DEUTSCH-FRANZÖSISCHES INSTITUT
FÜR AUTOMATION UND ROBOTIK
TEILINSTITUT KARLSRUHE

DIPLOMARBEIT

für

cand.el. Martin Maier

Thema:

Dimensionalitätsreduktion von Sprachsignalen

mit statistischen und neuronalen Methoden

Ausgabe: 10.07.1993

Abgabe: 10.01.1994

Prof. Dr.-Ing. K. Kroschel

Ich erkläre hiermit an Eides Statt, daß ich die vorliegende Diplomarbeit selbstständig und ohne unzulässige fremde Hilfe angefertigt habe. Die verwendeten Literaturquellen sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, den 8.1.1993

Martin Maier
Durlacher Str.13
76229 Karlsruhe

Mein Dank gilt den Herren Prof. Kroschel und Prof. Waibel, die mir diese Diplomarbeit ermöglicht haben.

Ganz besonders herzlichen Dank möchte ich Ivica Rogina aussprechen, den ich als Betreuer für weitere Diplomarbeiten wärmstens empfehlen kann.

Unterstützung fand ich auch bei den Mitarbeitern des Instituts für Logik, Komplexität und Deduktionssysteme, besonders bei Thomas und Michael mit denen ich so manche Diskussion geführt habe. Thanks to some other fellows at Carnegie Mellon, Pittsburgh as well.

Inhaltsverzeichnis

Nach einer allgemeinen Einführung in die Spracherkennung und einer Motivation zur Dimensionalitätsreduktion im ersten Kapitel beschreibt Kapitel 2 eine konkrete Anwendung, den JANUS-Spracherkenner. In Kapitel 3 werden die theoretischen Grundlagen für eine Dimensionalitätsreduktion durch statistische Methoden erläutert. In den anschließenden Kapiteln 4 und 5 werden die praktische Realisierung und die erzielten Ergebnisse vorgestellt. Kapitel 6 enthält einen neuen diskriminativen Ansatz, der mit Hilfe eines neuronalen Netzes arbeitet. Das letzte Kapitel gibt noch einmal einen zusammenfassenden Überblick der erzielten Ergebnisse und enthält Vorschläge für weiterführende Arbeiten..

Formelzeichen und Abkürzungen

Formelzeichen:

allgemeine Funktionen

$E\{...\}$	Erwartungswert
$E\{...\|...\}$	bedingter Erwartungswert
$P(...)$	Wahrscheinlichkeit
$P(...\|...)$	bedingte Wahrscheinlichkeit
$f_{...}(...)$	Wahrscheinlichkeitsdichtefunktion
$f_{...\|..}(...\|...)$	bedingte Wahrscheinlichkeitsdichtefunktion
x^T, A^T	transponierter Vektor x, transponierte Matrix A
$\|x\|$	Betrag des Vektors x
Δm	Vektordifferenz
$\|A\|$ oder $\det(A)$	Determinante einer Matrix A
A^{-1}	Inverse der Matrix A
argmin(..)	Index des Minimums
$f'(x)$ oder $\frac{df(x)}{dx}$	Ableitung der Funktion

Signale, Vektoren und Räume

s(t)	Sprachsignal	Zeitpunkt	t
v(k)	Datenvektor	Rahmen (*frame*)	k
x oder x(k)	Daten- oder Mustervektor der Dimension n		
y oder y(k)	Mermalsvektor der Dimension m		
R^n	Datenraum mit der Dimension n		
R^m	Merkmalsraum mit der Dimension m		

n, m	Dimensionen ($m \leq n$)

Hidden-Markov-Modell und Spracherkenner

s	HMM-Zustand
c	Codebuchindex
α	Gewichtungsfaktor, Formel (2.7)
h	Skalierungsfaktor, Formel (2.6)

Scattermatrizen

S	Scattermatrix allgemein, entweder T, W oder B
S_n	Scattermatrix im Datenraum R^n
S_m	Scattermatrix im Merkmalsraum R^m
T	Total-Scattermatrix (Kovarianzmatrix), Formel (3.1)
W	Within-Class-Scattermatrix, Formel (3.2)
B	Between-Class-Scattermatrix, Formel (3.3)
A	Transformationsmatrix, Formel (3.4) und (3.10)

Gütekriterium und Lösung

J(m)	Gütekriterium im Raum R^m, Formel (3.6)-(3.8)
λ_i	Eigenwert
Φ	Eigenvektormatrix
ϕ_i	Eigenvektor, die i-te Spalte der Eigenvektormatrix
d_r	Datenreduktions-Faktor, Formel (3.5)

Neuronales Netz

u_i	Einheit i	(englisch: *unit*)
a_i	Aktivierung, S.52	
w_{ij}	Gewicht von Einheit u_j zur Einheit u_i	(englisch: *weight*)
$o_i = f(a_i)$	Ausgangswert, bzw. Ausgangsfunktion der Einheit u_i	(englisch: *output*)
x oder x(k)	Eingangssignal des Netzes	
y oder y(k)	Ausgangssignal des Netzes	

t oder t(k)	Lernsignal	(englisch: *target*)
t_i	Lernsignal für Muster y aus der Klasse ω_i	
ω_i	Klasse	
m_i	Mittelwert einer Klasse	
Δm_{ij}	Differenzvektor = m_i - m_j	
$b(\lvert\Delta m_{ij}\rvert)$	Längenfunktion, Formel (6.20)	
α, β	Gewichtungsfaktoren, S.59	
rad	Radialkomponente, Formel (6.21)	
t*	modifiziertes Lernsignal, Formel (6.23)	
w(rad)	Wannenfunktion, Formel (6.22)	

Abkürzungen:

HMM	Hidden-Markov-Modell	
MSC	Melscale-Koeffizienten	(engl: *melscale coefficients*)
HAT	Hauptachsentransformation	
LDA	Lineare Diskriminanzanalyse	
DAN	Diskriminanzanalyse-Netzwerk	
MLP	Multi-Layer-Perceptron	
WA	Erkennungsrate	(engl: *word accuracy*)
WER	Fehlerrate	(engl: *word error rate*)

1. Einführung

1.1. Spracherkennung mit JANUS

Computer haben unsere Welt erobert. Diese unermüdlichen Rechenknechte nehmen dem Menschen monotone Arbeit ab und eröffnen ihm ganz neue Möglichkeiten. Auf Grund ihrer Funktionsweise sind sie genau für die Aufgaben geeignet, bei denen sich Menschen schwer tun. Dieser Umstand führt aber auch zu einem Kommunikationsproblem (siehe Bild 1.1). Während Menschen sich vorwiegend durch Gestik und Sprache auf nicht immer ganz eindeutige Art verständigen, brauchen Maschinen definierte, meist binäre Werte.

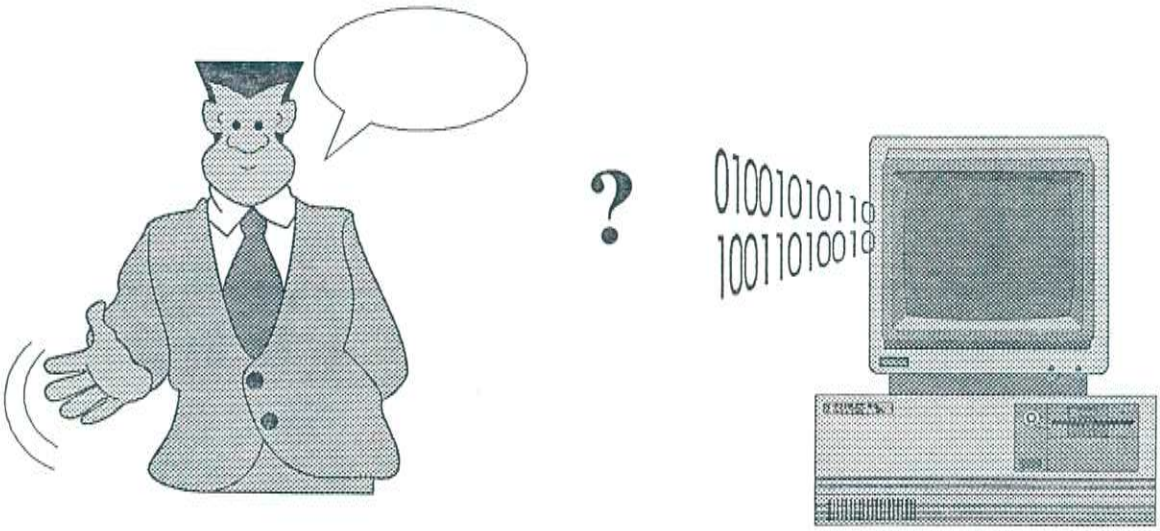


Bild 1.1: Mensch, Maschine und ihr Kommunikationsproblem

In den frühen Tagen der Computer paßte man sich ganz der Maschine an und fütterte sie mit Zahlenkolonnen und Lochstreifen. Angesichts der gesteigerten Leistungsfähigkeit der Maschinen geht man immer mehr dazu über, Kommunikationsmittel zu benutzen, die dem Menschen entgegen kommen oder ihm sogar vertraut sind. Die gesprochene Sprache steht dabei als Informationsüberträger ganz oben an. Was liegt also näher, als dem Computer das Hören und Sprechen beizubringen.

Das Übersetzungssystem JANUS[1], welches innerhalb des C-STAR Projekts[2] verwendet wird, geht dabei noch einen Schritt weiter. Der Computer wird benutzt, um die Kommunikation zweier Menschen zu ermöglichen die verschiedene Sprachen benutzen.

Ein Sprecher sagt einen ganzen Satz. Wenig später erscheint der Text in seiner und der fremden Sprache auf dem Bildschirm. Gleichzeitig ist er auch für den Gesprächspartner in dessen eigener Sprache über einen Lautsprecher zu hören.

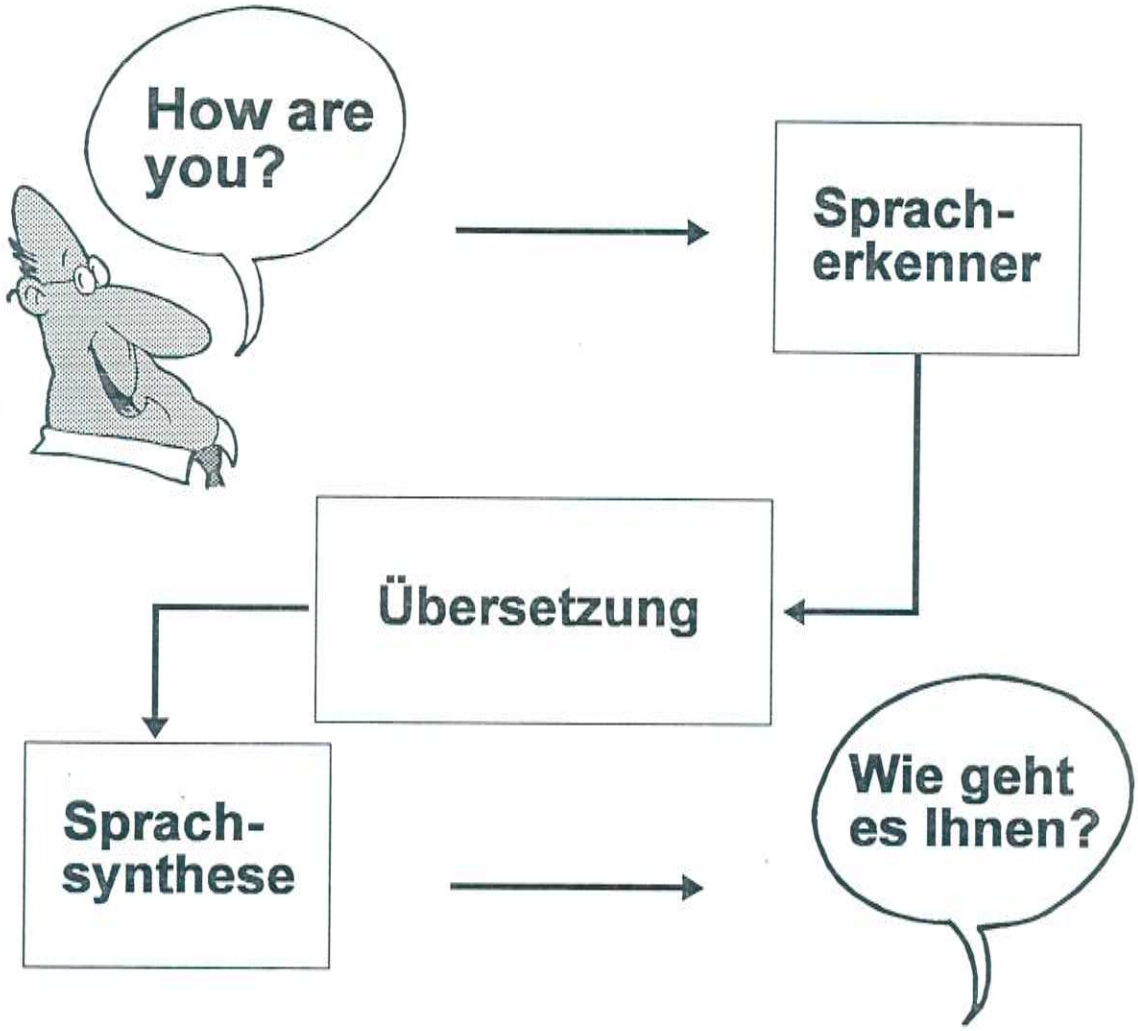


Bild 1.2: Das Janusprojekt ermöglicht die Übersetzung gesprochener Sätze

Es sind also drei Aufgaben zu erfüllen (Bild 1.2). Erstens muß eine Spracherkennung die gesprochenen Laute in geschriebene Worte und Sätze umwandeln. Dann werden

[1]JANUS, nach dem römischen Gott mit zwei Gesichtern.

[2]C-STAR ist eine Kooperation der Universität Karlsruhe mit der Carnegie-Mellon University (Pittsburgh), ATR Interpreting Telephone Laboratories (Osaka) und der SIEMENS AG.

durch ein Übersetzungsprogramm die Sinnzusammenhänge ermittelt und in eine andere Sprache übertragen. Als Drittes werden die Äußerungen in der anderen Sprache mittels eines Sprachsyntheseverfahrens ausgegeben. Diese drei Teilaufgaben ergeben auch getrennt interessante Anwendungen. Im folgenden wird nur noch die Spracherkennung betrachtet. Der JANUS-Spracherkenner arbeitet auf Grund der Aufgabenstellung im Projekt

- mit kontinuierlicher Sprache und
- sprecherunabhängig.

Da Janus ständig weiterentwickelt wird, gibt es diverse Kombinationsmöglichkeiten von Verfahren, die im Spracherkennungsteil ihre Anwendung finden. Eine Unterscheidung in weitere Teilaufgaben ist also erforderlich. Eine Strukturübersicht eines solchen Mustererkennungssystems, was die Spracherkennung ja ist, findet man in Bild 1.3 (Funktionsblöcke und Bezeichnungen aus [Käm 90]). Nach der Vorverarbeitung und der Merkmalsgewinnung folgt die Klassifikation, die mit Hilfe zuvor bestimmter Prototypen und einem Lexikon eine Zuordnung von Merkmalsvektoren zu bestimmten Klassen treffen kann. Als Klassen bei der Spracherkennung dienen zum Beispiel Phoneme, also Laute wie Vokale und Konsonanten. Ergebnis eines Spracherkenners ist ein erkanntes Wort (Einzelworterkenner) oder, wie bei JANUS, ein ganzer Satz (kontinuierliche Spracherkennung).

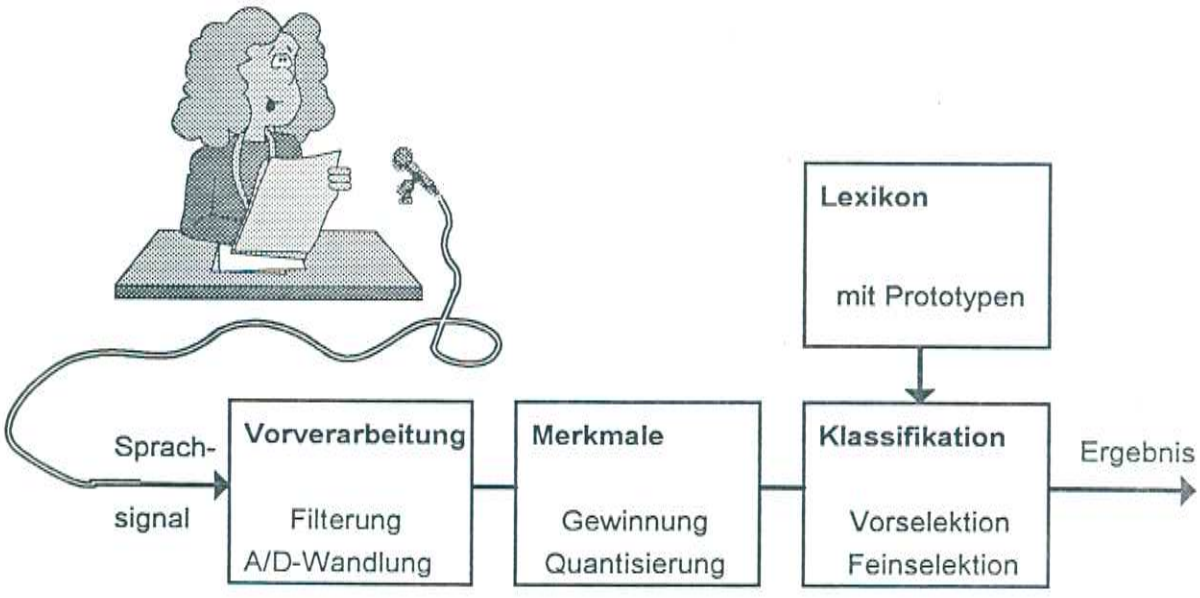


Bild 1.3: Die Komponenten des Spracherkennungssystems

1.2. Merkmalsextraktion, Dimensionsreduktion

Diese Diplomarbeit beschäftigt sich mit der Gewinnung von *relevanten* Merkmalen für den JANUS-Spracherkenner. Mögliche Verfahren bauen auf der Vorverarbeitung auf und liefern ein Ergebnis an den Klassifizierer. Vorverarbeitung und Klassifikation des Spracherkenners sind aus diesem Grunde ebenfalls zu berücksichtigen und werden deshalb in Kapitel 2 vorgestellt.

Bei der Spracherkennung hat man es stets mit einer großen Datenmenge zu tun, aus der über verschiedene Verfahren meist zwischen 10 bis 20 Merkmale gewonnen werden. Diese faßt man zu einem Vektor zusammen, der im weiteren als Datenvektor bezeichnet wird. Gängige Verfahren, wie z.B. die Verwendung von Filterbänken, schaffen zwar auch diese Reduktion, sind aber nicht auf die verwendeten Sprachdaten zugeschnitten. Außerdem berücksichtigen sie die speziellen Anforderungen des Klassifizierers nicht.

Obwohl diese Verfahren auch "Merkmale" gewinnen, werden sie hier als Teil der Vorverarbeitung angesehen, um sie von den in dieser Diplomarbeit untersuchten Verfahren abzugrenzen. Durch Hinzunahme von zeitlich benachbarten Datenvektoren oder der Kombination der Ergebnisse aus verschiedenen Vorverarbeitungsverfahren läßt sich nämlich die Erkennungsrate noch steigern. Dieses hat aber wieder eine Zunahme der Dimensionalität und dadurch des Rechenaufwands zufolge. Sie kann durch die hier untersuchten Methoden wieder reduziert werden und zwar so, daß der Klassifizierer möglichst gut zwischen den einzelnen Klassen unterscheiden kann.

Ein einfacher Weg wäre, die besten Merkmale auszusuchen und diese zu verwenden (*Selektion*). Hier sollen aber leistungsfähigere Verfahren untersucht werden, die den gesamten Merkmalsvektor abbilden, so daß man einen Vektor erhält, der weniger Komponenten beinhaltet. Es handelt sich also um eine *Dimensionalitätsreduktion*, die alle Muster eines hochdimensionalen Raumes in einen Raum kleinerer Dimensionalität abbildet. *Lineare* Abbildungen lassen sich mit *statistischen* Methoden herleiten. Sie haben die Form:

$$y = A \cdot x, \tag{1.1}$$

wobei x der Datenvektor mit der Dimension n und y ein neuer Vektor der Dimension m mit $m < n$ ist. Die Transformationsmatrix A ist eine $m \times n$ Matrix (m Zeilen und n Spalten).

Auf diese Weise funktionieren die Hauptachsentransformation (HAT) und die Lineare Diskriminanzanalyse (LDA), die in Kapitel 3 vorgestellt werden. Beide wurden im JANUS-Spracherkenner implementiert (Kapitel 4). Bei der LDA handelt es sich um ein Verfahren, das die zu unterscheidenden Klassen berücksichtigt und dadurch sehr leistungsfähig ist. Obwohl sie seit Jahren bekannt ist ([Fuk 72] u.a.), wurden erst in letzter Zeit Anwendungen in der Spracherkennung veröffentlicht, wie beispielsweise [Hae 92] und [Aub 93]. In der Spracherkennung sind verschiedene Klassendefinitionen denkbar. Auch ist offen, wie stark die Dimensionalität reduziert werden kann. Eine

Antwort auf diese Fragen geben die experimentellen Ergebnisse aus Kapitel 5. Für Versionen des JANUS-Spracherkenners konnten dabei Verbesserungen der Wortfehlerrate bis zu 47% erreicht werden.

Nichtlineare Abbildungen zur Dimensionalitätsreduktion sind ebenfalls denkbar, jedoch existiert kein entsprechendes Lösungsverfahren durch statistische Methoden. Durch ein *Neuronales Netz* könnte eine nichtlineare Abbildung gelernt werden, die die gestellte Aufgabe noch besser als HAT oder LDA erfüllt. Tatsächlich sind solche Netze Gegenstand der Forschung, z.B. [Kra 91], [Kam 93]. Bisher fehlt es aber an Lernverfahren, die wie die LDA klassenspezifische Dimensionalitätsreduktion bewerkstelligen. In dieser Diplomarbeit wurde ein neuer Ansatz entwickelt, der eine Lösung dieses Problems bietet (Kapitel 6).

2. Der Janus-Spracherkenner

Um Methoden zur Dimensionalitätsreduktion innerhalb der Merkmalsextraktion zu realisieren, muß man sich zunächst mit der zuvor stattfindenden Vorverarbeitung und der nachfolgenden Klassifikationseinheit beschäftigen. Als Grundlage für eine Merkmalsextraktion im JANUS-Spracherkenner dienen 16 melscale-Koeffizienten. Als Ausgabe soll die Merkmalsextraktion einen für die Klassifikationsstufe geeigneten Merkmalsvektor liefern.

2.1. Vorverarbeitung

Vom Sprachsignal zu den melscale-Koeffizienten.

Zum Verstehen von Sprache reicht ein Frequenzbereich bis ca. 8 kHz vollständig aus (beim Telefon werden nicht einmal 4 kHz übertragen). Für eine digitale Weiterverarbeitung des Sprachsignals muß laut dem Abtasttheorem die Abtastfrequenz mindestens doppelt so groß sein. In der Vorverarbeitung des JANUS-Spracherkenners erhält man bei einer Abtastrate von 16 kHz zu jedem Abtastzeitpunkt einen 16-Bit-Wert.

