Karlsruhe Institute of Technology

Implementierung und Evaluation eines Large Margin Estimation Algorithmus für HMMs

Diplomarbeit am Cognitive Systems Lab
Prof. Dr.-Ing. Tanja Schultz
Fakultät für Informatik
Karlsruher Institut für Technologie

von

cand. inform.
Michael Ikkert

Betreuer:
Prof. Dr.-Ing. Tanja Schultz
Dipl.-Math. Michael Wand
Dipl.-Inform. Ngoc Thang Vu

Tag der Anmeldung: 11. Februar 2013
Tag der Abgabe: 10. August 2013

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 9. August 2013

Zusammenfassung

Kontinuierliche Hidden-Markov-Modelle stellen für viele moderne Spracherkenner die zentrale Komponente dar. Diese Modelle werden aus Beispielaufzeichnungen von Sprache mit bekanntem Inhalt automatisch geschätzt. Da letztlich ungesehene Sprache erkannt werden soll, ist es entscheidend dass die Modelle generalisierbar sind, also auch unbekannte Sprachaufzeichnungen korrekt beschreiben. Das Large-Margin-Verfahren wendet ein an Support Vector Machines angelehntes Optimierungskriterium an, um die Qualität der Modelle im Hinblick auf ihre Generalisierbarkeit zu bewerten. Die Maximierung dieses Kriteriums soll die Generalisierungsfähigkeit gegenüber anderen Verfahren verbessern.

In dieser Arbeit wird das beschriebene Verfahren im Rahmen des JANUS Recognition Toolkit implementiert und evaluiert. Bei der Evaluation wird für das Verfahren neben der Erkennung akustischer Sprachsignale auch die Anwendung auf EMG-Signale untersucht. Es werden auch unterschiedliche Modellierungs- und Optimierungsebenen betrachtet. Die Modellierung wird, beginnend auf der niedrigsten Modellierungsebene, zur Erkennung kontinuierlicher Sprache mit einem großen Vokabular erweitert. Die Ergebnisse werden im Hinblick auf die Effektivität und Effizienz des Verfahrens für die Optimierung von Modellen bewertet.

Inhaltsverzeichnis

Inhaltsverzeichnis **7**

1 Einleitung **11**

1.1 Motivation 11

1.2 Stand der Forschung 12

1.3 Zielsetzung 12

1.4 Gliederung der Arbeit 13

2 Grundlagen **15**

2.1 Physiologische Grundlagen 15

2.2 Elektromyographie 15

2.2.1 Muskelkontraktion 16

2.2.2 Messung von Muskelaktivität 16

2.3 Automatische Spracherkennung 17

2.3.1 Merkmalsextraktion 18

2.3.1.1 Merkmalsextraktion für ein akustisches Signal 18

2.3.1.2 Merkmalsextraktion für EMG 21

2.3.2 Akustisches Modell 22

2.3.2.1 Hidden Markov Model 22

2.3.2.2 Modellierung durch Phoneme 23

2.3.2.3 Kontextabhängige Modellierung 23

2.3.2.4 Aussprachewörterbuch 24

2.3.2.5 Modellierung der Emissionswahrscheinlichkeiten . . . 24

2.3.3 Sprachmodell 25

2.3.4 Decoding 25

2.4 Training akustischer Modelle 26

2.4.1 Maximum Likelihood Estimation 26
2.4.2 Maximum Mutual Information Estimation 27
2.4.3 Minimum Classification Error 29
2.4.4 Large/Maximum Margin HMM 29

3 Mathematische Grundlagen des implementierten Verfahrens 35
3.1 Marginbegriff 35
3.2 Zielfunktion 38
3.3 Optimierung 39

4 Implementierung 43
4.1 Das JANUS Recognition Toolkit 43
4.1.1 Merkmalsextraktion 44
4.1.2 Akustische Modellierung 44
4.1.3 MLE-Training 44
4.2 Verwendung des Large-Margin-Training 45
4.3 Implementierungsgrundlagen 46
4.4 Training auf Fensterebene 50
4.5 Training auf kontinuierlicher Sprache 51

5 Experimente 55
5.1 Korpora 55
5.1.1 EMG-UKA-Korpus 55
5.1.2 GlobalPhone 56
5.2 Training auf Fensterebene 57
5.2.1 Atomare Modelle auf EMG-UKA-Datensatz 57
5.2.2 Atomare Phonemmodelle auf GlobalPhone-Datensatz 58
5.2.3 Fensterbasiertes Training kontextunabhängiger Phonemmodelle auf GlobalPhone-Datensatz 61
5.2.4 Fensterbasiertes Training kontextabhängiger Phonemmodelle auf GlobalPhone-Datensatz 63
5.3 Training auf kontinuierlicher Sprache 65

6 Zusammenfassung und Ausblick 67

Anhang A Codebeispiele **69**

A.1 Training auf Fensterebene 69

A.2 Training auf Äußerungsebene 69

Anhang B Ergebnisse **71**

B.1 Fensterbasiertes Training atomarer Modelle auf dem EMG-UKA-Datensatz 71

B.1.1 Phonemerkennung auf Audiosignal 71

B.1.2 Erkennung phonetischer Klassen auf Audiosignal 72

B.1.3 Phonemerkennung auf EMG-Signal 73

B.1.4 Erkennung phonetischer Klassen auf EMG-Signal 74

B.2 Fensterbasiertes Training atomarer Modelle auf dem GlobalPhone-Datensatz 75

B.2.1 Sprecherabhängiges Training 75

B.2.2 Sprecherunabhängiges Training, Test auf bekannten Sprechern 75

B.2.3 Sprecherunabhängiges Training, Test auf unbekannten Sprechern 76

B.3 Fensterbasiertes Training kontextunabhängiger Phonemmodelle . . . 77

B.3.1 Training kontextunabhängiger Subpolyphone 77

B.4 Fensterbasiertes Training kontextabhängiger Phonemmodelle 78

B.4.1 Training von 250 kontextabhängigen Modellen 78

B.4.2 Training von 2500 kontextabhängigen Modellen 78

B.5 Training auf Äußerungsebene 79

B.5.1 Training auf Äußerungsebene 79

B.5.2 Training auf Wortebene 81

Literaturverzeichnis **85**

Kapitel 1

Einleitung

Ein großer Teil der modernen Spracherkenner nutzt *Hidden Markov Modelle (HMMs)* um den Vorgang der menschlichen Sprachproduktion zu repräsentieren. HMMs sind gut geeignet Prozesse zu modellieren, bei denen ein verborgener Vorgang ein beobachtetes Signal in Form einer Folge diskreter Ereignisse erzeugt. Auf Sprache übertragen bietet sich die Vorstellung an, dass einem gedachten Wort unbewusst eine Reihe von Lauten zugeordnet wird. Ein Außenstehender kann allerdings weder das gedachte Wort, noch die Lautfolge beobachten. Erst durch Bewegungen des Artikulationsapparates werden Schallwellen erzeugt, die auch ein Gesprächspartner wahrnehmen kann. Da bei der maschinellen Verarbeitung eine Diskretisierung von Signalen kaum zu vermeiden ist, ist in diesem Fall die Wahrnehmung in Aufzeichnungen für diskrete Zeitpunkte unterteilt.

Bei dieser Vorstellung ist es also die zu einem Wort gehörende Lautfolge, die durch die verborgenen Zustände modelliert wird. Eine Wahrscheinlichkeitsverteilung beschreibt dann die Variation bei der Umsetzung dieser Laute durch den menschlichen Sprachapparat. Diese Verteilungen erlauben es eine beobachtete Audiosequenz im Hinblick auf die Wahrscheinlichkeit ihrer Erzeugung durch ein Modell zu bewerten und auf diese Weise Modelle verschiedener Wörter zu vergleichen.

Zunächst müssen diese Modelle jedoch erstellt werden. Diese Arbeit befasst sich speziell mit der Festlegung der Wahrscheinlichkeitsverteilungen für die einzelnen Zustände. Durchgeführt wird diese Festlegung anhand von Audiosequenzen, für die das korrekte Modell bereits bekannt ist. Der Vorgang ein Modell zu finden, das eine Menge von Beispielsequenzen beschreibt, wird als *Training* bezeichnet. Um der, im Verhältnis zum Volumen des Merkmalsraumes extrem begrenzten, Menge an *Trainingsbeispielen* Rechnung zu tragen, legt man sich normalerweise auf eine *parametrisierte Verteilung* – typischerweise *Gaußmischverteilungen (GMMs)* – fest. Auf diese Weise muss nur eine relativ geringe Anzahl von Parametern geschätzt werden, um den gesamten Merkmalsraum zu beschreiben.

1.1 Motivation

Die einfachste Strategie die Parameter eines Modelles zu schätzen ist, sie so zu wählen, dass die Trainingsdaten bestmöglich beschrieben werden. Dieser *generative* Ansatz garantiert jedoch keine optimale Unterscheidung der durch verschiedene

Modelle repräsentierten Klassen. *Diskriminative* Trainingsverfahren setzen an dieser Stelle an und suchen nach einer Beschreibung, in der nicht nur die typischen Eigenschaften einer Klasse im Vordergrund stehen, sondern die Eigenschaften in denen sie sich von anderen Klassen unterscheidet.

1.2 Stand der Forschung

Diese Arbeit untersucht ein spezielles Verfahren, das in [12] vorgestellt und in einer Reihe folgender Arbeiten weiterentwickelt wurde. Der Ansatz soll die diskriminativen Eigenschaften einer *Support Vector Machine (SVM)* auf Hidden Markov Modelle übertragen. Angelehnt an ein als "Margin" bezeichnetes Optimierungskriterium für SVMs wird dieser Arbeit die Optimierung des Modells im Hinblick auf ein entsprechendes Kriterium als "Large Margin Estimation" (LME) bezeichnet. Die Autoren versprechen sich davon eine deutlichere Übertragung von auf dem Trainingsdatensatz erzielten Verbesserungen auf ungesehene Testdaten, als bei anderen diskriminativen Ansätzen.

In [20] werden auch Ergebnisse zu Experimenten auf OGI ISOLET vorgestellt. In dieser Datenbank liegt eine Reihe separat gesprochener Buchstaben aus dem englischen Alphabet vor. Es wurden Hidden Markov Modelle mit jeweils 16 Zuständen für jeden Buchstaben trainiert, und die Ergebnisse von *Maximum Likelihood (MLE)*, *Minimum Classification Error (MCE)* und LME wurden miteinander verglichen. Das LME Training wurde auf den besten Modellen aus dem MCE Training durchgeführt. Als bestes Ergebnis wurden 95.19% mit durch LME geschätzten Modellen erreicht, ausgehend von mit MCE geschätzten Modellen mit einer Genauigkeit von 94.07%.

Ein verfeinertes Verfahren mit einem abgewandelten Marginbegriff wird in folgenden Arbeiten eingeführt und auf dem TIDIGITS Datenkorpus getestet [23]. Damit wurde der Sprung auf kontinuierliche Sprache gemacht, wenn auch mit kleinem Vokabular, das durch HMMs auf Wortebene modelliert wurde. Es wird gezeigt, dass es für Mischverteilungen mit zunehmender Anzahl von Gaußglocken kaum noch eine Rolle spielt, ob auf MLE oder MCE aufgebaut wird und auch die Nutzung von Negativbeispielen wird untersucht. Insgesamt werden bis zu 99.24% Genauigkeit erreicht, gegenüber 98.89% bei MCE und 98.34% bei MLE.

Zuletzt wurde eine Definition des LME Prinzipes als Instanz semidefiniter Programmierung (SDP) zu vorgeschlagen [13]. Das hat vor allem den Vorteil, dass diese Optimierung konvex ist und somit lokale Optima vermieden werden. Während auch hier eine Verbesserung erreicht wurde, mussten die Autoren eine Lösung für den rapide mit der Modellkomplexität wachsenden Aufwand finden. Dazu wird die zu optimierende Matrix durch eine Reihe kleinerer Matrizen ersetzt, wodurch der Lösungsaufwand sinken soll [37].

1.3 Zielsetzung

Das Ziel ist ein ausgewähltes Large-Margin-Verfahren im Kontext eines vorhandenen Systems zu implementieren und diesen zusätzlichen Schritt zu verwenden um Modelle mit besserer Erkennungsleistung zu erhalten. Als System wurde das in einer Kooperation des KIT und der Carnegie Mellon University entwickelte JANUS

Recognition Toolkit (JRTk) gewählt. Neben Audioaufzeichnungen von Sprache liegen für das JRTk auch Systeme vor, die auf *Elektromyographischen Aufzeichnungen (EMG-Aufzeichnungen)* von Sprache operieren. Das erlaubt die Erprobung sowohl auf Audioaufzeichnungen, als auch auf EMG-Aufzeichnungen.

Da allerdings in einem Spracherkenner dieser Komplexität viele Faktoren eine Rolle spielen, wurde eine schrittweise Herangehensweise gewählt, die es ermöglicht zunächst wichtige Aspekte des Trainingsverfahrens zu beleuchten. Es wird mit minimalen Modellen begonnen, die jeweils nur einen Zustand haben. Einerseits umgeht man auf diese Weise Einflüsse durch Entscheidungen, die bei komplexeren Modellen zusätzlich getroffen werden müssen, andererseits zeigen sich bei der Auswertung Einflüsse durch den zusätzlichen Trainingsschritt klarer. So ist auch die Entscheidung motiviert die Arbeit auf einem Konzept aufzubauen, das nicht als Instanz semidefiniter Programmierung definiert wird.

In diesem minimalen Basisfall erreichte Verbesserungen sind das Fundament der darauf folgenden Erweiterungen. Spätere Experimente haben es vorwiegend zum Ziel den Einfluss verschiedener Faktoren auf die Leistung des Large-Margin-Trainings zu untersuchen. Da Diskriminierbarkeit eine wichtige Rolle spielt, werden Experimente mit einer unterschiedlichen Anzahl Klassen durchgeführt und verglichen. Die Vermutung, dass die hohe Anzahl teilweise sehr ähnlicher Klassen bei komplexen Spracherkennungssystemen eine Herausforderung für dieses Verfahren darstellt, soll an dieser Stelle geprüft werden. Auch das Training mit einer unterschiedlichen Menge an Trainingsbeispielen wird untersucht. Der hohe Zusatzaufwand durch diskriminatives Training wirft die Frage auf, ob das Verfahren genauso von zusätzlichen Trainingsdaten profitiert, oder die Verbesserung mit wachsendem Trainingsdatensatz den Aufwand nicht mehr rechtfertigt. Schließlich werden auch verschiedene Ansätze erprobt, die Erkennung kontinuierlicher Sprache zu verbessern und die Ergebnisse werden ausgewertet.

1.4 Gliederung der Arbeit

In Kapitel 2 findet sich eine Übersicht von Hintergrundwissen, das zum Verständnis der Arbeit wichtig ist. Die Beschreibung der Umsetzung beginnt in Kapitel 3 mit den mathematischen Grundlagen der implementierten Variante des Large-Margin-Trainings. Im Anschluss daran befasst sich Kapitel 4 mit Details der tatsächlichen Implementierung des Algorithmus. Kapitel 5 enthält Information zu den verwendeten Korpora, sowie Aufbau und Ergebnissen der Experimente. Kapitel 6 fasst die Ergebnisse der Arbeit zusammen und beschäftigt sich mit möglichen Ansätzen für weiterführende Arbeit.

Kapitel 2

Grundlagen

Dieses Kapitel dient der Übersicht über vorausgesetztes Hintergrundwissen. Zu Beginn beschreibt der Abschnitt 2.1 den Vorgang menschlicher Sprachproduktion. Abschnitt 2.2 gibt eine Einführung in die Grundlagen, auf denen Elektromyographie basiert. Anschließend wird in Abschnitt 2.3 die Funktionsweise eines Spracherkenners umrissen. Zuletzt erläutert Abschnitt 2.4 das Large-Margin-Verfahren im Kontext ausgewählter Verfahren für das Training von Hidden Markov Modellen.

2.1 Physiologische Grundlagen

Menschliche Sprache ist die Modulation eines durch die Lungen erzeugten Luftstromes. Als erstes beeinflussen ihn die Stimmbänder. Der als Stimmritze bezeichnete Spalt zwischen ihnen lässt sich durch Muskelspannung an den Stimmbändern verengen. Zusammen mit dem Druck der durch die Luftröhre strömenden Luft führt dies durch den Bernoullieffekt zu einem annähernd periodischen Öffnen und Schließen der Stimmbänder. Die durchströmende Luft wird dabei in quasiperiodische Schwingungen versetzt. Diese Schwingung bietet die Grundlage für stimmhafte Laute. Für stimmlose Laute wird die Stimmritze weit geöffnet, so dass Verwirbelungen der durchströmenden Luft den Klang eines aperiodischen Rauschens haben.

Nach der Stimmritze gelangt die Luft in den Vokaltrakt. Zu diesem gehören Rachenraum, Mund- und Nasenhöhle und er wirkt wie ein Filter auf die Schallwellen. Die Eigenschaften dieses Filters können durch Bewegung der Artikulationsorgane, zu denen beispielsweise Unterkiefer und Zunge gehören, verändert werden. Als letztes haben auch Zähne und Lippen einen starken Einfluss auf den Klang der Schallwellen, die der Luftstrom trägt. Für ausführliche Erläuterungen zu dem Thema sei an dieser Stelle auf [10] verwiesen.

2.2 Elektromyographie

Elektromyographie (EMG) bezeichnet die Aufzeichnung und Untersuchung von elektrischen Potentialen, die Muskelaktivität begleiten. Speziell ist für die hier betrachteten Anwendungen die Skelettmuskulatur interessant, die aufgrund ihrer Struktur auch als quergestreifte Muskulatur bezeichnet wird. Daneben gibt es im menschlichen Körper auch sogenannte glatte Muskeln. Die glatte Muskulatur wird direkt vom zentralen Nervensystem gesteuert und ist für Bewegung in inneren Organen

und Blutgefäßen verantwortlich. Für die Sprachartikulation kommen die willentlich steuerbaren Skelettmuskeln zum Einsatz. Diese Muskeln stehen daher im Rahmen dieser Arbeit im Vordergrund.

2.2.1 Muskelkontraktion

Muskelkontraktion wird ausgelöst, indem über das Nervensystem eine spezielle Zelle, das *Motoneuron*, durch einen Nervenimpuls aktiviert wird. Deren Axone enden in der *motorischen Endplatte*, einer chemischen Synapse, die für die Übertragung des Impulses an die Muskelfasern verantwortlich ist.

Die Ausschüttung eines Botenstoffes, die durch den über das Axon ankommenden Nervenimpuls ausgelöst wird, führt zum Einströmen von Kationen in eine angrenzende Muskelzelle. Dadurch verändert sich mit der Ionenverteilung auch die Potentialdifferenz über die Zellmembran hinweg. Während die Zellen der Muskelfaser im Ruhezustand ein negatives *Ruhepotential* haben, verändert es sich durch die einströmenden Kationen positiv. Überschreitet diese Veränderung einen Grenzwert, so beginnen Ionenkanäle an der Zellmembran sich zu öffnen und lassen weitere Kationen einströmen. Diese Kettenreaktion verändert das Potential an der Membrangrenze zu einem positiven sogenannten *Aktionspotential*. Dieses Aktionspotential ist unabhängig von der Stärke des auslösenden Impulses. Die etwa gleichzeitig geöffneten, aber langsamer reagierenden Kaliumkanäle bewirken über die Zeit die Wiederherstellung des Ruhepotentiales. In einer folgenden *Refraktärzeit* reagiert die Zelle gar nicht bis kaum auf ankommende Impulse.

Benachbarte Zellen reagieren ebenfalls auf das Aktionspotential, auch sie werden durch den Ablauf aktiviert, so dass eine Ausbreitung des Aktionspotentials über den gesamten Muskel stattfindet. Durch die Unempfindlichkeit der Zellen während der Refraktärzeit breitet sich das Aktionspotential nach der Entstehung an der motorischen Endplatte nur in Richtung der noch nicht erfassten Zellen eines Muskels aus.

Innerhalb einer Muskelzelle verursacht das Aktionspotential auch das Einströmen von Kalziumkationen aus einem intrazellulären Speicher (dem *sarkoplasmatischen Retikulum*) in die Muskelfasern. Eine Reaktion innerhalb der Muskelfaser führt zu deren Kontraktion, solange Kalziumkationen zur Verfügung stehen. Um diesen Prozess zu wiederholen, aber auch zum Entspannen des Muskels, wird außerdem Energie in Form von *Adenosintriphosphat (ATP)* benötigt.

Eine einzelne Aktivierung führt nur zu einer geringfügigen Verkürzung der Muskelfaser. Erst periodische Wiederholung des Vorganges durch eine Reihe aufeinanderfolgender Nervenimpulse und folgender Aktionspotentiale führt zu einer deutlichen Verkürzung.

2.2.2 Messung von Muskelaktivität

Elektromyographie misst das der Muskelaktivität vorangehende Aktionspotential. Zwei Elektroden an verschiedenen Positionen eines Muskels zeigen eine Potentialdifferenz, wenn die Zellen an einer Elektrode bereits vom sich ausbreitenden Aktionspotential erfasst wurden, die Zellen an der zweiten Elektrode jedoch noch nicht.

Wertet man die Potentialdifferenz auf diese Weise zwischen zwei Stellen auf einem Muskel aus, so spricht man von *bipolarer Ableitung*. Die Alternative stellt *unipolare*

Ableitung dar, für die eine Referenzelektrode an einer Stelle angebracht wird, an der keine Muskelaktivität vermutet wird. Eine Potentialdifferenz zu dieser Elektrode zeigt dann Muskelaktivität.

Unabhängig von der Ableitungsmethode wird stets eine Potentialdifferenz zwischen zwei Elektroden bestimmt. Die bipolare Ableitung zeigt vorwiegend bessere Ergebnisse [24], da hier viele lokale Artefakte durch die räumliche Nähe häufig beide Elektroden in gleichem Maße beeinflussen und damit bei der Differenzbildung abgemildert werden. Unipolare Ableitung liefert dagegen ein Signal, das nicht von Potentialen an anderen Positionen abhängt.

2.3 Automatische Spracherkennung

Automatische Spracherkennung beschäftigt sich damit, einer Aufzeichnung von Sprache die entsprechende Darstellung in Textform zuzuordnen. Der Aufbau eines Spracherkennungssystems wird in Abb. 2.2 dargestellt.

In dieser Arbeit werden zwei mögliche Quellen für die Aufzeichnung von Sprache verwendet: die akustische Aufzeichnung über ein Mikrofon und die Aufzeichnung von EMG-Signalen über Oberflächenelektroden. Die klassische dabei ist eine Aufzeichnung über ein Mikrophon. Da die Wahrnehmung des Klanges die typische Wahrnehmung von Sprache durch den Menschen ist, steht die Untersuchung dieser Aufzeichnungsmethode im Zentrum der Forschung.

Spracherkennung per EMG hat gegenüber der Audioaufzeichnung den Vorteil, dass sie nicht auf die Schallwellen angewiesen ist, sondern sich darauf beschränkt zur Artikulation verwendete Muskulatur zu beobachten. Umgebungsgeräusche haben beispielsweise so keinen Einfluss auf den Vorgang und es wird sogar möglich, geflüsterte Sprache oder eine Artikulation ohne hörbaren Schall zu erkennen. Arbeiten wie [32] gehen diese Aufgabe an, jedoch sind die Ergebnisse noch deutlich von der bei traditioneller Spracherkennung erreichten Genauigkeit entfernt.

Die zur EMG-Spracherkennung verwendeten Oberflächenelektroden werden auf Gesicht und Hals angebracht, um möglichst viele an der Artikulation beteiligte Muskeln zu erfassen. Die in dieser Arbeit verwendeten EMG-Signale wurden mit den in [24] gewählten Elektrodenpositionen aufgezeichnet. Abbildung 2.1 zeigt diese Positionen.

Die Unterscheidung zwischen den untersuchten Quellen beschränkt sich vorwiegend auf die Vorverarbeitung. Neben einer *Analog-Digital-Wandlung* des Eingabesignales, die für jede maschinelle Verarbeitung notwendig ist, umfasst diese die *Merkmalsextraktion*. Dieser zweite Teil der Vorverarbeitung konvertiert das Signal durch eine Reihe von Zwischenschritten in eine Folge von Zahlenvektoren, den sogenannten *Merkmalsvektoren*. Idealerweise wird an dieser Stelle von der Form des Eingabesignales abstrahiert. Verfahren für die Suche nach Mustern in einer vorgegebenen Sequenz von Punkten lassen sich dann auf berechnete Merkmalsvektoren anwenden, unabhängig davon, ob sie aus Schallwellen oder aus EMG-Signalen erzeugt wurden. Auf die unterschiedlichen Verfahren für diese beiden Quellen Merkmalsvektoren aus dem Signal zu extrahieren wird in Abschnitt 2.3.1 eingegangen.

Ein akustisches Modell beschreibt eine idealisierte Approximation an das stochastische Verhalten der Signalquelle. Für das Training dieses Modells liegt neben einer Sequenz von Merkmalsvektoren auch eine Transkription der Trainingsdaten vor, also

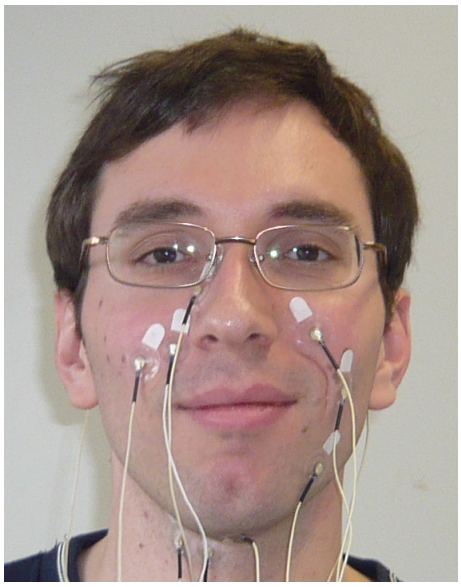


Abbildung 2.1: Für EMG-Spracherkennung verwendete Elektrodenpositionen [32]

die Ausgabe, die bei der Erkennung für diese Daten zu erzeugen wäre. Da die Trainingsdatenmenge im Verhältnis zur Größe des zu beschreibenden Merkmalsraumes verschwindend gering ist, ist die Struktur des Modells in Grundzügen vorgegeben, eine Reihe von Parametern muss im Training auf die tatsächlichen Daten abgestimmt werden. Die verwendeten Modelle werden in Abschnitt 2.3.2 beschrieben. Da in dieser Arbeit ein Trainingsverfahren für akustische Modelle untersucht wird, wird in Abschnitt 2.4 separat eine Übersicht über einige ausgewählte Verfahren gegeben.

Neben dem Verhalten der Signalquelle, wird auch die Sprache als Raum verschiedener Äußerungen modelliert. Das Sprachmodell approximiert die Wahrscheinlichkeitsverteilung der möglichen Wortfolgen anhand eines Textkorpus. Ein solcher Korpus enthält eine Vielzahl von geschriebenen Sätzen, aus denen ein Modell der Struktur von Sätzen gebildet wird. Diese Modellierung wird in Abschnitt 2.3.3 beschrieben.

