ETH Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Studienarbeit SA-2003.05

Institut für technische Informatik und Kommunikationsnetze
Fachgruppe der Sprachverarbeitung

Professor: Prof. Lothar Thiele

Betreuung: René Beutler, Dr. Beat Pfister

Thema:

Lautmodellierung mit neuronalen Netzen

Florian Kaufmann, Markus Tuor

Wintersemester 2002/2003

Zusammenfassung

Diese Arbeit hat zum Ziel, ein möglichst effizientes neuronales Netz zu finden, mit welchem Phoneme in einem Sprachsignal erkannt werden können. Dazu werden verschiedene Fehlermasse definiert, mit welchen die Erkennungsleistung eines neuronalen Netzes bestimmt werden kann und verschiedene Netze untereinander verglichen werden können. Als entscheidendes Fehlermass wird das Resultat eines Wordspotters verwendet.

Untersucht werden gängige Parameter neuronaler Netze wie Anzahl Neuronen, Anzahl Hidden Layer, Anzahl Kontext Frames sowie Parameter des Backpropagation Trainingsalgorithmus wie Gain und Momentum. Weiter werden unterschiedliche Trainingsmethoden untersucht wie gleichverteiltes Training, d.h. jedes Phonem wird gleich oft gelehrt, und pädagogisches Training, d.h. schlecht erkannte Phoneme werden häufiger gelehrt. Um ein neuronales Netz zu trainieren, braucht es eine Vorgabe, in der der gewünschte Output steht. Diese Vorgabe wurde in unserem Fall von Hidden Markov Models erstellt. Die zeitliche Abgrenzung verschiedener Phoneme ist jedoch ungenau, und somit wird dem Netz etwas Falsches gelehrt. Mit einem New Alignment genannten Algorithmus wird während des Trainings die Vorgabe vom Netz selbst periodisch korrigiert.

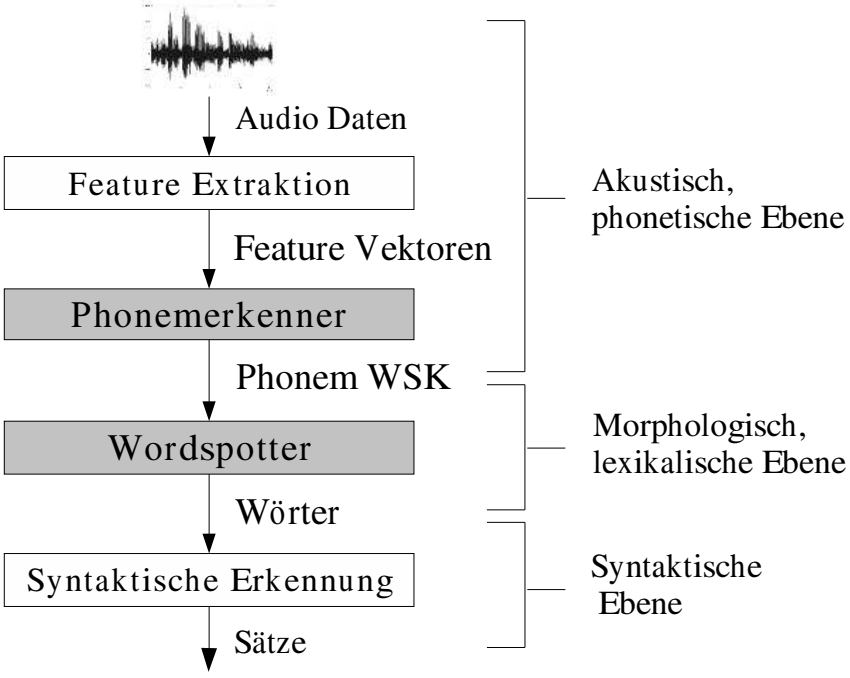
In Bezug auf das wichtigste Fehlermass, die Erkennungsleistung des Wörtererkenners, hat sich gezeigt, dass diejenigen neuronale Netze die Effizientesten sind, welche eine hohe und ausgewogene Erkennungsleistung aufweisen. Dies wird am Besten mit pädagogischem Training erreicht.

Inhaltsverzeichnis

1 Einleitung

1.1 Das Erkennen kontinuierlicher Sprache

Die Spracherkennung befasst sich mit dem Problem, eine geeignete Funktion zu finden, welche vorhandener akustischer Information eine sprachliche Bedeutung zuweist. Diese Erkennungsaufgabe kann auf verschiedene Ebenen aufgeteilt werden (Figur 1). In der untersten, akustisch-phonetischen Ebene, werden zuerst aus den Audiodaten Merkmale (Features) extrahiert, die den Klang des Sprachsignals beschreiben. Diese Werte werden der ersten Erkennungsstufe zugeführt, welche dann die Wahrscheinlichkeit berechnet, dass ein bestimmtes Phonem im beobachteten Signalabschnitt auftritt. Diese Information wird in der morphologisch-lexikalischen Schicht weiterverarbeitet und der Wordspotter versucht die Phoneme in ganze Wörter zusammenzufassen. Die Stellung dieser Wörter und die Anwendung grammatischer Regeln wird auf der syntaktischen Ebene analysiert, um abschliessend die eigentliche Bedeutung des Gesprochenen möglichst gut erkennen zu können.



Figur 1: *Datenfluss durch die Erkennungsstufen*

Der Umfang und die Komplexität der einzelnen Erkennungsaufgaben hängen von den gewünschten Anforderungen ab. Je nachdem ob das System kontinuierlich gesprochene Sprache erkennen, oder beispielsweise "nur" auf dedizierte Schlüsselwörter reagieren soll, sind unterschiedliche Ansätze geeignet. Ein weiteres wesentliches Kriterium ist die Sprecherabhängigkeit. Im sprecherunabhängigen Fall, muss eine vom Sprecher und dessen spezifischen Spracheigenschaften losgelöste Erkennung stattfinden, was einen höheren Anspruch an die Verallgemeinerungsfähigkeit stellt.

Diese Arbeit ist in den untersten zwei Ebenen der kontinuierlichen Spracherkennung angesiedelt (Figur 1, hervorgehobener Bereich). Sie befasst sich mit der Implementation zweier Systeme; ein erstes, welches Phoneme erkennt und ein zweites, welches Wörter findet. Dabei dient letzteres ausschliesslich dazu, die Erkennungsleistung des Phonemerkenners zu bewerten.

1.1.1 Phoneme erkennen

Für die Phonemerkennung sind verschiedene Ansätze bekannt. Am verbreitetsten ist die Erkennung mittels eines statistischen Modells, des HMM (Hidden-Markov-Model).

Ein HMM ist ein stochastischer endlicher Automat, der aus einer endlichen Menge von Zuständen mit je einer zugeordneten Wahrscheinlichkeitsverteilung besteht. Dabei wird für jedes Phonem ein solches Modell in Form eines Automaten erstellt, um damit die Auftrittswahrscheinlichkeit des Phonems an einer Stelle im Sprachsignal zu bestimmen. Weitere Informationen zu diesem Ansatz sind in [2] zu finden.

Ein anderer Ansatz, welcher in dieser Arbeit aufgegriffen wird, stellt das MLP (Multi-Layer-Perceptron) dar. Beim MLP handelt es sich um ein künstliches neuronales Netz, welches für Erkennungssaufgaben in verschiedenen Bereichen (z.B. in der Bildverarbeitung) eingesetzt wird. MLPs können beliebig komplexe Klassifizierungsaufgaben lösen. Der Output des MLP kann als Wahrscheinlichkeit interpretiert werden, dass der präsentierte Input einer bestimmten Klassse angehört. Dies sind Gründe, welche darauf hinweisen, dass MLPs auch für die Spracherkennung geeignet sind.

Da MLPs dazu dienen, statische Muster zu erkennen, können für die Verarbeitung dynamischer Prozesse Verzögerungselemente eingesetzt werden. Dadurch hat das Netz die Eingangsdaten während eines ganzen Zeitfensters gespeichert und kann zeitlichen Kontext zur Erkennung miteinbeziehen.

1.2 Aufgabenstellung

Unsere primäre Aufgabe besteht darin, einen sprecherabhängigen Phonemerkenner mittels eines neuronalen Netzes zu realisieren. Dies beinhaltet die Auswahl von Netzen und Techniken, welche für diese Problemlösung geeignet sind, sowie das Bestimmen von Fehlermassen, um die Lösungsansätze untereinander zu vergleichen.

Damit die Erkennungsleistung verschiedener Netze miteinander verglichen werden kann, müssen geeignete Masse definiert werden. Die aussagekräftigste Bewertung des Phonemerkenners liefert uns das nächsthöhere Erkennungssystem: der Wordspotter. Denn das Ergebnis des Wordspotters zeigt auf, wie Erfolgreich die gesamte Erkennung bis zu dieser Stufe arbeitet. Da aus Effizienzgründen die Bewertung nicht für jedes Experiment mittels Wordspotter ermittelt werden kann, müssen zusätzliche Fehlermasse definiert werden, welche die Erkennungsleistung auf der Laut- und Wortebene bestimmen.

Bei der Suche nach dem geeigneten Netz muss ein mehrdimensionaler Lösungsraum untersucht werden, gegeben durch die grosse Zahl von Netz- und Trainingsparametern. Um die Lösungsfindung möglichst effizient zu gestalten, müssen Skripts geschrieben werden,

die das Experimentieren möglichst automatisieren.

Aufgaben und Ziele

1. Finden eines geeigneten neuronalen Netzes für die Phonemerkennung
2. Implementieren eines Wordspotters
3. Definieren geeigneter Masse, um die Erkennungsleistung zu bewerten
4. Automatisierte Trainingsmethoden entwickeln

2 Phonemerkennung

2.1 Einführung in neuronale Netze

Neuronale Netze sind aus Neuronen aufgebaut, die mit gewichteten, gerichteten Kanten verbunden sind. Figur 2 zeigt ein einfaches neuronales Netz. Hierbei handelt es sich um ein MLP (Multi Layer Perceptron), welches durch den schichtweisen Aufbau der Neuronen charakterisiert ist. Die Eingangsneuronen in der Eingangsschicht (Input Layer) werden aktiviert, indem beliebige Werte angelegt werden. Diese Aktivierungen propagieren dann über die Verbindungen durch die Neuronen der Zwischenschicht (Hidden Layer) und produzieren eine Antwort an den Ausgangsneuronen des Output Layers.

Die einzelnen Neuronen besitzen eine so genannte Aktivierungsfunktion. Aus allen in einem Neuron ankommenden Werten wird zuerst die Summe gebildet. Diese Summe wird als Argument der Aktivierungsfunktion übergeben und das Resultat entspricht dann dem Ausgang dieses Neurons.

Für unsere Arbeit haben wir Time Delay Neuronal Networks (TDNN) verwendet, wie sie vom Softwarepaket NICO bereitgestellt werden. Bei unserem TDNN handelt es sich um ein erweitertes Multi Layer Perceptron (MLP), ein neuronales Netz, das für komplexe Erkennungsaufgaben geeignet ist. Das TDNN unterscheidet sich vom MLP insofern, als dass es für dynamische Erkennungsaufgaben angepasst wurde. Das heisst, für die Verarbeitung zeitkontinuierlicher Prozesse können Verzögerungselemente eingesetzt werden. Dadurch hat das Netz die Eingangsdaten während eines ganzen Zeitfensters gespeichert und kann zeitlichen Kontext zur Erkennung miteinbeziehen.

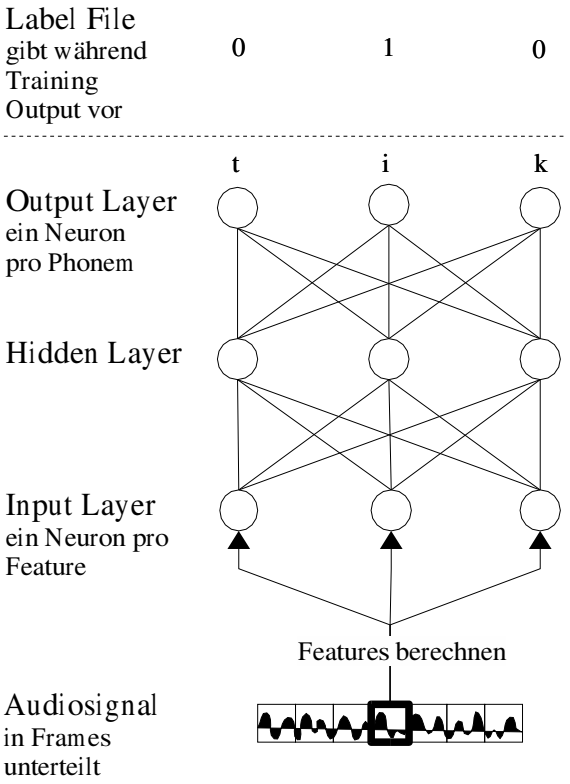
Weitere Informationen zum Aufbau und zur Funktionsweise von neuronalen Netzen sind in [1] und [3] zu finden.

Das 'Wissen' des neuronalen Netzes ist in seinen Gewichten gespeichert. Diese werden durch ein Training evaluiert. Das heisst, bevor das Netz eine gewünschte Funktion, wie bei uns die Phonemerkennung, implementieren kann, muss es trainiert werden.

Der Trainingsprozess läuft wie folgt ab. Dem Netz werden an den Eingangsneuronen verschiedene Lernbeispiele präsentiert. Gleichzeitig wird die Anwort des Netzwerks auf den Input mit dem gewünschten Resultat verglichen. Anhand der Abweichung der Aktivierungen der Ausgangsneuronen von der Vorgabe wird eine Anpassung der Gewichte vorgenommen. Wie diese Gewichtsänderung von statten geht, hängt vom Trainingsalgorithmus ab. Wir verwendeten den Backpropagation-Algorithmus, welcher uns das NICO Toolkit zur Verfügung stellt. Das Set aller Beispielsdaten, die für das Training eingesetzt werden, bildet das so genannte *Trainingsset*. Ein Durchgang, in welchem dem Netz alle Beispiele des Trainingssets einmal präsentiert werden, wird *Epoche* genannt. Die exakte Funktionsweise ist in [1] definiert.

2.1.1 Overfitting

Je länger ein Netz mit den Trainingsdaten gelehrt wird, desto besser wird seine Erkennungsleistung. Ab einem gewissen Punkt lernt das Netz die Trainingsdaten jedoch



Figur 2: *Datenfluss durch das für Phonemerkennung verwendete neuronale Netz.*

auswendig. Dies bedeutet, dass die Erkennungsleistung des Netzes wieder anfängt zu sinken, wenn es auf ein unabhängiges Set von Testdaten angewendet wird. Das Netz hat in diesem Fall verlernt, zu verallgemeinern und ist zu stark auf die Trainingsdaten fixiert.

Aus diesem Grund wird während des Trainings in Intervallen die Erkennungsleistung des Netzes auf einem Testset kontrolliert. Nimmt diese eine gewisse Zeit lang nicht mehr zu, wird das Training abgebrochen. Diese Idee haben wir bei allen unseren Tests angewandt. In Abschnitt 8.3 wird erklärt, wie das Overfitting überwacht wird.

2.2 Phonemerkennung mit TDNN

In der Sprachverarbeitung können neuronale Netze als Phonemerkenner verwendet werden, wie bereits in der Einleitung 1.1.1 erläutert wurde. Wir haben das frei erhältliche NICO Softwarepaket [6] benutzt, um neuronale Netze zu erstellen, zu trainieren und anzuwenden. NICO wurde speziell für neuronale Netze entwickelt, die für die Sprachverarbeitung geeignet sind.

Die Figur 2 veranschaulicht wie ein neuronales Netz als Phonemerkenner arbeitet. Mit einem NICO Tool wird ein Audiofile in Frames von 10ms Dauer unterteilt. Aus diesen Frames extrahiert das Tool die so genannten Features, welche pro Frame aus 13 Koeffizienten bestehen [6]. Diese werden auf die Neuronen des Input Layers abgebildet

und produzieren eine entsprechende Aktivierung am Ausgang des Netzes. Jedem Neuron im Output Layer wird ein Phonem zugeordnet. Die Werte dieser Neuronen haben die Bedeutung der Wahrscheinlichkeit, dass das jeweilige Phonem im angelegten Frame vorkommt.

2.3 Freiheitsgrade

In den nächsten Abschnitten werden einige Freiheitsgrade der Erstellung und des Trainings eines neuronalen Netzes aufgezeigt. Um ein geeignetes Netz zu finden, muss ein mehrdimensionaler Raum von Parametern abgesucht werden. Dass heisst, es wird eine Vielzahl von Tests benötigt, welche ausgewählte Freiheitsgrade auf deren optimalen Bereich hin untersuchen. Konkret werden wir also Netze trainieren, wobei wir Topologie- oder Trainingsparameter einzeln verändern. Die Resultate dieser Tests geben uns Aufschluss darüber, welches die optimale Konfiguration sein könnte (Abschnitt 6). Da es in der uns zur Verfügung stehenden Zeit nicht möglich ist, alle möglichen Freiheitsgrade zu testen, beschränken wir uns auf eine Auswahl von Versuchen. Diese Auswahl trafen wir gestützt auf Ergebnisse von verschiedenen Quellen, die im einzelnen angegeben werden.

2.4 Topologie

Die Topologie, der Aufbau des Netzes, muss beim Erstellen des Netzes definiert werden. Diese Parameter können den verschiedenen NICO Tools übergeben werden, welche für das Kreieren eines neuen Netzes vorhanden sind.

