit die N besten Hypothesen gespeichert.

Wie im normalen Beam-Search wird *nur* die Bewertung der besten Hypothese $Q_1(t)$ als Basis für zukünftige Wörter verwendet. Die Bewertung eines Wortes w, das zwischen t_a und t_b liegt, betrangt somit $Q(w, t_b)/Q_1(t_a)$. Diese Bewertung wird zum Zeitpunkt t_b mit den vorherigen N besten Hypothesen $Q_1(t_a) \dots Q_N(t_a)$ multipliziert. Falls diese Gesamt-Bewertung ausreichend gut ist und es keine bessere Hypothese mit gleicher Wortsequenz gibt, wird die Hypothese dann in die N-Best-Liste des Timeframe t_b eingetragen.

Am Satzende brauchen dann nur die N Tracebackfelder der N-Best-Liste des Timeframe T bis zum Satzanfang zurückverfolgt werden.

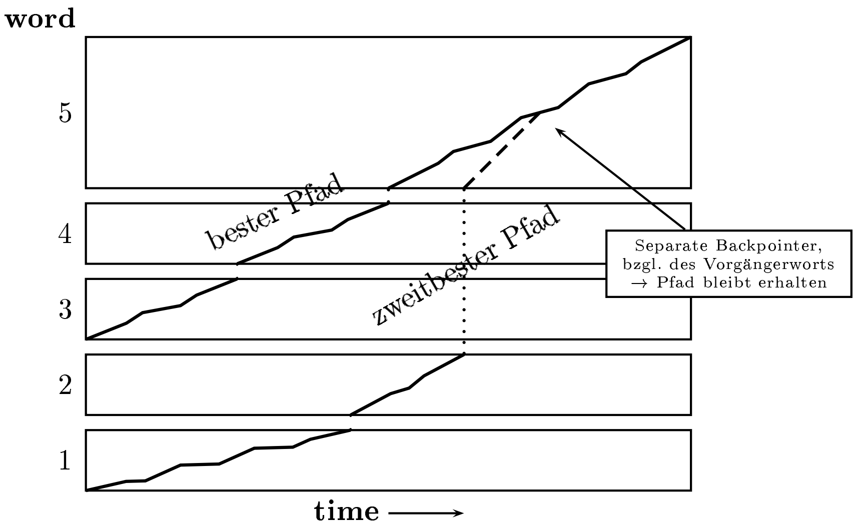


Abbildung 10: Vorteil des Word-Dependent N-Best

Es stellt sich jedoch heraus, dass der Traceback-N-Best-Algorithmus den gravierenden Nachteil hat, einige hoch zu bewertende Hypothesen einfach auszulassen. Dies ist darauf zurückzuführen, dass bei der Zeitanpassung für jede Hypothese $(w, s, t; q, b)$ nur *eine* Bewertung q berechnet wird und nur *ein* Backpointer b auf den Anfang des Wortes mitgeführt wird. Dieser Sachverhalt ist in Abb. 9 dargestellt.

4.1.3 Word-Dependent N-Best

Der Word-Dependent N-Best Algorithmus stellt einen Kompromiss zwischen dem Sentence- und der Traceback-Dependent N-Best dar.

Im Traceback-Ansatz wird angenommen, dass die Startzeit des Wortes unabhängig von der Historie ist, weshalb sich Situationen wie in Abb. 9 ergeben. Im Sentence-Dependent Algorithmus nimmt man hingegen eine Abhängigkeit der Startzeit von *allen* Wörtern der Historie an.

Als Kompromiss hält der Word-Dependent N-Best die Startzeiten eines Wortes w genau dann getrennt, wenn das Vorgängerwort v, d.h. nur das letze Wort der Historie, unterschiedlich ist.

Diese sogenannte *Wortpaar-Approximation* wird algorithmisch umgesetzt, indem man für jedes Vorgängerwort v eine eigenständige Hypothese $(w, s, t; v, q, b)$ propagiert und den Startzustand von w wird mit Wahrscheinlichkeiten $Q_1(v, t)$ initialisiert, die vom Vorgängerwort v abhängen.

In einer Situation wie in Abb. 10 kann der Word-Dependent N-Best deshalb im Gegensatz zum Traceback-Dependent N-Best die Startzeiten getrennt halten.

In einer Situation, wie sie in Abb. 11 dargestellt ist, versagt jedoch auch die Wortpaar-Approximation. Dies ist allerdings nicht weiter schlimm, weil eine solche Situation in der Praxis äußerst selten vorkommt.

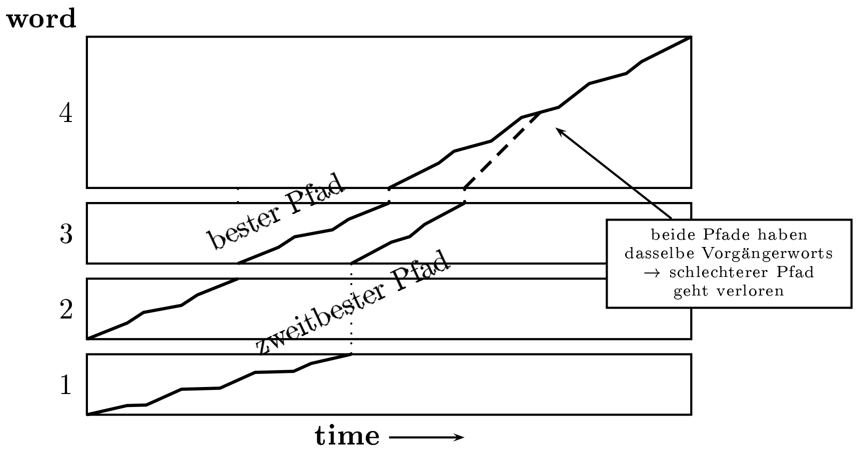


Abbildung 11: Versagen der Wortpaar-Approximation

Da es nur eine kleine Anzahl $k << N$ von Historien mit unterschiedlichen Vorgängerwörtern gibt, arbeitet dieser Algorithmus schneller als der exakte Sentence-Dependent N-Best.