Die beiden Modelle stellen bei der eigentlichen Erkennung eine Bewertung zur Verfügung. Sie werden im Rahmen des Trainings eines Spracherkenners erzeugt und bleiben bei dessen Einsatz normalerweise unverändert. Die *Decoding*-Komponente nutzt eine Kombination aus beiden Bewertungen bei der Suche nach einer optimalen Wortfolge. Die bestbewertete Wortfolge stellt dann das Ergebnis des Erkennungsvorganges dar. Mit diesem Suchvorgang beschäftigt sich Abschnitt 2.3.4.

2.3.1 Merkmalsextraktion

Vor einer Verarbeitung durch den Spracherkenner muss ein erhaltenes Signal in eine maschinell nutzbare Form gebracht werden. Diese Merkmalsextraktion hat zum Ziel redundante Information zu reduzieren, während die relevanten Bestandteile für folgende Schritte hervorgehoben werden sollen. Dies ist auch der wichtigste Punkt, in dem sich Spracherkennung auf akustischen Daten von einer auf einem EMG Signal unterscheidet.

2.3.1.1 Merkmalsextraktion für ein akustisches Signal

Eine typische Merkmalsvektordarstellung für eine Audioaufzeichnung von Sprache sind *Mel-Frequency Cepstral Coefficients (MFCC)*. Dieser Begriff fasst eine Strate-

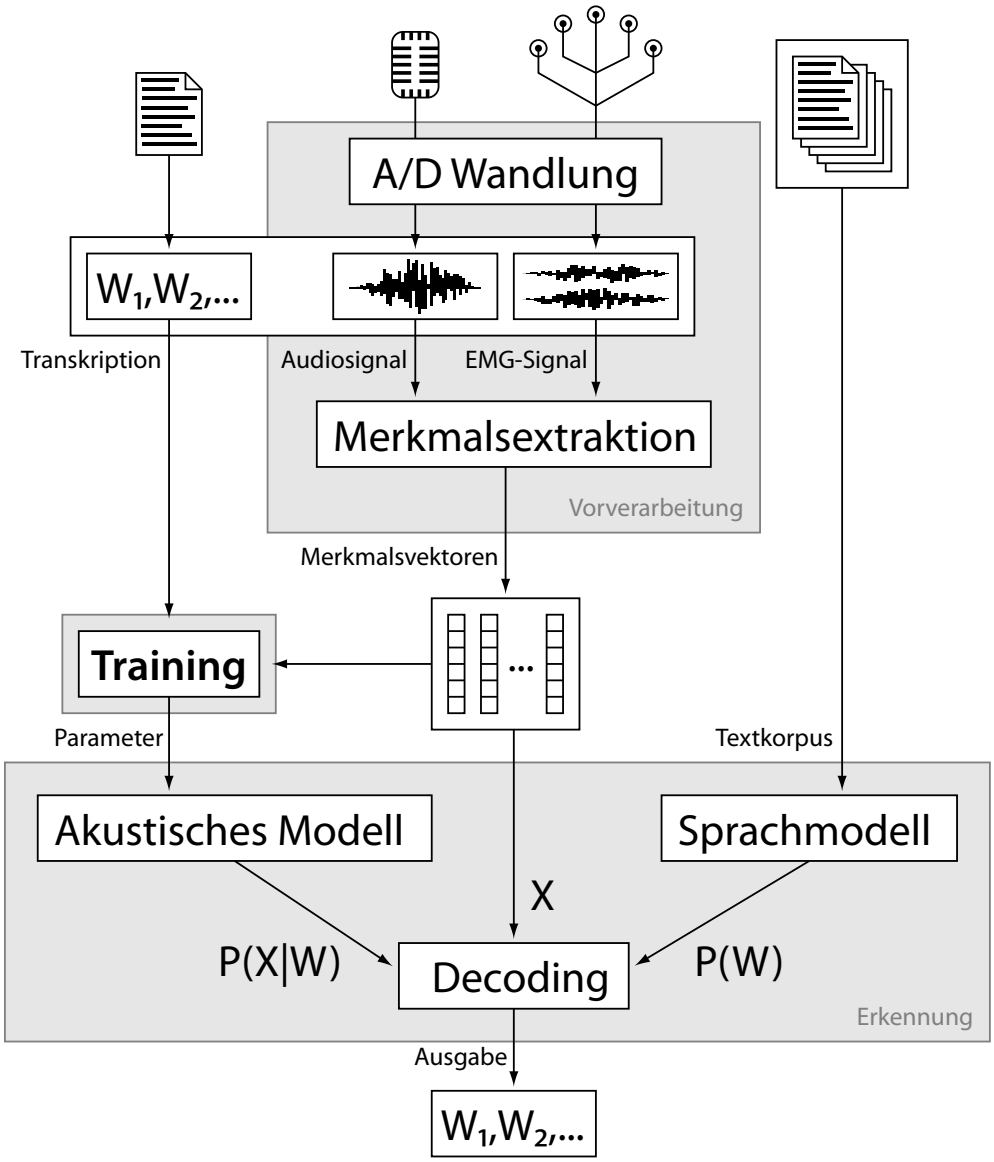


Abbildung 2.2: Übersicht über ein Spracherkennungssystem

gie für die Vorverarbeitung zusammen, die den Schall als eine Überlagerung von Schallwellen verschiedener Frequenz betrachtet und ihre Wahrnehmung durch den Menschen berücksichtigt. Im Folgenden werden die in dieser Arbeit zur Merkmalsextraktion angewandten Schritte beschrieben:

Fensterung: Der erste Schritt trägt der Veränderung über die Zeit Rechnung. Die für die Gewinnung des Frequenzspektrum verwendete *Fouriertransformation* erwartet ein *stationäres* Signal, also eines über dessen Verlauf die Frequenzanteile sich nicht verändern. Dagegen ist das vorliegende Signal *quasi-stationär*, die Zusammensetzung verändert sich über die Zeit, jedoch verändert der Vokaltrakt innerhalb von kurzen Zeitfenstern seine Form nur geringfügig und damit ist in einem solchen Fenster auch das Spektrum stabil. Die Fensterung unterteilt daher das Signal durch Multiplikation mit einer Fensterfunktion in kurze Abschnitte, 16 ms in dieser Arbeit. Die für diese Multiplikation verwendete Funktion soll die Auswirkungen einer unvermeidlichen Verzerrung des

Spektrums durch eine solche Operation beschränken, in dieser Arbeit wird das sogenannte *Hamming-Window* verwendet. Um die Folgen einer Zerschneidung an willkürlich gezogenen Fenstergrenzen zu minimieren, gibt es eine Überlappung der Fenster dadurch, dass in jedem Schritt die Fensterfunktion zeitlich um 10 ms und nicht die vollen 16 ms verschoben wird.

FFT: Die *Fast Fourier Transform (FFT)* stellt ein effizientes Verfahren zur Bestimmung der *diskreten Fouriertransformation* einer diskreten, periodischen Funktion dar. Aus dem Ergebnis dieser Transformation lässt sich die Energieverteilung über die Frequenzanteile für jedes Fenster bestimmen.

VTLN: In diesem Zwischenschritt soll durch *Vocal Tract Length Normalization* für die Erkennung irrelevante Verzerrung des Spektrums ausgeglichen werden, die aus Unterschieden bei der Beschaffenheit des Vokaltraktes verschiedener Sprecher folgt.

Mel Filterbank: Die Fähigkeit des menschlichen Ohres Töne voneinander zu unterscheiden hängt stark davon ab, in welchem Frequenzbereich sie liegen. So können Menschen vor allem in niedrigeren Frequenzbereichen Töne viel feiner unterscheiden, als in höheren. Da menschliche Sprache auf Wahrnehmung durch den Menschen ausgerichtet ist, werden auch Besonderheiten bei der menschlichen Klangwahrnehmung berücksichtigt. Aus dem Spektrum des Signales werden daher durch Multiplikation mit Dreiecksfunktionen auf Mel-Frequenzbänder verteilt. Diese Aufteilung liefert einen Vektor von *Mel-Scale Spectral Coefficients*.

Logarithmierung: Auch die Lautstärke nimmt das menschliche Ohr nicht linear wahr. Den logarithmischen Verlauf der Wahrnehmung linear ansteigender Lautstärke kompensiert eine Logarithmierung der erhaltenen Werte.

DCT: Eine *diskrete Kosinustransformation (Discrete Cosine Transform, DCT)* überführt den Vektor in den *Cepstralbereich*, in dem sich viele an der Sprachproduktion beteiligte Einflüsse leichter isolieren lassen. Verwendet werden nur die ersten 13 Koeffizienten und der aus ihnen gebildete Vektor enthält die Mel-Frequency Cepstral Coefficients.

Mean Substraction: Der Merkmalsvektor wird durch Subtraktion des Mittelwertes seiner Einträge normalisiert.

Stacking: Für Untersuchungen des Spektrums musste ein Fenster möglichst frei von Effekten zeitlicher Veränderung in der Sprache sein. Für die Erkennung haben sich aber Informationen über die Tendenz einer Veränderung in der Frequenzzusammensetzung als vorteilhaft erwiesen. In dieser Arbeit wird der Kontext durch sogenanntes *Stacking* berücksichtigt. Es wird für jedes Fenster ein neuer Merkmalsvektor durch Konkatenation von diesem und den Merkmalsvektoren von fünf vorhergehenden, sowie fünf nachfolgenden Fenstern gebildet.

LDA: Durch Konkatenation nimmt auch die Dimensionalität des Merkmalsraumes erheblich zu. Eine Anwendung *linearer Diskriminanzanalyse (Linear Discriminant Analysis, LDA)* liefert eine Basis für diesen Vektorraum, in der ein

Großteil der Varianz zwischen Klassen bereits durch einen von wenigen Basisvektoren erzeugten Teilraum abgedeckt wird. So lässt sich die Unterscheidbarkeit der Klassen erhalten, aber die Dimensionalität stark reduzieren. In dieser Arbeit durchgeführte Experimente reduzieren den Merkmalsraum dadurch auf 32 Dimensionen.

2.3.1.2 Merkmalsextraktion für EMG

Gegenüber der Spracherkennung auf akustischen Signalen ändert sich die Merkmalsdarstellung erheblich. Während für Schallwellen eine Spektralanalyse verwendet wird, liefert bei EMG eine Darstellung im Zeitbereich bessere Ergebnisse [15]. Weiterhin liegen im Falle von EMG-Signalen mehrere Kanäle mit Information zu Aktivität unterschiedlicher Muskelgruppen vor. Als Kanal wird hier jeweils ein Elektrodenpaar bezeichnet, dessen Potentialdifferenz über die Zeit untersucht wird. Die fünf Kanäle, die hier betrachtet werden, werden über einen großen Teil der Merkmalsextraktion hinweg unabhängig voneinander ausgewertet.

Die Merkmalsextraktion für EMG-Signale beginnt ebenfalls mit einer Fensterung. Die Fenster sind 27 ms lang und es wird eine Verschiebung von 10 ms bei der Fensterung verwendet. Für jeden Datenpunkt $x[t]$ in Fenster x eines einzelnen Kanales wird zunächst ein doppelt über neun Werte gemittelter Wert w als Hilfsgröße bestimmt, also ein

$$w[t] := \frac{1}{9}\sum_{n=-4}^{4} v[t+n] \text{ mit } v[t] := \frac{1}{9}\sum_{n=-4}^{4} x[t+n].$$

Daraus wird eine zweite Hilfsgröße abgeleitet, das *hochfrequente Signal* $p[t]$:

$$p[t] := x[t] - w[t].$$

Mit diesen beiden Hilfswerten für jeden Datenpunkt in einem Fenster werden die folgenden Merkmale für das komplette Fenster und für jeden Kanal bestimmt:

$\overline{w}$: Mittelwert von w, definiert durch

$$\overline{w} := \frac{1}{|w|}\sum_{t=1}^{|w|} w[t].$$

P_w: Energie von w, definiert durch

$$P_w := \sum_{t=1}^{|w|} |w[t]|^2.$$

$\overline{r}$: Durchschnitt über den Betrag des hochfrequenten Signals, definiert durch

$$\overline{r} := \frac{1}{|p|}\sum_{t=1}^{|p|} |p[t]|.$$

P_p**:** Energie von p, definiert durch

$$P_p := \sum_{t=1}^{|p|} |p[t]|^2.$$

z_p**:** Anzahl der Nulldurchgänge in p.

Konkatenation dieser Merkmale bildet einen vorläufigen Merkmalsvektor $f := (\overline{w}, P_w, \overline{r}, P_p, z_p)$ für jeden EMG-Kanal. Daraus wird durch Stacking mit einem Kontext von 15 – also Konkatenation von dem Merkmalsvektor für ein Fenster und den Merkmalsvektoren von je 15 vorhergehenden und 15 nachfolgenden Zeitfenstern – der Merkmalsvektor um Kontextinformation erweitert. Der *TD15-Merkmalsvektor* wird nun durch Konkatenation dieser Vektoren für die fünf verwendeten Kanäle zusammengesetzt.

Bei einem Merkmalsvektor aus fünf Kanälen mit jeweils fünf Merkmalen für jedes Fenster und einem Kontext von insgesamt 31 Fenstern ergibt sich eine Dimensionalität von 775 für den TD15-Merkmalsvektor. Wie bereits im Falle von Spracherkennung auf akustischen Daten, wird auch hier LDA eingesetzt, um den Merkmalsvektor auf 32 Dimensionen zu reduzieren.

2.3.2 Akustisches Modell

Ein verbreiteter Ansatz für die akustische Modellierung von Sprache ist das Hidden Markov Modell (HMM). Da sich diese Arbeit im Speziellen mit dem Training der Parameter eines kontinuierlichen HMM beschäftigt, wird dieses nun vorgestellt.

2.3.2.1 Hidden Markov Model

Definiert wird ein HMM durch ein 5-Tupel $\lambda = (S, V, A, B, \pi)$ mit Einträgen, die im Folgenden beschrieben werden. Das Modell hat eine Menge $S = \{s_1, s_2, \ldots s_{|S|}\}$ von Zuständen, zu jedem Zeitpunkt nimmt es einen davon an. Eine Matrix von $A = \{a_{ij}\}$ beschreibt eine Verteilung von Übergangswahrscheinlichkeiten. Befindet sich also das Modell in einem Zustand s_i, so hat es die Wahrscheinlichkeit a_{ij} nach einem diskreten Zeitschritt den Zustand s_j anzunehmen. Hier fließt auch die *Markoveigenschaft* ein, die besagt dass bei der Entscheidung für einen Zustandsübergang nur der gegenwärtige Zustand eine Rolle spielt.

Bei jedem Schritt emittiert das Modell auch ein Symbol aus dem Ausgabealphabet V. Die Verteilung dieser Emission legt $B = \{b_i\}$ in Abhängigkeit vom gegenwärtigen Zustand fest. Ist das Modell in Zustand s_i, so ist die Wahrscheinlichkeit ein Symbol $v \in V$ zu emittieren $b_i(v)$. Zuletzt legt die Verteilung π fest, in welchem Zustand das Modell mit welcher Wahrscheinlichkeit beginnt.

Das Wort "Hidden" bezieht sich darauf, dass bei Prozessen, die durch ein HMM modelliert werden, der Zustand nicht beobachtet werden kann. Im Falle der Spracherkennung kann es die Repräsentation des Wortes als Lautfolge sein, die durch die versteckte Zustandsfolge dargestellt wird. Beobachtet wird aber erst ein in einem zweiten Zufallsprozess vom Sprachapparat erzeugtes Signal, das als Merkmalsvektor vorliegt.

2.3.2.2 Modellierung durch Phoneme

Verschiedene Worte als Ketten von Lauten darzustellen und unabhängig voneinander zu trainieren, ist nur für ein kleines Vokabular realistisch. Für komplexere Spracherkennungssysteme reicht dieser Ansatz längst nicht mehr aus. Jedes Wort, das ins Vokabular aufgenommen wird, würde viele Trainingsbeispiele benötigen, um eine zuverlässige Schätzung zu garantieren. Gleichzeitig kommt bei der Erkennung ganzer Sätze ein hoher Verzweigungsgrad an Übergängen zwischen den Worten hinzu, was den Aufwand dramatisch erhöht.

Da viele Wörter gemeinsame Laute enthalten, lohnt es sich, alle Vorkommen eines Lautes durch eine einzige Modellierungsklasse zu beschreiben. Das HMM für ein Wort wird dazu in Teilmodelle aufgebrochen, von denen jedes einen eigenen Laut darstellt. Wörter oder ganze Äußerungen lassen sich anschließend durch Verkettung von HMMs für einzelne Laute konstruieren. Auf diese Weise lassen sich auch Trainingsbeispiele für einen Laut in einem Wort ebenfalls als Beispiele für den gleichen Laut in einem anderen Wort nutzen. Die Phonolgie beschäftigt sich bereits mit der Suche nach einem solchen Satz von minimalen Sprachbausteinen für einzelne Sprachen. Zwei Laute werden hier als unterschiedliche *Phoneme* bezeichnet, wenn durch Austausch von einem durch den Anderen ein Wort in eines mit einer anderen Bedeutung überführt werden kann.

Obwohl Phoneme eine sehr feine Einheit für die Unterteilung von Sprache sind, sind auch sie keinesfalls grundsätzlich homogen in ihrem Klang über die Zeit. Bei der Modellierung von Phonemen durch ein HMM nutzt man daher üblicherweise drei Zustände (*Subphoneme*) für Beginn, Mitte und Ende eines Phonems. So soll der Tatsache Rechnung getragen werden, dass viele Phoneme sich über die Zeit verändern.

2.3.2.3 Kontextabhängige Modellierung

Der menschliche Artikulationsapparat unterliegt selbstverständlich physikalischen Einschränkungen. Es ist daher nicht verwunderlich, dass Phoneme nicht isoliert gesprochen werden, sondern die Bewegungen der Artikulationsorgane Effekte über Phonemgrenzen hinweg haben. Natürlich wird eine Bewegung zum aktuellen Phonem hin auch vom vorhergehenden beeinflusst, aber es gibt auch eine antizipierende Bewegung, die noch in dem aktuellen Phonem bereits auf den nächsten Laut abzielt.

Um diese sogenannten *Koartikulationseffekte* zu erfassen, bietet die Spracherkennung das Konzept *kontextabhängiger Modellierung* durch *Subpolyphone*. Das gleiche Subphonem kann hier durch mehrere verschiedene Verteilungen beschrieben werden. Zwischen ihnen gewählt wird dann aufgrund der Nachbarphoneme des übergeordneten Phonems. Selbst bei *Triphonen*, also Polyphonen mit jeweils einem Nachbarn in beiden Richtungen, stößt man schnell auf verschiedene Probleme, versucht man sämtliche Kontexte zu modellieren. Neben dem Zusatzaufwand der schieren Anzahl an Klassen, die zu unterscheiden wären, spielt auch eine Rolle, dass erst für alle Kontexte genügend Trainingsbeispiele vorliegen müssten. Ebenfalls möchte man nicht riskieren, die erhaltenen Klassen zu spezifisch zu modellieren, als dass sie unbekannte Daten noch immer korrekt beschreiben.

Ein Lösungsansatz hier ist es, ein Subphonem nur in Abhängigkeit vom Kontext zu trennen, falls die vorliegenden Trainingsdaten einen großen Unterschied in der

Modellierung durch den Kontext nahelegen. Hier öffnen sich auch neue Möglichkeiten für eine Betrachtung des Kontextbegriffes. Es muss nicht mehr explizit nach speziellen Phonemen als Nachbarn unterschieden werden, stattdessen kann für eine Gruppe von Nachbarphonemen mit bestimmten Eigenschaften (beispielsweise nach Zungenposition ausgewählt), eine Abspaltung vorgenommen werden. Das in dieser Arbeit verwendete JRTk nutzt zu diesem Zweck einen Fragenkatalog von phonetisch modellierten Unterscheidungsmerkmalen. Ein Entropiemaß auf den Mixturgewichteverteilungen wird für die Erzeugung eines Entscheidungsbaumes genutzt. Der Algorithmus beginnt mit allen, für ein Phonem im Trainingskorpus angetroffenen, Kontexten als einzigem Blattknoten. In mehreren Schritten wird jeweils ein Blatt so aufgeteilt, dass der Entropieverlust bei der Aufspaltung maximiert wird. Im Detail wird das Verfahren in [29] erläutert.

2.3.2.4 Aussprachewörterbuch

Durch die Modellierung der Sprachproduktion durch Zustandsfolgen auf Phonemebene und der geschriebenen Sprache durch Folgen ganzer Wörter tut sich eine Lücke auf. Diese wird durch das Aussprachewörterbuch geschlossen. Im Aussprachewörterbuch wird einem geschriebenen Wort eine Darstellung als Phonemfolge zugeordnet, aus der sich anschließend ein HMM für dieses Wort konstruieren lässt.

Zwar gibt es meist eine kanonische Aussprache für jedes Wort, aber auch Abweichungen sind häufig. Neben situationsbedingten Versprechern, beispielsweise durch Ablenkung, oder Folgen von Lauten, die eine besondere Koordination erfordern, existieren auch Varianten in der Sprache, die systematische Abweichungen einführen. Diese können sprecherspezifisch sein oder – wie im Falle von Dialekten – typisch für eine Gruppe von Menschen.

Bei der Konstruktion eines Spracherkenners lohnt es sich, häufige Aussprachevarianten für ein Wort in das Aussprachewörterbuch aufzunehmen. Jede mögliche Variante lässt sich allerdings nicht berücksichtigen, da dies sowohl durch den Aufwand für das Zusammentragen dieser Varianten für jedes Wort, als auch durch die Zunahme der Verwechselbarkeit bei der Erkennung unpraktikabel ist.

2.3.2.5 Modellierung der Emissionswahrscheinlichkeiten

Bislang wurde nicht näher auf die Modellierung der in der Definition als B bezeichneten Verteilungen eingegangen. Tatsächlich ist das HMM hier sehr flexibel. Es wird ausschließlich eine Verteilung für jeden Zustand benötigt, die die Emissionswahrscheinlichkeit für jedes Symbol im Ausgabealphabet – in diesem Fall also jeden Merkmalsvektor – festlegt.

Für diese Arbeit werden an dieser Stelle *Gaußmischverteilungen* betrachtet. Eine Gaußmischverteilung ist durch die gewichtete Summe von *Normalverteilungen* definiert. Parametrisiert wird daher die Emissionswahrscheinlichkeit in einem Zustand durch eine Liste von Gaußglocken mit ihren Gewichten. Jede Gaußglocke wird wiederum durch Mittelwert μ und Kovarianzmatrix Σ parametrisiert. Ein HMM, das Gaußmischverteilungen verwendet, wird häufig als *kontinuierliches HMM (Continuous Density Hiddden Markov Model, CDHMM)* bezeichnet.

2.3.3 Sprachmodell

Die Wahrscheinlichkeit einer Wortfolge zu schätzen ist eine schwierige Aufgabe, besonders da sie für die optimale Lösung ein inhaltliches und kontextuelles Vorwissen erfordert, sowie die Fähigkeit den Inhalt der Sprache zu verstehen. Die hier verwendete Approximation ist weit davon entfernt. Im Rahmen dieser Arbeit wird ein Sprachmodell angenommen, das auf *Trigrammen* basiert. Für ein solches Sprachmodell wird ein Trainingstextkorpus rein statistisch ausgewertet. Bei einem *n-Gramm* wird für Wort auch ein Kontext von $n-1$ Vorhergehenden berücksichtigt, im Falle von Trigrammen wären das zwei. Die Bewertung eines Wortes in einem solchen Kontext ist dann gerade die Wahrscheinlichkeit, dass auf den Kontext im Trainingskorpus dieses Wort folgt.

Da auch bei großen Mengen Text nicht jede mögliche Wortsequenz beobachtet wird, gibt es mehrere Ansätze zu vermeiden, dass Worte eine Wahrscheinlichkeit von 0 erhalten. Neben der Zuweisung von einem Teil der Wahrscheinlichkeitsmasse an nicht vorkommende Wortfolgen (*Discounting*) gibt es auch die Möglichkeit bei seltenen Vorkommen auf n-Gramme niedrigerer Ordnung zurückzufallen (*Backoff*), oder von Anfang an die Bewertung durch eine gewichtete Summe von n-Grammen verschiedener Ordnung zu bestimmen (*Interpolation*). Ein Vergleich dazu wird in [2] besprochen.

2.3.4 Decoding

Betrachtet man das Problem der Erkennung menschlicher Sprache aus dem Blickwinkel der Statistik, dann will man für eine in eine Sequenz von Merkmalsvektoren umgewandelte Sprachaufzeichnung X eine Wortfolge W wählen, so dass die bedingte Wahrscheinlichkeit $P(W|X)$ der Wortfolge gegeben der Aufzeichnung maximiert wird. Nach der *Bayesregel* entspricht es folgender Maximierung:

$$\operatorname*{argmax}_W P(W|X) = \operatorname*{argmax}_W \frac{P(X|W)P(W)}{P(X)}$$

Da $P(X)$ nicht von der Wortfolge abhängt und folglich für eine Maximierung keine Rolle spielt, wird es vernachlässigt. Die verbleibenden beiden Faktoren $P(X|W)$ und $P(W)$ repräsentieren die beiden zentralen probabilistischen Komponenten eines Spracherkenners. Die Wahrscheinlichkeit $P(X|W)$ liefert das akustische Modell. Nach der Konstruktion eines zu der textuellen Repräsentation gehörenden HMM, kann dieses die Wahrscheinlichkeit schätzen, dass beim Sprechen dieses Textes das aufgezeichnete Signal entsteht. Für $P(W)$ ist das Sprachmodell verantwortlich. Die Rolle dieser Komponente ist die Bewertung der Plausibilität einer hypothetischen Wortsequenz.

Auch die eigentliche Suche nach der optimalen Textdarstellung, die durch argmax_W repräsentiert wird, stellt für komplexere Anwendungsfälle ein eigenständiges Problem dar. Schnell wird hier der Raum aller möglichen Wortfolgen so groß, dass alle Möglichkeiten auszuwerten nicht praktikabel ist. Eine Lösung dafür ist der *Beam-Search-Algorithmus* [10]. Statt ein HMM für einen ganzen Satz zu konstruieren, wird die Konstruktion des akustischen Modells mit dessen Auswertung verbunden. Vergleichbar mit Breitensuche werden bei diesem Ansatz zunächst unterschiedliche Pfade einer Länge betrachtet, bevor sie um einen weiteren möglichen Folgezustand

erweitert werden. Da ein HMM für eine Teilsequenz von Merkmalsvektoren und eine Zustandsfolge gleicher Länge eine Bewertung liefern kann, lassen sich bereits für eine solche Teilsequenz mehrere Pfade vergleichen. Zugunsten einer Reduktion des Aufwandes werden beim Beam-Search-Algorithmus nach jeder Erweiterung nur die bestbewerteten Pfade weiterverfolgt. Durch diese Einschränkung kann nicht mehr garantiert werden, dass der so bestimmte Pfad optimal ist, da sich die Rangfolge in der Bewertung im Laufe der Erweiterung verändern kann. Durch einen, als *Beam Width* bezeichneten, Parameter lässt sich dieses Risiko steuern. Dieser Parameter legt fest, wie viele parallele Pfade weiterverfolgt werden. Mehr Pfade zu verfolgen senkt das Risiko einen Pfad aufzugeben, der sich im späteren Verlauf als optimal herausstellen würde, erhöht jedoch den Rechenaufwand.

