POLITECNICO DI TORINO UNIVERSITÄT KARLSRUHE (TH)

Fakultät für Informatik

Studienarbeit

Wordspotting-Techniken

Betreuer:
Prof. Alex Waibel
Prof. Pietro Laface
Dipl.-Phys. Monika Woszczyna

Autor:
Peter Scheytt

Januar 1995

Dank an (in alphabetischer Reihenfolge)
Adrian, Àngels, Alex, Ana, Claudio, Cristina, Daniele & Daniele & Daniele, Else, Erika, Georgina, Gianni, Guillermo, Julius, Luana, Marylène, Melanie, Monika, Oliver, Pietro, Pietro, Robin, Rudi, Silvio, Sonia, Tyler, Werner.

Inhalt

Verzeichnis der Abbildungen

Verzeichnis der Tabellen

Kapitel 1

Einführung

Das gebräuchlichste Mittel zur gegenseitigen Verständigung zwischen Menschen dürfte Wohl die gesprochene Sprache sein. Mittels ihrer kann sich der einzelne am einfachsten und sichersten ausdrücken. Der Wunsch, eine Maschine auch mit gesprochenen Worten steuern zu können, liegt deshalb nicht allzufern, denn das würde auf vielen Gebieten zu einer erheblichen Erleichterung der Aufgaben führen (wer kann schon mit einer Tastatur umgehen?). Schon kurz nach Kriegsende wurde mit der Forschung auf diesem Gebiet begonnen, das sich als äußerst harte Nuß erwies. Denn was für das menschliche Gehirn eine leichte Übung ist, stellt für einen Rechner ein schwer zu lösendes Problem dar. Vor allem der Umfang der zu verarbeitenden Datenmenge erwies sich in der Vergangenheit als eine hohe Hürde auf dem Weg zur Lösung der Schwierigkeiten. Doch mit dem gewaltigen Anwachsen der Rechnerleistung in den vergangenen Jahren ist ein Silberstreif am Horizont zu entdecken, und zumindest die Erfüllung eines Teils des ursprünglichen Wunsches in greifbare Nähe gerückt.
Die Spracherkennung läßt sich im wesentlichen in zwei Schritte unterteilen:

- Die Erfassung des akustischen Signals und dessen Umwandlung in eine Form, die einem Computer eine weitere Behandlung ermöglicht.
- Die Bearbeitung der in der Signalerfassung gewonnen Daten mittels geeigneter Modelle, die schließlich die Erkennung des Sprachsignals erlauben.

Im zweiten Kapitel wollen wir zunächst die grundlegenden Prinzipien der Signalerfassung vorstellen, das dritte Kapitel behandelt die sogenannten HMM (**H***idden* **M***arkov* **M***odels*), die eine Möglichkeit darstellen, Sprache mit Hilfe eines stochastischen Ansatzes zu modellieren. In den letzten beiden Kapiteln geht es um ein Spezialgebiet der Spracherkennung, dem **Wordspotting**,

d. h. dem Auffinden von Schlüsselworten innerhalb eines Satzes oder Textes. Zunächst geben wir eine kurze Einführung in dieses Thema und stellen dann eine Implementation eines Wordspotters vor.

Kapitel 2

Signalerfassung

2.1 Einführung

Will man natürliche Ereignisse eines beliebigen Typs von einer Maschine untersuchen lassen, ist es zunächst notwendig, die in dem physikalischen Phänomen enthaltenen Informationen in eine für den Rechner verwertbare Form aufzubereiten.
Bei der Sprachverarbeitung heißt das, daß die Sprachwellen zuerst in ein elektrisches Signal umgewandelt werden müssen, aus dem danach die parametrische Darstellung der Daten gewonnen wird. Die so erhaltenen Werte sollten dabei die in den Eingangsdaten enthaltenen Informationen widergeben. Der gesamte Prozeß der Signalanalyse ist in 2.1 dargestellt.

2.2 Abtastung und A/D-Wandlung

Im allgemeinen wird das vom Mikrofon bereitgestellte analoge elektrische Signal, mit dem wir die Funktion $g(t)$ verbinden wollen, zuerst tiefpaßgefiltert. Dadurch verhindert man, daß es bei der Abtastung zu Überlappungen (aliasing) kommt. Das Theorem von NYQUIST schreibt hierbei vor, daß die Abtastfrequenz mindestens doppelt so hoch sein muß wie die höchste auftretende Frequenz im Sprachsignal, die sogenannte obere Grenzfrequenz.

Theorem 1 (NYQUIST)

$$\Delta x \leq \frac{1}{2W}$$

wobei Δx die Zeit zwischen zwei Abtastungen und W die obere Grenzfrequenz ist.

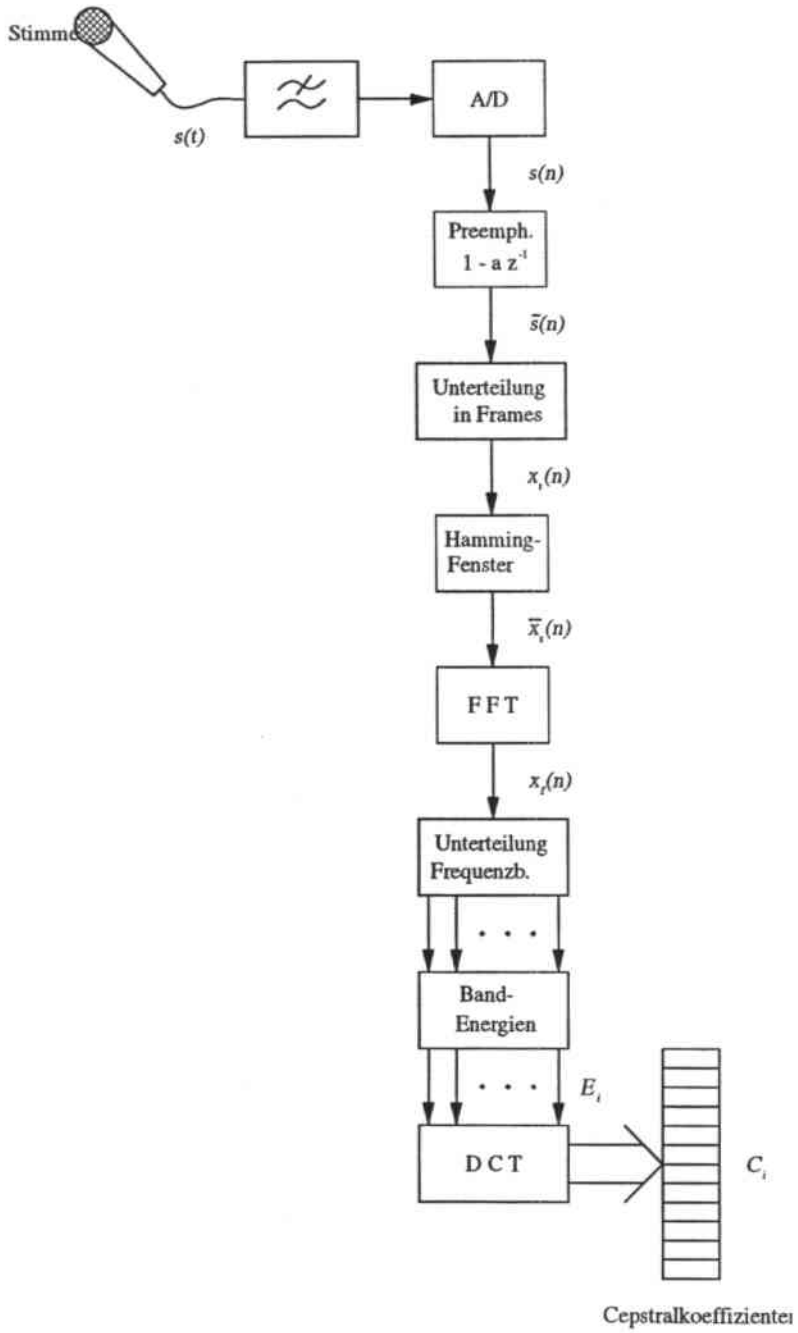

Abbildung 2.1. Schritte der Signalerfassung.

Im Falle eines Telefongesprächs, in dem Frequenzen zwischen 100 und 3400 Hz auftreten können, müßte die Abtastfrequenz also mindestens 6800 Hz betragen, in der Regel wählt man aber 8000 Hz, bei manchen Anwendungen auch 12000 Hz.
Als Abtastfunktion $a(n)$ wird häufig die δ-(Dirac)-Funktion verwendet, sie ist wie folgt definiert:

$$\int_{-\infty}^{\infty} f(t)\,\delta(t-t_0)\,dx = f(t_0)$$

Man erhält somit das diskrete, digitalisierte Zeitsignal $s(n)$, mit

$$s(n) = g(n)\,a(n).$$

2.3 Präemphase

Um das Signal innerhalb des besetzten Bandes zu glätten, d. h. die hohen Frequenzen stärker zu betonen, wird ein digitaler Filter zwischengeschaltet, seine Übertragungsfunktion ist die folgende:

$$H(z) = 1 - az^{-1}$$

was im diskreten Falle

$$\overline{s}(n) = s(n) - as(n-1)$$

ergibt. In Anwendungen erhält a den konstanten Wert $a = 0{,}95$.

2.4 Frames, Fensterfunktion, Fouriertransformation

Die Eigenschaften des menschlichen Sprachapparates sind so beschaffen, daß sich die Stimme in kontinuierlicher Weise entwickelt. Dies legt nahe, das Sprachsignal innerhalb kurzer Zeitintervalle zu beschreiben, in denen folglich noch keine wesentlichen Änderungen zu erwarten sind. In anderen Worten heißt das, daß man das Sprachsignal als semi-stationär betrachtet.
Die Einteilung in diese kurzen Zeitabschnitte, die sogenannten Frames, wird durch eine Fensterfunktion $w(k)$ vorgenommen; daraus läßt sich dann das digitale Kurzzeitspektrum $x_t(n)$ mittels der diskreten Fouriertransformation berechnen:

$$x_f(n) = \sum_{k=n-\frac{N}{2}}^{n+\frac{N}{2}-1} s(k)\, w(n-k)\, e^{-\frac{2\pi tk}{N}},$$

wobei

- n Index über Abtastungen
- t Index über Frames
- N Anzahl der Abtastungen in einem Fenster.

In der Regel wählt man als Dauer eines Fensters die Zeit von 20 ms, man erhält somit alle 10 ms das Kurzzeitspektrum eines Frames; d. h. die einzelnen Frames überlappen sich teilweise (vergleiche auch Abbildung 2.4).

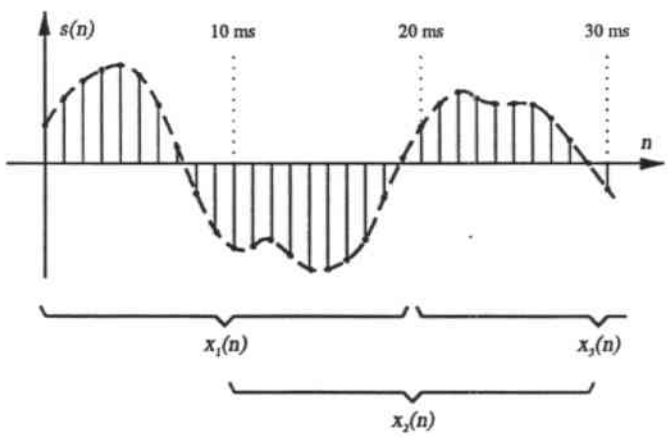

Abbildung 2.2. Einteilung der Abtastwerte in Frames der Länge 10 ms mit einer Fensterlänge von 20 ms.

Bei der Einteilung in Frames kann es zu Signalverzerrungen an den Rändern kommen (Gibbs-Phänomen); dem kann teilweise dadurch begegnet werden, daß die Fensterfunktion $w(k)$ entsprechend so gewählt wird, daß die Werte an den Framegrenzen weniger Gewicht erhalten und sich somit eine klarere Abgrenzung zum folgenden Fenster ergibt.

Anstatt eines rechtwinkligen Fensters verwendet man deshalb häufig das sogenannte Hamming-Fenster $w_H(k)$, das die oben beschriebenen Anforderungen recht gut umsetzt und wie folgt definiert ist:

$$w_H(k) = \begin{cases} a + (1-a)\cos\left(\frac{2\pi k}{N}\right) & \text{wenn } -\frac{N}{2} \leq k \leq \frac{N}{2} - 1 \\ 0 & \text{sonst} \end{cases}$$

mit $a = 0{,}54$.
Man erhält somit

$$\overline{x}_f(n) = \sum_{k=n-\frac{N}{2}}^{n+\frac{N}{2}-1} s(k)\, w_H(n-k)\, e^{-\frac{2\pi i k}{N}}$$

als Ausgabe dieser Verarbeitungsstufe.

2.5 Untergliederung in Frequenzbänder

Durch die Anwendung der FFT (Fast Fourier Transformation) hat man eine Darstellung des Signals in Abhängigkeit der Frequenz, das sogenannte Energiespektrum, erhalten. Die einzelnen Frequenzen werden nun in 18 Bänder eingeteilt; hierfür benützt man die Mel-Skala, die sich dadurch auszeichnet, daß sie unterhalb von 1000 Hz linear und oberhalb von 1000 Hz logarithmisch angeordnet ist. Diese Einteilung spiegelt gewisse Eigenschaften des auditiven Systems des Menschen wider. Die charakteristischen Frequenzen dieser Skala sind in Tabelle 2.1 dargestellt.
Für jeden Frame wird nun die Energie in Abhängigkeit des betrachteten Bandes berechnet, und zwar nach folgender Formel:

$$E_i = \sum_{j=L_i}^{U_i} |x_f(j)|^2 \qquad i = 1, \ldots, N_f$$

wobei

- N_f Anzahl der Filter
- L_i untere Grenzfrequenz des i-ten Filters
- U_i obere Grenzfrequenz des i-ten Filters.

Band No.	Cutoff Freqs. [Hz]		Center Freq. [Hz]
1	187	280	229
2	280	374	324
3	374	476	422
4	476	588	529
5	588	710	646
6	710	850	777
7	850	1009	926
8	1009	1186	1094
9	1186	1382	1281
10	1382	1606	1490
11	1606	1868	1732
12	1868	2167	2012
13	2167	2522	2338
14	2522	2942	2724
15	2942	3456	3189
16	3456	4110	3769
17	4110	4950	4510
18	4950	6071	5482

Tabelle 2.1. Grenz- und zentrale Frequenzen der digitalen Filterbank

2.6 Cepstralkoeffizienten

Schließlich erhalten wir in Form der Cepstralkoeffizienten die endgültige parametrische Darstellung des Signals. Man berechnet deren Werte, indem man eine diskrete Cosinus-Transformation (DCT) auf den Logarithmus des Energiespektrums, das innerhalb der Bänder ausgerechnet wurde, anwendet. Eine derartige Transformation nähert eine Rotation um die Hauptachsen der Spektraldarstellung an und bewirkt, daß die einzelnen Parameter unkorreliert bleiben. Die Berechnung erfolgt nach folgender Formel:

$$C_i = \sum_{j=1}^{N_f} \log[E_j] \cos\left[i\,(j - \frac{1}{2})\frac{\pi}{N_f} \right] \qquad i = 0, \ldots, N_f - 1.$$

E_j ist dabei die Energie am j-ten Ausgang einer Filterbank des Umfanges von N_f Filtern.

Die Parameter werden gemäß ihrer Varianz absteigend geordnet; in der Regel wird dann nur noch eine Untermenge (meist zwölf Parameter) benutzt, denn es ist zu bedenken, daß die Parameter mit großer Varianz nur einen geringen Beitrag zur Beschreibung des ursprünglichen Signals leisten. Den so erhaltenen Werten C_1 bis C_{12} fügt man meist noch einen Parameter C_0 hinzu, der die aufsummierte Energie aller Bänder enthält:

$$E = \sum_{j=1}^{N_f} E_j.$$

Somit ist nun die Parametrisierung des Eingangssignals abgeschlossen. Für jeden Frame fassen wir die Werte im Vektor $\vec{x}(t)$ zusammen, dessen Elemente die 13 erhaltenen Parameter sind.
Wenn man von einer Abtastfrequenz von 12000 Hz ausgeht, beträgt die Datenreduktion etwa eine Größenordnung: Man ist von 12000 Werten in der Sekunde ausgegangen und bei 1300 angekommen, 13 Parameter jede 10 ms.

2.7 Liftering und Depstralkoeffizienten

Im Sprachspektrum kann man charakteristische Leistungsspitzen an den sogenannten fundamentalen Frequenzen erkennen, sie werden auch als Formanten bezeichnet. Benützt man nun die erhaltenen Cepstralkoeffizienten, erweitert um den aufsummierten Energiewert, im weiteren Prozeß der Sprachanalyse, kann es in der Trainingsphase des Spracherkenners – das ist die Einstellung der verwendeten Modelle (siehe Kapitel 3 auf die zu verarbeitenden Daten – zu einer Polarisation um diese Formanten kommen. Dies führt in der Regel zu einer Leistungsverschlechterung des Erkenners. Das Liftering erlaubt nun diese Polarisierung zu vermindern; die erhaltenen Parameter werden mit festen Werten multipliziert und somit verschiedenartig gewichtet. Die verwendeten Werte berechnen sich im allgemeinen nach folgender Formel:

$$W_i = 1 + \frac{p}{2} \sin\left(\frac{\pi i}{p}\right) \qquad i = 1, \ldots, p$$

wobei $p = 12$, die Anzahl der verwendeten Parameter. Die neuen Parameterwerte berechnen sich dann folgendermaßen:

$$\hat{C}_i = C_i \, p \, \frac{W_i}{\sum_{j=1}^{p} W_j} \qquad i = 1, \ldots, p$$

Um die Räpresentation des Sprachsignals weiter zu verbessern, kann man zu den Cepstral- noch die sogenannten Depstralkoeffizienten hinzufügen. Sie erlauben eine Beschreibung der zeitlichen Entwicklung des Signals und berechnen sich als eine Annäherung an die Ableitung der Cepstralkoeffizienten über die Zeit und heißen deshalb auch abgeleitete Cepstralkoeffizienten. Man wählt ein Fenster der länge $2K + 1$ (gewöhnlicherweise K = 2, das Fenster umfaßt also 5 Frames, was 50 ms entspricht), das um den zu betrachtenden Frame zentriert wird und errechnet dann die Werte für die Depstralkoeffizienten mittels folgender Formel:

$$\Delta\hat{C}_i(t) = G \sum_{k=-K}^{K} k\,\hat{C}_i(t-k) \qquad\qquad i = 1, \ldots, p.$$

wobei

- t Index über Frames
- p Anzahl der verwendeten Cepstralkoeffizienten
- G Koeffizient, um die Varianz von Cepstral- und Depstralkoeffizienten etwa gleichzuhalten.

Ähnlich kann man auch mit dem Parameter $\hat{C}_0(t)$, der aufsummierten Energie vorgehen; man errechnet dann $\Delta\hat{C}_0(t)$, einen Wert, der deshalb von besonderer Bedeutung ist, weil er die Intensitätsveränderungen des Signals widergibt.
Schließlich erhält man für jeden Frame 26 Parameter, je 12 Cepstral- und Depstralkoeffizienten, die aufsummierte Energie und die abgeleitete aufsummierte Energie. Der Vektor $\vec{x}(t)$ stellt sich nun wie folgt dar:

$$\begin{aligned} \vec{x}(t) &= (x_1(t), \ldots, x_k(t)) \\ &= (\hat{C}_1(t), \ldots, \hat{C}_{12}(t); \Delta\hat{C}_1(t), \ldots, \Delta\hat{C}_{12}(t); E(t), \Delta E(t)) \end{aligned}$$

wobei t der Index über die Frames ist.

2.8 Vektorquantisierung

Mittels der Parametrisierung in Cepstral- und Depstralkoeffizienten ist es uns gelungen, die vorhandene Redundanz im Signal zu vermindern. Eine weitere Methode, die anfallende Datenmenge zu reduzieren und dabei dennoch die charakteristischen Eigenschaften der Laute zu bewahren, bietet die sogenannte Vektorquanitsierung. Sie bewirkt, daß das Eingangssignal in eine Folge von Symbolen umgewandelt wird.

Das Konzept, das der Vektorquantisierung zugrunde liegt, besteht darin, daß jedem Eingangsvektor

$$\vec{x}(t) \in S^n$$

ein Stellvertretervektor

$$\vec{y}(t) = q(\vec{x}(t)) \in A^n$$

mittels der Quantisierungsfunktion

$$q : S^n \longrightarrow A^n$$

zugeordnet wird. Entscheidend dabei ist, daß der Vektor $\vec{y}$ aus einer endlichen Menge der Kardinalität L gewählt wird, dieses Alphabet

$$A = \{ y_1, \ldots, y_L \}$$

wird auch als Codebuch (*codebook*) bezeichnet. Die einzelnen Codewörter (*codewords*) y_i werden dabei so gewählt, daß die Verzerrung zwischen $\vec{x}$ und $\vec{y} = q(\vec{x})$ möglichst gering bleibt; als Maß der Distorsion wählt man der mathematischen Einfachheit halber häufig den euklidischen Abstand

$$d(\vec{x},\vec{y}) = \sum_{i=1}^{n} |x_i - y_i|^2,$$

wobei x_i und y_i (mit $i = 1, \ldots, n$) die Komponenten der Vektoren $\vec{x}$ und $\vec{y}$ sind.
Auf diese Weise kann man jedem Eingangsvektor $\vec{x}$ das Etikett eines Codewortes (meist die Nummer im Codebook) zuteilen und erhält somit das Eingangssignal in Form einer Sequenz von Symbolen, die die jeweiligen $\vec{y}$ repräsentieren. Die Wahl des Stellvertretervektors erfolgt dabei nach der Regel der minimalen Distanz:

$$q(\vec{x}) = \vec{y}_i \quad \Leftrightarrow \quad d(\vec{x},\vec{y}_i) \leq d(\vec{x},\vec{y}_j) \qquad \forall j \neq i \quad j = 1,\ldots,L$$

Die Beschaffenheit des Codebooks ist für die Güte der Quantisierung von entscheidender Bedeutung. Die Einträge sind so zu wählen, daß das "Rauschen", d. h. der Informationsverlust zwischen Original und Repräsentant, möglichst gering ausfällt. Das Codebook sollte deshalb eine ähnliche Wahrscheinlichkeitsverteilung wie die Frames am Eingang haben.
Da meist keine Informationen über die Verteilung dieser Vektoren zur Verfügung stehen, benützt man eine große Menge von Trainingsdaten, um das Codebook zu generieren. Der LBG-Algorithmus (so benannt nach seinen Erfindern Linde, Buzo und Gray) geht dabei auf iterative Weise vor:

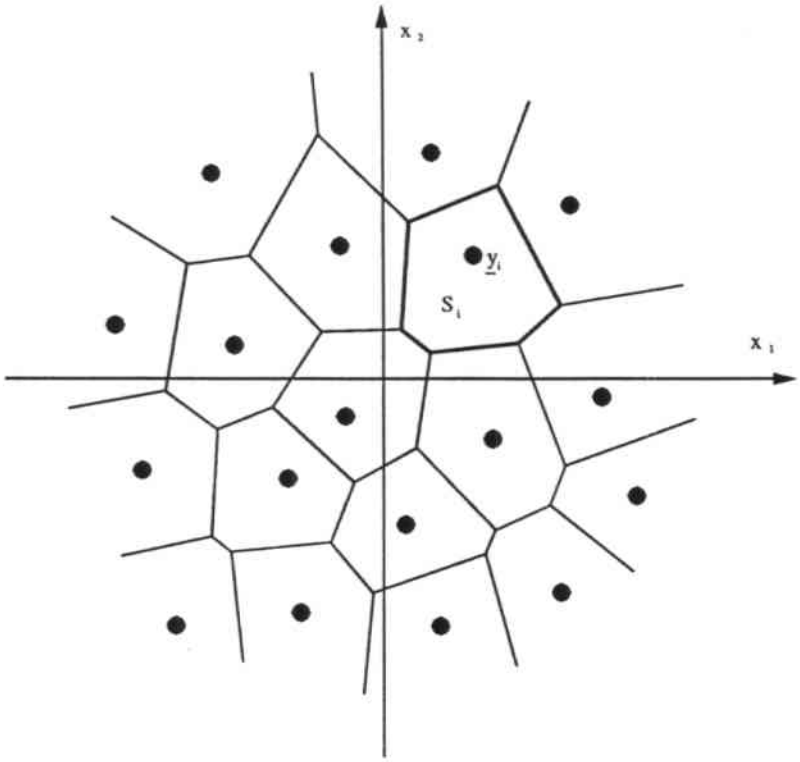

Abbildung 2.3. Unterteilung eines zweidimensionalen Emissionsraumes S^2 in Zellen. Die schwarzen Rauten stellen den Repäsentanten $\vec{y_i}$ (im Bild mit $\underline{y_i}$ bezeichnet) einer jeden Zelle dar.

1. **Initialisierung**
   Berechne $\vec{y_1}$ Zentroid aller Trainingsdaten P und
   setze $l := 1$, mit l Anzahl der Codewörter.
   Allgemein berechnet sich ein Zentroid nach der Formel:

$$\vec{y_i} = \frac{1}{P_i} \sum_{\vec{x} \in S_i} \vec{x} \, .$$

   wobei P_i die Anzahl der Vektoren ist, die in die Partitionierung $S_i \subset S^n$ fällt.

2. **Splitting**
   Mittels eines konstanten Vektors $\epsilon \in S^n$ erzeugt man aus jedem $\vec{y}_i$ zwei neue Codewörter $\vec{y}_{2i}$ und $\vec{y}_{2i+1}$ und setzt $l := 2l$.

$$\vec{y}_{2i} = \vec{y}_i + \epsilon \qquad i = 1, \ldots, l$$

$$\vec{y}_{2i+1} = \vec{y}_i - \epsilon \qquad i = 1, \ldots, l$$

3. **Clustering des Trainingssets**
   Partitionierung von P gemäß der neuen Einträge im Codebook.

4. **Bestimmung der Zentroide**
   Die Zentroide $\vec{y}_1, \ldots, \vec{y}_l$ werden neu berechnet, ebenso die Verzerrung D mit

$$D = \frac{1}{|P|} \sum_{i=1}^{|P|} d(\vec{x}(i), q(\vec{x}(i))).$$

   Ist $D \geq D^*$ (D^* bezeichnet den Wert der Verzerrung der vorhergehenden Iteration), dann fahre fort mit Schritt 3.

5. **Terminierung**
   Der Algorithmus terminiert, wenn l = L ist,
   ansonsten fahre mit Schritt 2 fort.

Auf diese Weise erhält man also ein Codebook des Umfanges L; in der Spracherkennung liegen die Werte dafür typischerweise zwischen 32 und 256.

Kapitel 3

Markov Modelle

3.1 Einführung

Im vorangehenden Kapitel haben wir gesehen, wie man die Signale, die von einem natürlichen Ereignis ausgehen, in eine maschinenverwertbare Form bringt. Nun interessiert uns, wie man die so erhaltenen Daten anhand mathematischer Modelle charakterisieren kann. Diese formale Beschreibung erfüllt im wesentlichen zwei verschiedene Aufgaben: Zum einen ist es uns möglich, Kenntnisse über dem Signal inhärente Eigenschaften und ganz allgemein über dessen Generierung zu gewinnen. Zum anderen helfen sie uns bei der Identifikation von Signalen und deren "semantischen" Gehaltes.
Man kann zwischen zwei Grundtypen von mathematischen Modellen unterscheiden, den deterministischen auf der einen Seite und den stochastischen auf der anderen. Das erste beruht auf dem Prinzip des *Pattern* oder *Template Matching*. Im Bereich der Spracherkennung besteht diese Methode darin, daß während der Trainingsphase jeder phonetischen Einheit — seien es nun einzelne Phoneme, Paare von Phonemen (*Diphone*), Tripel derselben (*Triphone*) oder, im Falle eines beschränkten Vokabulars, ganze Wörter — ein oder mehrere Elemente innerhalb eines Wörterbuches zugeordnet werden. Bei der späteren Erkennung werden dann die Eingangsdaten mit dem Prototypen verglichen (da die Dauer der beiden jedoch verschieden sein kann, müssen sie zunächst zeitlich angepaßt werden — **D***ynamic* **T***ime* **W***arping*) und es wird dasjenige Muster als zutreffend ausgewählt, das nach einem Kriterium der minimalen Distanz am nächsten liegt.
Mit dieser Methode wurden schon gute Ergebnisse erzielt. Eine Verbesserung derselben versucht man durch die Anwendung von stochastischen Modellen zu erreichen. Wir nehmen dabei an, daß sich das Ergebnis der physikalischen Signalerzeugung mittels eines Wahrscheinlichkeitsprozesses beschreiben läßt.

Hierfür eignen sich die **H***idden* **M***arkov* **M***odels*, die aus einer Zustandskette bestehen und denen zwei verschiedene stochastische Prozesse zugrunde liegen. Einer dieser Prozesse ist versteckt (daher der Name *hidden*), d.h. nicht direkt beobachtbar, und bezieht sich auf die interne Entwicklung innerhalb der Zustandskette, auf die Evolution des Modells anhand der Wahrscheinlichkeiten der Zustandsübergänge, die sich zu diskreten Zeitpunkten vollziehen. Der zweite Prozeß benützt Wahrscheinlichkeitsfunktionen, um jedem Zustand einen Wert — abhängig vom akustischen Ereignis am Eingang — zuzuordnen. Die wesentlichen Eigenschaften und Prinzipien eines solchen HMM sollen nun genauer beschrieben werden.

3.2 Elemente eines HMM

Ein HMM läßt sich durch folgende Punkte charakterisieren:

1. **N**,
   die Anzahl der Zustände des Modells. Auch wenn diese Zustände versteckt sind, so besteht doch ein gewisser Zusammenhang zwischen ihnen und dem Ereignis, das sie beschreiben (in der Spracherkennung korrespondiert eine Gruppe von Zuständen oft mit der Modellierung eines Phonems).
   Im allgemeinen gibt es Übergänge zwischen allen vorkommenden Zuständen; ein solches Modell bezeichnet man als *ergodisch* (siehe Abbildung 3.1). Später werden wir noch von anderen Typen hören, die für die Spracherkennung wichtiger sind.
   $S = \{ S_1, \ldots, S_N \}$ sei die Menge der Zustände, q_t bezeichne den Zustand zum Zeitpunkt t, mit $t = 1, \ldots, N$.