2.4.1 Anzahl Neuronen im Hidden Layer

Je mehr Neuronen im Hidden Layer sind, desto mehr Verbindungen ergeben sich. Das heisst, da das Wissen des neuronalen Netzes in den Gewichten dieser Verbindungen gespeichert ist, steigt mit zunehmender Neuronenzahl die Kapazität des Netzes. Oder anders ausgedrückt: Je mehr Neuronen ein neuronales Netz hat, desto komplexere Entscheidungsregionen kann es bilden [1]. Werden jedoch zu viele Neuronen verwendet, kann es passieren, dass das Netz die Trainingsdaten sozusagen auswendig lernt. Bei anderen, neuen Daten, erreicht es dann eine tiefere Erkennungsrate als ein Netz mit weniger Neuronen. Auf der anderen Seite ist die Sprache sehr komplex, und es hat sich gezeigt [3], dass die Anzahl der Neuronen nur durch die Trainingszeit limitiert ist, die man gewillt ist, zu investieren.

2.4.2 Anzahl Hidden Layers

Gemäss [1] kann ein Netz ohne Hidden Layer nur lineare Entscheidungsregionen bilden. Hat das Netz zwei Hidden Layer (mit genügend Neuronen), kann es beliebige Regionen formen. In [3] wird jedoch ein Beweis erwähnt, dass auch ein Netz mit nur einem Hidden Layer jede beliebige Funktion bilden kann, wenn genügend Neuronen vorhanden sind. In

diesem Beweis wird jedoch keine Aussage darüber gemacht, ob mehrere Hidden Layer eine Auswirkung auf die Trainingszeit haben oder beispielsweise auf die Wahrscheinlichkeit, in einem lokalen Minimum stecken zu bleiben. Deshalb haben wir entgegen dem Vorschlag von [3] auch Netze mit zwei Hidden Layer ausprobiert.

2.4.3 Verbindungen

In dem in Figur 2 gezeigten einfachen Netz sind nur aufeinanderfolgende Schichten miteinander verbunden. Es könnten jedoch auch Verbindungen direkt zwischen dem Input Layer und dem Output Layer gezogen werden. Weiter wären auch rückwärtsgerichtete Verbindungen möglich, oder Verbindungen innerhalb desselben Layers. [3] kommt jedoch zum Schluss, dass solche spezielle Klassifizierungs-Netze keinen entscheidenden Vorteil ergeben. Deshalb haben wir auf eigene Tests verzichtet.

2.4.4 Anzahl Kontextframes

Die dynamischen Eigenschaften eines TDNN erlauben dem Netz, zeitlich aufeinanderfolgende Frames gleichzeitig zu betrachten. Man kann sich vorstellen, dass die Erkennung sich verbessert, wenn dem Netz neben dem aktuellen Frame auch noch ein paar Frames links und rechts davon präsentiert werden. Mit links und rechts ist zeitlich vor- und nachher gemeint. Denn die Aussprache der Laute ist grundsätzlich kein statischer Prozess. Wird dem Netz zu wenig oder gar kein Kontext gegeben, könnte zu wenig Information aus der Umgebung des aktuellen Frames in den Erkennungsprozess einfliessen, um eine akkurate Antwort zu geben. Wird jedoch zu viel Kontext verwendet, könnte zu viel Information von benachbarten Phonemen hinzukommen, was für die Klassifizierung des aktuellen Frames nicht mehr nützlich ist. Desweitern entstehen mit mehr Kontext auch mehr Verbindungen, und damit erhöht sich die Trainingsdauer.

2.5 Wahl der Trainingsdaten

Sprachdaten Für das Training des Netzes stehen uns ungefähr 900 von unserem Tutoren gesprochenen Sätze zur Verfügung. Dies sind im Ganzen etwa 50'000 Phoneme.

Phonem Mapping Zunächst wurde ein Satz von 41 Phonemen verwendet. Mit der Überlegung, dass das neuronale Netz besser wird, wenn es gewisse Phoneme nicht unterscheiden muss, wurde ein kleinerer Satz von 30 Phonemen eingeführt. Unter anderem wurden verschiedene Varianten von i,o und e auf genau eine Variante i,o bzw. e abgebildet (mapped). Die Unterschiede der ursprünglichen Phoneme sind zum Teil tatsächlich sehr gering. Dazu kommt, dass der Sprecher ein Wort eventuell nicht ganz genau so ausspricht, wie das Wörterbuch es vorgibt.

Das Wörterbuch, mit dem der Wordspotter arbeitet, wird jedoch in jedem Fall noch mit den orginalen Phonemen geführt.

2.5.1 Grösse des Trainingsset

Darunter versteht man die Anzahl verschiedener Lernbeispiele, die dem Netz präsentiert werden. Wir vermuten, dass wenn mehr Lernbeispiele gezeigt werden, das Netz bessere Ergebnisse liefert. Diese Vermutung wird in [3] bestätigt. Jedoch steigt damit die Lerndauer.

2.5.2 Gleichverteilt

Jedes Phonem wird dem Netz gleich oft vorgestellt. Werden dem Netz ganze Sätze präsentiert, dann werden Phoneme, die in der deutschen Sprache selten vorkommen, weniger oft gelehrt als jene, die oft vorkommen. Das Netz wird demzufolge Mühe haben, seltene Phoneme richtig zu erkennen. Es zeigt sich jedoch [3], dass der Wordspotter dann am Besten arbeiten kann, wenn alle Phoneme in etwa gleich gut erkannt werden.

Folgendes Beispiel kann zur Unterstützung dieser Annahme dienen. Angenommen der zu erkennende Wortschatz besteht aus nur drei Wörtern 'kuss' 'fuss' und 'buss'. Werden nun alle Wörter als ganzes trainiert, so wird die häufig vorkommende Lautfolge 'uss' sehr gut erkannt. Umso schlechter erkannt werden aber die selteneren Phoneme 'k', 'f' und 'b'. Doch steckt in diesen drei Phonemen die gesamte Entscheidungsinformation, welche benötigt wird um das korrekte Wort zu erkennen. Denn es ist nicht sehr hilfreich für den Wordspotter, wenn er von einem zu erkennenden Wort weiss, dass 'uss' ziemlich sicher darin vorkommt aber der Rest nicht erkannt wurde.

2.5.3 Pädagogisch

Mit dem Ziel vor Augen, dass alle Phoneme gleich gut erkannt werden sollten, haben wir das pädagogische Training angewandt. Das pädagogische Lernen funktioniert nach dem Karteikarten-Prinzip, bei welchem schlecht Gelerntes häufiger vorkommt. Die Erkennungsleistung eines Netzes wird in Intervallen überprüft. Nach der Überprüfung wird das Trainingsset neu zusammengestellt. Je schlechter ein Phonem vom Netz erkannt wurde, desto häufiger kommt es im neuen Trainingsset vor und umgekehrt.

2.5.4 New Alignment

Die Labelfiles, die dem Netz während des Trainings die richtigen Phoneme vorgeben, wurden von HMM's (Hidden Markov Models) erstellt. Leider stimmen die Zeitpunkte, an denen die HMM's entschieden haben, dass ein Phonem aufhört und das Nächste beginnt, nicht immer mit der Wirklichkeit überein. Natürlich ist bei einem Übergang tatsächlich nicht eindeutig bestimmbar, wann genau das eine Phonem aufhört und wann das Nächste beginnt. Es hat sich jedoch durch Anhören von Beispielen gezeigt, dass die von den HMM's erstellte Unterteilung (Alignment), über dieses Unsicherheitsmass hinaus ungenau ist. Das führt dazu, dass die Netze teilweise falsch gelehrt werden.

Aus diesem Grund entstand die Idee, die Labelfiles vom Netz und vom Wordspotter nachbessern zu lassen. Der Output eines stückweit trainierten Netzes wird dem

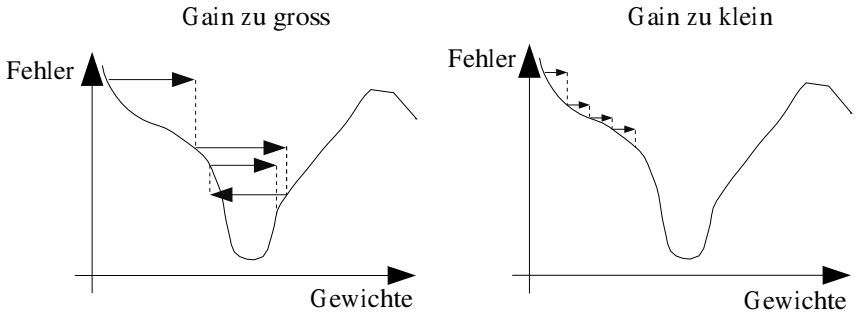
Wordspotter übergeben. Dieser versucht anstatt die vorkommenden Wörter zu finden (Abschnitt 3), nur die Phonemübergänge zu detektieren. Der Weg, den der Wordspotter durch die Wahrscheinlichkeitsmatrix findet, definiert die neuen Phonemübergänge, das neue Alignment. Damit dies richtig funktionert, muss natürlich das Lautdauermodell des Wordspotters ausgeschaltet werden. Nun wird das Netz mit den neuen Labelfiles weiter trainiert und der ganze Prozess kann wieder von vorne beginnen.

2.6 Parameter des Backpropagation-Algorithmus

Die Parameter des Trainingalgorithmus, können dem NICO Tool *Backprop* direkt übergeben werden.

2.6.1 Gain

Der wichtigste Parameter des Backpropagation-Algorithmus ist der Gain; anschaulich übersetzt mit Lernrate. Der Gain ist der Faktor der Gewichtsänderung. Figur 3 veranschaulicht die Wirkungsweise des Gains. Er ist proportional zur Schrittweite, mit welcher der Lernprozess in der Fehlerlandschaft voranschreitet. Ist der Gain zu klein, lernt das Netz nur sehr langsam. Ist er jedoch zu gross, dann osziliert der Algorithmus auf einer höheren Ebene, wenn er in einem engen Tal angelangt ist. Folglich wäre es am



Figur 3: *Auswirkung des Gain auf die Lerngeschwindigkeit.*

Besten, am Anfang des Trainings den Gain eher gross zu halten, und dann immer kleiner werden zu lassen. Der Backpropagation-Algorithmus von NICOS unterstützt eigentlich eine solche Funktion, aber sie scheint nicht zu funktionieren. Aus Zeitmangel haben wir in unsere Trainingskripts keine eigene solche Gewichtszerfalls-Funktion eingeführt.

2.6.2 Momentum Term

Der Momentum Term, auch Überrelaxation genannt, kann als Trägheitsfaktor betrachtet werden. Genauer gesagt bestimmt er, in welchem Masse die vorangehende Gewichtsänderung in die aktuelle Gewichtsänderung einfliessen soll. Bildlich betrachtet

gleicht die Bewegung des Lernvorganges auf der Fehlerlandschaft einer Kugel. Das heisst, bei einer negativen Steigung beschleunigt sich die Bewegung (Der Faktor der Gewichtsänderung nimmt zu) und beim Ankommen in der Talebene schwingt das Netz über das Minimum hinaus ein Stück weit die positive Steigung hinauf. Je nachdem wie gross der Momentum Term gewählt wird, verstärkt sich dieses Verhalten. Wird der Momentum Term auf null gesetzt, wiederfährt der Gewichtsveränderung gar keine Beschleunigung und es ist keine "Trägheit" mehr auszumachen. Die Erhöhung des Momentum Term hat zur Folge, dass lokale Minimas übersprungen werden können und der Lernprozess beschleunigt wird.

2.6.3 Update Frequenz

Mit der Update Frequenz wird das Gewichtsänderungs-Intervall bezeichnet. Sie gibt an, nach wievielen Frame-Präsentationen die Gewichte des Netzes angepasst werden. Der Defaultwert des NICO Tools *Backprop* ist nach der Grösse des Kontextfensters plus eins definienert. Wird also ein Fenster von insgesamt neun Frames gewählt, werden die Gewichte nach zehn Frames "upgedatet".

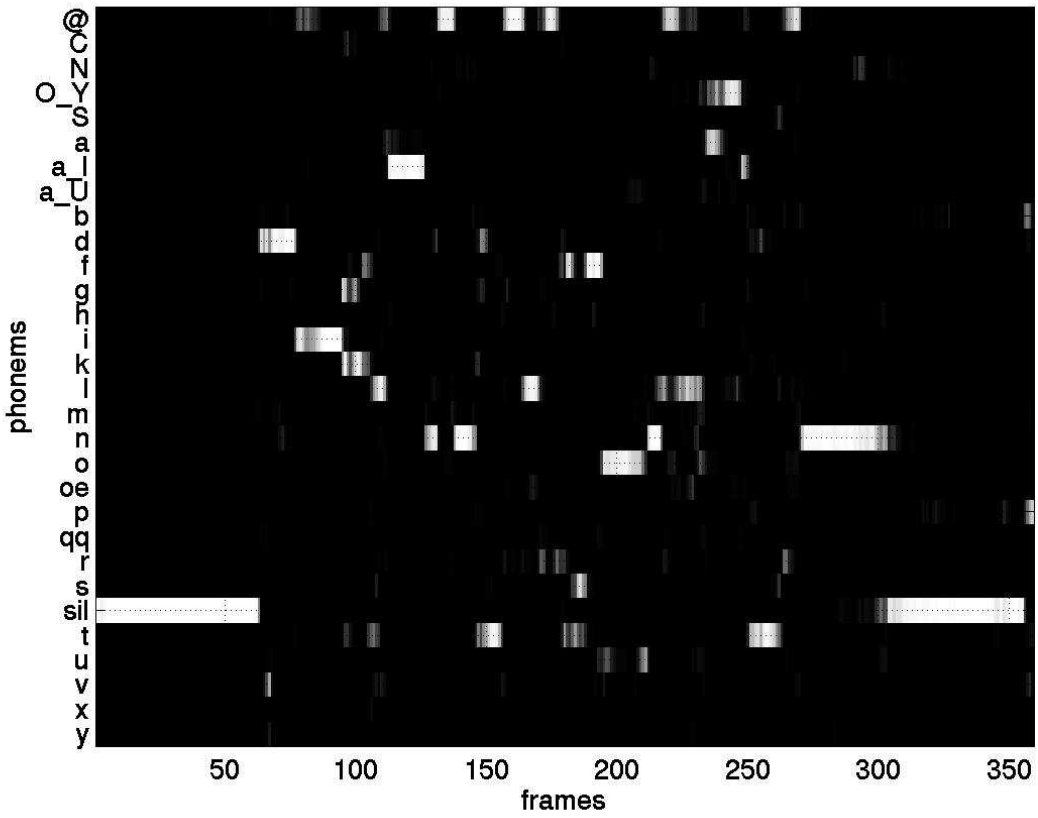
2.6.4 Aktivierungsfunktion

Bei der Wahl der Aktivierungsfunktion haben wir uns aus Zeitgründen auf die Standartfunktion des NICO Pakets, *tanh(x)*, beschränkt. NICO würde auch nur die Sigmoid-Funktion als Alternative anbieten. [3] schlägt jedoch entschieden vor, dass die Aktivierungsfunktion symetrisch sein sollte, was die Sigmoid-Funktion nicht ist.

3 Wordspotter

3.1 Aufgabe

Der Wordspotter liegt auf der dritten Stufe der Spracherkennung (Figur 1). Sein Input ist eine zwei-dimensionale Matrix: die Auftritts-Wahrscheinlichkeiten der Phoneme, für jedes Frame eines Satzes. Daraus soll er bestimmen, welche Wörter aus einem gegebenen Wörterbuch am Wahrscheinlichsten im betreffendem Satz vorkommen.



Figur 4: *'Die kleinen Telefone läuten': Beispiel eines Outputs des neuronalen Netzes bzw. Inputs des Wordspotters. Je heller die Schattierung, desto höher die Wahrscheinlichkeit des jeweiliges Phonems im jeweiligen Frame.*

Figur 4 zeigt ein Beispiel eines typischen Inputs. Der gesprochene Satz war 'die kleinen Telefone läuten'. Je heller die Schattierung, desto höher die Wahrscheinlichkeit, dass das entsprechende Phonem im entsprechenden Frame ausgesprochen wurde. Der Satz fängt mit dem Pseudo-Phonem 'sil' (Silence) an, dann folgen 'd','i' und so weiter.

Für unsere Arbeit haben wir den Wordspotter jedoch ausschliesslich dazu gebraucht, das neuronale Netz, welches den Input des Wordspotters liefert, zu qualifizieren. Dies wird fortan Qualifying genannt. Denn je besser der Wordspotter Wörter finden kann, desto besser war sein Input.

3.2 Konzepte und Algorithmen

In diesem Abschnitt werden verwendete Algorithmen kurz erklärt. Bis auf den Backtrace Algorithmus, den wir selbst entwickelt haben, sind dies alles bekannte Algorithmen; tiefergehende Erläuterungen sind in der Fachliteratur zu finden.

3.2.1 Wordspotter

Da die Aufgabe des Wordspotters bereits vorgestellt wurde, ist hier gleich der Algorithmus angegeben:

- erstelle eine leere Liste
- für jedes Wort im Wörterbuch
  - bestimme den Score des Wortes
  - füge Wort nach dem Score geordnet in die Liste ein
- die Liste bildet den Output. Die im Satz vorkommenden Wörter sollten in den besten Positionen zu finden sein.

Abschnitt 3.2.2 erläutert, wie denn der Score eines Wortes bestimmt wird. Im Wörterbuch sind *alle* bekannten Wörter zusammen mit ihrer phonetischen Übersetzung aufgelistet.