2.4 Training akustischer Modelle

Der am weitesten verbreitete Ansatz für das Training der akustischen Modelle von Spracherkennern ist die *Maximum Likelihood Estimation (MLE, Maximum-Likelihood-Schätzung)*. Dabei wird jedes Modell so trainiert, dass die Bewertung diesem Modell zugeordneter Trainingsbeispiele maximiert wird. Effiziente Algorithmen machen diese Methode attraktiv. Erforscht werden aber auch Verfahren, die sich nicht darauf beschränken, Klassen zu modellieren, sondern bei denen Unterscheidbarkeit im Vordergrund steht. Der größte Nachteil dieser *diskriminativen Algorithmen* liegt in ihrem Aufwand. Bereits dadurch, dass für das Training eines Modells auch Trainingsbeispiele anderer Klassen als Negativbeispiele berücksichtigt werden müssen, wächst der Aufwand. Weiterhin sind für viele Verfahren keine effizienten Algorithmen bekannt, so dass auf Standardalgorithmen für die Optimierung zurückgegriffen werden muss. Im Folgenden wird zunächst Maximum Likelihood Estimation als klassischer Ansatz in Abschnitt 2.4.1 vorgestellt. Anschließend werden in den Abschnitten 2.4.2 und 2.4.3 zwei populäre diskriminative Ansätze beschrieben. Da sich Verfahren, die als "Large-Margin-Estimation" bezeichnet werden teils erheblich unterscheiden, wird in Abschnitt 2.4.4 zuletzt ein Überblick über verschiedene Varianten bei der Übertragung des Large-Margin-Kriteriums für die Anwendung auf HMMs gegeben.

2.4.1 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) hat als Optimierungsziel die Maximierung der Likelihood $p(X|\omega_{\mathrm{id}(X)})$, also der Wahrscheinlichkeit dass ein der Transkription zugeordnetes Modell $\omega_{\mathrm{id}(X)}$ das Trainingsbeispiel X emittiert. Im hier betrachteten Fall handelt es sich bei $\omega_{\mathrm{id}(X)}$ um ein HMM, das die in der durch eine Folge X von Merkmalsvektoren beschriebene Äußerung darstellt. Durch Auswertung der Übergangswahrscheinlichkeiten, sowie der Emissionswahrscheinlichkeiten einzelner Zustände kann dann die Likelihood $p(X|\omega_{\mathrm{id}(X)})$ für ein gegebenes HMM bestimmt werden. Mit dem Baum-Welch Algorithmus, der eine Instanz des allgemeinen Expectation-Maximization-Algorithmus (EM-Algorithmus) [4] darstellt, ist auch eine effiziente Optimierungsmethode für HMMs bekannt.

Beim allgemeinen EM Algorithmus handelt es sich um ein iteratives Verfahren, das bevorzugt verwendet wird, wenn keine geschlossene Lösung bekannt ist. Grund dafür ist typischerweise, dass nicht alle Zufallsvariablen des modellierten Vorganges beobachtet werden können und daher ebenfalls geschätzt werden müssen. Es ergibt sich

eine gegenseitige Abhängigkeit zwischen verborgenen Zufallsvariablen und den Modellparametern, bei der die Modellparameter sowohl an die beobachteten als auch an die verborgenen Zufallsvariablen anzupassen sind, jedoch die Identität der verborgenen Zufallsvariablen nur geschätzt werden kann, wobei die Schätzung wiederum von den Modellparametern abhängt.

Im Spezialfall Baum-Welch ist es vor allem die Zustandssequenz durch das HMM, die nur aus den von den Zuständen emittierten Merkmalsvektoren geschätzt werden kann. Die Überlegung hinter dem Modell ist, dass jeder Merkmalsvektor von genau einer Mixturkomponente in einem Zustand des Modells emittiert wird. Die Modellparameter sind also so zu wählen, dass jede Mixturkomponente die zu ihr gehörenden Merkmalsvektoren bestmöglich beschreibt. Da allerdings keine eindeutige Zuordnung zu Zuständen, geschweige denn Mixturkomponenten, vorgegeben ist, kann diese Zuordnung ausschließlich aus dem vorliegenden Modell geschätzt werden.

Ein EM-Algorithmus löst diese Situation auf, indem er abwechselnd die beiden Seiten der Abhängigkeit fixiert. Aus diesen beiden Schritten ergibt sich auch der Name des Algorithmus.

Im *Expectation-Schritt* werden auf Basis einer Schätzung der Parameter Vermutungen über die Identität der korrekten Zustandsfolge angestellt. Die vorhandene Schätzung der Parameter wird als korrekt angenommen und auf ihrer Basis werden die Merkmalsvektoren Zuständen und Mixturkomponenten innerhalb der Zustände zugeordnet. Diese Zuordnung ist im Normalfall probabilistisch, einem Merkmalsvektor werden somit mehrere Mixturkomponenten in verschiedenen Zuständen des Modells mit unterschiedlicher Wahrscheinlichkeit zugeordnet.

Im darauffolgenden *Maximization-Schritt* wird die bestimmte Zuordnung als korrekt angenommen und liefert die Grundlage für eine Anpassung der Modellparameter. Die neuen Parameter werden so gewählt, dass jede Klasse die ihr zugeordneten Merkmalsvektoren optimal beschreibt, wobei die Zuordnungswahrscheinlichkeit als Gewichtung einfließt.

Es lässt sich beweisen, dass wenn sich die neue Schätzung von der alten unterscheidet, sich die Likelihood wie gewünscht verbessert. Auch wenn die Konvergenz zu einem globalem Optimum nicht garantiert werden kann, hat sich das Verfahren in der Praxis bewährt. Für eine tiefergehende Einführung in den EM Algorithmus im Allgemeinen sei an dieser Stelle auf [8] verwiesen, für das HMM Training mit dem Baum-Welch Algorithmus im Speziellen auf [28].

2.4.2 Maximum Mutual Information Estimation

In [1] wird unter dem Begriff *Maximum Mutual Information Estimation (MMIE)* eine Maximierung der *punktweisen Transinformation* $I(X, w_{id(X)})$ *(Pointwise Mutual Information)* als Optimierungskriterium vorgeschlagen. Dieser Wert beschreibt die Stärke des statistischen Zusammenhanges zwischen einem Trainingsbeispiel X und

der Transkription $w_{\mathrm{id}(X)}$ aus der Menge aller Transkriptionen W und ist wie folgt definiert:

$$\begin{aligned} I(X,w) &= \log \frac{p(X, w_{\mathrm{id}(X)})}{p(X)p(w_{\mathrm{id}(X)})} \\ &= \log \frac{p(X|\omega_{w_{\mathrm{id}(X)}})p(w_{\mathrm{id}(X)})}{\sum\limits_{w \in W} (p(X|\omega_w))p(w))\, p(w_{\mathrm{id}(X)})} \\ &= \log \frac{p(X|\omega_{w_{\mathrm{id}(X)}})}{\sum\limits_{w \in W} (p(X|\omega_w)p(w))} \\ &= \log \frac{p(X|\omega_{w_{\mathrm{id}(X)}})}{p(X)} \end{aligned}$$

Für die Bestimmung der bedingten Wahrscheinlichkeit $p(X|w)$ wird hier der Ausdruck $p(X|\omega_w)$ verwendet, in dem ω_w, das die Transkription w repräsentierende Modell bezeichnet. Die Optimierung dieser Funktion ist eng verwandt mit einer Optimierung der a posteriori Wahrscheinlichkeit $p(w_{\mathrm{id}(X)}|X)$, dass ein Trainingsbeispiel X von dem der Transkription $w_{\mathrm{id}(X)}$ zugeordneten Modell erzeugt wird. Daher findet sich oft auch folgendes Optimierungskriterium im Zusammenhang mit MMIE [26]:

$$\log p(w_{\mathrm{id}(X)}|X) = \log \frac{p(X|\omega_{w_{\mathrm{id}(X)}})p(w_{\mathrm{id}(X)})}{\sum\limits_{w \in W} (p(X|\omega_w)p(w))} = \log \frac{p(X|\omega_{w_{\mathrm{id}(X)}})p(w_{\mathrm{id}(X)})}{p(X)}$$

Sofern die Sprachmodellwahrscheinlichkeit nicht von den zu trainierenden Parametern abhängt, liefert die Optimierung beider Varianten einen äquivalenten Klassifikator.

Das primäre Unterscheidungsmerkmal gegenüber MLE ist die Berücksichtigung von $p(X)$ im Training. Auch bei der Nutzung von MLE wird zur Klassifikation die a posteriori Wahrscheinlichkeit $p(w|X)$ bestimmt. Da allerdings $p(X)$ für alle Klassen den gleichen Wert annimmt, kann es zu diesem Zeitpunkt vernachlässigt werden. Im Training dagegen gibt es einen Grund diesen Wert zu berücksichtigen.

Im Kontrast zu MMIE maximiert MLE nur die Likelihood $p(X|w_{\mathrm{id}(X)})$ für das der Transkription zugeordnete Modell, X zu erzeugen. MMIE optimiert das Verhältnis von $p(X|w_{\mathrm{id}(X)})$ zu $p(X)$. Das verbessert sich, wenn die Emissionswahrscheinlichkeit der korrekten Klasse für X steigt, aber auch, wenn die Emissionswahrscheinlichkeiten der falschen Klassen sinken und damit die gesamte Wahrscheinlichkeitsmasse $p(X)$ geringer wird.

Über alle Trainingsbeispiele im Trainingsdatensatz $\mathcal{O}$ summiert ergibt sich als Zielfunktion:

$$O_{\mathrm{MMIE}}(\Lambda) = \sum_{X \in \mathcal{O}} \log \frac{p\left(X|\omega_{w_{\mathrm{id}(X)}}\right) P\left(w_{\mathrm{id}(X)}\right)}{\sum_w p\left(X|\omega_w\right) P\left(w\right)}$$

Durch die Abhängigkeit von allen Klassen und nicht nur von $\omega_{\mathrm{id}(X)}$ steigt natürlich der Aufwand, und bei MLE verwendete Trainingsverfahren lassen sich nicht in der gleichen Form anwenden. In [7] wurde ein Neuschätzungsverfahren für rationale Zielfunktionen vorgestellt, das den Zusatzaufwand begrenzen soll. In [26] wird dieser *Extended-Baum-Welch-Algorithmus (EBW-Algorithmus)*, unter Zuhilfenahme einer Approximation der Normalverteilung durch eine Treppenfunktion, auch für eine Schätzung der Modellparameter von Gaußmischverteilungen beschrieben.

2.4.3 Minimum Classification Error

Das *Minimum-Classification-Error-Kriterium (MCE-Kriterium)* verfolgt einen naheliegenden Ansatz. Die Zielfunktion bei dieser Klasse von Verfahren ergibt sich direkt aus der Fehlerrate auf den Trainingsdaten. Die Annahme ist entsprechend, dass eine niedrige Fehlerrate auf dem Trainingsdatensatz sich auch in der Fehlerrate auf dem Testdatensatz widerspiegelt. Für die formale Definition einer Zielfunktion muss zunächst ein *Fehlklassifikationsmaß (Misclassification Measure)* definiert werden:

$$d'_{\mathrm{MCE}}(X,\Omega) := -g(X,\omega_{\mathrm{id}(X)}) + \max_{\omega\in\Omega,\omega\neq\omega_{\mathrm{id}(X)}} g(X,\omega)$$

In dieser Funktion bezeichnet Ω eine Menge von Modellen, zwischen denen entschieden werden soll. Das zu der Transkription gehörende Referenzmodell ist als $\omega_{\mathrm{id}(X)}$ bezeichnet. Mit $g(X,\omega)$ ist die Log-Likelihood gemeint, die bei der Anwendung des Viterbi-Algorithmus mit Modell ω auf ein Trainingsbeispiel X, erhalten wird. Ist $d'_{\mathrm{MCE}} < 0$, so wäre das Trainingsbeispiel X einer anderen Klasse als der durch die Transkription festgelegten Identität zugeordnet und damit falsch klassifiziert. Diese Fehlzuordnung über alle Trainingsbeispiele im Trainingsdatensatz zu zählen, würde bereits eine Zielfunktion liefern, jedoch wird für einige der verwendeten Optimierungsverfahren Differenzierbarkeit (insbesondere also Stetigkeit) vorausgesetzt. In der Praxis wendet man daher zumeist ein Fehlklassifikationsmaß an, das max durch die in einem späteren Abschnitt beschriebene *"soft-max" Funktion* (3.3) oder eine Variante davon [16] ersetzt:

$$d_{\mathrm{MCE}}(X,\Omega) := -g(X,\omega_{\mathrm{id}(X)}) + \frac{1}{\eta}\log\left(\frac{1}{|\Omega|-1}\sum_{\omega\in\Omega,\omega\neq\omega_{\mathrm{id}(X)}} \exp(\eta g(X,\omega))\right)$$

Weiterhin wird im Allgemeinen für die Verlustfunktion auch die Sigmoidfunktion $\sigma(x) := (1+\exp(-\alpha x))^{-1}$ der binären Entscheidung nach dem Vorzeichen vorgezogen, da diese einen stetigen Übergang an der Vorzeichengrenze liefert:

$$L_{\mathrm{MCE}} = \frac{1}{1+\exp(-\alpha d_{\mathrm{MCE}}(X,\Omega))}$$

Eine Summe dieser Verlustfunktion über alle Trainingsbeispiele liefert eine differenzierbare Zielfunktion. Zur Optimierung finden verschiedene Ansätze Verwendung. Frühe Arbeiten wenden vorwiegend Gradientenabstieg an [17]. In [18],[25] werden verschiedene Ansätze untersucht, die vor allem Varianten von Gradientenabstieg umfassen. Es wird auch die Anwendung des im Zusammenhang mit MMIE besprochenen EBW Algorithmus angesprochen.

2.4.4 Large/Maximum Margin HMM

Der Begriff "Large Margin" bzw. "Maximum Margin" lehnt sich an den Randbegriff bei Support Vector Machines an. Eine *Support Vector Machine (SVM)* führt eine binäre Klassifikation durch, indem sie den Merkmalsraum durch eine Hyperebene aufteilt. Das Training eines SVM hat es zum Ziel, die Hyperebene so zu legen, dass Trainingsbeispiele beider Klassen einen maximalen Abstand zu der Hyperebene haben. Davon verspricht man sich eine optimale Generalisierung bei der Übertragung

auf ungesehene Daten. Es ist dann von einem "Large Margin Classifier" bzw. "Maximum Margin Classifier" die Rede, da er als Optimierungskriterium hat diesen – als *"Margin" ("Rand")* bezeichneten – Mindestabstand zu einem Trainingsbeispiel zu maximieren (Abb. 2.3).

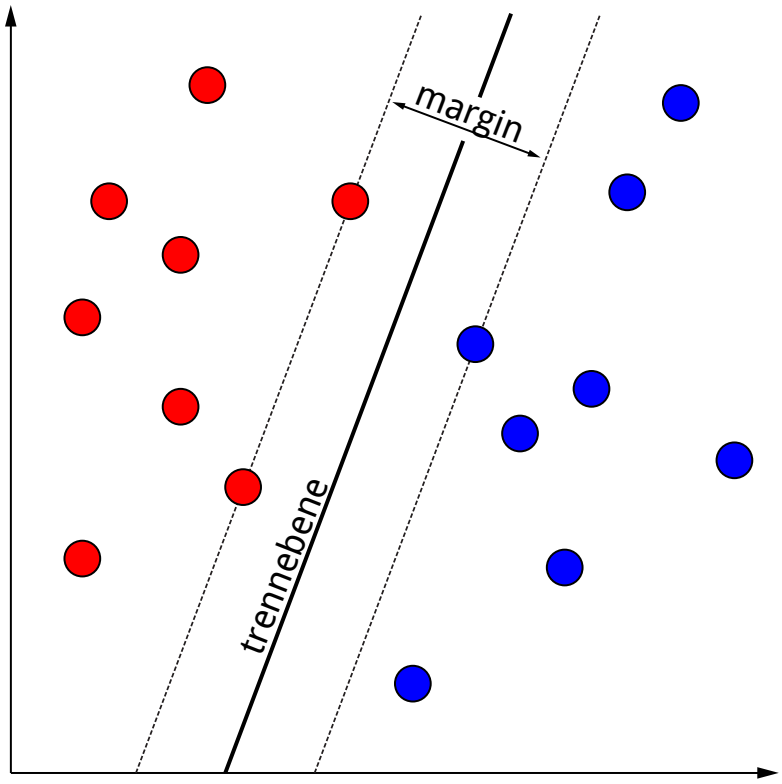


Abbildung 2.3: Beispieldarstellung eines zweidimensionalen Klassifikationsproblems, mit SVM Trennebene und hervorgehobener Margin

Bei der Übertragung dieses Prinzips auf Gaußmischverteilungen unterscheiden sich die Ansätze teils erheblich. Dieser Abschnitt gibt eine Übersicht über verschiedene Interpretationen des Begriffes. Im Detail wird die in dieser Arbeit implementierte Variante in Abschnitt 3 beschrieben.

Häufig wird anstelle der Distanz zu einer Trennebene eine, mit dem im Zusammenhang mit MCE beschriebenen Fehlklassifikationsmaß d_{MCE} vergleichbare, Distanzfunktion genutzt. Varianten unterscheiden sich hier vorwiegend darin, dass anstelle der Log-Likelihood $g(X, \omega)$ beispielsweise die Likelihood genutzt wird (Abb. 2.4). Der Begriff "Fehlklassifikationsmaß" hat im Zusammenhang mit MCE den Fehler hervorgehoben, durch die Sigmoidfunktion sollte das Maß vorwiegend auf das Vorzeichen reduziert werden. Der Marginbegriff dagegen sieht bei korrekter Klassifikation in dieser Differenz einen Spielraum für Abweichungen der Bewertung. Dieser Vorsprung, den das Referenzmodell vor konkurrierenden Modellen hat, wird damit als Maß für Robustheit betrachtet.

Eine solche Definition findet sich als (Separation) Margin auch in einer Reihe von Arbeiten beginnend mit [12]. Für ein Trainingsbeispiel wird in dieser Arbeit folgende Definition verwendet:

$$d_{\mathrm{LME}_1}(X, \Omega) := p(w_{\mathrm{id}(X)})p(X|\omega_{w_{\mathrm{id}(X)}}) - \max_{w \in W, w \neq w_{\mathrm{id}(X)}} \{p(w)p(X, \omega_w)\}$$

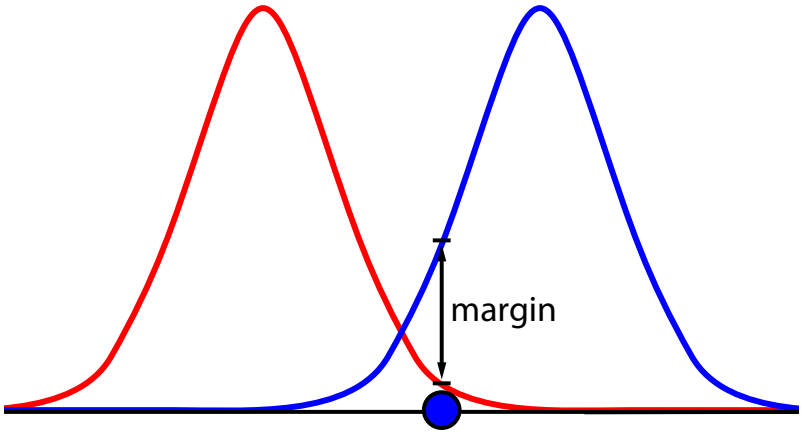


Abbildung 2.4: Einfache Darstellung des Marginbegriffes für zwei Klassen (modelliert durch Normalverteilungen) und ein Trainingsbeispiel.

Es wird somit die durch das Sprachmodell gewichtete Likelihood einer Referenztranskription mit der besten Konkurrenztranskription verglichen. Ein wesentlicher Unterschied zu d_{MCE} wäre die Nutzung der Likelihood anstelle der Log-Likelihood. Es wird aber in späteren Arbeiten (beispielsweise [21]) auch die Nutzung der Log-Likelihood angesprochen.

Die Eigenschaft der Margin als Spielraum gegen Abweichungen wird auch bei der Wahl der Trainingsbeispiele hervorgehoben. Auf dem Trainingsdatensatz wird bereits in [12] zunächst eine Teilmenge von, als "Stützvektoren" bezeichneten, Trainingsbeispielen bestimmt. Für das eigentliche Large-Margin-Training werden in dieser und einem Großteil der folgenden Arbeiten nur die Beispiele in dieser Stützvektormenge berücksichtigt. Im Zusammenhang mit Support Vector Machines werden so Trainingsbeispiele genannt, die durch ihre Position die Lage der Trennebene im Raum und den Rand bestimmen. Nach dem Training liegen sie am nächsten an der Trennebene. Hier werden in diese Menge nur Beispiele aufgenommen, deren Margin positiv ist – die also richtig klassifiziert werden würden – aber einen durch einen Trainingsparameter festgelegten Wert nicht übersteigt. Das Optimierungsziel ist nun die minimale Margin über alle Stützvektoren zu maximieren. Diese Einschränkung reduziert allerdings den Trainingsdatensatz. Um diesen Effekt zu beschränken wird in [12] vorgeschlagen, durch Anwendung von MCE auf fehlerhaft klassifizierte Trainingsbeispiele zunächst die Stützvektormenge zu maximieren. Es wird in [21] die Möglichkeit angesprochen alle Parameter der Gaußmischverteilungen zu trainieren, jedoch beschränken sich alle Arbeiten dieser Gruppe darauf die Mittelwerte neu zu schätzen.

In [20] wird auf ein Problem am Kriterium hingewiesen, das eine Optimierung ohne Veränderung der Klassifikationsgrenze ermöglicht. Als Lösung wird in dieser Arbeit vorgeschlagen, stets nur den Mittelwert mit der niedrigsten Margin zu aktualisieren. Parallel wird in [21] eine Abwandlung des Kriteriums untersucht, um dem gleichen Problem entgegenzusteuern. Die Bewertungsdifferenz der beiden Modelle wird durch die Bewertung des Referenzmodells dividiert und so normalisiert. Diese relative Bewertungsdifferenz wird als "relative Margin" bezeichnet. Ein weiterer, in [14] beschriebener Ansatz untersucht die Verwendung von Straftermen in der Zielfunktion.

Auf kontinuierliche Sprache wird in [22] das auf einer relativen Margin basierende Verfahren übertragen. Während die vorhergehenden Arbeiten für die Bestimmung der Margin das Minimum aller Konkurrenzmodelle verwenden, ist dieses Vorgehen im Falle kontinuierlicher Spracherkennung nicht anwendbar. Es müssten dafür alle möglichen Sätze ausgewertet werden, was bereits bei dem in dieser Arbeit verwendeten Vokabular von elf Wörtern sehr aufwendig wird. Es wird daher als Approximation eine Liste der am besten bewerteten Wortfolgen verwendet. Auch hier werden nur Wortfolgen genutzt, die der Klassifikator korrekt erkennen würde. Ein MCE-Schritt soll zunächst die Anzahl für das Training geeigneter Wortfolgen erhöhen.Weitere Verbesserungen an diesem Algorithmus werden in [23] untersucht. Hier wird zum einen ein bereits in ersten Untersuchungen angesprochener Ansatz weiterverfolgt auf falsch klassifizierte Trainingsbeispiele das MCE Kriterium anzuwenden und zum anderen wird getestet die grundsätzliche Beschränkung auf richtig klassifizierte Trainingsbeispiele aufzugeben.

Alle bis zu diesem Punkt beschriebenen Arbeiten nutzen ein Gradientenabstiegsverfahren zur Optimierung der Zielfunktion. In [19] wird die Zielfunktion als Instanz Semidefiniter Programmierung (SDP) neu formuliert. Während in diesem Fall wieder nur korrekt klassifizierte Trainingsbeispiele genutzt werden, wird in [13] eine als soft-LME bezeichnete Erweiterung vorgenommen, die auch falsch klassifizierte Trainingsbeispiele nutzt. Ein Problem stellen allerdings für das Training auf Basis von SDP Modelle mit einer großen Anzahl von Gaußglocken dar, da sie Operation auf großen Matrizen erfordern. In [38] und [37] werden verschiedene Möglichkeiten untersucht die Anwendbarkeit zu verbessern.

Eine andere Herangehensweise an die Optimierung einer Margin wird durch eine Forschergruppe um Fei Sha und Lawrence K. Saul in [33] beschrieben. Zunächst wird eine alternative Formulierung des Klassifikationsproblems auf der Basis von Gaußmischverteilungen definiert. Für die Parameter (μ, Σ, θ) einer Normalverteilung wird eine Matrix

$$\Phi := \begin{pmatrix} \Sigma & -\Sigma\mu \\ -\mu^\intercal\Sigma & \mu^\intercal\Sigma\mu + \theta \end{pmatrix}$$

definiert. Damit kann eine Entscheidung der Form

$$y = \operatorname*{argmin}_c \{(x - \mu_c)^\intercal \Sigma (x - \mu_c) + \theta_c\}$$

zwischen den – durch jeweils eine Normalverteilung modellierten – Klassen c auch durch

$$y = \operatorname*{argmin}_c \{z^\intercal \Phi_c z\} \text{ mit } z = \begin{pmatrix} x \\ 1 \end{pmatrix}$$

beschrieben werden. Das liefert somit eine Darstellung mit der alle Parameter gleichzeitig optimiert werden können.

Mit Hilfe dieser Darstellung wird zunächst für einzelne Merkmalsvektoren eine Verlustfunktion wie folgt bestimmt:

$$L_{\mathrm{LME}_2} = \gamma \sum_x \sum_{c \neq c_{\mathrm{id}(x)}} \left[1 + x\left(\Phi_{\mathrm{id}(X)} - \Phi_c\right)\right]_+ + \sum_c \mathrm{Spur}\left(\Sigma_c\right)$$

Im Zusammenhang mit Large-Margin-Verfahren ist der erste Term dieser Verlustfunktion von Bedeutung, der zweite Term, der durch den Parameter γ relativ gewichtet wird, dient der Regulierung. Die dem Merkmalsvektor zugeordnete Klasse –

hier repräsentiert durch $\Phi_{\mathrm{id}(X)}$ – erhält einen Strafterm von 1 auf die Bewertung gegenüber den Konkurrenzklassen Φ_c. Der Ausdruck $[f]_+ := \max\{0, f\}$ bedeutet, dass ausschließlich die Merkmalsvektoren berücksichtigt werden, die das Referenzmodell nach Abzug des Straftermes schlechter bewertet, als ein Konkurrenzmodell. Auf diese Weise soll bei der Optimierung ein Bewertungsspielraum erzwungen werden, der als Margin bezeichnet wird.