2. **M**,
   die Anzahl der möglichen Ausgabewerte in einem Zustand, diese korrespondieren direkt mit dem Output des physikalischen Prozesses. Die Menge der einzelnen Symbole wollen wir als $V = \{ \vec{v}_1, \ldots, \vec{v}_M, \}$[1] bezeichnen.

[1] In der Regel gilt, daß $V \subset \mathbf{R}^n$ ist.

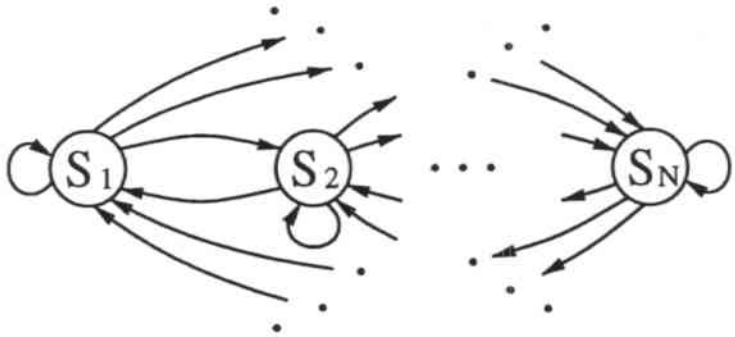

Abbildung 3.1. Graphische Darstellung der Zustände eines HMM.

3. $\mathbf{A} = (\mathbf{a_{ij}})$,
die Wahrscheinlichkeitsverteilung der Zustandsübergänge, wobei

$$a_{ij} = P(q_{t+1} = S_j \,|\, q_t = S_i) \qquad i,j = 1, \ldots, N$$

und

$$\sum_{j=1}^{N} a_{ij} = 1 \qquad \forall i$$

mit

$$a_{ij} \geq 0 \qquad \forall i,j.$$

4. $\mathbf{B} = (\mathbf{b_j}(\vec{\mathbf{v}}_\mathbf{k}))$,

die Wahrscheinlichkeitsverteilung der Ausgabewerte, wobei

$$b_j(\vec{v}_k) = P(\vec{v}_k \; zum \; Zeitpunkt \; t \,|\, q_t = S_j) \qquad \begin{matrix} j = 1, \ldots, N \\ k = 1, \ldots, M. \end{matrix}$$

d. h. $b_j(\vec{v}_k)$ gibt die Wahrscheinlichkeit an, daß $\vec{v}_k$ zum Zeitpunkt t ausgegeben wird und sich das System dabei im Zustand j befindet.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & 0 & 0 \\ 0 & a_{32} & a_{33} \end{bmatrix}$$

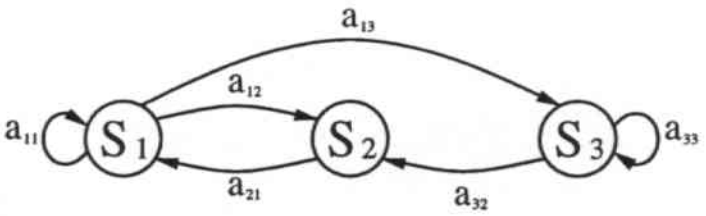

Abbildung 3.2. Übergangsmatrix für ein HMM mit drei Zuständen.

Diese diskrete Wahrscheinlichkeitsverteilung gilt im Falle, daß die Menge der Ausgabewerte endlich ist (Codebook, siehe Abschnitt 2.8)[2].

Für manche Anwendungen kann es aber sinnvoll oder gar zwingend sein, eine kontinuierliche Wahrscheinlichkeitsverteilung der Ausgabewerte zu benutzen; unter anderem dann, wenn die Informationsreduktion aufgrund einer Quantifizierung nicht akzeptabel wäre.

In diesem Falle gilt dann:

$$b_j(\vec{v}) = \sum_{m=1}^{M} c_{jm}\, \mathcal{N}(\vec{x};\, \vec{\mu}_{jm}, \mathbf{U}_{jm}) \qquad j = 1, \ldots, N$$

b_j ist die Emissionswahrscheinleichkeit des Vektors $\vec{v} \in V \subset \mathbf{R}^n$ im Zustand j, und berechnet sich als Linearkombination aus M Funktionen[3] $\mathcal{N}$, die entweder log-konkave oder ellyptisch-symmetrische Dichten haben. Mittelwertvektor ist $\vec{\mu}_{jm}$, $\mathbf{U}_{jm}$ ist Kovarianzmatrix, die c_{jm} sind die sogenannten Mischungsgewichte (mixture gains) der m-ten Mischung im j-Zustand. Gewöhnlich verwendet man Gauß- oder Laplace-Dichten.

[2] Durch die Verwendung eines Codebooks (siehe 2.8) wird dies beispielsweise gewährleistet; hierbei reduziert sich der Ausgabevektor $\vec{v}_k$ sogar auf einen skalaren Wert, weil nur noch das ihn repräsentierende Etikett aus dem Codebook widergegeben wird.

[3] M aus Punkt 2 erhält hier also seine neue Definition

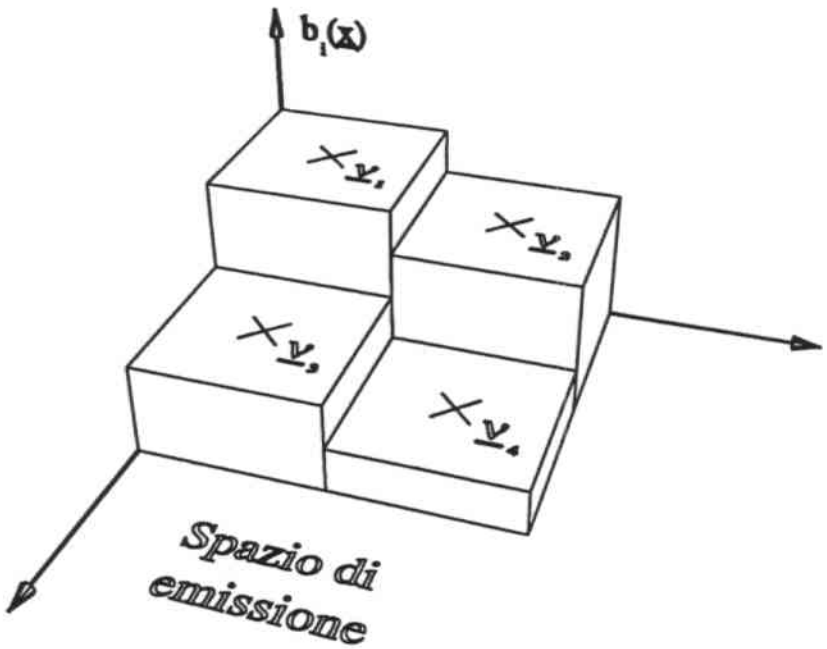

Abbildung 3.3. Emissionswahrscheinlichkeit: diskreter Fall ($\underline{v_i}$ für $\vec{v_i}$).

Weiter gilt:

$$\sum_{m+1}^{M} c_{jm} = 1, \qquad j = 1, \ldots, N$$

$$c_{jm} \geq 0, \qquad j = 1, \ldots, N, \qquad m = 1, \ldots, M$$

und

$$\int_V b_j(\vec{v})\, d\vec{v} = 1 \qquad j = 1, \ldots, N,$$

um die Wahrscheinlichkeitsdichten zu normalisieren.

5. $\vec{\pi} = (\pi_i)$
die Wahrscheinlichkeitsverteilung, zu Beginn im Zustand i zu sein, also:

$$\pi_i = P(q_1 = S_i) \qquad i = 1, \ldots, N.$$

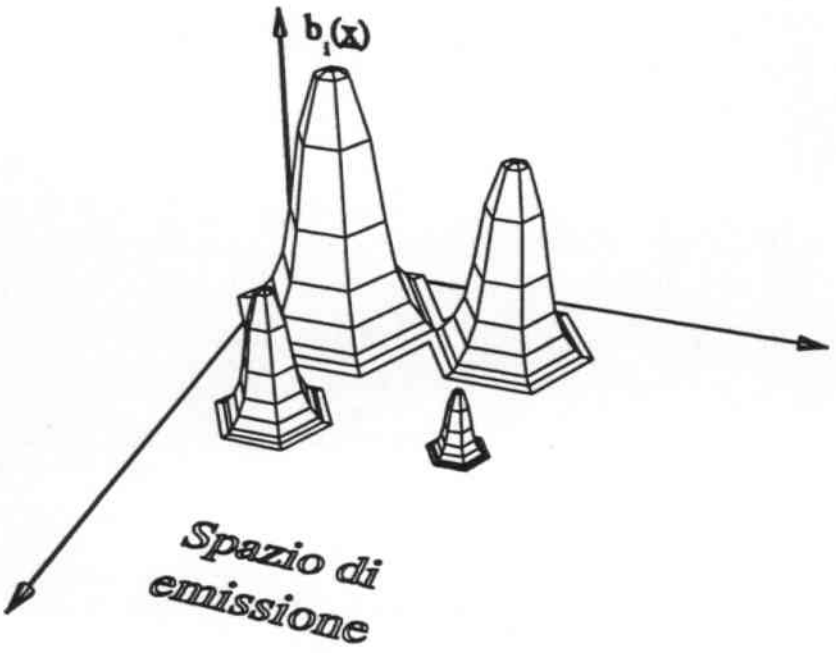

Abbildung 3.4. Emissionswahrscheinlichkeit: kontinuierlicher Fall ($\underline{x}$ statt $\vec{x}$).

Hat man nun also die Werte für N und M festgelegt, die Wahrscheinlichkeitsverteilungen A, B und $\vec{\pi}$ bestimmt, so ist ein HMM eindeutig definiert, wir wollen hierfür im weiteren die kompakte Notation

$$\lambda = (A,B,\vec{\pi})$$

verwenden.

3.3 Die drei Grundprobleme eines HMM

Nun wollen wir einen Algorithmus vorstellen, der es uns einerseits erlaubt eine zeitdiskrete Sequenz von Ausgabewerten $\mathbf{X} = \{\vec{x}_1, \ldots, \vec{x}_T\}$[4] zu erzeugen, andererseits als Modell dafür benützt werden kann, um zu rekonstruieren, wie

[4]nota bene: $\mathbf{X}$ ist ein Vektor von Vektoren

eine Ausgabefolge $\mathbf{X}$ von einem geeigneten HMM generiert wurde:

1. Wähle einen Startzustand $q_1 = S_i$ gemäß der Anfangszustandsverteilung $\vec{\pi}$;
2. Setze $t := 1$;
3. Bestimme $b(\vec{x}_t)$;
4. Übergang zum neuen Zustand $q_{t+1} = S_j$ gemäß der Übergangswahrscheinlichkeitsverteilung von S_i nach S_j, d. h. a_{ij};
5. Setze $t := t + 1$ und fahre bei Schritt 3 fort, wenn $t < T$ gilt, sonst endet die Prozedur hier.

Diese Prozedur führt uns zu den drei grundsätzlichen Problemen, die sich im allgemeinen für ein HMM ergeben:

1. Problem
   Sind die Ausgabesequenz $\mathbf{X} = \{\vec{x}_1, \ldots, \vec{x}_T\}$ und das Modell $\lambda = (A,B,\vec{\pi})$ gegeben, wie berechnet man auf effiziente Weise $P(\mathbf{X} \mid \lambda)$, die Wahrscheinlichkeit der Ausgabefolge unter der Bedingung, daß das Modell gegeben ist?
2. Problem
   Sind die Ausgabesequenz $\mathbf{X} = \{\vec{x}_1, \ldots, \vec{x}_T\}$ und das Modell $\lambda = (A,B,\vec{\pi})$ gegeben, wie wählt man eine Zustandsfolge $Q = (q_1, \ldots, q_T)$, die im Sinne eines bestimmten Kriteriums optimal ist?
3. Problem
   Wie müssen wir die Modell-Parameter $\lambda = (A,B,\vec{\pi})$ einstellen, um $P(\mathbf{X} \mid \lambda)$ zu maximieren?

Problem 1 ist auch als Evaluationsproblem bekannt, d. h. wie berechnet man bei einem gegebenem Modell und einer gegebenen Sequenz die Wahrscheinlichkeit, daß diese Sequenz vom besagten Modell erzeugt wurde. Hat man mehrere, konkurrierende Modelle so kann man den jeweiligen *Score* (die Trefferquote) dazu verwenden, um dasjenige auszuwählen, das am besten mit der Ausgabesequenz zusammenpaßt.
Problem 2 bezieht sich auf den versteckten Teil des Modells; es geht darum, eine "optimale" Reihenfolge der besuchten Zustände zu bestimmen.
Problem 3 beinhaltet die Einstellung der Parameter des HMM auf ein reelles Phänomen, um dieses möglichst genau beschreiben zu können. Die dabei verwendeten Ausgabesequenzen bezeichnet man auch als *Trainings-Set*, weil damit die Modellparameter "trainiert", d. h. möglichst optimal eingestellt werden.

3.4 Der Forward-Backward-Algorithmus

Hat man eine Sequenz $\mathbf{X} = \{\vec{x}_1, \ldots, \vec{x}_T\}$, ein Modell λ gegeben und möchte die Wahrscheinlichkeit dieser Ausgabefolge, also $P(\mathbf{X} \mid \lambda)$ berechnen, so bietet sich einem zunächst der direkte Weg an. Hierfür ist es notwendig, alle möglichen Zustandsfolgen $Q = (q_1, \ldots, q_T)$, wobei T die Länge der Ausgabesequenz ist, zu betrachten. Die gesuchte Wahrscheinlichkeit ergibt sich dann folgendermaßen:

$$P(\mathbf{X} \mid \lambda) = \sum_{\forall Q} P(\mathbf{X}, Q \mid \lambda) = \sum_{\forall Q} P(\mathbf{X} \mid Q, \lambda) \cdot P(Q \mid \lambda),$$

wobei

$$\begin{aligned} P(\mathbf{X} \mid Q, \lambda) &= b_{q_1}(\vec{x}_1) \cdot b_{q_2}(\vec{x}_2) \cdot \ldots \cdot b_{q_T}(\vec{x}_T) \\ P(Q \mid \lambda) &= \pi_{q1} \cdot a_{q_1 q_2} \cdot a_{q_2 q_3} \cdot \ldots \cdot a_{q_{T-1} q_T}. \end{aligned}$$

Die mittlere Gleichung ergibt dabei die Wahrscheinlichkeit für $\mathbf{X}$ im Falle, daß die Zustandsfolge Q vorliegt. Die letzte Gleichung stellt die Wahrscheinlichkeit für das Vorliegen eben dieses Q dar. Die Summe über das Produkt beider Wahrscheinlichkeiten liefert schließlich $P(\mathbf{X}|\lambda)$.
Betrachtet man den Rechenaufwand für diese direkte Methode, erkennt man, daß sie mit der heutigen Rechnerleistung nicht durchführbar ist. In Figur 3.5 sehen wir die möglichen Zustandpfade des Automaten. Auf der Abszisse sind die diskreten Zeitschritte aufgetragen, auf der Ordinate die erreichten Zustände des versteckten Modellteiles. Bei N Zuständen und T Beobachtungswerten sind $(2T-1)N^T$ Multiplikationen und $N^T - 1$ Additionen zu bewältigen, also insgesamt die Anzahl von

$$N^T(2T-1) + N^T - 1 = 2T N^T - 1$$

Operationen, was in die Klasse von $O(N^T)$ fällt. Wählt man $N = 5$ und $T = 100$ ergibt das einen Wert von fast 10^{72} Berechnungen. Glücklicherweise gibt es einen effizienteren Weg, Problem 1 zu lösen, den *Forward-Backward-Algorithmus.*

3.4.1 Die Forward-Prozedur

Zunächst definieren wir die *Forward-Variable* $\alpha_t(i)$ wie folgt:

$$\alpha_t(i) = P((\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_t), q_t = S_i \mid \lambda)$$

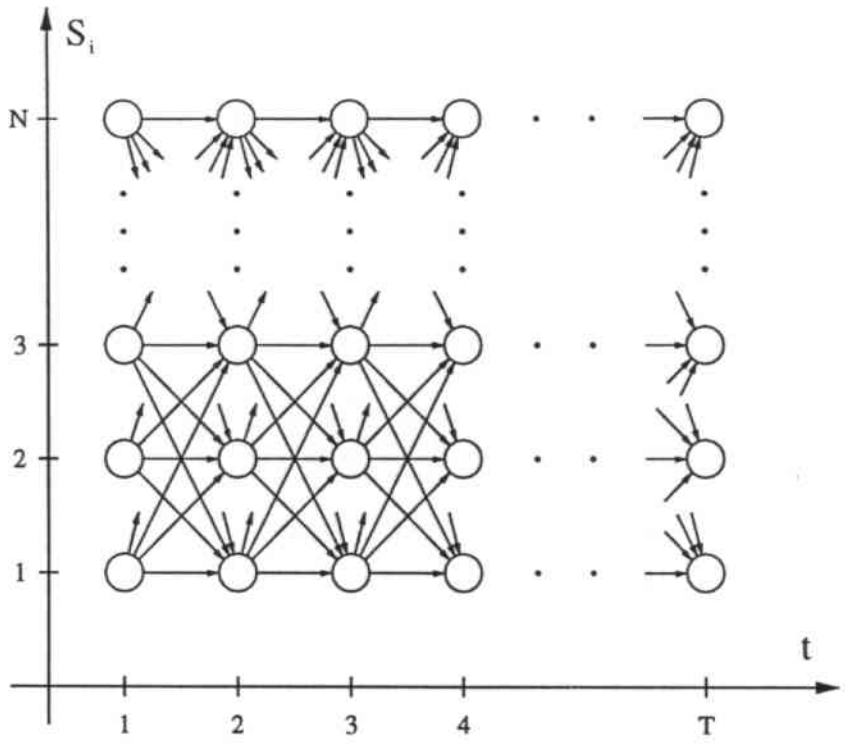

Abbildung 3.5. Forward-Backward-Algorithmus: Gitterstruktur.

Man betrachtet also die Ausgabesequenz bis zum Zeitpunkt t und erhält mit $\alpha_t(i)$ die Wahrscheinleichkeit für diese verkürzte Sequenz, wobei sich das gegebene Modell im Zustand S_i befindet.
Die $\alpha_t(i)$ lassen sich rekursiv definieren:

1. **Initialisierung**

$$\alpha_t(i) = \pi_i \, b_i(\vec{x}_1), \qquad i = 1, \ldots, N$$

2. **Induktion**

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i) \, a_{ij}\right) b_j(\vec{x}_{t+1}), \quad \begin{array}{l} t = 1, \ldots, T-1 \\ j = 1, \ldots, N \end{array}$$

3. **Terminierung**

$$P(\mathbf{X} \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

Schritt 3 wird klar, wenn man sich klarmacht, daß

$$\alpha_T(i) = P((\vec{x}_1, \ldots, \vec{x}_T), q_T = S_i \mid \lambda),$$

d. h. die Wahrscheinlichkeit ist, daß sich das Modell λ zum Endzeitpunkt T im Zustand S_i befindet. Die Summe über alle S_i ergibt nun gerade das gewünschte $P(\mathbf{X} \mid \lambda)$. In Bild 3.6 sehen wir an einem Ausschnitt aus der vollständigen Gitterstruktur (Bild 3.5) die Berechnung der Forward-Variablen dargestellt. Für diesen Algorithmus sind $N(N+1)(T-1)+N$ Multiplikationen und $N(N-1)(T-1)$ Additionen nötig, was einem Aufwand von $O(N^2)$ entspricht. Somit reduziert sich die aufzubringende Rechenleistung um einen Faktor der Größenordnung $O(N^{T-2})$. Legen wir unsere Beispiel von vorhin zugrunde, sind also nur noch rund 3000 Berechnungen durchzuführen.

3.4.2 Die Backward-Prozedur

In derselben Weise wie vorhin $\alpha_i(t)$ definieren wir nun die *Backward*-Variable

$$\beta_t(i) = P((\vec{x}_1, \ldots, \vec{x}_T) \mid q_t = S_i, \lambda)$$

d.h. die Wahrscheinlichkeit der anteiligen Ausgabesequenz zwischen $t+1$ und dem Ende, wobei sich das gegebene Modell λ zum Zeitpunkt t im Zustand S_i befindet.
Wiederum können wir $\beta_t(i)$ rekursiv lösen:

1. **Initialisierung**

$$\beta_T(i) = 1, \qquad i = 1, \ldots, N$$

2. **Induktion**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(\vec{x}_{t+1})\, \beta_{t+1}(j), \qquad t = T-1, \ldots, 1, \;\; i = 1, \ldots, N$$

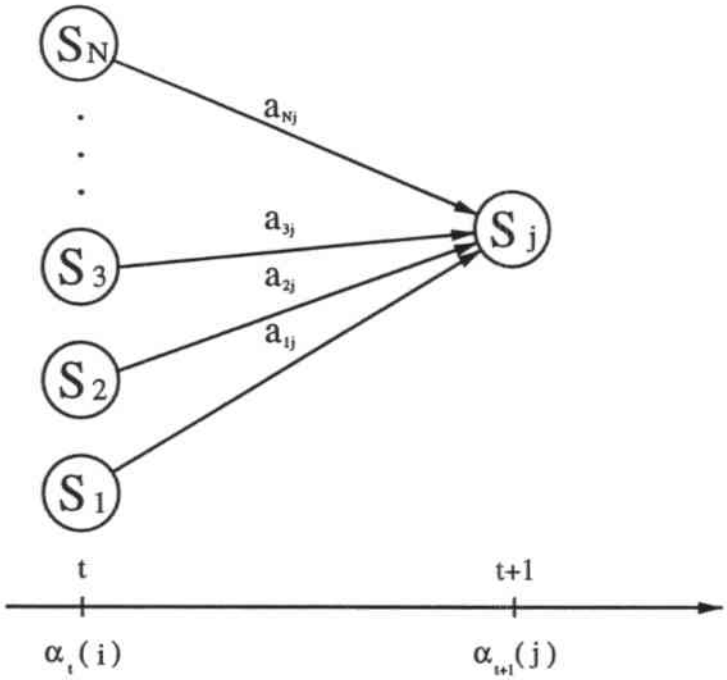

Abbildung 3.6. Berechnung der Forward-Variablen.

Daraus erhält man

$$\begin{aligned} P(\mathbf{X} \mid q_1 = S_i, \lambda) &= P((\vec{x}_2, \ldots, \vec{x}_T) \mid q_1 = S_i, \lambda) \cdot P(\vec{x}_1 \mid q_1 = S_i, \lambda) \\ &= \beta_1(i)\, b_i(\vec{x}_1) \end{aligned}$$

und schließlich, wenn man über alle Zustände $S_i \in S$ aufsummiert und $\vec{\pi}$ miteinbezieht:

$$
\begin{aligned}
P(\mathbf{X} \mid \lambda) &= \sum_{i=1}^{N} P((\vec{x}_1, \ldots, \vec{x}_T), q_1 = S_i \mid \lambda) \\
&= \sum_{i=1}^{N} P((\vec{x}_1, \ldots, \vec{x}_T), \mid q_1 = S_i, \lambda) \cdot P(q_1 = S_i \mid \lambda) \\
&= \sum_{i=1}^{N} b_i(\vec{x}_1)\, \beta_1(i)\, \pi_i
\end{aligned}
$$

Eine graphische Darstellung dieses Prozesses finden wir in Abbildung 3.7. Der Aufwand für die Backward-Prozedur fällt ebenfalls in $O(N^2)$.

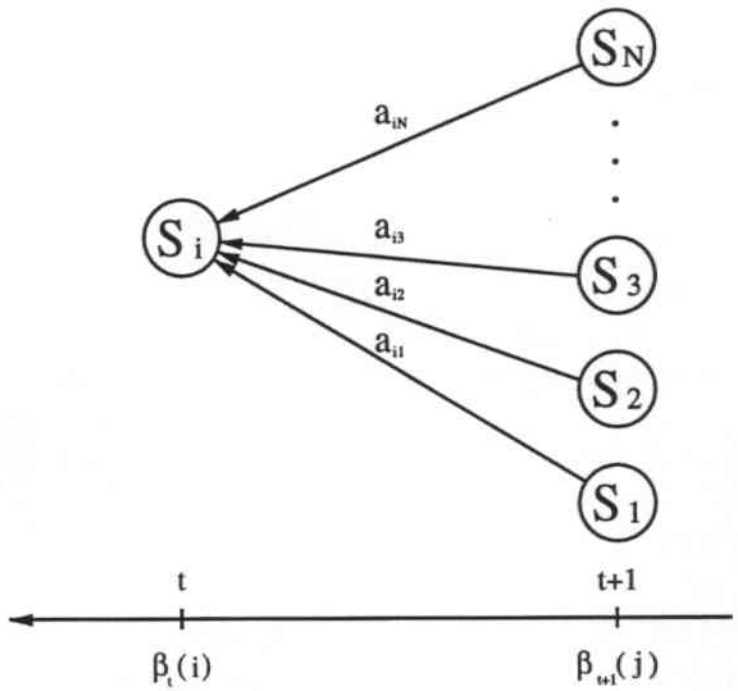

Abbildung 3.7. Berechnung der Backward-Variablen.

3.4.3 Die Forward-Backward-Prozedur

Vereinigt man nun die Ergebnisse der Forward- und der Backward-Prozedur erhält man die Emissions-Wahrscheinlichkeit der Sequenz $\mathbf{X}$ zu jedem Zeitpunkt t, bei gegebenem λ als:

$$P(\mathbf{X} \mid \lambda) = \sum_{i=1}^{N} \alpha_t(i)\, \beta_t(i), \qquad t = 1, \ldots, N$$

denn es gilt:

$$\begin{aligned} P(\mathbf{X} \mid \lambda) &= P(\vec{x}_1, \ldots, \vec{x}_T \mid \lambda) \\ &= P(\vec{x}_1, \ldots, \vec{x}_t, q_t = S_i \mid \lambda)\, P(\vec{x}_{t+1}, \ldots, \vec{x}_T \mid q_t = S_i, \lambda) \\ &= \sum_{i=1}^{N} \alpha_t(i)\, \beta_t(i) \end{aligned}$$

Wie sich später noch zeigen wird, ist der Forward-Backward-Algorithmus auch für die Lösung der Probleme 2 und 3 von großer Bedeutung.

3.5 Der Viterbi-Algorithmus

Konnte man für Problem 1 eine exakte Lösung angeben, so ist das in diesem Falle nicht möglich. Die Schwierigkeit liegt darin, das Kriterium zu bestimmen, gemäß dessen die gesuchte Zustandsfolge für eine gegebene Ausgabesequenz "optimal" sein soll. Die hier verwendete Methode beruht darauf, daß wir die Zustände q_t wählen, die einzeln, für sich gesehen am wahrscheinlichsten sind.
Hierfür definieren wir ein neue Variable

$$\gamma_t(i) = P(q_t = S_i \mid \mathbf{X}, \lambda), \qquad t = 1, \ldots, N.$$

$\gamma_t(i)$ bezeichnet die Wahrscheinlichkeit zum Zeitpunkt t im Zustand S_i zu sein, bei gegebener Ausgabesequenz $\mathbf{X}$ und gegebenem Modell λ. $\gamma_t(i)$ läßt sich mit den Forward- und der Backward-Variablen der letzten Abschnitte ausdrücken.

$$\begin{aligned}\gamma_t(i) &= \frac{P(\mathbf{X}, q_t = S_i \mid \lambda)}{P(\mathbf{X} \mid \lambda)} \\ &= \frac{\alpha_t(i)\,\beta_t(i)}{P(\mathbf{X} \mid \lambda)} \\ &= \frac{\alpha_t(i)\,\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\,\beta_t(i)}.\end{aligned}$$

Für $\gamma_t(i)$ gilt somit:

$$\sum_{i=1}^{N} \gamma_t(i) = 1, \qquad \forall t.$$

Den Zustand q_t, der für sich gesehen am wahrscheinlichsten ist, erhält man dann mittels $\gamma_t(i)$ als:

$$q_t^\star = \underset{i=1,\ldots,N}{\operatorname{argmax}} \left[\gamma_t(i)\right], \qquad t = 1, \ldots, T$$

Mit dieser Formel maximiert man die Anzahl richtiger Zustände. Jedoch kann es sein, daß die resultierende Zustandsfolge nicht zulässig ist, z. B. dann, wenn die Transitionsmatrix Null-Elemente enthält. Dieses Problem könnte man dadurch lösen, daß man ein anderes Kriterium zugrunde legt[5]. Das gebräuchliche Kriterium ist aber, die Folge der Zustände zu finden, die für sich allein genommen am besten sind, d. h. man maximiert $P(Q \mid \mathbf{X}, \lambda)$ bzw. $P(Q, \mathbf{X} \mid \lambda)$, da für bedingte Wahrscheinlichkeiten

$$P(Q \mid \mathbf{X}, \lambda) = \frac{P(Q, \mathbf{X} \mid \lambda)}{P(\mathbf{X} \mid \lambda)}$$

gilt.
Eine Methode, diese Zustandsfolge zu finden, ist der *Viterbi*-Algorithmus, der auf den Techniken der dynamischen Programmierung beruht.