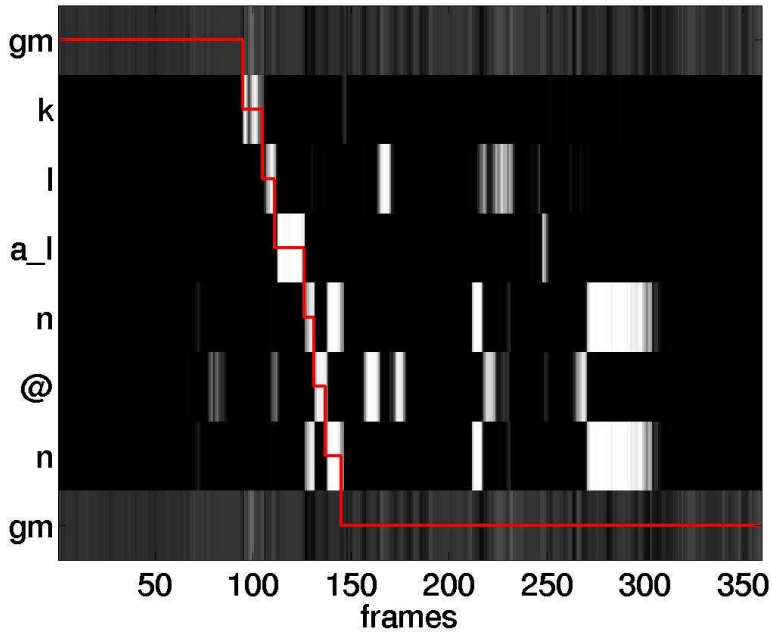
Dieser vorgestellte Spot-Algorithmus hat jedoch einen Nachteil. Kommt ein Wort in einem Satz mehrmals vor, wird nur eines dieser Wörter gefunden. Zusätzlich kommt es häufig vor, dass kleine Wörter in grösseren enthalten sind. Dies kann dazu führen, dass das enthaltene Wort gefunden wird obwohl es als Einzelwort auch noch vorkommt. Zum Beispiel könnte das Wort "*ein*" im Satz "Ein kleiner Mann" in "kl*ein*er" gefunden werden. Dies bedeutet, dass das zu suchende Wort an einer falschen zeitlichen Position gefunden wurde. Aus Zeitgründen wurde kein effizienterer Spot-Algorithmus implementiert.

3.2.2 Tokenpasser

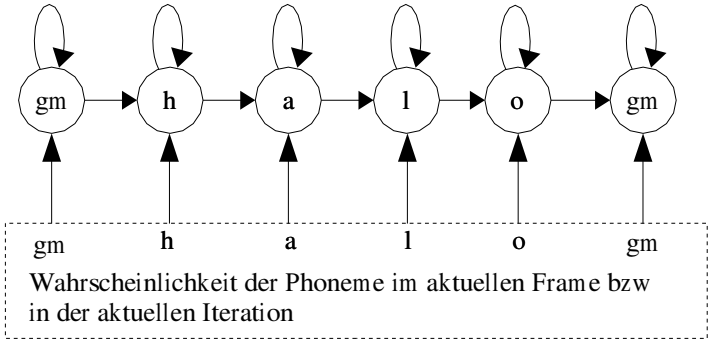
Die Aufgabe des Tokenpasser ist es, einem gegebenem Wort einen Score zuzuordnen. Nach diesem Score wird das Wort vom Wordspotter in der Output Liste eingeordnet. Der Score entspricht der Wahrscheinlichkeit, dass das Wort im Satz vorkommt.

Aus einer anderen Sichtweise gesehen liegt die Aufgabe des Tokenpassers darin, die Stelle im Input des Wordspotters zu finden, wo ein gegebenes Wort hineinpasst. Im Beispiel von Figur 4 soll er herausfinden, dass das Wort "kleinen" in den Frames 94 bis 146 vorkommt.

Veranschaulicht wird diese zweite Auffassung durch Figur 5. Gegeben ist wiederum die Wahrscheinlichkeit für jedes Phonem, dass es in einem bestimmten Frame vorkommt. Jedoch werden die Phoneme auf diejenigen beschränkt, die im zu suchenden Wort vorkommen. Zusätzlich sind die Phoneme auf der Ordinate in der Reihenfolge geordnet,



Figur 5: *Beispiel für 'kleinen': Tokenpasser sucht einen Weg, so dass das Produkt der Wahrscheinlichkeiten entlang dieses Weges maximal wird*



Figur 6: *Der Tokenpasser benützt ein Petri Netz*

wie sie im Wort vorkommen. Nun muss ein Weg von links oben nach rechts unten gefunden werden, sodass das Produkt der Wahrscheinlichkeiten entlang dieses Weges möglichst gross ist. Von links oben nach rechts unten deshalb, weil dann zeitlich gesehen die Phoneme in der richtigen Reihenfolge ausgesprochen werden und somit das gesuchte Wort ausgesprochen wurde. Dieses Produkt entspricht nun der Score des Wortes, womit die zwei Sichtweisen in der Aufgabenstellung des Tokenpassers miteinander verbunden sind. *gm* steht für Garbage Modell. Darauf wird in Abschnitt 3.2.4 näher eingegangen.

Die Aufgabe wird mit einem Petri Netz gelöst (Figur 6). Jeder State des Netzes entspricht einem Phonem des gesuchten Wortes. Ohne weitere Erklärungen sei hier der Algorithmus angegeben, der den besten Weg findet. Für eine tiefer gehende

Betrachtungen des Tokenpasser sei der Leser auf [5] verwiesen.

- Lege ein Token mit dem Wert 0 in jeden State; ausser dem ersten State, welcher ein Token mit Wert 1 erhält.
- Für alle Frames des Satzes:
  - Multipliziere das eine Token in jedem State mit der Wahrscheinlichkeit des zum State gehörenden Phonems
  - Für jeden State
    - Lege eine Kopie des Tokens in den nächsten State
  - Für jeden State
    - Lösche das kleinere der beiden Token
- Das Token im letzten State entspricht nun der Score des gesamten Wortes

Der Score des Wortes ist nun der Wert des Tokens im letzten State. Da die Wahrscheinlichkeiten in jedem Frame immer miteinander multipliziert werden, hat der Score die Bedeutung der Wahrscheinlichkeit, dass das gesamte Wort im Satz vorkommt.

3.2.3 Backtrace

Der Tokenpasser hat nun wie gewünscht den Score eines Wortes bestimmt. Aber der vorgestellte Algorithmus hat kein Gedächtnis; er weiss am Ende nicht mehr, zu welchem Zeitpunkt ein Token den State gewechselt hat. Sein Wissen beschränkt sich auf den Inhalt der States; gültig für genau ein Frame. Ein Gedächtnis wird jedoch dazu benötigt, um im Nachhinein zu wissen, wann welches Phonem ausgesprochen worden ist. Denn das Wechseln eines Tokens vom State k in den State l im Frame x, ist gleichbedeutend mit "im Frame x war das Phonem k zu Ende gesprochen und Phonem l hat begonnen".

Jeder State des Tokenpassers bekommt einen eigenen Stack zugewiesen. Ein Stack-Element ist ein Tupel aus einer Framenummer und einer Stackposition. Die Framenummer gibt das Frame an, in welchem der Tokenpasser in den dem Stack zugeordneten State gewechselt hat; bzw. wann er ein Token aus dem vorangehenden State ausgewählt hat. Die Position kann als Zeiger auf ein Stackelement im vorangehenden Stack aufgefasst werden. Der folgende Algorithmus funktioniert nicht autonom; er wird in den Tokenpasser-Algorithmus eingefügt.

- Leere alle Stacks
- Dieser Punkt wird am Ender der "Für alle Frames des Satzes" Schlaufe des Tokenpasser-Algorithmus eingefügt.

  Für alle States

  - Falls das Token des vorangehenden States ausgewählt wurde:
    Stelle ein neues Element auf den Stack des aktuellen States. Der Frameteil des Tupels wird dem aktuellen Frame gleichgesetzt. Der Positionsteil wird mit der Position desjenigen Elementes gleichgesetzt, das zuoberst im vorangehenden Stack liegt.

- Beginnend beim Top-Element des letzten Stacks kann mit Hilfe des Positionsteils jedes Stackelements ein Weg durch alle Stacks hindurch gefunden werden. Der Framenummerteil jedes Elementes auf diesem Weg gibt an, wann das dem Stack zugeordnete Phonem angefangen hat.

3.2.4 Garbage Modell

Damit der Tokenpasser Wörter, die mitten im Satz vorkommen, überhaupt finden kann, braucht es zwei zusätzliche States. Dies sind die Garbage States, die in den Figuren 5 und 6 ersichtlich sind. Ohne diese würde der Tokenpasser gleich im ersten Frame mit dem ersten Phonem beginnen und müsste mit dem letzten Phonem im letzten Frame aufhören.

Diese Garbage States sind aus der Sicht des Tokenpassers genau gleich wie alle anderen States auch. Statt einer Phonem-Wahrscheinlichkeit wird ihnen jedoch ein Garbage-Score zugewiesen. Zur deren Berechnung wird das Online Garbage Modell [4] verwendet: Für jedes Frame wird der Mittelwert aus den n grössten Phonem-Wahrscheinlichkeiten dieses Frames berechnet.

[4] schlägt vor, dass n=20 ein guter Erfahrungswert sei. Wir hatten hingegen mit n=4 die besten Erfolge.

3.2.5 Lautdauer-Modell

Um die Erkennungsleistung des Wordspotters zu erhöhen, haben wir ein Lautdauer-Modell eingeführt. Gegeben ist die durchschnittliche Dauer und die Varianz dieser Dauer jedes Phonems. Es wird angenommen, dass die Dauer gammaverteilt ist.

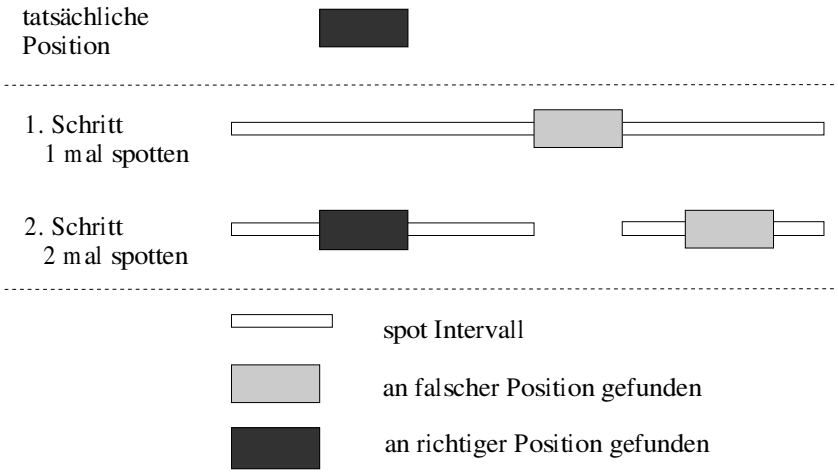
Angenommen, der Tokenpasser ist seit einer bekannten Anzahl Frames im State a, dann kann er mit der gegebenen Gammaverteilung berechnen, wie wahrscheinlich es ist, im State a zu bleiben oder den State zu verlassen. Diese Wahrscheinlichkeit wird mit dem Token multipliziert, das im State bleibt, bzw. mit dem kopierten Token, das in den nächsten State weitergegeben wird. Der Leser sei zum besseren Verständnis nochmals auf den in Abschnitt 3.2.2 angegebenen Algorithmus verwiesen.

Der Gewinn liegt darin, dass Wörter, die gar nicht im Satz vorkommen, eine noch tieferen Score bekommen als ohne Lautdauer-Modell. Denn diese Wörter haben einige States, in denen der Tokenpasser nur ganz kurz verweilt. Es sind dies diejenigen States, die Phonemen entsprechen, die eben nicht im Satz vorkommen. Es resultiert eine "Strafe", da das Token mit der tiefen Wahrscheinlichkeit einer solch kurzen Aufenthaltsdauer multipliziert wird.

3.2.6 Qualifying

Wie bereits erwähnt, liegt die eigentlich Aufgabe des Wordspotters für unsere Zwecke in der Qualifikation von sich selbst, bzw. des neuronalen Netzes, das ihm den Input liefert. Denn je besser der Wordspotter ist, desto besser war das neuronale Netz. Mit welchen Fehlermassen dies gemessen wird, ist Gegenstand von Abschnitt 5.

Damit dieses Kapitel verstanden werden kann, wird ein Fehlermass kurz eingeführt. Die *Worst Wordposition* sagt aus, an welcher Position in der Word Score List das im gesprochenen Satz vorkommende Wort mit der tiefsten Score liegt. Dazu ein Beispiel. Angenommen der zu verarbeitende Satz lautet "Ich heisse Ernst" und "heisse" liegt an Position 1, "Ernst" an Position 3 und "ich" an Position 6. Dann ist die *Worst Wordposition* 6. Dabei sei die Liste so geordnet, dass das Wort mit dem höchsten Score an Position 1 liegt.



Figur 7: *Für das Qualifying wird rekursiv so lange gespotted, bis ein Wort an der richtigen Position gefunden wurde.*

In Abschnitt 3.2.1 wurde erklärt, dass der Spot Algorithmus manchmal Wörter an einer falschen zeitlichen Position findet. Es sei nochmals darauf hingewiesen, dass immer alle Wörter 'gefunden' werden, d.h. dass der Tokenpasser immer einen Weg findet, aber eben manchmal an der falschen zeitlichen Position. Es ist jedoch für die Qualifikation unbedingt erforderlich, dass alle im Satz vorkommenden Wörter an der richtigen Position gefunden werden. An welcher Position ein Wort denn zu stehen hat, wird vorgegeben durch das Labelfile (Abschnitt 8.3). Eine geringe zeitliche Verschiebung wird noch toleriert. Originalwort und gefundenes Wort werden als an der gleichen Position stehend betrachtet, wenn der Mittelpunkt des einen innerhalb des gesamten Bereichs des anderen liegt oder umgekehrt.

Dies führt dazu, dass wenn ein Wort nicht gefunden wurde, es nochmals gespottet werden muss. Nur dieses Mal in den zwei Intervallen, in denen es bisher nicht gefunden wurde. Figur 7 veranschaulicht diesen Prozess. Würde das Wort auch im zweiten Schritt nicht an der richtigen Position gefunden werden, müsste es in einem dritten Schritt in

vier Intervallen gesucht werden usw. Nachdem alle Wörter an der korrekten zeitlichen Position gefunden worden sind, können die Fehlermasse berechnet werden.

3.3 Implementation

In diesem Abschnitt soll dem Leser ein Überblick über den Java Sourcecode vermittelt werden. Die verwendeten Klassen und der Ablauf der wichtigsten Prozesse wie das eigentliche Wordspotten werden grob vorgestellt. Für eine tiefer gehende Beschreibung des Sourcecodes sei der Leser auf die Javadoc Kommentare im Sourcecode verwiesen. Eine Beschreibung der vom Wordspotter verwendeten Files ist im Anhang A zu finden.

Alle in Variablen gespeicherten Wahrscheinlichkeiten und alle Scores sind jeweils logarithmisiert. Dies deshalb, weil in dem Tokenpasser-Algorithmus (Abschnitt 3.2.2) viele Multiplikationen vorkommen. Sind die Multiplikanten jedoch im logarithmischen Mass, kann man sie einfach addieren, welches eine schnellere Operation ist. Alle Bezeichner, die logarithmisierte Werte bezeichnen, haben die Endung `log`.

3.3.1 Begriffe und Hilfsklassen

In der folgenden Liste werden einige häufig gebrauchte Begriffe eingeführt.

Phord Steht für 'Phonem Word', also ein Wort geschrieben mit Phonemen.

PhordMap Analog zu Phord, nur werden hier gemappte Phoneme verwendet, um das Wort zu bilden.

Wpp Abkürzung für die Klasse WordPhordPair

Vorweggenommen ist hier eine Liste von Klassen, die in den weiteren Abschnitten referenziert werden. Es ist jedoch keine vollständige Liste aller Klassen.

OriginalWord Beschreibt wann, gemäss dem Labelfile, ein Wort tatsächlich ausgesprochen wurde.

OriginalWordDictionary Sammlung von `OriginalWords`; liest die zu Grunde liegenden Daten aus einem Labelfile. Implementiert das Interface `Iterator`, über dessen Methoden der Zugriff auf die Daten erfolgt.

PhonemCoder Übersetzt Phoneme gegeben als Strings in Integers und umgekehrt.

PhonemScore Die Wahrscheinlichkeit jedes Phonems in einem Frame.

PhonemMapping Übersetzt ein Phonem in sein gemapptes Phonem.

Preferences Sammlung von Parametern, die das Programmverhalten beeinflussen. Ein Teil dieser Werte kann beim Programmstart aus einem Preferences File (Anhang B: Konfigurationsfiles) eingelesen werden.

Tools Sammlung von Funktionen, die den Wordspotter steuern oder dessen Output, die Wordscore Liste, auf verschiedene Weisen verarbeiten können.

WordPhordPair Fasst ein Wort, sein zugehöriges Phord und das PhordMap zusammen.

WordScore Fasst ein `WordPhordPair` und alle Informationen, die der Wordspotter darüber herausfindet, zusammen. Der Name rührt von der wichtigsten Information her, der Score des Wortes.

