Lookahead mit Neuronalen Netzen in JANUS

Studienarbeit von Kathrin Paschen

28. März 1997

Inhaltsverzeichnis

Kapitel 2

Spracherkennung in JANUS

Ein Sprachsignal, das von JANUS verarbeitet werden soll, wird zunächst mit einer (innerhalb gewisser Grenzen) frei wählbaren Frequenz (meist 16KHz) abgetastet und danach in wenige Millisekunden lange Zeitabschnitte, die sogenannten *frames* zerlegt. Die so erhaltenen Parameter können mit verschiedenen Methoden vorverarbeitet werden, bevor die eigentliche Arbeit des Erkenners beginnt.

2.1 Dynamic Time Warping

Es liegt jetzt ein Eingabemuster mit $i = 1, ..., N$ *frames* vor, die jeder durch einen Vektor von Werten repräsentiert werden. Außerdem ist bekannt, daß das Sprachsignal aus Wörtern besteht, die im Wörterbuch von JANUS vorkommen, für die also Wortmodelle existieren. Demnach muß man jetzt herausfinden, welche Wörter in dem Sprachsignal vorkommen und in welcher Reihenfolge sie geäußert wurden. Dazu versucht man, einen sogenannten Weg (*path*) zu finden, der die *frames* des Eingabemusters und die Wortmodelle einander zuordnet. Dabei muß sowohl innerhalb der Eingabe als auch innerhalb der Wortmodelle die zeitliche Reihenfolge berücksichtigt werden, und der Pfad muß Eingabemuster und Wortmodelle jeweils lückenlos überdecken. Zu diesem Zweck geht JANUS die *frames* in ihrer zeitlichen Reihenfolge durch, stellt in jedem Schritt Hypothesen auf und bewertet diese, weist ihnen also eine Bewertung (*score*) zu.

2.2 Hidden Markov Modelle

Die Modellierung von Worten und Phonemen geschieht in JANUS mit Hidden Markov Modellen (HMMs). Solche Modelle werden zur statistischen Modellierung von Signalen eingesetzt; es handelt sich um Erweiterungen der Diskreten Markov–Modelle. Ein Diskretes Markov–Modell läßt sich als nicht–deterministischer endlicher Automat auffassen, dessen Zustandsübergänge mit Wahrscheinlichkeiten versehen sind. Bei Markov–Prozessen werden normalerweise nur der aktuelle und der direkt vorhergehende Zustand in die probabilistische Betrachtung einbezogen. Außerdem werden die Wahrscheinlichkeiten als konstant in Bezug auf die Zeit betrachtet. Solche stochastischen Prozesse bezeichnet man auch als beobachtbar, da man zu jedem Zeitpunkt die Ausgabe (den aktuellen Zustand, in dem sich der Automat befindet, und der mit einem beobachtbaren Ereignis gleichgesetzt wird) feststellen kann. Im allgemeinen braucht aber die Beobachtung eines Ereignisses nicht korrekt zu sein — sie ist vielmehr eine Wahrscheinlichkeitsfunktion des Ereignisses. Man hat es hier also eigentlich mit zwei ineinander verschachtelten stochastischen Prozessen zu tun, von denen nur einer beobachtbar ist; daher kommt der Name *Hidden Markov Model*.

Es gibt drei Aufgabenstellungen, die im Zusammenhang mit Hidden Markov Modellen wichtig sind:

1. Berechne die Wahrscheinlichkeit einer Ausgabesequenz für ein bestimmtes Modell.

2. Welche Folge von Zuständen eines gegebenen Modells kann eine bestimmte Ausgabesequenz produziert haben?

3. Wie lassen sich die Wahrscheinlichkeiten für Zustandsübergänge und Beobachtungen so anpassen, daß die Wahrscheinlichkeit für eine bestimmte Ausgabesequenz möglichst groß wird?

Das zweite Problem stellt sich zum Beispiel dann, wenn man eine Hypothese aufstellen will, um einen Pfad weiterzuführen. Der Versuch, diese Hypothese zu bewerten, führt dagegen auf das erste Problem. Das dritte Problem gehört eher zum Bereich des Trainings und ist für diese Arbeit weniger interessant.