Der Word-Dependent-Algorithmus weist in der Implementierung sowohl Merkmale des Traceback- als auch des Sentence-Dependent N-Best-Algorithmus auf.

Weitergehende Informationen finden sich in [5].

4.2 Wortgitter

Die in 4.1 beschriebenen N-Best-Listen haben, ob sie nun in der Sentence-, Traceback- oder Word-Dependent Variante implementiert werden, einen gemeinsamen Nachteil: da sich viele Historien nur an paarweise einer Wortposition unterscheiden, birgt die Datenstruktur einer N-Best-Liste ein hohes Maß an Redundanz. Diese Redundanz, die sich in einem höheren Speicher- und Zeitbedarf niederschlägt, kann durch die Verwendung eines *Wortgitters* (*word lattice*) als alternativer Datenstruktur beseitigt werden.

Ein Wortgitter (*word lattice*) ist im allgemeinen eine Menge von einzelnen Wortkanten (w, t_a, t_b, q), wobei q die Wahrscheinlichkeit ist, dass das Wort w in der Zeit $t_a \dots t_b - 1$ gesprochen wurde.

Eine Folge $w_1 \dots w_N$ dieser Wortkanten ist genau dann eine Satzhypothese, wenn die Kanten zeitlich hintereinander liegen und sich nicht überlappen. Die Wahrscheinlichkeit einer Satzhypothese ergibt sich aus dem Produkt der Wahrscheinlichkeiten der Wortkanten mit den Wahrscheinlichkeiten der dazwischenliegenden Silence-Abschnitte.

Es gibt mehrere Möglichkeiten, den Suchvorgang so zu gestalten, dass ein Wortgitter erzeugt wird. Diese Möglichkeiten werden in den folgenden Abschnitten erläutert.

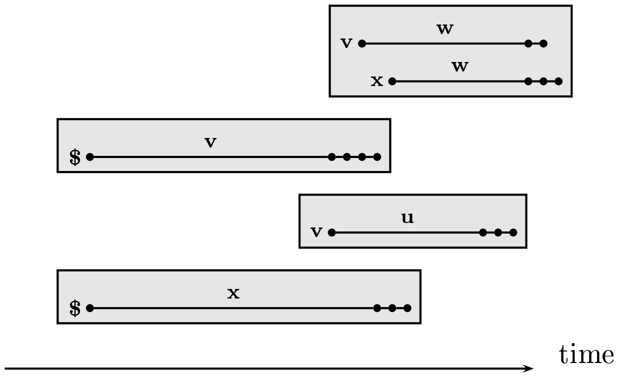


Abbildung 12: Hypothesen im wortabhängigen Lattice

4.2.1 Wortabhängige Strukturierung des Suchraums

Die Algorithmen für die Generierung einer wort- oder Traceback-abhängigen-N-Best-Liste lassen sich mit einfachsten Mitteln umfunktionieren, dass anstatt einer N-Best-Liste ein Wortgitter erzeugt wird.

Alle Vor- und Nachteile der Verfahren wie die Wortpaar-Approximation werden dabei „geerbt", mit dem zusätzlichen Vorteil der Kompaktheit des Wortgitters. Bei einem großen Wert für N liefern die entsprechenden N-Best-Implementierungen exakt dieselbe Fehlerrate.

Bei einem Traceback-abhängigen Lattice haben die Einträge des Gitters die Form (w, t_a, t_b, q). Der Nachteil des Traceback-abhängigen N-Best, hochbewertete Hypothesen auszulassen, macht sich im Gitter dadurch bemerkbar, dass die Wörter zwar einen großen Fächer unterschiedlich bewerteter Endzeiten besitzen, aber nur eine einzige Startzeit zur Auswahl steht.

Bei dem wortabhängigen Lattice, in dessen Gittereinträgen (w, t_a, t_b, v, q) die Wortidentität des Vorgängerwortes mitgespeichert wird, stehen hingegen in der Regel mehrere unterschiedliche Zeiten für den Wortanfang zur Auswahl.

4.2.2 Zeitabhängige Strukturierung des Suchraums

Eine andere Möglichkeit, um an eine Menge bewerteter Wortkanten $(w; t_a, t_b, q)$ zu gelangen ist es, zu jedem neuen Timeframe eine neue Baumkopie zu starten. Der Startzustand $(w, s = 1, t_a)$ wird mit der Wahrscheinlichkeit 1 initialisiert, alle anderen Zustände erhalten die Wahrscheinlichkeit 0. Weil als einziger Startzustand nur t_a zugelassen wird, braucht bei der Expansion eines Zustands *kein Backpointer* auf die beste Startzeit weitergereicht werden.