Für eine Klassifikation einzelner Laute werden aus dem digitalisierten Sprachsignal Zeitfenster (englisch: *frames*) herausgeschnitten und daraus das Kurzzeitspektrum bestimmt. Das Signal kann für diese kurze Zeitdauer als hinreichend stationär angesehen werden. Verwendung finden dabei Hamming-Fenster[1] einer Breite von 16 ms, die in Abständen von 10 ms über die Abtastwerte geschoben werden. Durch eine FFT (*fast fourier transform*) erhält man so alle 10 ms 256 Spektralwerte. Man verzichtet auf die darin enthaltene Phaseninformation und bildet das Leistungsspektrum, also das jeweilige Betragsquadrat der Spektralwerte.

Mit einer Filterbank von 16 trapezförmigen Bandpässen gewinnt man aus dem Leistungsspektrum die sogenannten *melscale*-Koeffizienten (MSC, englisch: *melscale coefficients*). Die Anordnung der Bandpässe über der Frequenz erfolgt dabei über die logarithmisch skalierte *melscale*. Diese ist der frequenzabhängigen Empfindlichkeit des menschlichen Gehörs nachvollzogen. Die Gewinnung der 16 MSC ist eine Form der Dimensionsreduktion, und man könnte sie auch der Merkmalsextraktion zuordnen.

[1] Hamming-Fenster und FFT werden beispielsweise in [Kam 89] behandelt.

Im Unterschied zu den später vorgestellten Methoden wird hierbei jedoch keine Dekorrelation durchgeführt. Auch wird die Unterscheidbarkeit (Diskriminanz) der Klassen nicht berücksichtigt. Lediglich das Spektrum wird quantisiert, wobei die Auflösung wie beim menschlichen Ohr für die hohen Frequenzen abnimmt. Besonders benachbarte Koeffizienten, die sich aus sogar etwas überlappenden Bandpässen ergeben, sind stark miteinander korreliert.

Eine ausführliche Beschreibung der Vorverarbeitung im JANUS-Spracherkenner findet man in [Suh 92]. Hier noch einmal, tabellarisch zusammengefaßt, die einzelnen Schritte der Vorverarbeitung:

Abtastung:	16 kHz Abtastrate, 16-Bit
256 Punkte FFT	16 ms Hamming-Fenster
Verschiebung der Fenster:	10 ms
Datenvektor:	16 *melscale*-Koeffizienten (MSC)

Tabelle 2.1: Die Vorverarbeitung im Janus-Systems

Neben den MSC finden auch Delta-MSC Verwendung. Durch sie wird die Dynamik, die zeitliche Änderung der Datenvektoren, beschrieben. Im Kapitel 4 über die Realisierung wird auf die Delta-MSC noch genauer eingegangen.

2.2. Klassifikation

Alle 10 ms liefert die Vorverarbeitung einen Datenvektor v(k). In der Merkmalsextraktionsstufe ensteht daraus der Merkmalsvektor y(k). Ohne einen solchen Verarbeitungsschritt stimmen beide überein. Aus den gegebenen Merkmalsvektoren wird vom Klassifizierer der wahrscheinlichste Satz bestimmt.

Das Wörterbuch

Die Aufgabe dieses Spracherkenners ist es, einen ganzen, gesprochenen Satz möglichst fehlerfrei zu erkennen. Es handelt sich um kontinuierliche Sprache, bei der Wörter ohne Sprechpause ineinander überfließen können. Um überhaupt Wörter aus den empfangenen Lauten identifizieren zu können, braucht man ein Wörterbuch, das die Aussprache dieser Wörter enthält. Jedes Wort ist darin durch Phoneme, wie Konsonanten oder Vokale, beschrieben.

Hier ein kurzer Auszug aus einem englischen Aussprachewörterbuch, wie es der JANUS-Erkenner verwendet.

Beispiel:

Wort	Aussprache (Phoneme)
ALL	AA L
AND	EH N DD
ANY	EH N IY
...	...

Die Grammatik

Obwohl die fließende Abfolge der Sprache die Erkennung der einzelnen Wörter erschwert, kann man einen Vorteil daraus ziehen, daß gewöhnlich die Wörter nicht in einer beliebigen Reihenfolge hintereinander vorkommen. Dies wird durch eine *Grammatik* berücksichtigt, die z.B. beschreibt mit welcher Wahrscheinlichkeit ein Wort hinter einem vorgegebenem Wort folgt (Bigramm-Grammatik). Für die Versuche innerhalb dieser Diplomarbeit wurde eine sehr einfache Grammatik, die nur Wortpaare (englisch: *word pairs*) enthält, benutzt. Sie gibt an, ob ein Wort hinter einem anderen folgen darf oder nicht und ist damit ein Sonderfall der Bigramm-Grammatik, bei der als Wahrscheinlichkeiten lediglich Null und Eins auftreten.

Das Hidden-Markov-Modell (HMM)

An dieser Stelle wird nicht auf die Trainingsalgorithmen für Hidden-Markov-Modelle eingegangen, sondern HMM's nur insofern erläutert, wie es in diesem Zusammenhang nötig ist. Ausführliche HMM Literatur findet man in [Hua 89].

Um den wahrscheinlichsten Satz für eine gegebene Folge von Merkmalsvektoren zu finden, bedient man sich eines *Hidden-Markov-Modells* (HMM). In diesem Modell gibt es Zustände (englisch: *states*) und Übergänge. Die Zustände entsprechen den zu unterscheidenden Klassen, z.B. Phonemen. Für die Zustände sind Emissionswahrscheinlichkeiten und für die Übergänge Übergangswahrscheinlichkeiten definiert. Unter der Emmissionswahrscheinlichkeit $f_{y|s}(y|s)$ versteht man diejenige Wahrscheinlichkeit, mit der ein Merkmalsvektor y in einem gegebenen Zustand s beobachtet wird. Die Übergangswahrscheinlichkeit $a_{s_i s_j}$ gibt an, mit welcher Wahrscheinlichkeit der Zustand s_j dem Zustand s_i folgt.

Zeichnet man die Zustände eines Wortes nebeneinander, so müssen diese Zustände, wie in Bild 2.1 dargestellt, von links nach rechts durchlaufen werden. Da mehrere hintereinander empfangene Merkmalsvektoren zu dem selben Zustand gehören können, gibt es auch einen Übergang in sich selbst (englisch: *self loop*).

aus dem Beispiel von oben

AND: EH..... N.....DD

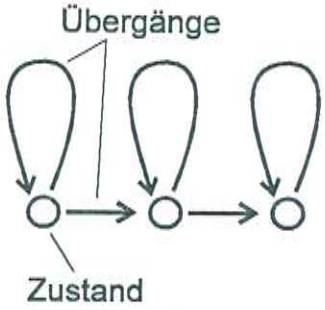


Bild 2.1: Ein Hidden-Markov-Modell

Nun sucht man die Zustandsfolge S mit der größten Gesamtwahrscheinlichkeit P(S). P(S) ergibt sich als Produkt der Emissions- und Übergangswahrscheinlichkeiten in der Folge.

$$\mathrm{P}(S) = \prod_k \mathrm{f}_{\mathrm{x}|\mathrm{s}}\left(y(k)\middle|s_k\right) \cdot a_{s_k s_{k+1}} \qquad (2.1)$$

Auf den Viterbi-Algorithmus, der die wahrscheinlichste Zufallsfolge findet, soll hier nicht näher eingegangen werden (siehe z.B. [Rab 89]).

In der Erkennungsphase werden HMM's für verschiedene Phoneme und Wörter aneinandergehängt. Auf Grund des im Wörterbuch festgelegten Vokabulars können nur bestimmte Phoneme hintereinander folgen und durch die Grammatik auch nur bestimmte Wortkombinationen auftreten (siehe Bild 2.2). Als Anfangs- und Endzustand eines Satz-HMM's verwendet JANUS stets das Stille-Modell (*silence*). Das geometrische Mittel der Anzahl der Wörter die einem Wort folgen können heißt *Perplexität* und ist ein Maß für die Schwierigkeit der Erkennungsaufgabe.

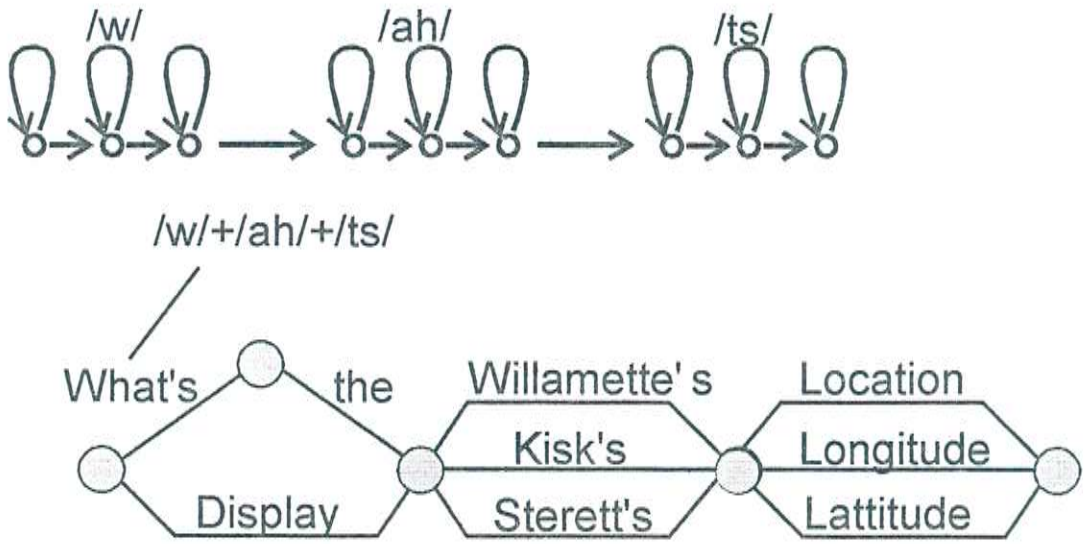


Bild 2.2: Mögliche Zustandsfolgen auf Grund von Wörterbuch und Grammatik. In diesem Beispiel wird ein Phonem aus jeweils drei verschiedenen Zuständen (Subphonemen) gebildet.

Emissions-Wahrscheinlichkeit

Anhand der empfangenen Mermalsvektoren wird festgestellt wie ähnlich ein Merkmalsvektor y einem zuvor gelernten Modell für ein Phonem ist. Dazu wird er mit prototypischen Merkmalsvektoren eines Phonems verglichen, die in einem Codebuch zusammengefaßt sind. Sie werden im folgenden Prototypen oder Codebuchvektoren m_c genannt.

Die Emissionswahrscheinlichkeit $f_{y|s}(y|s)$ für ein empfangenes Muster y bei gegebenem Zustand s wird approximiert. Bei *diskreten* HMM's sind die Emmissionswahrscheinlichkeiten diskrete Verteilungen über dem Codebuch, bei *kontinuierlichen* HMM's sind es Wahrscheinlichkeitsdichten über dem Merkmalsraum. Bei einem *semi-kontinuierlichen* HMM [Hua 89] setzt sich die Emissionswahrscheinlichkeit aus einer diskreten Verteilung über einem Codebuch und einer kontinuierlichen Verteilung für jedes Codebuchelement wie folgt zusammen:

$$f_{y|s}(y|s) = \sum_{c} f_{y|c}(y|c,s) \cdot P(c|s) \qquad (2.2)$$

$f_{y|s}(y|s)$ ist hier als Summe von gewichteten Wahrscheinlichkeitsdichtefunktionen $f_{y|c,s}(y|c,s)$ über den Codebuchindex c dargestellt . Der JANUS-Spracherkenner verwendet eine Approximination der semi-kontinuierlichen HMM's, die im folgenden erläutert wird.

Häufig nimmt man $f_{y|c,s}(y|c,s)$ als Gaußverteilung an [Kro 86]:

$$f(y|c,s) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_c|}} \cdot \exp\left(-\frac{(y-m_c)^T \cdot \Sigma_c^{-1} \cdot (y-m_c)}{2} \right), \quad (2.3)$$

wobei m_c der Mittelwert und Σ_c die Kovarianzmatrix der Verteilung ist. Dabei entspricht der Mittelwert dem schon erwähnten Codebuchvektor.

Beim JANUS-Spracherkenner wurden folgende vereinfachende Annahmen gemacht:

Die Kovarianzmatrix Σ_c und damit auch ihre Inverse Σ_c^{-1} wurde durch die Einheitsmatrix ersetzt (mit ausreichend vielen varianzlosen Gaußverteilungen lassen sich varianzbehaftete Gaußverteilungen annähern). Der Vorfaktor in obiger Formel ist dann für alle Zustände konstant und braucht nicht weiter berücksichtigt zu werden. Außerdem steht daraufhin im Exponent ein Ausdruck, der lediglich die Distanz des Merkmalsvektors y zum Prototyp m_c enthält:

$$f(y|c,s) \approx const \cdot \exp\left(-\frac{|y-m_c|^2}{2} \right). \quad (2.4)$$

Statt der vollständigen Summe über die Verteilungen wurde nur der größte Wert genommen. Dieser ergibt sich für die kleinste Distanz. Es wird also lediglich der zum Merkmalsvektor y nächstliegende Prototyp m_c gesucht und die Distanz zwischen ihnen berechnet.

Um nicht numerische Probleme bei der Realisierung des Viterbi-Algorithmus zu bekommen, ist es vorteilhafter, anstatt viele Wahrscheinlichkeiten zu multiplizieren, deren Logarithmus zu addieren. Dann ergibt sich Formel (2.1) zu

$$\log(P(S)) = \sum_k \log\left(f_{x|s}(y(k)|s_k)\right) + \sum_k \log\left(a_{s_k s_{k+1}}\right). \quad (2.5)$$

Das hat auch den Vorteil, daß die Exponentialfunktion aus der Gaußverteilung verschwindet. Im JANUS-Spracherkenner verwenden wir den negativen Logarithmus und nennen diesen *Strafe*.

Die Gewichtung P(c|s) in Formel (2.2) ist die Häufigkeit der Vektoren y, die dem Prototypen m_c nächstliegend sind, bezogen auf die Gesamtzahl der Merkmalsvektoren y des Zustandes s. Sie entspricht den diskreten Verteilungen des semi-kontinuierlichen HMM's.

Faßt man alle Näherungen und Umrechnungen zusammen und läßt konstante Terme beiseite, so erhält man:

$$\text{Strafe}(y|s) = -\log\left(\text{P}(c|s)\right) + h \cdot |y - m_c|^2 \qquad (2.6)$$

$$\text{mit } c = \arg\min_i\left(|y - m_i|^2\right)$$

Die Strafe wird dem Viterbi-Algorithmus anstatt $-\log(f_{y|c,s}(y|c,s))$ als Funktion der Emmissionswahrscheinlichkeit gegeben. Der darin enthaltene Skalierungsfaktor *h* läßt sich auf Grund der starken Vereinfachungen nicht herleiten, sondern muß experimentell bestimmt werden. Er wird dabei so gewählt, daß die Erkennungsrate des Gesamtsystems möglichst groß ist.

Fassen wir noch einmal zusammen, welche Dinge zur Berechnung der Strafen für einen bestimmten Zustand notwendig sind:

- Ein Codebuch mit den Prototypen des Zustandes.
- Eine Verteilung der Häufigkeit der Prototypen über diesem Codebuch.
- Ein Faktor h der die Distanz zur logarithmierten Häufigkeit gewichtet.

Codebuch und Häufigkeitsverteilungen werden durch ein Viterbi-Verfahren für HMM's für die zur Verfügung stehenden Trainingssätze (siehe Abschnitt: 5.2) iterativ gelernt. Alle Codebücher und Verteilungen, die sich durch ein Training ergeben haben, werden fortan als *Parametersatz* bezeichnet.

Übergänge

Wir haben gesehen, wie die Strafe für einen Zustand bestimmt wird. Da Experimente ergeben haben, daß ein Phonem im Mittel aus sechs hintereinander folgenden Merkmalsvektoren besteht, wurden pro Phonem ebenfalls sechs verschiedene Zustände gewählt (siehe Bild 2.3). Es gibt jeweils drei mögliche Übergänge a, b und c mit festen Übergangswahrscheinlichkeiten, die hier exemplarisch nur für einen Zustand eingezeichnet sind. Die b-Übergänge sind somit am häufigsten und werden auch am wenigsten bestraft. Auch eine Übergangsstrafe in ein neues Wort ist gesondert festgelegt und ergibt sich aus dem negativen Logarithmus der Bigramm-Wahrscheinlichkeit, bzw. aus einer Konstanten, wenn eine Wortpaar-Grammatik verwendet wird.

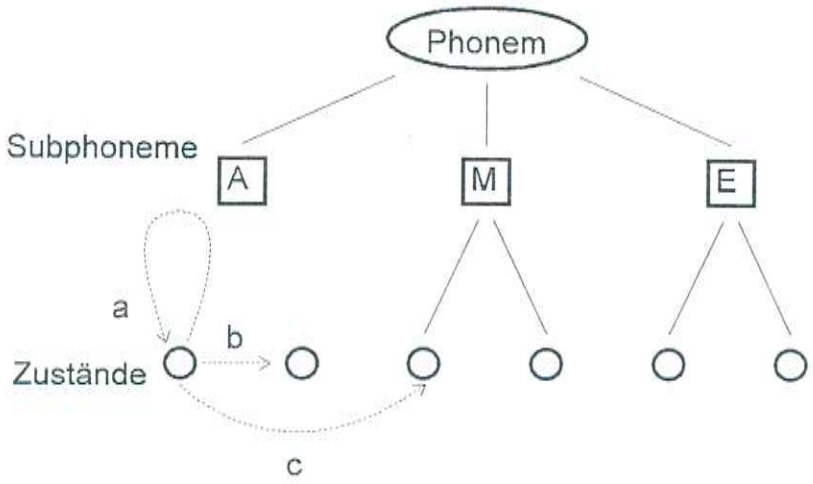


Bild 2.3: Übergänge zwischen den Zuständen

Subphoneme

Für jeden Zustand ein eigenes Codebuch und eine Häufigkeitsverteilung einzurichten hat sich als zu feine Unterteilung herausgestellt. Deshalb teilen sich alle Zustände ein gemeinsames Codebuch und jeweils zwei Zustände eine Häufigkeitsverteilung. Diese Zustandspaare bilden gewissermaßen den Anfangs-, Mittel- und Endlaut eines Phonems. Sie sind hier als Subphoneme A, M und E aufgeführt. Bild 2.4 vermittelt noch einmal einen Eindruck über das Codebuch und die Häufigkeitsverteilungen.

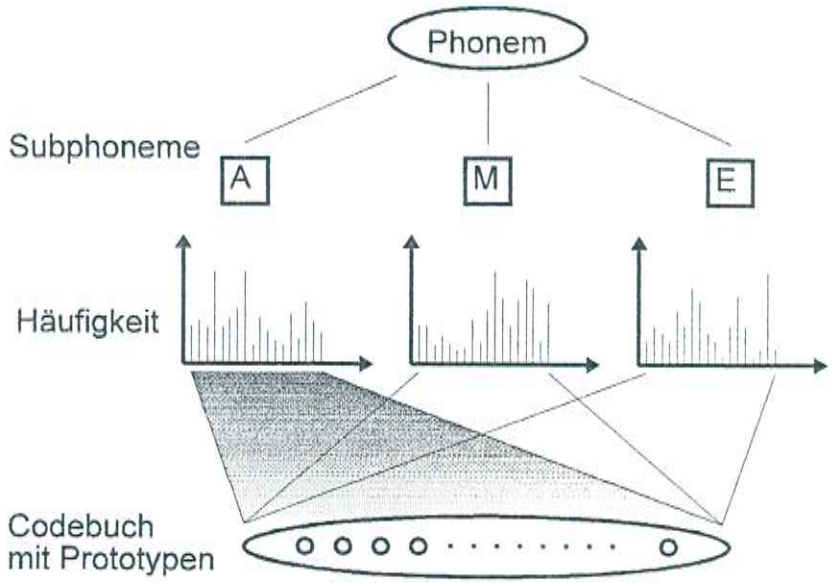


Bild 2.4: Codebuch und Häufigkeitsverteilung eines Phonems

Bemerkung

Ob man Phoneme oder Subphoneme als Klassen ansieht und versucht zwischen diesen Klassen zu unterscheiden, wird später noch eine große Rolle spielen. Wenn sich durch eine Änderung der Merkmale die Größenordnungen der Distanzen |y-m| ändert, so stimmt das durch den Skalierungsfaktor h eingestellte Verhältnis zu der logarithmierten Häufigkeit nicht mehr überein. Der Faktor muß dann neu bestimmt werden.

2.3. Versionen, Training und Test

Die einzelnen Verarbeitungsstufen des JANUS-Spracherkenners lassen sich variabel kombinieren. Durch höheren Aufwand läßt sich die Leistung des Systems steigern. Neue Methoden sollten erst einmal an den überschaubaren, weniger aufwendigen Versionen getestet werden. Erst eine Leistungssteigerung des Systems hier ist Grund für die zeit- und rechenintensiven Versuche mit einem leistungsstarken System. In diesem Abschnitt werden drei unterschiedlich aufwendige und damit leistungsfähige Systeme vorgestellt, die im Rahmen dieser Diplomarbeit untersucht wurden.

Versionen

Für die Versuche zur Dimensionalitätsreduktion wurden drei verschiedene Systeme bzw. Versionen verwendet. Das erste arbeitet nur mit 16 *melscale*-Koeffizienten (MSC). Für jedes von 48 Phonemen existiert ein Codebuch mit 50 Merkmalsvektoren. Zu jedem Codebuch existieren 3 Häufigkeitsverteilungen entsprechend der 3 Subphoneme (Segmente eines Phonems). Außerdem gibt es noch ein Codebuch und eine Verteilung zur Modellierung von Stille (englisch: *silence*), also Zeitpunkten, zu denen nicht gesprochen wird.

Das zweite System verwendet zusätzlich noch 16 Delta-MSC. Für diese existieren eigene Codebücher und Häufigkeitsverteilungen. MSC und Delta-MSC werden zunächst total getrennt voneinander behandelt. Die berechneten Strafen für einen gegebenen Zustand werden anschließend gewichtet und addiert.

$$\text{Strafe} = \alpha \cdot \text{Strafe}_{\text{MSC}} + (1 - \alpha) \cdot \text{Strafe}_{\text{Delta-MSC}} \qquad (2.7)$$

Der Faktor α wurde einmal experimentell bestimmt und danach nicht mehr verändert. Da hier praktisch zwei Erkenner parallel laufen, ist der Rechenaufwand und der Speicherbedarf fast doppelt so groß.

Das dritte System arbeitet, was die Koeffizienten angeht, genau wie das zweite. Bei der Beschreibung der Wörter werden aber kontextabhängige Phoneme (*Triphone*) verwendet. Im Aussprachewörterbuch wird bei den oben verwendeten *Monophonen* noch unterschieden vor und nach welchen Phonemen sie stehen, denn durch diesen Kontext wird die Artikulation der Sprachlaute beeinflußt.

Mit einem Auszug aus dem Triphon-Wörterbuch des JANUS-Systems soll der Unterschied von Monophonen und Triphonen verdeutlicht werden:

Beispiel

Auszug aus dem JANUS-Wörterbuch für kontextabhängige Phoneme:

Wort	Aussprache (Phoneme)
ALL	AA17 L74
AND	EH49 N32 DD17
ANY	EH49 N68 IY79
...	...