2.3 Suche und Hypothesenbewertung

JANUS erkennt Wörter aus einem Vokabular von etwa 20.000. Jedes Wort ist durch Phoneme repräsentiert, jedes Phonem wiederum durch ein HMM. Der Erkenner kann für jedes Phonem und jeden *frame* bewerten, wie gut der *frame* zu dem entsprechenden HMM paßt.
Für die Suche nach einem DTW-Pfad kann man sich Hypothesen in ein Koordinatensystem eingetragen denken, auf dessen x–Achse die Zeit und auf dessen y–Achse die möglichen Wörter mit ihren Phonemen und den Zuständen der

entsprechenden HMMs stehen. Eine Zelle in diesem Koordinatensystem enthält immer zwei Einträge, nämlich eine lokale und eine globale Bewertung für:

`Zur Zeit t wurde der X-te Zustand von Phonem Y in Wort Z durchlaufen.`

Dabei berücksichtigt die lokale Bewertung nur die aktuelle Zelle, während für die globale Bewertung auch die Summe der Bewertungen der Zellen entlang des Pfades herangezogen werden, der zur aktuellen Zelle führt.
Hat der Erkenner die Hypothesen für den Zeitpunkt t bewertet, dann müssen Hypothesen für den Zeitpunkt $t + 1$ aufgestellt werden. Dabei kommen Zustandsübergänge innerhalb eines Phonems in Frage oder solche Phoneme, die eine der aktivierten Worthypothesen fortsetzen oder mit denen ein neues Wort beginnen könnte.
Für die aktivierten Phoneme müssen dann Bewertungen berechnet werden, um daraus Bewertungen für die Worthypothesen zu erhalten. Die Menge der zu untersuchenden Hypothesen wird mit einem *Viterbi beam search* eingeschränkt. Das bedeutet, daß nur diejenigen Hypothesen weiterverfolgt werden, deren Bewertung von der besten um höchstens die *Strahlbreite* abweicht.

2.4 Der Einsatzort der neuronalen Netze

Die Hypothesenbewertung ist relativ langsam, weil häufig sehr viele Phoneme in Frage kommen und die entsprechenden Phonem- und Wortmodelle bearbeitet werden müssen. Die Anzahl Phoneme ist zwar normalerweise niedrig, die HMMs verwenden aber kontextabhängige Modelle, und die meisten Phoneme können in vielen verschiedenen Wörtern vorkommen.
Ein neuronales Netz, das in der Lage wäre, a–posteriori–Wahrscheinlichkeiten für das Vorliegen eines bestimmten Phonems anzugeben, könnte die Aufstellung und Bewertung von Hypothesen unterstützen. Es könnte diejenigen Phoneme von der Bewertung ausschließen, deren Vorhandensein es als sehr unwahrscheinlich einstuft und für die verbleibenden eine Bewertung abgeben. Das könnte die Anzahl zu evaluierender HMMs sehr verringern.
Außerdem wäre es interessant, wenn das Netz lernen könnte, zuverlässig diejenigen *frames* zu erkennen, in denen ein Phonem beginnt. Da dies nur in etwa jedem sechsten oder siebten *frame* der Fall ist, müßte man dann auch nur an den Stellen, wo tatsächlich ein Phonem anfängt, Hypothesen aufstellen und bewerten.

Die neuronalen Netze können darauf trainiert werden, zu erkennen,

1. ob in einem bestimmten *frame* überhaupt ein Phonem beginnt,
2. welches Phonem dort mit welcher Wahrscheinlichkeit beginnt
3. welche Phonemklasse aus einer vorher festgelegten Menge dort mit welcher Wahrscheinlichkeit beginnt

Die Netze sollen in der Ausführung schnell sein, sonst bringt der Lookahead keinen Geschwindigkeitsvorteil. Erstrebenswert wäre eine Geschwindigkeit deutlich über 100 *frames* pro Sekunde, denn falls man von einer Echtzeitverarbeitung ausgeht, kommen bei der gegenwärtigen Konfiguration 100 *frames* pro Sekunde an.
Gleichzeitig sollen die Netze zuverlässig sein — wenn sie zu oft eine falsche Hypothese unterstützen, ist der Lookahead zwecklos.