Der Vorteil einer solchen Implementierung liegt darin, dass alle möglichen Start- und Endzeiten der Worte berechnet werden, d.h. der Algorithmus liefert immer die *exakte* Lösung. Leider geht das Starten einer neuen Baumkopie zu jedem Timeframe auf Kosten der Effizienz, weil man kein Bigramm-Lookahead-Pruning während der Erkennung durchführen kann, sondern nur ein Unigramm-Lookahead-Pruning. Außerdem lässt sich wegen dem unbekannten Vorgängerwort im Zusammenhang mit diesem Algorithmus keine effiziente

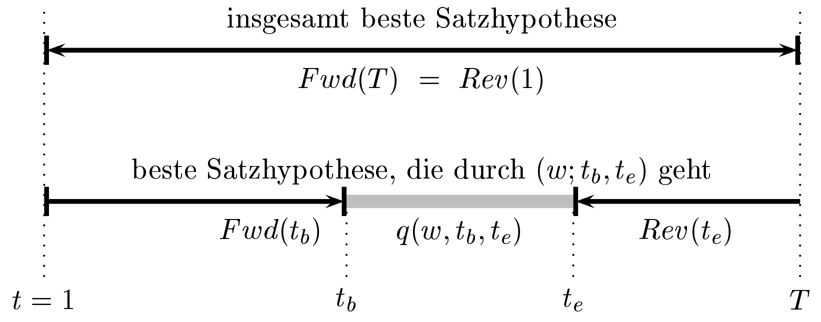


Abbildung 13: Forward-Backward-Pruning

Acrossword-Modellierung realisieren.

4.2.3 Lattice-Pruning

Das Ziel des Lattice-Pruning ist es, die Größe des Gitters vor der eigentlichen Rescoring-Phase noch einmal zu reduzieren. Mit einem solches Pruning sollen möglichst viele Wortkanten aus dem Gitter eliminiert werden, ohne dass dabei die richtige Wortfolge verloren geht.

Der Trick des Lattice-Pruning besteht in der Ausnutzung des zusätzlichen Wissens, dass der Satz zum Timeframe $t = T$ endet. Darauf aufbauend, lässt sich die Wahrscheinlichkeit einer Kante $(w; t_a, t_b)$ im Wortgitter ein erneutes Mal schätzen.

Diesmal wird die Wahrscheinlichkeit der Wortkante im Kontext des gesamten Satzes berechnet, d.h. nicht nur von $t = t_a \ldots t_b$, sondern von $t = 1 \ldots T$.

Dies geschieht, indem man für jede Kante $(w; t_a, t_b)$ im Wortgitter die Bewertung $Q(w; t_a, t_b)$ des besten Wortfolge durch dieses Wort berechnet. Es gilt

$$Q(w; t_a, t_b) \;=\; Fwd(t_a) \cdot q(w; t_a, t_b) \cdot Rev(t_b) \tag{5}$$

wobei

$Fwd(t_a)$	max. Wahrscheinlichkeit aller Wortfolgen von $t = 1 \ldots t_a$
$Rev(t_b)$	max. Wahrscheinlichkeit aller Wortfolgen von $t = t_b \ldots T$
$q(w; t_a, t_b)$	Wahrscheinlichkeit der Hypothese $(q; t_a, t_b)$

Die Größen $Fwd(t)$ und $Rev(t)$ werden rekursiv mittels Dynamischer Programmierung berechnet. Weil $Fwd(t)$ „vorwärts“ in der Zeit aus $Fwd(1) \ldots Fwd(t-1)$ berechnet wird, $Rev(t)$ hingegen in einem späteren „Rückwärtslauf“ aus $Rev(T) \ldots Rev(t+1)$, spricht man von *Forward-Backward-Pruning*.

Die Größe $Fwd(t)$ wird schon während des Decodierens berechnet, um die Pruningschwelle zu ermitteln, wohingegen $Rev(t)$ anschließend explizit berechnet werden muss.

Es überleben nur diejenigen Wortkanten im Gitter, für die

$$Q(w; t_a, t_b) \;>\; f_{lat\_fb} \cdot Fwd(T) \tag{6}$$

bzw.

$$\frac{Fwd(t_a) \cdot q(w; t_a, t_b) \cdot Rev(t_b)}{Fwd(T)} < f_{lat\_fb} \tag{7}$$

gilt, wobei $Fwd(T) = Rev(1)$ die Wahrscheinlichkeit des insgesamt besten Satzes ist. $f_{lat\_fb} < 1$ heißt *Forward-Backward-Pruningschwelle.*

4.2.4 Die Forward-Backward-Suche

Das Forward-Backward-Verfahren bietet eine Möglichkeit, die akustische Suche sehr kompakt und effizient in zwei Suchphasen durchzuführen, wobei das Verfahren sehr eng mit der Schnittstelle des Wortgitters verwurzelt ist. Die Vorteile der Forward-Backward-Suche kurz im Überblick:

- erzeugt Wortgitter mit vielen Kombinationsmöglichkeiten, weil für jede Worthypothese ein großer Fächer von Anfangs- und Endzeiten produziert wird
- läßt sich mit Forward-Backward-Pruning kombinieren
- erlaubt schnellen 1. Pass mit einfachen Modellen
- 2. Pass benötigt am Satzende nur Sekundenbruchteile

Wie ihr Name schon vermuten läßt und in Abb. 4.2.4 dargestellt, besteht die Forward-Backward-Suche aus einem Vorwärts- und einem Rückwärtsdurchlauf zur Berechnung der Wahrscheinlichkeiten der Wortkanten im Gitter.

Der erste Pass ist eine zeitsynchrone Viterbi-Suche mit einem einfachen Sprachmodell in Vorwärtsrichtung. Dabei wird ein Wortgitter angelegt, das für jede Wortkante $(w; t_a, t_b, q)$ einen großen Fächer von Endzeiten t_b mit den jeweiligen Wahrscheinlichkeiten q zur Verfügung stellt.