Die Beiden Monophone /EH/ des Beispiels ergeben das gleiche Triphon /EH49/, da sie beide zwischen Stille(*Silence*) und /N/ stehen. Das /N/ folgt zwar in beiden Wörtern einem /EH/, der nächste Laut ist aber ein anderer, darum ergeben sich diesmal verschieden Triphone /N32/ und /N68/.

	Merkmale	Speicherbedarf: (für Codebücher und Häufigkeitsverteilungen im Vergleich zu System 1)	Rechenaufwand: (im Vergleich zu System 1)
System I	16 MSC	einfach	einfach
System II	16 MSC + 16 Delta-MSC	zweifach	doppelt
System III (kontext-abhängig)	16 MSC + 16 Delta-MSC	Codebücher: zweifach Häufigkeitsverteilung: hundertfach	doppelt

Tabelle 2.2:Eigenschaften der verwendeten Systeme

Ein Codebuch wird von allen Triphonen eines Phonems geteilt, es bleiben also insgesamt 48 Codebücher. Nicht so die Anzahl der Häufigkeitsverteilungen; sie steigt bei ca. 50 Triphonen pro Phonem mit einer Unterteilung in jeweils drei Sub-Triphonen auf ungefähr

$$50 \cdot 48 \cdot 3 = 7200.$$

Der Rechenaufwand entspricht dem von System 2, da bei jedem Wort bekannt ist für welches Triphon eine Strafe berechnet werden soll. Der Speicherbedarf ist allerdings beträchtlich größer.

In Tabelle 2.2 sind noch einmal die wichtigsten Unterschiede der drei verwendeten Versionen zusammengestellt.

Training und Test

Es gibt jeweils ein Programm zum Trainieren und Testen. Trainiert werden die Codebücher und Häufigkeitsverteilungen mit dem angesprochenen Virterbi-Algorithmus.

Zunächst werden alle Trainingssätze eingelesen, dann die Codebuchvektoren durch das Mittel der Merkmalsvektoren, die ihnen jeweils zugeordnet wurden, ersetzt. Zusammen mit den neu ermittelten Häufigkeitsverteilungen werden sie als Parametersatz abgelegt. Um zu einem Testergebnis zu gelangen, das sich durch Weitertrainieren nicht mehr wesentlich verbessern läßt, sind ungefähr 10 dieser

Iterationen nötig. Zum Initialisieren werden ca. 300 handsegmentierte[2] Sätze eingelesen und die Merkmalsvektoren nach Phonemen getrennt. Auf den so entstandenen Phonemgruppen wird eine Ballungsanalyse mit der "k-means"- (oder Basic-ISODATA [Qua 92]) Methode durchgeführt, d.h. in Untergruppen aufgeteilt. Die Mittelpunktsvektoren der 50 Untergruppen sind die initialen Codebuchvektoren.

Um einen Parametersatz für das System III (kontextabhängig) zu erhalten, wird von einem Parametersatz des Systems II ausgegangen. Die nun verwendeten Triphone übernehmen zunächst das Codebuch des entsprechenden Monophons, ebenso die Häufigkeitsverteilungen. Danach werden diese kontextabhängigen Modelle trainiert. Hierbei sind ebenfalls ca. 10 Iterationen notwendig bis man ein Minimum der Fehlerrate für die Testdaten erreicht.

Beim Test wird zu jedem Testsatz eine Hypothese ausgegeben. Mit einem weiteren Programm, das diese Hypothesen mit den korrekten Sätzen vergleicht, können automatisch Fehler- und Erkennungsrate, verwechselte Wörter und vieles mehr festgestellt werden.

---

[2] *Handsegmentiert* bedeutet, die Merkmalsvektoren wurden durch einen Menschen einem bestimmten Phonem zugeordnet. Dabei wird auch falsche oder unvollständige Aussprache der Wörter berücksichtigt. Im Gegensatz dazu werden beim *automatischen Segmentieren* die Merkmalsvektoren anhand des Aussprachewörterbuchs und des vorgegebenen Wortlautes des Satzes möglichst gut angepaßt.

3. Statistische Methoden zur Dimensionalitätsreduktion

In diesem Kapitel werden zwei statistische Methoden zur Dimensionalitätsreduktion vorgestellt. Sie arbeiten mit einer Transformationsmatrix, die die Anzahl der Koeffizienten reduziert. Die Hauptachsentransformation (HAT) und dazu sehr ähnliche Verfahren findet man auch unter dem Namen diskrete Karhunen-Loève-Entwicklung, in der englischen Literatur als Principle Components Analysis (PCA) oder Factor Analysis [Fuk 72]. Die Lineare Diskriminanzanalyse (LDA) berücksichtigt dem gegenüber noch die vom Klassifizierer zu trennenden Klassen und ist somit sehr viel leistungsfähiger.

3.1. Merkmalsextraktion durch lineare Transformation

Ein wichtiges Teilproblem in der Mustererkennung ist die Gewinnung relevanter Merkmale. Die Verarbeitungsstufe, die dies bewerkstelligen soll, bekommt als Eingabe zu jedem diskreten Zeitpunkt k von der Vorverarbeitungsstufe einen Datenvektor x(k) geliefert (siehe Bild 1.3). Dieser Datenvektor wird in einen Merkmalsvektor y(k) abgebildet, so daß der nachfolgende Klassifikator nach [Rus 88]

- weniger Merkmale verarbeiten muß und
- nur noch für die Klassentrennung wichtige Information erhält.

Die einfachste Methode wäre, sich nach bestimmten Kriterien einige Koeffizienten des Datenvektors auszuwählen. Dieses Vorgehen nennt man *Selektion*. Weit besser ist aber, einen funktionalen Zusammenhang zwischen Daten- und Merkmalsvektor aufzustellen und diese Abbildung bezüglich eines Gütekriteriums zu optimieren. Dabei ist sichergestellt, daß prinzipiell Informationen über alle Koeffizienten des Datenvektors in die Merkmale eingehen können. Das Gütekriterium sollte nur von den Merkmalsvektoren abhängen, so daß man unabhängig vom Klassifizierer ist. Im folgenden werden *lineare* Abbildungen behandelt, für die optimale Lösungen durch statistische Methoden bekannt sind. Es liegt eine Abbildung der Form $y = A \cdot x$ vor, wobei es sich bei A um eine $m \times n$ Matrix (m Zeilen und n Spalten) handelt. Sie bildet die n Koeffizienten des Datenvektors x in m ($m < n$) Koeffizienten des Merkmalsvektors y ab (Bild 3.1).

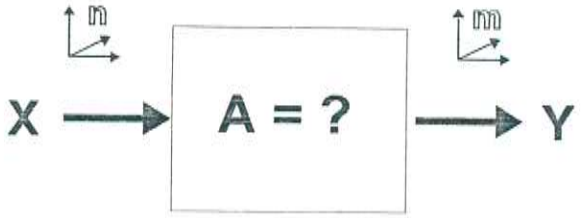


Bild 3.1: Dimensionsreduzierende,
lineare Abbildung mit m < n.

Für das Verständnis der Gütekriterien sind folgende Definitionen[1] notwendig:
Total-Scattermatrix (Kovarianzmatrix):

$$T = \mathrm{E}\left\{(\mathbf{X} - M_0)(\mathbf{X} - M_0)^{\mathrm{T}}\right\} \tag{3.1}$$

Within-Class-Scattermatrix (mittlere Klassenkovarianz, Klasse ω_i):

$$W = \sum_i \mathrm{P}(\omega_i) \cdot \mathrm{E}\left\{(\mathbf{X} - M_i)(\mathbf{X} - M_i)^{\mathrm{T}} | \omega_i\right\} \tag{3.2}$$

Between-Class-Scattermatrix (Kovarianz der Mittelwerte)

$$B = \sum_i \mathrm{P}(\omega_i) \cdot (M_i - M_0)(M_i - M_0)^{\mathrm{T}} \tag{3.3}$$

Dabei sind:

X	Zufallsvariable (Datenvektoren oder Merkmalsvektoren)
M_0	Gesamtmittelwert
M_i	Mittelwert der Muster aus Klasse ω_i

Sämtliche Matrizen sind symmetrisch. Die Diagonalelemente von T sind die Varianzen der einzelnen Koeffizienten. Die Nichtdiagonalelemente stellen die Kovarianzen zwischen den Koeffizienten dar. Sie geben an wie stark die Koeffizienten miteinander korreliert sind. In Bild 3.2 ist eine Punktmenge im zweidimensionalen Raum dargestellt, bei der die Koeffizienten x1 und x2 sehr stark miteinander korreliert sind. Wenn z.B. x1 groß ist, dann trifft das auch für x2 zu und umgekehrt. Am geeignetsten für die Klassifikation sind nicht korrelierte Koeffizienten, gleichbedeutend mit einer

[1]Die Notation und Benennung ist an [Fuk 72] angelehnt, die deutschen Ausdrücke sind in Klammer zugefügt.

diagonalen Kovarianzmatrix T mit von Null verschiedenen Varianzen in der Diagonale und allen Kovarianzen gleich Null.

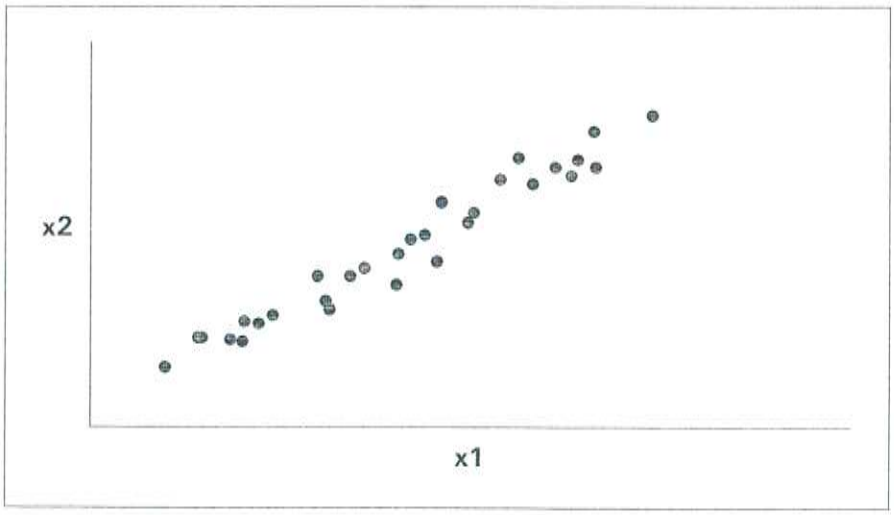


Bild 3.2: Punktmenge mit stark korrelierten Koeffizienten

3.2. Die Hauptachsentransformation (HAT)

Bei der *Hauptachsentransformation* (HAT) wird keine Voraussetzung über die zu trennenden Klassen gemacht. Die Datenvektoren bilden eine Punktwolke im Datenraum. Die Kovarianzmatrix T läßt sich dann als Ellipsoid (bei 2 Dimensionen als Ellipse) in diesem Raum darstellen. Bild 3.3 soll dies verdeutlichen.

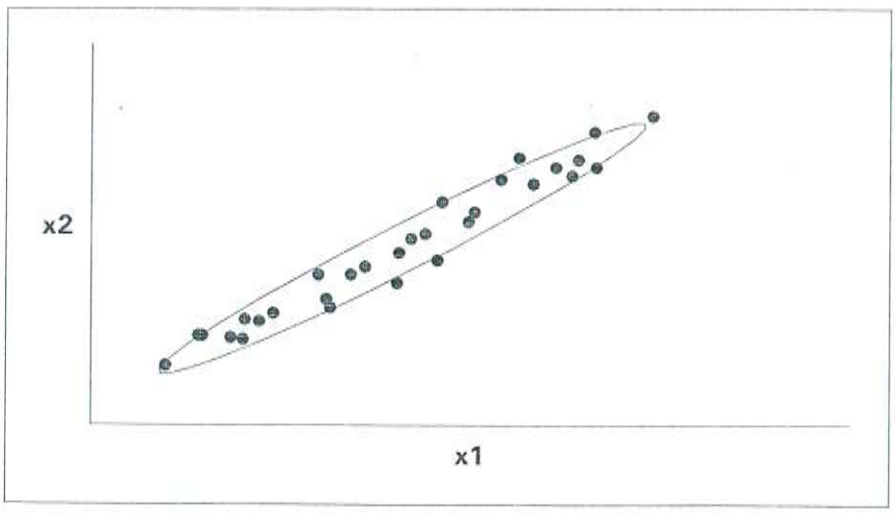


Bild 3.3: Kovarianzmatrix als Ellipse

Die Eigenvektoren der Kovarianzmatrix T entsprechen den Hauptachsen der Ellipse (des Ellipsoids). Die Hauptachsen sind stets orthogonal zueinander. Sie werden normiert und können somit als ein neues orthonormales Basissystem (y_1, y_2) angesehen werden.

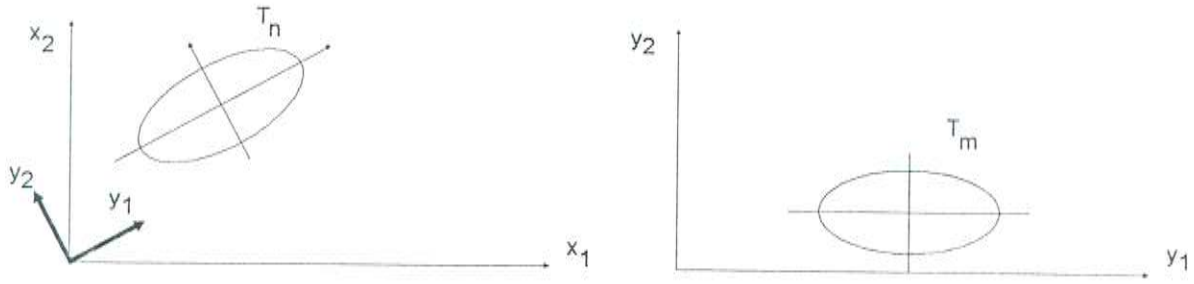


Bild 3.4: Kovarianzmatrix vor und nach der Transformation

Transformiert man auf die neue Basis, erhält man eine diagonale Kovarianzmatrix für die Punkte im neuen Raum. Die Hauptachsen (Eigenvektoren) der neuen Kovarianzmatrix sind achsenparallel (siehe Bild 3.4). Das bedeutet auch, daß die Koeffizienten dekorreliert sind. Die einzelnen Varianzen entsprechen den jeweiligen Eigenwerten der zuvor berechneten Eigenvektoren. Diese Eigenwerte bleiben bei der Transformation erhalten. Um eine Dimensionanlitätsreduktion zu erreichen, ordnet man die n Eigenwerte nach ihrer Größe und wählt die m Eigenvektoren aus, die zu den größten Eigenwerten gehören . Dadurch hat man sich für die Komponenten entschieden, die am stärksten variieren.

Die Transformationsmatrix A setzt sich folgendermaßen zusammen:

$$A = (\phi_1 \phi_2 \ldots \phi_m)^T, \quad \lambda_1 > \lambda_2 > \ldots > \lambda_n \tag{3.4}$$

mit den normierten Eigenvektoren ϕ_i und den Eigenwerten λ_i der Kovarianzmatrix T.

Die Hauptachsentransformation hat starke Ähnlichkeit mit der diskreten Karhunen-Loève-Entwicklung. Bei dieser wird der quadratische Fehler, der durch eine Reihenentwicklung mit m orthonormalen Basisvektoren entsteht minimiert. Der Unterschied in der Lösung für die Transformationsmatrix A besteht nur darin, statt der Kovarianzmatrix die Streumatrix zu verwenden (Streumatrix $S=E\{\mathbf{X}\cdot\mathbf{X}^T\}$).

Sind die Daten mittelwertfrei stimmt die Streumatrix mit der Kovarianzmatrix überein, und die Lösungen sind damit identisch.

In [Rus 88] wird ein *Datenreduktions-Faktor* d_r definiert, mit dem man das Verhältnis der nach der Transformation verbleibenden Varianz zur ursprünglichen Gesamtvarianz beschreiben kann:

$$d_r = \frac{\sum_{i=1}^{m} \lambda_i}{\sum_{j=1}^{n} \lambda_i}, \tag{3.5}$$

mit den Eigenwerten λ_i der Kovarianzmatrix T. Da die Varianz der Koeffizienten zur Klassifikation benutzt wird, sollte dieser Faktor nicht zu klein werden, sondern möglichst nahe bei 1 bleiben.

3.3. Die Lineare Diskriminanzanalyse (LDA)

Die *Lineare Diskriminanzanalyse* (LDA) maximiert ebenfalls die Varianz der verbleibenden Koeffizienten. Darüber hinaus sollen die Vektoren, die zur selben Klasse gehören, dicht beieinander bleiben. Dies erreicht man dadurch, daß man die Varianz innerhalb einer Klasse minimiert. Beide Ziele vereinigt man in dem folgenden Gütekriterium:

$$J_1(m) = \frac{\det(T_m)}{\det(W_m)} = \det\left(W_m^{-1} \cdot T_m\right) \tag{3.6}$$

Die Variable, bzw. der Index m sollen dabei andeuten, daß das Kriterium und die Scattermatrizen in dem auf m Dimensionen reduzierten Raum R^m definiert sind.

In der Literatur findet man Gütekriterien, die statt der Determinante die Spur der Matrizen verwenden. Die Lösung für die Transformationsmatrix (s.u.) ist dieselbe. Es gibt aber auch Gütekriterien (3.7) und (3.8), die mit der Between-Class-Scattermatrix arbeiten. Dabei wird dann die Varianz der Klassenmittelwerte maximiert. Die Lösungen sind entsprechend gleich, nur mit den jeweils anderen Matrizen:

$$J_2(m) = \frac{\det(B_m)}{\det(W_m)} = \det\left(W_m^{-1} \cdot B_m\right) \tag{3.7}$$

$$J_3(m) = \frac{\det(B_m)}{\det(T_m)} = \det\left(T_m^{-1} \cdot B_m\right) \tag{3.8}$$

Die Scattermatrizen im Raum R^m lassen sich natürlich nicht ohne bekannte Transformationsvorschrift berechnen. Durch die Voraussetzung einer linearen Transformation der Datenvektoren x mit der Matrix A gilt jedoch für alle Scattermatrizen[2] S_m :

$$S_m = A \cdot S_n \cdot A^T \tag{3.9}$$

Nun können die bekannten Scattermatrizen im Raum R^n zusammen mit der Transformationsmatrix A in das jeweilige Kriterium eingesetzt werden. Auf den relativ langen Lösungsweg (siehe z.B. [Fuk 72]) wird hier verzichtet und nur das Endergebnis angegeben:

$$A = (\phi_1 \phi_2 \ldots \phi_m)^T, \quad \lambda_1 > \lambda_2 \geq \ldots > \lambda_n \tag{3.10}$$

Dies ist dieselbe Lösung wie bei der Hauptachsentransformation, diesmal sind aber ϕ_i die Eigenvektoren der Matrix $W_n^{-1} \cdot T_n$ (im Falle des Kriteriums J_1, bei J_2 und J_3 mit den entsprechenden Matrizen).

Die Eigenvektoren sind hier nicht orthogonal, da $W_n^{-1} \cdot T_n$ bis auf Ausnahmefälle keine symmetrische Matrix ist. Deshalb stimmen auch die Basisvektoren nicht mehr unbedingt mit den Eigenvektoren überein, denn es gilt nicht wie bei der Hauptachsentransformation:

$$A^{-1} = A^T \tag{3.11}$$

Es handelt sich um ein *Eigenwertproblem*, dessen Lösung nicht eindeutig ist, denn die Länge der Eigenvektoren ist noch nicht festgelegt. Für das Gütekriterium spielt das keine Rolle. Ein konstanter Faktor, zum Beispiel in der Matrix A würde beide Scattermatrizen W_m und T_m im Raum R^m gleichermaßen vergrößern und sich bei der Berechnung des Gütekriteriums wieder herauskürzen. Um zu einer eindeutigen Lösung

[2] S läßt sich beliebig durch eine der Matrizen T, W oder B ersetzen.

zu gelangen, hält man die Within-Class-Scattermatrix W_m konstant und maximiert die Total-Scattermatrix T_m.

Bei der praktischen Umsetzung geht man nicht den direkten Weg. Hierbei müßten nämlich die Eigenwerte und Eigenvektoren der nicht symmetrischen Matrix $W_n^{-1}T_n$ berechnet werden. Das Verfahren der *simultanen Diagonalisierung* hingegen liefert in mehreren Schritten eine Transformationsmatrix mit Eigenvektoren ϕ_i, die gleich so skaliert sind, daß

- die Matrix T_m eine Diagonalmatrix und
- die Matrix W_m eine Einheitsmatrix ist.

Dabei müssen nur Eigenwerte und Eigenvektoren *symmetrischer*[3] Matrizen berechnet werden. Mit der oben eingeführten Ellipsendarstellung kann die Lösung der simultanen Diagonalisierung sehr anschaulich erklärt werden [Fuk 72, S.33ff].

Die Abbildung der Within-Class-Scattermatrix auf die Einheitsmatrix erleichtert die Approximation der Verteilung der Merkmalsvektoren im Raum R^m durch einfache Gaußverteilungen, wie sie auch vom JANUS-Spracherkenner benutzt werden. Durch diese Art der Skalierung der einzelnen Koeffizienten des Datenvektors lassen sich unterschiedlich vorverarbeitete Größen zu einem hochdimensionalen Vektor zusammenfassen und geeignet in den Merkmalsraum transformieren. Im nächsten Kapitel wird dieser zusammengesetzte Vektor zur Unterscheidung zum ursprünglichen Datenvektor mit Mustervektor bezeichnet. Die Koeffizienten dieses Vektors werden durch die LDA-Transformationsmatrix A in den Merkmalsraum R^m abgebildet, so daß dort die Trennbarkeit der Klassen optimal bezüglich des Gütekriteriums (3.6) unter Voraussetzung einer linearen Transformation ist.

[3]Für die Berechnung von Eigenwerten und Eigenvektoren von symmetrischen Matrizen eignet sich z.B. das Jacobi-Verfahren in [Bro 87, S.742], das auch für diese Diplomarbeit verwendet wurde. Andere numerische Verfahren und Quellcodes in der Programmiersprache C findet man in [Pre 88].

3.4. Anwendung im Überblick

Variationsmöglichkeiten bei der Anwendung der LDA, neben der Wahl eines der oben vorgestellten Gütekriterien (3.6)-(3.8) u.a., sind

- Wahl der Dimensionalität m des Merkmalsraumes,
- Definition der zu unterscheidenden Klassen,
  (in der Spracherkennung z.B.:
  Phoneme, Subphoneme, kontextabhängige Phoneme, u.a.)
- modifizierte Scattermatrizen,
  durch Ersetzen der A-Priori-Wahrscheinlichkeiten mit anderen Gewichtungen
- Zusammensetzung des Mustervektors x(k).

Durch Programme in der Programmiersprache C und der Erweiterung der Sprach-erkennersoftware wurde im Laufe dieser Diplomarbeit die Möglichkeit geschaffen, die vorgestellten Verfahren in vielen Variationsmöglichkeiten auf dem JANUS-Sprach-erkenner anzuwenden. Auf Grund deren Vielfalt und den langen Trainings- und Testzeiten wurden nur gezielte Versuchsreihen unternommen, um die Auswirkungen auf die Erkennungsleistung in Abhängigkeit von der ***Merkmalsdimension m***, der verwendeten ***Klassendefinition*** und der Zusammensetzung des ***Mustervektors x(k)*** zu untersuchen. Einzelheiten der Realisierung folgen im nächsten, Ergebnisse der Experimente im übernächsten Kapitel.