Kapitel 3

Neuronale Netze

Bei den Netzen, die ich im Rahmen dieser Studienarbeit untersucht habe, handelt es sich ausschließlich um feed-forward-Netze. Eine Untersuchung alternativer Architekturen war aus Zeitgründen nicht möglich. Daher beschränkt sich auch dieser Abschnitt auf 'klassische' feed-forward-Netze mit Backpropagation als Trainingsverfahren. Zwar existiert für solche Netze kein schnelles und zuverlässiges Trainingsverfahren, dafür sind sie aber in der Ausführung schneller als die meisten anderen Architekturen, und darauf kommt es in der beabsichtigten Anwendung an.

3.1 Allgemeines

Neuronale Netze wurden ursprünglich als vereinfachte Modelle der Informationsverarbeitung im Gehirn entwickelt. Das einfachste Netz besteht aus nur einem Neuron, dessen Verhalten durch zwei Parameter bestimmt wird:

1. Einen Vektor von Gewichten
2. Einen Schwellwert (bias)

Erhält jetzt das Neuron einen Eingabevektor, so bildet es das Skalarprodukt mit dem Gewichtsvektor und zieht davon den Schwellwert ab. Das Ergebnis wird — eventuell nach Anwendung einer Aktivierungsfunktion — als die Aktivierung des Neurons betrachtet.
Neuronale Netze werden häufig (wie auch hier) zur Klassifikation von Eingaben eingesetzt. Man versucht also, zwischen verschiedenen Klassen von Mustern eine Klassengrenze zu finden; stellt man die Muster als Vektoren in einem n–dimensionalen Vektorraum dar, so ist eine solche Klassengrenze eine Hyperebene. Allgemein bezeichnet man Musterklassen, die sich durch eine Hyperebene voneinander trennen lassen, als *linear separierbar*.
Insbesondere kann ein einzelnes Neuron eine Hyperebene im zweidimensionalen

Raum (mit anderen Worten, eine Gerade) als Trennebene modellieren.
Für nicht zusammenhängende Klassen ist es häufig nötig, mehrere Hyperebenen zur Trennung einzusetzen. So kommt man zur Definition von mehrschichtigen neuronalen Netzen oder *multilayer perceptrons*. In vollständig verbundenen feed-forward-Netzen, wie ich sie verwende, erhalten alle Neuronen einer Schicht die gleichen Eingaben und reichen ihre Ausgaben an die Neuronen der folgenden Schicht weiter.
Ich bezeichne hier ein Netz mit einer verborgenen Schicht als zweischichtig. Solange jedes Neuron seine Aktivierung linear aus der Summe der Eingaben und dem Schwellwert berechnet, kann das gesamte Netz nur lineare Funktionen berechnen. Insbesondere läßt sich dieses Netz dann durch ein einschichtiges ersetzen. Daher wendet man zusätzlich eine nichtlineare Funktion f auf die Ausgaben der Neuronen an. Diese Funktion sollte monoton sein, nichtlinear und — für den Lernalgorithmus — differenzierbar. Außerdem muß gelten:

$$\lim_{x \to \infty} f(x) = 1, \lim_{x \to -\infty} f(x) = 0.$$

Es kommen eine Vielzahl von Funktionen in Frage; die sogenannte logistische Funktion

$$f(x) = \frac{1}{(1 + e^{-x})}$$

wird gerne verwendet, weil ihre Ableitung

$$f'(x) = f(x) \cdot (1 - f(x))$$

besonders leicht zu berechnen ist.
Allgemein gilt, daß man mit einem zweischichtigen Netz, das zumindest zwischen der ersten und zweiten Schicht eine nichtlineare Aktivierungsfunktion verwendet, entweder eine vorgebene Menge von diskreten Funktionswerten direkt darstellen oder in einem Intervall eine stetige Funktion beliebig approximieren kann ([2]). Die Funktion braucht nicht unbedingt stetig zu sein, sie ist schon dann lernbar, wenn sie einen kompakten Träger hat oder auch nur 'hinreichend gutartig' ist. Es ist allerdings kein Verfahren bekannt, mit dem die optimalen Parameter für ein neuronales Netz in vertretbarer Zeit gefunden werden können; das Lernproblem für neuronale Netze ist NP-vollständig ([5]).
Bezeichnet man jetzt die Eingabe mit $x = (x_1, x_2, ..., x_n)$, die Gewichte zwischen der Eingabe und der ersten Schicht mit $w = (w_{11}, w_{12}, ..., w_{mn})$ und die Aktivierungsfunktion mit f, so erhält man als Ausgabe des $i$ten Neurons der ersten Schicht:

$$f(\sum_k x_k \cdot w_{ik}).$$

Bezeichnet man die Gewichte zwischen der ersten und der zweiten Schicht mit $v = (v_{11}, v_{12}, ..., v_{mp})$, so erhält man als Ausgabe des $j$ten Ausgabeneurons:

$$f(\sum_i v_{ji} \cdot f(\sum_k x_k \cdot w_{ik}))$$

Der Schwellwert taucht in diesen Formeln nicht explizit auf; er wird meist über einen zusätzlichen Eingang für jedes Neuron realisiert, dessen Gewicht gerade der Schwellwert ist und dessen Eingabe immer auf -1 gesetzt wird.

3.1.1 Training mit 'Backpropagation of Errors'

Beim Training eines Netzes versucht man, die Gewichte so zu verändern, daß das Netz die gewünschte Ausgabe berechnet. Zu diesem Zweck sucht man mittels Gradientenabstieg ein Minimum der Fehlerfunktion oder *objective function*, wobei die Fehlerfunktion die Abweichung der tatsächlichen von der Sollausgabe angibt. Häufig wird der sogenannte *mean square error* verwendet — die gemittelte Summe der quadratischen Fehler der Ausgabeneuronen. Weiter unten werde ich darauf eingehen, wann es sinnvoll sein kann, eine andere Fehlerfunktion zu verwenden. Sei $E(w)$ der Fehler in Abhängigkeit von den Gewichten w, also die Abweichung von tatsächlicher und Sollausgabe in Abhängigkeit von den Gewichten. $E(w)$ ist zu minimieren; es sei bekannt, daß ein lokales Minimum in der Nähe von w existiert. Bekanntlich zeigt der Gradient

$$\nabla_w E(w) := (\frac{\partial}{\partial_{w1}}, \frac{\partial}{\partial_{w2}}, ..., \frac{\partial}{\partial_{wn}})$$

in der Nähe des Minimums in die Richtung des stärksten Anstiegs von $E(w)$; daher sollte die Gewichtskorrektur proportional zum negativen Gradienten sein. Tatsächlich besagt das Perzeptron-Konvergenztheorem, daß mit der wiederholten Gewichtskorrektur

$$w_t := w_{t-1} - \nabla_w E(w)$$

ein Perzeptron die Trennung linear separierbarer Mengen lernen kann. Häufig wird diese Korrektur noch mit einer 'Schrittweite' γ multipliziert, die mit der Zeit und bei steigendem Fehler kleiner wird und bei sinkendem Fehler leicht vergrößert wird.
Ist jetzt auf die Ausgabe eine Sigmoid-Funktion angewandt worden, so hängt die Aktivierung der Neuronen nicht mehr linear von der Eingabe ab, die Ableitung von $E(w)$ muß also entsprechend anders berechnet werden. Sei dazu z_j die Aktivierung des $j$ten Ausgabeneurons vor Anwendung der Sigmoidfunktion f, und $y_j := f(z_j)$. Für die Änderung des Gewichtes w_{ij} braucht man

$$\frac{\partial E_x}{\partial w_{ij}} = (\frac{\partial E_x}{\partial z_i}) \cdot (\frac{\partial z_i}{\partial w_{ij}}).$$

Mit

$$\partial_i := \frac{\partial E_x}{\partial z_i} = \frac{\partial E_x}{\partial y_i} \cdot \frac{\partial y_i}{\partial z_i} = \frac{\partial E_x}{\partial y_i} \cdot \frac{\partial f(z_i)}{\partial z_i} = \frac{\partial E_x}{\partial y_i} \cdot f'(z_i)$$

und

$$\frac{\partial z_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \cdot \sum_k w_{ik} \cdot x_k = x_j$$

erhält man als Gewichtsänderung

$$w_{ij}(t) := w_{ij}(t-1) + \gamma \cdot \partial_i \cdot x_j.$$

Für die Neuronen der Ausgabeschicht kann man die Fehlerfunktion direkt an Ausgabe und Zielausgabe ablesen; Für die Neuronen der anderen Schichten gilt eine kompliziertere Fehlerfunktion. Sei dazu n ein Neuron aus der Schicht vor der Ausgabeschicht, das mit den Neuronen m_i der Ausgabeschicht verbunden ist. Das Fehlersignal für n muß dann berücksichtigen, wie stark n zum Fehler von m_i beigetragen hat, man multipliziert also den Fehler von m_i mit dem Gewicht der Verbindung von n zu m_i und summiert über die Ausgabeneuronen. Dieses Verfahren läßt sich für die weiteren Schichten fortsetzen, der Rest der Formel für die Gewichtsänderung ist für alle Schichten gleich.