Wenn der Satz beendet wird (oder ein Silence-Abschnitt auftritt), wird erneut eine Viterbi-Suche durchgeführt, diesmal allerdings in Rückwärtsrichtung. Diese Rückwärtssuche, unterscheidet sich prinzipiell von der Vorwärtssuche nur durch eine umgedrehte Orientierung der Kanten in den Hidden-Markov-Modellen und durch ein modifiziertes m-Gramm-Modell.

Der Vorteil einer solchen Vorgehensweise ist, dass in der Rückwärtssuche zu jedem Timeframe t nur solche Wörter w gestartet werden brauchen, die als Kandidaten $(w; t_a, t_b, q)$ mit Endzeitpunkt $t_b = t$ im Wortgitter vermerkt sind. Dadurch erreicht man eine gewaltige Einschränkung des Suchraums, was die Verwendung eines genaueren Sprachmodells in der Rückwärtssuche ermöglicht.

Der Suchraum läßt sich aber noch weiter einschränken, indem man *während* des Rückwärtslaufs ein Forward-Backward-Pruning durchführt. Die Größe $Fwd(t)$ wird wieder während des Vorwärtslaufs implizit berechnet, um die Pruningschwelle für den Vorwärtslauf zu ermitteln. Beim Rückwärtslauf wird jetzt nicht nur die Wahrscheinlichkeit $Rev(t)$ der besten Wortfolge von $T \dots t$ zur Berechnung der Pruningschwelle herangezogen, wie es die Symmetrie zum Vorwärtslauf vermuten ließe, sondern es wird $Rev(t)$ zusammen mit $Fwd(t)$ zur Berechnung der Pruningschwelle benutzt.

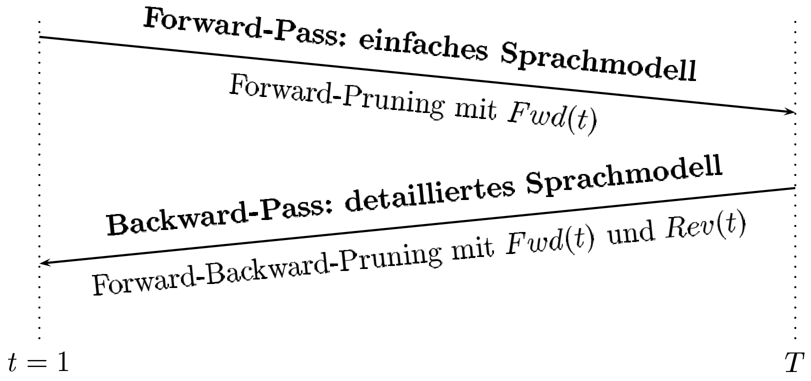


Abbildung 14: Schaubild Forward-Backward-Suche

Wenn man im Vor- und Rückwärtslauf unterschiedliche Sprachmodelle verwendet, ist jedoch zu beachten, dass im Allgemeinen $Fwd(T) \neq Rev(1)$ gilt. Folglich muss die Normierung von Gleichung (7) geändert werden, um korrektes Pruning zu gewährleisten. Dies soll hier jedoch nicht diskutiert werden.

Insgesamt erzielt man durch die Einschränkung des Suchraums beim Rückwärtspass laut [5] eine etwa 1000 mal schnellere Laufzeit bzw. einen Realzeit-Faktor von 1/60 im Vergleich zu einem Faktor von 20.

Die Forward-Backward-Suche beschleunigt aber auch indirekt die Vorwärtssuche. Mit dem Rückwärtslauf lassen sich viele Fehler des Vorwärtslaufes korrigieren, insbesondere die Anfangszeit eines Wortes betreffende Fehler, so dass man dort ein einfaches Sprachmodell anwenden kann. Außerdem lässt sich mit dem Rückwärtslauf ein größerer Fächer von Anfangszeiten für die einzelnen Wörter im Wortgitter erzeugen, was die Kombinationsmöglichkeiten im Wortgitter erhöht.

Leider besitzt die Forward-Backward-Suche aber auch eine offensichtliche Schwäche: der zweite Pass mit dem genaueren Sprachmodell kann erst gestartet werden, nachdem der erste Pass beendet ist. Somit kann die Ausgabe der erkannten Wortsequenz erst mit einiger Zeitverzögerung erfolgen. Bei Diktiersystem-Anwendungen ist es deshalb nicht möglich, die Wörter schon während des Sprechens zu verifizieren, wie dies bei einigen 1-Pass-Algorithmen möglich ist.

Weitere Informationen zur Forward-Backward-Suche finden sich in [8] und [5].

5 Das akustische Fast-Match

Die in diesem Kapitel vorgestellte akustische Fast-Match-Suche kann einer detaillierten Suche vorangestellt werden, um die Anzahl der später zu untersuchenden Wortkanten signifikant zu reduzieren.

Grundlage der Fast-Match-Suche ist ein stark vereinfachtes Phonemmodell, das eine besonders effiziente, aber relativ ungenaue Erkennung erlaubt.

Dieses vereinfachte Phonemmodell lässt sich aus dem Standard-HMM-Phonem-Modell (siehe Abschnitt 2.2.2) herleiten.