Der Viterbi-Algorithmus

Um die gesuchte Zustandsfolge $Q = (q_1, \ldots, q_T)$ für die gegebene Sequenz $\mathbf{X} = (\vec{x}_1, \ldots, \vec{x}_T)$ zu finden, führen wir

[5] Denkbar wäre etwa, daß man versucht die Anzahl der korrekten Paare (q_t, q_{t+1}) oder Tripel (q_t, q_{t+1}, q_{t+2}) zu maximieren.

$$\delta_t(i) = \max_{(q_1,\ldots,q_{t-1})} P((q_1, \ldots, q_t = S_i), (\vec{x}_1 \ldots, \vec{x}_t) \mid \lambda)$$

ein. $\delta_t(i)$ gibt die beste Wahrscheinlichkeit entlang eines einzelnen Pfades zum Zeitpunkt t an, der sich über die ersten t Elemente der Sequenz $\mathbf{X}$ erstreckt und im Zustand S_i endet. Führen wir den Induktionsschritt über t durch, erhalten wir:

$$\delta_{t+1}(j) = \max_{i=1,\ldots,N} [(\delta_t(i)\, a_{ij})]\; b_i(\vec{x}(t+1)).$$

Die sich ergebende Zustandsfolge halten wir in der Matrix $\Psi = (\psi_t(j))$ fest, die für jeden Zeitpunkt t und Zustand j das maximierende Argument aus der vorangehenden Gleichung enthält. Die resultierende Gesamtprozedur stellt sich wie folgt dar:

1. **Initialisierung**

$$\begin{aligned} \delta_1(i) &= \pi_i\, b_i(\vec{x}_1) \qquad i = 1, \ldots, N \\ \psi_1(i) &= 0 \qquad\qquad\quad i = 1, \ldots, N \end{aligned}$$

2. **Induktion**

$$\begin{aligned} \delta_t(j) &= \max_{i=1,\ldots,N} [\,\delta_{t-1}(i)\, a_{ij}\,]\; b_i(\vec{x}_t) \qquad t = 2, \ldots, T,\;\; j = 1, \ldots, N \\ \psi_t(j) &= \operatorname*{argmax}_{i=1,\ldots,N} [\,\delta_{t-1}(j)\, a_{ij}\,] \qquad\quad t = 2, \ldots, T,\;\; j = 1, \ldots, N \end{aligned}$$

3. **Terminierung**

$$\begin{aligned} P^\star &= \max_{i=1,\ldots N} [\delta_T(i)] \\ q_T^\star &= \operatorname*{argmax}_{i=1,\ldots,N} [\delta_T(i)] \end{aligned}$$

4. **Resultierender Pfad** (*backtracking*)

$$q_t^\star = \psi_{t+1}(q_{t+1}^\star) \qquad t = T-1, \ldots, 1$$

Man erkennt die Ähnlichkeit zwischen diesem Algorithmus und der Forward-Backward-Prozedur; der wesentliche Unterschied liegt in der Maximierung im Induktionsschritt (es wird nur der optimale Pfad verwendet, während bei der vorhergehenden Methode alle Wege in Betracht gezogen wurden[6]). Eine schematische Darstellung des Viterbi-Algorithmus' findet sich in Bild 3.8 wieder. Der Aufwand beträgt $O(T\,N^2)$ Rechenoperationen und $O(T\,N^2)$ Vergleiche. Der Algorithmus fällt also in die Klasse $O(N\,T^2)$.

3.6 Der Baum-Welch-Algorithmus

Die Lösung des dritten Problems (der Einstellung der Modellparameter $(A,B,\vec{\pi})$, um die Wahrscheinlichkeit der Ausgabesequenz bei gegebenem Modell zu maximieren) ist mit Abstand das schwierigste. Ist eine endliche Beobachtungssequenz gegeben, gibt es keinen optimalen Weg, die Parameter zu schätzen. Jedoch kann man iterative Methoden verwenden, die zu einer lokalen Maximierung von $P(\mathbf{X} \mid \lambda)$ für ein gewähltes $\lambda = (A, B, \vec{\pi})$ führen. Im folgenden wollen wir uns das Baum-Welch-Verfahren genauer ansehen, das versucht die Wahrscheinlichkeit einer Ausgabesequenz (über alle möglichen Zustandsfolgen) über alle Modellparameter zu optimieren. Die von RABINER in [21] vorgestellte segmentielle K-means Methode versucht hingegen, die Wahrscheinlichkeit der Ausgabesequenz **und** der Zustandsfolge über alle Modellparameter zu optimieren.

3.6.1 Der Maximum-Likelihood-Ansatz

Um das Verfahren für die Neuberechnung der Modellparameter des HMM beschreiben zu können, definieren wir zunächst die Variable

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j \mid O, \lambda),$$

die Wahrscheinlichkeit zum Zeitpunkt t im Zustand S_i und in $t+1$ in S_j zu sein, wobei das Modell λ und die Ausgabesequenz $\mathbf{X}$ gegeben sind. $\xi_t(i,j)$ läßt sich mittels der Forward- und Backward-Variablen α und β darstellen. Zunächst gilt:

[6] In der Praxis hat sich aber gezeigt, daß diese Approximation zu keiner signifikanten Leistungsverschlechterung führt.

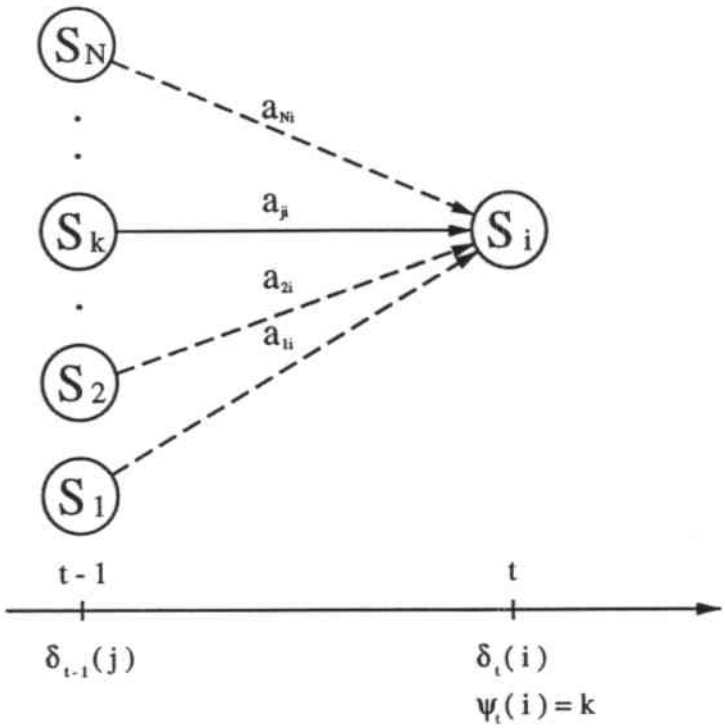

Abbildung 3.8. Viterbi-Algorithmus.

$$\begin{aligned} \xi_t(i,j) &= \frac{P(q_t = S_i, q_{t+1} = S_j, O, | \lambda)}{P(\mathbf{X} | \lambda)} \\ &= \frac{\alpha_t(i)\, a_{ij}\, b_j(\vec{x}_{t+1})\, \beta_{t+1}(j)}{P(\mathbf{X} | \lambda)}, \end{aligned}$$

wobei die Division durch $P(\mathbf{X} | \lambda)$ die gewünschte Wahrscheinlichkeitsgröße ergibt.
Im letzten Abschnitt hatten wir $\gamma_t(i)$ als Wahrscheinlichkeit bezeichnit, bei gegebenem Modell λ zum Zeitpunkt t im Zustand S_i zu sein. Summieren wir nun über alle Zustände auf, können wir zwischen $\gamma_t(i)$ und $\xi_t(i,j)$ folgende Beziehung herstellen:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j).$$

Für $\gamma_t(i)$ weiß man bereits, daß

$$\sum_{i=1}^{N} \gamma_i(t) \;=\; 1 \tag{3.1}$$

gilt, was auf $\xi_t(i,j)$ ausgedehnt,

$$\sum_{i=1}^{N}\sum_{j=1}^{N} \xi_t(i,j) \;=\; 1 \tag{3.2}$$

ergibt.
Für $\gamma_t(i)$ und $\xi_t(i,j)$ lassen sich folgende Bedeutungen finden:

$\sum_{t=1}^{T-1} \gamma_t(i)$ $\equiv$ Mittelwert der Anzahl von Transitionen, die in S_i beginnen,

$\sum_{t=1}^{T-1} \xi_t(i,j)$ $\equiv$ Mittelwert der Anzahl von Transitionen von S_i nach S_j.

Neuberechnung der Parameter

Mit diesen Gleichungen können wir die Formeln für die Neuberechnung der Modellparameter angeben und erhalten von $\lambda = (A, B, \vec{\pi})$ ausgehend das Modell $\hat{\lambda} = (\hat{A}, \hat{B}\,\hat{\vec{\pi}})$. Zu unterscheiden ist dabei noch zwischen dem diskreten und kontinuierlichen Fall.

- **Für** *diskret* **und** *kontinuierlich* **gilt:**

$$\hat{\pi}_i \;=\; \gamma_1(i)$$

$$\hat{a}_{ij} \;=\; \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$\hat{\pi}_i$ bezeichnet den Erwartungswert im Zeitpunkt $t = 1$ im Zustand S_i zu sein, $\hat{a}_{ij}$ das Verhältnis zwischen dem Erwartungswert von Transitionen von S_i nach S_j und dem Erwartungswert von Transitionen von S_i aus überhaupt. Die Normalisationbedingungen $\sum_{i=1}^{N} \vec{\hat{\pi}} = 1$ und $\sum_{j=1}^{N} \hat{a}_{ij} = 1$ werden durch die Gleichungen 3.1 und 3.2 eingehalten.

- **Für *diskret* gilt:**

  Die Formel für die fehlende Variable $b_j(k)$ ist:

$$\hat{b}_j(k) = \frac{\sum_{t=1 \mid \vec{x}_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

  $\hat{b}_j(k)$ ist das Verhältnis zwischen dem Erwartungswert im Zustand j zu sein, wobei Symbol v_k beobachtet wird, und dem Erwartungswert überhaupt im Zustand j zu sein.

- **Für *kontinuierlich* gilt:**

  Neben dem Mittelwertvektor $\vec{\mu}_{jk}$ und der Kovarianzmatrix $\mathbf{U}_{jk}$ sind auch noch die Gewichte c_{jk} neu zu bestimmen. Man erhält dann:

$$\hat{c}_{jk} = \frac{\sum_{t=1}^{T} \left\{ \gamma_t(j) \frac{c_{jk}\, \mathcal{N}(\vec{x}(t), \vec{\mu}_{jk}, \mathbf{U}_{jk})}{\sum_{l=1}^{K} c_{jl}\, \mathcal{N}(\vec{x}(t), \vec{\mu}_{jl}, \mathbf{U}_{jl})} \right\}}{\sum_{t=1}^{T} \gamma_t(j)}$$

$$\vec{\hat{\mu}}_{jk} = \frac{\sum_{t=1}^{T} \left\{ \gamma_t(j) \frac{c_{jk}\, \mathcal{N}(\vec{x}(t), \vec{\mu}_{jk}, \mathbf{U}_{jk})}{\sum_{l=1}^{K} c_{jl}\, \mathcal{N}(\vec{x}(t), \vec{\mu}_{jl}, \mathbf{U}_{jl})} \cdot \vec{x}(t) \right\}}{\sum_{t=1}^{T} \left\{ \gamma_t(j) \frac{c_{jk}\, \mathcal{N}(\vec{x}(t), \vec{\mu}_{jk}, \mathbf{U}_{jk})}{\sum_{l=1}^{K} w_{jl}\, \mathcal{N}(\vec{x}(t), \vec{\mu}_{jl}, \mathbf{U}_{jl})} \right\}}$$

$$\hat{\mathbf{U}}_{jk} = \frac{\sum_{t=1}^{T} \left\{ \left[\gamma_t(j) \frac{c_{jk}\,\mathcal{N}(\vec{x}(t), \vec{\mu}_{jk}, \mathbf{U}_{jk})}{\sum_{l=1}^{K} c_{jl}\,\mathcal{N}(\vec{x}(t), \vec{\mu}_{jl}, \mathbf{U}_{jl})} \right] (\vec{x}(t) - \vec{\mu}_{jk})(\vec{x}(t) - \vec{\mu}_{jk})' \right\}}{\sum_{t=1}^{T} \left\{ \gamma_t(j) \frac{c_{jk}\,\mathcal{N}(\vec{x}(t), \vec{\mu}_{jk}, \mathbf{U}_{jk})}{\sum_{l=1}^{K} c_{jl}\,\mathcal{N}(\vec{x}(t), \vec{\mu}_{jl}, \mathbf{U}_{jl})} \right\}}$$

Eine optimierende Folge für λ ergibt sich, wenn man die erhaltenen Parameterwerte beim nächsten Iterationsschritt in die obigen Formeln einsetzt. Bei jedem Durchgang dieser Prozedur verbessert sich somit das Modell, denn man hat ein $\hat{\lambda}$ gefunden, für das $P(\mathbf{X} \mid \hat{\lambda}) > P(\mathbf{X} \mid \lambda)$ gilt; d. h. die Wahrscheinlichkeit, daß die Ausgabesequenz $\mathbf{X}$ vom neuen Modell produziert wurde, ist größer als die Wahrscheinlichkeit, daß sie vom alten Modell erzeugt wurde. Schließlich wählt man noch ein $\epsilon > 0$ und beendet den Algorithmus, sobald

$$P(\mathbf{X} \mid \hat{\lambda}) - P(\mathbf{X} \mid \lambda) < \epsilon$$

gilt und erhält so die *Maximum-Likelihood-Schätzung* des HMM.

3.6.2 Der Maximum-Mutual-Information-Ansatz

Neben der eben geschilderten Maximum-Likelihood-Methode wird oft noch das Maximum-Mutual-Information-Verfahren verwendet. Ihr liegt als Idee zugrunde, daß man mehrere Modelle zur selben Zeit erstellen möchte und zwar auf eine Art und Weise, die die Fähigkeit eines jeden dieser Modelle maximiert, zwischen Beobachtungssequenzen, die vom richtigen Modell erzeugt wurden, und solchen, die von anderen Modellen generiert wurden, zu unterscheiden. Wir bezeichnen die unterschiedlichen HMM als λ_v mit $v = 1, \ldots, V$. Das Maximum-Likelihood-Kriterium ist nun, jeweils eine eigene Trainingssequenz $\mathbf{X}^v$ zu verwenden, um die Modellparameter für λ_v zu berechnen. Dieses Standardverfahren genügt also der Gleichung

$$P_v^\star = \max_{\lambda_v} P(\mathbf{X}^v \mid \lambda_v).$$

Beim MMI-Verfahren wird die gegenseitige Information $I_v^\star$ zwischen der Beobachtungsfolge O_v und *allen* Modellen $\lambda = (\lambda_1, \ldots, \lambda_v)$ maximiert, also

$$I_v^* = \max_{\lambda} \left(\log P(\mathbf{X}^v \mid \lambda_v) - \log \sum_{w=1 \mid w \neq v}^{V} P(\mathbf{X}^v \mid \lambda_w) \right),$$

d. h. man wählt λ so, daß es das richtige Modell λ_v am besten von allen anderen Modellen trennt, die auf die Trainingssequenz $\mathbf{X}^v$ angewendet werden. Summiert man nun noch über alle möglichen Trainingsfolgen, erhält man die Menge der am besten trennenden Modelle. Das resultierende Kriterium ist also:

$$I^* = \max_{\lambda} \left[\sum_{v=1}^{N} \left(\log P(\mathbf{X}^v \mid \lambda_v) - \log \sum_{(w=1 \mid w \neq v)}^{V} P(\mathbf{X}^v \mid \lambda^w) \right) \right].$$

Um diese Gleichung zu lösen benützt man herkömmliche Optimierungsverfahren, z. B. die Gradientenabstiegs-Methode.

3.7 Besonderheiten eines HMM bei der Spracherkennung

Das Sprachsignal verändert sich mit fortschreitender Zeit. Es wäre wünschenswert, ein Modell zu finden, das diese Eigenschaft besser repräsentiert als ein Standard-HMM. Hierfür bietet sich das sogenannte *left-right-Modell* an, bei dem die einzelnen Zustände von links nach rechts durchlaufen werden; d. h. mit zunehmender Zeit erhöht sich auch die Indexnummer der Zustände. Ein solches Modell ist durch eine Übergangsmatrix A charakterisiert, für deren Elemente a_{ij} gilt:

$$a_{ij} = 0, \quad j < i,$$

d. h. heißt, daß keine Rücksprünge zu Zuständen mit kleinerem Index möglich sind. Das Modell hat **einen** festen Startzustand und **einen** festen Endzustand. Für die Elemente des Anfangszustandsvektors $\vec{\pi}$ ergibt sich also:

$$\pi_i = \begin{cases} 1 & \text{wenn } i = 1 \\ 0 & \forall i \neq 1 \end{cases}$$

Außerdem gilt:

$$a_{Nj} = \begin{cases} 1 & \text{wenn } j = N \\ 0 & \forall j \neq N \end{cases}$$

Um zu große Sprünge innerhalb der Transitionsfolge zu verhindern, führt man gewöhnlich noch eine Begrenzungsvorschrift der Form

$$a_{ij} = 0, \text{ für } j > i + \Delta,$$

d. h. es sind keine Sprünge erlaubt, die mehr als Δ Zustände auslassen. In der Regel wählt man $\Delta = 2$ (Modell mit Skip, auch als BAKIS-Modell bekannt) oder $\Delta = 1$ (Modell ohne Skip).

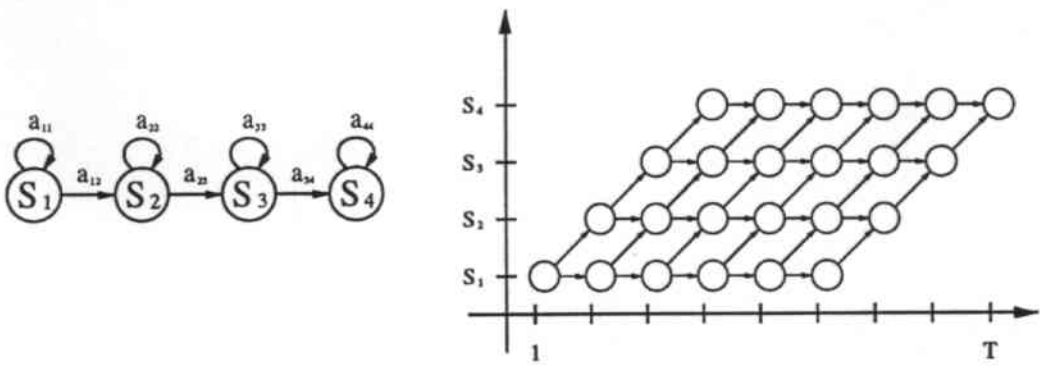

Abbildung 3.9. Links-Rechts-HMM ohne Skip mit zugehöriger Gitterstruktur.

In Abbildung 3.9 ist ein Markov-Automat mit vier Zuständen und $\Delta = 1$ sowie die zugehörige Trellis dargestellt; die Transitionsmatrix A hat folgendes Aussehen:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

Manche Modelle, die in der Spracherkennung verwendet werden, sehen das Hinzufügen eines speziellen Endzustandes, des sogenannten *dummy* oder

absorbierenden Zustandes vor, der keine Emissionsmöglichkeit besitzt und besonders als Verkettungspunkt bei verbundener Sprache nützlich ist.

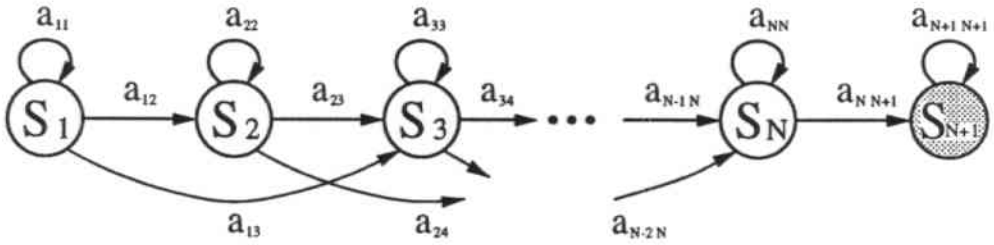

Abbildung 3.10. Links-Rechts-HMM ohne Skip mit Dummy-Zustand.

Wenn wir diesem Zustand den Index $N+1$ geben, erhalten wir folgende Eigenschaften:

$$\begin{aligned} a_{i\,N+1} &= 0 \qquad \forall i < N \\ a_{N+1\,N+1} &= 1 \\ b_{N+1}(\vec{x}) &= 0 \qquad \forall \vec{x} \end{aligned}$$

Es ist zu bemerken, daß diese besonderen Eigenschaften eines HMM keinen Einfluß auf die Neuberechnung der Modellparameter im Baum-Welch-Algorithmus haben, denn Werte die zu Anfang auf 0 gesetzt waren, bleiben während der ganzen Prozedur unverändert.

Kapitel 4

Wordspotting

4.1 Einführung

Im Gegensatz zu klassischen Spracherkennungssystemen, bei denen die gesamte Eingabe klassifiziert werden soll, geht es beim *Wordspotting* darum, ganz bestimmte Worte, die sogenannten *key words*, herauszupicken. Eine Motivation für dieses Anwendungsgebiet sollen die folgenden drei Beispiele illustrieren.

- Bereits heute bieten etliche Firmen ihren Kunden telefonische Dienstleistungen an. Der Kunde steuert dabei mit Hilfe seiner Stimme das System am anderen Ende der Leitung. Dabei wird angenommen, daß der Benutzer kooperativ ist und nur die zur Steuerung erlaubten Worte verwendet. Neuere Untersuchungen belegen aber, daß bis zu 20% des Inputs zwar das Befehlswort enthalten, jedoch oft eingehüllt in einen ganzen Satz bzw. mit Nebengeräuschen untermalt. Ein Wordspotter könnte hier nützliche Dienste leisten, indem er die Eingabe auf das Vorkommen der erlaubten Schlüsselworte untersucht.

- Ein anderes denkbares Anwendungsgebiet stellt die Klassifizierung von gesprochen Botschaften (z. B. auf einem Anrufbeantworter) dar. Hierbei geht es darum, die eingegangen Nachrichten auf bestimmte Worte hin zu untersuchen, und demnach in Gruppen einzuteilen, um eine weitere Bearbeitung zu erleichtern (Topicspotting).

- Außerdem kann man Wordspotting dazu verwenden, längere Tondokumente zu indizieren oder Stellen zur Korrektur und Änderung zu finden.

Die anfangs verwendeten Methoden basierten auf den Verfahren des *Template Matching* verbunden mit den Techniken der dynamischen Programmierung (DTW) [11]-[7]. In diesen Systemen wird ein Score dadurch berechnet, daß jedes Schlüsselwortmuster (*keyword template*) auf alle möglichen Unterteilungen der Eingabe angepaßt wird. Jeder Pfad der dynamischen Programmierung, der ein Auftreten eines Keywords bezeichnen könnte, stellt einen möglichen Treffer (*putative hit*) dar. In einem zweiten Durchgang müssen nun noch die Überlappungen zwischen diesen möglichen Treffern eliminiert werden. Außerdem ist eine Normalisierung der Wahrscheinlichkeitsscores der Pfade durchzuführen, um so eine Schranke festlegen zu können, um die wahren Treffer von den Falschalarmen trennen zu können.
In neuerer Zeit wurden gute Ergebnisse durch den Einsatz von HMM erzielt. Dieser stochastische Ansatz hat den Vorteil, daß sich die Eigenschaften des Signals besser modellieren lassen als bei der reinen Musteranpassung. In den folgenden Abschnitten wollen wir die grundlegenden Prinzipien dieser Methode genauer untersuchen.
Es sei noch darauf hingewiesen, daß auch bereits *Neuronale Netze* mit Erfolg auf diesem Gebiet eingesetzt wurden [32], worauf wir hier aber nicht näher eingehen wollen.

4.2 HMM-Modellierung

4.2.1 Aufbau eines HMM-Netzwerkes

Das HMM-Netzwerk für einen Wordspotter besteht im wesentlichen aus zwei Teilen. Der eine repräsentiert die Schlüsselworte, der andere wird aus den sogenannten Füll-Modellen [1](*filler models*) gebildet. Diese dienen dazu, Sprache zu modellieren, die nicht in die Menge der Schlüsselworte fällt. Diese Füll-Modelle spielen beim Scoring (siehe 4.3) eine große Rolle, weil sie helfen, die Rate der falsch entdeckten Keywords zu reduzieren. Wie man solche Alternativ-Modelle erstellt, werden wir später noch genauer sehen.
Der grundsätzliche Aufbau eines solchen HMM-Netzwerkes aus N Schlüsselworten und M Filler-Modellen ist in Bild 4.1 dargestellt. Es sei dabei noch auf die von ROSE in [23] eingeführten Gewichte an den Wortübergängen ($w_{k,1}, \ldots, w_{k,N}$ für Schlüsselworte und $w_{f,1}, \ldots, w_{f,M}$ für die Garbage-Modelle) hingewiesen. Diese dienen dazu, das Verhältnis zwischen nicht entdeckten Schlüsselworten und falschen Alarmen zu optimieren. Für dieses Verhältnis findet man auch den Begriff ROC (**R**eceiver **O**perator **C**haracteristic),

[1]In der Literatur findet man auch die Begriffe *Garbage-* oder *Alternativ-Modell*

der letztendlich die Qualität des Wordspotters beschreibt; seine Maßeinheit ist *falsche Alarme je Schlüsselwort je Stunde* (fa/kw/h).

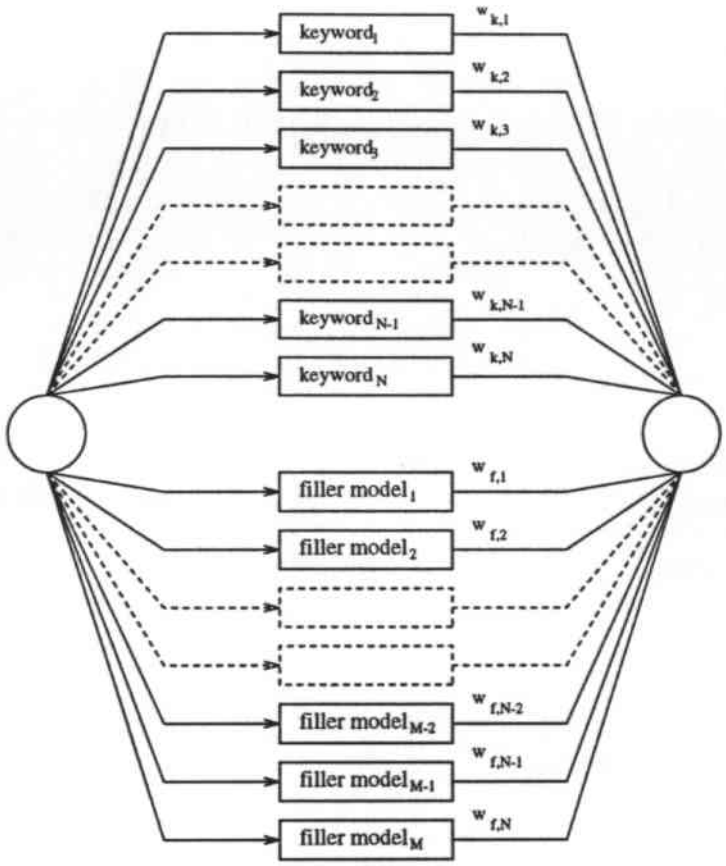

Abbildung 4.1. HMM-Netzwerk mit Schlüsselwort- und Füller-Modellen.

4.2.2 Der Schlüsselwortteil eines HMM-Netzwerkes

Für die Schlüsselworte werden sowohl Ganz-Wort- wie auch Teil-Wort-Modelle verwendet, abhängig von der beabsichtigen Anwendung. Für die HMM wählt man in der Regel eine Links-Rechts-Struktur wie in Bild 3.9 bereits dargestellt. Es gibt auch andere, kompliziertere Ansätze, bei denen versucht wird, die Dauer einzelner Wortabschnitte explizit zu modellieren, was beim einfachen Links-Rechts-Fall nur implizit durch die Verweilzeit in jedem Zustand - ihr liegt eine geometrische Verteilung zugrunde - geschieht.

Diese Methode wurde von ROHLICEK [22] vorgeschlagen. Er geht dabei von einer linearen Links-Rechts-Ordnung aus, wie sie in Bild 4.2a) dargestellt ist, wobei je 3 Zustände ein Phonem repräsentieren. Die Anzahl der Zustände wird zunächst verdreifacht, danach bildet man Gruppen aus je 3 Zuständen, die dieselbe Ausgabeverteilung haben, zuletzt wird noch die Anzahl der Verteilungen verdoppelt, so daß man schliesslich auf 18 Zustände je Phonem kommt. Ein Ausschnitt eines solchen Modelles ist in Bild 4.2b) zu sehen.

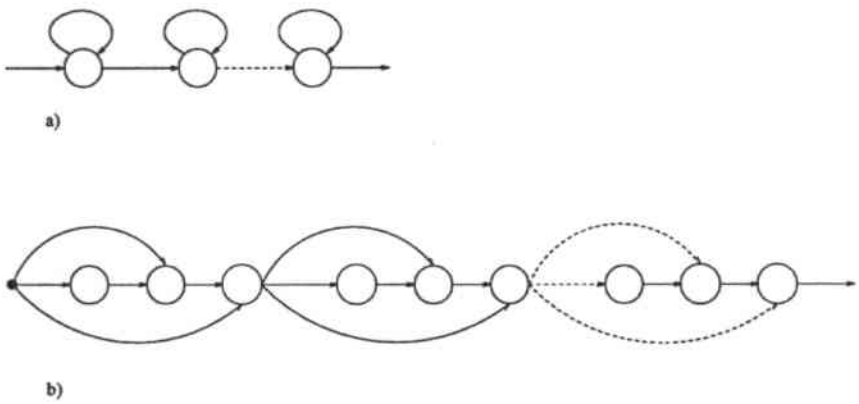

Abbildung 4.2. Schlüsselwort-Modell-Strukturen.

4.2.3 Der Garbage-Teil eines HMM-Netzwerkes

Auch hier gibt es zuerst einen einfachen Typen (Bild 4.3a), bestehend aus einem Zustand mit einer GAUSS-Mixture-Verteilung, die auf der gleichwertigen Gewichtung der Verteilungen aller Schlüsselwortzustände basiert. Die komplizierteren Modelle bestehen aus einem Netzwerk, deren Komponenten aus einzelnen Abschnitten der Schlüsselworte gewonnen wurden (Bild 4.3b) [22].
Wie man weitere Füller-Modelle — auch aus Nicht-Schlüsselwort-Sprache — erzeugen kann, werden wir im Abschnitt 4.5 genauer betrachten.