4. Realisierung der statistischen Methoden

Trainings- und Testprogramm des Janus-Spracherkenners wurden im Laufe dieser Diplomarbeit so erweitert, daß die vorgestellten statistischen Methoden in vielerlei Variationen angewandt werden können. Das Trainingsprogramm kann nun auch eine Klassendatei erstellen, mit deren Hilfe eine Transformationsmatrix zur Dimensionsreduktion ermittelt werden kann. Die Berechnung dieser Matrix geschieht durch separat erstellte Programme, die deshalb auch für andere Anwendungen verwendet werden können.

4.1. Transformationsmatrix

Bevor der Spracherkenner mit neu eingebauter Dimensionalitätsreduktion trainiert werden kann, muß die Transformationsmatrix berechnet werden. Im vorigen Kapitel haben wir gesehen, daß die Lösung für die Hauptachsentransformation (HAT) und die Lineare Diskriminanzanalyse (LDA) sehr ähnlich sind. Beide arbeiten mit Scattermatrizen, die aus den Trainingsdaten erzeugt werden können. Die Trainingsdaten liegen als vorverarbeitete Sprachaufnahmen in Form von 16 MSC-Datenvektoren v(k) vor.

Die LDA benötigt dabei eine Zuordnung dieser Datenvektoren in Klassen. Welche Klassen sollen aber nun berücksichtigt werden? Die Wörter bestehen zwar aus verschiedenen, eindeutig festgelegten Phonemen (Lauten), aber das verwendete Hidden-Markov-Modell unterscheidet auch noch zwischen Subphonemen, von denen jeweils drei ein Phonem bilden. Verschiedene Klassendefinitionen sind also möglich und sollten getestet werden können. Es stellt sich aber auch die Frage, wie Datenvektoren einer größeren Trainingsmenge vorklassifiziert werden.

Klassifikation der Trainingsdaten

Mit dem unmodifizierten Trainingsprogramm ohne dimensionalitätsreduzierende Merkmalsextraktion wird ein Parametersatz erzeugt, der Codebücher und Häufigkeitsverteilungen enthält. Diese werden später vom Testprogramm benutzt, um dem Programm unbekannte Sätze zu erkennen. Während des Trainings ist der korrekte Satz bekannt und wird mit Hilfe des Aussprachewörterbuchs in eine Phonemfolge um-

gewandelt. Obwohl diese Folge nicht exakt mit der tatsächlichen, eventuell etwas fehlerhaft oder unvollständig ausgesprochenen Folge von Lauten der Äußerung übereinstimmen muß, findet während der Trainingsphase eine Zuordnung der Datenvektoren auf Phoneme, Subphoneme und Zustände durch den Viterbi-Algorithmus statt. Diese Zuordnung ist abhängig von dem verwendeten Parametersatz. Trotz dieser Einschränkungen läßt sich eine solche Zuordnung auf diese Weise zufriedenstellend und vor allem automatisch durchführen.

Mit einem schon ausreichend trainierten Parametersatz wird eine weitere Iteration durchgeführt, bei der alle Trainingsdaten eingelesen und in der oben beschriebenen Weise den Klassen zugeordnet werden. Diese Information wird in einer Klassendatei für jeden Satz abgespeichert. Von welcher Art die Klassen hierbei sind (Phoneme, Subphoneme oder Zustände) ist durch einen Parameter des Programms einstellbar.

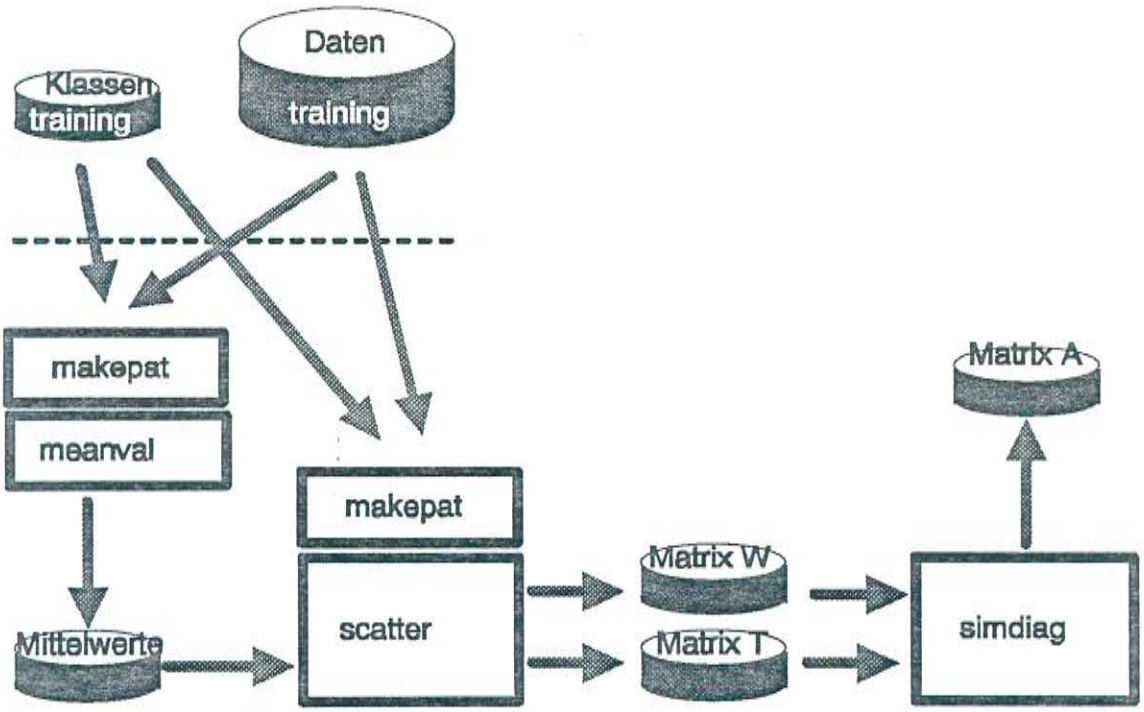


Bild 4.1: Übersicht der Programme zur Ermittlung der Tranformationsmatrix

Programme sind als Blöcke, Dateien als Scheiben dargestellt. Die Programme **meanval** und **scatter** benutzen dieselben Routinen **makepat** zum Einlesen der Daten und Klassen. In dieser können Delta-MSC oder andere Koeffizienten aus den aus 16 Melscale-Koeffizienten (MSC) bestehenden Datenvektor v(k) berechnet werden. Außerdem können die eingelesenen Daten durch eine Matrix transformiert werden (s.u.).

Berechnung der Transformationsmatrix

Mit den Trainingssätzen und der Klassendatei kann nun die Transformationsmatrix bestimmt werden. Dies geschieht in drei Schritten, die jeweils durch ein Programm realisiert sind.

1. **meanval** Die relative Häufigkeit der Klassen und ihre Mittelwerte werden bestimmt.
2. **scatter** Mit Hilfe der Mittelwerte werden Total-Scattermatrix, die Within- und Between-Class-Scattermatrix ermittelt.
3. **simdiag** Mit zwei der Matrizen (z.B. T und W) wird eine simultane Diagonalisierung durchgeführt, deren Ergebnis schließlich die Transformationsmatrix A ist.

Die Aufteilung in drei Programme erscheint zunächst nicht zwingend, doch ist sie einfacher und bietet außerdem noch einige Vorteile. Zunächst einmal beachte man, daß die Datenmenge sehr groß ist. Die 2830 Trainingssätze, die für die Versuche benutzt wurden, enthalten beinahe 1 Million Datenvektoren mit je 16 Koeffizienten. Deshalb können nicht alle Daten gleichzeitig in den Speicher eingelesen und verarbeitet werden. Die Trainingsmenge muß also in Partitionen abgearbeitet werden.

Die Datei mit den Mittelwerten enthält auch die Häufigkeit der einzelnen Klassen. Das eröffnet die Möglichkeit bei Bestimmung des Within-Class-Scatters die Klassen gezielt zu gewichten. Durch gezielte Gewichtung der einzelnen Klassenscatter kann zum Beispiel der Einfluß der Klasse *Silence* auf den Gesamt-Within-Class-Scatter verringert werden. Diese einzelne Klasse neben den verschiedenen Phonemen wird denjenigen Datenvektoren zugeordnet, die keinen sprachlichen Laut darstellen, also Pausen zwischen Wörtern oder etwaigen Störgeräuschen. Sie machen einen großen Teil der Datenvektoren aus. Ihr Anteil betrug bei den verwendeten Trainingsdaten 7,65 % gegenüber 1,92 % im Mittel für die Phoneme. Da sie für die Erkennung weniger wichtig sind, sollte ihr Einfluß auf die Matrix verringert werden.

Mit dem Programm zur simultanen Diagonalisierung **simdiag** kann neben einer Linearen Diskriminanzanalyse (LDA) auch die Transformation auf die Hauptachsen (HAT) ermittelt werden. Als Eingabe benötigt es immer zwei Matrizen. Sind dies die Total- und die Within-Class-Scattermatrix wird eine LDA nach dem Gütekriterium $J_1(m)$ in Formel (3.6) durchgeführt. Gibt man statt der Within-Class-Scattermatrix eine Einheitsmatrix I an, so ist das Ergebnis genau die Hauptachsentransformation. Eine Übersicht der Möglichkeiten gibt folgende Tabelle:

Methode, Gütekriterium	Matrix 1	Matrix 2
HAT	T	I
LDA, $J_1(m)$	T	W
LDA, $J_2(m)$	B	W
LDA, $J_3(m)$	B	T

Tabelle 4.1: Eingabematrizen für das Programm:
Simultane Diagonalisierung (**simdiag**)

Zusätzliche Transformation

Die Programme zur Mittelwert- und Scattermatrix-Bestimmung sind so realisiert, daß man zusätzlich eine Transformation der eingelesenen Datenvektoren durchführen kann. Dieses bietet die Möglichkeit die LDA- oder HAT-Matrix zu testen, da man ja dann diagonale Scattermatrizen erhält. Außerdem kann man auch die Scattermatrizen der transformierten Testdaten berechnen. Die Total-Scattermatrix der Trainingsdaten, die für die Versuche im nächsten Kapitel verwendet wurden, zeigt Bild 4.2. Die aus 390 Testsätzen gewonnenen Total-Scattermatrizen mit und ohne Transformation sind in Bild 4.3 zu sehen. In diesen Fällen wurde die Dimensionalität nicht reduziert, sondern nur dekorreliert.

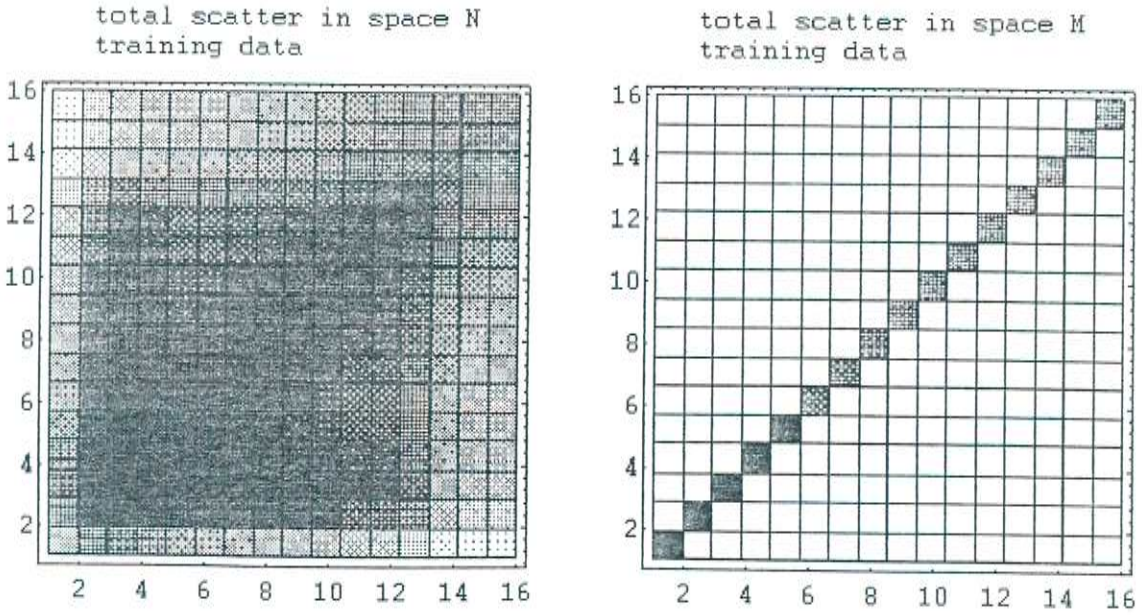


Bild 4.2: Total-Scattermatrix der Trainingsdaten vor und nach der Transformation

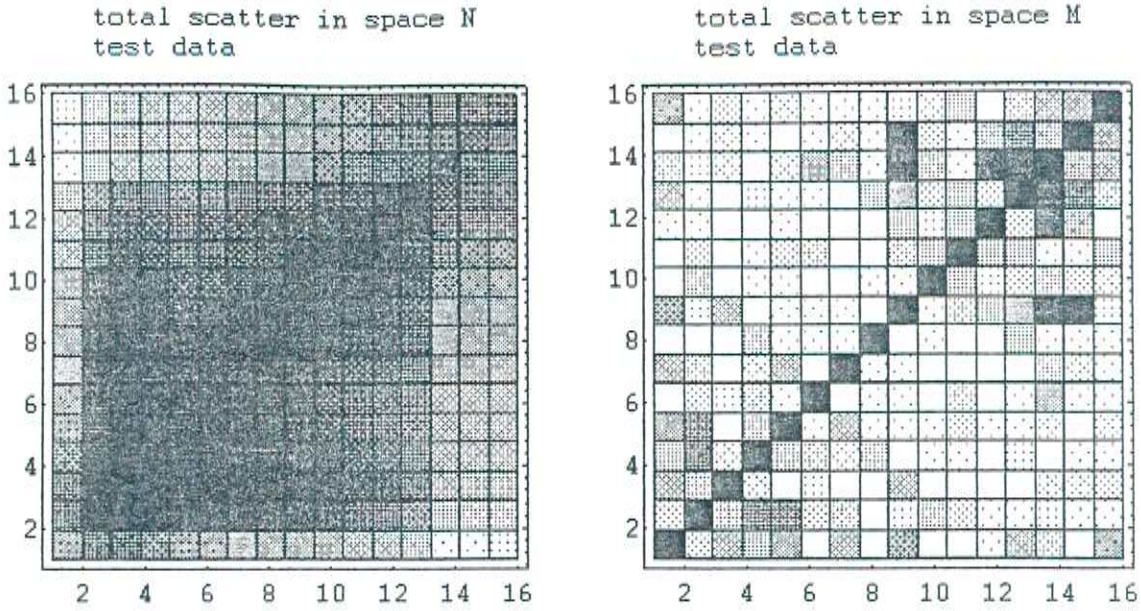


Bild 4.3: Total-Scattermatrix der Testdaten vor und nach der Transformation

Die Matrizen sind als Grauwertbild dargestellt. Betragsmäßig große Werte sind dunkel, kleine hell. Die erste Zeile ist entgegen der gewohnten Darstellung von Matrizen unten. Bei den Matrizen ohne Transformation (links) erkennt man sehr gut die starke Korrelation der benachbarten Koeffizienten. Die Total-Scattermatrix der transformierten Trainingsdaten (Bild 4.2 rechts) ist wie gewollt diagonal. Die Diagonalelemente sind der Größe nach geordnet und nehmen deshalb in der Helligkeit ab. Obwohl bei der entsprechenden Matrix der transformierten Testdaten von Null verschiedene Werte in der Nichtdiagonalen auftauchen, ist die Dekorrelation auch hier annähernd gelungen. Die Grauabstufung bei diesem letzten Bild ist etwas anders gewählt, damit man die sehr kleinen Werte der Nichtdiagonalen überhaupt sieht. Dadurch erscheinen auch alle Diagonalwerte gleich, obwohl sie in Wirklichkeit fast mit denen der Trainingsdaten in Bild 4.2 rechts übereinstimmen. Damit ist gezeigt, daß die mit den Trainingsdaten gewonnene Transformationsmatrix ihren Zweck auch für die Testdaten erfüllt.

Zusammenfassung: Transformationsmatrix

Hier noch einmal die zwei Schritte zur Erlangung der Transformationsmatrix:

- mit vorhandener Gewichtsdatei eine Trainingsiteration durchführen und damit Klassenzuordnung finden,
- aus Trainingssätzen und Klassenzuordnung die Mittelwerte, Scattermatrizen und schließlich die Transformationsmatrix bestimmen.

4.2. Merkmalsextraktionsstufe für den Spracherkenner

Um im Janus-Spracherkenner eine Dimensionalitätsreduktion mit den in Kapitel 3 vorgestellten Verfahren durchführen zu können, wurde er in dieser Diplomarbeit um eine Merkmalsextraktionsstufe mit vorgeschalteter Mustergenerierung, wie in Bild 4.4 zu sehen, erweitert.

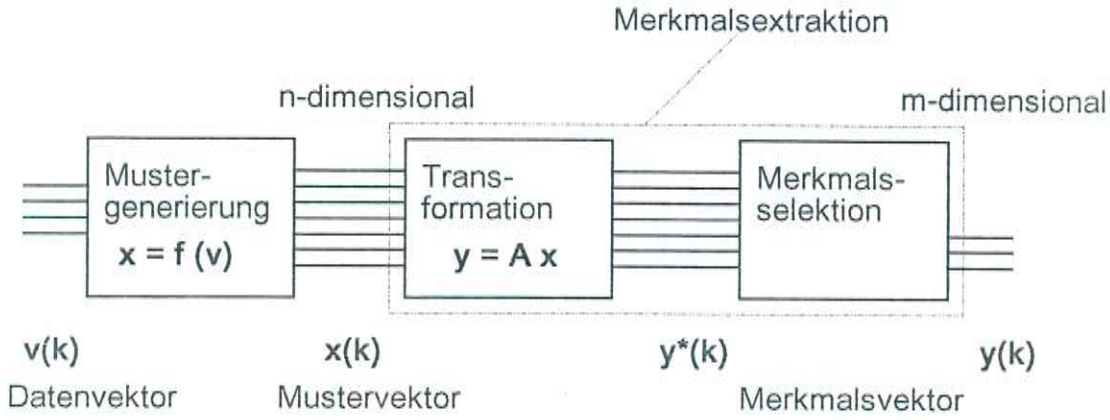


Bild 4.4: Merkmalsextraktionsstufe mit Mustergenerierung

In der Merkmalsextraktion wird der Mustervektor x(k) zum Zeitpunkt k mit der Matrix A transformiert und anschließend die ersten m Koeffizienten selektiert. Verwendet wird also eine n x n Matrix, die aus allen Eigenvektoren (siehe Formeln (3.4) und (3.10)) besteht. Die Anzahl m der Koeffizienten des Merkmalsvektors y(k) kann deshalb beliebig gewählt werden.

Die Mustergenerierung kann den Datenvektor v(k) um andere Koeffizienten erweitern. Zum Beispiel können Delta- und Deltadelta-Koeffizienten hinzugefügt werden:

$$\mathbf{x}(k) := \begin{bmatrix} \mathbf{v}(k) \\ \mathbf{v}'(k) \\ \mathbf{v}''(k) \end{bmatrix} = \begin{bmatrix} \mathbf{v}(k) \\ \mathbf{v}(k+2) - \mathbf{v}(k-2) \\ \mathbf{v}(k+4) - 2\mathbf{v}(k) + \mathbf{v}(k-4) \end{bmatrix} \tag{4.1}$$

Sie sind eine Näherung der ersten und zweiten Ableitung nach der Zeit. Durch diese zusätzliche Information über die Änderung des Datenvektors lassen sich bessere Erkennungsraten erzielen.

Da der so aufgebaute Mustervektor x(k) sich in seiner Dimensionalität durch eine einfache Matrixmultiplikatin reduzieren läßt, wodurch sich der Aufwand für den Klassifizierer ebenfalls reduziert, spricht nichts dagegen, sehr viele Komponenten zu verwenden. Dies können neben den bereits erwähnten Delta-Koeffizienten

- Energiekoeffizient und Ableitungen (*power* und *delta-power*),
- benachbarte Datenvektoren v(k ± i) und deren Ableitungen v´(k ± i) oder
- andere aus der Spracherkennung bekannte Merkmale sein.

Die Mustergenerierung muß natürlich auch bei der Bestimmung der Transformationsmatrix durchgeführt werden. Die selben Routinen, die der Spracherkenner zur Mustergenerierung verwendet, finden sich im Programmteil **makepat** (siehe S. 30) zur Erzeugung der Transformationsmatrix.

4.3. Das Training des Spracherkenners

Bei den ersten Versuchen dieser Diplomarbeit hat sich herausgestellt, daß man nicht einfach die Datenvektoren beliebig in einen neuen Merkmalsraum transformieren kann, denn der Klassifikator ist mit seinen Parametern für die bisherige Datenstruktur optimiert worden. Bei Berechnung der Strafen sind die Distanzen zu den Prototypen der Klassen gegenüber der logarithmierten Häufigkeit mit einem Skalierungsfaktor h gewichtet. Formel (2.6) aus Kapitel 2 sei hier nocheinmal angeben:

$$Strafe(y|s) = -\log(\mathrm{P}(c|s)) + h \cdot |y - m_c|^2 \qquad (2.6)$$

Der Faktor h wurde experimentell so bestimmt, daß sich eine hohe Erkennugsrate ergibt. Es stellt sich heraus, daß beide Summanden im Mittel etwa gleich groß sind

Bei der Dimensionsreduktion mit der Hauptachsentransformation (HAT) handelt es sich um eine Drehung des Koordinatensystems und anschließender Selektion von m Koeffizienten. Die Größenordnung der Koeffizienten bleibt erhalten, nur ihre Anzahl wird von n auf m verringert, was zu einer Verkleinerung des mittleren Distanzquadrates führt. Das dadurch gestörte Verhältnis der Summanden kann durch Ersetzen von h mit

$$\mathrm{h}^* = \mathrm{h} \cdot \frac{\mathrm{n}^2}{\mathrm{m}^2} \qquad (4.2)$$

wieder in den ursprünglichen Zustand gebracht werden.

Bei der LDA-Transformation ist es nicht so einfach. Die neuen Basisvektoren sind nicht orthogonal und nicht normiert. Nach einigen Versuchen habe ich mich für nachfolgende Lösung entschieden. Sie ergibt einen eindeutigen Faktor, mit dem das sehr zeitaufwendige Training durchgeführt werden kann. Bevor man zum Erkennen übergeht wird der Faktor mittels einiger Testläufe optimiert.

- Nach dem Initialisieren der Codebücher und Häufigkeitsverteilungen (siehe S. 17f) mit Verwendung der Dimensionalitätsreduktion wird eine erste Trainingsiteration durchgeführt. Die berechneten Distanzquadrate werden gemittelt und mit dem Erfahrungswert verglichen. Auf diese Weise kann ein neuer Skalierungsfaktor h^* bestimmt werden, der dazu führt, daß die Distanzquadrate im Mittel die bisherige Größenordnung beibehalten.