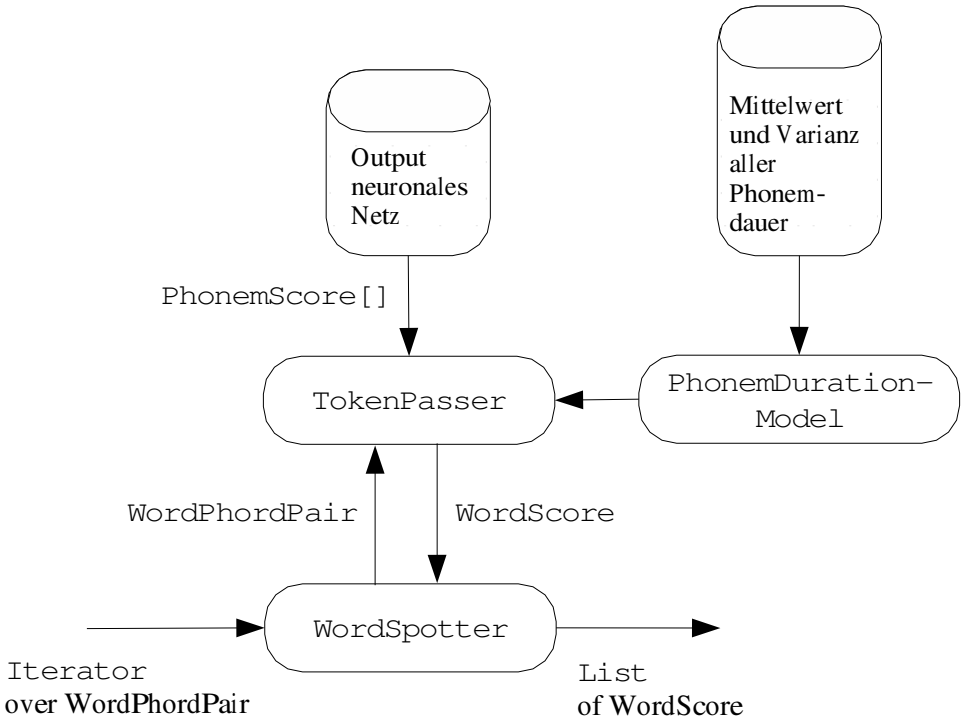
WppDictionary Sammlung von `WordPhordPairs`; liest die zu Grunde liegenden Daten aus dem Wörterbuch. Implementiert das Interface `Iterator`, über dessen Methoden der Zugriff auf die Daten erfolgt.

3.3.2 Spot

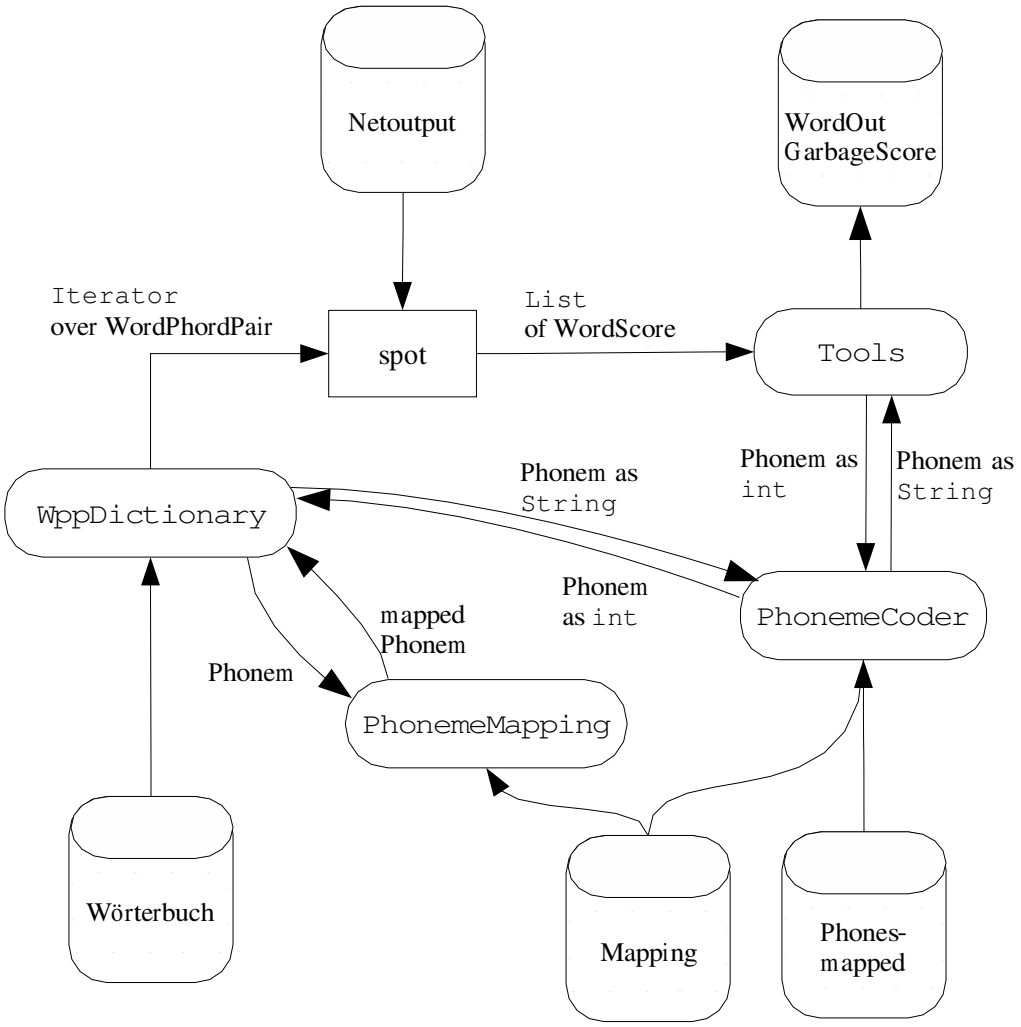
Figur 8 veranschaulicht die Umsetzung des Wordspottings-Algorithmus, wie er in Abschnitt 3.2.1 beschrieben wurde. Dem Wordspotter werden alle Wörter gegeben, die er spotten muss. Diese Wörter werden durch einen `Iterator` geliefert. Für jedes Wort lässt der Wordspotter den Tokenpasser den Score dieses Wortes bestimmen. Gleich danach wird das Wort in die Word Score Liste eingeordnet. Der Tokenpasser liest bei seiner Initialisierung die `PhonemScores` selber ein. Das `PhonemDurationModel` liefert dem Tokenpasser die Gammaverteilung jedes Phonems, die dann für das Lautdauer Modell verwendet wird. Das Garbage Modell und der Backtrace Algorithmus sind als Methoden in der Tokenpasser Klasse implementiert.

Figur 9 zeigt das Wordspotting von einer höheren Ebene aus betrachtet. Gesteuert werden die im Folgenden aufgeführten Prozesse durch die SpotWords Klasse, die auch die `main` Methode enthält. Es geht nun darum, tatsächlich Wörter aus einem gegebenem Wörterbuch zu spotten und das Ergebnis in das Wordout-File zu schreiben. Die Klasse WppDictionary liest das Wörterbuch ein. Mit Hilfe von PhonemeMapping und PhonemCoder werden WordPhordPair Objekte erstellt. Diese werden dem Wordspotter via einen Iterator übergeben. Der Output, die Liste von Wordscores, wird von der Klasse Tools in ein Textfile geschrieben. Dieses Textfile kann dann wiederum von Matlab Methoden eingelesen und visualisiert werden. Dies ist praktisch um zu kontrollieren, ob der Wordspotter so funktioniert, wie er sollte.

Es ist weiterhin ersichtlich, dass Phoneme in ihrer String Repräsentation ausschliesslich in In- und Output-Methoden vorkommen. Im weitaus grössten Teil des Programms sind sie als Integers repräsentiert. Ähnliches gilt für gemappte Phords, die PhordMaps. Im grössten Teil des Codes erscheint ein Phord immer zusammen mit seinem PhordMap, gebündelt durch die Klasse WordPhordPair. Ausser beim Einlesen muss nirgends ein Phonem in ein gemapptes Phonem umgerechnet werden.



Figur 8: *Implementation des eigentlichen Spot-Prozesses*



Figur 9: *Implementation des Wordspotters*

3.3.3 Qualifying und New Alignement

New Alignment und Qualifying sind als Methoden in der Klasse Tools implementiert. Der Nutzen und die Arbeitsweise dieser Konzepte wird in Abschnitt 6.2.4 (New Alignment) und Abschnitt 3.2.6 (Qualifying) erläutert.

4 Durchführung des Training- und Test-Prozesses

In diesem Abschnitt wird anhand von Beispielen erklärt, wie unsere Skripts und Java Programme anzuwenden sind. Im Anhang A befindet sich eine nützliche Liste aller benötigten Datenfiles. Im Anhang B kann die Bedienung der einzelnen Skripts, Konfigurationsfiles und Programme nachgeschlagen werden.

4.1 Erzeugen der Featurefiles und Targetfiles

Für das Erkennen sowie für das Training müssen zuerst aus den rohen Audiodaten die Featurevektoren gewonnen werden. Das vom NICO Toolkit gelieferte Tool Barkfib erzeugt aus Audiofiles die Featurefiles. Mit diesen Featurefiles und zusammen mit den dazugehörigen Labelfiles kann das Tool Lab2Targ die Targetfiles produzieren. Diese dienen dem Netz während der Lernphase als Zielvorgabe.

4.2 Erstellen einer neuen Testumgebung

Eine neue Testumgebung kann in nur einem Schritt neu erstellt werden.

1. Erstellt eine Testumgebung mit dem Namen test001.
   createtest.sh test001

Alle Files, welche spezifisch für einen Test sind, sind jeweils in einem Testverzeichnis zusammengefasst, in diesem Beispiel test001/. Vorgefertigte Konfigurationsfiles, Skripts usw., welche sich von Test zu Test gar nicht oder nur leicht ändern, befinden sich in testbasis/. createtest.sh erstellt ein neues Testverzeichnis mit dem angegebenem Namen und kopiert diese Daten von testbasis/ ins neu erstellte Verzeichnis. In diesem können sie dann nach Belieben geändert werden, um einen neuen, individuellen Test zu kreieren.

4.3 Erstellen und trainieren neuer Netze

In den meisten Fällen will man eine ganze Serie von Netzen erstellen, denn man möchte testen, wie sich das Ändern eines bestimmten Parameters in einem bestimmten Intervall auswirkt. Dies soll mit dem folgenden Beispiel erklärt werden. Es sollen 10 Netze erstellt werden, mit der Anzahl Neuronen im Hidden Layer von 50 bis 500 in 50er Schritten variierend. Es wird angenommen, dass die Testumgebung bereits erstellt ist (Abschnitt 4.2).

1. In createnet.sh, verändere folgende Variablen. `i` ist die Laufvariable, die von 1 bis 10 läuft, und die Nummer des aktuellen Tests bezeichnet
   ```
   ITER_CNT=10
   HIDDEN_CNT=$(($i*50))
   ```
2. Starte das Erstellen und Trainieren.
   createnet.sh

Es wurden nun 10 neue Netze erstellt, net1.rtdnn bis net10.rtdnn. Die Files logfile1.txt bis logfile10.txt enthalten Informationen über den Trainingsfortschritt. In crestult1.txt bis cresult10.txt ist der Output vom NICO Tool CResult, angewendet auf die fertig trainierten Netze. Der Wordspotter wird nicht automatisch aufgerufen.

Anstatt der Anzahl der Neuronen können natürlich beliebige andere Parameter geändert werden. Dazu muss man einfach createnet.sh oder trainnet.sh entsprechend abändern. Vorgefertigte Parameter wie `HIDDEN_CNT` gibt es jedoch kaum mehr, da dies zu weit geführt hätte. Entweder, man erstellt eigene neue Variablen oder man macht direkt Änderungen in den Aufrufen der NICO Tools. Auf die Laufvariable `i` kann auch in trainnet.sh zugegriffen werden.

Die Pfadinformationen aller benötigten Files liefert environment.sh. Da sehr oft ein Test von nicht Standarddaten abhängt, sollte dieses File vor dem Starten des Skripts auf Richtigkeit überprüft werden.

Wurde aus irgendeinem Grund createnet.sh abgebrochen, kann die Testserie fortgesetzt werden, indem am Anfang des Skipts `i=x-1` gesetzt wird (anstatt `i=0`); wobei x die Nummer des ersten fortzuführenden Tests bezeichnet. Danach kann createnet.sh erneut gestartet werden.

4.4 Gleichverteiltes Training

Um gleichverteilt zu trainieren, muss dem Netz ein Set mit einer gleichverteilten Anzahl Phonemen präsentiert werden. Daher können nicht die vorhandenen Sätze für das Training verwendet werden, da in der deutschen Sprache Phoneme unterschiedlich häufig vertreten sind.

Die Phoneme müssen deshalb aus den Sätzen extrahiert werden. Das heisst, aus den Featurefiles, welche die Sprachdaten enthalten, müssen die einzelnen Phoneme ausgeschnitten und in neue Files gespeichert werden. Dasselbe muss auch mit den Labelfiles geschehen, welche beschreiben, an welcher Position die Phoneme auftreten. Beim Ausschneiden der Phoneme muss zusätzlich ein halbes Kontextfenster mitausgeschnitten werden. Dieser Prozess kann mit dem Java Tool PhonemExtracter ausgeführt werden. Folgender Aufruf extrahiert Phoneme, mit einem Kontextfenster von acht Frames.

java ch.ethz.ee.lnn.PhonemExtracter.PhonemExtracter 4 data/mfcc data/phnlab

Die ausgeschnittenen Files werden in Unterverzeichnissen der im Aufruf angegebenen Pfade gespeichert. Anhang 8.3 geht genauer auf diesen Vorgang ein.

Aus den neuen Labelfiles müssen für das Training Targetfiles erstellt werden. Dazu kann der Lab2Targ Befehl des NICO Toolkit verwendet werden.

Nachdem für einen neuen Test ein entsprechendes Verzeichnis mit den nötigen Files erstellt wurde (Abschnitt 4.2) kann ein gleichverteiltes Trainingsset zusammengestellt werden. Das heisst, es muss eine bestimmte Zahl Feature- und Targetfiles von jedem Phonem in die lokalen Verzeichnisse im aktuellen Testverzeichnis kopiert werden. Dieses

Zusammenkopieren aus den phonem-spezifischen Unterverzeichnissen erledigt das Java Tool CreateDataSet2.

java ch.ethz.ee.lnn.CreateDataSet.CreateDataSet2 20 data/ test001/

Dieses Beispiel würde aus jedem Phonemverzeichnis in *data/mfcc* und *data/phntarg* jeweils 20 Beispielfiles holen und nach *test001/mfcc* und *test001/phntarg* kopieren. Gleichzeitig erstellt CreateDataSet ein File namens *data_utts.txt*, welche eine Liste der Namen aller kopierten Files enthält. Dieses File kann dann in *trainutts.txt* umbenannt und dem Trainingsalgorithmus übergeben werden.

Bevor das Training beginnen kann, müssen die Beispielfiles fürs Testset ebenfalls in die selben Ordner kopiert werden, in welchen sich die Trainingsdaten befinden (*test001/mfcc* und *test001/phntarg*).

4.5 Pädagogisches Training

Damit pädagogisch trainiert werden kann, muss nach einer gewissen Trainingszeit das Trainingsset wieder neu zusammengestellt werden. Das heisst, die bisherigen Trainingsdaten müssen gelöscht und durch ein neues Set ersetzt werden. Dazu löscht man die Target- und Labelfiles, welche sich lokal im Testverzeichnis befinden und kopiert mit dem CreateDataSet Java Tool ein neues Set zusammen. Für das pädagogische Lernen sollen jedoch nicht von jedem Phonem gleich viele Files kopiert werden, denn es sollen ja diejenigen Phoneme häufiger trainiert werden, welche vom Netz schlecht erkannt werden. Um die Erkennungsleistung pro Phonem beurteilen zu können, erstellen wir mit dem CResult Befehl des NICO Toolkit ein Resultfile des Netzes. Dieses File enthält die phonemspezifischen Erkennungswerte. Anhand dieser Erkennungsinformationen kann mit CreateDataSet ein neues pädagogisches Trainingsset zusammengestellt werden. Hierfür kann das Resultfile direkt als Parameter übergeben werden.

java ch.ethz.ee.lnn.CreateDataSet.CreateDataSet2 20 data/ test001/ test001/results_net1.txt

Damit wird anhand der im *results_net1.txt* gespeicherten Erkennungswerten, eine unterschiedliche Anzahl Beispielfiles ins Testverzeichnis kopiert.

Während des Trainingsprozesses sollte fortlaufend immer wieder ein solches neues Set erstellt werden. Um diesen Vorgang zu automatisieren, wurden die oben beschriebenen Aktionen im Trainingsskript *trainnet.sh* integriert. So muss in diesem Skript nur noch angegeben werden, ob pädagogisches Training erwünscht ist und falls ja, nach wie vielen Epochen das Trainingsset neu erstellt werden soll.

Wie beim gleichverteiltem Training, müssen zum Schluss die Files des Testsets ebenfalls lokal ins Testverzeichnis kopiert werden. Dann kann mit dem Training begonnen werden.

4.6 New Alignment

Ein einfacher Test mit neuem Alignment kann mit den folgenden Schritten erstellt werden.

1. Erstelle zwei neue Verzeichnisse an einem beliebigen Ort, je eines für die Label- und die Targetfiles, die dann später neu erstellt werden.

2. Erstelle eine neue Testumgebung und wechsle ins neu erstellte Verzeichnis. Als Testname wurde in diesem Beispiel test001 gewählt.
   createnet.sh test001
   cd test001

3. Kopiere ein bereits ansatzweise trainiertes Netz in das neue Testverzeichnis.

4. Mache folgende Änderungen in trainnet.sh. Damit wird alle 5 Epochen ein neues Alignment erstellt.
   `MAKE_NEW_ALIGNEMENT=1`
   `NEW_ALIGNEMENT_EPOCHS=5`

5. Mache folgende Änderungen in envvariables.sh
   `PREFERENCES_PATH=newalign_preferences`
   `PHNLAB_PATH=` im Punkt 1 erstelltes Verzeichnis
   `PHNTARG_PATH=` im Punkt 1 erstelltes Verzeichnis

6. Starte das Training. Im gegebenen Beispiel ist net1.rtdnn das im Punkt 3 kopierte Netz.
   trainnet.sh net1.rtdnn logfile cresult

Bevor das Training beginnt, werden automatisch die ursprünglichen Label- und Targetfiles von den Verzeichnissen `PHNLAB_ORG_PATH` bzw `PHNTARG_ORG_PATH` in die Verzeichnisse `PHNLAB_PATH` bzw `PHNTARG_PATH` kopiert. Die vorhin angegebenen Variablen sind in envvariables.sh definiert. Die Files in den Verzeichnissen `PHNLAB_PATH` bzw `PHNTARG_PATH` werden während des Trainings immer wieder neu erstellt. Sie sollten deshalb nicht den Verzeichnissen entsprechen, in denen die ursprünglichen Label- und Targetfiles sind. Das neue Alignment wird vom Wordspotter erstellt, der automatisch aufgerufen wird.