5.1 Detailliertes HMM-Phonemmodell

Für eine gegebene Folge akustischer Vektoren x_1^T und eine Menge von verborgenen ('hidden') Referenzzuständen $s = 1 \ldots S(\alpha)$ für ein Phonem α ist es Aufgabe des Phonemmodells, die Wahrscheinlichkeit

$$p(x_1^T|\alpha) = \sum_{s_1^T} p(x_1^T, s_1^T|\alpha) \approx \max_{s_1^T} p(x_1^T, s_1^T|\alpha) \tag{8}$$

möglichst realistisch zu schätzen. Die Wahrscheinlichkeit in Gleichung (8) ergibt sich dabei als eine Summe der Wahrscheinlichkeiten aller möglichen Pfade $s_1 \ldots s_t$ durch den Phonem-Automaten und läßt sich approximieren, indem man nur die Wahrscheinlichkeit entlang des besten Pfades nimmt (Maximum-Approximation). Diese Pfadwahrscheinlichkeit lässt sich aufspalten in eine Transitions- und eine Emissionswahrscheinlichkeit:

$$\begin{aligned}
\max_{s_1^T} p(x_1^T, s_1^T|\alpha) &= \max_{s_1^T} \prod_{t=1}^{T} p(x_t, s_t|x_1^{t-1}, s_1^{t-1}, \alpha) && (9)\\
&= \max_{s_1^T} \prod_{t=1}^{T} \underbrace{p(s_t|x_1^{t-1}, s_1^{t-1}, \alpha)}_{Transitionsw.} \cdot \underbrace{p(x_t|x_1^{t-1}, s_1^{t}, \alpha)}_{Emissionsw.} && (10)\\
&\approx \max_{s_1^T} \prod_{t=1}^{T} p(s_t|s_{t-1}, \alpha) \cdot p(x_t|x_{t-1}, s_{t-1}, \alpha) && (11)\\
&=: D(x_1^T|\alpha) && (12)
\end{aligned}$$

Der Schritt von (10) nach (11) beinhaltet die Annahme, dass die Transitionswahrscheinlichkeit nur vom Vorgängerzustand abhängt und dass die Emissionswahrscheinlichkeit von x_t allein vom Zustand s_t abhängt, d.h. dass es sich bei dem vorliegenden Phonemautomaten um ein sog. *first order Hidden-Markov-Modell* (HMM) handelt.

Die mit Hilfe eines solchen HMM (siehe Abb. 15 berechnete *detaillierte Wahrscheinlichkeit* $D(x_1^T|\alpha)$ wird in den detaillierten Phonemmodellen zur Bewertung der Phonem-Hypothesen benutzt.

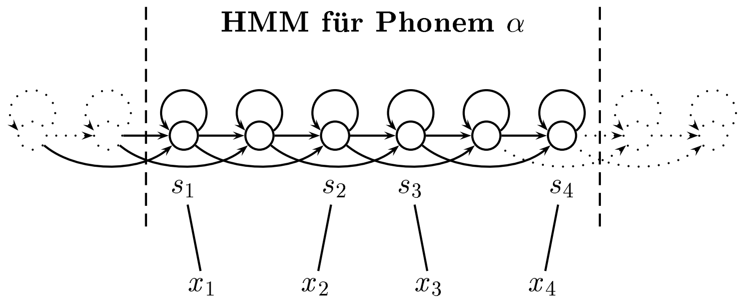


Abbildung 15: Hidden Markov Modell eines Phonems

Pseudocode für die Berechnung von $D(x_1^T|\alpha)$ gemäß dem HMM aus Abb. 15:

procedure DetailedMatch($x_1 \dots x_T$, α):$D(x_1^T|\alpha)$
begin
  for $t := 1$ **to** T **do**
    for $s := 1$ **to** S **do**

$$q(t,s) \;:=\; \max_{\Delta s=0}^{2} q(t-1, s-\Delta s) \;\cdot\; p(s|s-\Delta s, \alpha) \;\cdot\; p(x_t|s_t, \alpha)$$

    end
  end
  return $D(x_1^T|\alpha) \;:=\; q(T,S)$
end

Der Algorithmus benutzt zur Berechnung der Wahrscheinlichkeit zwei ineinander verschachtelte Schleifen. Die äußere Schleife läuft über die Timeframes, die innere über die Zustände des Automaten. Innerhalb der Schleifen wird die Zeitanpassung mittels Dynamischer Programmierung vorgenommen. Dabei gibt es entsprechend Abb. 15 drei erlaubte Zustandsübergänge $\Delta s = 0 \dots 2$.

Man erkennt, dass dieser Algorithmus eine Zeitkomplexität von $T \cdot S$ Vergleichen zwischen akustischen Vektoren und Referenzvektoren aufweist.

5.2 Fast-Match-Phonemmodell

Indem man das Phonemmodell weiter vereinfacht, lässt sich die Zeitkomplexität auf T Vergleiche herunterdrücken.

Dafür müssen jedoch die Zustände $s = 1 \dots S(\alpha)$ in einem einzigen Zustand $\hat{s}$ zusammengefaßt werden. Dies geschieht, indem man die Emissionswahrscheinlichkeiten des HMM von α nach oben hin abschätzt:

$$p(x_t|s_t, \alpha) \;\leq\; \max_{s \in \{1 \dots S(\alpha)\}} p(x_t|s, \alpha) \;=:\; p(x_t|\hat{s}, \alpha) \tag{13}$$

Weiterhin wird die Transitionswahrscheinlichkeit durch den besten aller kompletten Pfade s_1^T nach oben abgeschätzt:

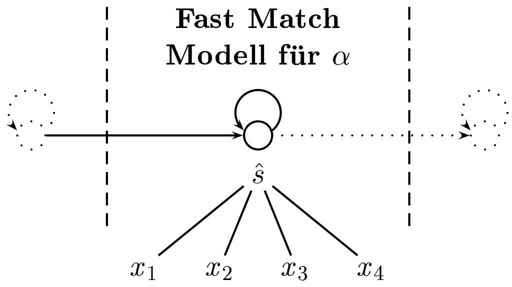


Abbildung 16: Fast Match Modell eines Phonems

$$\max_{s_1^T} p(s_1^T|\alpha) \;\leq\; \max_T \max_{s_1^T} \; p(s_1^T|\alpha) \;=:\; p(\hat{s}|\alpha) \tag{14}$$

Auf diese Weise erhält man eine obere Schranke für die gesamte detaillierte Wahrscheinlichkeit, die sogenannte *Fast-Match-Wahrscheinlichkeit* $F(x_1^T|\alpha)$:

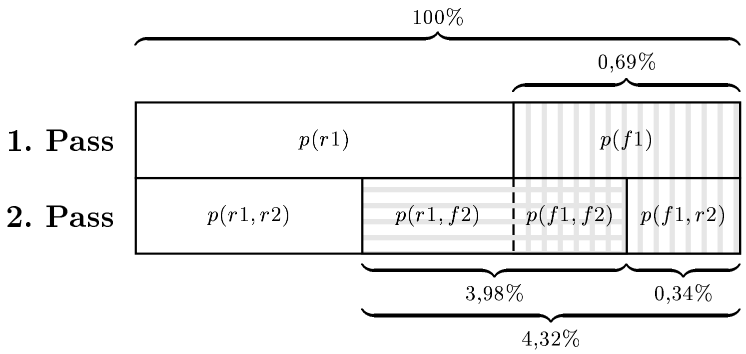
$$\begin{aligned} D(x_1^T|\alpha) &= \max_{s_1^T} \; p(s_1^T|\alpha) \cdot \prod_{t=1}^{T} p(x_t|s_t,\alpha) && (15) \\ &\leq \max_{s_1^T} \; p(\hat{s}|\alpha) \cdot \prod_{t=1}^{T} p(x_t|\hat{s},\alpha) \;=:\; F(x_1^T|\alpha) && (16) \end{aligned}$$

Pseudocode für die Berechnung von $F(x_1^T|\alpha)$ gemäß Abb. 16:

```
procedure FastMatch(x_1 ... x_T , α):F(x_1^T|α)
begin
  for t := 1 to T do
      q(t) := q(t − 1) · p(x_t|ŝ, α)
  end
  return F(x_1^T|α) := q(T) · p(s|α)
end
```

Das dazugehörige Phonemmodell (Abb. 16) kann nur noch als 'Pseudo Markov Modell' bezeichnet werden, weil die Summe der Transitionswahrscheinlichkeiten ausgehend vom Knoten $\hat{s}$ größer als 1 ist.

Weil die Fast Match Bewertung unabhängig von der Reihenfolge der Knoten ist, ist es offensichtlich, dass die Anwendbarkeit der Approximation von der Größe der akustischen Einheit abhängt. Mit der Wahl von Phonemen als akustischer Einheit können zwar keine zeitliche Variationen innerhalb des Phonems modelliert werden, man kann aber noch weiterhin den Phonemen eine Reihenfolge innerhalb des Wortes zuordnen. Die Anwendung des Fast Match Modells auf Wortebene macht hingegen keinen Sinn, weil eine solche Approximation selbst für einen ersten Pass zu grob wäre. Man könnte in diesem Fall z.B. nicht einmal einen Unterschied zwischen den Wörtern 'Erde' und 'der' feststellen.



(Notation: r: richtig, f: falsch; 1: erster Pass, 2: zweiter Pass)

Abbildung 17: Zusammensetzung der Fehlerrate im IBM-Erkenner

Tabelle 1: Rang der korrekten Wörter in der Fast-Match-Liste des IBM-Erkenners

	∅-Länge der Liste	∅-Rang des korrekten Worts	Prozentsatz der Worte mit Rang			
			$= 1$	> 1	> 10	> 100
AC	1736	14.61	46.8	53.2	15.53	2.92
AC+LM	47.95	1.88	86.39	13.61	2.84	0.69

Der Fast Match Algorithmus kann benutzt werden, um sowohl eine feste als auch eine variable Anzahl von Timeframes in die Zukunft zu schauen und eine Vorauswahl zu startender Hypothesen zu treffen.

Im i6-Spracherkenner [3] findet sich eine Variante des Fast Matches mit festem Vorschau-Fenster in der Baumsuche integriert. Mit einem kontext*unabhängigen* Phonemmodell mit einem einzigen Zustand wird dabei ein Lookahead von genau 6 Timeframes bzw. 60ms auf die zu startenden kontext*abhängigen* Hidden Markov Modelle mit 6 Zuständen erzeugt. Auf diese Weise wird ein frühzeitigeres Pruning ermöglicht, was den Erkennungsvorgang bei gleicher Fehlerrate um etwa den Faktor 2 beschleunigt.

Der IBM-Spracherkenner [2] [4] benutzt hingegen das Fast Match, um ein ganzes Wort nach vorne zu schauen. Durch die damit verbundene variable Größe des Vorschau-Fensters ergibt sich jedoch ein Problem: da an die Bewertung zu jedem Timeframe eine Wahrscheinlichkeit ≤ 1 multipliziert wird, erhalten längere Wörter grundsätzlich eine schlechtere Bewertung als kurze.

Das Problem kann teilweise beseitigt werden, indem man eine von der Länge der Wörter abhängige Normalisierung der Scores vornimmt. Als Normalisierungsfaktor für Worthypothesen der Länge t kann z.B. der Reziprokwert der Bewertung der besten Hypothese mit Länge t genommen werden.