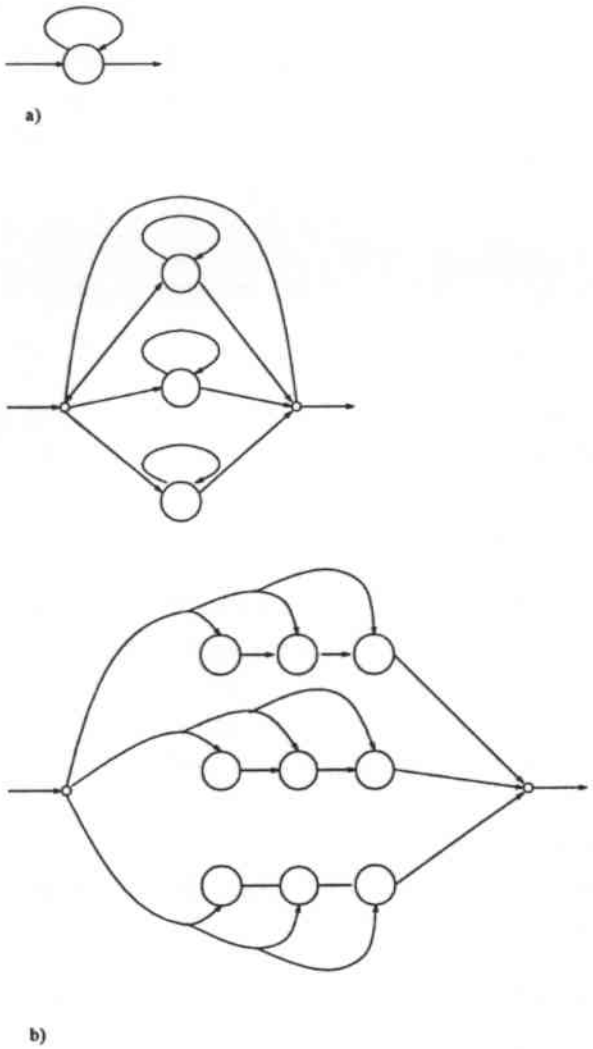

Abbildung 4.3. Grundlegende Strukturen des Garbage-Teils eines HMM-Netzwerkes.

4.3 Scoring-Methoden

4.3.1 Primäres Scoring

Das in 4.2 beschriebene Netzwerk kann nun verwendet werden, um einen Score für die darin enthaltenen Schlüsselworte und Füller-Modelle zu errechnen.

4.3.1.1 Zeitnormalisierte Wahrscheinlichkeitsscores

4.3.1.1.1 Die Methode von ROSE Die von ROSE [23] vorgeschlagene zeitnormalisierte Wahrscheinlichkeit[2] (*duration normalized word likelihood*) $W_{t_s}^{t_e}$ für ein Schlüsselwort w_n im Intervall t_s bis t_e mit Endzustand s_e ist wie folgt definiert:

$$W_{t_s}^{t_e}(w_n) = \frac{\log P(S_{t_e}, (\vec{x}_{t_s}, \ldots, \vec{x}_{t_e}))}{t_e - t_s}.$$

wobei $\mathbf{X} = (\vec{x}_{t_s}, \ldots, \vec{x}_{t_e})$ die gegebene Beobachtungssequenz aus Vektoren, die die Cepstral- und andere Koeffizienten enthalten, ist.
Da die Eingabe kontinuierlich erfolgt, und das Auftreten eines Schlüsselwortes so schnell wie möglich berichtet werden soll, verwendet ROSE zur Berechnung obiger Werte einen zeitsynchronen Viterbi-Decoder mit partiellem Backtrace[3]. Der Vorteil dieses Vorgehens liegt darin, daß mögliche Treffer durch den optimalen Pfad der maximalen Wahrscheinlichkeit bestimmt werden und nicht durch lokale Maxima des Wahrscheinlichkeitsscores (was im folgenden Paragraphen 4.3.1.2 beschrieben wird). Ein Nachteil ist darin zu sehen, daß das Auftreten eines Schlüsselwortes verspätet gemeldet werden kann, wenn nämlich der wahre Zeitpunkt und der durch den Viterbi-Algorithmus entdeckte weit auseinanderliegen.

4.3.1.1.2 Die Methode von ROHLICEK Eine andere zeitnormalisierte Wahrscheinlichkeit — ebenfalls auf den kontinuierlichen Fall angewandt — wird von ROHLICEK [22] verwendet, sie ist wie folgt definiert:

$$\hat{W}_{w_n}(t_e) = P(\vec{x}_{t_e-d+1}, \ldots, \vec{x}_{t_e} \mid w_n)^{\frac{1}{d}}$$

wobei d die geschätzte Dauer des Wortes w_n ist.

4.3.1.2 A posteriori Wahrscheinlichkeitsscores

4.3.1.2.1 Die Methode von ROHLICEK Die Idee hinter dem Verfahren von ROHLICEK[4] [22] ist es, die mit dem zeitsynchronen Viterbi-Algorithmus erhaltenen Scores

$$W^{t_e}(i) = P(S_e = i, \vec{x}_{t_s}, \ldots, \vec{x}_{t_e}) = \alpha_{t_e}(i)$$

[2] für kontinuierliche Sprache

[3] Der Viterbi-Algorithmus mit partiellem Traceback (Rückverfolgen) erlaubt es, einen initialen Teil des besten Pfades schon vor Ende der Eingabe zu errechnen, was bei der herkömmlichen Prozedur (siehe Abschnitt 3.5) nicht möglich war

[4] Er verwendet kontinuierliche Sprachdaten.

zum Zeitpunkt t_e und im Zustand i zu normalisieren. Dies hat folgenden Hintergrund: Eine niedrige Wahrscheinlichkeit für ein Schlüsselwort kann in einer niedrigen Wahrscheinlichkeit für das Sprachsignal überhaupt begründet sein. Um diesen Effekt auszugleichen, verwendet man die Füller-Modelle und erhält die *a posteriori* Wahrscheinlichkeit

$$\begin{aligned} PP(S_{t_e} = i \,|\, \vec{x}_{t_s}, \ldots, \vec{x}_{t_e}) &= \frac{P(S_{t_e} = i, \vec{x}_{t_s}, \ldots, \vec{x}_{t_e})}{P(\vec{x}_{t_s}, \ldots, \vec{x}_{t_e})} \\ &= \frac{W_{t_e}(i)}{\sum_j W_{t_e}(j)} \\ &= \frac{\alpha_{t_e}(i)}{\sum_j \alpha_{t_e}(j)}, \end{aligned}$$

einen ganz bestimmten Zustand zu besetzen. Dabei geht die Summe im Nenner über alle Zustände des Netzwerkes aus Schlüsselwort- und Garbage-Modellen. Die Alternativ-Modelle dominieren hierbei gewöhnlicherweise den Nenner und beeinflussen somit entscheidend das Ergebnis.
Schließlich kann man noch die Wahrscheinlichkeit für das Ende eines Schlüsselwortes definieren:

$$\hat{PP}_{w_n}(t_e) = PP(S_{t_e} = E_{w_n} \,|\, \vec{x}_{t_s} \ldots \vec{x}_{t_e}),$$

wobei E_{w_n} der Index des Endzustandes von Schlüsselwort w_n ist. Die möglichen Endpunkte für ein Schlüsselwort werden durch die *lokalen Maxima* in $\hat{PP}_{w_n}(t_e)$ bestimmt.
Der Nutzen dieses Scores wird jedoch dann gemindert, wenn das Netzwerk kein adäquates Modell für die ganze Sprache ist, d. h. wenn die Garbage-Modelle die Nicht-Schlüsselwort-Sprache nicht ausreichend beschreiben.

4.3.1.2.2 Die Methode von ROSE Eine ähnliche Motivation liegt dem *Likelihood-Ratio-Score* von ROSE [23] zugrunde. Um der Zeitveränderlichkeit der Schlüsselwortscores zu begegenen, führt er neben dem Schlüsselwort-Füller-Netzwerk, das im vorigen Abschnitt beschrieben wurde, noch ein sogenanntes *Background-Netzwerk* aus Garbage-Modellen ein (Bild 4.4). Dieses berechnet parallel den Score $W_{t_s}^{t_e}(b)$ für die überlappenden Alternativ-Modelle, welcher zusammen mit $W_{t_s}^{t_e}(w)$, dem Score für das Schlüsselwort, den Likelihood-Ratio-Score $\hat{W}$ für das Schlüsselwort w ergibt. Man erhählt:

$$\hat{W}_{t_s}^{t_e}(w_n) = W_{t_s}^{t_e}(w_n) - W_{t_s}^{t_e}(b).$$

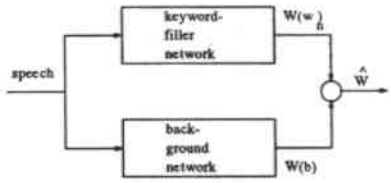

Abbildung 4.4. Schlüsselwort-Füller-Modell mit parallelem Background-Modell.

Wir erinnern uns, daß die zeitnormalisierten Wahrscheinlichkeiten $W_{t_s}^{t_e}(w_n)$ bereits logarithmisiert waren. Dies ist auch der Grund, weshalb $\hat{PP}$ (*ohne* Logarithmisierung) und $\hat{W}$ vergleichbar sind, denn es gilt:

$$\log \frac{a}{b} = \log a - \log b,$$

wobei die Logarithmusfunktion bekanntlich monoton ist.

4.3.1.3 "Kandidaten"-Score

Diese von WILPON [30] vorgestellte Methode geht wie folgt vor: Die Eingabesequenz wird mittels eines Viterbi-Algorithmus' Frame für Frame mit den vorhandenen Bezugsmodellen (in diesem Falle sind es HMM mit jeweils 10 Zuständen) verglichen, so daß man eine Folge von Modellen erhält, die am besten mit dem jeweiligen Frame übereinstimmt. (Dieses Verfahren ähnelt demjenigen von CHRISTIANSEN und RUSHFORTH [11] für den Fall der Musteranpassung.) Für die zeitliche Ausrichtung verwendet WILPON eine framesynchrone Viterbi-Prozedur (so beschrieben von RABINER und LEE in [13]), die für jeden möglichen Startframe i und Endframe j einen Schlüsselwort-Kandidaten $c(i,j)$ erzeugt, dessen durchschnittliche Modellwahrscheinlichkeit $p(i,j)$ und durchschnittliche Zustandswahrscheinlichkeit $s(i,j)$ wie folgt definiert sind:

$$p(i,j) = \frac{1}{j-i+1} \sum_{k=i}^{j} m_p(k),$$

wobei $m_p(k)$ der Wahrscheinlichkeitsscore des am besten passenden Modelles im Frame k für Kandidat $c(i,j)$ ist.

$$s(i,j) = \frac{1}{N} \sum_{n=1}^{N} s_p(n),$$

wobei $s_p(n)$ die durchschnittliche Zustandswahrscheinlichkeit im Zustand n für Kandidat $c(i,j)$ ist.
Die Berechnung von $s(i,j)$ läßt sich wie folgt erklären: Die Verweildauer in einem Zustand eines HMM ist von exponentieller Natur [20]. Während des Viterbi-Prozesses ist es möglich, daß das Spektrum der Eingabesequenz gut zu den spektralen Dichten einiger Zustände paßt, mit anderen hingegen nur dürftig übereinstimmt. Dies kann zur Folge haben, daß ein Pfad in einem oder mehreren Zuständen sehr lange verweilt (bei einem sehr guten Wahrscheinlichkeitsscore) und in anderen hingegen nur sehr kurz (mit einerm sehr schlechten Score). Das führt dazu, daß der Score für die gesamte Eingabe hoch sein wird, die durchschnittliche Zustandswahrscheinlichkeit jedoch niedrig.
Den Ausgabewerten $p(i,j)$ und $s(i,j)$ werden wir später noch einmal begegnen (Abschnitt 4.4).

4.3.2 Sekundäres Scoring

Die Ergebnisse der ersten Scoring-Prozedur werden nun in einem zweiten Schritt verfeinert. Das *Secondary Scoring* dient dazu, sowohl zwischen Schlüsselworten und anderer Sprache zu unterscheiden, als auch um die wahren Treffer besser von den falschen Alarmen trennen zu können, worauf wir in Abschnitt 4.4 über *Rejection* noch einmal zurückkommen werden.

4.3.2.1 Die Methode von ROSE

ROSE führt den unter Abschnitt 4.3.1.2.2 vorgestellten Gedanken in [24] fort, indem er eine zweite Scoring-Prozedur auf die Schlüsselwortkandidaten der ersten anwendet, deren Ergebnisse dann als "wahre" Risultate für die Keywords berichtet werden. Dieses sogenannte *Secondary Scoring* läßt sich wie folgt charakterisieren:

- Die vom zeitsynchronen Viterbi-Algorithmus berechnete logarithmisierte Wahrscheinlichkeit $\log P(\mathbf{X}_{t_s}^{t_e} \mid w_n)$ für den Endzustand S_{e_n} eines Schlüsselwortes w_n und gegebener Sequenz $\mathbf{X}_{t_s}^{t_e} = (\vec{x}_{t_s}, \ldots, \vec{x}_{t_e})$ erhalten wir als[5]:

$$\log P(\mathbf{X}_{t_s}^{t_e} \mid w_n) = \log P(S_e = e_n, \vec{x}_{t_s}, \ldots, \vec{x}_{t_e}).$$

  Für das gleiche Zeitintervall t_s bis t_e berechnen wir die größte logarithmisierte Wahrscheinlichkeit des Background-Netzwerkes (siehe 4.4) aus M Füller-Modellen:

[5] Aus Gründen der Übersichtlichkeit verwenden wir eine vereinfachte Notation.

$$\begin{aligned}\log P(\mathbf{X}_{t_s}^{t_e} \mid f) &= \max_{m=1,\ldots,M}(\log P(\mathbf{X}_{t_s}^{t_e} \mid f_m)) \\ &= \max_{m=1,\ldots,M}(\log P(S_e = d_m, \vec{x}_{t_s}, \ldots, \vec{x}_{t_e})),\end{aligned}$$

wobei S_{d_m} der Endzustand des Füller-Modells f_m ist.

- Diese beiden log-Wahrscheinlichkeiten werden nun benützt, um den endgültigen Score $W_{w_n}^{\star}(t_e)$ für ein Schlüsselwort w_n im Intervall zwischen t_s und t_e (dekodierte Grenzzeitpunkte der ersten Scoring-Prozedur) zu erhalten:

  $$W_{w_n}^{\star}(t_e) = \max_{t=t_s,\ldots,t_e}(\log P(\mathbf{X}_{t_s}^{t_e} \mid w_n) - \log P(\mathbf{X}_{t_s}^{t_e} \mid f)).$$

  Eine graphische Veranschaulichung findet sich in Bild 4.5 wieder. Die beiden oberen Kurven repräsentieren die Pfad-Wahrscheinlichkeiten, ausgehend vom dekodierten Startframe t_s des Schlüsselwortes. Man erhält sie aus den gesamten Pfad-Wahrscheinlichkeiten, dargestellt durch die unteren beiden Kurven, indem man den Abstand zwischen ihnen zum Anfangszeitpunkt entfernt.

Der so erhalten Score $W_{w_n}^{\star}$ stellt eine Approximation an die a posteriori log-Wahrscheinlichkeit des Schlüsselwortes dar. Das wollen wir uns klarmachen, indem wir die a posteriori Keyword-Wahrscheinlichkeit $\log P_{\lambda_w}(w \mid u)$ des Wortes w, modelliert durch λ_w, für die Ausgabesequenz $u = \mathbf{X}_{t_s}^{t_e}$ betrachten. Nach der Regel von BAYES (siehe auch Kapitel 3) gilt:

$$\begin{aligned}\log P_{\lambda_w}(w \mid u) &= \log \frac{P(u \mid w, \lambda_w)\, P(w)}{P(u)} \\ &= \log \frac{P(u \mid w, \lambda_w)\, P(w)}{\sum_{i=1}^{M} P(u \mid f_i, \lambda_{f_i})\, P(f_i)} \\ &\approx \log P(u \mid w, \lambda_w) - \log P(u \mid f_{max}, \lambda_{f_{max}})\end{aligned}$$

Um die Abschätzung im letzten Schritt durchzuführen, nimmt man an, daß alle vorhergehenden Schlüsselwortwahrscheinlichkeiten gleich sind und man die Wahrscheinlichkeit der Beobachtungsfolge als Ausdruck der größtwahrscheinlichen Füller-Sequenz darstellen kann.

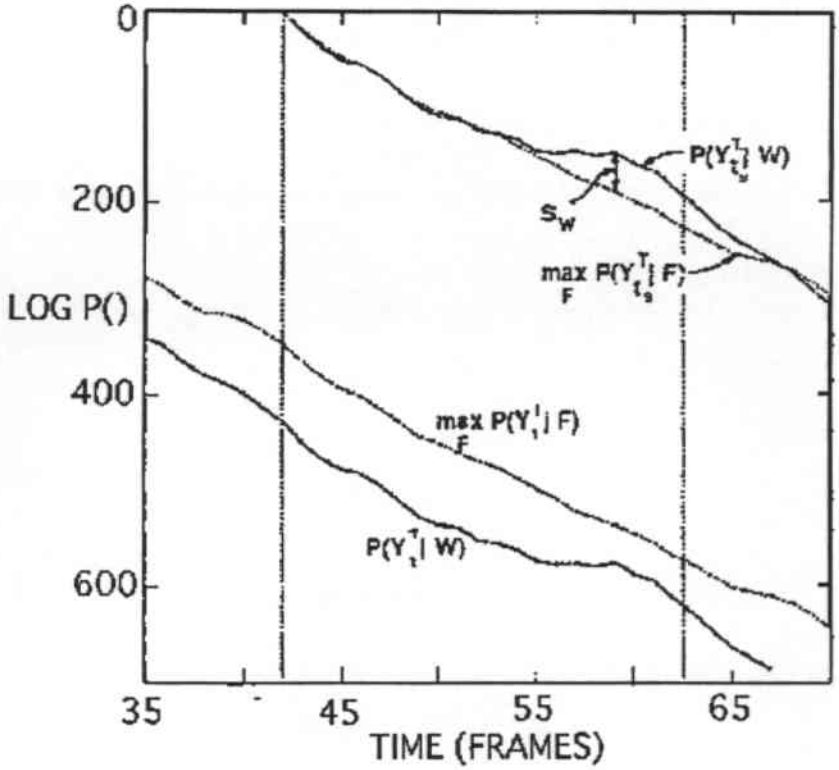

Abbildung 4.5. Sekundäres Scoring nach der Methode von ROSE.

Ähnlich geht man vor, um eine Beziehung zwischen dem sekundären Score $W^*_{w_n}(t_e)$ und der gemeinsamen (*mutual*) Information

$$I_{\lambda_w} = \log \frac{P(u \mid w, \lambda_w)}{P(u)},$$

die Schlüsselwort und Ausgabesequenz in Bezug setzt, herzustellen. Auf diese Beobachtung geht Abschnitt 4.6.1 noch genauer ein. Wir werden dort eine Technik zum Training eines HMM vorstellen, die mit der MMI-Methode eng verbunden und besonders dazu geeignet ist, Schlüsselworte von anderer Sprache zu trennen, wobei als Basis der sekundäre Score $W^*_{w_n}(t_e)$ zugrunde liegt.

4.3.2.2 Die Methode von WILCOX

Um mögliche Schlüsselwortendpunkte zu finden, verwendet WILCOX [29] zunächst den von ROHLICEK (Abschnitt 4.3.1.2.1) eingeführten a posteriori Wahrscheinlichkeitsscore. Ein sogenannter *Peak Detector* berichtet die lokalen Maxima für ein Schlüsselwort w_n und liefert somit die möglichen Endframes zum Zeitpunkt t_e. In einem zweiten Durchgang werden dann die Backward-Variablen $\beta_t(j)$ ausgehend von t_e sowohl für das Schlüsselwort-HMM w_n als auch für den Garbage f berechnet.
Um einen zeitnormalisierten Wahrscheinlichkeitsscore $\overline{W}_t^{t_e}(w_n)$ für das Schlüsselwort w_n, das in t beginnt und in t_e endet, berechnen zu können, benötigt man zunächst

$$\begin{aligned} L_t^{t_e}(w_n) &= P(\mathbf{X} = \vec{x}_t, \ldots, \vec{x}_{t_e})^{\frac{1}{t-t_e}} \\ &= \beta_t(S_1)^{\frac{1}{t-t_e}}. \end{aligned}$$

für das Schlüsselwort und $L_t^{t_e}(f)$ für den Garbage, was sich auf ähnliche Weise berechnet.
Wir erhalten damit dann

$$\overline{W}_t^{t_e}(w_n) = \frac{L_t^{t_e}(w_n)}{L_t^{t_e}(w_n) + L_t^{t_e}(f)}.$$

Diese Formel ähnelt der in Abschnitt 4.3.1.1.1 von ROSE vorgeschlagenen.
Den Startframe t_s wählt man dabei so, daß er $\overline{W}_t^{t_e}(w_n)$ maximiert. Nimmt man an, daß die Dauer des Schlüsselwortes im Bereich zwischen der Hälfte und des Eineinhalbfachen der Trainingssequenz liegt, so erhält man den Anfangszeitpunkt als

$$t_s = \underset{t=t_e-1.5\,T,\ldots,t_e-0.5\,T}{\operatorname{argmax}} \overline{W}_t^{t_e}(w_n),$$

wenn T die Dauer der im HMM-Training verwendeten Sequenz bezeichnet.

4.4 Rejection

Schon in Abschnitt 4.3.2 über sekundäres Scoring haben wir Verfahren diskutiert, die dazu dienen, unwahrscheinliche Schlüsselwort-Kandidaten zurückzuweisen. Neben der Berechnung eines neuen Scores bieten sich noch weitere Methoden an, die wir in diesem Paragraphen vorstellen wollen.
WILPON unterwirft die in 4.3.1.3 erhaltenen Schlüsselwort-Kandidaten folgenden 4 Kriterien, denen sie genügen müssen.

1. **Verhältnis-Test:** Ein Kandidat wird als wahrscheinlich angesehen, wenn

$$0{,}65 \leq \frac{p(i,j)}{s(i,j)} \leq 1{,}35$$

   für ihn erfüllt ist. Dies ist dann nicht der Fall, wenn das Modell für wenige Zustände sehr gut paßt, in den restlichen jedoch sehr schlecht.

2. **Mindestdauer-Test** Eine gewisse Mindestdauer muß überschritten werden.

3. **Energie-Test** Die Energie innerhalb des Schlüsselwortes muß für jeden Frame oberhalb einer gewissen Grenze liegen.

4. **Wahrscheinlichkeits-Test** Der Wahrscheinlichkeitsscore muß eine vordefinierte Schranke überschreiten.

Die Grenzwerte für die einzelnen Schlüsselworte erhält man in der Regel dadurch, daß man den Durchschnitt der Wahrscheinlichkeiten über die Vorkommen im Trainingsset berechnet. Diese so erhaltenen Schranken lassen sich dann noch mit einem Koeffizienten ρ gewichten, der die Empfindlichkeit des Wordspotters steuert, so daß je nach gewähltem Wert mehr oder weniger Kandidaten diese Grenze passieren.
Dieses letzte Kriterium wird von praktisch allen Autoren auf den primären Score angewandt, ROSE definiert zusätzlich noch Schranken für seinen sekundären Score (siehe 4.3.1.2.2), um unwahrscheinliche Kandidaten zu verwerfen.

4.5 Garbage-Modelle und ihre Erzeugung

In Abschnitt 4.2.3 haben wir die Grundstruktur eines Garbages-Modelles und dessen Generierung schon einmal kurz betrachtet. Im nun folgenden wollen wir uns ausführlicher mit der Konstruktion des Alternativ-Teiles eines HMM-Netzwerkes beschäftigen, besonders unter dem Aspekt, daß man die Nicht-Schlüsselwort-Sprache explizit modelliert, d. h. zum Trainieren der Modelle verwendet man Sprache, die nicht zum Vokabular des Wordspotters gehört. Um es noch einmal zu betonen: Eine gute Darstellung der Außer-Schlüsselwort-Sprache hat den Erfolg, daß sich die Rate der falsch entdeckten Keywords senkt, und somit der ROC insgesamt ansteigt.
Zunächst stellen wir die von ROSE [23] beschriebenen Garbage-Modelle vor.

4.5.1 Ganzwortmodelle

Als der beste Ansatz, einen Wordspotter zu bauen, wird vorgeschlagen, einen Spracherkenner für ein umfangreiches Vokabular zu konstruieren, wobei der Großteil des Wortschatzes zur Modellierung der Nicht-Schlüsselwort-Sprache verwendet wird [3]. Ein derartiges System würde viel Arbeit mit den Trainingsdaten verlangen, und auch zu einem äußerst großen Erkennungsnetzwerk führen. (ROSE konstruiert $M = 80$ Filler-Modelle (vgl. Bild 4.1) aus derselben Anzahl von Worten, was ein Netzwerk von insgesamt 804 Zuständen ergibt. Dieses liefert bemerkenswert gute Ergebnisse, wenn man bedenkt, daß die Anzahl von 80 Worten im Vergleich zur unbeschränkten Nicht-Keyword-Sprache sehr klein ist.)

4.5.2 Teilwortmodelle

4.5.2.1 Triphoneme

Verwendet man ein umfassendes Vokabular an Nicht-Schlüsselworten und zum Training steht nur begrenzt Material zur Verfügung, wird der größte Teil der Eingabeäußerung nicht in den Trainingsdaten vorkommen. Hieraus entspringt die Idee, sogenannte *Triphoneme* zu verwenden, das sind Modelle, die im Zusammenhang mit den links uns rechts angrenzenden Phonem definiert sind. Sie ermöglichen es, Füller-Modelle über verschiedene Kontexte hinweg zu verwenden, wobei die Komplexität des Spotters relativ unverändert bleibt. (ROSE modelliert für die 80 Wörter 264 Triphone aus je einem linearen HMM mit drei Zuständen, insgesamt also 804 Zustände. Die Leistung verbesserte sich im Vergleich zur vorher beschriebenen Methode, logischerweise verschlechtert sie sich wieder, wenn man im Garbage-Netzwerk Triphoneme aus dem aktuellen Schlüsselwortsatz hinzunimmt. Dies führt nämlich oft dazu, daß anstatt des Schlüsselwortes das Alternativ-Modell dekodiert wird und somit zu einer erhöhten Anzahl von nicht entdeckten Keywords.)

4.5.2.2 Monophoneme

Hier verwendet man im Filler-Netzwerk eine kleine Anzahl von Monophonemen über einen allgemeinen Kontext, was zu einer Verringerung der Modellgröße führt. (ROSE erhält insgesamt 135 Netzwerkzustände, bei einer nur gering gesunkenen Leistung gegenüber des Triphonem-Ansatzes.)

4.5.3 Unbeaufsichtigtes Clustering

Der Unterschied zu den eben beschrieben Verfahren besteht bei diesem Ansatz darin, daß man ungelabelte Sprache clustert, um daraus die Alternativ-Modelle zu erhalten. Die Mittelwerte der GAUSS-Verteilungen der Ein-Zustand-Filler-Modelle würde man als Cluster-Zentroide des k-means-Algorithmus erhalten. (ROSE erzeugt insgesamt 128 solcher HMM, wobei die Zustandsübergangswahrscheinlichkeit so gewählt ist, daß jedes Garbage-Modell eine erwartete Dauer von etwa 200 ms hat. Die Erkennungsleistung in Verbindung mit dieser Methode ist dürftig, weil die Rate falsch erkannter Schlüsselworte sehr hoch ist. Dies läßt vermuten, däs die Cluster-Modelle, die Nicht-Keyword-Sprache nur sehr schlecht beschreiben.)
Die nun folgenden Filler-Modellierungen stammen von BOITE [5] [6] und stellen im wesentlichen eine Verfeinerung der eben beschriebenen Methoden dar. Im letzten Punkt werden wir jedoch noch einen völlig neuen Ansatz kennenlernen.

4.5.4 1-Garbage-Modell

BOITE repräsentiert die Schlüsselworte mit kontextabhängigen (CD) und kontextunabhängigen (CI) Phonemmodellen. Für den Garbage kreiert er zunächst auch ein einziges Alternativ-Modell, das durch die gesamten Trainingsdaten (ausgenommen den Keywords) gewonnen wird. Aufgrund dieser Definition des Garbages fallen die lokalen Wahrscheinlichkeiten dafür sehr niedrig aus. Deshalb wird eine sogenannte Worteintrittsstrafe (*word entrance penalty*) eingeführt, die den Score für ein Schlüsselwort herabsetzt. Die Erkennungsrate wird dadurch nicht beeinflußt, jedoch erhöht sich die Rate der Zurückweisungen. Wählt man die Worteingangsstrafe zu hoch, sinkt allerdings die Erkennungsquote stark ab.

4.5.5 CI-Phonem-Modelle

In diesem Falle werden die CI-Phonem-Modelle zur Beschreibung des Garbages und die CD-Phonem-Modelle zur Darstellung der Schlüsselworte benützt (siehe auch Abschnitt 4.5.2). Da jede Folge von Phonemen (auch Schlüsselworten) für die Garbage-Modellierung zulässig ist, erhält man eine zu hohe Rejection-Rate. Man begegnet dem dadurch, daß man eine Garbage-Übergangs-Strafe einführt, die nun den Score für das Alternativ-Modell senkt.

4.5.6 N-Garbage-Cluster

Die CI-Phoneme werden in eine Menge von N Garbage-Clustern eingeteilt; man geht dabei wie folgt vor: Für je zwei Klassen wird der Informationsverlust (Enthropie) berechnet; dasjenige Paar, das diesen Wert minimiert, kommt gemeinsam in den neuen Cluster. Man terminiert den Algorithmus, sobald die gewünschte Anzahl von N Klassen erreicht ist. Bei diesem Verfahren müssen dann sowohl die Worteingangs- als auch die Garbageübergangsstrafe eingestellt werden. (Für die Anwendung wurde $N = 7$ gewählt.)