3.1.2 Konvergenzbedingungen und –geschwindigkeit

Die Bedingungen, unter denen ein Netz mit dem eben beschriebenen Trainingsverfahren konvergiert, sind weitgehend ungeklärt. Ein Netz kann aus mehreren Gründen nicht oder gegen einen 'falschen' Grenzwert konvergieren. Ist beispielsweise die Schrittweite zu groß oder wiederholen sich die Trainingsmuster zu regelmäßig, so können die Gewichte zu oszillieren anfangen. Daher paßt man gern die Schrittweite dynamisch an und verwendet einen sogenannten *momentum term*: in jeder Iteration geht in die Gewichtskorrektur auch die Korrektur aus der vorangegangenen Iteration, multipliziert mit einem Faktor, dem *momentum term*, ein.

Beim Training soll das Netz lernen, relevante Eigenschaften von Mustern zu erkennen. Dies hat nur begrenzte Aussicht auf Erfolg, falls das Netz über zu wenige Neuronen in der verborgenen Schicht verfügt, um die wichtigen Eigenschaften zu kodieren; bei einer zu großen Zahl von Neuronen besteht dagegen die Gefahr, daß das Netz Muster 'auswendiglernt', wodurch die Erkennungsleistung auf nicht–trainierten Mengen sinkt. Daher beobachtet man häufig das Verhalten des Netzes auf einer sogenannen Kreuz–Validierungs–Menge, mit der nicht trainiert wird und bricht das Training ab, sobald die Leistung auf dieser Menge abzusinken beginnt.

Schwieriger zu lösen ist das Problem der lokalen Minima. An sich kann ein Netz sehr viele gleich gute lokale Minima haben, da die Ausgabe des Netzes häufig gegenüber Operationen wie dem Vertauschen von Neuronen einer Schicht invariant ist. Es gibt aber auch 'suboptimale' Minima, in denen das Netz 'hängenbleiben'

kann. Außerdem kommt es gerade bei kleinen Gradienten leicht vor, daß durch numerische Ungenauigkeit die Gewichtskorrektur in eine falsche Richtung vorgenommen wird, oder daß der Gradient zu verschwinden scheint. Es gibt mehrere Möglichkeiten, lokalen Minima zu entkommen, die allerdings meist recht aufwendig sind und deren Beschreibung den Rahmen dieser Arbeit sprengen würde. Sie sind zum Beispiel in [3] beschrieben.

3.2 Implementierung

Für die Arbeit mit den neuronalen Netzen habe ich ein Modul *nnet.c* für JANUS programmiert. Ein neuronales Netz besteht dort aus einer Liste von Gewichtsmatrizen und einer Reihe von zusätzlichen Parametern, die beim Erzeugen des Netzes angegeben werden können. Dazu gehören:

1. Der Schwellwert oder Bias
2. Die Schrittweite
3. Der Impuls oder Momentum Term

Ein neuronales Netz kann seine Parameter und Gewichtsmatrizen aus einer Datei einlesen, die Gewichte können aber auch mit Zufallswerten initialisiert werden. Verschiedene Übergabeparameter der Auswertungs- und Trainingsfunktionen steuern das Verhalten des neuronalen Netzes. Im einzelnen kann man angeben:

1. Soll die Sigmoidfunktion angewandt werden?
2. Soll die Sigmoidfunktion auch auf die Ausgabe der letzten Schicht angewandt werden?
3. Sollen Ausgaben auf den Bereich $[-1, 1]$ normalisiert werden?
4. Soll die Schrittweite angepasst werden?
5. Soll *Classification Figure of Merit* anstelle von *Mean Square Error* zur Berechnung des Fehlers eingesetzt werden?
6. Soll die Sigmoidfunktion interpoliert werden?
7. Sollen Fehler mitprotokolliert werden?