Um die folgenden vertiefenden Erläuterungen muss man sich nur kümmern, wenn das präsentierte Beispiel zu wenig tief geht und man weitere Änderungen vornehmen möchte.

Anstatt wie im Punkt 5 vorgeschlagen `PREFERENCES_PATH=newalign_preferences` kann natürlich auch ein eigenes Preferences File angegeben werden. Die fürs Alignment wichtigen Einstellungen im Preferences File sind `MAKE_NEW_ALIGNEMENT=true` und `WITH_DURATION_MODEL=false`. Es ist des weitern zu beachten, dass die folgenden drei Angaben Files referenzieren, die dieselbe Zusammenstellung von Trainingsdaten machen. `TESTUTTS_PATH` und `NEWALIGNUTTS_PATH` in envvariables.sh und `TEST_SET_FILE_PATH` in dem Preferences File, welches in envvariables.sh unter `PREFERENCES_PATH` angegeben ist.

4.7 Qualifying

Im Qualifying werden die Masse für die Worterkennungsleistung durch den Wordspotter berechnet.

1. Erstelle und trainiere ein neuronales Netz wie in den vorhergehenden Beispielen beschrieben.

2. Benütze das NICO Tool Excite, um alle Sätze, die für das Qualifying benützt werden sollen, vom Netz erkennen zu lassen. net1.rtdnn sei das trainierte Netz. Das Set der zu verarbeitenden Sätze ist in in diesem Beispiel in spotutts.txt aufgelistet. Das Ergebnis ist je ein Netzoutput File für jeden Satz.
   Excite -S -X _OutStream net1.rtdnn spotutts.txt

3. Benütze SpotWords, um das eigentliche Qualifying durchzuführen. Der Parameter `TEST_SET_FILE_PATH` im Preferences File, hier *spot_preferences*, gibt das Wordspottertestset File an, in welchem die zu verarbeitenden Netzoutput Files stehen. Zusätzlich sind darin in der zweiten Spalte jeweils die Labelfiles angegeben, die als Referenz benützt werden. Das Ergebnis ist ein Unterverzeichnis für jeden Satz, in welchem die Files *wordout*, *matlab_wordout* und *garbage_score* erstellt werden. Im aktuellen Verzeichnis wird das File *wordout_quali* erstellt, welches die Effizienzmasse des Netzes beinhaltet.
   java SpotWords spot_preferences

Wenn anstatt *spot_preferences* ein anderes Preferences File angegeben wird, ist folgendes zu beachten. Als Erstes muss natürlich `QUALIFY_WORDSPOTTER=true` gesetzt sein. Des weitern muss darauf geachtet werden, dass die im Wordspottertest File (referenziert durch `TEST_SET_FILE_PATH`) angegebenen Netzoutput Files auch tatsächlich vorhanden sind, bzw. mit Excite erstellt wurden. Deshalb ist es am Besten, wenn das Setfile, das Excite als Argument übergeben wird, und das Wordspottertestset File ein und dasselbe File sind. Excite kümmert sich nicht darum, dass im Wordspottertestset File zwei Spalten statt nur eine stehen.

5 Qualitäts- und Fehlermasse

Um die Erkennungsleistung des Phonemerkenners auf verschiedenen Ebenen beurteilen zu können, müssen entsprechend Fehlermasse definiert werden.

Das ausschlaggebende Mass für die Bewertung der Erkennung liefert der Wordspotter. Dieser liefert uns als Ergebnis, welche Wörter an welcher zeitlichen Position im verarbeiteten Sprachausschnitt erkannt wurden.

5.1 Masse für die Worterkennungsleistung

Zum einen gibt der Wordspotter eine Rangliste aller Wörter aus, die so genannte *Word Score List*, welche nach der Auftrittswahrscheinlichkeit der Worte sortiert ist. Diese Liste enthält alle Wörter aus dem Wörterbuch, welches der Wordspotter für die Erkennung benutzt hat. Das heisst, je höher in der Rangliste ein Wort fungiert, umso wahrscheinlicher wird dessen Auftreten im Sprachsignal beurteilt. Daraus lassen sich zwei Erkennungsmasse ableiten:

Worst Wordposition gibt die schlechteste Position eines Wortes in der *Word Score List* an, welches tatsächlich im zu erkennenden Sprachausschnitt vorkommt.

Mean Wordposition gibt die durchschnittliche Positionierung der Worte in der *Word Score List* an, welche tatsächlich im zu erkennenden Sprachausschnitt vorkommen.

Als Weiteres gibt uns der Wordspotter aus, an welcher zeitlichen Position im Sprachsignal die Wörter vorkommen sollen. Diese Positionierung wird in Frames angegeben. Da dieses Resultat nicht immer mit der Realität übereinstimmt, wird die Exaktheit dieser Angabe als Qualitätsmass angegeben.

Mean Displacement gibt an, um wieviele Frames im Durchschnitt die zeitliche Positionierung der Worte vom tatsächlich vorkommenden Ort abweichen.

5.2 Masse für die Phonemerkennungsleistung

Da das Testen der Erkennungsleitstung mittels Wordspotter viel Zeit benötigt, macht es Sinn, weitere Masse einzuführen, um die Güte eines neuronalen Netzes auf der Phonemerkennungsebene zu beurteilen. Das erste Mass gibt an, in wievielen Frames das korrekte Phonem erkannt wurde. Dabei gilt jenes Phonem als erkannt, welchem das Netz die höchste Auftrittswahrscheinlichkeit zuordnet.

Frame Correctness gibt an, wieviel Prozent der verarbeiteten Frames korrekt klassifiziert werden.

Dieses Mass gibt an, wie gut die Phoneme im Durchschnitt erkannt werden, sagt aber nichts darüber aus, wie gross sich die Erkennungsleistung zwischen den Phonemen unterscheidet. Denn beim Trainieren des Netzes mit ganzen Sätzen kann davon

ausgegangen werden, dass häufig vorkommende Phoneme besser erkannt werden als andere. Man kann sich jedoch vorstellen, dass der Wordspotter korrekter arbeiten kann, wenn die Phonemerkennungsleistung ausgeglichen ist. Siehe dazu [3]. Aus diesem Grund führten wir als ein weiteres Mass die Ausgeglichenheit der Phonemerkennung ein.

Balance gibt die Varianz der phonemspezifischen Frame Correctness an.

Bei der Balance ist zu beachten, dass je grösser ihr Wert, umso schlechter ist die Ausgeglichenheit.

Bevor die oben definierten Masse zur Beurteilung des Netzes angewendet werden können, muss das Netz trainiert werden. Während dem Training mit dem NICO Tool, haben wir die Möglichkeit nach jeder Lernepoche ein Fehlermass des Netzes anzeigen zu lassen, welches vom NICO Tool berechnet wird. Das erlaubt uns, die Entwicklung des Lernprozesses zu beobachten.

Validation Error gibt die Abweichung des Netzoutputs vom vorgegebenen Output an.

5.3 Beurteilung der Verallgemeinerungsfähigkeit

Um eine Aussage über die Verallgemeinerungsfähigkeit des Erkenners zu erhalten, muss darauf geachtet werden, dass die Masse auf einem definierten Testset berechnet werden. Das Testset soll ein Set von Sprachdaten sein, welches nicht zum Trainieren des Netzes verwendet wurde.

6 Resultate

In diesem Abschnitt werden die Resultate aller Tests in Form von Tabellen und Grafiken präsentiert. Die Auswertung der Resultate erfolgt jedoch erst in Abschnitt 7. Die folgende Tabelle vermittelt einen Überblick über alle Tests.

Tabelle 1: *Üebersicht der Testresultate*

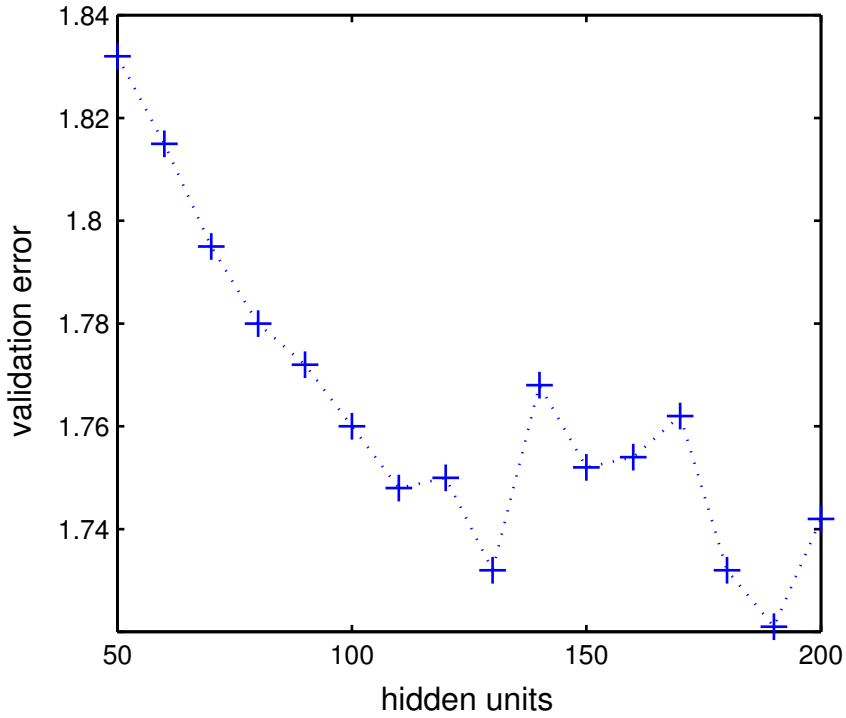
6.1 Netztopologien

6.1.1 Anzahl Hidden Units

Test 1.1: Netze mit unterschiedlicher Anzahl Neuronen in einem Hidden Layer.

Übersicht Parameter Test 1.1	
Netztopologie	
Anzahl Hidden Layer	1
Anzahl Hidden Units Layer 1	**50 - 200**
Anzahl Hidden Units Layer 2	0
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	Gleichverteilt
Grösse des Trainingsset	600 Phoneme
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	0.7
Update Frequenz	10 Frames

Tabelle 2: *Test 1.1*

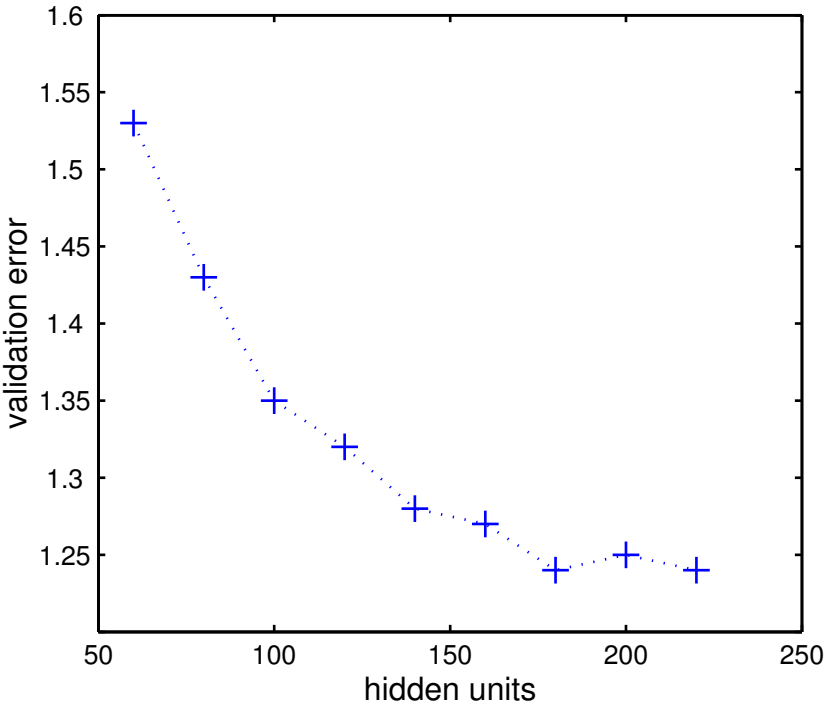


Figur 10: *Test 1.1: 16 Netze mit unterschiedlich grossem Hidden Layer. Auf der x-Achse ist die Anzahl Neuronen im Hidden Layer aufgetragen.*

Test 1.2: Netze mit unterschiedlicher Anzahl Neuronen in zwei Hidden Layer. Dabei haben beide Hidden Layer dieselbe Anzahl Neuronen.

Übersicht Parameter Test 1.2	
Netztopologie	
Anzahl Hidden Layer	2
Anzahl Hidden Units Layer 1	**30 - 110**
Anzahl Hidden Units Layer 2	**30 - 110**
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	Gleichverteilt
Grösse des Trainingsset	600 Phoneme
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	0.7
Update Frequenz	10 Frames

Tabelle 3: *Test 1.2*



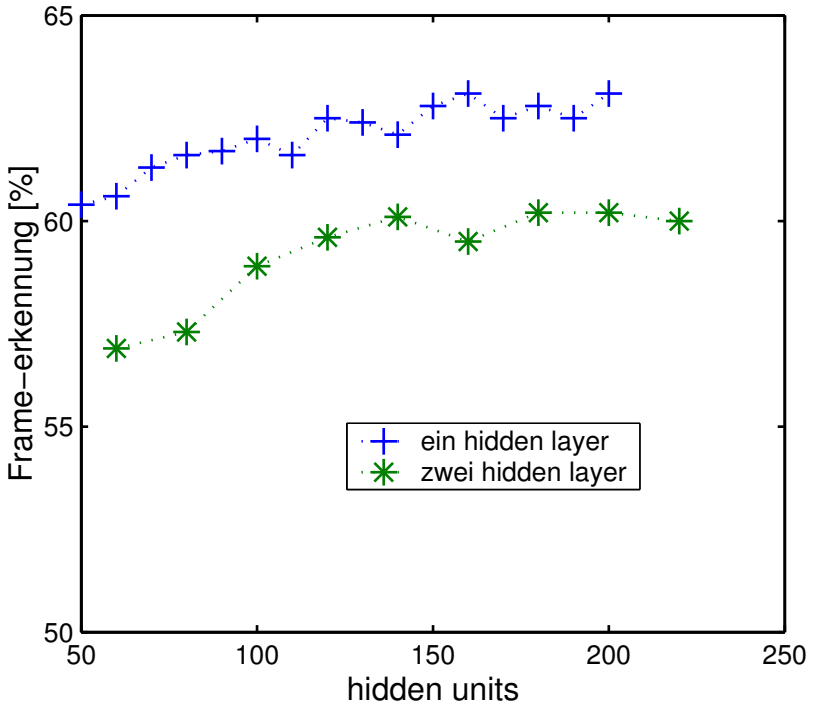
Figur 11: *Test 1.2: Neun Netze mit unterschiedlich vielen Hidden Units. Auf der x-Achse ist die gesamte Anzahl Neuronen beider Hidden Layer aufgetragen.*

6.1.2 Anzahl Hidden Layer

Test 2: Netze mit einem und mit zwei Hidden Layer. Dabei werden Netze mit unterschiedlich grossen Hidden Layers verglichen.

Übersicht Parameter Test 2	
Netztopologie	
Anzahl Hidden Layer	**1 - 2**
Anzahl Hidden Units Layer 1	**30 - 200**
Anzahl Hidden Units Layer 2	**30 - 110**
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	Gleichverteilt
Grösse des Trainingsset	600 Phoneme
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	0.7
Update Frequenz	10 Frames

Tabelle 4: *Test 2*



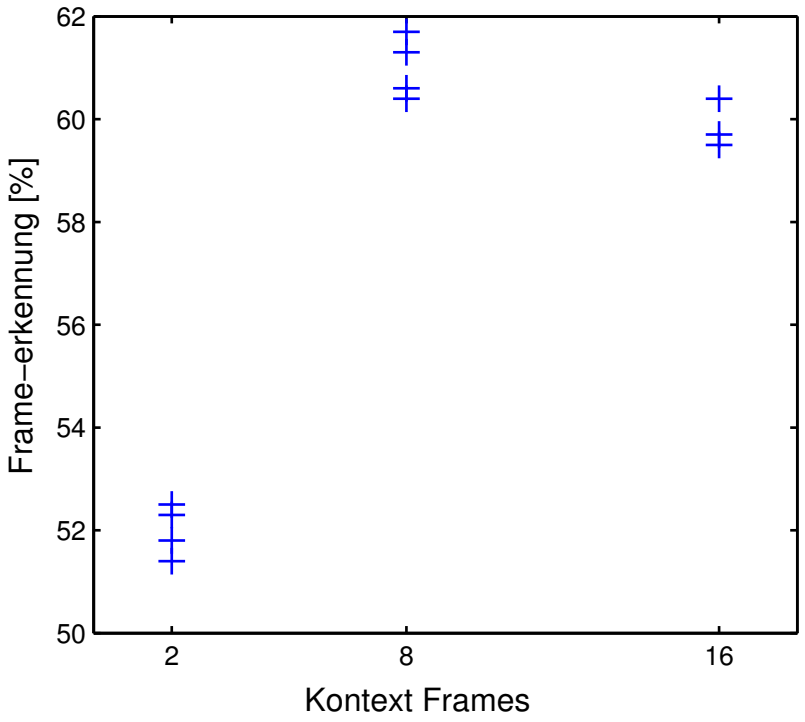
Figur 12: *Test 2: Je neun Netze mit unterschiedlich grossen Hidden Layers. Auf der x-Achse ist die gesamte Anzahl Neuronen beider Hidden Layer aufgetragen.*

6.1.3 Kontext Frames

Test 3: Netze mit einem 2, 8 und 16 Frames grossen Kontextfenster. Dabei werden für jedes einzelne Kontextfenster wiederum mehrere Netze mit unterschiedlich grossem Hidden Layer trainiert.