4.5.7 On-line Garbage

Dieses Verfahren stellt einen neuen Ansatz dar, denn es wird nicht versucht, Alternativ-Modelle explizit zu modellieren, vielmehr werden die lokalen Scores für den Garbage on-line, Frame für Frame berechnet. Dieser Score ergibt sich als der Durchschnitt der N besten lokalen Wahrscheinlichkeiten für die CI und CD. Auf diese Art und Weise ist der Filler-Score nie der beste, aber immer unter den aussichtsreichsten Kandidaten. Das hat zur Folge, daß Garbage nur dann dekodiert wird, wenn das Schlüsselwort global schlecht getroffen wird. Während beim 1-Garbage- und beim N-Cluster-Verfahren global über die Nicht-Schlüsselwortsprache geglättet wird, geschieht es hier lokal, on-line. Außerdem ist der Score resistenter gegen Noise, denn bei den Standardverfahren, wo das Garbage-Modell der einfache Durchschnitt der anderen Phonem-Modelle ist, kann die Überlagerung mit Störgeräuschen zur Erhöhung des Scores führen. Schließlich wird auch hier noch eine Worteingangsstrafe hinzugenommen, um die Rejecton-Rate zu steigern. (In der Anwendung erwies sich $N = 20$ bei 42 CI und 35 CD als gut geeignet.)

4.6 Fortgeschrittene HMM-Modellierung

Wir sind bereits mehreren Verfahren begegnet, die Parameter eines HMM zu trainieren. Wir haben die Maximum-Likelihood-Methode betrachtet, die die Wahrscheinlichkeit der Beobachtungssequenz über alle Modellparameter maximiert. Die K-means-Prozedur optimiert die Wahrscheinlichkeit der Beobachtungssequenz und der Zustandsfolge über alle Modellparameter. Das Maximum-Mutual-Information-Kriterium versucht, die verschiedenen Modelle möglichst scharf von einander abzugrenzen. Im folgenden Abschnitt stellen wir ein ähnliches Verfahren vor, das speziell auf die Anforderungen beim Wordspotting abgestimmt ist. Es ermöglicht, Schlüsselwortmodelle von einer breiten Klasse akustischer Ereignisse (andere Sprache, Geräusche) abzutrennen.

4.6.1 Discriminant-Training

Bei dieser Methode führt ROSE die Überlegungen zum MMI-Verfahren hinsichtlich seines sekundären Scores (Abschnitt 4.3.2.1) fort.

4.6.1.1 Gebundene Gauß-Mischungen

ROSE verwendet Mischungen aus sogenannten gebundenen Gauß-Verteilungen [4]. Die Dichte $B = (b_l(\vec{x})$ für alle Zustände $i = 1, \ldots, N$ des HMM-Systems ist eine Mischung über Gauß-Dichten $f_m(\vec{x}), m = 1, \ldots, M$. Alle Zustände teilen sich dieselben Dichten, so daß sowohl die Schlüsselwort-Modelle als auch die Füller-Modelle dieselben Gauß-Parameter haben. Zustandsabhängige Mischungsgewichte $b_{l,m}$ binden die Gauß-Dichten $f_m(\vec{x})$ an den Zustand i, so daß sich die Emissionswahrscheinlichkeit wie folgt darstellt:

$$b_l(\vec{x}) = \sum_{m=1}^{M} b_{l,m}\, f_m(\vec{x}).$$

Die Maximum-Likelihood-Schätzung aller HMM-Parameter (Mittelwert und Varianz der f_m, Mischungsgewichte $b_{i,m}$, Transitionswahrscheinlichkeiten $a_{i,j}$) sind simultan mittels des Forward-Backward-Algorithmus' berechnet.

4.6.1.2 Unterscheidungs-Kriterium

Der sekundäre Score

$$W^{\star}_{w_n}(t_e) = \max_{t=t_s,\ldots,t_e} (\log P(\mathbf{X}_{t_s}^{t_e} \,|\, w_n) - \log P(\mathbf{X}_{t_s}^{t_e} \,|\, f))$$

aus Abschnitt 4.3.2.1 stellte ein gutes Maß dar, Schlüsselworte von anderer Sprache abzugrenzen. Bisher ging er jedoch nicht in die Maximum-Likelihood-Schätzung der HMM-Parameter für Keywords und Garbage ein; man versuchte nämlich nur die Wahrscheinlichkeit eines Modelles für ein Schlüsselwort zu maximieren, jedoch nicht die Wahrscheinlichkeit der anderen Modelle für dieses Schlüsselwort zu verringern. Für einen Wordspotter, der ein Schlüsselwort w über eine Beobachtungsfolge $u = \mathbf{X}_{t_s}^{t_e} = (\vec{x}_{t_s}, \ldots, \vec{x}_{t_e})$ verläßlich dekodiert, wünschen wir uns daß

$$P(w \,|\, u, \lambda_w) > P(f \,|\, u, \lambda_f)$$

für alle Füller-Modelle f.
Wir stellen nun eine Abgrenzungs-Prozedur vor, die ein Kriterium maximiert, das direkt mit dem Score $W^{\star}$ verbunden ist.

In unserem Falle werden nur die Parameter für die Schlüsselwortmodelle neu eingestellt; die Garbage-Modelle bleiben unverändert. Da die Füller-Modelle eine breite Klasse von Sprache darstellen, liegt die Vermutung nahe, daß sie sich mit einer vernünftigen Menge an Trainingsdaten verfeinern lassen. Da die Gauß-Dichten f_m über alle Zustände gleich sind, müssen wir nur noch die Mischungsgewichte $b_{l,m}$ aktualisieren, denn auch die Transitionswahrscheinlichkeiten bleiben unverändert.
Es gibt zwei Alternativen, zum gewünschten Ziel zu gelangen.

- Wir könnten die in Paragraph 4.3.2.1 erhaltene Beziehung $\log P(u \mid w, \lambda_w) - \log P(u \mid f_{max}, \lambda_{f_{max}})$ maximieren, indem wir die Gradienten-Abstiegs-Methode verwenden und danach eine Geraden-Suche entlang desselben durchführen, um den Randbedingungen der Parameter zu genügen.

- Die zweite Möglichkeit folgt dem Ansatz von BAHL [1]. Dieser *perceptron*-ähnliche Algorithmus erlaubt es, die Parameter der Modelle iterativ zu verfeinern, um so die Wahrscheinlichkeit des richtigen Modelles gegenüber der des falschen bei einer bestimmten Beobachtungssequenz zu vergrößern.

Wir werden die zweite Methode bevorzugen, indem wir zunächst — dem ersten Verfahren folgend — den Gradienten der letzen Gleichung mit Bezug auf die Mischungsgewichte $b_{i,m}$ ableiten, und dann seine Ähnlichkeit mit demjenigen in der Corrective-Training-Prozedur verwendeten herausarbeiten. Der Gradient der a posteriori Schlüsselwortwahrscheinlichkeit nach dem Mischungsgewicht $b_{l,m}$ errechnet sich nun wie folgt:

$$\frac{\partial \log P(w \mid u, \lambda_w)}{\partial b_{l,m}} = \frac{\frac{\partial P(u \mid w, \lambda_w)}{\partial b_{l,m}}}{P(u \mid w, \lambda_w)} - \frac{\frac{\partial P(u \mid f_{max})}{\partial b_{l,m}}}{P(u \mid f_{max}, \lambda_f)}.$$

Wir erhalten diesen Gradienten, wenn wir die Wahrscheinlichkeit für eine Beobachtungsfolge mittels der HMM-Parameter berechnen

$$P(u \mid w, \lambda_w) = \sum_{s \in \{S_1, \ldots, S_n\}} \sum_{o \in \{1, \ldots, M\}} \Pi_{t=t_s}^{t_e} a_{s(t-1),s(t)}\, b_{s(t),o(t)}\, f_{o(t)}(\vec{x}_t),$$

wobei $s(t)$ den Zustand zum Zeitpunkt t bezeichnet und $o(t)$ den Index der Mischung ebenfalls zur Zeit t. Daraus ergibt sich:

$$\frac{\partial P(u \mid w, \lambda_w)}{\partial b_{l,m}} = \frac{1}{b_{l,m}} \gamma^{u,w}(l,m)\, P(u \mid w, \lambda_w),$$

wobei

$$\gamma^{u,w}(l,m) = \sum_{t=t_s}^{t_e} P(s(t) = l, o(t) = m \,|\, u, w, \lambda_w)$$

die Wahrscheinlichkeit ist, den Zustand l und die Mischung m über die Sequenz u zu besetzen. Den gesuchten Gradienten erhält man nun als:

$$\frac{\partial \log P(w \,|\, u, \lambda_w)}{\partial b_{l,m}} = \frac{1}{b_{l,m}} \left(\gamma^{u,w}(l,m) - \gamma^{u,f}(l,m)\right),$$

mit der Randbedingung $\sum_{m=1}^{M} b_{l,m} = 1$ für die Mischungsgewichte.
Dieser Ausdruck für den Gradienten ist dem für den MMI-Fall in [1] ähnlich, stellt jedoch nur eine Annäherung an denselben dar, denn es wird nur eine falsche Dekodierung über der Beobachtungsfolge betrachtet und nicht alle. Der soeben berechnete Gradient kann für kleine Werte der Mischungsgewichte $b_{l,m}$ sehr groß werden; dies kann dazu führen, daß die Bestimmung des Gradienten von diesen sehr kleinen, nur sehr schlecht geschätzten Werten beherrscht wird.
Anstatt nun eine Gradienten-Abstiegs-Prozedur mit Berechnung dieses Gradienten und einer Geraden-Suche für die Einhaltung der Randbedingungen durchzuführen, benützen wir den Corrective-Training-Algorithmus von BAHL [2].

4.6.1.3 Das Corrective-Training-Verfahren

Den Gradienten, den wir für diesen Algorithmus verwenden, ähnelt demjenigen des vorigen Abschnittes, außer daß er proportional zur Differenz zwischen den logarithmisierten Wahrscheinlichkeiten der "richtigen" und "falschen" Modelle ist. Für jede Schlüsselwort-Äußerung im Corrective-Training werden die Werte gemäß des folgenden Ausdruckes aktualisiert:

$$\hat{\gamma}^{w}(l,m) = \hat{\gamma}^{w}(l,m) + \nu \left(\gamma^{u,w}(l,m) - \gamma^{u,f}(l,m)\right),$$

wobei ν eine Funktion der Differenz $W_w^u = \log P(u \,|\, w) - \log P(u \,|\, f)$ zwischen den logarithmisierten Schlüsselwort- und Füller-Wahrscheinlichkeiten ist, mit:

$$\nu = \begin{cases} \beta, & W_w^u < 0 \\ -\frac{\beta}{\delta} W_w^u + \beta, & 0 \le W_w^u \le \delta \end{cases}$$

wobei β und δ empirisch gewählte Konstanten sind, die bestimmen wie der Gradient von W_w^u beeeinflußt wird. Die neuen Mischungsgewichte erhält man mittels der $\hat{\gamma}^w$ als:

$$\hat{b}_{l,m} = \frac{\hat{\gamma}^w(l,m)}{\sum_{m=1}^{M} \hat{\gamma}^w(l,m)}.$$

Als Ausgangsbasis für diesen Algorithmus kann man die, mittels der Maximum-Likelihood-Methode trainierten Schlüsselwortmodelle λ_w und die Garbage-HMM λ_f verwenden. Auch die Werte für $\hat{\gamma}^w(l,m)$, einen bestimmten Markov-Zustand l und eine Mischung m zu besetzen, berechnet man zunächst mit dem Standard-Forward-Backward-Verfahren. Beim eigentlichen Corrective-Training führt man dann den Aktualisierungsprozeß über die $\hat{\gamma}^w(l,m)$ für eine Schlüsselwortäußerung und die Neuschätzung der Mischungsgewichte $b_{l,m}$ solange durch, bis ein Konvergenzkriterium bezüglich des durchschnittlichen W_w^u erfüllt ist.

4.6.1.4 Nahe "Fehlschüsse"

Bei der Erkennung isolierter Worte wurden für das Corrective-Training sogenannte nahe Fehlschüsse (*near misses*) definiert [2]. Das sind diejenigen Worte aus dem Vokabular des Erkenners, die für eine bestimmte Äußerung mit dem richtigen Wort akustisch verwechselbar sind. Dasselbe wird man beim Word Spotting tun, außerdem wird die Garbage-Sequenz als Near Miss bezeichnet, die über ein Teil der kontinuierlichen Eingabe hinweg mit einem Schlüsselwort verwechselt werden kann. Diese Folge von Füller-Modellen wird mittels eines Null-Grammatik-Garbage-Netzwerkes erzeugt, wobei jedes Alternativ-Modell ein Phonem repräsentiert. Die Randbedingung dabei ist, daß die Filler-Sequenz und das Schlüsselwortmodell die gleiche Anzahl von Zuständen haben. Dies wird noch deutlicher, wenn man darauf hinweist, daß bei der Gradientenberechnung sowohl für das Schlüsselwortmodell als auch für das Garbage-Modell derselbe Index l verwendet wird, was eine ähnliche Struktur beider HMM voraussetzt.

4.6.1.5 Zusätzliche Maßnahmen

- Nach dem Corrective-Training ist der Vektor der Mischungsgewichte für einen HMM-Zustand sehr spärlich besetzt. Um dies auszugleichen, werden aus den neuen $\hat{b}_{l,m}$ und aus den ursprünglichen $b_{l,m}$ die endgültigen $b^{\star}_{l,m}$ gewonnen:

  $$b^{\star}_{l,m} = \rho\, \hat{b}_{l,m} + (1-\rho)\, b_{l,m},$$

  wobei hier $\rho = 0{,}8$ gewählt wurde.

- Erhält man bei der Iteration für $\hat{\gamma}^w(l,m)$ negative Werte, so setzt man dafür einen kleinen, positiven, konstanten Wert [2].

4.6.1.6 Schlußbemerkung

Das beschriebene Verfahren erhöht die a posteriori Schlüsselwortwahrscheinlichkeit, die Leistung des framesynchronen Viterbi-Dekoders bleibt jedoch unverändert. Insgesamt führt die Methode zu einer Verbesserung des ROC.

4.6.2 Error-Correcting-Training

Eine ähnliche Methode wie die im vorangehenden Abschnitt wird von NILES vorgestellt. Dieser fehlerkorrigiernde Trainingsalgorithmus für HMM entspricht grundlegend dem Backpropagation-Algorithmus für ein rekursives Neuronales Netz. Die notwendigen Ableitungen werden von einer Backward-Rekursion berechnet, die dem herkömmlichen Forward-Backward-Verfahren entspricht. Der verwendete Fehler kann relativ frei gewählt werden, er muß nur lokal in der Zeit (Frame-für-Frame-Basis) und differenzierbar sein.

4.6.3 Cluster-Training

Ein ähnliches Verfahren wie ROSE in [23] (Abschnitt 4.5.3) zur Bestimmung der Füller-Modelle benützt, wird von WILCOX in [29] verwendet, um die Parameter der Schlüsselwortmodelle festzulegen. Mittels eines Cluster-Algorithmus' (vergleichbar mit der Standardmethode in 2.8) werden der Mittelwert (Zentroid des Clusters) und die Kovarianz-Matrix für die Gauß-Verteilung der Keyword-HMM bestimmt.

4.7 Grammatiken

Wie auch bei anderen Anwendungen im Bereich der Spracherkennung läßt sich durch die Verwendung von Grammatiken, die auf der Beschaffenheit der zugrundeliegenden Testsequenzen basieren, die Leistung eines Wordspotters erhöhen. Bei der Schlüsselworterkennung spielt hierbei wiederum die Integration der Garbage-Modelle ein bedeutende Rolle.
Weiß man beispielsweise, daß in einer Sequenz genau ein Keyword vorkommt, kann man eine denkbare, resultierende Grammatik recht einfach anhand eines Zustandsübergangsgraphen (Bild 4.6) veranschaulichen.
Der Knoten 0 ist dabei der Anfangszustand, in dem sowohl Stille (*silence*) wie auch Nicht-Schlüsselwort-Sprache (*garbage*) von unbegrenzter Dauer dekodiert werden können. Ein Übergang zum Endzustand (Knoten 1) findet

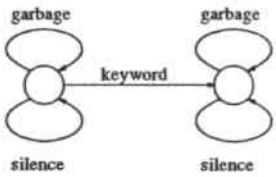

Abbildung 4.6. Zustandsübergangsdiagramm einer Ein-Schlüsselwort-Grammatik

statt, sobald ein Schlüsselwort entdeckt wird. Im Endzustand ist dann wieder die zeitlich nicht beschränkte Erkennung von Silence und Garbage möglich.

Kapitel 5

Der Turiner Wordspotter

5.1 Einführung

In diesem Kapitel wollen wir die Implementation eines Wordspotters vorstellen. Zunächst beschreiben wir die Datenbasis, im Abschnitt über Implementierungstechniken erörtern wir einige prinzipielle Methoden und präsentieren dann schließlich den eigentlichen Wordspotter. Von den Eigenschaften der verwendeten Datenbasis ausgehend, bietet sich dafür ein zweiteiliger Aufbau an, der sich an der von WILPON in [30] vorgestellten Struktur orientiert[1] und sich wie folgt charakterisieren läßt:

- Der **Wordspotting-Viterbi-Scorer** für verbundene Sprache, der auf einem Viterbi-Forward-Algorithmus für isolierte Worte beruht.
- Der **Score-Analysierer**, der die von der Viterbi-Prozedur zur Verfügung gestellten Werte weiterverarbeitet, um mögliche Schlüsselworttreffer aufzufinden.

Die grobe Strukturierung des Wordspotters ist noch einmal in Bild 5.1 dargestellt.

5.2 Die Datenbasis

5.2.1 Die Struktur der Datenbasis

Die in unserem Fall verwendete Datenbasis aus zusammenhängender (*connected*) Sprache umfaßt ein Vokabular von 10 Worten, namentlich die

[1] Dort wird hinter den Viterbi-Dekodierer ein sogenannter Postprozessor geschaltet, der zur weiteren Unterscheidung zwischen echten und falschen Treffern dient.

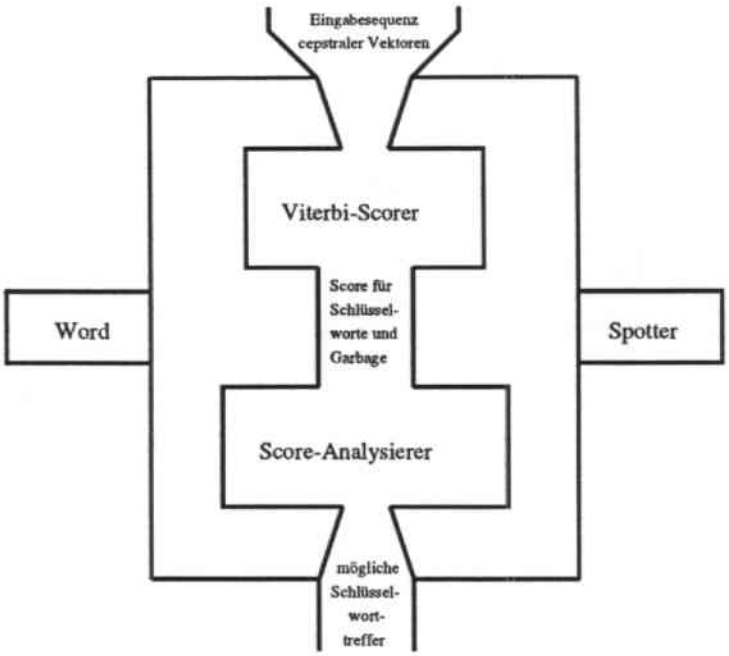

Abbildung 5.1. Grobstruktur des Wordspotters.

italienischen Ziffern: uno, due, tre, quattro, cinque, sei, sette, otto, nove, zero. In der Datenbasis befinden sich Sprechproben von 1000 Personen[2], mit gleichvielen weiblichen und männlichen Sprechern aus verschiedenen Altersgruppen und mit den unterschiedlichsten dialektischen Ausprägungen. Diese Datenbasis besteht zu gleichen Teilen aus Orts- und Ferngesprächen. Die Teilnehmer wurden zuhause angerufen und benutzten ihr eigenes Telefongerät, um die verlangten Sequenzen aus Telefon- und Vorwahlnummern nach einem Piepston aufzusprechen.
Das Sprachsignal wurde nach einer Tiefpaßfilterung (300 – 3400 Hz) mittels eines 16-Bit-A/D-Wandlers bei einer Abtastrate von 12800 Hz digitalisiert. Die Präemphase auf dem digitalen Signal wurde durch ein Netzwerk erster Ordnung und der Übertragungsfunktion $H(z) = 1 - 0{,}95z^{-1}$ vorgenommen.

[2]Vertreten waren Sprecher sowohl aus ländlichen als auch aus städtischen Gebieten.

Danach wurden die Anfangs- und Endpunkte der Proben bestimmt[3]. Verwendet wurde dabei ein energiebasierter Endpunkt-Auffindungs-Algorithmus (**E***nd*-**P***oint*-**D***etection*) in Abhängigkeit der Intensität des Signals und der vorhandenen Nebengeräusche. Die gefunden Begrenzungen wurden dann von Hand[4] auf ihre Genauigkeit hin überprüft. Nicht-sprachliche Elemente (Klicks, Atmen, Wähl- und Umgebungsgeräusche etc.) wurden ebenfalls gelabelt, um so explizite Modelle dafür erzeugen zu können.
Die parametrische Darstellung eines jeden Frames der Dauer 10 ms wurde mittels einer FFT, gewichtet durch ein Hamming-Fenster der Breite von 256 Frames erhalten. Die einzelnen Frames überlappen sich dabei zu etwa 50%. Die digitale Filterbank besteht aus 13 Bändern, ausgerichtet nach der Mel-Skala und zentriert um die kritischen Frequenzen des menschlichen Hörsystems. Auf jeden Frame wird schließlich eine diskrete Cosinus-Transformation (DCT) angewandt, um den Vektor mit den 12 Cepstralkoeffizienten zu erhalten. Im nächsten Schritt werden die zeitlichen Ableitungen der Cepstralkoeffizienten zu dem Vektor hinzugefügt, berechnet durch ein orthogonales Polynom erster Ordnung wie wir es aus Abschnitt 2.7 kennen. Also:

$$\Delta\hat{C}_i(t) = G \sum_{k=-K}^{K} k\,\hat{C}_i(t-k) \qquad i = 1, \ldots, p,$$

wobei jetzt $p = 12$ (Anzahl der Cepstralkoeffizienten) und $K = 2$ gelten. Um Lautstärkeschwankungen erfassen zu können, wird der abgeleitete Energiekoeffizient (siehe auch Abschnitte 2.6 und 2.7) zum Parametervektor hinzugenommen. Zuletzt wird ein, in geräuschvoller Umgebung vorteilhaftes, statistisch gewichtetes Abstandsmaß für die Koeffizienten eingeführt [25].
Die Beschaffenheit der Datenbasis war der Aufgabenstellung Wordspotting leider nicht sehr zuträglich. Die geringe Anzahl von Worten schuf ein ungünstiges Verhältnis zwischen Wordspotter-Vokabular und sonstiger Sprache, was das Wordspotting noch schwierger werden ließ als es schon ist; oft wurden Schlüsselworte anstatt Garbage dekodiert.

5.2.2 Die resultierenden Modelle

Die akustischen Eigenschaften des Phänomens, das man beschreiben will, spiegeln sich in den HMM (siehe Kapitel 3) wider. Jedes Modell repräsentiert dabei eine phonetische Einheit, die den kleinsten unterscheidbaren Wert festlegt. In unserem Falle (isoliert wie verbunden) ist das Vokabular mit

[3]Die Kenntnis der Grenzpunkte ist für die Trainingsphase der HMM wichtig.
[4]computergestützt durch Anhören und Betrachtung der Spektren

10 unterschiedlichen Worten sehr begrenzt, deshalb ist es ausreichend sogenannte Ganzwortmodelle zu verwenden; d. h. jedes HMM modelliert ein einzelnes Wort (die Modelle umfassen dabei zwischen 12 und 16 Zuständen). Für umfangreichere Aufgaben ist es notwendig, ein Alphabet auf der Basis kleinerer phonetischer Einheiten zu verwenden; diese bezeichnet man als *subword units*. Es gibt nun verschieden Möglichkeiten, diese Untereinheiten zu modellieren. Zuerst bieten sich die *Phoneme*, die kleinsten Elemente einer Sprache an (im Italienischen gibt es etwa 30 davon), die unabhängig vom Kontext definiert sind. Jedoch ist die Entwicklung des Sprachsignals kontinuierlich im Klang, was eine Abtrennung eines Phonems von seiner Umgebung schwierig macht. Deshalb neigt man dazu, ein Phonem durch mehrere *Allophone* zu repräsentieren; das /m/ in *amo* und in *uomini* stellen beispielsweise zwei verschiedene Allophone desselben Phonems /m/ dar. Hierfür verwendet man häufig *Diphone* oder auch *Triphone*, seltener *Polyphone* bzw. *Silben*.

Diphone umfassen das akustische Signal von der Hälfte eines Phonems bis zur Hälfte des folgenden; d. h. die Diphone hängen vom linken Kontext ab. Die Anzahl der Diphone ist je nach Sprache unterschiedlich, weil nicht immer alle möglichen Kombinationen auftreten müssen; so existieren im Italienischen beispielsweise /bv/ oder auch /vs/ nicht.

Triphone sind wie die Diphone kontextabhängig, jedoch sowohl links- als auch rechtsseitig.

Polyphone und Silben können kontextabhängig oder -unabhängig definiert sein. Werden häufig zusammen mit Diphonen und/oder Triphonen verwendet.

5.3 Implementierungstechniken

5.3.1 Logarithmische Wahrscheinlichkeiten

Die Techniken der Spracherkennung, die auf der Berechnung von Wahrscheinleikeiten als Maß für die Ähnlichkeit zwischen zwei Modellen basieren, erfordern spezielle Rechenmethoden. Es werden reelle Zahlen zwischen 0 und 1 behandelt, deren Unterschiede in der Größenordnung die Anwendung der üblichen Gleitkommarechnung nicht zulassen. In einem solchen Fall kann man eine *logarithmische Repräsentierung* wählen, die eine breite Streuung der vorkommenden Werte mit ausreichender Genauigkeit und vermindertem Speicheraufwand erlaubt.

In [19] finden wir eine ausführliche Abhandlung dieses Themas. In unserem Falle ist jedoch nur die Durchführung von Multiplikations- und Divisionsoperationen mit ausschließlich positiven Zahlen von Interesse.
Eine logarithmische Darstellung einer reellen Zahl x erhält man mittels der Funktion $F : \mathbf{R} \rightarrow \mathbf{R}$, die wie folgt definiert ist:

$$F(x) = \log_\xi x.$$

Bezeichnen dann K_x und K_y die internen logarithmischen Darstellungen der Zahlen x und y, $e_x = |K_x - \log_\xi(x)|$ und $e_y = |K_y - \log_\xi(y)|$ den absoluten Fehler[5], erhält man als Produkt bzw. Division folgendes:

$$\begin{aligned} x \cdot y \quad &\Rightarrow \quad K_x + K_y \\ \frac{x}{y} \quad &\Rightarrow \quad K_x - K_y, \end{aligned}$$

und für den absoluten Fehler

$$|K_x + K_y - \log_\xi(x \cdot y)| = |K_x - K_y - \log_\xi(\frac{x}{y})| \leq e_x + e_y.$$

Die resultierende Streuung der Werte hängt von der gewählten Basis ξ für die Berechnung des Logarithmus und vom internen Format für die Darstellung der Logarithmen ab. Wählt man eine ganzzahlige Darstellung mit n Bit, stehen Felder zwischen 0 und $2^n - 1$ für die Logarithmen $F(x)$ der Zahlen x zur Verfügung. Die mögliche Streuung für x in Zehnerpotenzen beträgt demnach:

$$D = \log_{10} \frac{\max(x)}{\min(x)} = \log_{10} \frac{\xi^{2^n-1}}{1}.$$

Wenn $2^n \gg 1$ ergibt sich:

$$D \approx 2^n \log_{10} \xi.$$

Für $n = 16$ und $\xi = 10$ erhält man eine Streuung von 65536 Größenordnungen.
In den Programmen zur Spracherkennung verwendet man $\xi = 10$ und intern eine Gleitkomma-Darstellung, die zwar zu einer überdimensionierten Streuung, aber auch zu einer erhöhten Verarbeitungsgeschwindigkeit führt[6].

[5] abhängig von der Registergröße und den Rundungen bei der Berechnung des Logarithmus'

[6] Die Möglichkeit der Parallelausführung zwischen FPU und CPU bei modernen Rechnern bietet ein günstigeres Verhalten für Operationen zwischen Gleitkommazahlen als zwischen ganzzahligen Werten.

5.3.2 Beam-Search

Im Kapitel 3 über Markov-Modelle haben wir zwei fundatmentale Algorithmen zur Bestimmung der Ähnlichkeit zwischen einer Beobachtungssequenz X und einem HMM λ kennengelernt. Sowohl für den Forward-Backward- als auch für den Viterbi-Algorithmus war es jedoch notwendig zu einem diskreten Zeitpunkt jeweils sämtliche Zustände eines Modelles und das parallel für alle HMM zu betrachten. Diese sogenannte Full-Search führt zu einem erheblichen Rechenaufwand für die Algorithmen. Die Technik der Beam-Search (auch: Methode der reduzierten Trellis) hat nun die Aufgabe, diesen Rechenaufwand zu mindern. Die Idee dahinter ist es, die Suche auf Knoten zu beschränken, die für die weitere Entwicklung der Trellis bedeutsam sind. Wir bezeichnen mit $\delta_t(i)$ die Wahrscheinlichkeiten beim Viterbit-Algorithmus und nehmen an, daß für die Wahrscheinlichkeiten $\gamma_t(i)$ der Forward-Backward-Prozedur ähnliche Überlegungen gelten.
Für ein Links-Rechts-HMM ohne Skip mit N Zuständen stellt sich die komplette Trellis für eine Sequenz der Länge T wie in Bild 5.2 da.