Außerdem kann man der *update*-Funktion als zusätzlichen Parameter die Blockgröße, also die Anzahl Trainingsbeispiele, für die Gewichte auf einmal angepaßt werden sollen, angeben.

Kapitel 4

Training

4.1 Konfiguration des neuronalen Netzes

Um ein neuronales Netz als Phonemerkenner zu trainieren, ist es nötig, einige Parameter festzulegen.

4.1.1 Interpretation der Ausgabe

Die Netze sollen nach den 1–aus–N–Prinzip arbeiten, das heißt, in der Ausgabe soll jeweils ein Neuron eine Alternative kodieren. Ich betrachte dabei ein Neuron als aktiv, falls seine Ausgabe über einem als Parameter anzugebenden Schwellwert liegt. Prinzipiell wäre es auch möglich gewesen, die Ausgaben mit weniger Neuronen zu kodieren; zum Beispiel könnten Kombinationen von aktivierten Neuronen jeweils ein Phonem kodieren, oder man könnte die Stärke der jeweiligen Aktivierung in die Auswertung einbeziehen. Eine 1–aus–N–Bewertung, bei der jeweils ein Neuron ein Phonem kodiert, bietet aber die folgenden Vorteile:

1. Die Ausgabe des Netzes ist intuitiver; es fällt leichter, Ähnlichkeiten zwischen Klassen zu erkennen.
2. Es wird leichter, statt nur dem am stärksten aktivierten Neuron die k am stärksten aktivierten Neuronen als Ausgabe zu betrachten, so daß das Netz dann nicht mehr das am wahrscheinlichsten vorliegende Phonem anzeigt, sondern mehrere Alternativen nennt.
3. Aus dem Vergleich der am stärksten aktivierten Neuronen kann man Aussagen über die Zuverlässigkeit der Ausgabe ableiten.
4. Eine Kodierung über die Höhe der Aktivierung impliziert eine Ordnungsrelation zwischen Phonemen beziehungsweise Phonemklassen, die nicht besteht.

Es sieht zwar auf den ersten Blick so aus, als würde es sich lohnen, die Anzahl der Ausgabeneuronen zu verringern, um so die Trainings- und Auswertungsgeschwindigkeit des Netzes zu steigern; andererseits ist die Parameteranpassung um so schwieriger, je kompliziertere Abhängigkeiten zwischen den Parametern bestehen. Ich fand es daher zur Reduktion der Anzahl Neuronen sinnvoller, die Phoneme in Klassen einzuteilen, so daß jedes Ausgabeneuron nicht für ein Phonem steht, sondern für eine Klasse. Im einfachsten Fall habe ich nur die Klassen *Konsonant*, *Vokal* und *Nicht-Phonemisch* trainiert, später auch kompliziertere Einteilungen — genauere Beschreibungen finden sich im Anhang unter *classes2* auf Seite 40 und *classes3* auf Seite 41.
Ein Netz mit 68 Ausgabeneuronen liefert natürlich im Idealfall ein Maximum an Information, allerdings ist es erheblich schwieriger zu trainieren und benötigt wesentlich mehr Neuronen in der verborgenen Schicht als ein Netz, das beispielsweise nur 20 verschiedene Ausgaben hat, denn je mehr Unterscheidungen man trainieren will, desto mehr Eigenschaften muß das Netz zu erkennen lernen. Die Anzahl benötigter Neuronen schlägt sich unmittelbar in Trainings- und Auswertungsgeschwindigkeit nieder — vor allem bei der Matrixmultiplikation mit kubischem Aufwand. Es bleibt im Einzelfall abzuwägen, ob der Informationsgewinn das größere Netz rechtfertigt.

4.1.2 Auswahl der Aktivierungsfunktion

In erster Linie aus Zeitgründen habe ich mich bei der Implementierung der Aktivierungsfunktion auf die logistische Funktion $1/1 + e^{-x}$ beschränkt; laut [3] hat die Wahl der Aktivierungsfunktion meist keinen großen Einfluß auf die Leistungsfähigkeit eines neuronalen Netzes, solange die Funktion ihre Eingabe monoton und differenzierbar auf das Intervall $(0, 1)$ abbildet. Die Trainingsgeschwindigkeit kann sich allerdings sehr wohl ändern.