Übersicht Parameter Test 3	
Netztopologie	
Anzahl Hidden Layer	1
Anzahl Hidden Units Layer 1	**60 - 150**
Anzahl Hidden Units Layer 2	**60 - 150**
Anzahl Kontextframes	**2 - 16**
Trainingsmethode	
Präsentationsmodus	Gleichverteilt
Grösse des Trainingsset	600 Phoneme
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	0.7
Update Frequenz	10 Frames

Tabelle 5: *Test 3*



Figur 13: *Test 3: Auf der x-Achse ist die gesamte Anzahl Frames aufgetragen, welche vor und nach dem zu klassifizierenden Frame dazugenommen wurden, das zu klassifizierende Frame nicht mitgezählt. Für alle drei Kontextvarianten wurden vier Netze mit 60, 90, 120 und 150 Hidden Units getestet.*

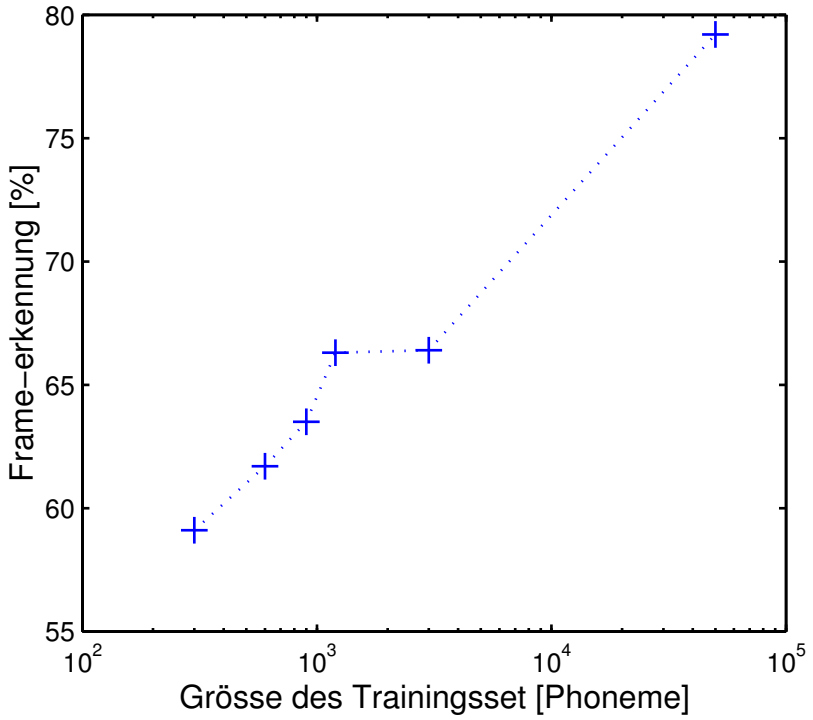
6.2 Trainingsmethoden

6.2.1 Grösse des Trainingsset

Test 4: Ein Vergleich zwischen sechs Netzen, welche mit unterschiedlich grossem Trainingsset trainiert wurden. Die ersten fünf Netze wurden mit einem gleichverteilten Phonemset trainiert, das letzte mit dem gesamten zur Verfügung stehenden Trainingsset von ca. 50'000 Phonemen.

Übersicht Parameter Test 4	
Netztopologie	
Anzahl Hidden Layer	1
Anzahl Hidden Units Layer 1	90
Anzahl Hidden Units Layer 2	0
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	**Gleichverteilt / Ungleichverteilt**
Grösse des Trainingsset	**300 - 50'000 Phoneme**
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	0.7
Update Frequenz	10 Frames

Tabelle 6: *Test 4*



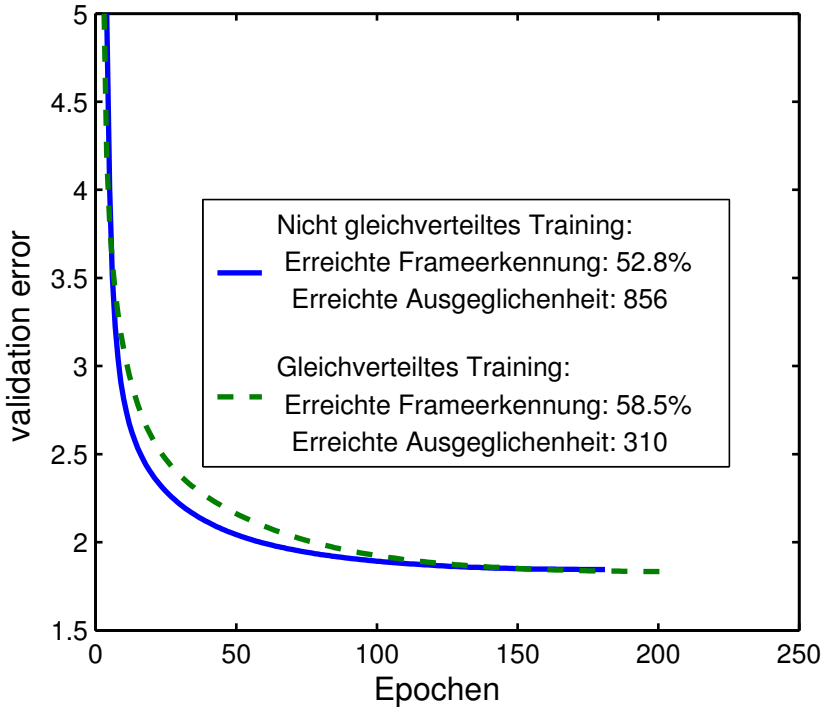
Figur 14: *Test 4: Sechs Netze, trainiert mit verschieden grossen Trainingssets. Sie umfassen 300, 600, 900, 1200, 3000 und ca. 50'000 Phoneme*

6.2.2 Gleichverteiltes Training

Test 5: Ein Vergleich der Lernkurven zweier Netze. Das erste wurde mit einem gleichverteilten Trainingsset trainiert, das zweite mit einem konventionellen (nicht gleichverteilten) Trainingsset.

Übersicht Parameter Test 5	
Netztopologie	
Anzahl Hidden Layer	1
Anzahl Hidden Units Layer 1	50
Anzahl Hidden Units Layer 2	0
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	Gleichverteilt Ungleichverteilt
Grösse des Trainingsset	300 Phoneme
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	0.7
Update Frequenz	10 Frames

Tabelle 7: *Test 5*



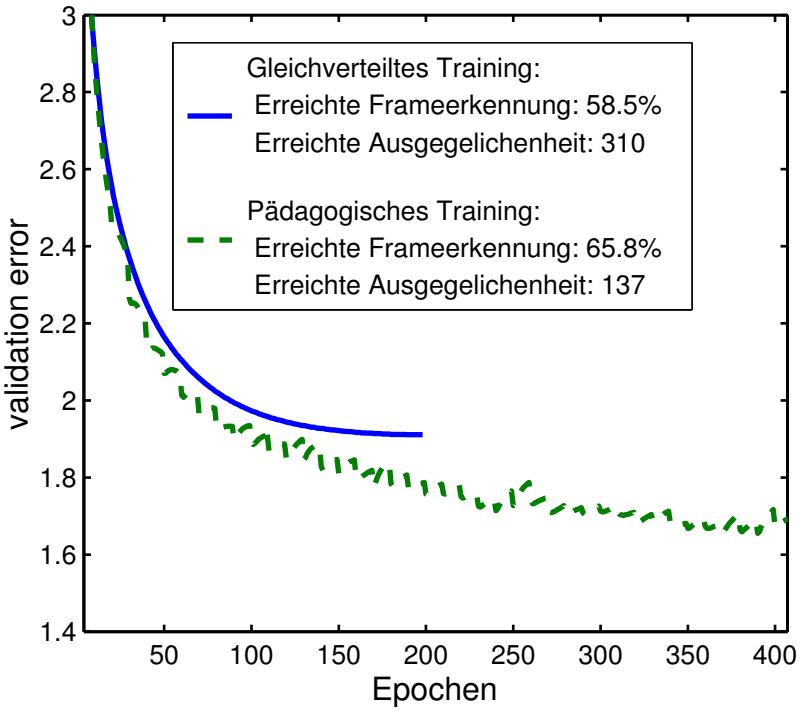
Figur 15: *Test 5: Die Trainingszeit der beiden Netze wird auf der X-Achse in Anzahl Epochen angezeigt.*

6.2.3 Pädagogisches Training

Test 6: Vergleich zwischen gleichverteiler und pädagogischer Phonempräsentation.

Übersicht Parameter Test 6	
Netztopologie	
Anzahl Hidden Layer	1
Anzahl Hidden Units Layer 1	50
Anzahl Hidden Units Layer 2	0
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	**Gleichverteilt / Pädagogisch**
Grösse des Trainingsset	300 Phoneme
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	0.7
Update Frequenz	10 Frames

Tabelle 8: *Test 6*



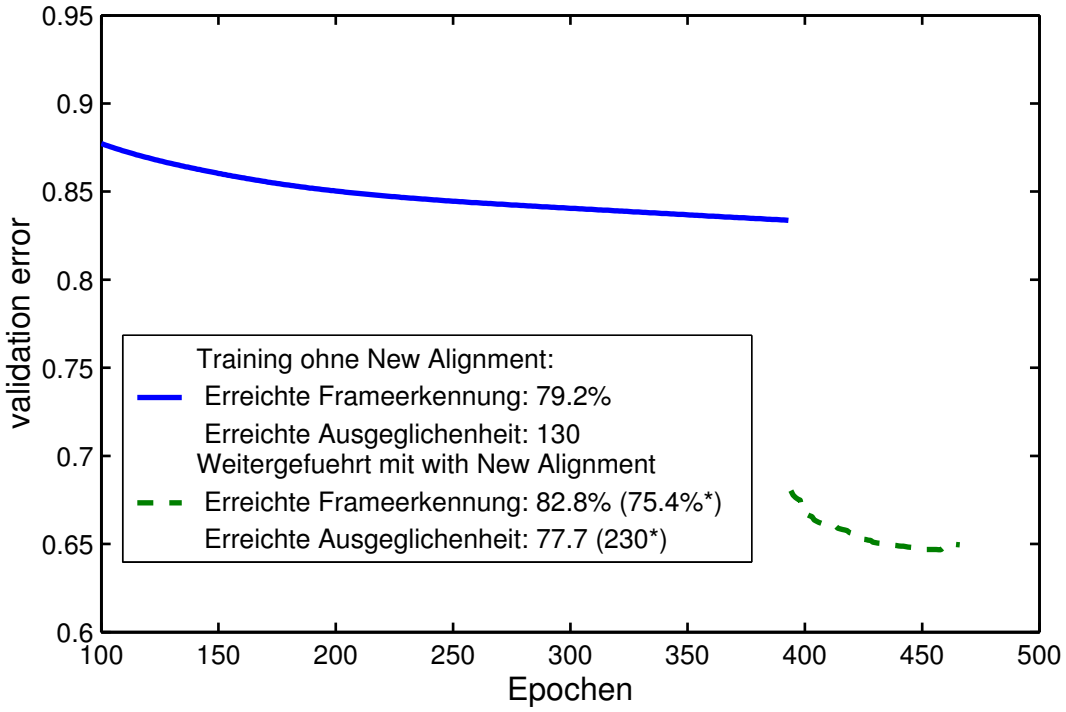
Figur 16: *Test 6: Die Trainingszeit der beiden Netze wird auf der X-Achse in Anzahl Epochen angezeigt. Beim gleichverteilt trainierten Netz wurde der Lernvorgang nach 200 Epochen abgebrochen.*

6.2.4 Training mit New Alignment

Test 7: Training eines Netzes mit dem gesamten Trainingsset bis der Lernvorgang keine wesentlichen Fortschritte mehr zeigte. Danach wurde das Trainingsset und das Testset mit New Alignment neu erstellt und damit weitertrainiert. Ab diesem Zeitpunkt wurde nach jeweils 5 Epochen das Trainings- und Testset mit New Alignment korrigiert.

Übersicht Parameter Test 7	
Netztopologie	
Anzahl Hidden Layer	2
Anzahl Hidden Units Layer 1	50
Anzahl Hidden Units Layer 2	50
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	Ungleichverteilt
Grösse des Trainingsset	50'000 Phoneme
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	0.7
Update Frequenz	10 Frames

Tabelle 9: *Test 7*



Figur 17: *Test 7: Die Gesamttrainingszeit des Lernvorgangs ist auf der X-Achse in Anzahl Epochen angezeigt.*

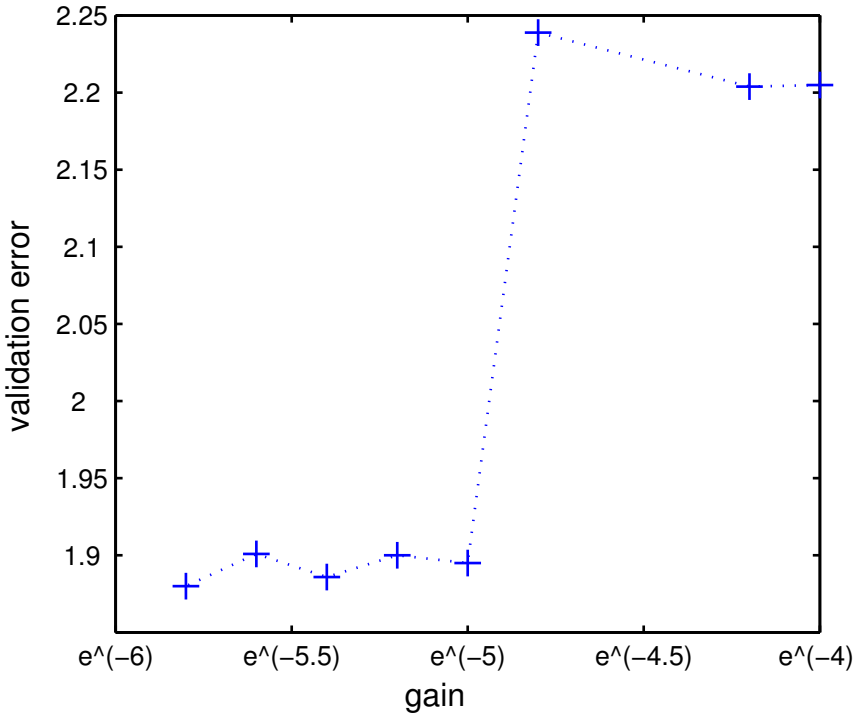
6.3 Parameter des Backpropagation-Algorithmus

6.3.1 Gain

Test 8: Ein Vergleich zwischen Netzen, welche mit unterschiedlichem Gain trainiert wurden.

Übersicht Parameter Test 8	
Netztopologie	
Anzahl Hidden Layer	1
Anzahl Hidden Units Layer 1	80
Anzahl Hidden Units Layer 2	0
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	Gleichverteilt
Grösse des Trainingsset	300 Phoneme
Trainingsparameter	
Gain	$1e^{-4}$ - $1e^{-5.8}$
Momentum Term	0.7
Update Frequenz	10 Frames

Tabelle 10: *Test 8*



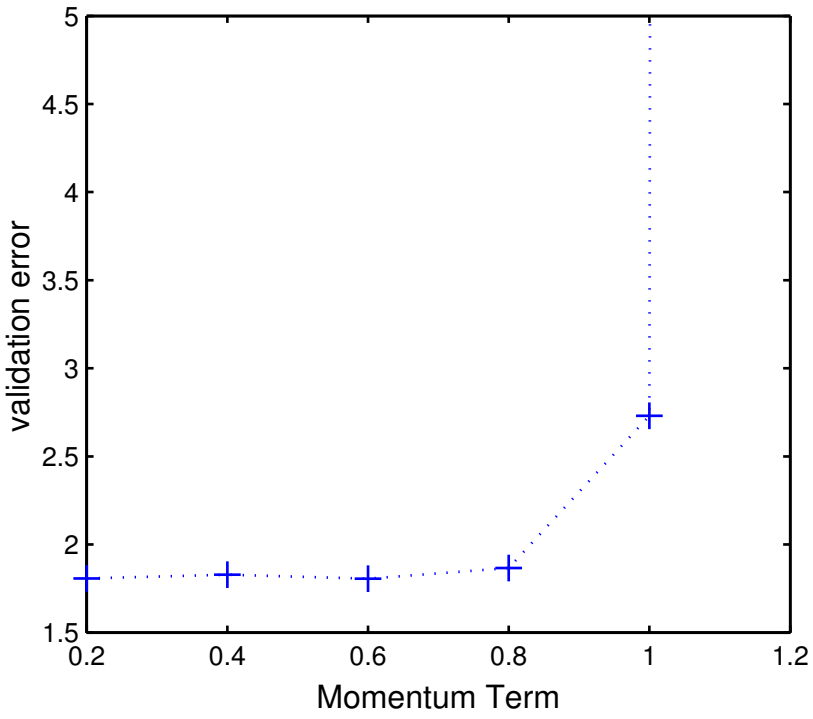
Figur 18: *Test 8*

6.3.2 Momentum Term

Test 9: Ein Vergleich zwischen Netzen, welche mit unterschiedlich grossem Momentum Term trainiert wurden.