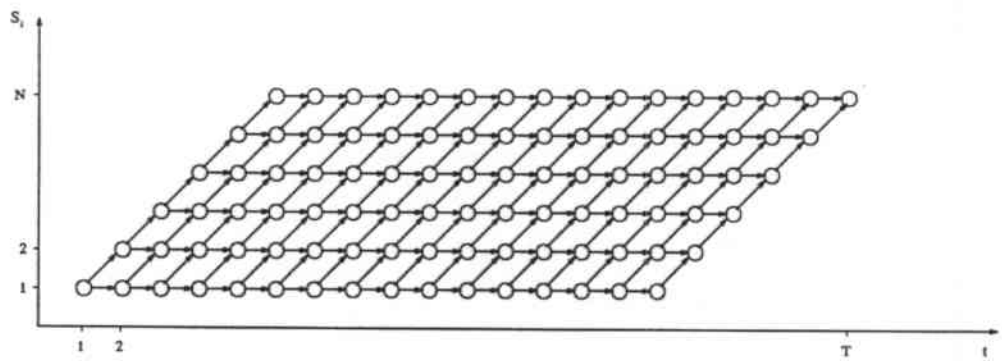

Abbildung 5.2. Vollständige Trellis für ein Links-Rechts-HMM ohne Skip.

Diese Trellis hat die typische Form eines Parallelogrammes und besteht aus $N \cdot (T-N+1)$ Knoten, wobei man für jeden Zustand S_i mit $\delta_t(i) > 0$ einen aktiven Knoten annimmt. Beam-Search basiert nun auf der Beobachtung, daß die Funktion $\delta_t(i)$ für festes t und variables i eine Glockenform hat, mit Zentrum um den Index i_{max}, den Zustand, der die größte Wahrscheinlichkeit hat, aktiv zu sein (Bild 5.3).

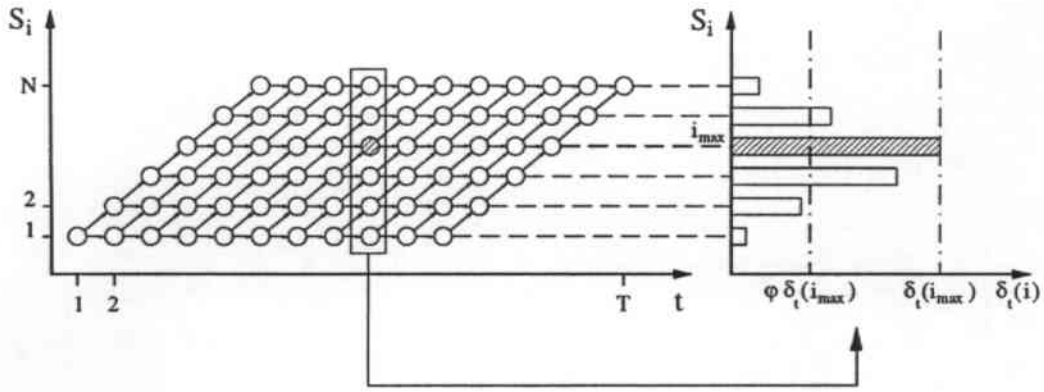

Abbildung 5.3. Typische Form von $\delta_t(i)$ mit festem t und variablem i. Der schwarz gefärbte Knoten hat die höchste Wahrscheinlichkeit, aktiv zu sein.

Ist $S_{i_{max}}$ der Zustand der $\delta_t(i)$ maximiert, dann werden diejenigen Knoten deaktiviert, deren Wahrscheinleichkeit, aktiv zu sein, unterhalb der von $\delta_t(i_{max})$ festgelegten Grenze liegt; d. h. S_i ist vernachlässigbar, wenn

$$\delta_t(i) \leq \varphi \cdot \delta_t(i_{max}) \qquad 0 \leq \varphi \leq 1.$$

Für einen solchen Zustand S_i unterhalb der Grenze wird dann die entsprechende Wahrscheinlichkeit $\delta_t(i)$ auf 0 gesetzt; der so deaktivierte Zustand ist also weder als Emissionsquelle noch als Basis für Übergänge zu anderen Zuständen im nächsten Schritt verfügbar. Es kann vorkommen, daß alle Zustände eines Modelles unterhalb der festgelegten Grenze liegen; d. h. das ganze Modell ist inaktiv. Die geeignete Wahl von φ verringert den Rechenaufwand drastisch und führt in der Regel zu einer reduzierten Trellis wie in Bild 5.4.

5.4 Der Viterbi-Scorer

Wie schon in der Einführung zu diesem Kapitel erwähnt, wurde die Viterbi-Prozedur für die verbundene Sprache des Wordspotters aus einem Viterbi-Dekodierer für isolierte Worte abgeleitet. Dieser Algorithmus soll nun an erster Stelle betrachtet werden; darauf aufbauend stellen wir dann den eigentlich verwendeten Viterbi-Forward-Scorer vor.

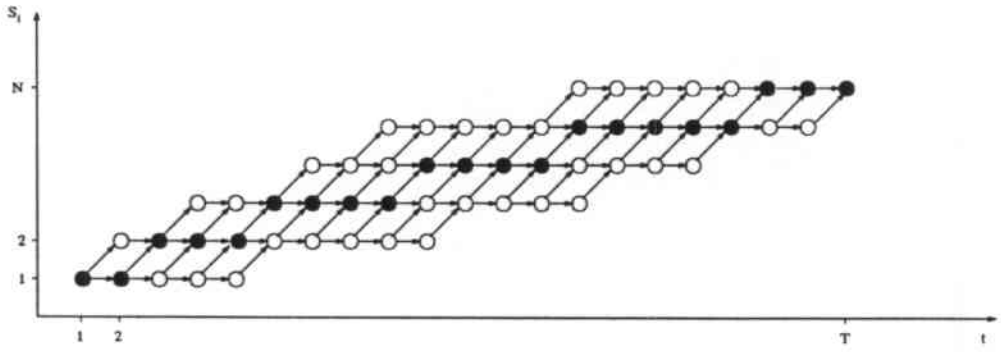

Abbildung 5.4. Durch Beam-Search reduzierte Trellis; die geschwärzten Knoten maximieren $\delta_t(i)$.

5.4.1 Die Viterbi-Prozedur für isolierte Worte

Bei der Behandlung einer Sequenz erzeugt der Viterbi-Forward-Algorithmus verschiedene Hypothesen für die Übereinstimmung der Modelle mit der Eingabe. Er berechnet dabei parallel für alle R Referenz-HMM, mit N_r Zuständen ($r = 1, \ldots, R$), die aus Kapitel 3 bekannten α-Variablen. Dies geschieht aufeinanderfolgend für jeden diskreten Zeitpunkt, wobei für jedes Modell die bekannte Tellis in Form eines Parallelogrammes entsteht (vgl. Bild 5.2) entsteht.
Wir verwenden jedoch die in Abschnitt 5.3.2 eingeführte Beam-Search, die den Suchraum auf einen Bereich um die Hauptdiagonale der vollen Gitterstruktur reduziert (vgl. Bild 5.4). Da wir die logarithmisierten Wahrscheinlichkeiten $\delta_t^\star(i)$ für einen Zustand i nicht normalisieren, bestimmen wir für jeden Zeitpunkt t den maximalen Score $\delta_t^\star(i_{max})$ über alle Zustände[7] und eliminieren einen Zustand i, wenn gilt:

$$\delta_t^\star(i) \leq \delta_t^\star(i_{max}) - \varphi^\star,$$

wobei $\varphi^\star$ eine festgewählte Konstante ist, in unserem Falle $\varphi^\star = 100$.
Der Unterschied zwischen der Formel in diesem Abschnitt und derjenigen im vorhergehenden beruht auf der Verwendung der logarithmischen Scores, die auch das Problem des Underflows beheben.

[7]Jeder Zustand wird dabei mit einem eigenen Index versehen.

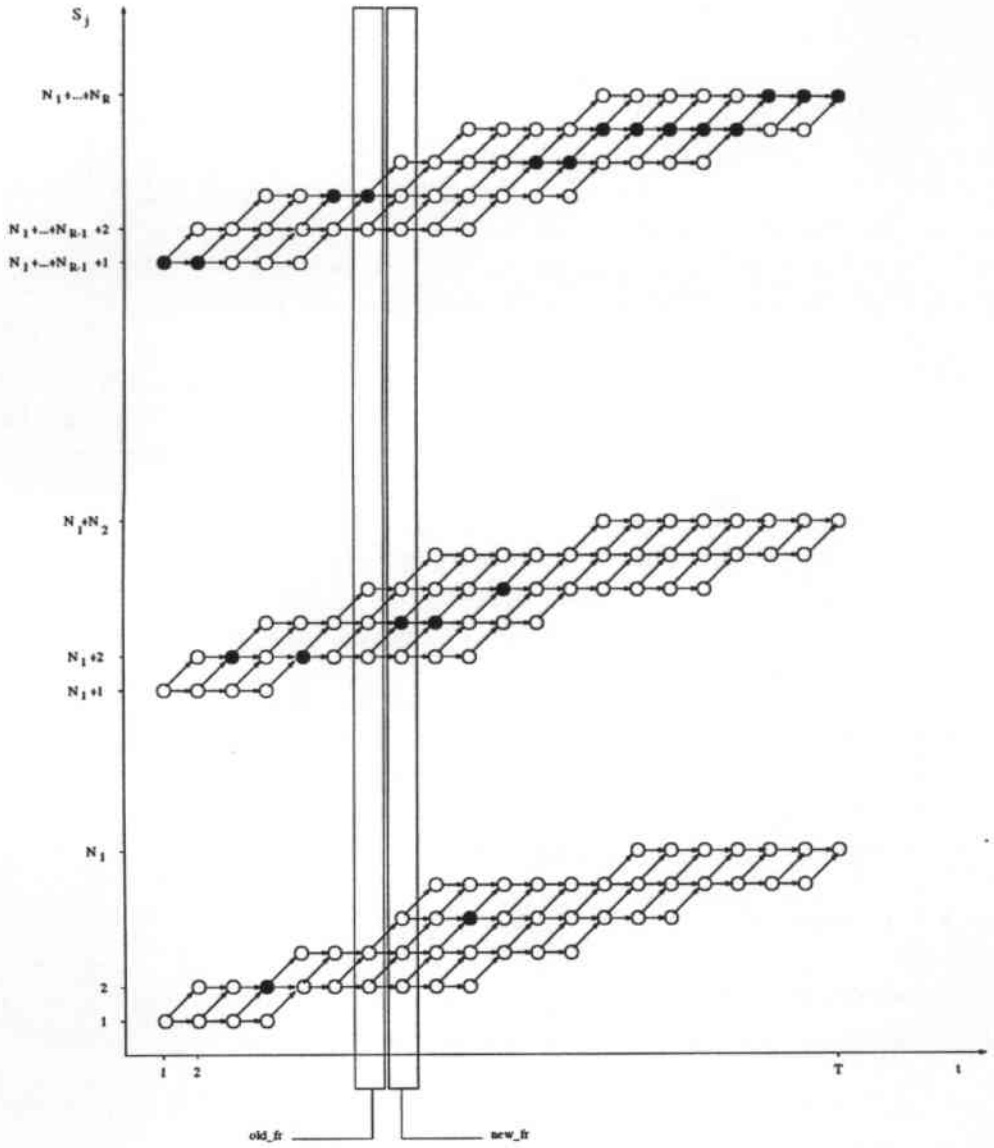

Abbildung 5.5. Reduzierte Trellis parallel für verschiedene Zustände mit Hervorhebung der zwei signifikanten Spalten.

Eine weitere Reduzierung des Suchraumes wird durch das Voranschreiten der

Prozedur entlang der zeitdiskreten Beobachtungspunkte bedingt. Für die Berechnung der Forward-Variablen $\alpha_t(i)$ sind nur der aktuelle (`new_fr`) und der vorangehende (`old_fr`) Frame notwendig, für die Beam-Search nur der momentane; deshalb läßt sich die Trellis auf jeweils zwei Spalten verringern (siehe Bild 5.5).
Das Vorgehen des Viterbi-Algorithmus ist in Bild 5.6 schematisch illustriert. Es fällt dabei auf, daß beim ersten Frame keine Beam-Search durchzuführen ist, da die Scores eben erst initialisiert wurden.
Im Anhang A.2 finden wir das komplette Listing der Viterbi-Prozedur. An dieser Stelle sei auf einige Besonderheiten derselben hingewiesen:

- Die Matrix **tr_score[][2]** enthält für jeden Zustand den Score des aktuellen und des vorhergehenden Frames und realisiert somit die Idee, daß zu jedem diskreten Zeitpunkt jeweils nur zwei Spalten betrachtet werden müssen.

- **tr_active[][2]** ist ein Matrix vom Typ `boolean`, die für den vorausgehenden und den momentanen Zeitpunkt festhält, ob ein Zustand von der Beam-Search deaktiviert wurde und somit in der weiteren Berechnung übergangen werden kann.

- Die Prozedur **preaccum()** berechnet die Emissionswahrscheinlichkeit $b_j(\vec{v})$ eines Vektors $\vec{v}$ Cepstralkoeffizienten für einen bestimmten Zustand S_j zu einem betimmten Zeitpunkt t mittels einer Linearkombination M verschiedener Gauß-Mischungen.

- Die Viterbi-Prozedur verwendet folgende globalen Variablen:

  nwords Anzahl der Referenzmodelle.

  num_sta Gesamtzahl aller Zustände in den HMM.

  vet_iaddr[] Vektor der Indizes des Startzustandes für jedes Modell.

  nframes Anzahl der Frames in einer Sequenz.

  tra[][] Matrix der Zustandsübergangswahrscheinlichkeiten

  vet_prob[] Matrix für die Hypothesen.

Es sei daraufhingewiesen, daß der größte Teil der Rechenleistung in der Prozedur **preaccum()** geleistet wird.

```
procedure: viterbi_fwd_iso()

begin
    Initialisierungen
    loop für alle Frame
        if Anfangsframe
            loop für alle Modelle
                Berechnung des Scores des Startzustandes;
            endloop
        else
            loop für alle Modelle
                loop für alle Zustände
                    if Anfangszustand
                        Neuberechnung des Scores für den Startzustand (nur
                        Self-Loop);
                    else
                        if Zustand ist aktiv
                            Neuberechnung des Scores für alle aktiven
                            Zustände;
                    endif
                endloop
            endloop
            loop für jeden aktiven Zustand (Beam-Search)
                if Score ist unterhalb des Schwellwertes
                    Deaktivierung des Zustandes;
            endloop
        endif
        loop für alle Modelle
            if Endzustand ist aktiv
                Übergang in absorbierenden Zustand und Ausgabe des Wahr-
                scheinlichkeitsscores
            endif
        endloop
    endloop
end.
```

Abbildung 5.6. Viterbi-Forward-Algorithmus für isolierte Worte.

5.4.2 Die Viterbi-Prozedur für den Wordspotter

Wir wollen hier zunächst den Algorithmus selbst vorstellen und dann noch genauer auf zwei neue Aspekte eingehen, die aus den Bedürfnissen des Wordspotters entspringen: Verschiedene Scoring-Methoden und die Erzeugung

eines Garbage-Modelles.

5.4.2.1 Der Aufbau der Viterbi-Prozedur

Der im vorangehenden Abschnitt vorgestellte Viterbi-Algorithmus wurde nun auf die Anforderungen des Wordspotting für verbundene Worte angepaßt. Seine Aufgabe besteht nun nicht mehr darin, verschiedene Hypothesen für die Erkennung von Worten zu produzieren, sondern vielmehr soll er für jedes der Schlüsselworte eine Reihe von Scoring-Werten erzeugen, mit deren Hilfe man dann mögliche Treffer identifizieren kann. Für jedes Schlüsselwortmodell wird dabei ein spezieller Wordspotting-Wahrscheinlichkeitswert berichtet, sobald der Endzustand seines HMM aktiv ist.
Die Grundstruktur dieses Viterbi-Mechanismus' ist in Bild 5.7 dargestellt. Man erkennt die folgenden wesentliche Unterschiede:

- Zu jedem Zeitpunkt t wird der Startzustand eines jeden Schlüsselwortmodelles neu initialisiert. Eine Normalisierung der Wahrscheinlichkeitswerte erreicht man dadurch, daß die Initialisierung mit dem höchsten Score aller Schlüsselwortzustände des vorangehenden Frames $t-1$ vorgenommen wird.

- Für jeden Zustand j wird zusätzlich zum Score und der Angabe über (De-)Aktivierung noch der Anfangszeitpunkt des Pfades gespeichert, der zu diesem Knoten geführt hat.

- Einen zusätzlichen Dynamic-Programming-Schritt. Zu jedem Zeitpunkt t und für jeden Zustand j wird nur der beste Pfad verfolgt; d. h. für jeden Frame entscheidet man sich bei jedem Zustand entweder für den Weg, der von einer Self-Loop dieses Zustandes kommt (der Anfangszeitpunkt bleibt demnach unverändert), oder man wählt denjenigen aus, der vom vorhergehenden Zustand ausgeht (unter Anpassung des Startzeitpunktes). Eine Illustration dieses Mechanismus' ist in Bild 5.8 zu finden.

- Der Übergang zum absorbierenden Zustand fällt weg, da für ein Schlüsselwort ein Scoring-Wert immer dann berichtet wird, wenn der Endzustand des zugehörigen HMM aktiv ist. Wir stellen später noch verschiedene Methoden vor, diesen speziellen Wordspotting-Score zu berechnen (siehe Abschnitt 5.4.2.2).

Das vollständige Listing dieser Prozedur ist im Anhang A.4 abgedruckt. An dieser Stelle einige Bemerkungen dazu:

```
procedure: viterbi_fwd_wsp()

begin
    Initialisierungen
    loop für alle Frame
        if Anfangsframe
            loop für alle Schlüsselwortmodelle
                Berechnung des Scores für den Startzustand;
                Festhalten des Startframes als Anfang des Pfades;
                Festlegung des maximalen Scores im Startframe;
            endloop
        else
            loop für alle Schlüsselwortmodelle
                Neuinitialisierung des Startzustandes mit dem maximalen Score des vor-
                angehenden Frames;
            endloop
            loop für alle Schlüsselwortmodelle
                loop für alle Zustände
                    if Anfangszustand
                        Berechnung des Scores für Self-Loop;
                        if Wert für Neuinitialisierung > Wert für Self-Loop
                            (explizite) Anpassung des Anfangs des Pfades zum
                            Startzustand;
                        Aktualisierung des maximalen Scores in diesem Frame;
                    else
                        if Zustand ist aktiv
                            Berechnung des Scores;
                            (implizite) Anpassung des Scores und des Pfadanfanges;
                    endif
                endloop
            endloop
            loop für jeden aktiven Zustand (Beam-Search)
                if Score ist unterhalb des Schwellwertes
                    Deaktivierung des Zustandes;
            endloop
        endif
        loop für alle Schlüsselwortmodelle
            if Endzustand ist aktiv
                Berechnung des speziellen Wordspotting-Scores;
            endif
        endloop
    endloop
end.
```

Abbildung 5.7. Viterbi-Forward-Algorithmus für Wordspotting.

- Zu den globalen Variablen der Viterbi-Prozedur für isolierte Worte kommt noch der Vektor **key[]** hinzu, der die Indizes der verwendeten Schlüsselworte enthält.
- Die Matrix **da_dove[][2]** enthält den Pfadanfang für jeden Zustand im

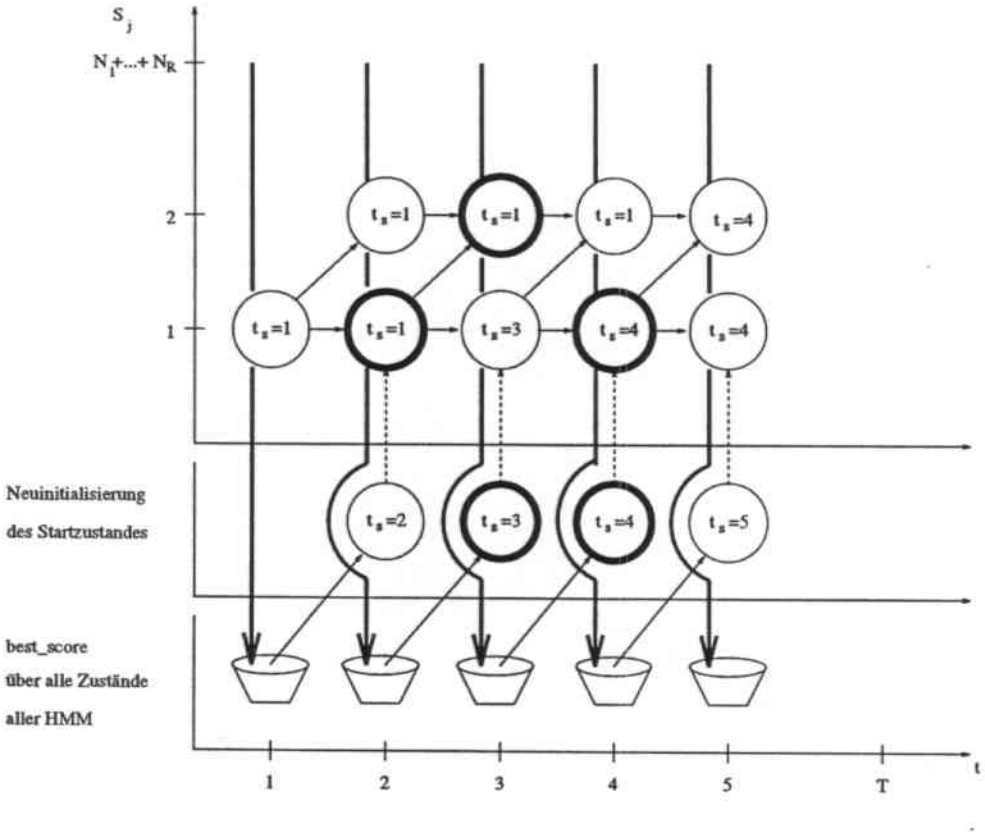

Abbildung 5.8. Neuinitialisierung des Startzustandes mit dem maximalen Score des vorausgehenden Frames und Anpassung des Pfadanfanges für jeden Zustand (an manchen Stellen sind die "Gewinner"-Zustände fett umrandet).

aktuellen und im vorausgehenden Frame.

- Der Vektor **best_score[]** beinhaltet die maximalen Scores für jeden Frame.
- Am Ende der Prozedur werden noch Werte für ein Garbage-Modell berechnet; dieses wollen wir uns im Abschnitt 5.4.2.3 genauer ansehen.

5.4.2.2 Das Scoring

Hier folgen wir dem von ROSE [23] [24] (s. a. Abschnitte 4.3.1.1.1 und 4.3.2.1) eingeschlagenen Weg, sowohl beim primären als auch beim sekundären Scoring (s. Abschnitt 5.5.4).

5.4.2.2.1 Primäres Scoring Zur ersten Bewertung der Eingabesequenzen dient uns der zeitnormalisierte Wahrscheinlichkeitsscore

$$W_{t_s}^{t_e}(w_n) = \frac{\log P(S_{t_e}, (\vec{x}_{t_s}, \ldots, \vec{x}_{t_e}))}{t_e - t_s}$$

für ein Schlüsselwort w_n. Da — durch den Aufbau des Wordspotters bedingt — eine unmittelbare Meldung über das Auftreten eines Schlüsselwortes nicht vorgesehen ist, verzichten wir jedoch auf die Anwendung des *partial tracebacks*, sondern berichten die Wahrscheinlichkeitswerte $W_{t_s}^{t_e}$ als Ergebnisse der Viterbi-Scoring-Prozedur. (Mögliche Schlüsselworttreffer sind in den lokalen Maxima dieser Kurve zu suchen.) Die zweite Stufe des Spotters nimmt dann die genauere Analyse dieser Resultate vor.
Eine Implementation dieser Scoring-Methode im Anhang A.5 zu finden; einen typischen Verlauf der Kurve zeigt Bild 5.9. Befindet sich zu einem bestimmten Zeitpunkt kein gültiger Pfad des Schlüsselwortmodelles im Endzustand, wird in der Viterbi-Prozedur für diesen Frame ein sehr kleiner negativer Score berichtet (MIN_SCORE); für den Plot ist dieser Wert auf -1500 gesetzt.

5.4.2.3 Erzeugung eines Garbage-Modelles

Wir verzichten darauf, ein Garbage-Modell explizit zu modellieren, sondern verwenden die Erkenntnisse von BOITE [5] [6] (vgl. a. Abschnitt 4.5.7), um die Emissionswahrscheinlichkeiten für ein Filler-Modell on-line zu berechnen. Die Ansätze in [5] [6], die N besten Emissionswahrscheinlichkeiten der kontextunabhängigen und kontextabhängigen phonembasierten HMM für die Berechnung des Alternativ-Scores zu jedem Zeitpunkt zu verwenden, führt in unserem Falle dazu, daß von allen im HMM-Netzwerk vorhanden Zuständen die N besten[8] herausgesucht werden und der Durchschnitt über diese Werte als Garbage-Score berichtet wird.
Da wir uns auf die Verwendung eines Schlüsselwortes beschränken, steht der Garbage-Score also in Konkurrenz zum Endzustand des betreffenden Keyword-HMM. Nur wenn dessen Score niedrig ist, wird er vom Filler-Modell überboten; dies ist in Bild 5.10 zu erkennen, wobei wir für die Anzahl der Zustände, über die der Durchschnitt gebildet wird, $N = 3$ gewählt haben.

[8] in Bezug auf die vom Viterbi-Algorithmus berechneten Forward-Variablen

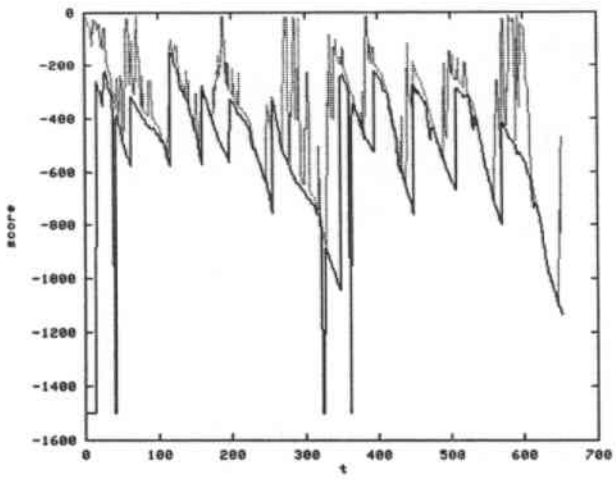

Abbildung 5.9. Verlauf der Kurve des normalisierten Wahrscheinlichkeitsscores (oben) und der Pfadwahrscheinlichkeit (unten) für Schlüsselwort "otto" (acht) in der Sequenz "nove-zero-sei-otto-otto-tre-due" (neun-null-sechs-acht-acht-drei-zwei).

5.5 Der Score-Analysierer

Dieser Teil des Wordspotters dient zur weiteren Untersuchung der vom Viterbi-Dekodierer errechneten Scores und läßt sich seinerseits in sechs Hauptschritte untergliedern:

1. Der **Peak-Detektor** dient zum Auffinden der Stellen, in denen ein lokales Maximum in der Folge der zeitnormalisierten Wahrscheinlichkeitsscores vorliegt.

2. Der **Schwellwert** für die vom Viterbi-Algorithmus berechneten Scores dient dazu, diejenigen Peaks zu eliminieren, die unterhalb dieser Grenze liegen.

3. Die verbleibenden lokalen Maxima werden in verschiedene **Cluster** eingruppiert.

4. Für jeden Cluster wird nun der **sekundäre Score** berechnet.

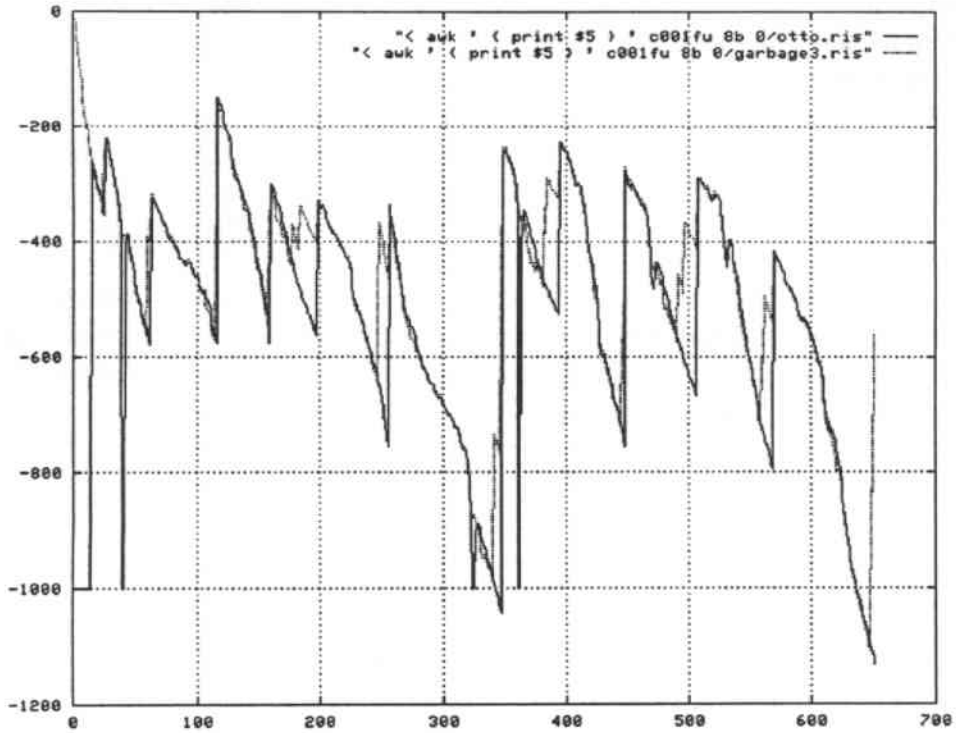

Abbildung 5.10. Schlüsselwortscore und On-line-Garbage mit $N = 3$.