In [34] wird dieses Vorgehen für Beobachtungsfolgen erweitert. Einem Referenzmodell von HMM-Zuständen wird eine Menge von Konkurrenzmodellen gegenüber gestellt. Dafür wird die Begrenzung der Verlustfunktion gelockert, die Bewertung der gesamten Sequenz durch das Referenzmodell soll die Bewertungen der Konkurrenzmodelle um die Hammingdistanz zu den zugehörigen Zustandsfolgen übersteigen. Es ist eine natürliche Erweiterung, da die ursprüngliche Definition gerade bei den Fenstern die Referenzbewertung um einen Strafterm von 1 verschlechtern würde, in denen durch das Konkurrenzmodell eine andere Zuordnung stattfindet. Die Lockerung besteht darin, dass im Fall von ganzen Folgen auch Fenster berücksichtigt werden, in denen das Referenzmodell das Konkurrenzmodell um mehr als den Strafterm übertrifft.

In [35] wird der Ansatz mit der Optimierung von Conditional Maximum Likelihood und Minimum Classification Error verglichen. Conditional Maximum Likelihood bezeichnet dabei einen Ansatz, der weitgehend zu MMIE äquivalent ist. In [3] wird das Verfahren weiter verfeinert, unter anderem wird ein Parameter eingeführt, der die Margin skaliert.

Eine in [27] als "Boosted MMIE" vorgestellte Erweiterung von MMIE führt ebenfalls einen Term ein, der die Bewertung von Konkurrenzmodellen in Abhängigkeit von der Fehleranzahl in der Zustandssequenz verbessert. Dieses Vorgehen wird als Optimierung einer "soft Margin" bezeichnet und in [30] direkt mit dem auf der Hammingdistanz basierten Ansatz verglichen.

Ebenfalls mit der Einführung eines als Margin bezeichneten Straftermes auf die korrekte Bewertung im Fehlklassifikationsmaß befasst sich [9]. Es handelt sich in diesem Fall jedoch explizit um eine Variante von MCE. So wird die Sigmoidfunktion genutzt, wodurch größere Schwankungen jenseits der Klassifikationsgrenze in beiden Richtungen geglättet werden. Unter dem Begriff "Margin" wird hier ein konstanter Strafterm als Trainingsparameter eingeführt, der beim Training der Modelle zu kompensieren ist. Für die Optimierung wird die Zielfunktion von MCE neu formuliert, was eine Anwendung des in Abschnitt 2.4.2 beschriebenen Extended-Baum-Welch-Algorithmus ermöglicht. In einer weiterführenden Arbeit [39] wird das Verfahren auch auf größeren Datenmengen erprobt.

Kapitel 3

Mathematische Grundlagen des implementierten Verfahrens

Die Grundlage für das hier verwendete Verfahren liefern die Arbeiten [14] und [23]. Dieses Kapitel beschreibt die aus den Referenzarbeiten übernommenen Ansätze und geht auf die mathematischen Überlegungen ein, die der Implementierung zugrunde liegen sollen.

3.1 Marginbegriff

Eine erste Definition für eine Margin ist mit

$$d(\omega_{\mathrm{id}(X)}, \omega_i, X)_{\mathrm{abs}} := \mathcal{F}\left(X|\omega_{\mathrm{id}(X)}\right) - \mathcal{F}\left(X|\omega_i\right) \tag{3.1}$$

gegeben [12]. Dabei ist $\mathcal{F}\left(X|\omega\right)$ die Bewertung für ein Trainingsbeispiel X durch das Modell ω. Diese Bewertungsdifferenz zwischen zwei Modellen liefert die Basis für den Begriff einer absoluten Margin. Es wird in den folgenden Überlegungen davon ausgegangen, dass für die Modellierung kontinuierliche HMMs verwendet werden.

In [23] wird eine andere Formulierung verwendet, die eine sogenannte *relative Margin* für zwei Modelle $\omega_{\mathrm{id}(X)}$ und ω_i wie folgt definiert:

$$d(\omega_{\mathrm{id}(X)}, \omega_i, X) := 1 - \frac{\mathcal{F}\left(X|\omega_i\right)}{\mathcal{F}\left(X|\omega_{\mathrm{id}(X)}\right)} \tag{3.2}$$

Diese Distanzfunktion soll garantieren, dass die Margin nach oben beschränkt ist, so dass ein Optimum gefunden werden kann. In [20] wird argumentiert, dass sonst die Zielfunktion ohne echte Verbesserung der Diskriminierung beliebig verbessert werden kann. Das in diesem Zusammenhang an einem Beispiel beschriebene Verhalten, ließ sich durch eigene Berechnungen in der gleichen Form nicht bestätigen, sehr wohl allerdings, dass abhängig von der Wahl der Marginfunktion die Margin unbeschränkt wachsen kann. Weiterhin muss aber beachtet werden, dass auch die Entscheidung zwischen Likelihood und der – für automatische Spracherkennung häufig verwendeten – logarithmierten Likelihood als Bewertungsfunktion $\mathcal{F}\left(X|\omega\right)$ eine Rolle spielt. In [21] wird dieser Unterschied zwar berücksichtigt, aber nur im Hinblick auf den Vorzeichenwechsel durch Logarithmierung der Likelihood.

Um dies zu verdeutlichen, lässt sich analog zu [20] ein einfacher Fall mit zwei durch jeweils eine eindimensionale Gaußglocke modellierten Klassen konstruieren. Man wählt die Mittelwerte so, dass sie sich nur durch das Vorzeichen unterscheiden. Für die beiden Mittelwerte μ_1 und μ_2 gilt also $\mu_1 = -\mu_2$. Die Kovarianz wählt man gleich. Als Wert der Kovarianz σ soll im Folgenden 1 angenommen werden. Die Entscheidungsgrenze ist in diesem Fall also stets bei 0. Auch wenn die beiden Mittelwerte gleichermaßen verschoben werden, bleibt die Entscheidungsgrenze unverändert. Nun bestimmt man für verschiedene Mittelwerte die Margin zu einem fest gewählten Trainingsbeispiel. In diesem Fall wurde für das Trainingsbeispiel der Wert 1 gewählt, da sich daran das Verhalten besser verdeutlichen lässt als für einen Wert, der näher an der Entscheidungsgrenze liegt.

Die beiden Möglichkeiten die Bewertungsfunktion zu wählen lassen sich in diesem Fall vereinfachen zu der Likelihood:

$$\mathcal{F}\left(X|\omega\right)_{\mathrm{1D}} := \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(x-\mu\right)^2\right)$$

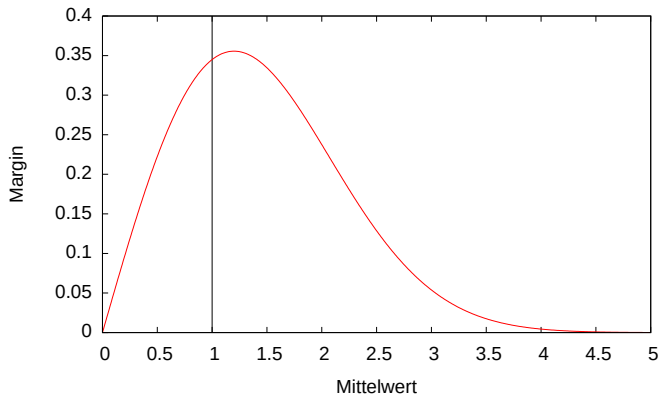
und der logarithmierten Likelihood:

$$\mathcal{F}\left(X|\omega\right)_{\mathrm{1D,log}} := \log\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)\right) = \log\left(\frac{1}{\sqrt{2\pi}}\right) - \frac{1}{2}\left(x-\mu\right)^2$$

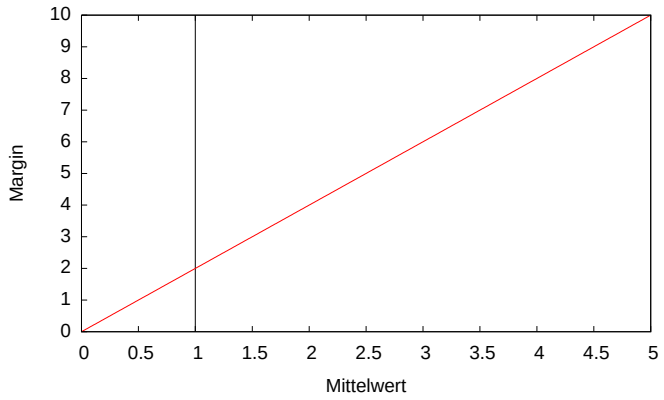
Zeichnet man den Wert der Margin nun in Abhängigkeit von der Wahl der Mittelwerte auf, so ist direkt zu erkennen (Abb. 3.1b), dass sich tatsächlich die Kombination von absoluter Margin (vgl. Gleichung (3.1)) und logarithmierten Likelihood linear verhält und damit nicht beschränkt ist. Das liegt daran, dass eine Logarithmierung den beschränkenden Einfluss der Exponentialfunktion in der Normalverteilung aufhebt und die Margin in diesem einfachen Beispiel nur noch linear mit den Mittelwerten wächst. Da die Likelihood dagegen beschränkt ist, gibt es im absoluten Fall ein Optimum (Abb. 3.1a).

Bei Nutzung einer relativen Margin beschränkt die Normierung durch die Bewertung des Referenzmodells (siehe Gleichung (3.2)) den Wert der Funktion sowohl bei Verwendung der Likelihood, als auch bei Verwendung einer logarithmierten Likelihood als Bewertungsfunktion (Abb. 3.1c und Abb. 3.1d). Bei der Nutzung der Likelihood fällt auf, dass durch das schnelle Wachstum der Exponentialfunktion das Verhältnis der Bewertungen zueinander ebenso schnell relativ zur Bewertung durch das Referenzmodell gering ausfällt. Daher konvergiert die Funktion gegen 1. Damit ist sie zwar beschränkt, weist aber kein Optimum auf, das ohne Verschiebung der Entscheidungsgrenze erreicht werden kann. Die Konvergenzgeschwindigkeit wird vom Abstand des Trainingsbeispieles von der Entscheidungsgrenze beeinflusst. Für weiter von der Entscheidungsgrenze entfernte Trainingsbeispiele ist die Konvergenz schneller, da hier der Bewertungsunterschied stärker ausfällt. Durch Nutzen logarithmierter Likelihood erhält man hingegen wieder ein ausgeprägtes Optimum.

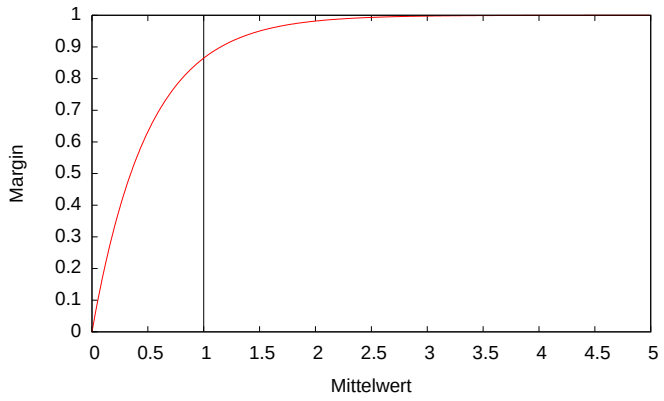
Die vorhergehenden Betrachtungen legen nahe entweder eine absolute Margin auf Likelihoods, oder eine relative Margin auf logarithmierten Likelihoods zu verwenden, möchte man vermeiden, dass selbst ohne eine Anpassung der Modelle das Auseinanderdriften der Mittelwerte zu einer kontinuierlichen Verbesserung der Margin führt. Da in den referenzierten Arbeiten von einer erfolgreichen Anwendung der relativen Margin berichtet wird, fiel die Entscheidung letztlich darauf.



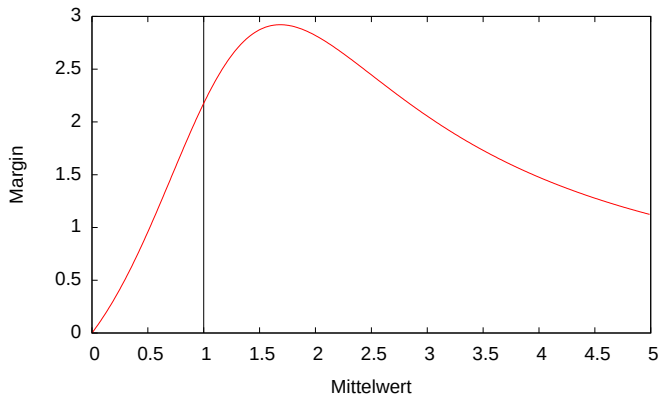
(a) Absolute Margin, Likelihood



(b) Absolute Margin, logarithmierte Likelihood



(c) Relative Margin, Likelihood



(d) Relative Margin, logarithmierte Likelihood

Abbildung 3.1: Margin für eine eindimensionale Konfiguration mit konstanter Entscheidungsgrenze

3.2 Zielfunktion

Die Distanzfunktion aus (3.2) gibt Auskunft darüber, wie viel Spielraum das Modell $\omega_{\mathrm{id}(X)}$ hat, bevor es das Trainingsbeispiel X schlechter bewertet als das Konkurrenzmodell ω_i. Eine Optimierung der Distanzfunktion wird abhängig von der Wahl der Bewertungsfunktion $\mathcal{F}(X|\omega)$ durch Minimierung oder Maximierung erreicht. Da aber ein Vorzeichenwechsel der Funktion reicht, um die Richtung umzukehren, kann im Folgenden von einer Minimierung ausgegangen werden.

Für eine Zielfunktion ist eine Bewertung des Gesamtsystems nötig. Eine solche Bewertung lässt sich aus dem "worst case" ableiten, also dem Paar von Modellen mit dem geringsten Spielraum bei der Bewertung. Durchführen lässt sich diese Erweiterung in zwei Schritten. Zunächst wird die für ein spezielles Tupel von Referenz- und Konkurrenzmodell sowie einem Trainingsbeispiel definierte Funktion auf alle Konkurrenzmodelle zu diesem Trainingsbeispiel ausgeweitet:

$$d(\Omega, X) := \max_{\omega \in \Omega, \omega \neq \omega_{\mathrm{id}(X)}} d(\omega_{\mathrm{id}(X)}, \omega, X)$$

wobei X beim Training Modell $\omega_{\mathrm{id}(X)}$ zugeordnet werden soll und Ω die Menge aller Modelle sei. Um das System als Ganzes zu bewerten, lässt sich die Funktion analog für alle verwendeten Trainingsdaten S erweitern:

$$d(\Omega, S) := \max_{X \in S} d(\Omega, X)$$

Insgesamt stellt somit $d(\Omega, S)$ den geringsten Vorsprung dar, den ein Referenzmodell vor einem Konkurrenzmodell auf einem der Trainingsbeispiele hat.

Viele Verfahren für die Suche nach Minima erfordern eine differenzierbare Funktion, so auch das hier eingesetzte Gradientenabstiegsverfahren. Da max nicht stetig ist, trifft es auf $d(\Omega, S)$ nicht zu, was Anpassungen erfordert. Im Folgenden wird daher für die Zielfunktion max durch eine differenzierbare Approximation ersetzt, die für eine Menge M folgendermaßen definiert ist:

$$s(M) := \frac{1}{\eta} \log\left(\sum_{m \in M} \exp(\eta m)\right) \tag{3.3}$$

Diese wegen der glättenden Eigenschaften als "soft-max" bezeichnete Funktion lässt sich durch

$$\frac{1}{\eta} \log\left((|M| - 1) \cdot 0 + \exp(\eta \max M)\right) = \max M$$

nach unten und mit

$$\frac{1}{\eta} \log\left(|M| \exp(\eta \max M)\right) = \frac{1}{\eta} \log |M| + \max M$$

nach oben abschätzen. Daraus folgt für eine Menge M:

$$\lim_{\eta \to \infty} s(M) = \max M$$

Weiterhin zeigt diese Abschätzung den Spielraum für die Abweichung vom tatsächlichen Maximum. Die untere Abschätzung zeigt, dass das Maximum nicht unterschätzt wird. Die maximale Überschätzung ist aus der oberen Schranke zu erkennen. Wie

sehr das Maximum überschätzt wird, hängt von der Varianz innerhalb der Menge ab. Je näher die Elemente am Maximum liegen, desto näher kommt die Funktion der Abschätzung. Die obere Schranke wird so genau dann angenommen, wenn alle Elemente den gleichen Wert haben.

Ein großes η zu wählen ist aus numerischen Gründen nicht praktikabel, so dass die Zielfunktion durch die Überschätzung geglättet wird. Der Effekt auf die Zielfunktion ist für einen einfachen Fall in Abbildung 3.2 zu sehen. Der dargestellte Merkmalsraum ist eindimensional und es gibt zwei durch Normalverteilungen mit Mittelpunkten μ_1 und μ_2 dargestellte Klassen. Das erlaubt es, die Zielfunktion als Funktion $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$ der beiden Mittelwerte aufzuzeichnen. Die Werte dieser Funktion werden durch einen Farbverlauf dargestellt, durch den die Topologie des Suchraumes sichtbar wird.

Es ist zu erkennen, dass ein höherer Wert von η die Topologie zwar nicht wesentlich beeinflusst, Konturen allerdings deutlicher hervorhebt, was im Falle von Gradientenabstieg auch Nachteile haben kann. Um das zu visualisieren, enthält die Abbildung ebenfalls die für das Large-Margin-Training bestimmten Gradienten für beide Mittelwerte. Diese Gradienten sind in der Abbildung durch Pfeile dargestellt, wobei Richtung und Größe der Pfeile Betrag und Richtung für die Änderung der Mittelwerte darstellen. So ist in der Abbildung für ein η von 32 der Abstieg rund um das Minimum sehr stark, was die Annäherung erschweren kann. Im Kontrast dazu sind die Gradienten nahe am Minimum in der Abbildung mit einem η von 2 wieder weniger steil, was die Annäherung erleichtern könnte.

Insgesamt ergibt sich als Zielfunktion

$$O(\Omega) := \frac{1}{\eta} \log \left(\sum_{X \in S, \omega \in \Omega, \omega \neq \omega_{\mathrm{id}(X)}} \exp\left(\eta d(\omega_{\mathrm{id}(X)}, \omega, X)\right) \right)$$

Es wird im Folgenden stets angenommen, dass Bewertungen durch Auswerten von Gaußmischverteilungen, gehörend zu Zuständen eines Hidden Markov Modells, entstehen. Die Parameter der k-ten Mixturkomponente von Zustand s des Hidden Markov Modelles m werden mit μ^m_{skl} für den Mittelwert und σ^m_{skl} für die Kovarianz bezeichnet. Der Index l bezeichnet den Dimensionsindex, da jede Dimension unabhängig betrachtet werden kann und wird. Es werden insbesondere diagonale Kovarianzmatrizen angenommen, so dass σ wie ein Vektor behandelt werden kann.

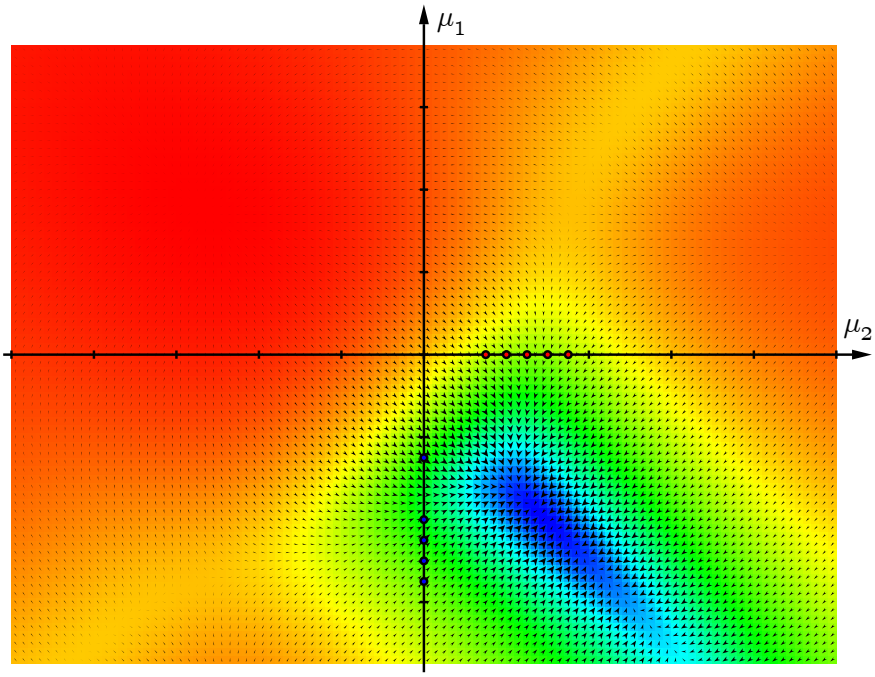
3.3 Optimierung

Das hier beschriebene Verfahren ändert ausschließlich die Mittelwerte der Gaußmischverteilungen aller Modelle. Die Kovarianzmatrizen behalten ihre ursprünglichen Werte. Für die Anwendung des Algorithmus werden die Mittelwerte der Gaußmischverteilungen zunächst wie folgt normalisiert [21]:

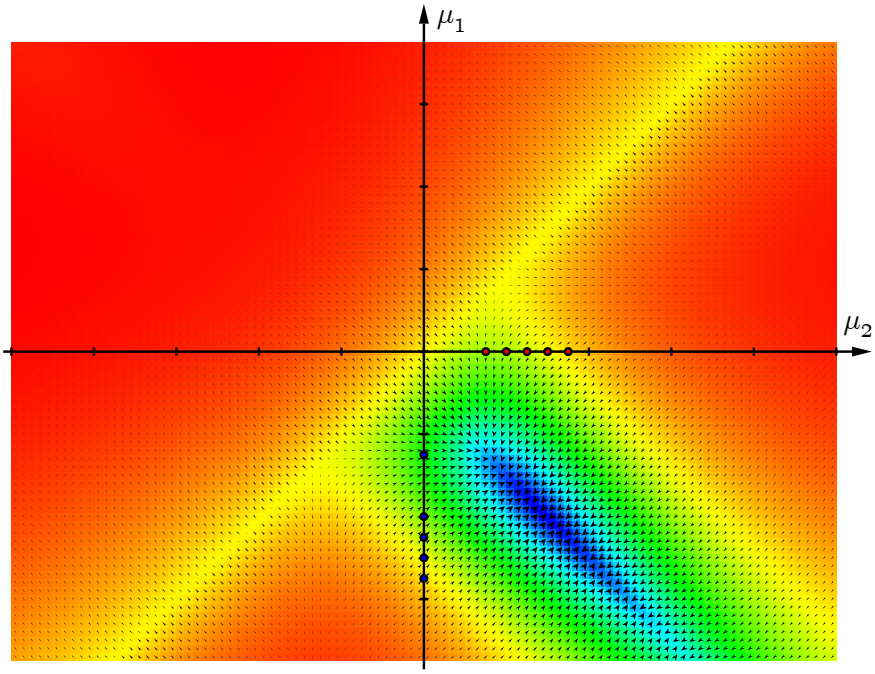
$$\tilde{\mu}^m_{skl} := \mu^m_{skl} / \sigma^m_{skl}$$

Als Verfahren zur Optimierung wird Gradientenabstieg verwendet. Dafür wird auf jedes $\tilde{\mu}^m_{skl}$ die Aktualisierungsfunktion

$$\tilde{\mu}^m_{skl}(i+1) := \tilde{\mu}^m_{skl}(i) - \epsilon \frac{\partial O(\omega)}{\partial \tilde{\mu}^m_{skl}} \tag{3.4}$$



(a) $\eta = 2$



(b) $\eta = 32$

Abbildung 3.2: Relative Zielfunktion in eindimensionalem Merkmalsraum als Heatmap mit Gradient. Aufgezeichnet als Funktion von zwei Mittelwerten μ_1 und μ_2. Die für die Berechnung verwendeten Trainingsbeispiele sind auf den Achsen markiert.

angewandt. Der Parameter ϵ wird als *Schrittgröße* bezeichnet und muss in einer Abwägung zwischen Konvergenzgeschwindigkeit und Präzision des Gradientenabstiegsverfahrens geeignet gewählt werden.

Die partielle Ableitung der Zielfunktion $\frac{\partial O(\omega)}{\partial \tilde{\mu}_{skl}}$ nach einem der Mittelwerte macht den Kern des Algorithmus aus. Für jeden Mittelwert wird die Ableitung wie folgt aufgelöst:

$$\frac{\partial O(\Omega)}{\partial \tilde{\mu}^m_{skl}} = \frac{1}{\eta}\frac{1}{Q(\Omega)}\frac{\partial Q(\Omega)}{\tilde{\mu}^m_{skl}} \tag{3.5}$$

wobei

$$Q(\Omega) := \sum_{X\in S, j\neq \mathrm{id}(X)} \exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \tag{3.6}$$

gilt. Es folgt dann

$$\begin{aligned}
\frac{\partial Q(\Omega)}{\partial \tilde{\mu}^m_{skl}} &= \sum_{X\in S, j\neq \mathrm{id}(X)} \frac{\partial \exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X))}{\partial \tilde{\mu}^m_{skl}} \\
&= \sum_{X\in S, j\neq \mathrm{id}(X)} \exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \frac{\partial(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X))}{\partial \tilde{\mu}^m_{skl}} \\
&= \sum_{X\in S, j\neq \mathrm{id}(X)} \exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \frac{\partial \left[\frac{\mathcal{F}(X,\omega_{\mathrm{id}(X)})}{\mathcal{F}(X,\omega_j)}\eta - \eta\right]}{\partial \tilde{\mu}^m_{skl}} \\
&= \eta \sum_{X\in S} \sum_{j\neq \mathrm{id}(X)} \exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \frac{1}{\mathcal{F}(X,\omega_j)^2} \\
&\quad \left(\frac{\partial \mathcal{F}(X,\omega_{\mathrm{id}(X)})}{\partial \tilde{\mu}^m_{skl}}\mathcal{F}(X,\omega_j) - \frac{\partial \mathcal{F}(X,\omega_j)}{\partial \tilde{\mu}^m_{skl}}\mathcal{F}(X,\omega_{\mathrm{id}(X)})\right)
\end{aligned}$$

Eine Umformung stellt eine Basis für die Implementierung dar:

$$\begin{aligned}
\frac{\partial Q(\Omega)}{\partial \tilde{\mu}^m_{skl}} =& \eta \sum_{X\in S} \left(\sum_{j\neq \mathrm{id}(X)} \exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \frac{1}{\mathcal{F}(X,\omega_j)} \frac{\partial \mathcal{F}(X,\omega_{\mathrm{id}(X)})}{\partial \tilde{\mu}^m_{skl}} \right. \\
&\left. - \sum_{j\neq \mathrm{id}(X)} \exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \frac{\mathcal{F}(X,\omega_{\mathrm{id}(X)})}{\mathcal{F}(X,\omega_j)^2} \frac{\partial \mathcal{F}(X,\omega_j)}{\partial \tilde{\mu}^m_{skl}} \right)
\end{aligned} \tag{3.7}$$

Es ist zu beachten, dass das aus der Summe gezogene η den Faktor in (3.5) aufhebt.