In Abb. 17 ist die Zusammensetzung der Gesamtfehlerrate im IBM-System graphisch dargestellt. Von der Gesamtfehlerrate 4,32% gehen dabei 0,34% allein auf das Konto des Fast Matches. Die Kandidatenliste, die maximal 100 Wörter umfasst, beinhaltet mit 99,31% das korrekte Wort.

In Tabelle 1 sind die von IBM erzielten Ergebnisse für das Fast Match aufgeführt. Die Zeile 'AC' beinhaltet das Ergebnis eines reinen akustischen Fast Matches. Die Zeile 'AC+LM' beinhaltet das Ergebnis, das man bei einer anschließenden Anwendung eines Unigramm-Modells auf die Fast-Match-Liste erhält. Die Wahrscheinlichkeit, dass sich das korrekte Wort nicht in der Liste befindet, beträgt im ersteren Fall 0,11% und im letzteren Fall 0,69%. Dafür ist aber auch die Liste im ersteren Fall 36 mal so lang.

6 Architektur zweier Multi-Pass-Systeme

6.1 BBN Byblos: Akustisches Rescoring mit Sprecheradaption

Neben der in Kap. 5 erläuterten Fast-Match-Suche für ein akustisches Front-End des Spracherkenners gibt es auch mehrere Möglichkeiten, das System „nach hinten“ zu erweitern.

Das BBN Byblos 2000 System [7] ist ein Paradebeispiel für ein Multi-Pass-System und zeigt viele Ideen für ein akustisches Rescoring auf. Die akustische Erkennung wird in diesem System in nicht weniger als sieben (!) Durchläufen vorgenommen:

1. Forward-Pass; keine Acrossword-Modelle und keine kontextabhängigen Phonemmodelle, Bigramm-Sprachmodell; Zwischenrepräsentation: Wortgitter

2. Backward-Pass; wie 1., aber mit Trigramm-Sprachmodell; 1. und 2. bilden eine Forward-Backward-Suche

3. Forward-Pruning mit Trigramm-Sprachmodell über dem Wortgitter

4. Forward-Pass; diesmal mit Acrossword-Modellen und kontextabhängigen Phonemmodellen (Phonemmodelle mit State-Tying mittels CART); weiterhin Trigramm-Modell; erzeugte Zwischenrepräsentation: N-Best-Liste mit N=100

5. akustisches Rescoring mit sprecherabhängigen Modellen (auf Basis der Maximum Likelihood Linear Regression (MLLR)-Methode mit 4 Transformationen); Parts-Of-Speech (POS) - geglättetes Trigramm-Modell

6. MLLR-Methode wie 5., aber mit 8 Transformationen und sprecherunabhängigen Modellen

7. MLLR-Methode wie 5., aber mit 8 Transformationen und sprecherabhängigen Modellen

6.2 AT&T: 6-gramm-Sprachmodell-Rescoring

Ein von AT&T entwickeltes System [1] besitzt hingegen einen anderen Systemaufbau:

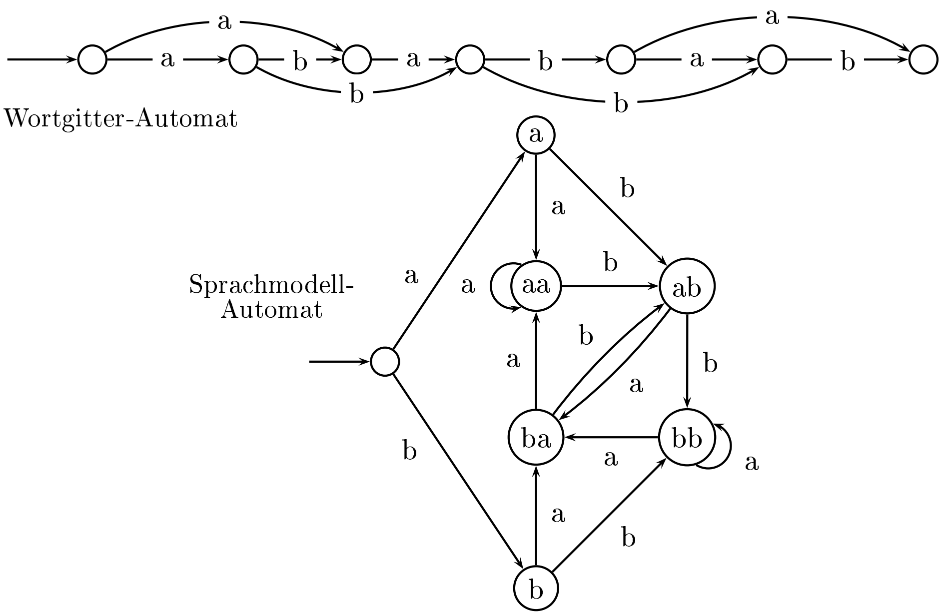


Abbildung 18: Produktautomaten-Modell für das LM-Rescoring

1. Erzeugung eines Phonemgitters mit kontextabhängigen Phonemen mittels Transducer-Viterbi-Suche
2. Konvertierung des Phonemgitters in ein Wortgitter; Bigramm-Sprachmodell mit 3,6 Mio. Bigrammen
3. Forward-Backward-Pruning des gesamten Wortgitters
4. Rescoring des Wortgitters mit einem 6-gramm-Sprachmodell mit 20 Mio. m-Grammen

Mit dem letzten Schritt, dem Rescoring des Wortgitters mit einem 6-gramm-Sprachmodell, wird im AT&T-Erkenner eine Reduzierung der Fehlerrate von 19% auf 11% erreicht.