5. Auch diese Werte werden nun mit einem **zweiten Grenzwert** verglichen und gegebenenfalls nicht weiter untersucht.

6. Schliesslich werden noch **zeitliche Überlappungen** der möglichen Schlüsselwortkandidaten beseitigt.

5.5.1 Der Peak-Detektor

Diese Prozedur wurde mittels eines endlichen Automaten mit 4 Zuständen realisiert; sein Zustandsübergangsdiagramm ist in Bild 5.11 dargestellt, das Listing findet sich im Anhang B.1 wieder. Anfangszustand ist 0, einen expliziten Endzutand gibt es nicht, die Terminierung fällt mit dem Ende der Eingabesequenz zusammen. (Man beachte: Befindet sich der Automat zu

Die noch gültigen Kandidaten kann man nun als Ergebnis berichten. Die gesamte Analysierungsprozedur (außer dem Peak-Detektor) ist im Anhang B.2 abgedruckt, die Definitionsdatei in B.3.

5.5.7 Resultate

Die erzielten Ergebnisse zeigen einige Schwierigkeiten in Zusammenhang mit der verwendeten Datenbasis auf:

- Das hohe Verhältnis zwischen Schlüsselworten und anderer Sprache und die daraus resultierende erhöhte Wahrscheinlichkeit von Verwechslungen erweist sich als ungünstig. So kommt es vor, daß ein Schlüsselwort für ein ähnliches Wort im Satz dekodiert wird, weil sein Score über den des Garbages dominiert (kritisch sind etwa: due-tre-nove, zero-uno), was zu einer gesteigerten False-Alarm-Rate führt. Auch die Gewichtung es Füller-Scores mittels eines konstanten Faktors ϱ brachte keine Besserung. Man scheint hier bei der Verwendung des On-Line-Garbages nicht um die Einführung einer Worteingangsstrafe als dynamischen Anpassungsprozeß des Alternativ-Scores herumzukommen.

- Auch gegenüber Lärm und Stille zeigten sich die verwendeten Modelle anfällig, so daß es zu Fehlkodierungen kam. Hier bietet sich eventuell an, den Gargabe explizit zu modellieren, um so eine bessere Repräsentation nicht-sprachlicher Ereignisse zu erhalten.

- Das Auftreten mehrerer gleicher Ziffern hintereinander (otto-otto) bereitet ebenfalls Probleme und führt zu Auslassungen (die zweite Ziffer wurde nicht erkannt).

In der folgenden Tabelle sind die Ergebnisse für das Schlüsselwort "otto" bei verschiedenen Werten der Sensibilitätskoeffizienten ρ_1 und ρ_2 des primären und des sekundären Scoring dargestellt.

keys	hits	missings	false alarms	ρ_1	ρ_2
176	29 (16.5%)	148	96	1.00	1.00
176	47 (26.7%)	130	212	1.07	0.80

Tabelle 5.1. Ergebnisse für Schlüsselwort "otto".

Das Testset umfaßt 96932 Frames (ca. 16 Minuten), die Schrankenwerte wurden über einem Trainingsset der Dauer von 58414 Frames (ca. 10 Minuten) ermittelt, wobei das Schlüsselwort "otto" 142 auftrat.

5.5.8 Alternative End-Point-Detection

Neben dem Verfahren, mögliche Schlüsselwortendpunkte an den Stellen lokaler Maxima der Scoring-Werte zu suchen, haben wir noch eine weitere Methode erprobt. Die Idee dabei beruht auf der Beobachtung, daß der referenzierte Anfangspunkt eines Pfades für einen gewissen Zeitraum stabil bleibt, auch wenn man über den eigentlichen Endpunkt hinausgeht. Dieses Verhalten ist in Bild 5.12 dargestellt. Hierbei wurden 18 Zustände zu 6 Gruppen aus je 3 Zuständen zusammengefaßt. Die Kurve zeigt die dekodierten Anfangspunkte. Man erkennt, daß am Ende einer Einheit vor dem Übergang zur nächsten der Anfangsframe sich nicht mehr ändert, was zum Auffinden möglicher Endpunkte benützt werden kann. Die Experimente haben jedoch gezeigt, daß dies in unserem Falle nicht den großen Durchbruch bringt.

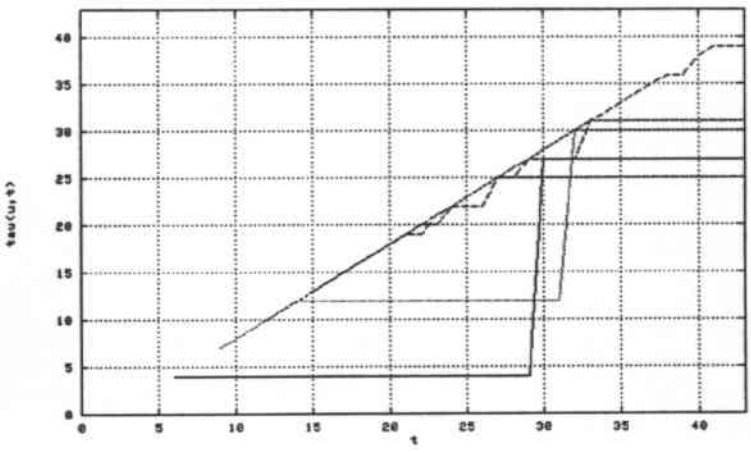

Abbildung 5.12. Stabilität des Anfangspunktes.

Anhang A

Listings der Viterbi-Prozeduren

A.1 viterbi_fwd_iso.h

Konstanten und Typen der folgenden Viterbi-Prozedur. (Es finden sich auch Definitionen wieder, die in anderen Teilen des Erkenners benützt werden.)

```
/* viterbi_fwd_iso.h
   costanti e tipi per il file viterbi_fwd_iso.c */

#define TRUE 1
#define FALSE 0

#define PI 3.14159265358979

/* min e max define */
#define min(a, b) (((a) <= (b)) ? (a) : (b))
#define max(a, b) (((a) >= (b)) ? (a) : (b))

/* dimensioni dei tipi */
#define SIZEC sizeof(char)
#define SIZES sizeof(short)
#define SIZEI sizeof(int)
#define SIZEL sizeof(long)
#define SIZEF sizeof(float)
#define SIZED sizeof(double)

/* limiti */
#define MX_STA  32 /* max # di stati diversi */
#define MX_FRA  300 /* max # di frame in una sequenza */
#define MX_PAR  24 /* # di coeffizienti cepstrali in un vettore */
#define MX_WORDS  80 /* max # di parole accettate nel lessico */
#define MX_MIX 32 /* # di misture gaussiane per ogni stato */
```

```
#define MX_STAWO  MX_STA*MX_WORDS
#define MX_LINE 81 /* max lunghezza di una linea */
#define MX_LEN 20 /* max lungehezza del nome di una parola */
#define WINDOW 5 /* ampiezza finestra per calcolo dep (sempre dispari) */
#define NWH 2 /* floor (WINDOW/2) */
#define LIM_GAUENE 1.e-15 /* limite della calcolazione della gauene */
#define LIM_UNDERFLOW 87.0
#define LIM_BEST_SCORE -1.e37 /* valore iniziale di best_score */
#define LIM_PRP -1.e37 /* valore standard della probabilita' di una parola */
#define ZERO 0.0
#define LIM_GAUVAR 1.e-30
#define VET_SIZ 5 /* grandezza del buffer per il carico delle gaussiane */
#define MX_FLT 14 /* max lunghezza di un float */
#define DOFI 28 /* # byte per un vettore con 14 elementi a due byte */

/* indici matrice probabilita' di transizione tra[] */
#define SELF 0 /* probabilita' di self looping */
#define NEXT 1 /* probabilita' di transire al prossimo stato */
#define SKIP 2 /* probabilita' di skip */
#define DIM 3 /* grandezza della seconda dimensione (SELF, NEXT, SKIP) */

/* Costanti per il liftering */
#define HEIGHT 0.5
#define BIAS 1.0

/* Costanti per la calcolazione di accum */
#define OMEGA_ENE 0.5
#define OMEGA_DELTAENE 2.0

/* Costante per la beam search */
#define BEAM_COST 100.0

/* Costante di normalizzazione per ricavare i parametri differenziali */
#define ALPHA 3.98

/* Altre costanti */

/* tipi */
typedef char line[80];
typedef char word[44];
```

A.2 viterbi_fwd_iso.c

Viterbi-Algorithmus für isolierte Sprache.

```
/*****************************************************************************/
/*****************************************************************************/
/* PROCEDURA: void viterbi_fwd_iso(void)                                     */
/*                                                                           */
/* PARAMETRI: nessuno                                                        */
/*                                                                           */
/* RITORNA: 0                                                                */
/*                                                                           */
/* DESCRIZIONE: La procedura implementa l'algoritmo di riconoscimento per il */
/*              parlato isolato.                                             */
/*                                                                           */
/* VARIABILI GLOBALI LETTE:                                                  */
/*           num_sta, nframes, nwords, vet_iaddr[], tra[][]                  */
/*                                                                           */
/* VARIABILI GLOBALI SCRITTE:                                                */
/*           vet_prob[]                                                      */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/
void viterbi_fwd_iso (void) {

    double  tr_score[MX_STAWO][2],
            aux_prb,
            pre_best,
            best_score,
            score_lim;

    short   old_fr = 0,
            new_fr = 1,
            tr_active[MX_STAWO][2],
            preaddr,
            sucaddr,
            inod,
            traind,
            vivid,
            ifr,
            istat,
            iaddr,
            iele,
            icol;

    int     i,
            imodel;

    for (i = 0; i < nwords; i++)
        vet_model[i] = i;
```

```
for (iele = 0; iele < num_sta; iele++) {
    for (icol = 0; icol < 2; icol++)
        tr_active[iele][icol] = FALSE;
}                           /* for(iele) */

for (ifr = 0; ifr < nframes; ifr++) {
    best_score = LIM_BEST_SCORE;

    if (ifr == 0) {
        for (imodel = 0; imodel < nwords; imodel++) {
            iaddr = vet_iaddr[imodel];
            tr_score[iaddr][old_fr] = pre_accum (iaddr, ifr);
            tr_active[iaddr][old_fr] = TRUE;
        }                   /* for(imodel) */
    }                       /* if(ifr) */
    else {
        for (imodel = 0; imodel < nwords; imodel++) {
            iaddr = vet_iaddr[imodel];
            for (istat = 0; istat < nstat[imodel]; istat++) {
                if (istat == 0) {
                    /* primo stato, solo self loop possibile */
                    if (tr_active[iaddr][old_fr]) {
                        tr_score[iaddr][new_fr] = tr_score[iaddr][old_fr] + tra[iaddr][SELF]
                                                  + pre_accum (iaddr, ifr);
                        tr_active[iaddr][new_fr] = TRUE;
                        if (tr_score[iaddr][new_fr] > best_score)
                            best_score = tr_score[iaddr][new_fr];
                    }       /* if(tr_active) */
                    else
                        tr_active[iaddr][new_fr] = FALSE;
                }           /* if(istat) */
                else {
                    pre_best = LIM_BEST_SCORE;
                    sucaddr = iaddr + istat;
                    /* # del stato succedente */

                    preaddr = sucaddr - 1;
                    /* # del stato precedente */

                    vivid = FALSE;

                    for (inod = 0; inod < 2; inod++) {
                        if (tr_active[preaddr + inod][old_fr])
                            vivid = TRUE;
                    }       /* for(inod) */

                    if (vivid) {
                        for (traind = NEXT; traind >= SELF; traind--) {
                            if (tr_active[preaddr][old_fr]) {
                                aux_prb = tr_score[preaddr][old_fr] + tra[preaddr][traind]
                                          + pre_accum (sucaddr, ifr);
                                if (aux_prb > pre_best)
                                    pre_best = aux_prb;
```

```
                                }/* if(tr_active) */
                                preaddr++;
                            }   /* for(traind) */
                            tr_score[sucaddr][new_fr] = pre_best;
                            tr_active[sucaddr][new_fr] = TRUE;
                            if (pre_best > best_score)
                                best_score = pre_best;
                        }       /* if(vivid) */
                        else
                            tr_active[sucaddr][new_fr] = FALSE;
                        /* else(vivid) */
                    }           /* else(istat) */
                }               /* for(istat) */
            }                   /* for(imodel) */

/* beam search */
            score_lim = best_score - BEAM_COST;
            for (iele = 0; iele < num_sta; iele++) {
                if (tr_active[iele][new_fr]) {
                    if (tr_score[iele][new_fr] < score_lim)
                        tr_active[iele][new_fr] = FALSE;
                }               /* if(tr_active) */

            }                   /* for(iele) */

/* cambio delle colonne */
            old_fr = TRUE - old_fr;
            new_fr = TRUE - new_fr;
        }                       /* else(ifr) */
    }                           /* for(ifr) */

/* stato finale - dummy trasizione */
    for (imodel = 0; imodel < nwords; imodel++) {
        preaddr = vet_iaddr[imodel] + nstat[imodel] - 1;
        if (tr_active[preaddr][old_fr]) {
            vet_prob[imodel] = tr_score[preaddr][old_fr] + tra[preaddr][NEXT];
        }                       /* if(tr_active) */
        else
            vet_prob[imodel] = LIM_PRP;
        /* else(tr_active) */

        if (vet_prob[imodel] != LIM_PRP)
            vet_prob[imodel] /= (double) nframes;

    }                           /* for(imodel) */

    return;

}                               /* viterbi_fwd_iso() */
```

A.3 viterbi_fwd_wsp.h

Konstanten und Typen der folgenden Viterbi-Prozedur. (Es finden sich auch Definitionen wieder, die in anderen Teilen des Erkenners benützt werden.)

```
/* viterbi_fwd_wsp.c
   costanti e tipi per il file viterbi_fwd_wsp.c */

#define TRUE 1
#define FALSE 0
#define SC 0
#define PP 1

#define MX_KWS 10
#define MX_KWSSTA 152
#define MX_N_BEST_MIN 3
#define MX_N_BEST_MAX 12

#define PI 3.14159265358979

/* min e max define */
#define min(a, b) (((a) <= (b)) ? (a) : (b))
#define max(a, b) (((a) >= (b)) ? (a) : (b))

/* dimensioni dei tipi */
#define SIZEC sizeof(char)
#define SIZES sizeof(short)
#define SIZEI sizeof(int)
#define SIZEL sizeof(long)
#define SIZEF sizeof(float)
#define SIZED sizeof(double)

/* limiti */
#define MX_STA  32 /* max # di stati diversi */
#define MX_FRA  800 /* max # di frame in una sequenza */
#define MX_PAR  24 /* # di coeffizienti cepstrali in un vettore */
#define MX_WORDS  80 /* max # di parole accettate nel lessico */
#define MX_MIX 32 /* # di misture gaussiane per ogni stato */
#define MX_STAWO  MX_STA*MX_WORDS
#define MX_LINE 81 /* max lunghezza di una linea */
#define MX_LEN 20 /* max lungehezza del nome di una parola */
#define WINDOW 5 /* ampiezza finestra per calcolo dep (sempre dispari) */
#define NWH 2 /* floor (WINDOW/2) */
#define LIM_GAUENE 1.e-15 /* limite della calcolazione della gauene */
#define LIM_UNDERFLOW 87.0
#define LIM_BEST_SCORE -1.e37 /* valore iniziale di best_score */
#define LIM_PRP -1.e37 /* valore standard della probabilita' di una parola */
#define ZERO 0.0
#define LIM_GAUVAR 1.e-30
#define VET_SIZ 5 /* grandezza del buffer per il carico delle gaussiane */
#define MX_FLT 14 /* max lunghezza di un float */
```

```
#define DOFI 28 /* # byte per un vettore con 14 elementi a due byte */
#define MX_LEN_NAME MX_LINE+MX_LEN+8
#define MX_FLFIELD 16 /* lunghezza del campo per first e last */
#define MX_TAGFIELD 4 /* lunghezza del campo per il tag */
#define MIN_SCORE -1000.0 /* min score per una parola, sarebbe pp = 0.0 */

/* indici matrice probabilita' di transizione tra[] */
#define SELF 0 /* probabilita' di self looping */
#define NEXT 1 /* probabilita' di transire al prossimo stato */
#define SKIP 2 /* probabilita' di skip */
#define DIM 3 /* grandezza della seconda dimensione (SELF, NEXT, SKIP) */

/* Costanti per il liftering */
#define HEIGHT 0.5
#define BIAS 1.0

/* Costanti per la calcolazione di accum */
#define OMEGA_ENE 0.5
#define OMEGA_DELTAENE 2.0

/* Costante per la beam search */
#define BEAM_COST 100.0

/* Costante di normalizzazione per ricavare i parametri differenziali */
#define ALPHA 3.98

/* Altre costanti */

/* tipi */
typedef char line[80];
typedef char word[44];
```

A.4 viterbi_fwd_wsp.c

Viterbi-Algorithmus für verbundene Sprache.

```
/*****************************************************************************/
/*****************************************************************************/
/* PROCEDURA: void viterbi_fwd_wsp ()                                        */
/*                                                                           */
/* PARAMETRI:   nessuno                                                      */
/*                                                                           */
/* RITORNA: 0                                                                */
/*                                                                           */
/* DESCRIZIONE: La procedura implementa l'algoritmo di riconoscimento per il */
/*              parlato continuo in un wordspotter.                          */
/*                                                                           */
/* VARIABILI GLOBALI LETTE:                                                  */
/*           num_sta, nframes, nwords, vet_iaddr[], tra[][], key[]           */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/
void viterbi_fwd_wsp (void) {

    double  tr_score[MX_STAWO][2],
            print_score,
            aux_prb,
            pre_best,
            best_score[MX_FRA],
            score_lim,
            inter,
            pp_max = 0,
            n_best[MX_KWSSTA],
            garbage_score[MX_N_BEST_MAX - MX_N_BEST_MIN + 1],
            r1dummy;

    short   old_fr = 0,
            new_fr = 1,
            tr_active[MX_STAWO][2],
            preaddr,
            sucaddr,
            inod,
            traind,
            vivid,
            ifr,
            istat,
            iaddr,
            iele,
            icol,
            ikey,
            da_dove[MX_STAWO][2],
            peak;
```

```
int     i,
        j,
        k,
        imodel,
        spot;

for (ifr = 0; ifr < nframes; ifr++) {
    for (iele = 0; iele < (short) kwssta; iele++) {
        for (icol = 0; icol < 2; icol++) {
            tr_active[iele][icol] = FALSE;
            tr_score[iele][icol] = 0.0;
          } /* for (icol) */
    }                       /* for(iele) */
}                           /* for(ifr) */


for (ifr = 0; ifr < nframes; ifr++) {
    best_score[ifr] = LIM_BEST_SCORE;

    if (ifr == 0) {
      ikey = 0;
      for (imodel = key[ikey]; imodel < sum_key; imodel+= key[++ikey]) {
        iaddr = vet_iaddr[imodel];
        pre_best = pre_accum (imodel, iaddr, ifr);
        tr_score[iaddr][old_fr] = pre_best;
        da_dove[iaddr][old_fr] = ifr;
        tr_active[iaddr][old_fr] = TRUE;
        if (pre_best > best_score[ifr])
          best_score[ifr] = pre_best;
      }                     /* for(imodel) */
    }                       /* if(ifr) */
    else {
      ikey = 0;
      for (imodel = key[ikey]; imodel < sum_key; imodel+= key[++ikey]) {
        iaddr = vet_iaddr[imodel];
        tr_score[iaddr][new_fr] = pre_accum (imodel, iaddr, ifr) + best_score[ifr - 1];
        tr_active[iaddr][new_fr] = TRUE;
      }                     /* for(imodel) */

      ikey = 0;
      for (imodel = key[ikey]; imodel < sum_key; imodel+= key[++ikey]) {
        iaddr = vet_iaddr[imodel];
        for (istat = 0; istat < nstat[imodel]; istat++) {
          if (istat == 0) {
            /* primo stato, solo self loop possibile */
            if (tr_active[iaddr][old_fr]) {
              inter = tr_score[iaddr][old_fr] + tra[iaddr][SELF]
                      + pre_accum (imodel, iaddr, ifr);
              if (inter > tr_score[iaddr][new_fr])
                /* vecchio cammino e' meglio */
                tr_score[iaddr][new_fr] = inter;
              else  /* nuovo cammino e' meglio */
                da_dove[iaddr][new_fr] = ifr;
```

```
          /* tr_active[iaddr][new_fr] = TRUE; */

          if (tr_score[iaddr][new_fr] > best_score[ifr])
            best_score[ifr] = tr_score[iaddr][new_fr];

        }       /* if(tr_active) */

      }         /* if(istat) */
      else {
        pre_best = LIM_BEST_SCORE;
        sucaddr = iaddr + istat;
        /* # dello stato succedente */

        preaddr = sucaddr - 1;
        /* # dello stato precedente */

        vivid = FALSE;

        for (inod = 0; inod < 2; inod++) {
          if (tr_active[preaddr + inod][old_fr])
            vivid = TRUE;
        }       /* for(inod) */

        if (vivid) {
          for (traind = NEXT; traind >= SELF; traind--) {
            if (tr_active[preaddr][old_fr]) {
              aux_prb = tr_score[preaddr][old_fr] + tra[preaddr][traind]
                        + pre_accum (imodel, sucaddr, ifr);
              if (aux_prb > pre_best) {
                pre_best = aux_prb;
                da_dove[sucaddr][new_fr] = da_dove[preaddr][old_fr];

              }/* if (aux_prb) */

            }/* if(tr_active) */
            preaddr++;
          }     /* for(traind) */
          tr_score[sucaddr][new_fr] = pre_best;
          tr_active[sucaddr][new_fr] = TRUE;
          if (pre_best > best_score[ifr])
            best_score[ifr] = pre_best;
        }       /* if(vivid) */
        else
          tr_active[sucaddr][new_fr] = FALSE;
        /* else(vivid) */
      }         /* else(istat) */
    }           /* for(istat) */
  }
}                       /* for(imodel) */

/* beam search */
score_lim = best_score[ifr] - BEAM_COST;
for (iele = 0; iele < (short) kwssta; iele++) {
```

```
    if (tr_active[iele][new_fr]) {
      if (tr_score[iele][new_fr] < score_lim)
        tr_active[iele][new_fr] = FALSE;
    }           /* if(tr_active) */
  }                     /* for(iele) */

} /* else (ifr) */

/* calcolazione della probabilita' normalizzata nel tempo */

/* se ifr = 0, allora cambio delle colonne, cosi' si puo' utilizzare
   la stessa routina per l'output, dopo ancora una volta un cambio
   delle colonne per produrre lo stesso stato come prima */

if (ifr == 0) {
  old_fr = TRUE - old_fr;
  new_fr = TRUE - new_fr;
} /* if (ifr) */

ikey = 0;
for (imodel = key[ikey]; imodel < sum_key; imodel+= key[++ikey]) {
  spot = vet_iaddr[imodel] + nstat[imodel] - 1;
  if (tr_active[spot][new_fr]) {
    if (da_dove[spot][new_fr] == 0)
      post_prob[imodel][new_fr] = tr_score[spot][new_fr] / (ifr + 1);
    else
      post_prob[imodel][new_fr] = (tr_score[spot][new_fr] -
        best_score[da_dove[spot][new_fr] - 1])
        / (ifr - da_dove[spot][new_fr] + 1);
  }             /* if(tr_active[spot]) */
  else {
    post_prob[imodel][new_fr] = MIN_SCORE;
    da_dove[spot][new_fr] = -1;

  }             /* else(tr_active[spot]) */

  if (da_dove[spot][new_fr] == -1)
    print_score = MIN_SCORE;
  else {
    if (da_dove[spot][new_fr] == 0)
      print_score = tr_score[spot][new_fr];
    else
      print_score = tr_score[spot][new_fr] - best_score[da_dove[spot][new_fr] - 1];
  } /* else(da_dove) */

  /*printf("numero  = %d\tframe = %d\tda_dove = %d\tpp = %le\n",
    (imodel + 1), ifr, da_dove[spot][new_fr], post_prob[imodel][new_fr]);*/
  fprintf(fris, "numero  = %d\tframe = %d\tda_dove = %d\tpp = %le\tscore = %le\n",
    (imodel + 1), ifr, da_dove[spot][new_fr], post_prob[imodel][new_fr], print_score);

}                       /* for(imodel) */

/* calcolazione e stampa per il modello garbage */
```

```
/* score per il modello garbage */

k = 0;
for (iele = 0; iele < (short) kwssta; iele++) {
  if (tr_active[iele][new_fr]) {
    if (da_dove[iele][new_fr] == 0)
      {
        n_best[k] = tr_score[iele][new_fr];
      } /* if(da_dove) */
    else
      {
        n_best[k] = tr_score[iele][new_fr] - best_score[da_dove[iele][new_fr] - 1];
      } /* else */
  } /* if(tr_active) */
} /* for(iele) */

for (i = 0; i < (k - 1); i++) {
  for (j = (k - 1); j > i; j--) {
    if (n_best[j] > n_best[j - 1]) {
      r1dummy = n_best[j][SC];
      n_best[j] = n_best[j - 1][SC];
      n_best[j - 1] = r1dummy;
    }                    /* if(vet_prob[]) */
  }                      /* for(j) */
}                                /* for(i) */

for (j = (short) n_best_min; j <= (short) n_best_max; j++) {
  garbage_score[j - (short) n_best_min] = 0.0;
  garbage_pp[j - (short) n_best_min] = 0.0;
  if (k > 0) {
    for (i = 0; i < j; i++) {
      if (i < k) {
        garbage_score[j - (short) n_best_min] += n_best[i];
      } /* if(i < k) */
    } /* for (i) */
  } /* if(k>0) */ /* esiste almeno un elemento con score > MIN_SCORE */
  else {
    garbage_score[j - (short) n_best_min] = MIN_SCORE;
    garbage_pp[j - (short) n_best_min] = MIN_SCORE;
  } /* else */
  (j <= k)?(garbage_score[j - (short) n_best_min] /= j):
           (garbage_score[j - (short) n_best_min] /= k);
  /* per casi in cui ci sono meno stati attivi del numero di N_BEST */

  fprintf(fris, "garbage = %d\tframe = %d\tda_dove = %d\t
    pp = %le\tscore = %le\n",j, ifr, ifr, garbage_pp[j
    - (short) n_best_min], garbage_score[j
    - (short) n_best_min]);

} /* for (j) */

 fprintf(fris, "best_sc = %e\n", best_score[ifr]);
```

```
        /* cambio delle colonne */
        old_fr = TRUE - old_fr;
        new_fr = TRUE - new_fr;

      }                          /* for(ifr) */

    return;

}                                /* viterbi_fwd_wsp() */
```

A.5 rohlicek_scoring.c

Berechnung der a posteriori Wahrscheinlichkeit nach der Methode von ROSE.

```
/* calcolazione della probabilita' normalizzata nel tempo */

/* se ifr = 0, allora cambio delle colonne, cosi' si puo' utilizzare
   la stessa routina per l'output, dopo ancora una volta un cambio
   delle colonne per produrre lo stesso stato come prima */

if (ifr == 0) {
  old_fr = TRUE - old_fr;
  new_fr = TRUE - new_fr;
} /* if (ifr) */

summa = 0;
for (iele = 0; iele < (short) kwssta; iele++) {
  if (tr_active[iele][new_fr]) {
    if (da_dove[iele][new_fr] == 0)
      summa += tr_score[iele][new_fr];
    else
      summa += (tr_score[iele][new_fr]
                - best_score[da_dove[iele][new_fr] - 1]);
  } /* if(tr_active) */
}                       /* for(iele) */

ikey = 0;
for (imodel = key[ikey]; imodel < sum_key; imodel+= key[++ikey]) {
  spot = vet_iaddr[imodel] + nstat[imodel] - 1;
  if (tr_active[spot][new_fr]) {
    if (da_dove[spot][new_fr] == 0)
      post_prob[imodel][new_fr] = tr_score[spot][new_fr] - summa;
    else
      post_prob[imodel][new_fr] = tr_score[spot][new_fr]
        - best_score[da_dove[spot][new_fr] - 1] - summa;
  }             /* if(tr_active[spot]) */
  else {
    post_prob[imodel][new_fr] = MIN_SCORE;
    da_dove[spot][new_fr] = -1;

  }             /* else(tr_active[spot]) */

  if (da_dove[spot][new_fr] == -1)
    print_score = MIN_SCORE;
  else {
    if (da_dove[spot][new_fr] == 0)
      print_score = tr_score[spot][new_fr];
    else
      print_score = tr_score[spot][new_fr]
        - best_score[da_dove[spot][new_fr] - 1];
  } /* else(da_dove) */
```

```
    /*printf("numero  = %d\tframe = %d\tda_dove = %d\tpp = %le\n",
      (imodel + 1), ifr, da_dove[spot][new_fr],
      post_prob[imodel][new_fr]);*/
    fprintf(fris, "numero  = %d\tframe = %d\tda_dove = %d\tpp = %le\t
            score = %le\n", (imodel + 1), ifr, da_dove[spot][new_fr],
            post_prob[imodel][new_fr], print_score);

}                       /* for(imodel) */
```

Anhang B

Listings des Scoring-Analysierers

B.1 peak_detector

Die Prozedur des Peak-Detektors, der durch einen endlichen Automaten realisiert wird.

```
short peak_detector (double pp_old, double pp_new) {

    switch  (stato) {
    case 0:
      if (pp_new == pp_old)
        return 0;
      if (pp_new > pp_old) {
        stato = 1;
        return 0;
      }
      if (pp_new < pp_old) {
        stato = 3;
        return 0;
      }
    case 1:
      if (pp_new == pp_old) {
        stato = 2;
        return 0;
      }
      if (pp_new > pp_old)
        return 0;
      if (pp_new < pp_old) {
        stato = 3;
        return 1;
      }
    case 2:
```

```
      if (pp_new == pp_old)
        return 0;
      if (pp_new > pp_old) {
        stato = 1;
        return 0;
      }
      if (pp_new < pp_old) {
        stato = 3;
        return 0;
      }
    case 3:
      if (pp_new > pp_old) {
        stato = 1;
        return 0;
      }
      if (pp_new <= pp_old)
        return 0;

    }                              /* switch() */

    return 0;

}                                  /* peak_detector() */
```