Gehört ein $\tilde{\mu}^m_{skl}$ zu genau einem Modell ω_m, so ist folgende Vereinfachung möglich:

$$\begin{aligned}
\frac{\partial Q(\Omega)}{\partial \tilde{\mu}^m_{skl}} = \eta \sum_{X\in S} \Bigg(& \delta\left(m - \mathrm{id}(X)\right) \left(\sum_{j\neq \mathrm{id}(X)} \exp(\eta \cdot d(\omega_m, \omega_j, X)) \frac{1}{\mathcal{F}(X,\omega_j)} \frac{\partial \mathcal{F}(X,\omega_m)}{\partial \tilde{\mu}^m_{skl}} \right) \\
& - \left(1 - \delta(m - \mathrm{id}(X))\right) \left(\exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_m, X)) \frac{\mathcal{F}(X,\omega_{\mathrm{id}(X)})}{\mathcal{F}(X,\omega_m)^2} \frac{\partial \mathcal{F}(X,\omega_m)}{\partial \tilde{\mu}^m_{skl}} \right) \Bigg)
\end{aligned} \tag{3.8}$$

Dabei ist δ das Diracdelta. Es wird ausgenutzt, dass in diesem Fall bei Auswertung anderer Modelle als ω_m der Mittelwert $\tilde{\mu}_{skl}^m$ nicht vorkommt. So bleiben nur die Fälle, in denen $\mathcal{F}(X, \omega_m)$ abgeleitet wird.

Es fehlt weiterhin noch die Ableitung von $\mathcal{F}(X, \omega)$ nach einem Mittelwert $\tilde{\mu}_{skl}^m$. Hier wird zur Vereinfachung die Annahme getroffen, dass die Emissionswahrscheinlichkeit für das Fenster x_t und einen Zustand im Hidden Markov Modell immer von einer Komponente der zugrundeliegenden Gaußmischverteilung dominiert wird (vgl. [14], [35]). Berücksichtigt man aufgrund dieser Annahme bei der Ableitung nur die Komponente mit dem größten Beitrag, so erhält man:

$$\frac{\partial \mathcal{F}(X, \omega)}{\partial \tilde{\mu}_{skl}^m} = \sum_{t=1}^{|X|} \delta(s_t - s) \frac{x_{tl} - \tilde{\mu}_{skl}^m}{\sigma_{skl}^m} \tag{3.9}$$

Damit liegt alles vor, was für die Implementierung der in Gleichung (3.4) beschriebenen Aktualisierungsvorschrift benötigt wird. Das Large-Margin-Training besteht nun aus der iterativen Anwendung dieser Vorschrift auf die Mittelwerte aller Gaußmischverteilungen der Modelle.

Kapitel 4

Implementierung

In diesem Kapitel wird die Umsetzung des Large-Margin-Trainings im Rahmen des Janus Recognition Toolkit (JRTk) beschrieben. Zunächst stellt dafür Abschnitt 4.1 das JRTk vor. Abschnitt 4.2 beschreibt die Verwendung der für diese Arbeit implementierten Komponente über die Tcl-Schnittstelle des JRTk. Die folgenden Abschnitte beschäftigen sich mit den internen Aspekten der Implementierung. In Abschnitt 4.3 wird die allgemeine Quellcodestruktur erläutert. Speziell wurden zwei Varianten implementiert das Training durchzuführen, die sich darin unterscheiden auf welchen Einheiten sie operieren. Eine Möglichkeit stellt das Training auf einzelnen Zuständen als Einheit dar und die andere auf Modellen ganzer Äußerungen. Abschnitt 4.4 befasst sich sich mit der Umsetzung des Trainings auf Fensterebene und Abschnitt 4.5 mit der Umsetzung für ganze Äußerungen.

4.1 Das JANUS Recognition Toolkit

Entwickelt wurde das JANUS Recognition Toolkit (JRTk) in einer Zusammenarbeit von KIT (früher Universität Karlsruhe) und der Carnegie Mellon University in Pittsburgh, USA [5].

Das JRTk enthält verschiedene Komponenten eines Spracherkenners, unter anderem einen One-Pass-Decoder unter dem Namen IBIS [36]. Diese zentralen Komponenten sind aus Effizienzgründen in objektorientiertem C-Code implementiert. Ein Tcl-Interpreter liefert eine Schnittstelle, über die aus diesen Komponenten ein Spracherkenner aufgebaut werden kann. Dieser modulare Aufbau erlaubt eine flexible Konfiguration, die speziell in der Forschung Vorteile hat. Datenquellen für die Modelle sind austauschbar und auch die Abfolge der Schritte beim Training lässt sich anpassen.

Dieser Abschnitt befasst sich zunächst mit dem MLE-Training, da viele im Zusammenhang damit verwendete Objekte auch beim Large-Margin-Training eine wichtige Rolle spielen und der Large-Margin-Trainingsschritt vorwiegend die gleiche Konfiguration erfordert. Der Abschnitt orientiert sich strukturell dabei an der Einführung in den Aufbau eines Spracherkennungssystems aus Abschnitt 2.3. Eine detaillierte Dokumentation liegt in [6] vor.

4.1.1 Merkmalsextraktion

Die Vorverarbeitung im JRTk beginnt mit der Organisation der Quelldaten in einer Datenbank. Über ein Objekt vom Typ `DBase` können für einzelne Äußerungen Sprecherinformationen, Transkription und Pfad zu der Signalaufzeichnung durch Aufruf der Funktion `uttInfo` abgerufen werden. Über ein `FeatureSet` wird der Zugriff auf die zu einer Äußerung gehörenden Signalaufzeichnungen geregelt. Dieses Objekt legt fest, wie aus dem Datenbankeintrag einer Äußerung die Aufzeichnung zu laden ist und welche Vorverarbeitungsschritte im Rahmen der Merkmalsextraktion anzuwenden sind, um die letztlich verwendeten Merkmalsvektoren zu erhalten. So ist durch `featDesc` die Merkmalsextraktion für nachfolgende Schritte transparent.

4.1.2 Akustische Modellierung

Das akustische Modell im JRTk unterscheidet zwischen Modellierungsklassen, die durch eine Kombination aus einem Eintrag im `CodebookSet` und einem im `DistribSet` beschrieben werden. Das `CodebookSet` enthält Mittelwerte und Kovarianzmatrizen aller Gaußmischverteilungen. Der Eintrag im `DistribSet` ergänzt für eine Klasse die Mixturgewichte. Durch diese Aufteilung lassen sich semikontinuierliche HMMs implementieren. Zusammengefasst werden diese beiden Einträge für ein Subpolyphon mit seinem Kontext in einem `Senone`, womit das `SenoneSet` die Menge aller Klassen beschreibt, zwischen denen das akustische Modell unterscheiden kann.

Ein Objekt vom Typ `HMM` kann zu der textuellen Repräsentation einer Äußerung das Modell mit Hilfe eines Aussprachewörterbuches erzeugen. Dieses Objekt repräsentiert anschließend selbst dieses Modell und speichert zu jedem Zustand mögliche Übergänge, sowie den Index des zugehörigen `Senone` im `SenoneSet` für die Bestimmung der Emissionswahrscheinlichkeiten.

4.1.3 MLE-Training

Gemäß dem Baum-Welch Algorithmus wird für das Training zunächst eine Schätzung über die – auch als "Labels" bezeichnete – Zuordnung von Merkmalsvektoren zu Zuständen eines zu der Transkription konstruierten HMMs bestimmt. Für die Bestimmung dieser Zuordnung bietet das JRTk die Anwendung von Viterbi oder Forward-Backward an. Einmal berechnete Zuordnungen lassen sich auch in einer Datei speichern, so dass sie nicht bei jedem Training neu bestimmt werden müssen. Das Laden vorberechneter Zuordnungen verwaltet ein Objekt vom Typ `Labelbox`. Eine Zuordnung liegt unabhängig von ihrer Quelle als Objekt vom Typ `Path` vor. Für jedes Zeitfenster in diesem Pfad können mehrere alternative Zustände mit ihrer Wahrscheinlichkeit gespeichert werden, da speziell Forward-Backward keine diskrete Zuordnung liefert.

Diesen Pfad erhält das `SenoneSet` für jede Trainingsäußerung über einen Aufruf von `accu` zur Auswertung. Das `SenoneSet` speichert für den Maximization-Schritt benötigte Statistiken in von `Codebook` und `Distrib` bereitgestellten Akkumulatoren, bis die Trainingsdaten alle ausgewertet sind. Erst mit einem Aufruf von `update` werden die Modellparameter im Maximization-Schritt angepasst.

4.2 Verwendung des Large-Margin-Training

Das gesamte Large-Margin-Training wird über ein Objekt vom Typ `LMHMMAccuSet` gesteuert. Dieses Objekt operiert direkt auf einer Struktur vom Typ `DistribSet`, die alle Verteilungen des akustischen Modells beschreibt. Diese Struktur muss bereits bei der Initialisierung des `LMHMMAccuSet` vorhanden sein und übergeben werden, da zu diesem Zeitpunkt Akkumulatoren bereitgestellt werden. Die Option `-eta` erlaubt es, den Wert von η in der Softmaxfunktion festzulegen, der für die Akkumulation mit diesem `LMHMMAccuSet` genutzt werden soll. Der Standardwert ist hier 2. Ebenfalls optional lässt sich `-usenegsamples 1` setzen, wodurch bei Training auf Fensterebene auch Trainingsbeispiele berücksichtigt werden, die falsch klassifiziert werden würden.

Listing 4.1: Erzeugung

```
#Erzeuge ein LMHMMAccuSet mit dem Namen lmhmm.
#Optionen:
#  - verwende ein η von 4 bei der Berechnung von Softmax
#  - verwende Trainingsbeispiele, die falsch erkannt
#        werden wuerden
LMHMMAccuSet lmhmm distribSet -eta 4 -usenegsamples 1
```

Zu Beginn jeder Trainingsiteration müssen die Akkumulatoren durch Aufruf von `clear` zurückgesetzt werden. In der folgenden Akkumulationsphase werden dem `LMHMMAccuSet` Trainingsbeispiele übergeben. Das genaue Vorgehen in dieser Phase hängt aber von der Ebene ab, auf der akkumuliert wird. Akkumulation auf Fensterebene bestimmt die Margin zwischen der das Fenster beschreibenden Modellierungsklasse und allen anderen Klassen. Auch die Parameteränderungen zur Optimierung werden für jedes Fenster einzeln bestimmt. Akkumulation auf Äußerungsebene dagegen bestimmt die Margin für die gesamte Äußerung und optimiert die Klassen so dass die Margin für die Äußerung als ganzes verbessert wird.

Bei Training auf Fensterebene ist die Komplexität folglich niedriger und es reicht ein Aufruf von `accuFrame` mit dem Namen der korrekten Klasse und dem Index des Fensters, für das diese Klasse als Referenz dienen soll. Intern werden dann alle dem System bekannten Klassen gegen die übergebene Referenz trainiert. Einschränken lässt sich die Auswahl implizit durch die Option `-epsilon`, die festlegt, wie viel schlechter als die Referenz eine Klasse höchstens bewertet sein darf, um berücksichtigt zu werden. Eine präzisere Einschränkung liefert die Option `-exclude`, der eine explizite Liste von Klassennamen übergeben wird. Diese Klassen werden ebenfalls nicht gegen die Referenz trainiert. Die Modellparameter werden während der Akkumulationsphase nicht verändert. Sind alle Trainingsbeispiele übergeben worden, so führt ein Aufruf von `update` die Änderung der Modellparameter durch. Als Option für `update` kann und sollte mit `-stepsize` die Schrittgröße für Gradientenabstieg übergeben werden. Ein Beispiel für diese Art von Training findet sich in Abschnitt A.1.

Eine Akkumulation auf Satzebene unterscheidet sich nur in der Akkumulationsphase. Da eine Reihe von Vorberechnungen nötig ist, muss hier zunächst die Referenz durch `setReference` separat als Pfad übergeben werden. Es muss auch das verwendete `SenoneSet` übergeben werden, da das `LMHMMAccuSet` keine Möglichkeit hat, über die ihm bekannten Objekte Zugriff darauf zu erhalten. Um Konkurrenzmodelle zu erhalten, wertet diese Variante eine übergebene Lattice aus. Ein Aufruf von

`accuGLat` berechnet aus der als Parameter übergebenen Lattice eine Liste von N am besten bewerteten Pfaden. Dieses N ist 100, sofern nicht über die Option `-N` ein anderer Wert übergeben wird. Auch bei dieser Akkumulation kann über die Option `-epsilon` festgelegt werden, wie viel schlechter ein Pfad gegenüber der Referenz sein darf, um berücksichtigt zu werden. Auf die Akkumulation der Konkurrenzpfade folgend, werden durch einen Aufruf von `finishReference` abschließende Berechnungen für ein einzelnes Trainingsbeispiel und den zugehörigen Referenzpfad durchgeführt. An der Aktualisierung der Modelle ändert sich hier nichts, auch in diesem Fall werden durch `update` die für die Mittelwerte der Mischverteilungen bestimmten Verschiebungen angewandt. In Abschnitt A.2 ist ein Beispiel für den Einsatz der Befehle in einem Trainingskontext.

Es wird auch eine Reihe von Funktionen im Zusammenhang mit Serialisierung angeboten. Die Funktionen `save` und `load` sind für binäre Serialisierung zum Austausch zwischen verschiedenen Instanzen des Programms gedacht. Analog zu anderen Komponenten wird der vorhandene Inhalt der Akkumulatoren bei einem Aufruf von `load` nicht ersetzt. Die in der geladenen Datei vorliegenden Werte werden auf die vorhandenen addiert, so dass bei Parallelisierung mehrere Instanzen unabhängig voneinander verschiedene Teilmengen der Trainingsdaten akkumulieren und den Akkumulatorinhalt speichern können. Vor dem Aufruf von `update` können dann diese Zwischenergebnisse durch einen Aufruf von `load` für jeden gespeicherten Akkumulatorinhalt kombiniert werden. Eine andere Aufgabe hat die Funktion `write`. Diese speichert den Akkumulatorinhalt in einem für Menschen lesbaren Format basierend auf JSON.

4.3 Implementierungsgrundlagen

Der in Abschnitt 4.2 angesprochene Objekttyp `LMHMMAccuSet` ist in einem neu angelegten Quelldateienpaar `lmhmm.h` und `lmhmm.c` als Struktur implementiert. Diese Struktur enthält zum einen sämtliche im Laufe der Akkumulation anfallenden Daten, zum anderen Einstellungen und Statistiken für eine Trainingsinstanz. Die wichtigsten Einträge in der Struktur `LMHMMAccuSet` sind:

`LMHMMAccuSet`:

`LMHMMAccu* accus`: Zeiger auf ein Array von Akkumulatoren. Es wird bei der Initialisierung ein Akkumulator für jede `Distrib` im zu optimierenden `DistribSet` angelegt.

`double objInnerAccu`: Ein Akkumulator für die in Gleichung 3.6 als $Q(\Omega)$ bezeichnete Funktion, die die Summe im Kern der Zielfunktion umfasst.

`LMHMMStreamReference* refStreamData`: Zeiger auf ein Array, das Referenzinformationen für die einzelnen Streams enthält.

`double refScore`: Bei Akkumulation auf Äußerungen wird die Bewertung des Referenzpfades $\mathcal{F}\left(X|\omega_{\mathrm{id}(X)}\right)$ hier zwischengespeichert, da sie bei jedem akkumulierten Konkurrenzpfad für die Bestimmung der relativen Margin benötigt wird.

`double refWeightSum`: Dieser Wert akkumuliert einen speziellen Summenterm aus Gleichung (3.7):

$$\frac{\partial Q(\Omega)}{\partial \tilde{\mu}^m_{skl}} = \eta \sum_{X \in S} \left(\overbrace{\sum_{j \neq \mathrm{id}(X)} \exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \frac{1}{\mathcal{F}(X, \omega_j)}}^{\texttt{refWeightSum}} \frac{\partial \mathcal{F}(X, \omega_{\mathrm{id}(X)})}{\partial \tilde{\mu}^m_{skl}} - \sum_{j \neq \mathrm{id}(X)} \exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \frac{\mathcal{F}(X, \omega_{\mathrm{id}(X)})}{\mathcal{F}(X, \omega_j)^2} \frac{\partial \mathcal{F}(X, \omega_j)}{\partial \tilde{\mu}^m_{skl}} \right) \tag{4.1}$$

Auf diese Weise muss die Ableitung nach dem Referenzpfad nur einmal bestimmt werden, nachdem die Summe für alle Konkurrenzmodelle zu einem Referenzmodell akkumuliert wurde.

Die hier über `accus` referenzierte Struktur `LMHMMAccu` dient dazu, die im Rahmen des Trainings vorzunehmenden Änderungen der Merkmalsvektoren zu speichern. Sie ist wie folgt aufgebaut:

`LMHMMAccu`:

`Codebook* cbP`: Ein Zeiger auf das zu dieser Mixtur gehörende `Codebook`. Dieser Zeiger wird benötigt, um die Mittelwerte des `Codebook` im Rahmen des Update zu aktualisieren.

`DMatrix* muAccu`: Eine `double`-Matrix, in der die Änderungen für eine Iteration akkumuliert werden. Gebildet wird sie aus den Differenzvektoren für die einzelnen Mittelwerte.

Es ist zu beachten, dass auch bei der Akkumulation auf Äußerungen die Verschiebung einzelner Mittelwerte akkumuliert wird. Die Informationen können also unabhängig von der Akkumulationsebene in dieser Struktur gespeichert werden. Damit wird `accus` in `LMHMMAccuSet` von beiden Akkumulationsmethoden gleichermaßen verwendet.

Nur für Training auf Äußerungen enthält die Struktur `LMHMMAccuSet` eine Reihe von Variablen, die Informationen zu einem Referenzpfad zwischenspeichern können. Die Struktur `LMHMMStreamReference` wird für die Speicherung dieser Referenzen genutzt und hat folgende Einträge:

`LMHMMStreamReference`:

`int refFrameN`: Speichert die Länge der Referenz als Anzahl von Zeitfenstern. Jeder der folgenden Einträge dieser Struktur ist ein Array, das Informationen zu jedem Zeitfenster der Referenz enthält. Damit nennt dieser Wert die Kardinalität der folgenden Elemente.

`int* refDsX`: Für jedes Zeitfenster enthält ein Array, das dieser Zeiger referenziert, den Index der als Referenzzuordnung bestimmten Mischverteilung.

`int* refCompX`: Als feinere Eingrenzung wird hier jedem Zeitfenster zusätzlich noch die beste Mixturkomponente innerhalb der Referenzmischverteilung zugeordnet.

Eine besondere Funktionalität vom JRTk, die bei der Entwicklung berücksichtigt, aber nicht in den Experimenten getestet wurde, ist die Operation auf mehreren Streams. Diese Funktionalität kommt beispielsweise bei Nutzung von *Bundled Phonetic Features (BDPFs)* [32] zum Einsatz. Dazu wird die Referenz für jeden Stream separat bestimmt, wofür `LMHMMAccuSet` mehrere Referenzpfade in `refStreamData` speichern kann. Auf die Implementierung der Akkumulatoren hatte dieses Konzept keinen Einfluss, da angenommen wird, dass unterschiedliche Streams auch separate Verteilungen verwenden und sich so das Training nicht gegenseitig beeinflusst. Eine korrekte Unterstützung von Streams könnte aber zusätzlich noch erfordern, dass der Akkumulator `objInnerAccu` für die Funktion $Q(\Omega)$ für jeden Stream getrennt behandelt wird.

Einige der Werte in `LMHMMAccuSet` werden nur bei einem Training auf Fenstern oder Äußerungen genutzt. Dazu gehört die Option `useMisclassifiedSamples`, die bestimmt, ob beim Training auf Fensterebene auch Trainingsbeispiele auch verwendet werden, falls ein Konkurrenzmodell für sie eine bessere Bewertung liefert als die Referenz. Eine solche Option wurde für Äußerungen nicht implementiert. Bei Training auf Äußerungsebene werden grundsätzlich alle Trainingsbeispiele verwendet.

Ebenfalls nur beim Training auf Fensterebene wird eine Reihe von Statistiken im Verlauf des Trainings bestimmt. Da bei dieser Trainingsmethode die Option besteht nur Trainingsbeispiele mit bestimmten Eigenschaften zu verwenden, wird beim Training festgehalten wie viele tatsächlich verwendet werden. Für Trainingsbeispiele, die nicht verwendet werden, existieren dann Statistiken über die Gründe dieser Entscheidung.

Nur für ganze Äußerungen werden in `refStreamData` vorberechnete Informationen für einen Referenzpfad pro Stream gespeichert, die Anzahl der Streams in `refStreamCount`. Zusätzlich werden auch Hilfsgrößen bei der Akkumulation eines einzelnen Trainingsbeispieles einmalig vorberechnet und in `refScore` und `refWeightSum` gespeichert.

Eine Reihe von Funktionen bildet die Implementierungsseite der Schnittstelle für die Manipulation der Strukturen über Tcl (in Klammern sind die Tcl-Befehle angegeben, über die diese Funktionen aufgerufen werden):

`lmhmmCreate`: Diese Funktion wird bei der Erzeugung aufgerufen, sie initialisiert die Struktur mit übergebenem `DistribSet`, sowie den Werten der Optionen `eta` und `useMisclassifiedSamples`.

`lmhmmFree`: Die Aufgabe dieser Funktion ist den reservierten Speicher freizugeben, sobald das Objekt über die Tcl-Schnittstelle zerstört wird.

`lmhmmClear` (`clear`): Setzt die Werte in den Akkumulatoren zwischen Trainingsiterationen zurück.

`lmhmmLoadAccus` (`load`): Addiert die Werte von in einer Datei gespeicherten Akkumulatoren auf die in diesem Akkumulator gespeicherten Werte. Diese Funktion kommt zum Einsatz wenn mit mehreren Instanzen der Struktur parallel über unterschiedliche Teile der Trainingsdaten akkumuliert wird. Diese Funktion erlaubt dann die Teilsummen in den gespeicherten Akkumulatoren zu einem Gesamtakkumulator zu addieren.

`lmhmmSaveAccus` (`save`): Diese Funktion stellt das Gegenstück zu `lmhmmLoadAccus` dar, sie ermöglicht das Speichern der Akkumulatoren in einer Datei.

`lmhmmWriteAccus` (`write`): Diese Funktion dient auch der Serialisierung, anders als `lmhmmSaveAccus` nutzt diese Funktion jedoch kein binäres Format, sondern ein an JSON angelehntes Klartextformat. Das ermöglicht sowohl manuelle, als auch automatische Analyse der exportierten Akkumulatordaten.

`lmhmmAccuFrame` (`accuFrame`): Dies ist die einzige Akkumulationsfunktion, die für Nutzung auf Fensterebene benötigt wird. Durch sie wird für ein Beispielfenster eine übergebene Referenzklasse gegen die anderen durch die Verteilungen im `DistribSet` definierten Klassen trainiert. Im Detail befasst sich Abschnitt 4.4 mit diesem Trainingsansatz.

`lmhmmSetReference` (`setReference`): Diese Funktion wird nur bei der Akkumulation auf ganzen Äußerungen genutzt, für jedes neue Trainingsbeispiel muss zunächst diese Funktion aufgerufen werden, um einen Referenzpfad zu definieren, gegen den trainiert werden soll.

`lmhmmAccuPath` (`accuPath`): In dieser Funktion wird einer der Konkurrenzpfade für ein Training auf Äußerungsebene gegen den zuletzt definierten Referenzpfad übergeben.

`lmhmmFinishReference` (`finishReference`): Sobald der letzte Pfad übergeben wurde, werden durch diese Funktion abschließende Berechnungen für den Referenzpfad durchgeführt.

`lmhmmAccuGLat` (`accuGLat`): Statt jeden Konkurrenzpfad separat zu akkumulieren, lässt sich auch eine Lattice über diese Funktion übergeben, aus der eine konfigurierbare Anzahl von Konkurrenzpfaden bestimmt wird. Diese Funktion führt dann den Aufruf von `lmhmmAccuPath` für diese Pfade durch.

`lmhmmUpdate` (`update`): Unabhängig davon, ob ein Training auf Fenster- oder Äußerungsebene durchgeführt wurde, liegen die Verschiebungen am Ende der Akkumulation in `accus` für einzelne Mischverteilungen im `DistribSet` vor. Eine Trainingsiteration wird abgeschlossen indem diese Verschiebungen auf die vorliegenden Modelle durch einen Aufruf von `lmhmmUpdate` angewandt werden.

Die folgenden beiden Abschnitte gehen im Detail auf die Unterschiede bei der Implementierung der beiden Trainingsmethoden ein.

4.4 Training auf Fensterebene

Im einfachsten Fall muss jeweils ein Referenzmodell gegen jedes andere auf Fensterebene trainiert werden. Für diese Art von Training wird ein Trainingsbeispiel in Form eines Merkmalsvektors übergeben, sowie das Referenzmodell, dem dieses Trainingsbeispiel zugeordnet werden soll.

Es werden zunächst die Bewertungen für alle Mischverteilungen vorberechnet. Gleichzeitig wird das Modell mit der besten Bewertung gesucht, da das Verfahren, wie es in [21] beschrieben wird, nur für Trainingsbeispiele anzuwenden ist, die richtig zugeordnet werden. Auch die Bedingung, dass der Abstand zu dem besten Konkurrenzmodell nicht zu groß sein darf, wird an dieser Stelle geprüft. Diese beiden Überprüfungen entsprechen somit den Stützvektorkriterien aus [21]. Abweichend von dieser Vorlage lässt sich das Trainingsverfahren auch so konfigurieren, dass es die erste Bedingung nicht überprüft, sondern auch Trainingsbeispiele verwendet, die falsch klassifiziert werden würden.

Für die Updateoperation in Gleichung (3.5) wird der Wert von $Q(\Omega)$, wie es in Gleichung (3.6) definiert ist, für die aktuelle Iteration benötigt. Für diese Funktion gibt es mit `objInnerAccu` einen eigenen Akkumulator in `LMHMMAccuSet`, auf den für jedes verwendete Trainingsbeispiel ein Term addiert wird.

Bei der tatsächlichen Bestimmung der Ableitung gemäß Gleichung (3.8) wird nicht für jeden Mittelwert über alle Trainingsbeispiele iteriert. In `accu` liegt für die Ableitung nach jedem Mittelwert ein Akkumulator vor, der stets eine Teilsumme der Akkumulation über alle Trainingsbeispiele $s \in S$ aus Gleichung (3.8) enthält. Für jedes Trainingsbeispiel wird der entsprechende Term auf die jeweiligen Teilsummen für die einzelnen Mittelwerte addiert.

In diesem Punkt findet sich eine wichtige Unterscheidung gegenüber dem EM-Ansatz, der den Mittelwert aller potentiellen Referenzmodelle für einen Merkmalsvektor gemäß ihrer Likelihood in Richtung des Trainingsbeispieles verschiebt. Hier wird dagegen ein einziges Referenzmodell gewählt, das jedem Konkurrenzmodell gegenübergestellt wird. Gesucht wird eine Anpassung, die eine bessere Unterscheidbarkeit des aktuellen Trainingsbeispieles erlaubt. Die Verschiebung des Referenzmodelles für ein einzelnes Trainingsbeispiel ergibt sich aus einer Überlagerung von Verschiebungen. Jede Paarung mit einem Konkurrenzmodell beeinflusst das Referenzmodell und auch das Konkurrenzmodell zugunsten einer besseren Unterscheidbarkeit.