Bei anschließenden Tests hat sich herausgestellt, daß eine Vergrößerung des Faktors für den Test bessere Erkennungsergebnisse liefert. Eine Vergrößerung des Faktors schon während des Trainings ergab keinen Unterschied der Erkennungsleistung, wenn danach wieder auf den optimalen Testfaktor eingestellt wurde. Das Training ist also sehr robust, was die Änderung des Faktors angeht und dieser kann deshalb auf die oben angeführte, einfache Weise bestimmt werden. Die Grafik (Bild 4.5) zeigt die Abhängigkeit der Wortfehlerrate vom Faktor im Test für verschiedene Faktoren wärend des Trainings. Bei h=13 handelt es sich um den erwähnten, einfach zu bestimmenden Skalierungsfaktor. Das Minimum der Wortfehlerrate über den Testfaktoren ist relativ flach, so daß der optimale Testfaktor nur grob bestimmt werden muß. Dies bevorteilt die hier verwendete Dimensionalitätsreduktion mit LDA gegenüber dem Basissystem ohne LDA, das empfindlicher auf Änderungen des Skalierungsfaktors ist.

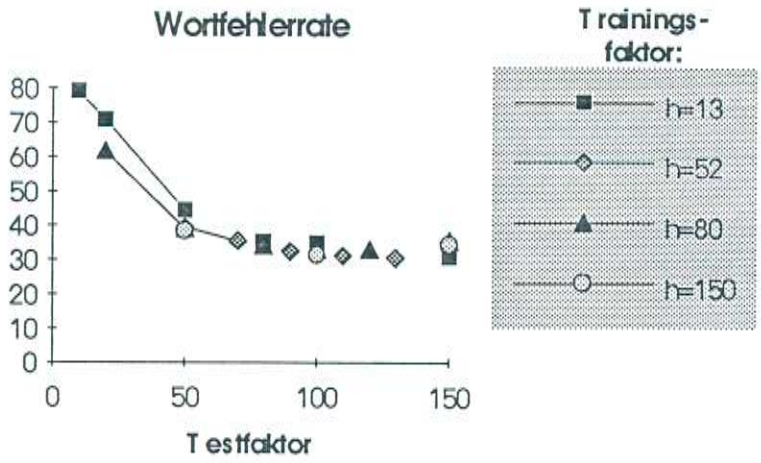


Bild 4.5: Fehlerrate mit verschiedenen Test- und Trainingsfaktoren

Zusammenfassend seien nun noch einmal die einzelnen Schritte beim Trainieren mit einer LDA-Matrix aufgeführt:

- Initialisierung mit ca. 300 handsegmentierten Trainingssätzen
- Bestimmen eines Skalierungsfaktors h für die Distanzen im Merkmalsraum durch Vergleich mit einem Erfahrungswert
- Mehrere Trainingsiterationen, eventuell Abbruch mit Cross Validation[1]
- Tests mit anderen Skalierungsfaktoren bis optimaler "Testfaktor" gefunden

Bei einer Erweiterung auf kontextabhängige Modelle sind folgende, zusätzlichen Schritte durchzuführen:

- Ein kontextunabhängig trainierter Parametersatz auf die neuen Phonemmodelle anpassen, indem die Häufigkeitsverteilungen der Monophone durch die zugehörigen Triphone übernommen werden
- Eventuell nochmaliges Bestimmen eines Trainingsfaktors
- Trainieren der kontextabhängigen Modelle
- Tests wie oben

Trotz leisungsfähiger Workstations (HP APOLLO, DEC 5000, DEC ALPHA) bewegt sich der Zeitbedarf für eine Versuchsreihe in der Größenordnung von ein paar Tagen. Nachdem die Vorgehensweise beim Training mit Dimensionalitätsreduktion durch lineare Abbildungen feststand, konnten Versuche mit einer bekannten Datenbasis durchgeführt werden, um den Einfluß verschiedener Parameter, wie Anzahl der Koeffizienten und Aufbau des Mustervektors, zu testen.

---

[1] *Cross Validation*: Zwischen den Iterationen testet man die Erkennungsleistung mit einer speziellen Testdatenmenge (Validationsdaten). Beginnt die Erkennungsleistung auf den Testdaten durch zuviele Trainingsiterationen wieder abzusinken, beendet man das Training, um sich nicht zu stark auf die Trainigsdatenmenge zu spezialisieren.

5. Testergebnisse der statistischen Methoden

Die Motivation und die Theorie zur Dimensionalitätsreduktion in der Spracherkennung sind in den vorigen Kapiteln dargelegt worden. Ebenso das JANUS-Spracherkennungssystem, auf das die statistischen Methoden angewandt wurden. Wie wir gesehen haben sind die Variationsmöglichkeiten vielseitig, die Versuche aber sehr (rechen)zeitintensiv. Ausgehend von den mir zur Verfügung gestellten Basissystemen habe ich Schritt für Schritt die zuvor erörterten statistischen Methoden angewandt und jeweils mit den Leistungsdaten der Basissysteme verglichen. Angefangen mit einer Vorstellung der verwendeten Sprachdatenbasis und den Erkennungsergebnissen der drei in Kapitel 2 behandelten Basissystemen folgt eine Reihe von Experimententen, die eine Antwort auf die folgenden Fragen suchen:

- *Auf wieviele Dimensionen kann reduziert werden, ohne Erkennungsleistung einzubüßen?*
- *Welches ist die beste Klassendefinition für die LDA?*

Zu Beginn aber, wird erst einmal auf die Definition der hier verwendeten Wortfehlerrate eingegangen.

5.1. Wortfehlerrate

Angenommen, ein Satz besteht aus n Wörtern. Ein Spracherkennungssystem, welches Einzelwörter erkennen soll, hat nur die beiden Möglichkeiten, falsch oder richtig zu klassifizieren. Ein Erkenner für kontinuierliche Sprache kann zusätzlich zu diesem Verwechseln noch fälschlicherweise Wörter einfügen oder wiederum andere ganz übergehen. Diese drei verschiedenen Fälle nennt man Substitution (*engl. substitution*), Einfügen (*engl. insertion*) und Auslassen (*engl. deletion*). Die Gesamtanzahl all dieser Fehler (*engl. error*) ist

$$\text{errors} = \text{substitutions} + \text{deletions} + \text{insertions} \quad (5.1)$$

Die Wortfehlerrate WER (*engl. word error rate*) ergibt sich aus dieser Anzahl, bezogen auf die Anzahl n der Wörter im richtigen Satz

$$\text{WER} = \frac{\text{errors}}{n} = \frac{\text{substitutions} + \text{deletions} + \text{insertions}}{n} \quad (5.2)$$

und wird in Prozent angegeben. Sie kann theoretisch durchaus 100% übersteigen, da sich die Gesamtzahl n der Wörter im richtigen Satz als

$$n = correct\_words + substitutions + deletions \tag{5.3}$$

ergibt. Übersteigt also die Anzahl der *insertions* die Anzahl der richtigen Wörter (*engl. correct words*) tritt genau dies ein.

Die Worterkennungsrate WA (*engl. word accuracy)*, nicht WER!, eines Einzelworterkenners

$$WA_{single} = \frac{correct\_words}{total\_words} \tag{5.4}$$

ändert sich beim kontinuierlichen Erkenner zu

$$\begin{aligned} WA_{continuous} &= 100\% - WER \\ &= \frac{correct\_words - insertions}{n} \end{aligned} \tag{5.5}$$

5.2. Basissysteme und Sprachdaten

In aller Kürze werden die vorigen Kapitel noch einmal zusammengefaßt und die Datenbasis mit den für die Experimente verwendeten Sprachdaten vorgestellt. Die erzielten Wortfehlerraten mit den drei verwendeten Versionen des JANUS-Sracherkenners ohne Dimensionalitätsreduktion sind ebenfalls angegeben.

Der zugrunde liegende Spracherkenner des JANUS-Systems, im folgenden Basissystem genannt, verwendet 16 Melscale-Koeffizienten (MSC) und je nach Version 16 Delta-Melscale-Koeffizienten, die über eine Vorverarbeitung aus dem Sprachsignal gewonnen werden. Die Koeffizienten bilden jeweils einen 16 dimensionalen Vektor. Diese werden mit einem Sprachmodell verglichen, um so Strafen zu berechnen. Eine Suche findet die Wortfolge mit der kleinsten Gesamtstrafe und gibt diese als Hypothese aus. Bei Verwendung von MSC und Delta-MSC werden zwei Datenvektoren mit den jeweiligen Koeffizienten getrennt verarbeitet und die erhaltenen Ergebnisse kombiniert.

Die Sprachdaten für die folgenden Experimente stammen aus der häufig verwendeten *Resource Management* Datenbasis die in [Pri 88] beschrieben ist. Da sie auf vielen Spracherkennungs-systemen verwendet wird, ist dadurch ein Vergleich dieser Systeme möglich.

Zum Training des JANUS Spracherkenners wurden 2830 Trainingssätze von 78 männlichen Sprechern benutzt. Als Testdaten dienen 390 Testsätze von 13 Sprechern. Die Perplexität (siehe Seite 11) der Testsätze mit der verwendeten Wortpaar-Grammatik beträgt 60. Da aus praktischen (Rechenzeit-)Gründen nicht für jedes Experiment alle Testsätze verwendet werden

konnten, sind jeweils drei Sätze von jedem Sprecher zu einem Testset von 39 Sätzen zusammengefaßt. Um den Aufwand im Rahmen zu halten, wurde zum Teil mit einem Set eine Voruntersuchung durchgeführt. Die unten angegebenen Wortfehlerraten sind aber stets über 4 Sets gemittelt, so daß sie das Testergebnis von 156 Sätzen darstellen. Die drei in Kapitel 3 vorgestellten Systeme wurden alle über 10 Iterationen hinweg mit kontextunabhängigen Phonemmodellen trainiert. Von diesen 10 Iterationen dienen allerdings nur 8 der Verbesserung des Parametersatzes, da die ersten beiden "Iterationen" nur dem Initialisieren dienen. Das Basissystem III durchläuft noch 10 weitere Iterationen, um den Parametersatz auf kontextabhängige Triphone anzupassen. Sicherlich lassen sich durch weitere Iterationen, eventuell mit *Cross Validation*, noch kleine Verbesserungen der Erkennungsrate erreichen. Davon wurde jedoch für folgende Tests Abstand genommen. Erstens aus Aufwandsgründen und zweitens, um einheitliche Prüfbedingungen zu schaffen. Die angegebenen Wortfehlerraten der drei Basissysteme sind Werte, die mit denen anderer durchgeführten Versuche auf der gleichen Datenbasis übereinstimmen. Bei den Verbesserungen, die wie weiter unten beschrieben durch die LDA-Transformation erreicht werden, handelt es sich hingegen um Mindestverbesserungen, die sich durch gezielte Optimierung vielleicht noch steigern lassen. Im Rahmen dieser Diplomarbeit sollte es hauptsächlich darum gehen, die Dimensionalitäts-reduktionsverfahren auf dem Spracherkenner zu implementieren und zu untersuchen, welche Auswirkungen die verschiedenen Parameter haben.

Die Fehlerraten, die sich für die drei Basissysteme ergeben, zeigt folgende Tabelle:

	Basissystem		
	I nur 16 MSC	**II** 16 MSC und 16 Delta-MSC	**III** wie II, Triphone
Wortfehlerrate in %	**35,3**	**23,4**	**13,6**

Tabelle 5.1: Wortfehlerraten der Basissysteme

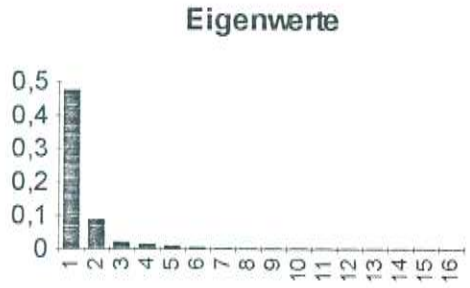
Die zusätzliche Verwendung von Delta-MSC bewirkt eine beachtliche Verringerung der Fehlerrate. Speicherbedarf für den Parametersatz und Rechenzeit sind doppelt so groß. Das System besteht praktisch aus zwei identischen Modulen, die jeweils 16 Koeffizienten bearbeiten. Außerdem muß ein geeigneter Faktor zur Kombination der beiden Ergebnisse experimentell bestimmt werden. Beim System III macht sich die Unterscheidung kontextabhängiger Triphone bezahlt. Abwohl der Speicherbedarf beträchlich steigt, bleibt die Rechenzeit gegenüber System II gleich.

5.3. Dimensionalität des Merkmalsvektors

Dimensionalitätsreduktion mit der Hauptachsentransformation (HAT):

Durch eine Hauptachsentransformation dekorreliert man die Koeffizienten der Trainingsdaten. Dabei sind die Koeffizienten nach fallenden Varianzen sortiert, die hier den Eigenvektoren der Kovarianzmatrix entsprechen. Eine Dimensionalitätsreduktion erreicht man durch Selektion der ersten m Koeffizienten nach der Transformation. Zur Hauptachsentransformation ist keine Klasseninformation über die Trainingsdaten nötig.

Bild 5.1 zeigt die errechneten Eigenwerte der Kovarianzmatrix für die Trainingsdatenmenge. Auffällig ist das starke Abfallen der Werte. Dies bedeutet, daß man die Daten auch noch mit wenigen Koeffizienten ohne großen Fehler darstellen kann. Der in (3.5) definierte Datenreduktionsfaktor nimmt dementsprechend zunächst sehr langsam ab, bis er bei zu kleiner Dimensionalität rapide abfällt.



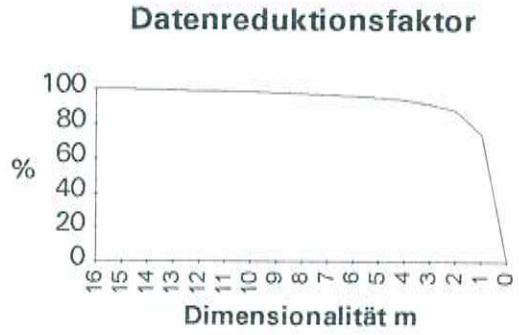


Bild 5.1: Eigenwerte der Kovarianzmatrix der Trainingsdatenmenge

Bild 5.1: Eigenwerte der Kovarianzmatrix der Trainingsdatenmenge

Bei so wenigen markanten Eigenwerten lohnt sich ein Blick auf die zugehörigen Eigenvektoren.

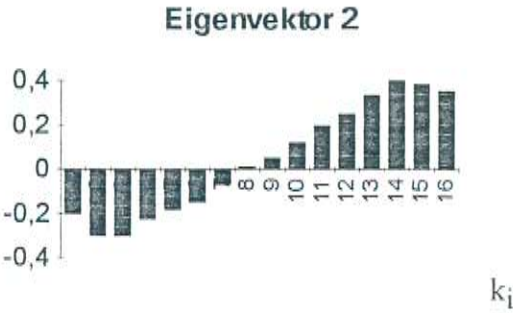


Bild 5.3: Koeffizienten k_i der Eigenvektoren mit den zwei größten Eigenwerten

Sie sind die Basisvektoren des Merkmalsraumes. Zur Errechnung des ersten Koeffizienten eines Merkmalsvektors wird der Datenvektor x mit dem ersten Basisvektor multipliziert. In Bild 5.3 links sieht man, daß dessen Koeffizienten bis auf die "Ränder" in etwa gleich groß sind. Was also hierbei als erster Koeffizient des Merkmalsvektors errechnet wird, entspricht in etwa dem Energiekoeffizienten (englisch: *power*, vergleiche S. 33):

$$\text{power} = \sum_{i=1}^{n} x_i \quad \text{mit} \quad x = (x_1, \; x_2, \; \ldots, \; x_n)^T. \tag{5.6}$$

Mit dem Eigenvektor 2 (Bild 5.3, rechts) wird eine Differenz zwischen hohen und tiefen Frequenzanteilen gebildet.

Obwohl der Datenreduktionsfaktor bis hin zu einer kleinen Dimensionalität nahezu konstant bleibt, kann man dieses Verhalten nicht vollständig auf die Fehlerrate des Spracherkenners übertragen. Die im Bild 5.4 für vier Testsets in Abhängigkeit von der Dimensionalität des Merkmalsvektors aufgetragene Wortfehlerrate ist auch in Tabelle 5.2, gemittelt über die vier Sets, zu finden.

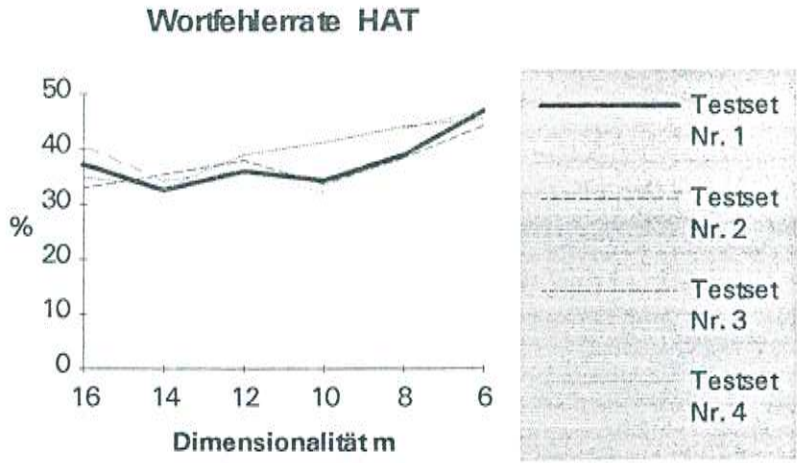


Bild 5.4: Wortfehlerrate der HAT für 4 Testsets

	Basissystem I	Hauptachsentransformation (HAT)					
Dimensionalität m	16	16	14	12	10	8	6
Durchschnitt Wortfehlerrate in %	**35,3**	**36,4**	**33,8**	**37,2**	**35,3**	**41,2**	**45,7**
Verbesserung gegenüber Basissystem		-3%	4%	-5%	0%	-17%	-29%

Tabelle 5.2: Wortfehlerrate und Verbesserung gegenüber des Basissystems I durch die HAT-Transformation.

Bei einer Reduktion auf 8 Dimensionen ist ein merkliches Ansteigen der Fehlerrate zu verzeichnen. Dennoch ist es sehr erfreulich, daß man mit einer Reduktion bis auf 10 Dimensionen die gleiche Erkennungsleistung behält. Bei der Transformation auf 16 Dimensionen führt man lediglich eine Drehung auf ein neues orthonormales Basissystem aus. Alle Distanzen im Merkmalsraum bleiben also gleich. Das Ergebnis für diesen Fall stimmt qualitativ mit dem des Basissystems überein. Bei kleinerer Dimension hat man natürlich den offensichtlichen Vorteil eines geringeren Rechenaufwandes und Speicherbedarfs.

Verwenden von Klasseninformation bei der LDA:

Mit einer Transformation durch eine Matrix, die durch die Lineare Diskriminanzanalyse gewonnen wurde, erhält man ebenfalls dekorrelierte Koeffizienten, deren Varianz abnehmend verläuft. In einem Raum niedrigerer Dimensionalität sind die Daten bezüglich ihrer Klassentrennbarkeit linear optimal transformiert. Da man zur Berechnung der Transformationsmatrix auch Informationen über die Klassenzugehörigkeit der Trainingsdaten verwendet, sollten sich beim Test bessere Ergebnisse erreichen lassen.

Wie zuvor bei der Hauptachsentransformation ist die Wortfehlerrate der 4 Testsets in Bild 5.5 und der Durchschnitt davon mit der prozentualen Verbesserung in Tabelle 5.3 zu sehen.

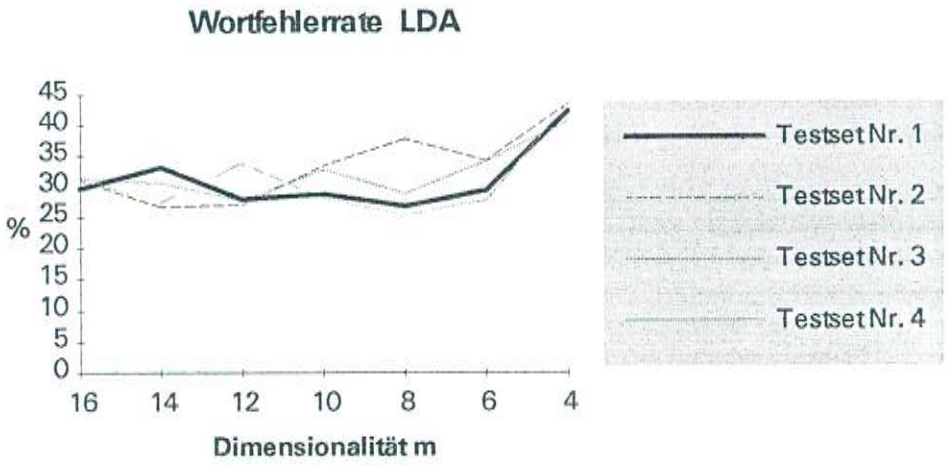


Bild 5.5:Wortfehlerrate der LDA für 4 Testsets

	Basissystem I	**LDA-Transformation**					
Dimensionalität m	16	16	14	12	10	8	6	4
Durchschnitt Wortfehlerrate in %	**35,3**	**30,8**	**29,4**	**29,0**	**30,7**	**29,6**	**31,3**	**42,5**
Verbesserung gegenüber Basissystem		13%	17%	18%	13%	16%	11%	-21%

Tabelle 5.5: Wortfehlerraten und Verbesserungen gegenüber dem Basissystem I durch die LDA-Transformation

Tatsächlich ergeben sich Verbesserungen bis zu 18 %, und dies, obwohl der Aufwand deutlich gesunken ist. Man kann also bei einer Reduktion, beispielsweise auf 6 Dimensionen, mehr als die Hälfte der Zeit für Distanzberechnungen einsparen und verbraucht somit weniger Speicherplatz. Natürlich kommt im Vergleich mit dem Basissystem eine Vektor-Matrix-Multiplikation zur Durchführung der Transformation hinzu. Diese Investition lohnt sich aber, da ein Großteil der für die Erkennung eines Satzes benötigten Zeit mit Distanzberechnungen zugebracht wird. Gerade bei Spracherkennung, bei der man immer Echtzeitverarbeitung anstrebt, ist der Zeitbedarf ein entscheidender Faktor. Wenn dabei noch die Erkennungsrate steigt, ist dies erst recht erfreulich.

Da die LDA der HAT durch die Verwendung von Klasseninformation überlegen ist, wurde im weiteren nur noch die LDA zur Dimensionsreduktion benutzt.

Weitere Verbesserung durch Deltakoeffizienten:

Durch die Verwendung von Delta-MSC wird Information über die zeitliche Änderung der Datenvektoren verfügbar gemacht. Anders als beim Basissystem II kann man bei der Transformation mit der LDA die 16 MSC und 16 Delta-MSC zu einem Vektor zusammenfassen, denn durch die lineare Diskriminanzanalyse werden automatisch die zur Trennung der Klassen geeignetsten Komponenten ausgewählt. Die mittlere Streuung der Klassen wird ebenfalls normiert, genauer gesagt, die mittlere Klassenkovarianzmatrix (*within class scatter*) wird zur Einheitsmatrix. Dadurch werden Variationsunterschiede zwischen den Dimensionen, wie z.B. zwischen MSC und den kleineren Delta-MSC, ausgeglichen.