Ein solches Sprachmodell-Rescoring (kurz: LM-Rescoring, LM steht für Language Model) läßt sich veranschaulichen durch einen probabilistischen Produktautomaten, der, wie in Abb. 18 dargestellt, aus dem Wortgitter und dem Automaten des zugehörigen m-Gramm-Modells gebildet wird.

Die beiden Automaten werden in jedem Schritt parallel durchlaufen. Dabei wird eine Menge von Zustandspaaren (s,t) generiert, die aus einem Zustand s im LM-Automaten und einem Zustand t im Wortgitter (im einfachsten Fall einem Timeframe) bestehen.

Die Wahrscheinlichkeit $p(s,t)$ ergibt sich aus dem Produkt der Wahrscheinlichkeiten entlang der beiden Pfade in den jeweiligen Automaten. Dabei wird die akustische Wahrscheinlichkeit von dem Wortgitter-Automat beigesteuert, die LM-Wahrscheinlichkeit hingegen vom LM-Automaten.

Wenn in einem Schritt mehrere Zustandspaare (s,t) auftauchen, die sich nur hinsichtlich ihrer Pfade und ihrer Wahrscheinlichkeiten $p_1, p2, \ldots$ unterscheiden, so werden diese zusammengefasst zu einem einzigen Paar, dessen Wahrscheinlichkeit entweder über die Summe oder das Maximum von $p_1, p2, \ldots$ gebildet wird.

7 Fazit

Wir haben Multi-Pass-Systeme als eine Art mehrstufiges Filtersystem für Phonem- oder Worthypothesen kennengelernt.

Multi-Pass-Systeme haben vor allem den Vorteil der höheren Effizienz gegenüber 1-Pass-Systemen, weil sie mit einfachen Modellen eine grobe Vorauswahl für die näher zu untersuchenden Hypothesen treffen. Als Algorithmus für eine grobe Vorauswahl haben wir das akustische Fast-Match kennengelernt.

Ein positiver Nebeneffekt von Multi-Pass-Systemen ist die bessere Strukturierung des Suchproblems, die bei der Programmierung und Analyse des Systems sehr hilfreich ist.

Als Schnittstellen kommen vor allem die N-Best-Liste und die Lattices in Betracht. Dazu haben wir uns angeschaut, welche möglichen Ansätze es gibt, die 1-Best-Suche dahingehend zu ändern, die N besten Hypothesen auszugeben.

Neben den vielen Vorteilen, die die Multi-Pass-Suche bietet, gibt es nur einen wesentlichen Nachteil: die eventuelle Verzögerung der Ausgabe eines solchen Systems. Rescoring-Algorithmen, die erst auf vollständigen Sätzen operieren können, wie z.B. die Forward-Backward-Suche besitzen leider diesen Nachteil. Für manche Anwendungen der Spracherkennung ist diese Verzögerung störend, so dass mit der zunehmenden Leistung der PCs der Einsatz dieser Rescoring-Algorithmen trotz ihrer Überlegenheit in puncto Effizienz fraglich sein wird.

Viele Rescoring-Algorithmen lassen sich jedoch im Gegensatz zur Forward-Backward-Suche dahingehend modifizieren, schon während der groben Erkennung des Satzes mit einer detaillierten Suche zu beginnen und nicht erst, wenn das Satzende erreicht ist. Damit wäre dann der einzige schwerwiegende Nachteil der Multi-Pass-Suche ausgeräumt.

Literatur

[1] Andrej Ljolje, Fernando Pereira, Michael Riley, *Efficient General Lattice Generation and Rescoring*, Tech. report, AT&T Labs Research, 1999.

[2] P.S. Gopalakrishnan L.R. Bahl, S.V. De Gennaro, *A Fast Approximate Acoustic Match for Large Vocabulary Speech Recognition*, IEEE Trans. on Speech and Audio Processing, vol. 1, January 1993, pp. 59–67.

[3] H. Ney, *Ausarbeitung zur Vorlesung Spracherkennung*, Tech. report, Lehrstuhl für Informatik VI, RWTH Aachen, 2001.

[4] L.R. Bahl P.S. Gopalakrishnan, *Fast Match Techniques*, Automatic Speech and Speaker Recognition (C.-H. Lee, F.K. Soong, and K.K. Paliwal, eds.), Kluwer Academic Publishers, 1996, pp. 413–428.

[5] J. Makhoul R. Schwartz, L. Nguyen, *Multiple-Pass Search Strategies*, Automatic Speech and Speaker Recognition (C.-H. Lee, F.K. Soong, and K.K. Paliwal, eds.), Kluwer Academic Publishers, 1996, pp. 429–435.

[6] E.G. Schukat-Talamazzini, *Automatische Spracherkennung*, Vieweg-Verlag, 1995.

[7] Thomas Colthurst, Owen Kimball, Fred Richardson, Han Shu, Chuck Wooters, Rukmini Iyer, Herbert Gish, *The 2000 BBN Byblos LVCSR System*, Tech. report, GTE/BBN Technologies, 2000.

[8] F. Alleva M. Hwang L. Jiang M. Mahajan X. Huang, A. Acero, *From SPHINX-II to Whisper - Making Speech Recognition Usable*, Automatic Speech and Speaker Recognition (C.-H. Lee, F.K. Soong, and K.K. Paliwal, eds.), Kluwer Academic Publishers, 1996, pp. 481–508.