Bei Training auf Fensterebene ergibt sich eine Vereinfachung für Gleichung (3.9). In der allgemeinen Form wird hier der Beitrag des Mittelwertes zur Bewertung berücksichtigt, also insbesondere auch für welche Fenster im Trainingsbeispiel der zum Mittelwert gehörende Zustand in Frage kommt. Die hier betrachteten Modelle und Trainingsbeispiele umfassen aber nur ein Fenster und sämtliche Modellierungsklassen haben nur einen Zustand, so dass nur der Bruch in einem Summenglied aus Gleichung (3.9) verbleibt.

Sobald alle Trainingsbeispiele akkumuliert sind, wird eine Aktualisierung der Parameter durchgeführt. Dafür muss ausschließlich Gleichung (3.5) ausgewertet werden, da sowohl $Q(\Omega)$ in `objInnerAccu`, als auch die Ableitung dieser Funktion nach allen im Training angetroffenen Mittelwerten in `accus` bereits vorliegen. Der Parameter ϵ, der die Schrittgröße bezeichnet, muss experimentell ermittelt werden.

4.5 Training auf kontinuierlicher Sprache

Einen komplexeren Fall stellt das Akkumulieren auf kontinuierlicher Sprache dar. Hierbei wird nicht mehr die Bewertung der Fenster verwendet, sondern stets eine ganze Äußerung ausgewertet. Eine Schwierigkeit stellt es in diesem Fall dar, die konkurrierenden Modelle zu bewerten. Idealisiert würde man an dieser Stelle alle möglichen Äußerungen auswerten, doch ist dieses Vorgehen praktisch nicht umsetzbar. Eine gängige Annäherung ist es, dem für die Äußerung angenommenen Referenzmodell eine vorgegebene Anzahl der bestbewerteten Hypothesen gegenüberzustellen [22]. Die zugrundeliegende Überlegung ist, dass so gerade die Modelle gegeneinander abgegrenzt werden, die bei der Klassifikation ebenfalls relevant sind. Weiterhin hofft man mit einer Liste der bestbewerteten Hypothesen bereits einen ausreichenden Teil der Wahrscheinlichkeitsmasse abzudecken, so dass der Fehler sich in Grenzen hält.

Das JRTk erzeugt bei der Auswertung der Modelle eine *Lattice* auf Wortebene. Dabei handelt es sich um einen azyklischen, gerichteten Graphen mit eindeutigem Start- und Endknoten. Ein Wort, das in mehreren Hypothesen für einen gemeinsamen Zeitraum vermutet wird, erhält in diesem Graphen nur eine Repräsentation. Zusätzlich bietet das JRTk bereits die Möglichkeit für einen Knoten und ein Paar von Ein- und Ausgang die Kosten zu bestimmen. Diese Bewertung wird durch `initAddon` vorberechnet. Zunächst wurden diese Daten vorgehalten, damit `pathAlignGlat` schnell einen Pfad bestimmen kann, der für jedes Fenster alle Zustände auflistet, die für eine gegebene Lattice zu diesem Zeitpunkt angenommen werden können. Für ein Training auf Satzebene muss jedoch konkret eine Liste von N besten Pfaden vorliegen.

Es gibt bereits verschiedene Werkzeuge, die dafür verwendet werden können. Sie erfordern allerdings einen Vorberechnungsschritt, der die Lattice in das erwartete Format bringt. Der so entstehende Aufwand macht dieses Verfahren zu zeitaufwendig, als dass es in jeder Iteration des Algorithmus genutzt werden könnte. Es wird daher ein eigens implementierter Algorithmus verwendet, der in zwei Durchgängen eine N-Bestenliste bestimmt.

Im ersten Durchgang wird für jeden Knoten und jede Eingangskante der günstigste Weg zum Endknoten bestimmt. Da der Graph azyklisch ist, muss jeder Knoten genau einmal betrachtet werden, nämlich sobald alle Ausgangskanten als Eingangskanten der Zielknoten betrachtet wurden. Vorberechnete Bewertungen liegen jeweils für Tripel von Knoten sowie Ein- und Ausgangskante vor. Als Folge lässt sich der beste Pfad über eine Eingangskante aus einer Iteration über alle Ausgänge bestimmen. Dazu wird für jeden Ausgang eine Summe aus den vorberechneten Kosten für das Paar von Ein- und Ausgang, sowie den zuvor bestimmten besten Kosten für die Ausgangskante als Eingangskante des Zielknotens berechnet.

Im zweiten Durchgang wird die im ersten Durchgang berechnete Bewertung der Knoten verwendet, um die Kosten für einen Pfad nach unten abzuschätzen. Das Verfahren lässt sich mit dem A*-Algorithmus [10] vergleichen, jedoch ist die dort verwendete Heuristikfunktion weniger streng. Im A*-Algorithmus wird an die Heuristik nur die Forderung gestellt, dass sie den Abstand zum Ziel nicht überschätzt. Das muss dadurch ausgeglichen werden, dass alle erreichten, aber noch nicht expandierten Knoten für eine mögliche Expansion erhalten bleiben. Im hier betrachteten Fall wird die Distanz zusätzlich nicht unterschätzt. Wie beim A*-Algorithmus wird dann die Summe aus bisherigen Kosten und der Abschätzung der Distanz verwendet, um Teilpfade zu bewerten. Die strengere Abschätzung garantiert aber, dass die

N besten Pfade durch Expansion der besten N Teilpfade erzeugt werden können. Daraus folgt, dass nur diese zur Expansion vorgehalten werden müssen. Sobald alle Pfade auf der Liste der zu expandierenden N Pfade den Endknoten erreicht haben, enthält die Liste somit die gewünschte N-Bestenliste.

Mit einer solchen Liste von Konkurrenzpfaden lässt sich das Trainingsverfahren für eine Referenzäußerung analog zu [22] auf ganze Pfade erweitern.

Aufgrund der zusätzlichen Komplexität bei der Bestimmung der Konkurrenzmodelle wurde auch der Akkumulationsvorgang neu strukturiert. Für jedes Trainingsbeispiel wird in separaten Funktionsaufrufen zunächst ein Referenzpfad über `lmhmmSetReference` übergeben, darauf folgend mehrere Konkurrenzpfade über `lmhmmAccuPath`, bevor ein Aufruf von `lmhmmFinishReference` abschließende Berechnungen für ein Trainingsbeispiel durchführt. Die Aufrufe von `lmhmmAccuPath` lassen sich durch einen Aufruf von `lmhmmAccuGLat` ersetzen. Diese Funktion bestimmt aus der Lattice nach dem zuvor beschriebenen Verfahren eine N-Bestenliste der Pfade und ruft für jeden davon `lmhmmAccuPath` auf.

Anders als bei Training auf Fensterebene liegen damit allerdings nicht mehr alle Konkurrenzmodelle in einem Akkumulationsaufruf vor. Stattdessen wird mit jedem Aufruf von `lmhmmAccuPath` ein neues Konkurrenzmodell für die aktuelle Trainingsäußerung als Pfad übergeben. Während also auf Fensterebene einmalig eine Akkumulation für ein Trainingsbeispiel gestartet wurde und innerhalb dieser Funktion alle Modelle außer der Referenz als Konkurrenzmodelle ausgewertet wurden, muss jetzt jeder Aufruf das Referenzmodell gegen ein neues Konkurrenzmodell auswerten. Um die Extraktion der relevanten Informationen aus dem Referenzpfad nicht für jeden Akkumulationsaufruf zu wiederholen, wird dieser Vorgang vor dem ersten Akkumulationsaufruf in `lmhmmSetReference` durchgeführt. Nach dem letzten Akkumulationsaufruf für ein Trainingsbeispiel wird auch ein Aufruf von `lmhmmFinishReference` benötigt, der abschließende Berechnungen durchführt. Speziell ist erst zu diesem Zeitpunkt `refWeightSum` (vgl. Gleichung (4.1)) vollständig akkumuliert und kann verwendet werden.

Die für Training auf Fensterebene beschriebenen Vereinfachungen gelten in diesem Fall nicht mehr, ein einzelner Merkmalsvektor kann zu mehreren Fenstern innerhalb eines Trainingsbeispieles beitragen und das sowohl zu Referenz- als auch zu Konkurrenzmodellen unterschiedlicher Fenster gehörend. Als Folge muss für die Anwendung der Aktualisierungsvorschrift in Gleichung (3.5) jetzt Gleichung (3.7) neben Gleichung (3.6) ausgewertet werden. In der neuen Form muss daher zwar weiterhin über alle Mittelwerte der Mischverteilungen iteriert werden, sie werden aber nicht mehr diskret als Referenz- oder Konkurrenzmodell betrachtet.

Zu jedem übergebenen Konkurrenzpfad ergeben sich in Gleichung (3.7) zwei Terme für jeden zu aktualisierenden Mittelwert:

$$\frac{\partial Q(\Omega)}{\partial \tilde{\mu}_{skl}^m} = \eta \sum_{X \in S} \left(\sum_{j \neq \mathrm{id}(X)} \overbrace{\exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \frac{1}{\mathcal{F}(X, \omega_j)} \frac{\partial \mathcal{F}(X, \omega_{\mathrm{id}(X)})}{\partial \tilde{\mu}_{skl}^m}}^{\text{erster Term}} \right.$$
$$\left. - \sum_{j \neq \mathrm{id}(X)} \underbrace{\exp(\eta \cdot d(\omega_{\mathrm{id}(X)}, \omega_j, X)) \frac{\mathcal{F}(X, \omega_{\mathrm{id}(X)})}{\mathcal{F}(X, \omega_j)^2} \frac{\partial \mathcal{F}(X, \omega_j)}{\partial \tilde{\mu}_{skl}^m}}_{\text{zweiter Term}} \right)$$

Ein Term kann jedoch gemäß Gleichung (4.1) unabhängig vom Mittelwert in `refWeightSum` berücksichtigt werden, nur der zweite Term wird direkt auf den Akkumulator für diesen Mittelwert addiert. Auch auf die in `objInnerAccu` gespeicherte Teilsumme für $Q(\Omega)$ muss ein Term für jeden Konkurrenzpfad addiert werden. Erst wenn alle Konkurrenzpfade übergeben wurden, wird in `lmhmmFinishReference` auch der in `refWeightSum` akkumulierte Term nach Multiplikation mit entsprechenden Ableitungen auf die Akkumulatoren der einzelnen Mittelwerte addiert.

Neben den in den Pfaden vorliegenden Bewertungen wird für die Auswertung von Gleichung (3.7) auch die Bestimmung der Ableitung nach jedem Mittelwert benötigt. Diese wird durch eine Hilfsfunktion `lmhmmCalculateScoreDeriv` bestimmt, die Gleichung (3.9) auswertet. Sie erhält für den betrachteten Pfad die den Zuständen zugeordneten Gaußmischverteilungen und die Mixturkomponenten mit der besten Bewertung innerhalb der Verteilungen. Zusätzlich werden eine Mischverteilung und eine Komponente übergeben, um den Mittelwert zu bezeichnen, nach dem abgeleitet werden soll.

Da die Akkumulatoren weiterhin für die einzelnen Mittelwerte definiert sind, ist bei der Aktualisierung der Mittelwerte keine Änderung gegenüber dem Training auf Fensterebene erforderlich.

Kapitel 5

Experimente

Dieses Kapitel stellt zunächst die beiden Korpora vor, die als Datenquelle für Experimente gedient haben. Anschließend werden die durchgeführten Experimente beschrieben. Aufgeteilt sind die Experimente in zwei Kategorien. Zunächst wurde eine Reihe von Experimenten auf Fensterebene durchgeführt. Schrittweise wird dann auf ein Training mit kontinuierlicher Sprache zugearbeitet. In der zweiten Experimentreihe werden dann Sequenzen von Merkmalsvektoren als Trainingsbeispiele und durch mehrere Zustände modellierte Klassen betrachtet. Ganzworterkenner als Zwischenstufe wurden nicht betrachtet, da dies im Rahmen der Arbeit nicht möglich war.

5.1 Korpora

Die durchgeführten Experimente nutzen Daten aus zwei Korpora, die nun vorgestellt werden sollen. Der EMG-UKA-Korpus ist vergleichsweise klein, enthält aber neben akustischer Aufzeichnung von Sprache auch eine Aufzeichnung von EMG-Signalen. Dem gegenüber steht der GlobalPhone-Datenkorpus, der nur Audioaufzeichnungen enthält, aber dessen Größe auch das Training eines Spracherkenners mit einer großen Anzahl Klassen erlaubt. In den folgenden beiden Abschnitten werden sie vorgestellt.

5.1.1 EMG-UKA-Korpus

Für Experimente mit EMG-Signalen wird der EMG-UKA-Korpus [11] verwendet. Dieser am Cognitive Systems Lab aufgezeichnete Korpus enthält EMG-Signale von lautlos artikulierter, geflüsterter und normal gesprochener Sprache von sieben männlichen und einem weiblichen Sprecher, zwischen 24 und 28 Jahren alt. Für alle Aufzeichnungen liegen EMG-Signale vor, für hörbare Sprache liegt zusätzlich ein, über ein Nahsprechmikrophon aufgezeichnetes, akustisches Signal vor.

Die hier durchgeführten Experimente nutzen nur die Aufzeichnungen hörbarer Sprache, so dass stets sowohl ein EMG- als auch ein Audiosignal vorliegt. Für einen Sprecher liegen zwischen einer und 26 Sitzungen vor. In einer Sitzung wurden 50 englischsprachige Sätze in einem stillen Raum zunächst laut gesprochen, dann geflüstert und dann lautlos artikuliert.

Von den 50 Sätzen waren zehn für alle Sprecher und Sitzungen gleich, während 40 zwischen Sitzungen und Sprechern variieren. Die gemeinsamen Sätze bilden den

	Anzahl Sprecher	Anzahl Sitzungen	Dauer [min] Training	Dauer [min] Test
Reguläre Sitzungen	8	63	115	32
Große Sitzungen	2	2	54	2
Gesamtkorpus	8	65	169	34

Tabelle 5.1: Übersicht über den genutzten Teil des EMG-UKA-Korpus

Testdatensatz, die verbleibenden werden ausschließlich für das Training genutzt. Bei der Aufzeichnung fand keine Gruppierung in diese beiden Datensätze statt, sondern alle Sätze wurden in zufälliger Reihenfolge aufgezeichnet. Zwei Sitzungen weichen in der Anzahl der Sätze ab, in einer liegen 500 Trainingssätze und 20 Testsätze vor, in einer zweiten sind es 496 Trainingssätze und 13 Testsätze. Insgesamt liegen 169 Minuten von Aufzeichnungen für das Training und 34 Minuten an Testdaten vor.

Die Anzahl der Sprecher und Sitzungen, sowie die Dauer der Aufzeichnungen im verwendeten Teil des EMG-UKA-Korpus ist in Tabelle 5.1 zusammengefasst. Die beiden Sitzungen, die in ihrem Umfang von den anderen abweichen, werden in dieser Tabelle als "große Sitzungen" bezeichnet. Die verbleibenden Sitzungen, die sich nicht in der Anzahl von Trainings- und Testsätzen voneinander unterscheiden, werden als "reguläre Sitzungen" bezeichnet. Für eine genaue Beschreibung der Methodik, einschließlich der verwendeten Ausrüstung und der Elektrodenpositionen, sei auf [11] verwiesen.

5.1.2 GlobalPhone

Der GlobalPhone Datenkorpus umfasst Sprachaufzeichnungen in 13 Sprachen. Für die Auswahl dieser Sprachen waren die Hauptmerkmale zum einen die Nützlichkeit von automatischer Erkennung dieser Sprachen, zum anderen eine hohe Diversität. Bei der Nützlichkeit wurden neben der Verbreitung auch volkswirtschaftliche Interessen berücksichtigt. Als Merkmale für Diversität wurden phonetische, phonologische, prosodische und morphologische Unterschiede betrachtet.

Die Aufzeichnungen enthalten abgelesene Texte aus überregionalen Tageszeitungen, deren Inhalt nationales und internationales Tagesgeschehen sowie Wirtschaftsberichte aus einem gemeinsamen Zeitraum umfasst. So wird die Vergleichbarkeit erhöht ohne das Vokabular einzuschränken und die Verschriftung muss nicht aufwendig im Nachhinein angefertigt werden. Für eine ausführliche Erläuterung zu der Auswahl der Sprachen und dem Vorgang der Datensammlung sei auf [31] verwiesen.

Für die Untersuchungen im Rahmen dieser Arbeit wurde nur der deutschsprachige Anteil aus dem Korpus verwendet. Damit stehen aus dem Datensatz 15 Stunden Sprachaufzeichnungen in deutscher Sprache für das Training zur Verfügung, 1,5 Stunden für Kreuzvalidierung und zwei Stunden für Evaluation. Die Sprecher sind vorwiegend männlich und 20-29 Jahre alt. In einem Experiment wurden aus dem Datensatz für das Training nur Aufzeichnungen von neun Sprechern mit 1,5 Stunden Dauer für das Training verwendet. Zusammengefasst wird die Aufteilung des Datenkorpus in Tabelle 5.2.

	Anz. Sprecher	Dauer [min]
Voller Trainingsdatensatz	65	894
Reduzierter Trainingsdatensatz	9	87
Kreuzvalidierungsdatensatz	6	88
Testdatensatz	6	133

Tabelle 5.2: Übersicht über den deutschsprachigen Teil des GlobalPhone-Korpus

5.2 Training auf Fensterebene

In diesem Fall werden für einzelne Fenster Klassenzuordnungen separat optimiert. Die Erweiterung um einen Large-Margin-Estimation Schritt hat hier zum Ziel die zugeordneten Modelle gegen alle anderen zu diskriminieren, wodurch eine bessere Erkennungsrate erreicht werden soll.

Im einfacheren ersten Fall haben die Modelle nur einen Zustand, sie werden hier dann als *atomar* bezeichnet, weil sie nur ein einzelnes Fenster beschreiben. Die relative Einfachheit der Konfiguration mit atomaren Modellen und Trainingsbeispielen, sowie der Möglichkeit den gesamten Modellraum als Konkurrenz zu betrachten, macht sie ideal als Ausgangspunkt. Da jeder Akkumulationsschritt auf nur einem Trainingsbeispiel von einem Fenster Länge durchgeführt wird und auch jedes Modell nur ein Fenster beschreibt, kann das Verfahren in einem kontrollierteren Rahmen untersucht werden. Es werden in den Abschnitten 5.2.1 und 5.2.2 Modelle dieser Art mit Daten der beiden Korpora trainiert und evaluiert.

Auch komplexere Modelle mit mehreren Zuständen lassen sich auf Fensterebene trainieren. Dabei werden einzelne Zustände gegen alle Zustände anderer Modelle trainiert, Zustände des gleichen Modelles werden nicht als Konkurrenzzustände verwendet. Dieses Trainingsverfahren wird in den Abschnitten 5.2.3 und 5.2.4 auf Daten des GlobalPhone-Korpus untersucht.

5.2.1 Atomare Modelle auf EMG-UKA-Datensatz

Ein erstes Experiment wurde für insgesamt vier Konfigurationen auf dem EMG-UKA Datensatz durchgeführt. Die Konfigurationen unterscheiden sich zum einen darin, ob EMG- oder Audiodaten verwendet werden, zum anderen ob tatsächliche Phoneme erkannt werden, oder phonetische Klassen, in denen mehrere Phoneme nach ihren Artikulationseigenschaften zusammengefasst werden. Das Training wurde für jeden Sprecher und jede Sitzung separat durchgeführt. Getestet wird auf beim Training zurückgehaltenen Daten einer Sitzung (vgl. Abschnitt 5.1.2). Im Zusammenhang mit EMG-Daten ist hier der Unterschied zu einem sprecherabhängigen Training wichtig. Auch bei dem gleichen Sprecher machen geringfügige Unterschiede bei Elektrodenpositionen zwischen Sitzungen einen erheblichen Unterschied bei der Erkennung.

Abbildung 5.1 stellt die durch ein zusätzliches Large-Margin-Training erreichte Verbesserung dar. Die x-Achse zeigt die Anzahl der durchgeführten Gradientenabstiegsiterationen. In der y-Achse ist die durchschnittliche Verbesserung dargestellt. Zusätzlich ist um die Verbesserung das 95%-Konfidenzintervall eingetragen, bestimmt über die Ergebnisse für die einzelnen Sitzungen.

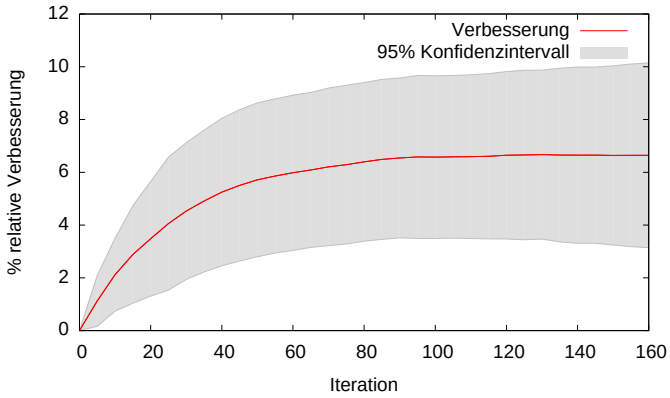
Bei geeigneter Wahl der Schrittgröße und Anzahl von Iterationen lassen sich hier bei Audiodaten über die Testsprecher gemittelt 5.02% relative Verbesserung erreichen, bei EMG Daten sind es 6.67%. Bei phonetischen Klassen lässt sich eine durchschnittliche relative Verbesserung um 17.38% für Audiodaten beobachten, bei EMG Daten sind es sogar 34.37%. Eine tabellarische Auflistung der Ergebnisse befindet sich in Abschnitt B.1 im Anhang.

Eine genaue Untersuchung der Ergebnisse zeigt, dass bereits sehr früh Overtraining stattfindet. Zwar verbessern sich die Testergebnisse über viele Iterationen, jedoch nur dank der effektiven Anpassung einiger weniger Klassen, die auf Kosten der anderen viel besser erkannt werden. In Abbildung 5.2a ist die Verwechslungsmatrix der Klassen nach 10 Gradientenabstiegsiterationen dargestellt, Abbildung 5.2b zeigt die Verwechslungsmatrix nach 50 Iterationen. Jede Zeile und Spalte steht für eine Klasse, wobei Spalten die erkannten Klassen darstellen und Zeilen die korrekte Zuordnung. Farblich ist die Zu- bzw. Abnahme von Zuordnungen hervorgehoben. So wird eine Verbesserung durch blaue Einträge auf der Diagonalen der Matrix erkennbar. Die Darstellung soll nicht die absolute Zu- bzw. Abnahme darstellen, sondern die Verteilung dieser Veränderungen über die Matrix. Daher sind sowohl positive, als auch negative Veränderungen durch den maximalen vorkommenden Wert normalisiert. Weiterhin ist zugunsten einer Hervorhebung der Unterschiede bei der Verteilung eine logarithmische Skala für den Intensitätsverlauf gewählt worden. Viele blaue Einträge in einer Spalte zeigen an, dass einer Klasse bei der Erkennung zusätzliche Merkmalsvektoren zugeordnet werden, die zu anderen Klassen gehören. Nach 50 Gradientenabstiegsiterationen tritt gerade das ein: einige wenige Klassen erhalten viele Merkmalsvektoren anderer Klassen zugewiesen, während ein großer Teil der Klassen auch die ihnen zugehörigen Merkmalsvektoren verliert.

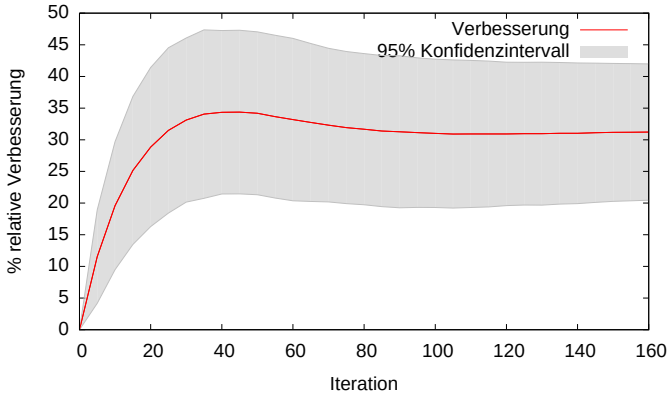
Es wurde untersucht, ob der die Aufhebung der Bedingung von korrekter Klassifikation für die Trainingsdaten hier einen entscheidenden Unterschied bewirkt. Tatsächlich wird der Effekt von Overtraining bei späteren Iterationen dadurch abgemildert (Abb. 5.2c). Das lässt sich damit erklären, dass bei Nichtberücksichtigung von falsch klassifizierten Trainingsbeispielen die Entscheidungsebene nur einmal an einem Trainingsbeispiel vorbeiziehen muss, damit dieses Trainingsbeispiel in den folgenden Iterationen keinen Einfluss mehr hat. Es besteht das Risiko, dass eine Klasse so immer mehr Trainingsbeispiele anderer Klassen umfasst und damit die Trainingsdaten für diese Klassen schwächt. Leider ist die Milderung durch Nutzung aller Trainingsdaten nur geringfügig.