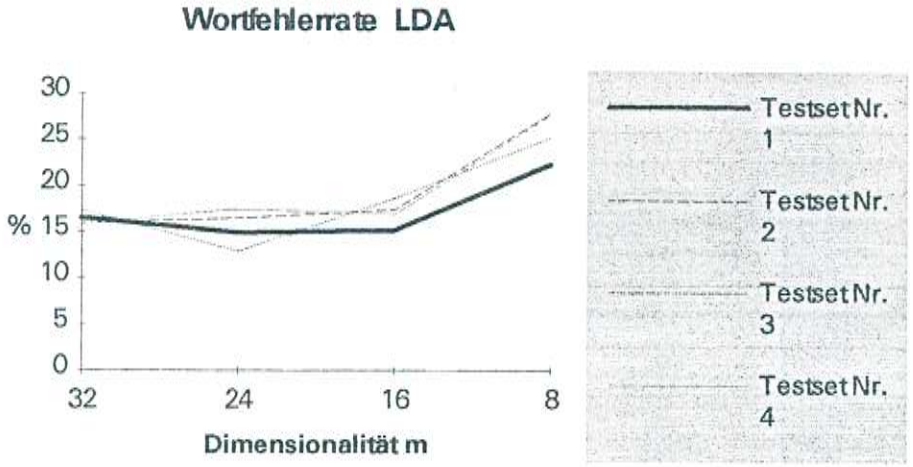


Bild 5.6: Wortfehlerrate der LDA unter Verwendung von Delta-MSC für 4 Testsets

	Basissystem II	**LDA-Transformation mit Delta-MSC**			
Dimensionalität m	32	32	24	16	8
Durchschnitt Wortfehlerrate in %	**23,4**	**16,4**	**15,4**	**17,0**	**25,7**
Verbesserung gegenüber Basissystem II		30%	34%	27%	-10%

Tabelle 5.4: Wortfehlerrate und Verbesserung gegenüber dem Basissystem II durch die LDA unter Verwendung von Delta-MSC

Der Gewinn durch Anwendung der LDA, der gegenüber dem Basissystem II erreicht wurde, ist hier noch deutlicher, als bei der Erkennung ohne Verwendung der Delta-MSC (siehe Bild 5.6 und Tabelle 5.4). Wiederum gelten die oben erörterten Vorteile.

Betrachten wir das "LDA-System" mit der Transformation von 32 auf 16 Koeffizienten und vergleichen es mit dem Basissystem I *ohne Delta-MSC (WER=35,3%). Die in beiden Systemen verwendeten MSC werden im LDA-System um die Deltakoeffizienten erweitert. Gleich nach dieser Erweiterung werden die 32 Koeffizienten durch eine Matrixmultiplikation wieder auf einen 16 dimensionalen Vektor reduziert. Danach sind die Bearbeitungsschritte in beiden Systemen identisch. Verglichen mit den vorangegangenen Berechnungen, angefangen bei den Abtastwerten bis zu den MSC, und erst recht mit den nachfolgenden Distanzberechnungen zur Auffindung des nächsten Codebuchvektors, ist diese Transformation wenig aufwendig. Dennoch führt sie zu einer Reduktion der Fehlerrate um mehr als 50% (siehe auch Bild 5.8)*

Das folgende Bild 5.7 gibt eine exemplarische Übersicht der bisher miteinander verglichenen Systeme bzw. Verfahren und den ermittelten Fehlerraten. Im Falle der "LDA-Systeme" findet jeweils eine Reduktion auf die Hälfte der Dimensionen des Datenvektors statt.

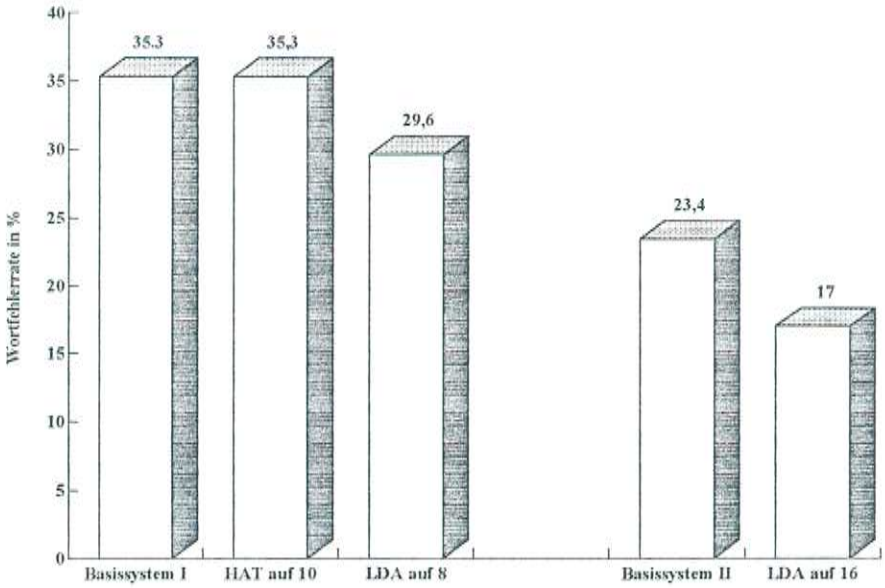


Bild 5.7: Wortfehlerrate der kontextunabhängigen Systeme im Überblick

5.4. Vergleich verschiedener Klassendefinitionen

Da zur Bestimmung der LDA-Transformationsmatrix durch die LDA eine Klassenzuordnung der Datenvektoren vorgenommen werden muß, ist die Frage nach der Definition der Klasse ganz entscheidend. Bei der Spracherkennung hat man die Wahl zwischen einer Reihe von Möglichkeiten, Klassen zu definieren. Bei der Zerlegung der Wörter in Lauteinheiten werden Phoneme unterschieden, im Hidden-Markov-Modell des Klassifizierers Subphoneme oder Zustände und bei der Berechnung der Emmissionswahrscheinlichkeit noch feinere Segmente, die durch die Codebuchvektoren bestimmt sind (siehe Kap. 2). In [Hae 92] werden bei Verwendung von Subphonemen als Klassen die besten Ergebnisse für einen sprecherabhängigen Erkenner erreicht. Für ein kontextabhängiges Phonemmodell wurden in [Hae 93] die Zustände der Triphone mit Erfolg als Klasse gewählt.

Neben den Wortfehlerraten für die Basissysteme I und II, finden sich in Bild 5.8 die ermittelten Werte für die entsprechenden Systeme, die eine LDA-Transformationsmatrix verwenden. Dabei wurden Phoneme, Subphoneme (kurz *sub*) und Subphoneme von Triphonen (kontextabhängige Phoneme) als Klassen verwendet. Obwohl die Sub-Triphone im kontextunabhängigen Erkenner nicht explizit unterschieden werden, ist ein solcher Parametersatz für einen Erkenner erzeugt worden, um auf ihm das Training für ein späteres Experiment mit kontextabhängigen Modellen aufzubauen. Da sich der Merkmalsraum nicht während des Trainings ändern sollte, wurde daher in diesem Fall von Anfang an die LDA-Transformationsmatrix für Sub-Triphone verwendet.

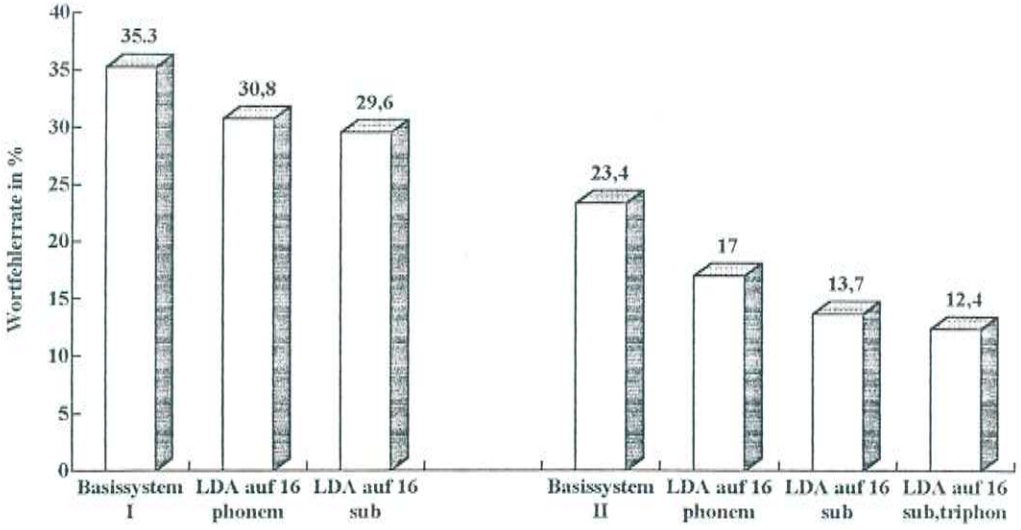


Bild 5.7: Wortfehlerrate für verschiedene Klassendefinitionen

Werden nur MSC verwendet (links im Bild), so spielt es anscheinend kaum eine Rolle, ob Phoneme oder Subphone als Klassen für die LDA genommen werden. Beide Ergebnisse sind zwar besser als das Basissystem I, *unterscheiden sich aber nicht sehr stark. Anders ist es bei Verwendung von Delta-MSC. Hier machen sich unterschiedliche Klassendefinitionen in der Fehlerrate bemerkbar. Mit der LDA wurde auf 16 Dimensionen reduziert, während das Basissystem* II *mit 32-dimensionalen Datenvektoren arbeitet (rechts im Bild). Die WER von 17% bei Unterscheidung von Phonemen sind weiter oben auch schon im Zusammenhang mit verschiedenen Dimensionen aufgeführt worden und stellen eine erhebliche Verbesserung gegenüber dem Basissystem* II *dar. Mit Subphonemen gelangt man allerdings mit der Fehlerrate schon in einen Bereich, wie er vom JANUS-Spracherkenner bisher nur mit kontextabhängigen Phonemmodellen erreicht wurde. Ob tatsächlich Subphoneme von Triphonen als Klassendefinition für einen kontextunabhängigen Erkenner noch besser sind, wie es das ganz rechte Versuchsresultat ergeben hat, läßt sich nicht mit Sicherheit sagen. Zumindest lagen die Fehlerraten für diesen Fall bei allen 4 verwendeten Testsets unter den 13,6% des Basissystems* III. *Gegenüber dem Basissystem II ist die Fehlerrate von 12,4% eine Verbesserung von beachtlichen 47%.*

Der Trainingsaufwand für einen Erkenner mit kontextabhängigen Phonemmodellen, den Triphonen, ist etwa doppelt so groß wie der für kontextunabhängige, weswegen Versuche mit diesem erst gegen Ende der Diplomarbeit unternommen wurden. Die Ergebnisse sind in Bild 5.9 zusammengestellt.

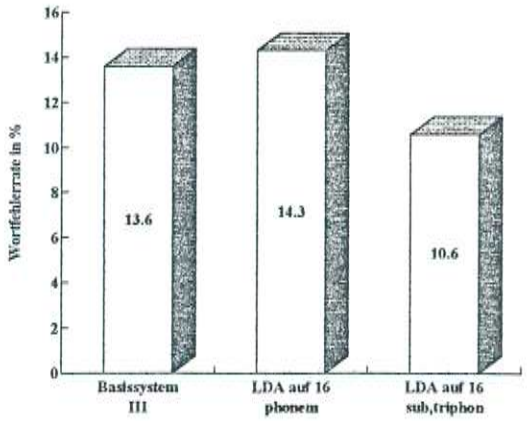


Bild 5.9: Wortfehlerrate der kontextabhängigen Systemen

Diesmal versagt eine Transformation, die mit Monophonen, also kontextunabhängigen Phonemen, erstellt wurde. Sie sorgt dafür, daß die Merkmalsvektoren eines Monophons möglichst dicht beeinander sind. Für den Klassifizierer, der seine Leistungsfähigkeit aus der Aufteilung von Monophonen in Triphone bezieht, ist es dadurch schwieriger geworden, zwischen diesen zu unterscheiden, also steigt die Fehlerrate. Bei Sub-Triphonen als Klassen ergeben sich, trotz Reduktion auf die Hälfte der Koeffizienten, die inzwischen schon gewohnten Verbesserungen durch die LDA. Die relative Änderung zur Fehlerrate des Basissystems III beträgt hierbei 22%.

5.5. Zusammenfassung

Die HAT, bei der im Falle des JANUS-Spracherkenners das Training gegenüber LDA einfacher ist, da kein neuer Skalierungsfaktor h (siehe S.33) gefunden werden muß, eignet sich hervorragend, um die Dimensionalität zu reduzieren.

Dennoch ist die LDA vorzuziehen, denn bei ihr handelt es sich um ein durchweg überzeugendes Verfahren, das zwei normalerweise gegensätzliche Eigenschaften in sich zu vereinen mag. Erstens wird die Leistungsfähigkeit des Systems verbessert, d.h. die Fehlerrate sinkt. Zweitens kann gleichzeitig durch die Reduktion auf weniger Koeffizienten der Speicherbedarf verringert und vor allem die für die Spracherkennung wichtige Geschwindigkeit erhöht werden. Umgekehrt können bei gleichem Aufwand mehr informationstragende Koeffizienten im Mustervektor enthalten sein. Da dieser so reduziert wird, daß die für die Klassifikation wichtige Information erhalten bleibt, kann auf diese Art die Erkennungsleistung weiter gesteigert werden.

Als Abschluß dieses Kapitels seien noch einmal alle vorgestellten Ergebnisse in den Tabellen 5.5 und 5.6 zusammengefaßt.

	16 MSC	16 MSC + 16 Delta-MSC	16 MSC + 16 Delta-MSC (kontextabh.)
Basissysteme	**35,3**	**23,4**	**13,6**
LDA auf 16 Dimensionen Klasse:			
Monophon	30,8 (13%)	17,0 (27%)	14,3 (-5%)
Subphon	29,6 (16%)	13,7 (42%)	-
Sub-Triphon	-	**12,4 (47%)**	**10.6 (22%)**

Tabelle 5.6: Wortfehlerrate in % (Verbesserung) für verschiedene Klassendefinitionen

	m	WER in% (Verbesserung)
Basissystem I	16	**35,3**
HAT	16	36,4 (-3%)
	14	**33,8 (4%)**
	12	37,2 (-5%)
	10	35,3 (0%)
	8	41,2 (-17%)
	6	45,7 (-29%)
LDA	16	30,8 (13%)
	14	29,4 (17%)
	12	**29,0 (18%)**
	10	30,7 (13%)
	8	29,6 (16%)
	6	31,3 (11%)
	4	42,5 (-21%)

	m	WER in% (Verbesserung)
Basissystem II	32	**23,4**
LDA	32	16,4 (30%)
	24	**15,4 (34%)**
	16	17,0 (27%)
	8	25,7 (-10%)

Tabelle 5.5: Wortfehlerraten in % (Verbesserungen) in Abhängigkeit der Dimensionalität

6. Neuronaler Ansatz zur Dimensionalitätsreduktion

6.1. Neuronale Netze

Die Nervenzelle

Das Vorbild der Neuronalen Netze in der Technik stammt, wie so vieles, aus der Natur. Schon immer war es ein Anliegen des Menschen, die Vorgänge im Gehirn besser zu verstehen. Im ersten Kapitel war die Rede vom Kommunikationsproblem zwischen Mensch und Maschine. Würde eine Maschine zumindest teilweise wie das menschliche Gehirn arbeiten, so wäre zu erwarten, daß sie dadurch besser "sehen" und "hören" kann. Gerade in der Spracherkennung liegt also die Verwendung Neuronaler Netze nahe.

Bild 6.1 zeigt die schematische Darstellung einer Nervenzelle (Neuron). Sie besteht aus einem Zelleib und einem Ausläufer, dem Axon. Am Ende des Axons befindet sich die sogenannte Synapse, die an der nächsten Nervenzelle anliegt und so für eine Verbindung der Nervenzellen sorgt. Überschreitet die Anregung durch Synapsen anderer Zellen einen bestimmten Schwellwert, so wird die Zelle aktiv und gibt ihrerseits eine Anregung an die nächste Zelle weiter.

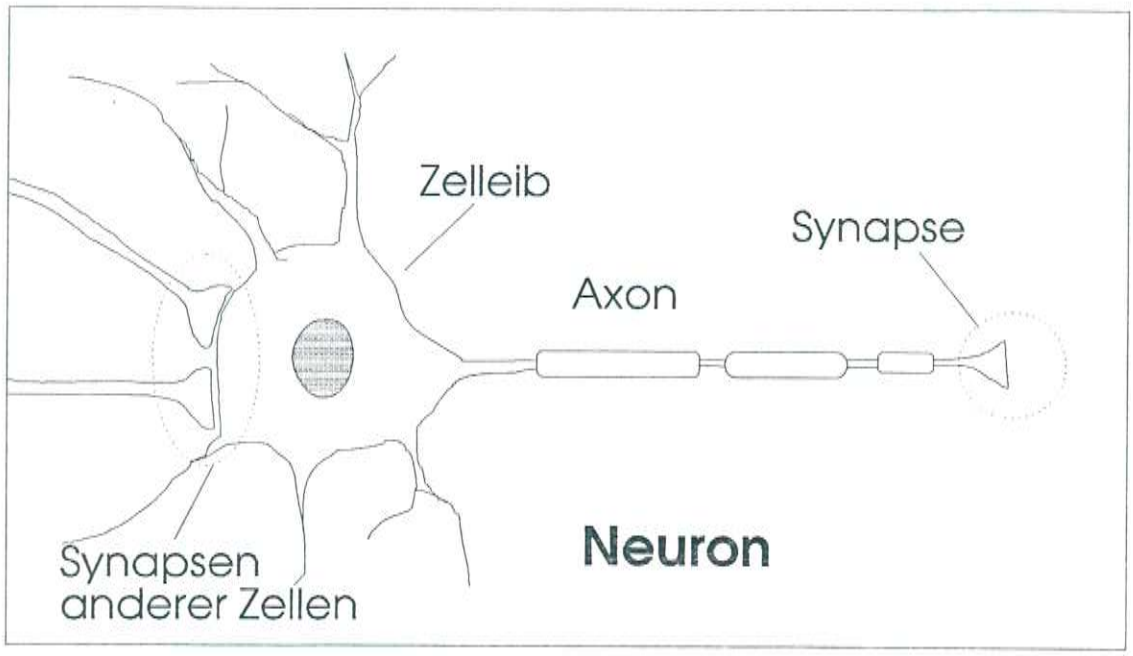


Bild 6.1: Eine Nervenzelle (Neuron)

Die Einheit eines künstlichen Neuronalen Netzes

Eine Darstellung eines künstlichen Neurons, wie es für technische Zwecke verwendet wird, findet man in Bild 6.2 aus [Köh 90]:

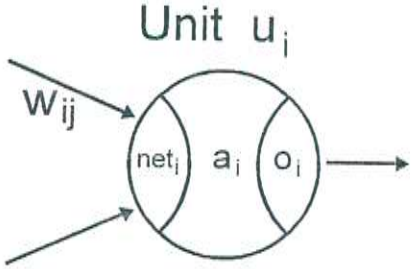


Bild 6.2: Unit u_i (künstliches Neuron)

Diese künstlichen Neuronen werden Einheiten (englisch: *units*) genannt. Eine Einheit u_i bekommt über gewichtete Verbindungen von anderen Einheiten Signale. Die Gewichtung für eine Verbindung von der Einheit u_j zur Einheit u_i wird mit w_{ij} bezeichnet. Aus den ankommenden gewichteten Signalen wird zunächst ein Netto-Input net_i bestimmt:

$$net_i = \sum_j w_{ij} \cdot o_j \qquad (6.1)$$

Daraus ergibt sich über eine weitere Funktion die Aktivierung a_i der Einheit. Häufig wird, wie auch in dieser Arbeit, einfach $a_i = net_i$ verwendet. Ist die Aktivierung groß genug, wird ein entsprechendes Ausgangssignal weitergegeben. Dieses wird mit der sogenannten Ausgabefunktion (englisch: *output function*) realisiert. Ein Beispiel dafür wäre eine binäre Schwellenfunktion Bild 6.3.

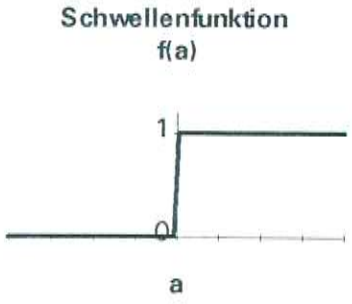


Bild 6.3: Schwellenfunktion

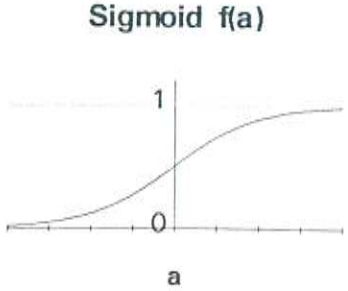


Bild 6.4: Sigmoidfunktion

Sie hat jedoch den Nachteil, daß sie nicht differenzierbar ist, weshalb man häufig eine Sigmoidfunktion verwendet (Bild 6.4 und Formel (6.2)).

Formel der Sigmoidfunktion:

$$o_i = f(a_i) = \frac{1}{1 + e^{-a_i}} \tag{6.2}$$

Lernen im Netz

Ordnet man die Einheiten in Schichten an und verbindet alle Einheiten einer Schicht mit denen der nächsten, so hat man sich ein häufig verwendetes Netz konstruiert, das *Multi-Layer-Perceptron*[1] (MLP). Die im Eingangssignal x zusammengefaßten Eingangswerte werden direkt in die unterste Schicht übernommen und durchlaufen dann die Einheiten der versteckten Schichten (englisch: *hidden units*). Die Ausgaben o_i der Einheiten in der obersten Schicht werden in einem Vektor zusammengefaßt und als Ausgangssignal y ausgegeben (siehe Bild 6.5).

$$y = (o_1, \quad o_2, \quad \ldots, \quad o_m)^T \tag{6.3}$$

Interpretiert man die Aktivierung der Einheiten einer Schicht ebenfalls als Koeffizienten eines Vektors, so entsprechen die Summationen der gewichteten Signale einer Matrixmultiplikation. Würde die Aktivierung direkt an die nächste Einheit weitergeleitet, könnte man die Matrixmultiplikationen jeder Schicht zu einer Gesamtmatrix zusammenfassen. Durch die Ausgabefunktion in den Einheiten kommt aber eine Nichtlinearität hinzu. Das Neuronale Netz führt daher eine nichtlineare Abbildung des Eingangssignals x auf das Ausgangssignal y aus.