Übersicht Parameter Test 9	
Netztopologie	
Anzahl Hidden Layer	1
Anzahl Hidden Units Layer 1	50
Anzahl Hidden Units Layer 2	0
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	Gleichverteilt
Grösse des Trainingsset	600 Phoneme
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	**0.2 - 1.2**
Update Frequenz	10 Frames

Tabelle 11: *Test 9*



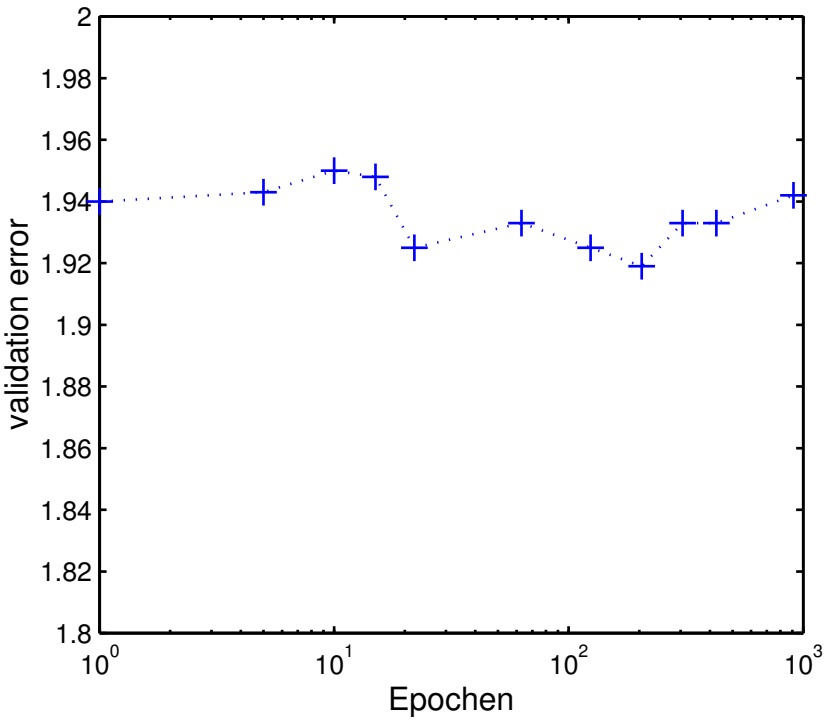
Figur 19: *Test 9*

6.3.3 Update Frequenz

Test 10: Ein Vergleich zwischen Netzen, welche mit unterschiedlicher Update Frequenz trainiert wurden. D.h. die Gewichte werden während dem Training nach der Präsentation einer gegebenen Anzahl Beispiele verändert. Diese Anzahl wird durch die Update Frequenz ausgedrückt.

Übersicht Parameter Test 10	
Netztopologie	
Anzahl Hidden Layer	1
Anzahl Hidden Units Layer 1	80
Anzahl Hidden Units Layer 2	0
Anzahl Kontextframes	8
Trainingsmethode	
Präsentationsmodus	Gleichverteilt
Grösse des Trainingsset	300 Phoneme
Trainingsparameter	
Gain	$1e^{-5}$
Momentum Term	0.7
Update Frequenz	**1 - 900 Frames**

Tabelle 12: *Test 10*



Figur 20: *Test 10*

6.4 Vergleich der besten Netze

In der Tabelle 13 sind diejenigen Netze eingetragen, welche, je nach betrachtetem Fehlermass, am Besten abgeschnitten haben. Da die Bewertungen des Wordspotters entscheidend sind, ist die Liste nach diesem Mass sortiert. Die fünf Netze haben folgende Eigenschaften gemeinsam: Alle Netze benützen ein Kontextfenster von acht Frames. Für das Training wurden bei allen Netzen dieselben Parameter für den Backpropagation-Algorithmus verwendet:

- Gain: $1e^{-5}$
- Momentum Term: 0.7
- Update Frequenz: 10 Frames

Netz 1: Ein Netz mit zwei Hidden Layer mit je 50 Neuronen. Ungleichverteiltes Training mit dem ganzen Trainingsset bis der Validation Error nicht mehr abnahm. Mit pädagogischem Training weiter trainiert.

Netz 2: Ein Netz mit zwei Hidden Layer mit je 110 Neuronen. Ungleichverteiltes Training mit dem gesamten Trainingsset.

Netz 3: Ein Netz mit einem Hidden Layer mit 120 Neuronen. Gleichverteiltes Training mit einem Trainingsset von 600 Phonemen. Nach ca. 20 Epochen wurde mit pädagogischem Training weitergefahren.

Netz 4: Dasselbe Netz wie Netz 1, jedoch mit einer noch längeren pädagogischen Trainingsphase.

Netz 5: Dasselbe Netz wie Netz 1, jedoch anstelle des pädagogischen Trainings wurde das Training mit New Alignment weitergeführt.

Netz	**Frame Correctness**	**Balance**	**Worst Wordposition**	**Mean Wordposition**	**Displacement**
Netz 1	77.9 %	40.9	404.6	87.6	9.62
Netz 2	81.3 %	102.3	485.6	105.0	9.86
Netz 3	75.9 %	62.1	539.6	106.4	10.70
Netz 4	76.7 %	28.03	552.2	111.3	11.30
Netz 5	75.4 % (85.8 %)*	230.5 (77.7)*	655.5	139.8	12.21

Tabelle 13: *Bewertung der Erkennungsleistung. *Erkennungleistung auf dem Testset, welches mit New Alignment erstellt wurde.*

7 Diskussion

7.1 Netztopologien

Die Testresultate der Versuche mit verschiedenen Netzgrössen und -topologien zeigen, dass diese Parameter innerhalb eines grossen Bereichs keinen wesentlichen Einfluss auf die Erkennungsleistung ausüben. Dies verdeutlicht sich vor allem im Vergleich zu anderen Parameteränderungen.

Beim Verändern der Anzahl Neuronen im Hidden Layer (*Test 1*) beträgt der Unterschied des Validation Error zwischen dem besten und schlechtesten Netz gerade mal 0.1. Oder in Frame Correctness ausgedrückt: Die Frame Correctness des Netzes mit 50 Neuronen beträgt 60.4% und das beste Netz steigert sich lediglich um 2.7% , auf 63.1% (Netz mit 160 und 200 Neuronen). Dieser Test zeigt weiter, dass die Erkennungsleistung mit steigender Neuronenzahl tendenziell besser wird. Wie erwartet wurde die obere Grenze nicht erreicht, wo das Netz Gefahr laufen könnte, sich zu genau aufs Trainingsset zu spezialisieren (Abschnitt 2.1.1). Trotzdem macht es Sinn, keine zu grosse Neuronenzahl zu wählen. Denn der zeitliche Trainingsaufwand steigt mit zunehmender Anzahl Neuronen und eine wesentliche Verbesserung des Netzes ist nicht zu erwarten, wenn eine untere Grenze von 50 - 90 Neuronen überschritten wurde.

Netze, welche sich durch die Anzahl Hidden Layers unterscheiden (*Test 2*), weisen hinsichtlich der Erkennungsleistung ebenfalls keinen grossen Unterschied auf. Beim direkten Vergleich zweier Netze mit derselben Neuronenzahl aber ungleicher Anzahl Hidden Layer, schneiden die Netze mit einem Hidden Layer 2-3% besser ab. Doch muss beachtet werden, dass ein Netz, welches die Neuronen auf zwei Hidden Layer verteilt, weniger Verbindungen aufweist, als die Topologie mit einem Hidden Layer. Um Netze mit gleicher Anzahl Verbindungen zu vergleichen, müsste ein Netz mit einer Zwischenschicht mit einem Netz mit zwei Hidden Layer verglichen werden, welches 30 - 40 Neuronen mehr enthält.

Beim Verändern der Grösse des Kontextfensters (*Test 3*), zeigt sich, dass durch hinzuziehen von Kontextframes das Netz verbessert wird (+10% Frame Correctness). Doch darf der Kontext nicht zu gross gewählt werden, da sonst die Gefahr besteht, dass zu viele unnütze Frames von zeitlich benachbarten Phonemen dem Eingang des Netzes zugeführt wird. Ein Fenster von acht Frames scheint ein sinnvoller Ansatz zu sein.

7.2 Trainingsmethoden

Ein sehr entscheidender Parameter bei der Suche nach einem erfolgreichen Netz ist die Grösse des Sets an Lernbeispielen. Die höchsten Erkennungswerte wurden durch das Training auf dem ganzen Lernset erreicht (*Test 4*). Dabei wurden dem Netz in jeder Epoche alle 900 Sätze oder umgerechnet etwas 50'000 Phoneme präsentiert. Das Training beanspruchte dadurch sehr viel Zeit. Um ein erfolgreiches Training abzuschliessen, wurden Netze mehrere Tage, bis zu einer Woche, mit dem ganzen Lernset trainiert. Ein weiterer Nachteil ist die Unausgeglichenheit der Phoneme im gesamten Trainingsset. Dies führt dazu, dass ein entsprechend trainiertes Netz Phoneme,

welche in der natürlichen Sprache häufiger vorkommen, besser erkennen kann. Ein solches Netz weist eine schlechte Balance aus.

Eine Steigerung der Balance und der Frame Correctness wird durch gleichverteiltes Training erzielt (*Test 5*). Die Präsentation einzelner Phoneme während dem gleichverteilten Training hatte andererseits den Nachteil, dass die Lerngeschwindigkeit abnahm. Die Anzahl Phoneme, welche dem Netz pro Epoche präsentiert wurden, war in etwa zehn mal kleiner gegenüber dem nicht gleichverteilten Training, bei welchem ganze Sätze dem Backpropagation-Algorithmus zugeführt werden konnten.

Ein weiterer Schritt in diese Richtung wird mit pädagogischem Training erreicht. Das heisst, Frame Correctness und vor allem die Balance konnten Dank dieser Präsentationsmethode im Beispiel von *Test 6* weiter gesteigert werden. Doch die Trainingszeit wird durch die periodisch neue Zusammenstellung des Lernsets um weitere Faktoren vergrössert.

Mit der Anwendung des New Alignment Algorithmus kann eine deutliche Senkung des Validation Errors verzeichnet werden (*Test 7*). Doch kann der Validation Error des ohne New Alignment trainierten Netzes nicht eins zu eins mit demjenigen des weitergeführten Lernprozesses verglichen werden. Denn durch das New Alignment wird die Beschreibung der Phonempositionen des Trainings- sowie des Testsets neu angepasst. Das heisst der Validation Error, welcher aufgrund der Testdaten berechnet wird, verbessert sich alleine schon durch das dem Netz angepasste Testset. Um die Erkennungsleistung des mit New Alignment trainierten Netzes zu bestimmen, müssen deshalb die Fehlermasse auf dem original sowie dem angepassten Testset berechnet werden. So widerfährt der Frame Correctness und der Balance eine deutliche Verbesserung auf dem neuen Testset, welche durch ein Miteinbeziehen der Masse auf dem original Testset relativiert werden muss.

7.3 Parameter des Backpropagation-Algorithmus

Die Parametereinstellungen des Lernalgorithmus tragen wesentlich zum Erfolg des Lernprozesses bei. So sind dem Anschein nach geringe Veränderungen entscheidend, ob das Netz konvergiert oder nicht.

Die Resultate der Gain- und Momentum-Testreihe lassen eindeutige Schlüsse zu.

Für ein erfolgreiches Training darf der Gain Wert nicht grösser als e^{-5} betragen (*Test 8*). Um aber das Lernen nicht unnötig zu verlangsamen, sollte dieser Wert auch nicht zu klein gewählt werden. Aus diesem Grund scheint ein Gain von e^{-5} optimal zu sein.

Ein ähnliches Bild zeigt *Test 9*. Bei zu gross gewähltem Momentum Term über 1.0 kann der Lernprozess nicht mehr konvergieren. Unter der 0.7 Marke scheint das Training am Erfolgversprechendsten. Auch hier gilt, ein kleiner Momentum Term verlängert die Trainingszeit. Wir vermuten deshalb, dass ein Wert von 0.7 angebracht ist.

Keinen Einfluss hingegen kann bei der Wahl der Update Frequenz ausgemacht werden (*Test10*). Deswegen beliessen wir diesen Parameter in den anderen Tests auf dem Default Wert, welcher 10 Frames beträgt.

7.4 Das beste Netz

Nach Auswertung der Resultate sind wir zum Schluss gekommen, dass folgende Parametereinstellungen des Lernprozesses optimal scheinen.

- Beim Backpropagation-Algorithmus scheint es vernünftig für die Werte Gain, Momentum Term und Update Frequenz die von uns gefundenen Einstellungen zu wählen: e^{-5}, 0.7 und 10 Frames.
- Bei der Lernmethode ist vor allem darauf zu achten, dass ein möglichst grosses Trainingsset benutzt wird. Gleichzeitig soll so trainiert werden (z.B. mit pädagogischem Training), dass die Erkennungsleistung der einzelnen Phoneme möglichst ausgeglichen ist.
- Die Grösse des Hidden Layer sollte mindestens 50 - 90 Neuronen betragen; gegen oben konnte keine Grenze ausgemacht werden. Bei dieser Wahl muss ein Kompromiss zwischen Trainingszeit und Lernkapazität gefunden werden
- Ein Kontextfenster von acht Frames scheint angemessen zu sein.

Von den gefundenen Netzen, welche gemäss Frame Correctness und Balance am besten abschneiden, liefert *Netz 1* die besten Resultate des Wordspotters (Abschnitt 6.4).

Es scheint sich zu bestätigen, dass sowohl die Frame Correctness wie auch die Balance entscheidend sind bei der Auswahl des besten Netzes. Dass eine hohe Frame Correctness alleine nicht reicht, zeigt *Netz 2*. Anderseits genügt eine gute Balance allein auch nicht. Dies war jedoch von vornherein bereits ersichtlich, denn könnte ein Netz, welches kein einziges Phonem erkennen kann, trotzdem eine hohe Balance erreichen.

8 Ausblick

Verschiedene Bereiche dieser Arbeit können weiter untersucht und Methoden können verbessert werden. In den folgenden Abschnitten geben wir unsere Vorschläge zur Weiterführung dieser Arbeit an.

8.1 Freiheitsgrade beim Training von neuronalen Netzen

Wir haben eine Anzahl Freiheitsgrade der Topologie und des Trainings der neuronalen Netze getestet. Eine Ergänzung dieser Testreihe könnte in drei Richtungen erfolgen. Erstens können die Parameter, welche von uns untersucht worden sind in einem grösseren Bereich getestet werden.
Zweitens sind weitere Tests möglich mit Freiheitsgraden, die von uns gar nicht erst untersucht worden sind.
Drittens kann davon ausgegangen werden, dass gewisse Freiheitsgrade nicht unabhängig sind. Das heisst es müssten noch weitere Tests gemacht werden, welche zwei oder drei Parameterkombinationen untersuchen.

8.1.1 Topologien

Beim Testen der Neuronenzahl sind wir auf keine obere Grenze gestossen. Denn mit stetig steigender Rechenleistung können auch sehr grosse Netze immer schneller trainiert werden.

Verschiedene Topologiemöglichkeiten mit Verbindungen zwischen Neuronen derselben oder der darunter liegenden Schicht könnten ebenfalls untersucht werden. Obschon nach [3] damit keine grossen Verbesserungen zu erwarten sind. Die Möglichkeit, verzögernde Elemente in höheren Layers einzusetzen, wurde von uns ebenfalls nicht getestet.

Bei der Suche nach der optimalen Kontextfenstergrösse der Inputframes haben wir uns mit drei Tests begnügt. Dieses Optimum könnte noch exakter ermittelt werden. Weiter wäre es denkbar, dass die optimale Fenstergrösse von der Grösse des Hidden Layer abhängt, da mit zunehmender Neuronenzahl mehr Information verarbeitet werden kann.

8.1.2 Trainingsmethoden

New Alignment und pädagogisches Training könnten kombiniert werden. Denn unsere Versuche mit dem pädagogischen Training basierten auf denjenigen Labelfiles, die mit HMM's erstellt wurden und eben Fehler aufweisen. Wir schlagen die folgende Vorgehensweise vor: Zuerst ein Netz mit geeigneter Topologie erstellen und einige Epochen lang standardmässig trainieren, bis eine 'gute' Erkennungsrate erreicht wird. Danach mit New Alignment weiter trainieren, um die Labelfiles zu korrigieren. Zum Schluss bis zum Overfitting mit den neuen Labelfiles pädagogisch trainieren, um eine gute Balance zu erreichen.

8.1.3 Trainingsparameter

Wir haben keine Versuche mit unterschiedlichen Aktivierungsfunktionen unternommen und uns auf *tanh(x)* beschränkt, die Standartfunktion des NICO Toolkit. Den Gain- und Momentumparameter haben wir unabhängig voneinander getestet. Es könnte eine optimalere Konfiguration der beiden Parameter gefunden werden, wenn kombinierte Untersuchungen durchgeführt würden.

8.2 Verbesserung der Trainingsgeschwindigkeit

Zur Erhöhung der Trainingsgeschwindigkeit sollte ein 'Gain Decay' eingeführt werden. Wie bereits erwähnt, funktioniert die von NICO bereitgestellte Gain Decay Funktion nicht. Das von uns erstellte Java Programm BackPropCtrl könnte neben dem Kontrollieren des Overfitting auch die Kontrolle über den Gain übernehmen. Dazu muss jedoch ein Weg gefunden werden, wie dieses Java Programm den Gain Parameter verändern kann, der im Skript trainnet.sh dem Backprop Tool übergeben wird.

Das pädagogische Training kann ebenfalls beschleunigt werden. Um ein neues angepasstes Trainingsset zusammenzustellen, wird das Java Programm *CreateDataSet2* benutzt. Es kopiert jeweils die benötigten Files von einem Quellenverzeichnis in das lokale Testverzeichnis, was ziemlich zeitintensiv ist. Dieses Konzept, dass wir ein Verzeichnis mit alles Trainingsdaten führen und gleichzeitig die Files, welche für einen Test benötigt werden in das entsprechende Testverzeichnis kopieren, kommt von folgender Überlegung: Da die Trainingsdaten durch Veränderung des Kontextfensters oder mit New Alignment nicht mehr für alle Tests die gleichen sind, wollten wir die für einen Test benötigten Daten im Testverzeichnis behalten. Dies führt jedoch wie erwähnt dazu, dass diese Daten, um ein neues Trainingsset zu erstellen, kopiert werden müssen. *CreateDataSet2* könnte so verändert werden, dass nur ein Trainutts File mit all den zu trainierenden Beispielen erstellt wird und der Kopiervorgang unterdrückt wird.