5.2.2 Atomare Phonemmodelle auf GlobalPhone-Datensatz

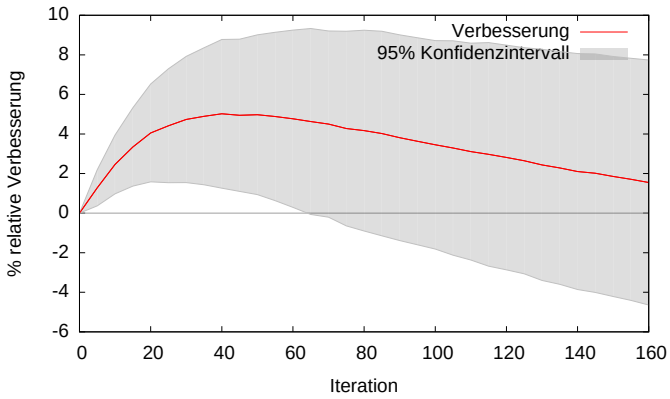
Dieses Experiment hatte zum Ziel zu überprüfen, inwieweit sich die Ergebnisse auf größere Mengen Sprachdaten übertragen lassen. Dafür wurden insgesamt drei Konfigurationen überprüft. In einem ersten Fall wurde analog zum vorhergehenden Experiment sprecherabhängig trainiert. Für jeden Sprecher wurde ein Teil der Daten beim Training zurückgehalten und bei der Evaluation verwendet (vgl. Abschnitt 5.1.2). Trainiert wurden ausschließlich Modelle von Phonemen. Diese Konfiguration wurde in zwei Schritten erweitert, um der größeren Menge Daten Rechnung zu tragen. Im ersten Schritt wurden nur die Trainingsdaten aller Sprecher verwendet, um ein gemeinsames Modell zu trainieren. Die Sprecher waren aber weiterhin bekannt, da auch hier auf zurückgehaltenen Daten der gleichen Sprecher getestet wurde. Im letzten



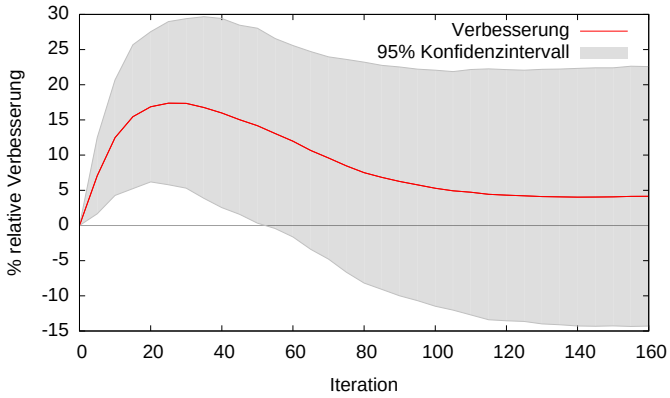
(a) EMG Daten, phoneme



(b) EMG Daten, phonetische Klassen

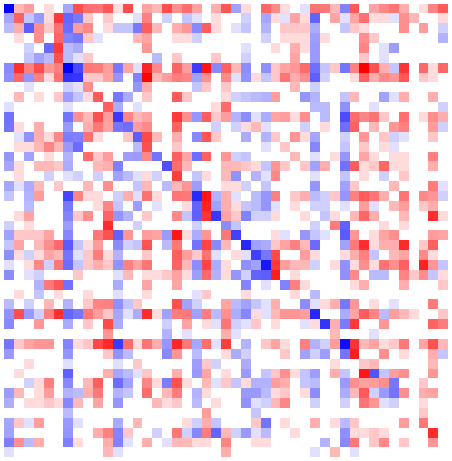


(c) Audiodaten, phoneme

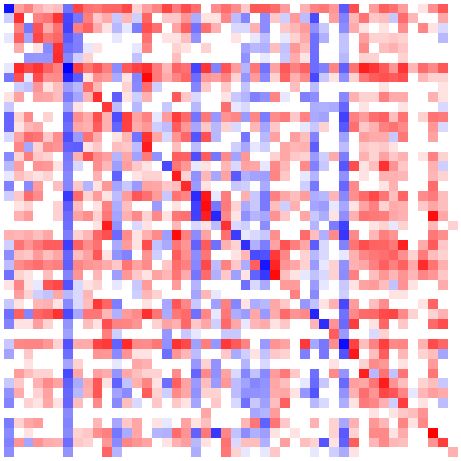


(d) Audiodaten, phonetische Klassen

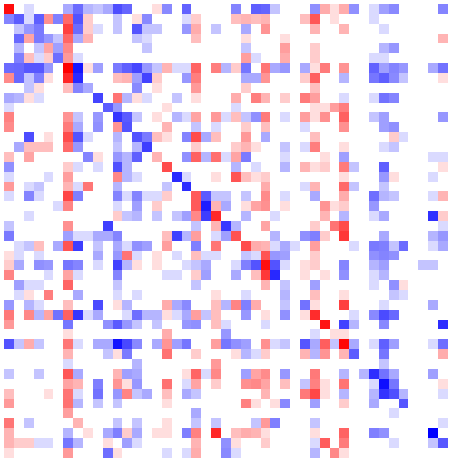
Abbildung 5.1: Über alle Sprecher gemittelte relative Verbesserung mit 95% Konfidenzintervall bei Fensterbasiertem Training atomarer Modelle auf dem EMG-UKA-Datensatz.



(a) Differenz zum Ausgangswert nach 10 Iterationen



(b) Differenz zum Ausgangswert nach 50 Iterationen



(c) Dfferenz zwischen 50 Iterationen ohne bzw. mit Nutzung falsch klassifizierter Trainingsbeispiele

Abbildung 5.2: Differenzen zwischen verschiedenen Verwechslungsmatrizen visualisiert durch Farbe und Intensität. Blau steht für eine Zunahme in einem Eintrag, Rot für eine Abnahme. Die Farbintensität nutzt eine relative, logarithmische Skala (der maximal vorkommende Wert wird der maximalen Intensität zugeordnet)

Schritt wurde ein Teil der Sprecher komplett zurückgehalten, trainiert wurde dann mit den kompletten Datensätzen einiger Trainingssprecher, so dass die Testsprecher komplett unbekannt waren.

Das Trainingsverfahren an sich unterscheidet sich dabei nur unwesentlich von der vorherigen Konfiguration. Da der Trainingsdatensatz in deutscher Sprache ist, wurde auch ein deutscher Phonemsatz verwendet. Jedes Phonem wird aber auch in dieser Konfiguration durch ein HMM mit nur einem Zustand dargestellt. Trainiert wurde die zu einem für ein Zeitfenster bestimmten Referenzphonem gehörende Gaußmischverteilung dann gegen die Verbleibenden. Da die Verwendung falsch klassifizierter Trainingsbeispiele zuvor bereits nur geringfügige Unterschiede bei den Ergebnissen zur Folge hatte, wurden sie auch hier beim Training verwendet.

Bei einem vergleichbaren Aufbau zeigen sich auch auf dem GlobalPhone-Datensatz Ergebnisse, die mit den Ergebnissen aus dem EMG-UKA Experiment vergleichbar sind (Abb. 5.3). Über alle Sprecher gemittelt wurde auch hier eine relative Verbesserung von 5.36% erreicht.

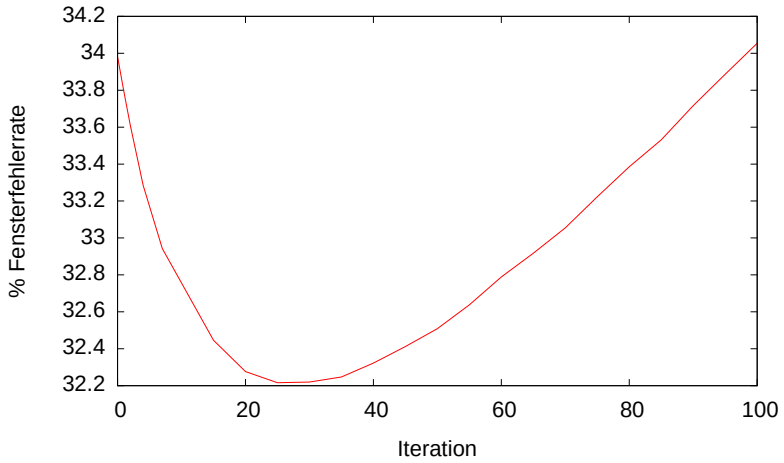


Abbildung 5.3: Sprecherabhängiges Training atomarer Phonemmodelle auf dem GlobalPhone-Datensatz

Trainiert man anstelle von sprecherabhängigen Modellen ein Modell mit den kombinierten Daten aller Sprecher, erhält man 5.79% relative Verbesserung (Abb. 5.4).

Beim Training eines sprecherunabhängigen Modells und Test auf ungesehenen Sprechern zeigt sich die Verbesserung noch deutlicher (Abb. 5.5). Hier wird über die Testsprecher gemittelt eine relative Verbesserung von 8.75% erreicht.

Eine tabellarische Auflistung der in den Graphen dargestellten Ergebnisse findet sich in Abschnitt B.2 im Anhang. Insgesamt werden die Modelle zwar bei jedem Verallgemeinerungsschritt erwartungsgemäß unzuverlässiger, das Large-Margin-Training zeigt aber eine zunehmende relative Verbesserung.

5.2.3 Fensterbasiertes Training kontextunabhängiger Phonemmodelle auf GlobalPhone-Datensatz

Eine weitere Annäherung an Nutzung für kontinuierliche Sprache liefert das diskriminative Training kontextunabhängiger Modelle auf Fensterebene. Die Komplexität

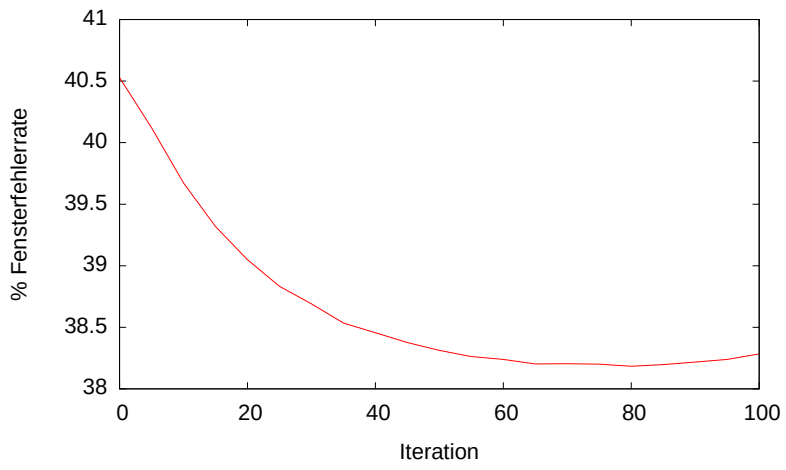


Abbildung 5.4: Sprecherunabhängiges Training atomarer Phonemmodelle auf dem GlobalPhone-Datensatz mit Test auf unbekannten Sätzen bekannter Sprecher

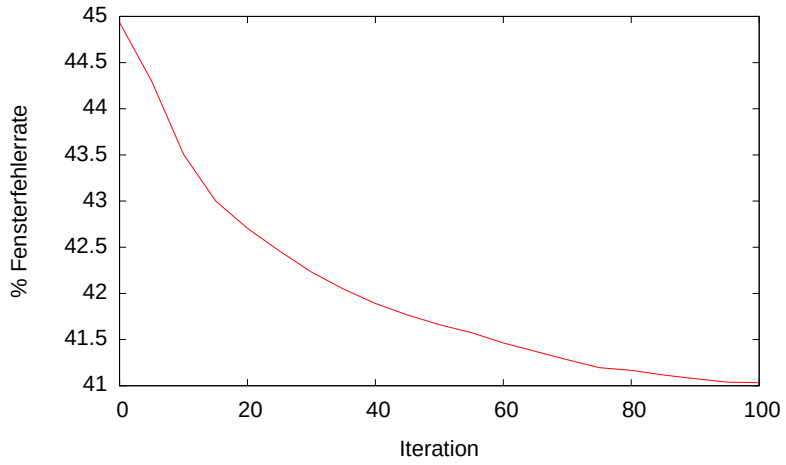


Abbildung 5.5: Sprecherunabhängiges Training atomarer Phonemmodelle auf dem GlobalPhone-Datensatz mit Test auf unbekannten Sprechern

wird von einfachen Fenstermodellen mit nur einem Zustand für ein Phonem, auf Hidden-Markov-Modelle mit drei Zuständen erhöht. Die Anzahl der unterscheidbaren Klassen wächst dadurch von 44 auf 130, was erwartungsgemäß die Separabelität verschlechtern sollte. Diese neuen Modelle können allerdings nun auch sinnvoll auf ganzen Sätzen getestet werden, was die Auswirkungen eines solchen diskriminativen Trainings für HMM-Zustände auf die Wortfehlerrate zeigt.

Das Training verändert sich in diesem Fall nur geringfügig. Es wird weiterhin für ein Zeitfenster aus der Zuordnung zu einem HMM-Zustand die Referenzmischverteilung bestimmt, die dann gegen die restlichen trainiert wird. Um zu vermeiden, dass mehrere Zustände des gleichen Phonems gegeneinander trainiert werden, wurde hier die Funktionalität eingeführt eine Liste von Mischverteilungen als Konkurrenzklasse auszuschließen (vgl. Abschnitt 4.2). Für jedes Fenster wurden, durch Ausschließen von Zuständen des gleichen Phonems, nur die Zustände anderer Phoneme als Konkurrenz verwendet. Falsch klassifizierte Trainingsbeispiele wurden auch hier für das Training verwendet.

Die beste relative Verbesserung, die dabei auf der Wortfehlerrate erreicht wurde beträgt 5.3%. Auffällig ist, dass dieses Optimum nicht mit dem Optimum für Genauigkeit auf Fensterebene zusammenfällt. Stattdessen verbessert sich die Fehlerrate auf Fensterebene weiterhin geringfügig, während die Wortfehlerrate nach der 80. Iteration stark ansteigt (Abb. 5.6). Für eine tabellarische Darstellung der Ergebnisse sei an dieser Stelle auf Abschnitt B.3 im Anhang verwiesen. Allgemein ist zu erkennen, dass sich Verbesserungen aus dem Training auf Fensterebene nur unzuverlässig auf Testergebnisse mit ganzen Sätzen übertragen lassen. Eine Erklärung könnte auch hier das bereits zuvor beobachtete Overtraining bieten. Sobald zunehmend wenige Klassen auf Kosten der verbliebenen vom Training profitieren, könnte auch die Nützlichkeit des Trainings für die Klassifikation auf Satzebene abnehmen.

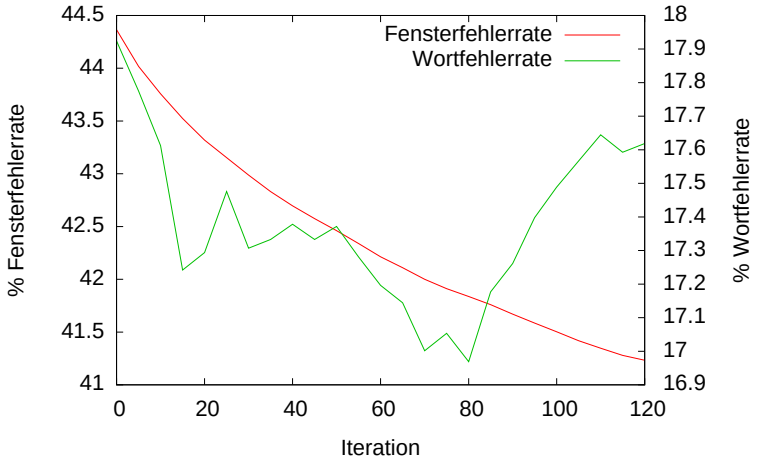


Abbildung 5.6: Vergleich von Fenster- und Wortfehlerrate bei fensterbasiertem Large-Margin-Training kontextunabhängiger Phonemmodelle auf dem GlobalPhone-Datensatz.

5.2.4 Fensterbasiertes Training kontextabhängiger Phonemmodelle auf GlobalPhone-Datensatz

Eine letzte Erweiterung der auf Fensterebene zu trainierenden Modelle wurde durch die Verwendung kontextabhängiger Modelle vorgenommen. Hier werden in einem ersten Experiment zunächst 250 Klassen basierend auf den 1,5 Stunden an Daten aus dem reduzierten Datensatz trainiert. Durch den eingeschränkten Trainingsdatensatz sollte überprüft werden, ob sich das Training auf Fensterebene verwenden lässt, um Modelle bei einem Mangel an Daten zu optimieren. Für ein zweites Experiment wurden die kompletten 15 Stunden verwendet, um 2500 Klassen zu trainieren. Hier lässt sich der Einfluss des Large-Margin-Verfahrens auf einen vollwertigen Spracherkenner untersuchen.

Bei der Nutzung von 250 Klassen setzt sich die Tendenz fort, dass mit wachsender Anzahl der Klassen die Verbesserung geringer ausfällt (Abb. 5.7). Die Wortfehlerrate beginnt bei 15.23% und hat im Laufe des Trainings ihr Minimum bei 14.96%, was einer relativen Verbesserung von gerade einmal 1.79% entsprechen würde. Durch die Schwankungen zwischen den Iterationen ist allerdings bei einer praktischen Anwendung mit einer noch geringeren Verbesserung zu rechnen. Wie bereits im kontextunabhängigen Fall sinkt die Fehlerrate auf Fenstern beständig über den Punkt hinaus,

an dem die Wortfehlerrate wieder zu steigen beginnt. Sie fällt im Trainingsverlauf von 52.76% auf 51.47%.

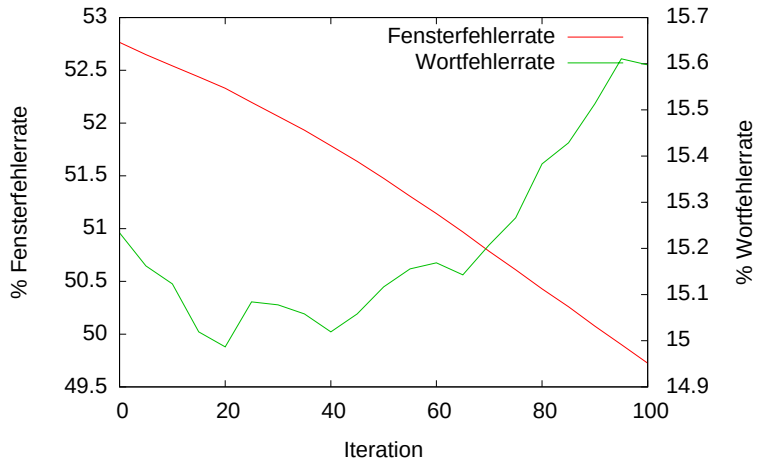


Abbildung 5.7: Fensterbasiertes Training von 250 kontextabhängigen Subpolyphonen

Die Kombination von 14 Stunden Trainingsdaten und der hohen Anzahl an Klassen schlägt sich auch auf der Dauer des Trainings nieder, da für jede Trainingsäußerung die Referenzklassen gegen alle Zustände anderer Phoneme trainiert werden. Gleichzeitig fällt die Verbesserung ausgesprochen gering aus und die Ergebnisse sind inkonsistent, was an der noch weiter reduzierten Separabelität liegen könnte. Vor dem Large-Margin-Training beträgt die Wortfehlerrate 10.33% und sinkt in dessen Verlauf nicht unter 10.3%. Auch die Fensterfehlerrate sinkt nur geringfügig: zu Beginn ist sie bei 50.15%. Im Laufe des Trainings sinkt sie zwar immer weiter, und erreicht 49.25% nach allen 100 Iterationen, jedoch zeigt sich auch hier, dass bereits früh Overtraining stattfindet (Abb. 5.8).

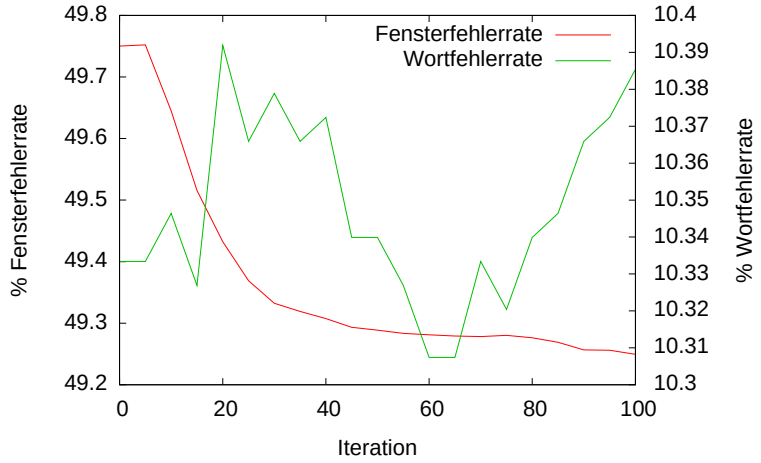


Abbildung 5.8: Fensterbasiertes Training von 2500 kontextabhängigen Subpolyphonen

Abschnitt B.4 enthält eine tabellarische Auflistung der Erkennungsergebnisse für beide Experimente. Insgesamt sprechen die Ergebnisse gegen ein Large-Margin-Training komplexerer Modelle auf Fensterebene, da die Verbesserungen den hohen Aufwand nicht rechtfertigen.

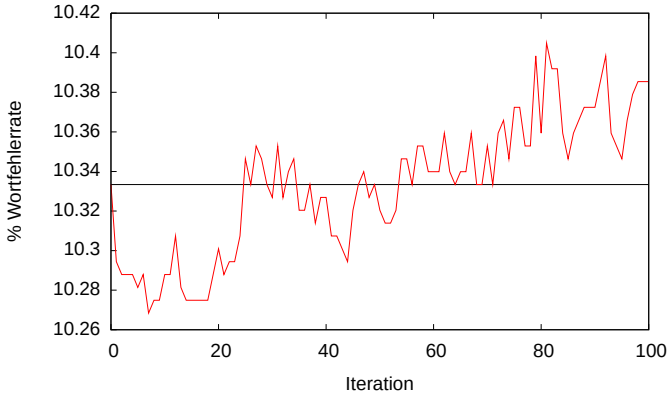
5.3 Training auf kontinuierlicher Sprache

In den vorhergehenden Experimenten war die korrekte Klassifikation von Wörtern oder Sätzen nicht das eigentliche Trainingskriterium. Die Beobachtung aus diesen Experimenten war, dass zwar die Fehlerrate auf Fensterebene infolgedessen konsistent gesunken ist, die Wortfehlerrate aber schnell Verschlechterung aufwies. Der Algorithmus wurde daher erweitert, um ein Training auf ganzen Äußerungen zu ermöglichen.

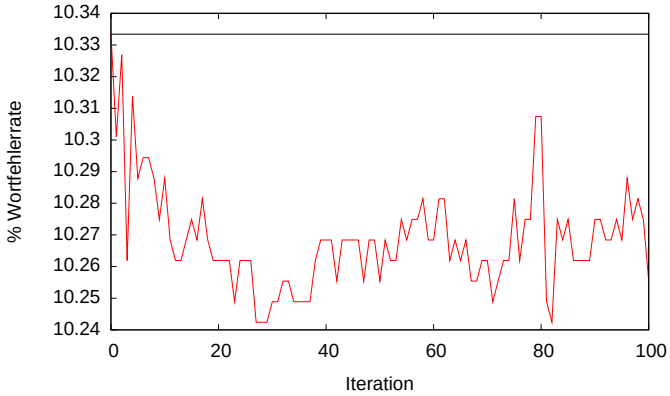
Diese Experimente wurden mit Äußerungen auf Wort- und auf Satzebene im Training und stets ganzen Sätzen im Test durchgeführt. Für jede Äußerung erhält der Trainingsalgorithmus zunächst eine Referenz vorgegeben, gegen die trainiert werden soll. Anschließend erhält er auch eine Lattice, aus der bis zu 100 beste Pfade bestimmt werden, die als Raum Ω der Konkurrenzmodelle dienen. Lattices werden vorberechnet und nicht zwischen den Trainingsiterationen neu bestimmt. Der Aufwand für eine Neuberechnung wäre nicht gerechtfertigt, da keine großen Veränderungen zu erwarten sind. Ziel des Large-Margin-Trainings ist es nicht, die in Frage kommenden Wortfolgen signifikant zu verändern, sondern den korrekten Pfad durch die Lattice zu stärken. Dennoch ist eine hohe Qualität der Lattices entscheidend.

Eine manuelle Kontrolle der beim Training mit ganzen Sätzen genutzten Lattices hat ergeben, dass ein Großteil der Varianz darin sich auf kurze Wörter und deren Varianten beschränkt. Obwohl kein Sprachmodell bei der Erzeugung verwendet wurde, finden sich so praktisch keine Abweichungen bei komplexeren Worten, die über zeitliche Verschiebung hinausgehen. Um hier eine stärkere Vielfalt zu erzwingen, wurde ein Experiment durchgeführt, bei dem für das Training die Sätze automatisiert in Wörter aufgespalten wurden. So erhaltene Lattices enthielten mehr Alternativen, aber bringen eigene Nachteile mit sich. Eine automatische Aufspaltung in Worte kann durch die Uneindeutigkeit der Wortgrenzen das Training erschweren. Auch alternative Modelle, die Wortgrenzen überschreiten oder andere Wortgrenzen erfordern, lassen sich nicht berücksichtigen. Zuletzt nimmt das JRTk im Normalfall an, dass Eingaben mit Stille beginnen und enden würden, was hier nicht mehr der Fall ist.

Leider wurde in keinem der beiden Experimente eine relevante Verbesserung erreicht. Die Ergebnisse sind in Abschnitt B.5 aufgelistet. Durch das Training auf ganzen Äußerungen sinkt die Wortfehlerrate von zunächst 10.33% auf 10.27% (Abb. 5.9a), bei Training auf einzelnen Wörtern ist das beste erreichte Ergebnis 10.24% Wortfehlerrate (Abb. 5.9b). Als ein Grund wird hier die Qualität der Lattices und daraus folgend auch der Konkurrenzmodelle vermutet. Da beim Training auf Äußerungen sämtliche Berechnungen auf der gesamten Äußerung durchgeführt werden, ist diese hohe Überlappung der Modelle von Nachteil. Hinzu kommt auch eine erschwerte Diskriminierbarkeit aufgrund der starken Zunahme bei der Anzahl der Klassen durch Verwendung kontextabhängiger Modelle, für die bereits beim Training auf Fensterebene nur wenig Verbesserung erzielt werden konnte.



(a) Lattices auf Satzebene



(b) Lattices auf Wortebene

Abbildung 5.9: Training auf Lattices, Wortfehlerrate der Ausgangsmodelle ist durch eine horizontale Linie markiert

Kapitel 6

Zusammenfassung und Ausblick

Es wurde eine Variante des Large-Margin-Trainingsverfahrens im Rahmen des JRTk implementiert und in einer Reihe von Experimenten evaluiert.

Mehrere Experimente untersuchten zunächst HMMs mit nur einem Zustand. Begonnen wurde mit Experimenten auf dem EMG-UKA-Datenkorpus. In diesem Datenkorpus liegen neben Audioaufzeichnungen der Sprache auch Aufzeichnungen von EMG-Signalen vor, wodurch das Verfahren auch für diese Signalform untersucht werden konnte. Es wurden neben Phonemmodellen auch Modelle für phonetische Klassen untersucht, für die mehrere Phoneme nach ihren Artikulationseigenschaften zusammengefasst werden. Die Experimente haben für alle Konfigurationen Verbesserungen ergeben. Die Verbesserung bei der Modellierung phonetischer Klassen fiel dabei stärker aus, als bei der Modellierung von Phonemen. Weiterhin war die Verbesserung bei der Nutzung des EMG-Signales als Datenquelle größer, als bei der Nutzung von Audioaufzeichnungen. Bei genauer Untersuchung wurde festgestellt dass nicht alle Klassen gleich vom Large-Margin-Training profitieren, eine zunehmende Optimierung weniger Klassen auf Kosten der verbleibenden wurde als Overtraining interpretiert.

Auf dem deutschsprachigen Teil des GlobalPhone-Datenkorpus wurden ebenfalls zunächst HMMs mit nur einem Zustand trainiert. Hier lag eine erheblich größere Menge an Trainingsdaten vor. Während für Experimente auf dem EMG-UKA-Datensatz nur Modelle für ein Paar von Sprecher und Sitzung trainiert wurden, wurden auf dem GlobalPhone-Korpus mit der größeren Menge an Trainingsdaten auch sprecherunabhängige Modelle trainiert. Diese Verallgemeinerung von Modellen, die nur einen einzelnen Sprecher verstehen mussten, auf Modelle, die auch für unbekannte Sprecher funktionieren müssen, wurde in zwei Schritten durchgeführt. In jedem solchen Verallgemeinerungsschritt wurde eine höhere relative Verbesserung durch Anwendung des Large-Margin-Trainings erreicht.