[1]Multi-Layer-Perceptron: Mehrschichtiges Netz aus Einheiten (Perceptrons)

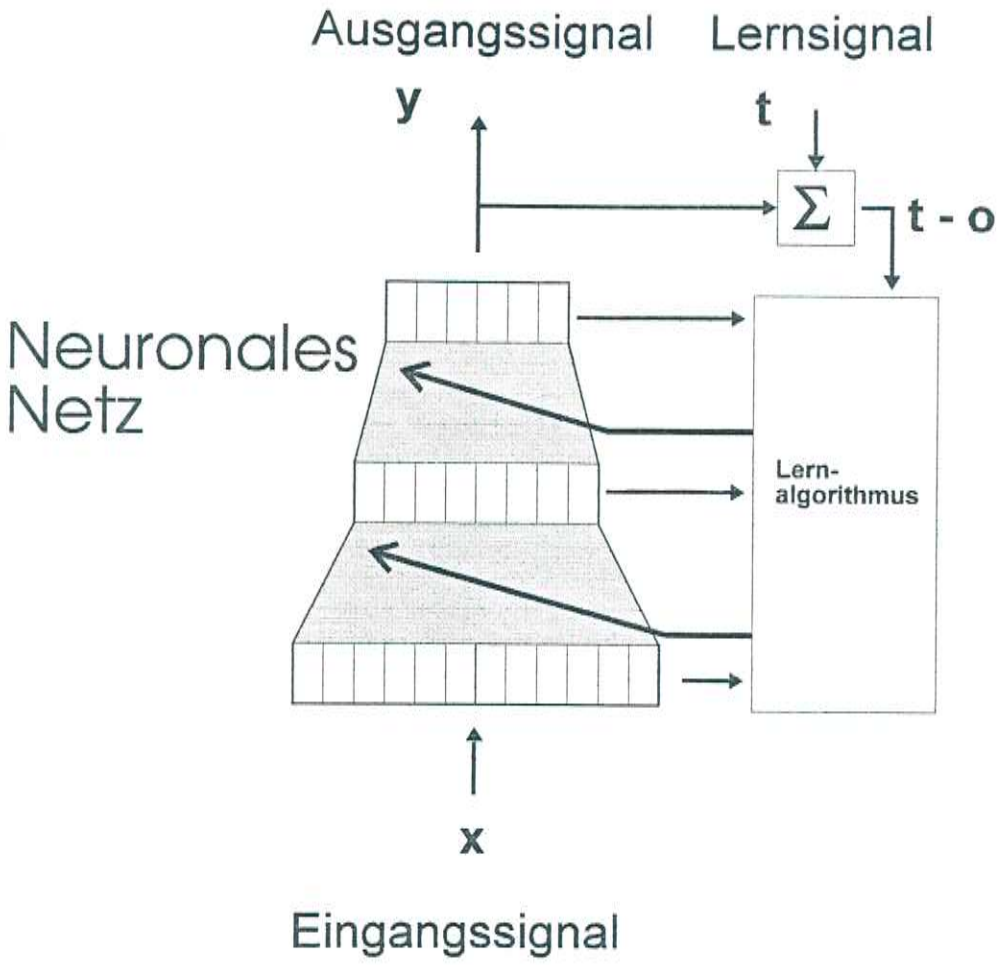


Bild 6.5: Wie ein Neuronales Netz lernt.

Aufgabe des Neuronalen Netzes ist es, zu bestimmten Eingangssignalen die dazu gewünschten Ausgangssignale zu liefern. Ähnlich wie Menschen etwas lernen können, vermag auch das Netz diese gewünschte Abbildung durch ein Training zu erlernen. Dabei werden dem Netz Eingabesignale präsentiert und gleichzeitig ein Ziel in Form eines Lernsignals t vorgegeben. Dieses wird mit dem Ausgangssignal verglichen und dazu benutzt über einen Lernalgorithmus die Gewichte der Verbindungen neu einzustellen. Beim nächsten Durchgang sollte das Ausgangssignal dem Lernziel schon ähnlicher sein. Dieses Training wird solange fortgeführt bis ein Abbruchkriterium erreicht ist. Möglichkeiten dafür sind:

- Das Mittel des quadratischen Fehlers $e^2 = |t - y|^2$ am Ausgang unterschreitet einen gewissen Wert (mit dem Lernsignal t und dem Ausgangssignal y).
- Eine bestimmte Anzahl von Iterationen ist erreicht.

Ein Lernalgorithmus, den man zum Neueinstellen der Gewichte in einem MLP verwendet, heißt *Back Propagation*. Hierbei wird der errechnete Fehler (t -y) von der

Ausgangsschicht auf die darunter liegenden Schichten zurückgerechnet und dort die Gewichte korrigiert. Für die mathematische Herleitung sei auf [Rum 86] verwiesen. Hier eine Angabe der Formeln zur Änderung der Gewichte:

Fehlersignal für eine Einheit u_i der Ausgangsschicht:

$$\delta_i = f'(a_i) \cdot (t_i - o_i) \tag{6.4}$$

Fehlersignal für eine Einheit u_i einer versteckten Schicht:

$$\delta_i = f'(a_i) \cdot \sum_k (\delta_k \cdot w_{ki}) \tag{6.5}$$

Gewichtsänderung:

$$\Delta w_{ij} = \eta \cdot \delta_i \cdot o_j \tag{6.6}$$

Dabei ist η die *Lernrate*, deren Wert für die meisten Anwendungen zwischen 0,1 und 10 liegt. Wird er zu groß gewählt, kann es zu Instabilitäten kommen, bei sehr kleinen Werten sind entsprechend viele Iterationen notwendig.

Nun wird klar, warum die Ausgabefunktion $f(a_j)$ differenzierbar sein muß, denn bei der Bestimmung des Fehlersignals wird ihre Ableitung verwendet. Die Ableitung der Sigmoidfunktion in Formel (6.4) läßt sich im übrigen als sehr einfache Funktion des ohnehin schon ermittelten Ausgangssignals o_i schreiben.

Ableitung der Sigmoidfunktion:

$$f'(a_i) = o_i \cdot (1 - o_i) \tag{6.7}$$

6.2. Diskriminativer Lernansatz

Bisher ist die Frage offen geblieben, was das Netz überhaupt lernen soll. In [Köh 90] sind vier Lernansätze für verschiede Anwendungen aufgeführt:

- Selbstassoziation (Auto Associator)
- Musterassoziation (Pattern Associator)
- Klasseneinteilung (Classification Paradigm)
- Klassenfindung (Regularity Detector)

Bei der *Selbstassoziation* wird als Lernsignal das Eingangssignal selbst verwendet (t = x). Bei einer späteren Anwendung kann das Netz gestörte Eingangssignale, wie das nur bruchstückhaft erhaltene A in Bild 6.6, rekonstruieren.

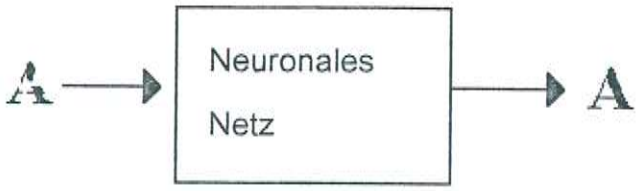


Bild 6.6: Rekonstruktion eines fehlerhaften Eingangssignals

Ein verallgemeinerter Fall dazu ist die *Musterassoziation*, bei der einem Eingangssignal ein zuvor bestimmtes Ausgangssignal als Lernziel zugeordnet wird. Also handelt es sich beim Zielsignal t um eine definierte Zuordnung zum Eingangssignal $t = g(x)$ /2.

Die *Klasseneinteilung* wird, wie der Name schon sagt, zur Klassifikation benutzt. Bei einem Eingangssignal, das zu einer Klasse ω_i gehört, soll eine der Klasse zugeordneten Ausgangseinheit einen maximalen Ausgabewert liefern. Oft wird

$$t = (t_1, \ \ldots, \ t_j, \ \ldots, \ t_m) \text{ mit } t_j = \begin{cases} 1 & \text{für } j = i \\ 0 & \text{sonst} \end{cases} \tag{6.8}$$

als Ziel verwendet.

Dies ist eine Funktion der Klasse des Eingangssignals ($t = g(\omega_i | \ x \in \omega_i)$.

Bei der *Klassenfindung* sind keine Klassen von vornherein bekannt, sondern durch Zusammenfassen ähnlicher Eingangssignale werden die Klassen erst durch das Netz gebildet.

Zusammenfassend betrachtet ergibt sich bei obigen Ansätzen das Lernziel entweder als Funktion des Eingangssignals oder dessen Klasse:

$$t = g(x) \text{ oder } t = g(\omega_i | x \in \omega_i) \tag{6.9}$$

Für ein Netz, das die Diskriminanz (Unterscheidbarkeit) vorgegebener Klassen im Raum der Ausgangsvektoren verbessern soll, passen diese Ansätze nicht. Im nächsten Abschnitt wird deshalb ein in dieser Diplomarbeit entwickelter neuer Ansatz beschrieben, der im Unterschied zu den obigen Ansätzen die Ausgabemuster y selbst

---

[2]Um Verwechslungen mit der Ausgabefunktion f(a) auszuschließen, wird hier g(..) als beliebige, aber feste Funktion verwendet

zur Bestimmung des Lernsignals t verwendet. Außerdem wird die Klasse des Eingangssignals verwendet:

$$t = g(y, \omega_i | x \in \omega_i) \tag{6.10}$$

6.3. Diskriminanzanalyse-Netzwerk (DAN)

Durch ein Rückbesinnen auf die Grundgedanken, die hinter den Gütekriterien der linearen Diskriminanzanalyse stecken, kann man sich einen diskriminativen Ansatz mit Neuronalen Netzen herleiten. Bei dem Gütekriterium $J_2(m)$ in Formel (3.7) wird die Determinante der Between-Class-Scattermatrix maximiert und die Determinante der Within-Class-Scattermatrix minimiert. In anderen Worten:

Die Merkmalsvektoren verschiedener Klassen sollen möglichst auseinander liegen, die Merkmalsvektoren einer Klasse aber gleichzeitig eng beieinander!

Wie soll nun aber ein Ziel bestimmt werden, mit dem das Netz lernen kann. Dazu ist eine Änderung der gewöhnlichen Vorgehensweise beim Lernen mit Neuronalen Netzen notwendig:

- Neu ist ein Durchgang, bei dem alle Eingangsmuster die zum Trainieren des Netzes benutzt werden und von denen bekannt ist, zu welcher Klasse sie gehören, das Netz durchlaufen. Sie werden zunächst nicht verwendet, um das Netz auf ein vorgegebenes Lernsignal zu trainieren. Die Ausgaben werden vielmehr dazu herangezogen, die Mittelwerte m_i der Merkmalsvektoren der einzelnen Klassen ω_i im Ausgangsraum zu bestimmen.
- Im zweiten Durchgang werden die Eingangsmuster dem Netz abermals präsentiert. Diesmal wird das Netz mit dem im ersten Durchgang erzeugten Wissen über die Muster im Merkmalsraum durch ein Standart-*back-propagation*-Verfahren trainiert. Dabei ist eine übliche Vorgehensweise, zunächst alle Gewichtsänderungen aufzusummieren und nachdem alle Muster der Trainingsmenge in das Netz gelangt sind, die Gewichte neu einzustellen.

Wollte man sich zwei Durchläufe ersparen, was theoretisch möglich ist, müßte man sich die Ausgänge aller Einheiten in den Schichten des Netzes für alle Trainingsmuster merken, was bei den großen Trainingsdatenmengen der Spracherkennung zu Speicherplatzproblemen führen könnte.

Nachdem die Mittelwerte m_i im ersten Schritt bestimmt worden sind, können diese bei der Ermittlung des Lernsignals t benutzt werden. Um die Merkmalsvektoren y einer Klasse ω_i näher zusammen zu bringen, soll ihr Ziel in Richtung zum Mittelwert der Klasse liegen (Bild 6.7).

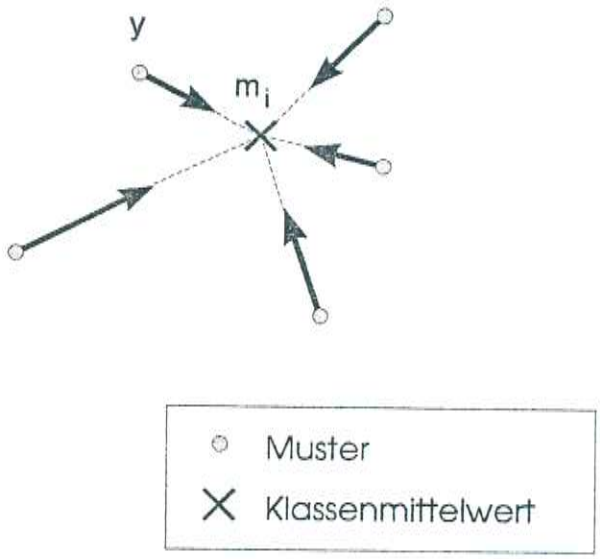


Bild 6.7: Die Merkmalsvektoren einer Klasse sollen näher zusammen.

Dabei ist es sicherlich sinnvoll, daß entferntere Merkmalsvektoren stärker an den Mittelwert gebracht werden als Merkmalsvektoren, die sowieso schon sehr nahe sind. Der Differenzvektor (m_i - y) kann also unmittelbar benutzt werden, um den Lernschritt zu bestimmen.

Ähnlich kann man die Differenzvektoren der Mittelpunkte dazu benutzen, eine Richtung zu bestimmen, in der sich die gesamte Klasse bewegen soll. Diesmal liegt der Fall allerdings umgekehrt. Je näher zwei Klassenmittelpunkte m_i und m_j beieinander sind, desto stärker soll das jeweilige Lernziel für die Merkmalsvektoren dieser Klassen in der entgegengesetzten Richtung liegen. Die Länge des gesuchten Richtungsvektors r_{ij} für die Klasse ω_i muß eine monoton fallende Funktion $b(|\Delta m_{ij}|)$ der Länge $|\Delta m_{ij}|$ des Differenzvektors sein:

$$r_{ij} = b\left(\left|\Delta m_{ij}\right|\right) \cdot \frac{\left(m_i - m_j\right)}{\left|\Delta m_{ij}\right|} \quad \text{wobei} \quad \frac{db\left(\left|\Delta m_{ij}\right|\right)}{d\left|\Delta m_{ij}\right|} < 0 \tag{6.11}$$

Alle Richtungsvektoren werden zu einem gemeinsamen Vektor, den ich Klassendrift d_i genannt habe, aufaddiert.

$$d_i = \sum_j r_{ij} \tag{6.12}$$

Eine Anschauung zur Bestimmung der Klassendrift findet man in Bild 6.8.
Beispielhaft werden dort drei Richtungsvektoren r zur Klassendrift zusammengefaßt.

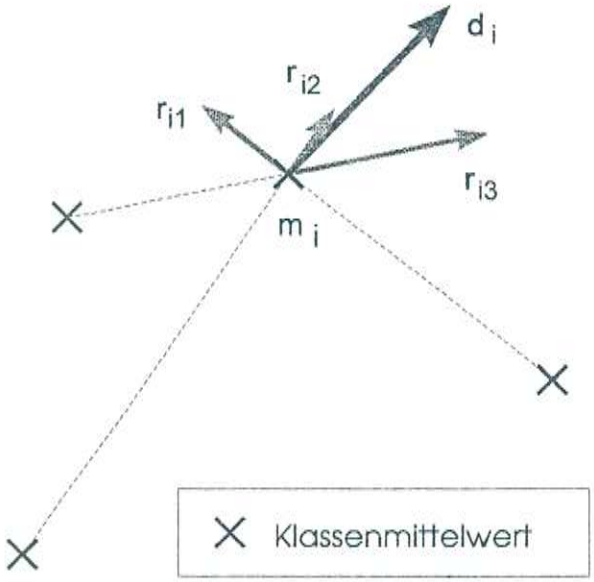


Bild 6.8: Bestimmung der Klassendrift

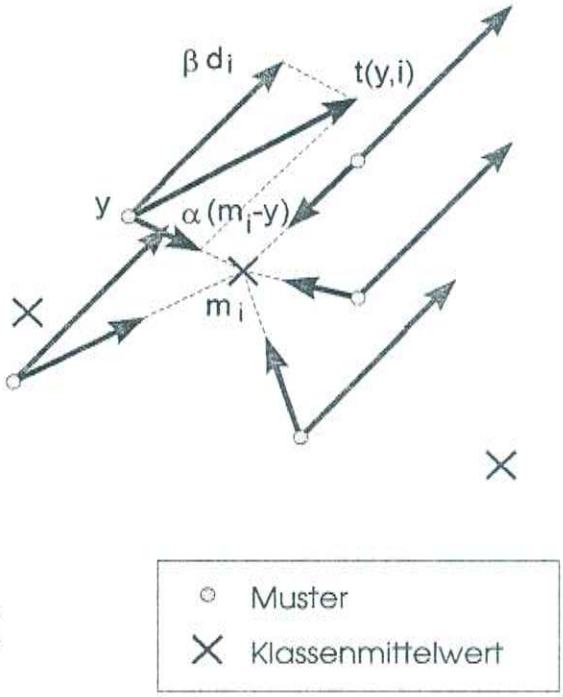


Bild 6.9: Das Ziel für jeden Merkmalsvektor wird durch Addition zweier Komponenten bestimmt.

Nun kann eine Formel zur Berechnung des Lernsignales t für ein dem Netz eingespeißtes Eingangssignal x angegeben werden. Es ist, wie im vorigen Abschnitt erläutert, eine Funktion des Ausgangssignales y und der Klasse ω_i des Eingangssignals (Vergleiche Formel (6.10)):

$$t = y + \alpha \cdot (m_i - y) + \beta \cdot d_i \qquad (6.13)$$

$$\text{mit } y = y(x) \text{ und } x \in \omega_i$$

Dabei ist der mit α gewichtete Teil der Vektor in Richtung des Mittelwerts und d_i die mit β gewichtete Klassendrift. Beide Gewichtungsfaktoren sind experimentell zu bestimmen. Eine Anschauung, wie das Ziel t für einen Ausgangsvektor oder Merkmalsvektor bestimmt wird, gibt Bild 6.9.

Die Funktion $b(|\Delta m_{ij}|)$ zur Festlegung der Länge des Richtungsvektors r_{ij} wurde bisher nicht angegeben, da eine weitere Bedingung zur Auswahl der Funktion $b(|\Delta m_{ij}|)$ beachtet werden sollte, die erst jetzt hergeleitet werden kann.Hierbei geht es nur um die Klassendrift, weshalb der mit α gewichtete Teil in Formel (6.13) in diesem Zusammenhang weggelassen wird.

Die Bedingung lautet:

- Niemals darf das Ziel für einen Mittelwert m_i, der näher zu m_j liegt, weiter weg von m_j sein, als das Ziel eines entfernteren Mittelwerts m_k, wenn nur die Richtungskomponente betrachtet wird, die durch m_j entsteht.

Bei einer Beschränkung auf die in Formel (6.11) gemachte Vorraussetzung für die Funktion $b(|\Delta m_{ij}|)$ könnte es nämlich passieren, daß die Länge eines Richtungsvektors für einen nahe gelegenen Mittelwert so groß wird, daß er das Ziel einer weiter entfernt gelegenen Klasse "überholt" (Bild 6.10).

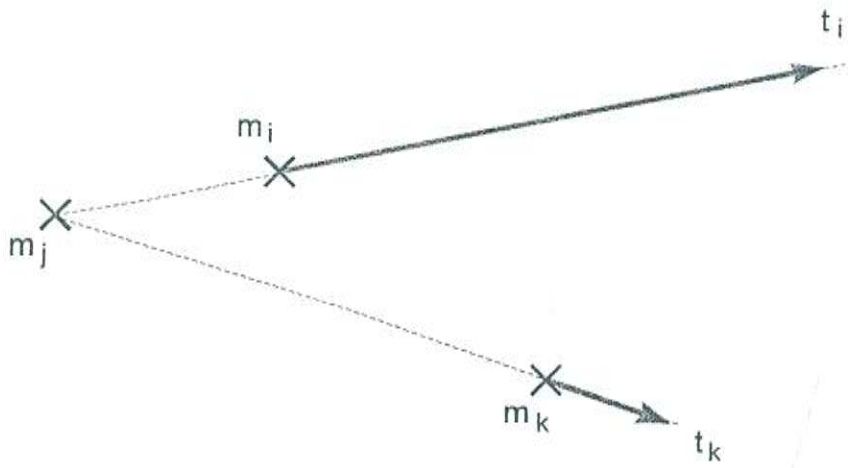


Bild 6.10: Das Ziel eines nahen Vektors liegt weiter weg, als das eines entfernten

Der *worst case* (schlimmste Fall) tritt ein, wenn drei Mittelpunkte auf einer Geraden liegen, so daß das Problem eindimensional angegangen werden kann. Betrachten wir den Zielvektor zweier Klassen mit den Mittelwerten m_i und m_k in Abhängigkeit eines

Mittelwertes m_j einer dritten Klasse. Ohne Beschränkung der Allgemeinheit wird der Mittelwert m_j der dritten Klasse als Nullpunkt angesehen. Als Ziele ergeben sich

$$t_i = |\Delta m_{ij}| + \beta \cdot b(|\Delta m_{ij}|) \tag{6.14}$$

$$t_k = |\Delta m_{kj}| + \beta \cdot b(|\Delta m_{kj}|), \tag{6.15}$$

wobei nun gelten soll, daß

$$t_k > t_i \quad \text{wenn} \quad |\Delta m_{kj}| > |\Delta m_{ij}|. \tag{6.16}$$

Dieses bedeutet aber nichts anderes, als daß

$$t_i(|\Delta m_{ij}|) = |\Delta m_{ij}| + \beta \cdot b(|\Delta m_{ij}|) \text{ monoton steigend ist} \tag{6.17}$$

oder

$$\frac{dt_i(|\Delta m_{ij}|)}{d|\Delta m_{ij}|} > 0. \tag{6.18}$$

Daraus folgt

$$\frac{db(|\Delta m_{ij}|)}{d|\Delta m_{ij}|} > -\frac{1}{\beta}. \tag{6.19}$$

Für $0 < \beta < 1$ sind die Bedingungen (6.11) und (6.19) für folgende einfach zu berechnende Funktion erfüllt:

$$b(|\Delta m_{ij}|) = \frac{1}{1 + |\Delta m_{ij}|}. \tag{6.20}$$

Wird das Netz nun auf diese Weise trainiert, versucht es, die Merkmalsvektoren einer Klasse zu dessen Mittelwert zu bewegen, wobei gleichzeitig die Klassen auseinanderwandern und so besser trennbar werden. Auf Grund der Dynamik der Ziele, die sich bei jeder Iteration ändern, haben wir den Lernansatz *moving targets*

genannt. Dank der Sigmoidfunktionen der Ausgangsschicht breiten sich die Merkmalsvektoren nicht bis ins Unendliche aus, sondern werden im Bereich zwischen Null und Eins gehalten.

Ein Problem ergibt sich dennoch mit dieser Beschränkung des Ausgangsraumes. Da die Sigmoidfunktion auf jede Komponente des Ausgangsvektors angewandt wurde, landen viele Ausgaben, die in den gesättigten Bereich des Sigmoids fallen, in den Ecken eines m-dimensionalen Würfels. In den Ecken dieses Hyperwürfels verfangen sich einzelne Klassen und sind durch die dann stets nach außen gerichtete Klassendrift nicht mehr "bewegungsfähig" (Bild 6.11).