B.2 analyzer.c

Der gesamte Analyzer.

```
/* analyzer */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include "analyzer.h"

typedef struct frames {
  long last;
  long first;
  double pp;
  double score;
  short winner;
  short peak;
  short cluster;
} frame_t;

typedef struct clusters {
  long last;
  long first;
```

```
  double pp;
  double score;
  short valid;
  double ppp;
} cluster_t;

typedef struct sequences {
  long last;
  long first;
  short number;
  short checked;
} sequence_t;

char str[] = "STR:";
char lbo[] = "LBO:";
char stringa_vuota[] = "";
char eins[] = ",1";
char zwei[] = ",2";
char drei[] = ",3";
char vier[] = ",4";
char fuenf[] = ",5";
char line_break[] = "\n";
char ausruf[] = "!";
char frage[] = "?";

short stato = 0;

cluster_t cluster_max[MX_KWS][MX_FRA];

short peak_detector (double, double);
void overlapping (short, short, short, short);
void exit_err (char *, short);

void main (void) {

  char    single,
          filtut[MX_LINE],
          filin[MX_LINE],
          filnom[MX_LINE],
          garb[MX_LINE],
          col1[MX_LINE],
          string0[MX_FLT],
          string1[MX_FLT],
          string2[MX_FLT],
          string3[MX_FLT],
          string4[MX_FLT],
          *pgarb,
          *pstring0,
          *pstring1,
          *pstring2,
          *pstring3,
          *pstring4;
```

```
line    vocab[MX_KWS],
        sequenza[MX_NUM_NUM],
        help_str[MIN_VECT_LENGTH];

short   i,
        ifr,
        ifr2,
        imod,
        jmod,
        igarb,
        icluster,
        jcluster,
        counter,
        nframes,
        ngarb,
        num_files,
        num_num,
        num_orig,
        ifiles,
        ilines,
        frame_max_mod,
        peaks_with_start_at[MX_FRA][MX_KWS][MX_CLU],
        num_cluster[MX_KWS],
        non_vuoto_cluster,
        nkws,
        nstat[MX_KWS],
        old_peak,
        memory,
        act_cluster,
        valid[MX_KWS][MX_CLU],
        inter,
        n_best_min,
        garb_att,
        str_valid,
        dummy,
        key[MX_KWS],
        sum_key,
        ikey,
        iorig;

long    ilast,
        fin_end_last[MX_KWS][MX_CLU][CRIT_ANZ],
        start_frame,
        primo_frame,
        ultimo_frame;

double  frame_max_value,
        sum,
        soglia[MX_KWS],
        soglia_ppp[MX_KWS],
        cmp[MX_KWS][MX_CLU], /* cmp = cluster_max_pp,
                                il maggiore valore pp di un cluster */
        mom_end_val,
```

```
          best_end_val,
          peak_max[MX_KWS],
          garb_peso,
          rdummy;

  frame_t voc[MX_FRA][MX_KWS + MX_GARB];
  sequence_t orig[MX_NUM_NUM];

  FILE * finp,
       * fseg;

  pgarb = garb;
  pstring0 = string0;
  pstring1 = string1;
  pstring2 = string2;
  pstring3 = string3;
  pstring4 = string4;

  fgets (pstring1, MX_LINE, stdin);
  sscanf (pstring1, "%d", &nkws);
/*  printf ("nkws = %d\n", nkws); */

  sum_key = 0;
  for (i = 0; i < nkws; i++) {
    fgets (pstring1, MX_LINE, stdin);
    sscanf (pstring1, "%d", &dummy);
    key[i] = dummy;
    sum_key += dummy;
  } /* for (i) */
  sum_key++;
  key[nkws] = 1; /* cosi' imod dopo salta la soglia */

  fgets (pstring1, MX_LINE, stdin);
  sscanf (pstring1, "%d", &ngarb);

  fgets (pstring1, MX_LINE, stdin);
  sscanf (pstring1, "%d", &n_best_min);
/*  printf ("n_best_min = %d\n", n_best_min); */

  fgets (pstring1, MX_LINE, stdin);
  sscanf (pstring1, "%d", &garb_att);
/*  printf ("garb_mod_usato = %d\n", garb_att); */
  garb_att = garb_att - n_best_min + MX_KWS;

  fscanf (stdin, "%s\n", pstring1);
  garb_peso = atof (string1);
/*  printf ("garb_peso = %e\n", garb_peso); */

  for (imod = 0; imod < MX_KWS; imod++) {
    fgets (pstring1, MX_LINE, stdin);
    sscanf (pstring1, "%s", vocab[imod]);
    fgets (pstring1, MX_LINE, stdin);
    sscanf (pstring1, "%d", &inter);
```

```
  nstat[imod] = inter;
  fscanf (stdin, "%s\n", pstring1);
  rdummy = atof (string1);
  soglia[imod] = rdummy;
  fscanf (stdin, "%s\n", pstring1);
  rdummy = atof (string1);
  soglia[imod] *= rdummy;
  fscanf (stdin, "%s\n", pstring1);
  rdummy = atof (string1);
  soglia_ppp[imod] = rdummy;
  fscanf (stdin, "%s\n", pstring1);
  rdummy = atof (string1);
  soglia_ppp[imod] *= rdummy;

} /* for(imod) */

fgets (pstring1, MX_LINE, stdin);
sscanf (pstring1, "%s", filnom);

fgets (pstring1, MX_LINE, stdin);
sscanf (pstring1, "%d", &num_files);

if ((fseg = fopen (filnom, "r")) == NULL)
  exit_err (filnom, DATA_ERR);

/* spostamento nel file di segmentazione fino alla prima stringa 'STR:' */

nochmal1: do {
  fgets (pstring1, MX_LINE, fseg);
  strcpy (help_str[MIN_VECT_LENGTH - 1], string1);
  sscanf (pstring1, "%s", garb);
}
while (((strcmp (garb, str)) != 0));

if ((strpbrk(help_str, ausruf) != NULL) || (strpbrk(help_str, frage) != NULL))
  goto nochmal1;


/* trattamento dei file con i risultati di viterbi */

for (ifiles = 0; ifiles < num_files; ifiles++) {


  /* carico della sequenza originale */

  num_num = 0;
  do {
    fgets (pstring1, MX_LINE, fseg);
    strcpy (help_str[MIN_VECT_LENGTH - 1], string1);
    sscanf (pstring1, "%s%s%s", col1, garb, garb);
    if ((!strcmp (col1, lbo)))
      strcpy(sequenza[num_num++], help_str);
  }
```

```
    while (((strcmp (col1, str)) != 0));

    num_orig = 0;

    for (i = 0; i < num_num; i++) {
      sscanf (sequenza[i], "%s%s%s", pgarb, pstring1, pstring2);
      single = pstring2[strlen(pstring2)-1];
      if (isdigit(single)) {
        orig[num_orig].number = (short) (single - ASCII_OFFSET);
        strcpy (pstring3, stringa_vuota);
        strncpy (pstring3, pstring1, (strlen(pstring1)-1));
        orig[num_orig].first = atol(pstring3);
        strcpy (pstring3, stringa_vuota);
        strncpy (pstring3, pstring2, (strlen(pstring2) - CODA));
        orig[num_orig].last = atol(pstring3);
        orig[num_orig++].checked = -1;
      } /* if(isdigit) */

    } /* for(i) */


    fgets (pstring1, MX_LINE, stdin);
    sscanf (pstring1, "%s", filtut);

/*    printf("\n%s\n", filtut); */

    fgets (pstring1, MX_LINE, stdin);
    sscanf (pstring1, "%d", &primo_frame);

    fgets (pstring1, MX_LINE, stdin);
    sscanf (pstring1, "%d", &ultimo_frame);

    nframes = (short) (ultimo_frame - primo_frame + 1);

    if ((finp = fopen (filtut, "r")) == NULL)
      exit_err (filtut, DATA_ERR);

    for (ifr = 0; ifr < nframes; ifr++) {
      frame_max_value = 0.0;
      frame_max_mod = -1;


      ikey = 0;
      for (imod = key[ikey]; imod < sum_key; imod+= key[++ikey]) {
        fscanf (finp, "%s%s%s", pstring0, pstring1, pstring2);

        voc[ifr][imod].last = ifr;
        voc[ifr][imod].first = atoi (string0);
        voc[ifr][imod].pp = atof (string1);
        voc[ifr][imod].score = atof (string2);
        voc[ifr][imod].winner = FALSE;
        voc[ifr][imod].peak = FALSE;
        voc[ifr][imod].cluster = -1;
```

```
    if (voc[ifr][imod].pp > frame_max_value) {
      frame_max_value = voc[ifr][imod].pp;
      frame_max_mod = imod;
    } /* if(voc) */

  } /* for(imod) */

  for (imod = MX_KWS; imod < (MX_KWS + ngarb); imod++) {
    fscanf (finp, "%s", pstring0);

    voc[ifr][imod].last = ifr;
    voc[ifr][imod].first = ifr;
    voc[ifr][imod].pp = 0.0;
    voc[ifr][imod].score = atof (string0);
    voc[ifr][imod].pp *= garb_peso;
    voc[ifr][imod].score *= garb_peso;
    voc[ifr][imod].winner = FALSE;
    voc[ifr][imod].peak = FALSE;
    voc[ifr][imod].cluster = -1;

    if (voc[ifr][imod].pp > frame_max_value) {
      frame_max_value = voc[ifr][imod].pp;
      frame_max_mod = imod;
    } /* if(voc) */

  } /* for(imod) */

  /* voc[ifr][frame_max_mod].winner = TRUE; */

} /* for(ifr) */

fclose (finp);


/* peak-detection */

ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  for (ifr = 1; ifr < nframes; ifr++) {
    if ((peak_detector(voc[ifr-1][imod].pp, voc[ifr][imod].pp))
        /* && (voc[ifr-1][imod].pp > soglia[imod])*/ )
      voc[ifr-1][imod].peak = TRUE;
  } /* for(ifr) */
  if (stato == 1) voc[ifr][imod].peak = TRUE;
    /* peak al margine a destra (rechtes randmaximum) */
} /* for(imod) */


/* stabilita' del frame iniziale */
ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  for (ifr = 0; ifr < nframes; ifr++) {
```

```
    if (voc[ifr][imod].peak) {
      if (voc[ifr][imod].first != voc[ifr + 1][imod].first) {
        voc[ifr][imod].peak = FALSE;
      } /* if (vod[][].first) */
    } /* if (voc[][].peak) */
  } /* for (ifr) */
} /* for (imod) */


/* clustering dei peaks */
ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  num_cluster[imod] = 0;
  old_peak = -2; /* old_peak = -2 significa che siamo all'inizio dell esame */
  for (ifr = 0; ifr < nframes; ifr++) {
    if (voc[ifr][imod].peak) {
      if (old_peak == -2) {
        cluster_max[imod][num_cluster[imod]].first = voc[ifr][imod].first;
        cluster_max[imod][num_cluster[imod]].last = voc[ifr][imod].last;
        cluster_max[imod][num_cluster[imod]].pp = voc[ifr][imod].pp;
        cluster_max[imod][num_cluster[imod]].score = voc[ifr][imod].score;
        cluster_max[imod][num_cluster[imod]].valid = TRUE;
        voc[ifr][imod].cluster = num_cluster[imod];
        num_cluster[imod]++;
        old_peak = ifr;
        goto dopo;
      } /* if (old_peak) */
      if (voc[ifr][imod].first != voc[old_peak][imod].first) {
        cluster_max[imod][num_cluster[imod]].first = voc[ifr][imod].first;
        cluster_max[imod][num_cluster[imod]].last = voc[ifr][imod].last;
        cluster_max[imod][num_cluster[imod]].pp = voc[ifr][imod].pp;
        cluster_max[imod][num_cluster[imod]].score = voc[ifr][imod].score;
        cluster_max[imod][num_cluster[imod]].valid = TRUE;
        voc[ifr][imod].cluster = num_cluster[imod];
        num_cluster[imod]++;
        old_peak = ifr;
        goto dopo;
      } /* if (voc != voc) */
      else {
        if (voc[ifr][imod].pp >= voc[old_peak][imod].pp) {
          /* se i valori di pp sono  uguali vince il precorso piu' lungo */
          cluster_max[imod][num_cluster[imod] - 1].last = voc[ifr][imod].last;
          cluster_max[imod][num_cluster[imod] - 1].pp = voc[ifr][imod].pp;
          old_peak = ifr;
        } /* if (voc > voc) */
        voc[ifr][imod].cluster = num_cluster[imod] - 1;
        goto dopo;
      } /* else (voc != voc) */
    } /* if (voc.peak) */
  dopo: if (num_cluster[imod] > MX_CLU) exit_err(stringa_vuota, CLUSTER_ERR);
  } /* for(ifr) */
} /* for (imod) */
```

```
    /* calcolazione di ppp */
    ikey = 0;
    for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
      for (icluster = 0; icluster < num_cluster[imod]; icluster++) {
        best_end_val = MIN_SCORE;
        start_frame = cluster_max[imod][icluster].first;
        for (ilast = start_frame; ilast <= cluster_max[imod][icluster].last; ilast++) {
          mom_end_val = fabs(voc[ilast][imod].score -
              voc[start_frame][imod].score - (voc[ilast][garb_att].score -
              voc[start_frame][garb_att].score));
          if (mom_end_val > best_end_val) {
            best_end_val = mom_end_val;
          } /* if(mom >= best) */
        } /* for(ilast) */
      cluster_max[imod][icluster].ppp = best_end_val;
      } /* for(icluster) */
    } /* for(imod) */

    /* stampa di controllo */
/*    printf ("stampa di controllo dopo clustering e calcolazione ppp-soglia\n"); */
    ikey = 0;
    for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
      for (icluster = 0; icluster < num_cluster[imod]; icluster++) {
        if (cluster_max[imod][icluster].valid)
/*        printf ("numero: %d\tfirst: %ld\tlast: %ld\tpp: %e\tppp: %e\n",
            (imod + 1), (cluster_max[imod][icluster].first + primo_frame),
            (cluster_max[imod][icluster].last + primo_frame),
            cluster_max[imod][icluster].pp, cluster_max[imod][icluster].ppp); */
      } /* for(icluster) */
    } /* for(imod) */


    /* valutazione nella fase del training */

    ikey = 0;
    for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
      for (icluster = 0; icluster < num_cluster[imod]; icluster++) {
        if (cluster_max[imod][icluster].valid) {
          for (iorig = 0; iorig < num_orig; iorig++) {
            if ((imod == (orig[iorig].number -1)) &&
                (labs(cluster_max[imod][icluster].last + primo_frame -
                orig[iorig].last) <= PERMITTED_OVERLAP) &&
                (labs(cluster_max[imod][icluster].first + primo_frame -
                orig[iorig].first) <= PERMITTED_OVERLAP)) {
              if (orig[iorig].checked == -1) {
                orig[iorig].checked = icluster;
              } /* if (orig[]) */
              else { /* overlap-check, il piu' alto valore viene scelto */
                if (cluster_max[imod][icluster].pp >
                    cluster_max[imod][orig[iorig].checked].pp) {
                  orig[iorig].checked = icluster;
                } /* if (cluster_max) */
```

```
          } /* else (orig[]) */
        } /* if (!iorig) */
      } /* for (iorig) */
    } /* if (cluster_max) */
  } /* for (icluster) */
} /* for (imod) */

for (iorig = 0; iorig < num_orig; iorig++) {
  if (orig[iorig].checked != -1) {
    imod = orig[iorig].number - 1;
    icluster = orig[iorig].checked;
    printf ("mod: %d\tpp: %le\tppp: %le\n", imod,
        cluster_max[imod][icluster].pp, cluster_max[imod][icluster].ppp);
  } /* if (orig[]) */
} /* for (iorig) */

goto nochmal2;


/* applicazione ppp-soglia */

ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  for (icluster = 0; icluster < num_cluster[imod]; icluster ++) {
    if (cluster_max[imod][icluster].ppp < soglia_ppp[imod]) {
      cluster_max[imod][icluster].valid = FALSE;
    } /* if(cluster_max) */
  } /* for (icluster ) */
} /* for (imod) */

/* stampa di controllo */
printf ("stampa di controllo dopo applicazione ppp-soglia\n");
ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  for (icluster = 0; icluster < num_cluster[imod]; icluster++) {
    if (cluster_max[imod][icluster].valid)
      printf ("numero: %d\tfirst: %ld\tlast: %ld\tpp: %e\tppp: %e\n",
             (imod + 1), (cluster_max[imod][icluster].first + primo_frame),
             (cluster_max[imod][icluster].last + primo_frame),
             cluster_max[imod][icluster].pp,
             cluster_max[imod][icluster].ppp);
  } /* for(icluster) */
} /* for(imod) */


/* eliminazzione delle intersecazioni fra i cluster di un modello */
ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  for (icluster = 0; icluster < num_cluster[imod]; icluster++) {
    for (jcluster = (icluster + 1); jcluster < num_cluster[imod];
         jcluster++) {
      overlapping(imod, imod, icluster, jcluster);
    } /* for(jcluster) */
```

```
  } /* for(icluster) */
} /* for(imod) */

/* stampa di controllo */
printf ("stampa di controllo dopo intersec-elim-1\n");
ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  for (icluster = 0; icluster < num_cluster[imod]; icluster++) {
    if (cluster_max[imod][icluster].valid)
      printf ("numero: %d\tfirst: %ld\tlast: %ld\tpp: %e\n", (imod + 1),
              (cluster_max[imod][icluster].first + primo_frame),
              (cluster_max[imod][icluster].last + primo_frame),
              cluster_max[imod][icluster].pp);
  } /* for(icluster) */
} /* for(imod) */


/* eliminazzione delle intersecazioni fra i cluster dei diversi modelli */
ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  for (icluster = 0; icluster < num_cluster[imod]; icluster++) {
    for (jmod = (imod + 1); jmod < nkws; jmod++) {
      for (jcluster = 0; jcluster < num_cluster[jmod]; jcluster++) {
        overlapping(imod, jmod, icluster, jcluster);
      } /* for(jcluster) */
    } /* for(jmod) */
  } /* for(icluster) */
} /* for(imod) */

/* stampa di controllo */
printf ("stampa di controllo dopo intersec-elim-2\n");
ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  for (icluster = 0; icluster < num_cluster[imod]; icluster++) {
    if (cluster_max[imod][icluster].valid)
      printf ("numero: %d\tfirst: %ld\tlast: %ld\tpp: %e\n", (imod + 1),
              (cluster_max[imod][icluster].first + primo_frame),
              (cluster_max[imod][icluster].last + primo_frame),
              cluster_max[imod][icluster].pp);
  } /* for(icluster) */
} /* for(imod) */


/* risultato finale */
printf ("risultato finale\n");
ikey = 0;
for (imod = key[ikey]; imod < sum_key; imod += key[++ikey]) {
  for (icluster = 0; icluster < num_cluster[imod]; icluster++)  {
    if (cluster_max[imod][icluster].pp < soglia[imod])
        cluster_max[imod][icluster].valid = FALSE;
    if (cluster_max[imod][icluster].valid)
      printf ("numero: %d\tfirst: %ld\tlast: %ld\tpp: %e\tscore: %e\n",
              (imod + 1), (cluster_max[imod][icluster].first + primo_frame),
```

```
                cluster_max[imod][icluster].last + primo_frame),
                cluster_max[imod][icluster].pp,
                cluster_max[imod][icluster].score);
    } /* for(icluster) */
  } /* for(imod) */


  /* stampa della sequenza originale */

  printf("sequenza:\n");
  for (i = 0; i < num_num; i++)
    printf ("%s", sequenza[i]);


  /* spostamento nel file di segmentazione fino alla prossima stringa 'STR:' */

nochmal2: if ((strpbrk(help_str, ausruf) != NULL) ||
              (strpbrk(help_str, frage) != NULL)) {
  do {
    fgets (pstring1, MX_LINE, fseg);
    strcpy (help_str[MIN_VECT_LENGTH - 1], string1);
    sscanf (pstring1, "%s", garb);
  }
  while (((strcmp (garb, str)) != 0));
} /* if (strpbrk) */

  if ((strpbrk(help_str, ausruf) != NULL) || (strpbrk(help_str, frage) != NULL))
    goto nochmal2;


} /* for(ifiles) */

close(fseg);

} /* main() */

void overlapping (short imod, short jmod, short icluster, short jcluster) {

  if ((cluster_max[jmod][jcluster].first >
       (cluster_max[imod][icluster].first + PERMITTED_OVERLAP)) &&
       (cluster_max[jmod][jcluster].first <= cluster_max[imod][icluster].last)
       && cluster_max[imod][icluster].valid && cluster_max[jmod][jcluster].valid)
    if (cluster_max[jmod][jcluster].pp > cluster_max[imod][icluster].pp)
      cluster_max[imod][icluster].valid = FALSE;
    else
      cluster_max[jmod][jcluster].valid = FALSE;

  else
    if ((cluster_max[imod][icluster].first >
         (cluster_max[jmod][jcluster].first + PERMITTED_OVERLAP)) &&
         (cluster_max[imod][icluster].first <= cluster_max[jmod][jcluster].last)
         && cluster_max[imod][icluster].valid
         && cluster_max[jmod][jcluster].valid)
```

```
      if (cluster_max[jmod][jcluster].pp > cluster_max[imod][icluster].pp)
        cluster_max[imod][icluster].valid = FALSE;
      else
        cluster_max[jmod][jcluster].valid = FALSE;

  return;

} /* overlapping() */


void exit_err (char *s, short i) {

  if (i == DATA_ERR) {
    printf ("\nImpossibile aprire file %s\n", s);
    exit (1);
  } /* if(i) */
  else {
    if (i == CLUSTER_ERR) {
      printf ("\n Troppi cluster\n");
      exit (1);
    } /* else-if(i) */
  } /* else(i) */
}                                /* exit_err */
```

B.3 analyzer.h

Konstanten und Typen des Analyzers.

```
/* analyzer.h
   costanti e tipi per il file analyzer.c */

#define TRUE 1
#define FALSE 0
#define ORIG 0

#define CRIT_1 0
#define CRIT_2 1
#define CRIT_ANZ 2
#define DATA_ERR 0
#define CLUSTER_ERR 1

#define MIN_SCORE -1000.0
#define MIN_VECT_LENGTH 1

/* min e max define */
#define min(a, b) (((a) <= (b)) ? (a) : (b))
#define max(a, b) (((a) >= (b)) ? (a) : (b))

/* limiti */

#define MX_FRA  800 /* max # di frame in una sequenza */
#define MX_LINE 81 /* max lunghezza di una linea */
#define MX_FLT 14 /* max lunghezza di un float */
#define MX_GARB 17 /* max numero dei modelli garbage */
#define MX_KWS 10 /* max numero delle parole chiave */
#define MX_CLU 50 /* max numero dei cluster */
#define MX_NUM_NUM 20 /* max numero di numeri in una frase */
#define PERMITTED_OVERLAP 15 /* max numero della intersecazione fra due parole chiave */
#define CODA 5 /* 3090,,,,5 --> lunghezza di ,,,,5 (file di segm) */
#define ASCII_OFFSET 48 /* 55 == 7 */
#define STAB_MIN 10 /* stabilita' minimale per un frame finale */

/* tipi */
typedef char line[80];
typedef char word[44];
```

Bibliographie

[1] L. R. BAHL, P. V. DE SOUZA & L. R. MERCER, **"Maximum Mutual Information Estimation Of Hidden Markov Model Parameters For Speech Recognition."**, *Proc. Int. Conf. on Acoust. Speech and Sig. Processing*, IV 1986, pp. 49–52.

[2] L. R. BAHL, P. V. DE SOUZA & L. R. MERCER, **"A New Algorithm For The Estimation Of Hidden Markov Model Parameters"**, *Proc. Int. Conf. on Acoust. Speech and Sig. Processing*, IV 1988, pp. 493–496.

[3] J. BAKER, **Private Communication** J. BAKER – R. C. ROSE.

[4] J. R. BELLEGARDA & D. NAHAMOO, **"Tied Mixture Continuous Parameter Modeling For Speech Recognition"**, *IEEE Trans. on Acoust. Speech and Sig. Processing*, ASSP-38(12), 1990, pp. 2033–2045.

[5] H. BOULARD, B. D'HOORE & J.-M. BOITE, **"Optimizing Recognition And Rejection Performance In Wordspotting Systems"**, *IEEE Conf. on ASSP*, Adilaide, 1994, Vol. I, pp. 373–376.

[6] J.-M. BOITE, H. BOULARD, B. D'HOORE & M. HAESEN, **"A New Approach Towards Keyword Spotting"**, *Eurospeech*, Berlin, 1993, pp. 1273–1276.

[7] J. S. BRIDLE, **"An Efficient Elastic-Template Method For Detecting Given Words In Running Speech"**, *Brit. Accoust. Soc. Meeting*, pp. 1–4, IV 1973.

[8] P. F. BROWN, J. C. SPOHRER, P. H. HOCHSCHILD & J. K. BAKER, **"Partial Traceback And Dynamic Programming"**, *Proc. of the Int. Conf. on ASSP*, Paris, France, V 1982, pp. 1629–1632.

[9] F. CANAVESIO, L. FISSORE, M. OREGLIA & P. RUSCITTI, **"HMM Modeling In The Public Telephone Network Environment: Experiments And Results"**, *Eurospeech*, Genova, 1991, pp. 731–734.

[10] M. CRAVERO, L. FISSORE, R. PIERACCINI, F. RAINERI, **"Il riconoscimento della voce"**, *Elettronica e Telecomunicazioni*, N.3, 1985.

[11] R. W. CHRISTIANSEN & C. K. RUSHFORTH, **"Detecting And Locating Key Word In Continuous Speech Using Linear Predective Coding"**, *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, no. 5, pp. 361–367, X 1977.

[12] A. L. HIGGINS & R. E. WOHLFORD, **"Keyword Recognition Using Template Concatenation"**, *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Tampa (FL), III 1985, pp. 1233–1236 .

[13] C. H. LEE & L. R. RABINER, **"A Network-Based Frame Synchronous Level Building Algorithm For Connected Word Recognition"**, *Conf. Rec. IEEE Int. Conf. Acous. Speech and Sig. Precessing*, Vol. I. pp. 410–413, New York, NY, IV 1988.

[14] H. LEPRIEUR, **"Rapport Bibliographique Et Experiences Comparatives Sur La Detection De Mot-Cles"**, *Centre National d'Etudes des Telecommunications, Centre Lannion A, Document de Travail 1419/LAA/TSS/RCP*, XII 1992.

[15] C. S. MYERS, L. R. RABINER & A. E. ROSENBERG, **"An Investigation Of The Use Of Dynamic Time Warping For Word Spotting And Connected Word Recognition"**, *Proc. Conf. Acoust., Speech, Signal Processing*, Denver (CO), IV 1980, pp. 173–177.

[16] H.-H. NAGEL ET AL., **"Skriptum Zur Vorlesung Kognitive Systeme"**, *Institut für Algorithmen und Kognitive Systeme, Fakultät für Informatik der Universität Karlsruhe (TH)*, VI 1992.

[17] L. T. NILES, L. D. WILCOX & M. A. BUSH, **"Error-Correcting Training For Phoneme Spotting"**, *Proc. IEEE Int. Conf. on ASSP*, 1992, Vol. I, pp. 425–428.

[18] R. PIERACCINI, **"Macchine che comprendono la voce"** , *Le scienze*, edizione italiana di *Scientific American*, V 1985.

[19] P. PRIOTTI, **"Tecniche Stocastiche Ad Elevata Efficienza Per Il Riconoscimento Vocale"**, *Tesi di Laurea*, Politecnico di Torino, II 1994.

[20] L. R. RABINER, " **A Tutorial On Hidden Markov Models And Selected Applications In Speech Recognition**", *Proc. IEEE*, Vol. 77, No. 2, II 1989, pp. 257–286.

[21] L. R. RABINER, J. G. WILPON & B. H. JUANG, " **A Segmental K-means Training Procedure For Connected Recognition** ", *AT&T Technical Journal*, V 1986.

[22] J. R. ROHLICEK, W. RUSSEL, S. ROUKOS & HERB GISH, **"Continuous Hidden Markov Modeling For Speaker-Independent Word Spotting"**, *Proc. IEEE Int. Conf. on ASSP*, V 1989, Glasgow, Vol. I, pp 627–630.

[23] R. C. ROSE & D. B. PAUL, **"A Hidden Markov Model Based Keyword Recognition System"**, *Proc. IEEE Int. Conf. ASSP*, IV 1990, Albukerque, pp. 129–132.

[24] R. C. ROSE, **"Discriminant Wordspotting Techniques For Rejecting Non-Vocabulary Utterances In Unconstrained Speech"**, *Proc. IEEE Int. Conf. on ASSP*, 1992, San Francisco, Vol. II, pp. 105–108.

[25] Y. TOHKURA, *"A Weighted Cepstral Distance Measure For Speech Recognition"*, *IEEE Trans. Acoust., Speech and Signal Processing*, ASSP-35(1), pp. 1414–1422, X 1987.

[26] C. VAIR, **"Tecniche Di Riconoscimento Per Vocabolari Di Grandi Dimensioni"**, *Tesi di Laurea*, Politecnico di Torino, VII 1994.

[27] L. C. VROOMEN & YVES NORMANDIN, **"Robust Speaker-Indipendent Hidden Markov Model Based Word Spotter"**, *NATO ASI Series*, Vol. F75, Speech Recognition and Understanding, Recent Advances, Edited by P. LAFACE & R. DE MORI, Springer-Verlag Berlin Heidelberg 1992, pp. 95–100.

[28] A. WAIBEL ET AL., **"Folienkopien Zur Vorlesung Kognitive Systeme"**, *Universität Karlsruhe*, VIII 1993.

[29] L. D. WILCOX & M. A. BUSH, **"HMM-Based Wordspotting For Voice Editing And Indexing"**, *Eurospeech*, 1991, Vol. II, pp. 25–28.

[30] J. G. WILPON, C. H. LEE & L. R. RABINER, **"Application Of Hidden Markov Models For Recognition Of A Limited Set Of Words In Unconstrained Speech"**, *Proc. ICASSP*, Glasgow, V 1989, pp. 254–257.

[31] J. G. WILPON, L. R. RABINER, C. H. LEE & E. R. GOLDMAN, **"Automatic Recognition Of Keywords In Unconstrained Speech Using Hidden Markov Models"**, *IEEE Trans. on ASSP*, Vol. 38, No. II, XI 1990, pp. 1870–1878.

[32] T. ZEPPENFELD & A. WAIBEL, **"A Hybrid Neural Network, Dynamic Word Spotter"**, *Proc. IEEE Int. Conf. on ASSP*, 1992, San Francisco, vol. II, pp., 77–80.