In weiteren Experimenten auf dem GlobalPhone-Datensatz wurde das fensterbasierte Training auf komplexere Modelle erweitert. Zunächst wurden Modelle untersucht, die Phoneme kontextunabhängig durch drei Zustände repräsentieren. In dieser Konfiguration konnten die Modelle nicht nur auf Fensterebene, sondern auch auf kontinuierlicher Sprache evaluiert werden. In den frühen Iterationen des Large-Margin-Training ist sowohl die auf Fenstern gemessene Fehlerrate, als auch die Wortfehlerrate gesunken. In späteren Iterationen ist die Fensterfehlerrate weiter gesunken,

während die Wortfehlerrate wieder gestiegen ist. Es wurde vermutet, dass dieses Ergebnis mit dem bereits zuvor beobachteten Overtraining verbunden ist.

Eine weitere Vergrößerung des Klassenraumes stellte das Training kontextabhängiger Subpolyphone dar. Es wurden neben 250 Klassen auf einem Teil der Trainingsdaten auch 2500 auf dem kompletten Datensatz trainiert. Wieder wurde mit den durch Anwendung des Large-Margin-Training angepassten Modellen neben der Fensterfehlerrate auch die Wortfehlerrate auf einem Testdatensatz bestimmt. Vergleichbar mit den Ergebnissen im kontextunabhängigen Fall wurden Verbesserungen der Wortfehlerrate nur bei einem Teil der Gradientenabstiegsiterationen beobachtet, während die Fensterfehlerrate weiter sank. Weiterhin waren nach jeder Vergrößerung des Klassenraumes Verbesserungen der Wortfehlerrate schwächer ausgeprägt. Gleichzeitig ist die Trainingsdauer, durch eine Zunahme an Trainingsdaten und Konkurrenzklassen, stark gestiegen.

Aus den Experimentergebnissen lässt sich ableiten, dass Large-Margin-Training gerade für Konfigurationen mit einem Modellraum von wenigen zu unterscheidenden Klassen durchaus deutliche Verbesserungen erzielen kann. Während aber für komplexere Modelle mit vielen Klassen der Trainingsaufwand erheblich ansteigt, gehen die Verbesserungen schnell zurück.

Durch ein Training auf ganzen Äußerungen sollte überprüft werden, ob eine Optimierung der Bewertung größerer Einheiten dem Verfahren zugutekommt. Da hier die Bewertungsfunktion direkter mit der korrekten Klassifikation der Trainingsdaten zusammenhängt, sollte sie weniger anfällig gegenüber Schwankungen bei der Zuordnung auf Phonemebene werden. Ein Problem, das hier angetroffen wurde, ist die Generierung von Konkurrenzmodellen. Die verwendeten Lattices boten vorwiegend nur geringfügig abweichende Pfade, wodurch auch die erzeugten Listen von Konkurrenzpfaden nur wenig Varianz hatten und damit eine schlechte Grundlage für diskriminatives Training darstellten. Es bleibt zunächst offen, ob mit einer besseren Liste von Konkurrenzpfaden auch eine Verbesserung der Erkennungsleistung erreicht werden kann.

Ebenfalls einer genauen Untersuchung bedarf der Einfluss durch alternative Definitionen des Verfahrens, da sie sich zum Teil erheblich unterscheiden (vgl. Abschnitt 2.4.4). In dieser Arbeit wurde nur logarithmierte Likelihood mit einem relativen Marginbegriff untersucht. Es mussten eine Reihe von Entscheidungen getroffen werden, die in Abschnitt 3 besprochen wurden. Es bietet sich an, in weiterführenden Arbeiten zu untersuchen, inwieweit durch Abwandlung des Verfahrens bessere Ergebnisse erreicht werden können.

Anhang A

Codebeispiele

In diesem Abschnitt ist die Nutzung des `LMHMMAccuSet`-Objektes über die Tcl-Schnittstelle sowohl für Training auf Fenster- als auch Äußerungsebene beispielhaft dargestellt. Beide Beispiele setzen eine vorherige Initialisierung analog zu einem MLE-Training voraus und beschränken sich nur auf das eigentliche Training.

A.1 Training auf Fensterebene

Listing A.1: Training auf Fensterebene

```
lmhmm clear
foreach spk $speakerList {
  labelbox clear
  labelbox load $labelPath
  foreachSegment utt uttDB $spk {
    set uttInfo [uttDB uttInfo $spk $utt]
    makeArray uttArray $uttInfo
    featureSet eval $uttInfo
#   erstelle HMM fuer Aeusserung
    hmm makeUtterance $uttArray(TEXT)
        -optWord $optWord -variants $variants
    set hmmList [lindex [hmm.phoneGraph] 0]
    set pathList [lrange [labelbox:$utt] 5 end]
    set uttLen [featureSet:$featureName configure -frameN]
    for { set frameX 0 } { $frameX < $uttLen }
        { incr frameX } {
      set phoneX [lindex $pathList $frameX 4 2]
      set phoneName [lindex $hmmList $phoneX]
#     trainiere Modell von phoneName gegen
#     alle anderen (bis auf excludelist)
      lmhmm accuFrame $phoneName $frameX
          -epsilon $epsilon -exclude $excludelist
    }
  }
}
lmhmm update -stepsize $stepsize
```

A.2 Training auf Äußerungsebene

Listing A.2: Training auf Satzebene

```
lmhmm clear
foreach spk $speakerList {
  foreachSegment utt uttDB $spk {
    featureSet eval [uttDB uttInfo $spk $utt]

#   finde Referenzpfad auf Fensterebene
    glat clear
    addRefToGLat glat $spk $utt
    glat initAddon hmm pathTmpP
        -mode $mode -verbose 0
    path alignGlat hmm glat pathTmpP
        -mode $mode -thresh $thresh -verbose 0
    glat clearAddon
#   uebergebe Pfad als Referenz
    lmhmm setReference path senoneSet

#   lade vollstaendige, vorberechnete Lattice
    glat read $latticeFile
    glat purify
#   Vorberechnung von Bewertungsinformation
    glat initAddon hmm pathTmpP
        -mode $mode -verbose 0
    path alignGlat hmm glat pathTmpP
        -mode $mode -thresh $thresh -verbose 0
#   Training der Lattice gegen den Referenzpfad
    lmhmm accuGLat glat -N 100
    glat clearAddon

#   Abschliessende Berechnungen
    lmhmm finishReference
  }
}
lmhmm update -stepsize $stepsize
```

Anhang B

Ergebnisse

Dieser Abschnitt enthält die tabellarische Auflistung der in vorhergehenden Abschnitten analysierten Erkennungsergebnisse. Iteration 0 bezeichnet bei allen Experimenten die Ausgangsmodelle für das Large-Margin-Training.

B.1 Fensterbasiertes Training atomarer Modelle auf dem EMG-UKA-Datensatz

B.1.1 Phonemerkennung auf Audiosignal

Iteration	Fehlerrate	Standardabweichung
0	48.09%	6.23
5	47.48%	6.20
10	46.91%	6.17
15	46.49%	6.15
20	46.14%	6.12
25	45.97%	6.11
30	45.81%	6.08
35	45.73%	6.08
40	45.67%	6.07
45	45.70%	6.04
50	45.69%	6.02
55	45.72%	5.97
60	45.78%	5.94
65	45.84%	5.96
70	45.90%	5.94
75	46.01%	5.94
80	46.06%	5.93
85	46.13%	5.95
90	46.23%	5.95
95	46.31%	5.95
100	46.40%	5.94

Fortsetzung auf der nächsten Seite.

Tabelle B.1 – Fortsetzung von der vorhergehenden Seite.

Iteration	Fehlerrate	Standardabweichung
105	46.48%	5.96
110	46.56%	5.95
115	46.63%	5.98
120	46.71%	5.97
125	46.78%	5.96
130	46.89%	5.98
135	46.96%	5.99
140	47.05%	6.01
145	47.09%	6.01
150	47.16%	5.98
155	47.23%	5.97
160	47.31%	5.97

B.1.2 Erkennung phonetischer Klassen auf Audiosignal

Iteration	Fehlerrate	Standardabweichung
0	25.56%	4.88
5	23.71%	4.52
10	22.32%	4.30
15	21.53%	4.05
20	21.16%	3.95
25	21.01%	3.82
30	21.00%	3.71
35	21.14%	3.63
40	21.33%	3.59
45	21.56%	3.49
50	21.78%	3.55
55	22.08%	3.60
60	22.35%	3.59
65	22.69%	3.63
70	22.97%	3.70
75	23.26%	3.76
80	23.51%	3.82
85	23.68%	3.86
90	23.83%	3.88
95	23.95%	3.88
100	24.08%	3.91
105	24.17%	3.91
110	24.22%	3.93
115	24.30%	3.97

Fortsetzung auf der nächsten Seite.

Tabelle B.2 – Fortsetzung von der vorhergehenden Seite.

Iteration	Fehlerrate	Standardabweichung
120	24.34%	3.98
125	24.36%	3.98
130	24.39%	3.98
135	24.40%	4.01
140	24.41%	4.01
145	24.41%	4.01
150	24.40%	4.02
155	24.38%	4.01
160	24.38%	4.02

B.1.3 Phonemerkennung auf EMG-Signal

Iteration	Fehlerrate	Standardabweichung
0	71.87%	6.19
5	71.07%	6.27
10	70.37%	6.32
15	69.83%	6.38
20	69.40%	6.42
25	69.00%	6.47
30	68.66%	6.45
35	68.38%	6.43
40	68.14%	6.42
45	67.95%	6.38
50	67.80%	6.38
55	67.69%	6.34
60	67.60%	6.31
65	67.52%	6.27
70	67.44%	6.27
75	67.38%	6.24
80	67.30%	6.24
85	67.24%	6.30
90	67.20%	6.29
95	67.17%	6.30
100	67.18%	6.29
105	67.17%	6.29
110	67.16%	6.28
115	67.15%	6.27
120	67.12%	6.24
125	67.11%	6.24
130	67.10%	6.23

Fortsetzung auf der nächsten Seite.

Tabelle B.3 – Fortsetzung von der vorhergehenden Seite.

Iteration	Fehlerrate	Standardabweichung
135	67.12%	6.26
140	67.12%	6.28
145	67.12%	6.27
150	67.13%	6.27
155	67.13%	6.28
160	67.13%	6.28

B.1.4 Erkennung phonetischer Klassen auf EMG-Signal

Iteration	Fehlerrate	Standardabweichung
0	42.97%	6.97
5	38.04%	6.74
10	34.58%	6.40
15	32.16%	6.03
20	30.52%	5.59
25	29.34%	5.13
30	28.62%	4.86
35	28.17%	4.54
40	28.01%	4.21
45	27.98%	4.03
50	28.05%	3.92
55	28.26%	3.80
60	28.45%	3.77
65	28.65%	3.72
70	28.82%	3.66
75	28.98%	3.61
80	29.09%	3.58
85	29.21%	3.52
90	29.25%	3.49
95	29.31%	3.54
100	29.37%	3.55
105	29.41%	3.55
110	29.41%	3.58
115	29.42%	3.60
120	29.42%	3.62
125	29.41%	3.61
130	29.41%	3.63
135	29.39%	3.63
140	29.39%	3.66
145	29.36%	3.67

Fortsetzung auf der nächsten Seite.

Tabelle B.4 – Fortsetzung von der vorhergehenden Seite.

Iteration	Fehlerrate	Standardabweichung
150	29.34%	3.69
155	29.33%	3.70
160	29.32%	3.69

B.2 Fensterbasiertes Training atomarer Modelle auf dem GlobalPhone-Datensatz

B.2.1 Sprecherabhängiges Training

Iteration	Fehlerrate
0	33.98%
1	33.79%
2	33.61%
4	33.29%
7	32.94%
15	32.45%
20	32.28%
25	32.22%
30	32.22%
35	32.25%
40	32.32%
45	32.41%
50	32.51%
55	32.64%
60	32.79%
65	32.92%
70	33.05%
75	33.22%
80	33.39%
85	33.53%
90	33.72%
95	33.89%
100	34.05%

B.2.2 Sprecherunabhängiges Training, Test auf bekannten Sprechern

Iteration	Fehlerrate
0	40.53%
5	40.12%
10	39.67%
15	39.32%
20	39.05%
25	38.83%
30	38.69%
35	38.53%
40	38.46%
45	38.38%
50	38.31%
55	38.26%
60	38.24%
65	38.20%
70	38.20%
75	38.20%
80	38.18%
85	38.20%
90	38.22%
95	38.24%
100	38.28%

B.2.3 Sprecherunabhängiges Training, Test auf unbekannten Sprechern

Iteration	Fehlerrate
0	44.93%
5	44.30%
10	43.51%
15	43.00%
20	42.70%
25	42.46%
30	42.23%
35	42.05%
40	41.89%
45	41.77%
50	41.66%
55	41.58%
60	41.46%
65	41.37%
70	41.28%

Fortsetzung auf der nächsten Seite.

Tabelle B.7 – Fortsetzung von der vorhergehenden Seite.

Iteration	Fehlerrate
75	41.20%
80	41.17%
85	41.12%
90	41.08%
95	41.04%
100	41.03%

B.3 Fensterbasiertes Training kontextunabhängiger Phonemmodelle

B.3.1 Training kontextunabhängiger Subpolyphone

Iteration	Fensterfehlerrate	Wortfehlerrate
0	44.36%	17.92%
5	44.02%	17.77%
10	43.76%	17.61%
15	43.52%	17.24%
20	43.32%	17.29%
25	43.15%	17.48%
30	42.99%	17.31%
35	42.83%	17.33%
40	42.70%	17.38%
45	42.58%	17.33%
50	42.46%	17.37%
55	42.34%	17.28%
60	42.21%	17.20%
65	42.11%	17.14%
70	42.00%	17.00%
75	41.91%	17.05%
80	41.84%	16.97%
85	41.76%	17.18%
90	41.67%	17.26%
95	41.59%	17.40%
100	41.50%	17.49%
105	41.42%	17.57%
110	41.35%	17.64%
115	41.28%	17.59%
120	41.23%	17.62%

B.4 Fensterbasiertes Training kontextabhängiger Phonemmodelle

B.4.1 Training von 250 kontextabhängigen Modellen

Iteration	Fensterfehlerrate	Wortfehlerrate
0	52.76%	15.23%
5	52.65%	15.16%
10	52.54%	15.12%
15	52.44%	15.02%
20	52.33%	14.99%
25	52.20%	15.08%
30	52.06%	15.08%
35	51.93%	15.06%
40	51.79%	15.02%
45	51.64%	15.06%
50	51.48%	15.12%
55	51.31%	15.16%
60	51.14%	15.17%
65	50.97%	15.14%
70	50.78%	15.21%
75	50.61%	15.27%
80	50.43%	15.38%
85	50.26%	15.43%
90	50.08%	15.51%
95	49.90%	15.61%
100	49.72%	15.60%

B.4.2 Training von 2500 kontextabhängigen Modellen

Iteration	Fensterfehlerrate	Wortfehlerrate
0	49.75%	10.33%
5	49.75%	10.33%
10	49.64%	10.35%
15	49.52%	10.33%
20	49.43%	10.39%
25	49.37%	10.37%
30	49.33%	10.38%
35	49.32%	10.37%
40	49.31%	10.37%
45	49.29%	10.34%
50	49.29%	10.34%

Fortsetzung auf der nächsten Seite.

Tabelle B.10 – Fortsetzung von der vorhergehenden Seite.

Iteration	Fensterfehlerrate	Wortfehlerrate
55	49.28%	10.33%
60	49.28%	10.31%
65	49.28%	10.31%
70	49.28%	10.33%
75	49.28%	10.32%
80	49.28%	10.34%
85	49.27%	10.35%
90	49.26%	10.37%
95	49.26%	10.37%
100	49.25%	10.39%

B.5 Training auf Äußerungsebene

B.5.1 Training auf Äußerungsebene

Iteration	Wortfehlerrate
0	10.33%
1	10.29%
2	10.29%
3	10.29%
4	10.29%
5	10.28%
6	10.29%
7	10.27%
8	10.27%
9	10.27%
10	10.29%
11	10.29%
12	10.31%
13	10.28%
14	10.27%
15	10.27%
16	10.27%
17	10.27%
18	10.27%
19	10.29%
20	10.30%
21	10.29%
22	10.29%
23	10.29%

Fortsetzung auf der nächsten Seite.

Tabelle B.11 – Fortsetzung von der vorhergehenden Seite.

Iteration	Wortfehlerrate
24	10.31%
25	10.35%
26	10.33%
27	10.35%
28	10.35%
29	10.33%
30	10.33%
31	10.35%
32	10.33%
33	10.34%
34	10.35%
35	10.32%
36	10.32%
37	10.33%
38	10.31%
39	10.33%
40	10.33%
41	10.31%
42	10.31%
43	10.30%
44	10.29%
45	10.32%
46	10.33%
47	10.34%
48	10.33%
49	10.33%
50	10.32%
51	10.31%
52	10.31%
53	10.32%
54	10.35%
55	10.35%
56	10.33%
57	10.35%
58	10.35%
59	10.34%
60	10.34%
61	10.34%
62	10.36%
63	10.34%
64	10.33%
65	10.34%
66	10.34%
67	10.36%

Fortsetzung auf der nächsten Seite.

Tabelle B.11 – Fortsetzung von der vorhergehenden Seite.

Iteration	Wortfehlerrate
68	10.33%
69	10.33%
70	10.35%
71	10.33%
72	10.36%
73	10.37%
74	10.35%
75	10.37%
76	10.37%
77	10.35%
78	10.35%
79	10.40%
80	10.36%
81	10.40%
82	10.39%
83	10.39%
84	10.36%
85	10.35%
86	10.36%
87	10.37%
88	10.37%
89	10.37%
90	10.37%
91	10.39%
92	10.40%
93	10.36%
94	10.35%
95	10.35%
96	10.37%
97	10.38%
98	10.39%
99	10.39%
100	10.39%

B.5.2 Training auf Wortebene

Iteration	Wortfehlerrate
0	10.33%
1	10.30%
2	10.33%

Fortsetzung auf der nächsten Seite.

Tabelle B.12 – Fortsetzung von der vorhergehenden Seite.

Iteration	Wortfehlerrate
3	10.26%
4	10.31%
5	10.29%
6	10.29%
7	10.29%
8	10.29%
9	10.27%
10	10.29%
11	10.27%
12	10.26%
13	10.26%
14	10.27%
15	10.27%
16	10.27%
17	10.28%
18	10.27%
19	10.26%
20	10.26%
21	10.26%
22	10.26%
23	10.25%
24	10.26%
25	10.26%
26	10.26%
27	10.24%
28	10.24%
29	10.24%
30	10.25%
31	10.25%
32	10.26%
33	10.26%
34	10.25%
35	10.25%
36	10.25%
37	10.25%
38	10.26%
39	10.27%
40	10.27%
41	10.27%
42	10.26%
43	10.27%
44	10.27%
45	10.27%
46	10.27%

Fortsetzung auf der nächsten Seite.

Tabelle B.12 – Fortsetzung von der vorhergehenden Seite.

Iteration	Wortfehlerrate
47	10.26%
48	10.27%
49	10.27%
50	10.26%
51	10.27%
52	10.26%
53	10.26%
54	10.27%
55	10.27%
56	10.27%
57	10.27%
58	10.28%
59	10.27%
60	10.27%
61	10.28%
62	10.28%
63	10.26%
64	10.27%
65	10.26%
66	10.27%
67	10.26%
68	10.26%
69	10.26%
70	10.26%
71	10.25%
72	10.26%
73	10.26%
74	10.26%
75	10.28%
76	10.26%
77	10.27%
78	10.27%
79	10.31%
80	10.31%
81	10.25%
82	10.24%
83	10.27%
84	10.27%
85	10.27%
86	10.26%
87	10.26%
88	10.26%
89	10.26%
90	10.27%

Fortsetzung auf der nächsten Seite.

Tabelle B.12 – Fortsetzung von der vorhergehenden Seite.

Iteration	Wortfehlerrate
91	10.27%
92	10.27%
93	10.27%
94	10.27%
95	10.27%
96	10.29%
97	10.27%
98	10.28%
99	10.27%
100	10.26%

Literaturverzeichnis

[1] Bahl, L., P. Brown, P. De Souza und R. Mercer: *Maximum mutual information estimation of hidden Markov model parameters for speech recognition*. In: *Proc. ICASSP*, Bd. 11, S. 49–52. IEEE, 1986.

[2] Chen, S. und J. Goodman: *An empirical study of smoothing techniques for language modeling*. In: *Proc. 34th annual meeting on Association for Computational Linguistics*, S. 310–318. Association for Computational Linguistics, 1996.

[3] Cheng, C.-C., F. Sha und L. K. Saul: *A fast online algorithm for large margin training of continuous density hidden Markov models.*. In: *Proc. Interspeech*, S. 668–671, 2009.

[4] Dempster, A., N. Laird und D. Rubin: *Maximum likelihood from incomplete data via the EM algorithm*. Journal of the Royal Statistical Society. Series B (Methodological), S. 1–38, 1977.

[5] Finke, M., P. Geutner, H. Hild, T. Kemp, K. Ries und M. Westphal: *The karlsruhe-verbmobil speech recognition engine*. In: *Proc. ICASSP*, Bd. 1, S. 83–86. IEEE, 1997.

[6] Fügen, C., F. Metze und H. Soltau: *JRTk and JANUS*.

[7] Gopalakrishnan, P., D. Kanevsky, A. Nadas und D. Nahamoo: *A generalization of the Baum algorithm to rational objective functions*. In: *Proc. ICASSP*, S. 631–634. IEEE, 1989.

[8] Gupta, M. und Y. Chen: *Theory and Use of the EM Algorithm*. Now Pub, 2011.

[9] He, X., L. Deng und W. Chou: *A novel learning method for hidden Markov models in speech and audio processing*. In: *Proc. 8th Workshop on Multimedia Signal Processing*, S. 80–85. IEEE, 2006.

[10] Huang, X., A. Acero, H.-W. Hon et al.: *Spoken language processing*. Prentice Hall PTR New Jersey, 2001.

[11] Janke, M., M. Wand und T. Schultz: *A Spectral Mapping Method for EMG-based Recognition of Silent Speech.*. In: *Proc. B-Interface*, S. 22–31, 2010.

[12] Jiang, H.: *Discriminative training for large margin HMMs*. Dept. Comput. Sci. Eng., York Univ., Tech. Rep. CS-2004-01, 2004.

[13] Jiang, H. und X. Li: *Incorporating training errors for large margin HMMs under semi-definite programming framework*. In: *Proc. ICASSP*, Bd. 4, S. IV–629. IEEE, 2007.

[14] Jiang, H., X. Li und C. Liu: *Large margin hidden Markov models for speech recognition*. IEEE Transactions on Audio, Speech, and Language Processing, 14(5):1584 –1595, sept. 2006.

[15] Jou, S.-C. S., T. Schultz, M. Walliczek, F. Kraft und A. Waibel: *Towards continuous speech recognition using surface electromyography.*. In: *Proc. Interspeech*, 2006.

[16] Juang, B.-H., W. Hou und C.-H. Lee: *Minimum classification error rate methods for speech recognition*. IEEE Transactions on Speech and Audio Processing, 5(3):257–265, 1997.

[17] Juang, B.-H. und S. Katagiri: *Discriminative learning for minimum error classification [pattern recognition]*. IEEE Transactions on Signal Processing, 40(12):3043–3054, 1992.

[18] Le Roux, J. und E. McDermott: *Optimization methods for discriminative training.*. In: *Proc. Interspeech*, S. 3341–3344, 2005.

[19] Li, X. und H. Jiang: *Solving large-margin hidden Markov model estimation via semidefinite programming*. IEEE Transactions on Audio, Speech, and Language Processing, 15(8):2383–2392, 2007.

[20] Li, X., H. Jiang und C. Liu: *Large margin HMMs for speech recognition*. In: *Proc. ICASSP*, Bd. 5, S. v–513. IEEE, 2005.

[21] Liu, C., H. Jiang und X. Li: *Discriminative training of CDHMMs for maximum relative separation margin*. In: *Proc. ICASSP*, Bd. 1, S. 101–104, 2005.

[22] Liu, C., H. Jiang und L. Rigazio: *Maximum relative margin estimation of HMMs based on N-best string models for continuous speech recognition*. In: *Proc. ASRU*, S. 420–425. IEEE, 2005.

[23] Liu, C., H. Jiang und L. Rigazio: *Recent Improvement on Maximum Relative Margin Estimation of HMMS for Speech Recognition*. In: *Proc. ICASSP*, Bd. 1, S. I, may 2006.

[24] Maier-Hein, L.: *Speech recognition using surface electromyography*. Diplomarbeit, Fakultät für Informatik der Universität Karlsruhe, 2005.

[25] McDermott, E., T. J. Hazen, J. Le Roux, A. Nakamura und S. Katagiri: *Discriminative training for large-vocabulary speech recognition using minimum classification error*. IEEE Transactions on Audio, Speech, and Language Processing, 15(1):203–223, 2007.

[26] Normandin, Y.: *Hidden Markov models, maximum mutual information estimation, and the speech recognition problem*. Doktorarbeit, McGill University, Montreal, 1991.

[27] Povey, D., D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon und K. Visweswariah: *Boosted MMI for model and feature-space discriminative training*. In: *Proc. ICASSP*, S. 4057–4060. IEEE, 2008.

[28] Rabiner, L.: *A tutorial on hidden Markov models and selected applications in speech recognition*. Proc. IEEE, 77(2):257–286, 1989.

[29] Rogina, I.: *Paramaterraumoptimierung für Diktiersysteme mit unbeschränktem Vokabular*. Doktorarbeit, Fakultät für Informatik der Universität Karlsruhe, 1997.

[30] Saon, G. und D. Povey: *Penalty function maximization for large margin HMM training.*. In: *Proc. Interspeech*, S. 920–923, 2008.

[31] Schultz, T.: *Globalphone: a multilingual speech and text database developed at karlsruhe university.*. In: *Proc. Interspeech*, S. 345–348, 2002.

[32] Schultz, T. und M. Wand: *Modeling coarticulation in EMG-based continuous speech recognition*. Speech Communication, 52(4):341–353, 2010.

[33] Sha, F. und L. K. Saul: *Large margin Gaussian mixture modeling for phonetic classification and recognition*. In: *Proc. ICASSP*, Bd. 1, S. I–I. IEEE, 2006.

[34] Sha, F. und L. K. Saul: *Large margin hidden Markov models for automatic speech recognition*. In: *Advances in neural information processing systems*, S. 1249–1256, 2006.

[35] Sha, F. und L. K. Saul: *Comparison of large margin training to other discriminative methods for phonetic recognition by hidden Markov models*. In: *Proc. ICASSP*, Bd. 4, S. IV–313. IEEE, 2007.

[36] Soltau, H., F. Metze, C. Fugen und A. Waibel: *A one-pass decoder based on polymorphic linguistic context assignment*. In: *Proc. ASRU*, S. 214–217. IEEE, 2001.

[37] Yin, Y. und H. Jiang: *A compact semidefinite programming (SDP) formulation for large margin estimation of HMMs in speech recognition*. In: *Proc. ASRU*, S. 312–317. IEEE, 2007.

[38] Yin, Y. und H. Jiang: *A fast optimization method for large margin estimation of HMMs based on second order cone programming.*. In: *Proc. Interspeech*, S. 34–37, 2007.

[39] Yu, D., L. Deng, X. He und A. Acero: *Large-margin minimum classification error training for large-scale speech recognition tasks*. In: *Proc. ICASSP*, Bd. 4, S. IV–1137. IEEE, 2007.