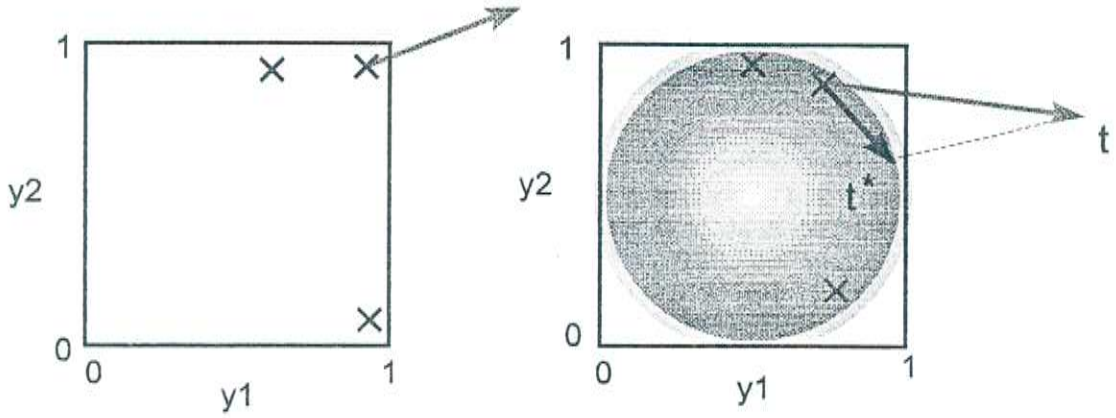
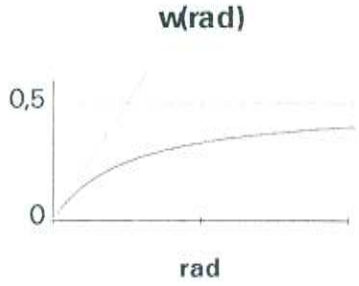


Bild 6.11: Damit sich die Klassen nicht in den Ecken des Ausgangsraumes verfangen, werden die Zielvektoren über eine radiale Funktion abgebildet.

Um dies zu verhindern beschränkt man die Zielvektoren auf eine Hyperkugel, in der es keine Ecken gibt. Nun können sich alle Klassen am Rand dieser Kugel frei ausbreiten. Zu dieser Hyperkugel gelangt man, indem die radiale Komponente *rad* zum Mittelpunkt m_0 des Ausgangsraumes betrachtet wird.

$$\text{rad} = |t - m_0| \quad \text{mit} \quad m_0 = \left(\tfrac{1}{2},\ \tfrac{1}{2},\ \ldots,\ \tfrac{1}{2}\right)^T. \qquad (6.21)$$

Statt eine Sigmoidfunktion zu verwenden wird bei der Berechnung des neuen Zielvektors t* eine ähnlich, aber sehr viel einfacher zu berechnende Funktion w(rad) benutzt.



$$w(rad) = \frac{rad}{(2 \cdot rad + 1)} \qquad (6.22)$$

Bild 6.12: Die Funktion w(rad)

$$t^* = w(rad) \cdot \frac{(t - m_0)}{rad} + m_0$$

$$= \frac{t - m_0}{(2 \cdot rad + 1)} + m_0 \qquad (6.23)$$

Dadurch werden die Zielvektoren wie in einer Wanne festgehalten. Die initiale Einstellung der Gewichte wird so vorgenommen, daß die Funktion w(rad) in ihrem relativ linearen Bereich ausgewertet wird. Die zunächst großen Klassendriften werden in dem Maße kleiner, je weiter die Mittelwerte nach außen driften. Durch die begrenzende "Wannen"-funktion w(rad) kann sich schließlich ein Gleichgewicht ergeben, in dem der Zielvektor (nur Klassendrift berücksichtigt) mit dem Klassenmittelpunkt übereinstimmt. In diesem Zustand werden dann verstärkt die Ballungen der einzelnen Klassen minimiert.

6.4. Initialisierung und Anwendung

Für den Einsatz in der Spracherkennung kann ein Neuronales Netz der Form, wie in Bild 6.13 zu sehen, verwendet werden. Mehrere Eingangsdaten v(k) verschiedener Zeitpunkte k oder Funtionen davon (z.B. Delta-Koeffizienten) werden zu einem großen Eingangsvektor x zusammengefaßt. Durch das Neuronale Netz wird die Dimensionalität wieder reduziert und für jeden Zeitpunkt ein Merkmalsvektor y ausgegeben.

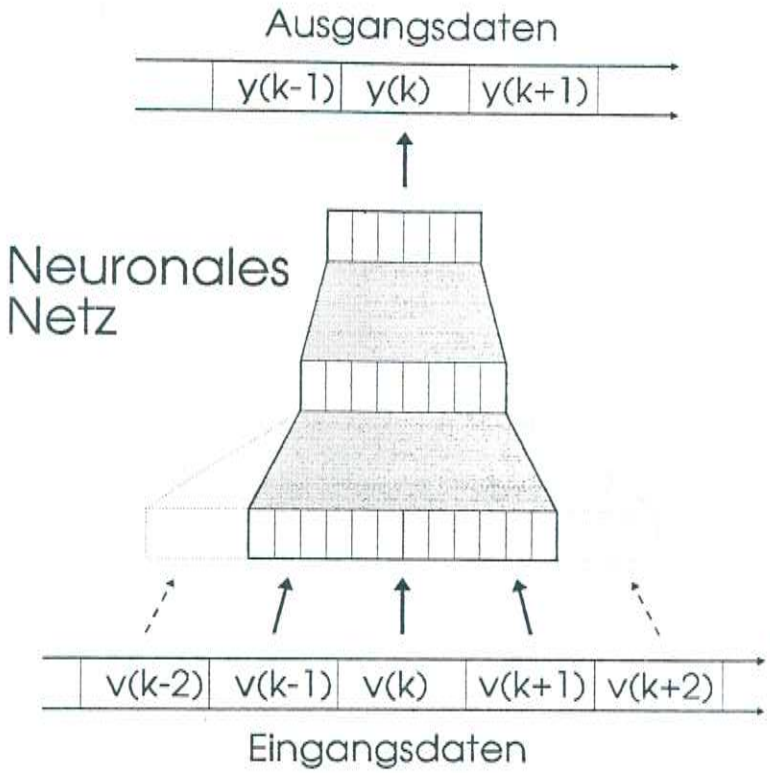


Bild 6.13: Neuronales Netz zur Merkmalsextraktion in der Spracherkennung

Ein nicht unwichtiges Problem bei Neuronalen Netzen stellt die Initialisierung dar. Bei einer geschickten Initialisierung kann man sich etliche Iterationen sparen, um zum gewünschten Ergebnis zu kommen. Bei Verwendung des Diskriminanzanalyse-Netzweks (DAN) kann auf dem Ergebnis der linearen Diskriminanzanalyse (LDA) aufgebaut werden, die eine optimale Lösung für die Gütekriterien (3.6) - (3.8) darstellen. Dies gilt nur für die Voraussetzung einer linearen Transformation. Mit der nichtlinearen Abbildung, die das Netz durchführen kann, lassen sich die Gütekriterien weiter maximieren. Zunächst wird das Netz so initialisiert, daß es gerade die Transformation der LDA ausführt. Für die Versuche im Rahmen dieser Diplomarbeit wurde dazu folgende Variante ausgewählt:

- Verwendet wird ein Netz mit einer versteckten Schicht (englisch: *hidden layer*)
- Die Gewichte von der Eingangsschicht zur versteckten Schicht wurden so eingestellt, daß man im linaren Bereich der Ausgabefunktion f(a) liegt und die Ausgabe dieser Schicht bis auf eine Skalierung mit der Eingabe identisch ist
- Die Gewichte zur Ausgangsschicht wurden mit der LDA-Transformations-matrix initialisiert.

Die Anzahl der Einheiten in den Schichten ergibt sich dadurch wie folgt:

- Eingangsschicht: n Einheiten
- Versteckte Schicht: n Einheiten
- Ausgangsschicht: m Einheiten (m<n)

Andere Initialisierungsmöglichkeiten seien hier für zukünftige Untersuchungen angegeben:

- LDA-Transformation gleich nach der Eingangsschicht, dadurch erhält man aber weniger Einheiten in der versteckten Schicht.
- LDA-Transformation von der Eingangsschicht direkt in die Ausgangsschicht. Zufällige Initialisierung der Gewichte zur und von der versteckten Schicht, die somit in der Anzahl ihrer Einheiten nicht beschränkt ist.
- Außerdem läßt sich das Netz mit weiteren versteckten Schichten ausbauen, die alle mit einer Einheitsabbildung initialisiert sind. Prinzipiell ist aber eine versteckte Schicht mit einer genügend großen Anzahl von Einheiten ausreichend.

Im Rahmen dieser Diplomarbeit wurde ein C-Programm erstellt, mit dem sich Neuronale Netze mit bekannten Lernansätzen, wie *Selbstassoziation*, *Musterassoziation* und *Klasseneinteilung* (siehe S.55f) und dem neu entwickelten Lernansatz des *DAN* realisieren lassen.

Da das Training Neuronaler Netze, besonders mit großen Datenmenge wie man sie in der Spracherkennung findet, sehr zeitintensiv ist, wurden Versuche nur mit einigen ausgewählten Klassen durchgeführt und die Ergebnisse mit denen durch LDA erreichbaren verglichen. Als Abbruchskriterium und zum Vergleich können die Gütekriterien, wie sie auch bei der LDA benutzt werden, herangezogen werden.

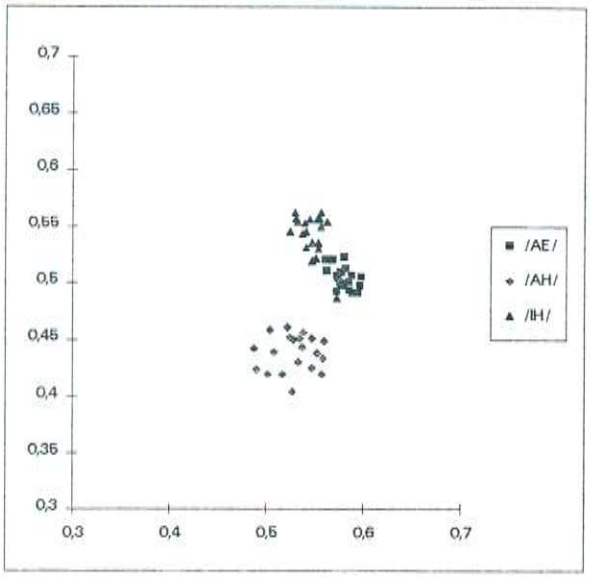
Es sei noch einmal erwähnt, daß diese Kriterien nicht von der absoluten Skalierung der Merkmalsvektoren abhängen. Streckt man den Raum, vergrößern sich zwar die Abstände, die Kriterien liefern aber immer noch dieselben Werte.

Es folgt ein Beispiel, das zur Anschauung dient und deshalb die in JANUS verwendeten 16dimensionalen Datenvektoren in einen 2dimensionalen Raum projeziert. Neben anderen Versuchen zeigt das Beispiel in Bild 6.14, bei dem übersichtshalber nur drei Klassen (Phoneme) verwendet wurden, den Effekt, der durch das DAN erreichbar ist. Im linken Bild findet man die Merkmalsvektoren, wie sie gleich nach der Initialisierung durch das Netz gebildet werden. Das Aussehen der Ballungen entspricht dem durch eine LDA-Transformation erreichbaren. Der Wert des

Gütekriteriums $J_1(m)$ und eines ähnlichen, auch bei der LDA gebräuchlichen Kriteriums mit den Spuren der Matrizen ist darunter angegeben. Nach einigen Iterationen mit dem DAN sind die Ballungen sehr viel leichter zu unterscheiden und die Werte der Gütekriterien entsprechend größer.

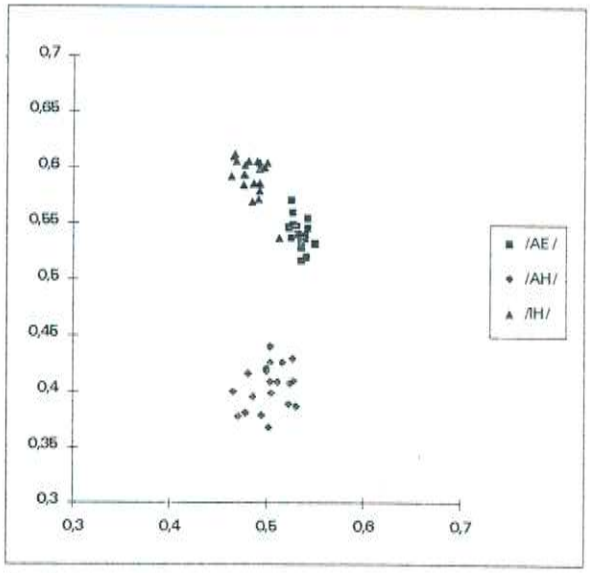
Ausgabe des neuronalen Netzes:

ohne Training



$\frac{|T|}{|W|}$ =25,8 $\frac{\text{Spur}(T)}{\text{Spur}(W)}$ =6,12

nach 120 Iterationen



$\frac{|T|}{|W|}$ =77,8 $\frac{\text{Spur}(T)}{\text{Spur}(W)}$ =15,5

Bild 6.14: Ausgaben des Neuronalen Netzes und Werte zweier Gütekritrien

Die Versuche mit dem DAN waren recht vielversprechend und bedürfen nun einer Anwendung im Rahmen des JANUS-Spracherkenners. Hier muß sich zeigen, ob Verbesserungen der Erkennungsrate erreichbar sind und wenn ja, ob sich der Aufwand für das Training des Netzes im Vergleich zur Leistungssteigerung lohnt. Die für die Merkmalsextraktion während Training und Anwendung des Spracherkenners notwendigen Berechnungen sind zwar etwas aufwendiger im Vergleich mit einer einfachen Matrixmultiplikation durch die Verfahren LDA und HAT, aber dieser Mehraufwand ist vernachlässigbar gegenüber den umfangreichen Berechnungen im Klassifizierer.

Das Neuronale Netz hat noch einen weiteren Vorteil gegenüber den aus statistischen Methoden abgeleiteten linearen Transfomationen LDA und HAT. Es läßt sich mit neuen Trainingsdaten jederzeit nachadaptieren, während die statistischen Methoden die gesamten Trainingsdaten, bzw. zumindest deren Scattermatrizen benötigen.

7. Zusammenfassung

In der Spracherkennung sind zwei Dinge besonders wichtig. Zum einen soll die Fehlerrate klein sein, damit eine sinnvolle Anwendung möglich ist. Außerdem muß das Sprachsignal in einer möglichst kompakten Form extrahiert werden. Mit wenigen aussagekräftigen Merkmalen ist eine schnelle und leistungsfähige Erkennung möglich. Am JANUS-Spracherkenner aus dem C-Star Projekt wurden zwei statistische Methoden zur Dimensionalitätssreduktion oder auch Merkmalsextraktion implementiert und untersucht. Beide Methoden verwenden eine Transformationsmatrix und führen damit eine *lineare* Abbildung der Sprachdaten auf die Merkmalsvektoren durch.

Die Hauptachsentransformation (HAT) reduziert dabei die Dimensionalität, ohne daß viel Information über die Sprache verloren geht. Mit der HAT ist ein Trainieren des Spracherkenners unkompliziert und sie ermöglicht eine Reduktion auf 10 Dimensionen der bei JANUS verwendeten und schon zuvor auf 16 Dimensionen vorverarbeiteten *melscale*-Koeffizienten des Sprachsignals, ohne dabei die Erkennungsleistung zu verschlechtern.

Mit der Linearen Diskriminanzanalyse (LDA) wird eine dimensionalitätsreduzierende Abbildung gefunden, die zum Unterschied zur HAT auch die zu unterscheidenden Klassen berücksichtigt. Dabei ist neben der Frage nach der Anzahl der Dimensionen auf die reduziert werden soll, auch die Definition der Klassen zu berücksichtigen. Für den verwendeten Spracherkenner haben sich Subphoneme, die aus einer Dreiteilung von Phonemen (Lauten) der Sprache entstehen, als beste Klassendefinition erwiesen. Durch die LDA konnte die Dimensionalität zum Teil auf mehr als die Hälfte reduziert werden und das bei gleichzeitiger relativer Verbesserung der Fehlerrate um 18% bis knapp 50%, je nach Version. Für die Zukunft sind Experimente wünschenswert, die eine größere Menge Koeffizienten aus dem Sprachsignal gewinnen und diese mit der LDA auf eine Anzahl reduzieren, die den Rechenaufwand gering hält. Dadurch wird dem Erkenner bei gleichem Aufwand mehr Information über das Sprachsignal bereitgestellt, so daß seine Leistungsfähigkeit gesteigert wird. Bei der Bestimmung der Transformationsmatrix ergibt sich noch die Möglichkeit, die einzelnen Klassen auf unterschiedliche Weise zu gewichten. Klassen, die bisher nur sehr wenig eingingen, aber vielleicht gerade auf Grund ihrer Seltenheit besonders wichtig sind, können stärker gewichtet werden. Das Umgekehrte gilt für Klassen wie *Silence*, die sehr dominant sind.

Eine der LDA entsprechende *nichtlineare* Transformation, die weitaus leistungsfähiger sein könnte, ist nicht durch statistische Methoden herleitbar. Mit einem hier neuentwickelten Ansatz für ein Neuronales Netz wurden erste Experimente unternommen, die zeigen, daß damit eine nichtlineare Transformation gefunden werden kann, die Verbesserungen der durch die LDA benutzten Gütekriterien liefert. Die Anwendung mit einem Spracherkenner bleibt noch zu untersuchen. Die positiven Ergebnisse der LDA lassen darauf schließen, daß sich zukünftige Versuche in dieser Richtung lohnen.

Literatur

[Aub 93] Aubert, X.; Haeb-Umbach, R.; Ney H.: **Continuous Mixture Densities and Linear Discriminant Analys for Improved Context-Dependent Acoustic Models**, Proc. ICASSP, Minneapolis MN, 1993, S. II648-651

[Bro 87] Bronstein, I.N.; Semendjajew, K.A.: **Taschenbuch der Mathematik**, 23. Aufl. B.G. Teubner Verlagsgesellschaft, Leipzig, 1987

[Dud 73] Duda, R.O.; Hart, P.E.: **Pattern Classification and Scene Analysis**, Wiley, New York, 1973

[Fuk 72] Fukunaga, K.: **Introduction to Statistical Pattern Recognition**, 1. Aufl., Academic Press, NewYork and London, 1972, S. 259-267

[Fuk 90] Fukunaga, K.: **Introduction to Statistical Pattern Recognition**, 2. Aufl., Academic Press, San Diego and London, 1990, S. 445ff

[Gei 90] Geiser, G.: **Mensch-Maschine-Kommunikation**, Oldenbourg Verlag, München, 1990

[Hae 92] Haeb-Umbach, R.; Ney, H.: **Linear Discriminant Analysis for Improved Large Vocabulary Continuous Speech Recognition**, Proc. ICASSP, San Francisco CA, 1992, S. I13-16

[Hae 93] Haeb-Umbach, R.; Geller, D.; Ney H.: **Improvements in Connected Digit Recognition Using Linear Discriminant Analysis and Mixture Densities**, Proc. ICASSP, Minneapolis MN, 1993, S. II239-242

[Hua 89] Huang, X.; Jack, M.A.: **Semi-Continuous Hidden Markov Models for Speech Signals**, Readings in Speech Recognition edited by A. Waibel & K.F. Lee, Morgan Kaufmann Publishers, San Mateo, 1990, S. 340-345

[Hua 90] Huang, X.; Lee, K.F.; Hon, H.: **On Semi-Continuous Hidden Markov Modelling**, Proc. ICASSP, Albuquerque, NM , 1990, S. 689-692

[Hun 91] Hunt, M.J.; Richardson, S.M.; Bateman, D.C. Piau, A.:**An Investigation of PLP and IMELDA Acoustic Representations and of their Potential for Combination,** Proc. ICASSP, Toronto, Canada, May 1991, S. 881-884

[Kam 93] Kambhatla, N.; Leen, T.: **Fast Non-Linear Dimensionen Reduction**, Vol. 3, Proc. IEEE Int. Conference on Neuronal Networks, 1993, S. 1213-1218

[Käm 90] Kämmerer, B.R..: **Sprecherunabhängigkeit und Sprecheradaption**, 1. Aufl., Springer-Verlag, Berlin, Heidelberg, 1990

[Kro 86] Kroschel, K.: Statistische Nachrichtentheorie - Erster Teil: **Signalerkennung und Parameterschätzung**, 2.Auflage, Springer-Verlag, Berlin, 1986

[Kam 89] Kammeyer, K.D.; Kroschel, K.: **Digitale Signalverarbeitung, Filterung und Spektralanalyse,** B.G.Teubner , Stuttgart, 1989

[Kra 91] Kramer, A. M.: **Nonlinear Principal Component Analysis Using Autoassociative Neural Networks,** AIChE Journal, 37:233-243, 1991

[Ney 90] Ney, H.: **Experiments on Mixture-Density Phenomene-Modelling for the Speaker-Independent 1000-Word Speech Recognition DARPA-Task,** Proc. ICASSP, Albuquerque, NM, 1990, S. 713-716

[Pre 88] Press, W.H.; Flannery, B.S.; Tukolsky, S.A.; Vetterling, W.T.: **Numerical Recipes in C**, 1. Aufl., Cambridge University Press, Cambridge, New York, Melbourne, 1988

[Pri 88] Price, P.; Fischer, W.; Bernstein, J.; Pallet, D.: **A Database for Continuous Speech Recognition in a 1000-Word Domain,** Proc. ICASSP, New York, April 1988, S.651-654

[Qua 92] Quante, A.:**Informationsverarbeitung für Diagnosesysteme,** Skript zur Vorlesung an der Universität Karlsruhe, 1992

[Rab 88] Rabiner, L.R.; Wilpson, J.G.; Soong, F.K.: **High Performance Connected Digit Recognicion Using Hidden Markov Models,** Proc. ICASSP, New York, April 1988

[Rum 86] Rumelhart, D.; McClelland,J.:**Parallel Distributed Processing,** Vol. 1. § 2. Cambridge, MA: MIT Press., 1986

[Rus 88] Ruske, G.: **Automatische Spracherkennung**, 1. Aufl., Oldenbourg Verlag, München Wien, 1988

[Suh 92] Suhm, B.: **Fließbandverarbeitung in der Vorverarbeitung von Sprachsignalen**, Studienarbeit Universität Karlsruhe, 1992

[Wai 91] Waibel, A.; Jain, A.N.; McNair, A.E.; Saito, H.; Hauptmann, A.G.; Tebelskis, J.: **JANUS: A Speech-to-Speech Translation System Using Connectionist and Symbolic Processing Strategies,** Proc. ICASSP, Toronto, Canada, 1991

[Wai 92] Waibel, A.; Jain, A.N.; McNair, A.;Tebelskis, J.; Osterholtz, L.; Saito, H.; Schmittbauer, O.; Sloboda, T.; Woszczyna, M.: **JANUS: Speech-to-Speech Translation Using Connectionist Techniques,** Advances in Neural Information Processing Systems 4, ed. J.E.Moody, S.J. Hansen, R.P.Lippman, San Mateo, CA:Morgan Kaufmann Publishers, 1992

[Web 89] Webb, A.; Lowe, D.: **The Optimised Internal Representation of Multilayer Classifier Networks Performs Nonlinear Discriminant Analysis,** Neural Networks, Vol. 3, U.S.A., S. 367-375

[Yu 90] Yu, G.; Russell, W.; Schwartz, R.; Makhoul, J.:**Discriminant Analysis and Supervised Vector Quantization for Continuous Speech Recognition,** Proc. ICASSP, Albuquerque, NM, April 1990, S 685-688

Abkürzungen :
ICASSP = International Conference on Acoustics, Speech and Signal Processing.
IEEE = Institute of Electrical and Electronics Engineers.