8.3 Skripts

Ein grosser Nachteil der Skripts und Konfigurationsfiles ist immer noch, dass einige Referenzen auf benötigte Files an verschiedenen Orten auftauchen. Wenn der Anwender nicht genügend acht gibt bei einem neuen Test alle Referenzen anzupassen, entstehen undefinierbare Fehler. Die Skripts und Konfigurationsfiles sollten also dahingehend verbessert werden, dass benötigte Files nur an einer einzigen Stelle referenziert werden.

Literatur

[1] Jakob Bernasconi. *Informationsverarbeitung in neuronalen Netwerken.* Skript zur Vorlesung vom Sommersemester 2001, ABB Corporate Research Ltd., CH-5405 Baden-Dätwil

[2] Beat Pfister, Hans-Peter Hutter *Sprachverarbeitung* Skript zur Vorlesung vom Wintersemester 02/03, Institut für technische Informatik und Kommunikationsnetze ETH Zürich

[3] Joe Tebelskis. *Speech Recognition using Neural Networks.* School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania

[4] Herve Bourlard, Bart D'hoore, Jean-Marc Boite. *Optimizing recognition and rejection performance in wordspotting systems.* L&H Speech Products, Rozendaalstraat 14, B-8900 Ieper, Belgium

[5] S.J. Young, N.H. Russell, J.H.S. Thornton. *Token Passing: a Simple Conceptual Model for Connected Speech Recognition Systems.* Cambridge University Engineering Departement

[6] Nikko Strom. *The NICO Toolkit.* http://www.speech.kth.se/NICO/MANUAL/index.html

Anhang A

Netz (*.rtdnn)
Das neuronale Netz gespeichert in einem File.

Audiofiles (*.au)
Enthalten die gesprochenen Audiodaten.

Featurefiles (*.mfcc)
Diese Files enthalten die Audiodaten in Form von Featurevektoren.

Labelfiles (*.rec)
Die Labelfiles beschreiben, an welcher zeitlichen Position in den Audiodaten welches Phonem vorkommt.

Targetfiles (*.targ)
Die Targetfiles enthalten die selbe Information wie die Labelfiles und dienen dem Netz als Zielvorgabe.

Lautdauer (single_state_duration_model_Lnn.txt)
Mittelwert und Varianz der Dauer für jedes Phonem.

Netzoutput (*.act)
Enthält die Wahrscheinlichkeit jedes Phonems für jedes Frame; Output des Netzes bei der Präsentation eines Featurefiles.

Phonemliste (*.lis)
Liste der, eventuell gemappten, Phoneme.

Preferences (spot_preferences, newalign_preferences)
Konfigurationsfile für den Wordspotter.

Wörterbuch (htk.dic)
Liste aller zu erkennenden Wörter zusammen mit ihrer Phonemschrift.

Matlabwordout (matlab_wordout)
Enthält Rangliste der Wörter, welche nach dem Score der Wörter sortiert ist. Entspricht dem Wordout File, ist jedoch in einem Format das für Matlab einfacher lesbar ist.

Mappingdaten (mapping.txt)
Information über die Abbildung eines Phonemsatzes auf einen neuen Phonemsatz.

Wordout (wordout)
Enthält Rangliste der Wörter, welche nach dem Score der Wörter sortiert ist.

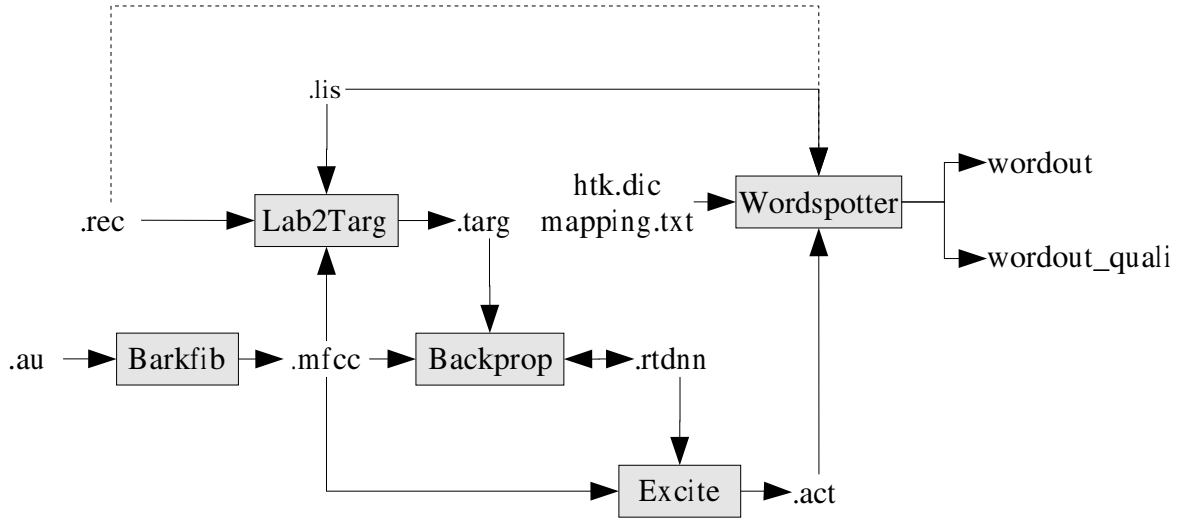
Wordoutquali (wordout_quali)
Enthält die Masse der Erkennungsleistung des Wordspotters.

Wordspottertestset (spotutts.txt)
Liste der vom Wordspotter zu verarbeitenden Netzoutput Files. In der zweiten Spalte steht das zum jeweiligen Netzoutput zugehörige Labelfile.

Trainset (trainutts.txt)
Liste der zu verarbeitenden Feature-, Label- oder Targetfiles für das Training des Netzes.

Testset (testutts.txt)
Liste der zu verarbeitenden Feature-, Label- oder Targetfiles für das Testen des Netzes.

Resultfile (result_net*.txt) Enthält die Bewertung eines Netzes, bewertet auch das NICO Tool CResult. Die Bewertung enthält unter anderem die Erkennungsrate jedes einzelne Phonems.



Vorhandene Files:		Erzeugte Files:	
Phonemliste	.lis	Feaurevektoren	.mfcc
Labeldaten	.rec	Targetdaten	.targ
Audiodaten	.au	Netz	.rtdnn
Mappingdaten	mapping.txt	Netoutput	.act
Dictionary	htk.dic	Wordscore	wordout
		Wordscorebewertung	wordout_quali

Figur 21: *Übersicht benötigter Files*

Anhang B

Shell Skripts

createtutts.sh

Ort	/
Aufruf	createtutts.sh

Erstellt die Trainset, Testset und Wordspottertestset Files. Dieses Skript wird kaum mehr gebraucht, da es im Prinzip nur einmal aufgerufen werden muss. Die von ihm erstellen Files befinden sich nun im Verzeichnis testbasis/, und werden von dort aus für jeden neuen Test kopiert (siehe createtest.sh).

Das Skript hat zwei Unschönheiten:

- Im selben Verzeichnis wie sich das Skript befindet muss auch das Verzeichnis testbasis/ und das Verzeichnis newdata/phnlab/ sein, in welchem sich die Labelfiles befinden.
- Beim Erstellen von newalignutts.txt und spotutts.txt werden feste Verzeichnisnamen in diese Files geschrieben. Dies, weil wir es nicht erreicht hatten, dass das awk Tool auf environment Variablen zugreifen kann. Dies ist jedoch nicht weiter tragisch, da beim Erstellen eines neuen Test mit createtest.sh diese Verzeichnisangaben automatisch korrekt abgeändert werden.

createtest.sh

Ort	/
Aufruf	createtest.sh testname [testbasis]
testname	Name des neuen Test.
testbasis	Verzeichnis, von dem benötigte vorgefertigten Files kopiert werden. Muss relativ zum aktuellen Pfad sein und muss mit / aufhören. Default: testbasis/

Erstellt einen neuen Test. Im aktuellen Verzeichnis wird ein neues Verzeichnis mit dem Namen *testname* erstellt. Alle für einen Test benötigten Files wie z.B. envvariables.sh, createnet.sh und trainnet.sh werden aus *testbasis* in das neue Verzeichnis kopiert. Abschnitt 4.2 gibt ein Anwendungsbeispiel.

createnet.sh

Ort	testbasis/
Aufruf	createnet.sh

Erstellt und trainiert eine Serie von Netzen. Nachdem ein Netz erstellt wurde, wird es automatisch auch trainnet.sh trainiert. Jede Testumgebung hat ihr eigenes createnet.sh Skript. Das Skript erwarte keine Parameter; stattdessen wird das Skript jeweils gleich

selbst abgeändert, um beliebige Netztopologien zu erstellen. Durch setzen der Variable `ITER_CNT` kann definiert werden, wieviele Netze erstellt und trainiert werden sollen. `i` bezeichnet die Nummer des aktuellen Netzes und bewegt sich im Bereich `1 - ITER_CNT`. Die erstellten Netze haben die Namen net$x$.rtdnn, wobei x für eine Zahl im Bereich `1 - ITER_CNT` steht. Zusätzlich werden Logfiles mit den Namen logfile$x$.txt erstellt, die Informationen über den Trainingsfortschritt enthalten. Siehe Abschnitt 4.3 für ein Anwendungsbeispiel.

trainnet.sh

Ort	testbasis/
Aufruf	trainnet.sh net log_file cresult_file
net	Das zu trainierende Netz.
log_file	Ein File mit diesem Namen wird erstellt, und der Trainingsfortschritt wird kontinuierlich darin rapportiert.
cresult_file	Ein File mit diesem Namen wird erstellt, und am Ende des Trainings wird der Output von CResult hineingeschrieben.

Trainiert ein bereits bestehendes Netz. Jede Testumgebung hat ihr eigenes trainnet.sh Skript. Das Skript erwartet keine Parameter; stattdessen wird das Skript jeweils gleich selbst abgeändert, um das Netz auf beliebige Weisen zu trainieren. Zu Beginn können einige Einstellungen gemacht werden, wie das Netz trainiert werden soll. Zum Beispiel ob pädagogisch trainiert werden soll, ob ein neues Alignment erstellt werden soll etc. Die genauen Einstellungsmöglichkeiten sind im Skript selbst gut beschrieben. Die Argumente für den zentralen BackProp Befehl werden direkt bei dessen Aufruf angegeben. Das Netz wird immer eine Epoche lang trainiert; danach wird das Tool BackPropCtrl.class aufgerufen, welches entscheidet, ob das Training fortgeführt wird oder abgebrochen werden soll. Damit kann Overfitting vermieden werden. Wie in createnet.sh, gibt die Laufvariable `i` die Nummer des aktuellen Netzes an. Siehe Abschnitt 4.3 für ein Anwendungsbeispiel.

Konfigurationsfiles

Preferences

Preferences Files steuern den den Wordspotter (SpotWords.class). Vorgefertigte Preferences Files sind spot_preferences und newalign_preferences, welche beide im Verzeichnis /testbasis zu finden sind. Es folgt eine Tabelle mit den Einstellungsmöglichkeiten.

Input	
`DICTIONARY_FILE_PATH`	Pfad zum Wörterbuch
`PHONEMS_FILE_PATH`	Pfad zur Phonemliste
`PHONEM_MAPPING_FILE_PATH`	Pfad zum Mappingdaten File
`PH_DUR_PROP_FILE_PATH`	Pfad zum Lautdauer File
`TEST_SET_FILE_PATH`	Pfad zum Wordspottertestset
Output	
`OUT_BASE_PATH`	Pfad, in den für jedes im Wordspottertestset angegebene Netzoutput File ein neues Verzeichnis erstellt wird, in welches dann der Output des Wordspotters (`WORDSCORE_FILE`, `MATLAB_WORDSCORE_FILE` und `GARBAGE_SCORE_FILE`) geschrieben wird. Das unter `QUALIFY_WORD_SCORE_FILE` angegebene File hingegen wird direkt in den hier angegeben Pfad geschrieben.
`WORDSCORE_FILE`	Name des zur erstellenden Wordscore File.
`MATLAB_WORDSCORE_FILE`	Name des zu erstellenden Matlabwordscore File.
`GARBAGE_SCORE_FILE`	Name des zu erstellenden Garbagescore File.
`QUALIFY_WORD_SCORE_FILE`	Name des zu erstellenden Wordoutquali File. Wird nur erstellt, falls `QUALIFY_WORDSPOTTER` auf `true` gesetzt ist.
Steuerung	
`WITH_DURATION_MODEL`	`true` falls Lautdauermodell verwendet werden soll, `false` sonst.
`QUALIFY_WORDSPOTTER`	`true` falls der Wordspotter zum Qualifying gebraucht werden soll, `false` sonst.
`MAKE_NEW_ALIGNEMENT`	`true` falls ein neues Alignment erstellt werden soll, `false` sonst. Bei `true` sollte `WITH_DURATION_MODEL` auf `false` gesetzt sein.

spot_preferences

Ort testbasis/

Vorgefertigtes Preferences File, um den Wordspotter zum Qualifying zu verwenden. Ein Beispiel dazu befindet sich in Abschnitt 4.7.

newalign_preferences

Ort testbasis/

Vorgefertigtes Preferences File, um den Wordspotter dazu zu verwenden, ein neues Alignment zu erstellen. Ein Beispiel dazu befindet sich in Abschnitt 4.6.

envvariables.sh

Ort testbasis/

Enthält Environment Variablen, die sowohl für createnet.sh wie auch für trainnet.sh Gültigkeit haben. Es sind dies mehrheitlich Pfadangaben für alle Files, die für das Erstellen und Trainieren eines Netzes benötigt werden. Für eine Beschreibung dieser Variablen sei der Leser auf die Kommentare im File selbst verwiesen.

Java Tools

PhonemExtracter.class

Ort	ch/ethz/ee/lnn/PhonemExtracter/
Aufruf	java PhonemExtracter context_frames mfcc_path label_path
context_frames	Anzahl Frames im einseitigen Kontextfenster.
mfcc_path	Verzeichnis mit den zu verarbeitenden Featurefiles (*.mfcc).
label_path	Verzeichnis mit den zu verarbeitenden Labelfiles (*.rec).

Das Skript extrahiert Phoneme aus allen Feature- und Labelfiles, die es in den angegebenen Verzeichnissen findet. In denselben Verzeichnissen wird für jedes Phonem ein Unterverzeichnis erstellt, in welches die neuen Feature- und Labelfiles geschrieben werden, die die extrahierten Phoneme enthalten. Siehe Abschnitt 4.4 für ein Anwendungsbeispiel.

CreateDataSet2.class

Ort	ch/ethz/ee/lnn/CreateDataSet/
Aufruf	java CreateDataSet2 size source_path dest_path [resultfile]
size	Anzahl Beispiele (Files) welche pro Phonem kopiert werden sollen. Für ein pädagogisches Set wird dieser Parameter als obere Grenze betrachtet.
source_path	Basispfad der Feature- und Targetfiles. Das Skript erwartet die Verzeichnisstruktur: source_path/mfcc und source_path/phntarg.
dest_path	Pfad der Testumgebung. Das Skript erwartet die Verzeichnisstruktur: dest_path/mfcc und dest_path/phntarg.
resultfile	Optionaler Parameter. Falls nicht gegeben, wird ein gleichverteiltes Set erstellt. Falls angegeben, bezeichnet er ein File mit dem Output von NICO's CResult, aufgrund dessen ein pädagogisches Set zusammengestellt wird.

Dieses Skript dient dem Erstellen von gleichverteilten oder pädagogischen Trainingssets. Es kopiert eine bestimmte Anzahl Feature (*.mfcc) - und Targetfiles (*.targ), die jeweils nur ein einzelnes Phonem enthalten, in das angegebene dest_path Verzeichnis. Dabei entscheidet das Vorhandensein des vierten Parameters (result), ob ein pädagogisches oder ein gleichverteiltes Set zusammenkopiert werden soll. Falls der Parameter weggelassen wird, wird ein gleichverteiltes Set erstellt, d.h. es werden von jedem Phonem gleich viele Files kopiert. Wird jedoch ein Resultfile angegeben, wird aufgrund der darin vermerkten Phonemerkennungsraten ein pädagogisches Set erstellt. Siehe Abschnitt 4.4 und 4.5 für Anwendungsbeispiele.

SpotWords.class

Ort	ch/ethz/ee/lnn/
Aufruf	java SpotWords [preferences]
preferences	Optionaler Parameter der den Pfad eines Preferences File angibt. Default: preferences

In Abschnitt 4.7 wird an einem Beispiel erklärt, wie SpotWords verwendet wird, um ein Netz zu qualifizieren. Abschnitt 4.6 erklärt an einem Beispiel, wie SpotWords verwendet wird, um ein neues Alignment zu erstellen. Anhang 8.3 beschreibt das anzugebende Preferences File.

NICO Tools

Diese Liste soll nur eine Idee vermitteln, was die Aufgabe der in diesem Bericht referenzierten NICO Tools ist. Eine ausführliche Beschreibung kann auf dem Internet nachgeschlagen werden [6].

Barkfib Extrahiert die Featurevektoren aus den Audiofiles

Lab2Targ Erstellt die Targetfiles für das Training des Netzes

CResult Beurteilt die phonemspezifische Erkennungsleistung eines Netzes auf einem Testset.

Backprop Trainiert ein Netz.

Excite Erstellt mit einem bereits trainierten Netz Netouput Files.