4.1.3 Auswahl der Blockgröße

Ebenfalls wichtig für das Training ist die Auswahl der Blockgröße, also der Anzahl Muster, für die Gewichte auf einmal angepaßt werden sollen. Ist sie zu klein gewählt, oszilliert das Netz, da die Gewichtsänderung in einem Schritt den Fehler im nächsten beinahe immer vergrößert. Ist sie dagegen zu groß, so kommt es vor, daß sich die Korrekturen für die einzelnen Muster gegenseitig so weit auslöschen, daß das Netz — wenn überhaupt — nur noch sehr langsam konvergiert. Ein besonderes Problem entstand beim Training dadurch, daß bestimmte Phoneme beziehungsweise Phonemklassen in einzelnen Blöcken gehäuft auftraten und das Netz dann nur genau diese Phoneme zu erkennen lernte. Um dem zu entgehen, kann man einzelne *frames* aus der Trainingsmenge entfernen oder sie zufällig mischen.

4.1.4 Auswahl der Fehlerfunktion

Ein Problem entsteht beim Aufbau der Trainingsmenge dadurch, daß im Durchschnitt nur alle sechs bis sieben *frames* ein Phonem beginnt. Würde ich also die *frames* so, wie sie sind, als Eingabe für das Training benutzen, dann würde das neuronale Netz lernen, daß die Ausgabe *Kein Phonemanfang* in gut 80% aller Fälle korrekt ist und seine Gewichte so einstellen, daß bei jeder Eingabe *Kein Phonemanfang* herauskäme. Dadurch würde die Fehlerquote zwar sehr klein, aber das Netz unbrauchbar.
Daher verwende ich zum Training nur jeweils die *frames*, in denen ein Phonem anfängt und jeweils einen *frame* aus dem Bereich zwischen zwei Phonemanfängen.
Ein weiteres Problem ergibt sich daraus, daß in der Zielausgabe nur jeweils das Ausgabeneuron für eine Phonemklasse aktiviert sein soll, für alle anderen ist das Ziel 0.0. Bei Verwendung einer Fehlerfunktion wie des *Mean Square Error* besteht die Gefahr, daß das Netz lernt, den Fehler gering zu halten, indem es an allen Ausgabeneuronen 0.0 ausgibt. Aus diesem Grund empfiehlt es sich speziell bei Netzen mit relativ vielen Ausgabeneuronen, statt des *Mean Square Errors* als Fehlerfunktion *Classification Figure of Merit* zu berechnen, also

$$1- \text{ Aktivierung des Neurons, das hätte feuern sollen} \\ + \text{ größte tatsächliche Aktivierung.}$$

4.2 Vorgehensweise beim Training

Für das Training benötigt man Matrizen mit der Eingabe und der Sollausgabe, sowie gegebenenfalls ein zusätzliches Matrizenpaar für die Validierung. Ich habe die Matrizen jeweils vorher mit einem separaten Skript erzeugt und in Dateien abgespeichert, die dann beim Training eingelesen wurden.
So erhält man eine Matrix, deren Zeilen mit 16 Melscale-Koeffizienten jeweils einen *frame* beschreiben. Man kann an sich nicht erwarten daß ein neuronales Netz lernt, anhand nur eines einzigen *frames* zu entscheiden, welches Phonem dort beginnt, denn Phoneme unterscheiden sich nicht an jeder Stelle deutlich voneinander. Und außerdem ist nicht sicher, daß die *frames* in der Trainingsmenge so gleichmäßig Phonemen zugeordnet sind, daß in einem einzelnen *frame* immer die relevanten Merkmale eines Phonems enthalten sind. Daher habe ich beim Training jeweils Fenster von zuerst 3, später 5 *frames* betrachtet. Das Netz erhält also 48 beziehungsweise 80 Eingaben, die (vor derNormalisierung) zwischen 0 und 10 liegen. Die Art von neuronalem Netz, die ich verwende, braucht Eingaben im Bereich $[-1, 1]$. Das hängt damit zusammen, daß die Ausgaben in $[0, 1]$ liegen sollen. Wären jetzt die Eingaben betragsmäßig groß, dann müsten die Gewichte sehr klein werden, wodurch Rundungsfehler gefährlicher werden. Außerdem könnte es dann leicht vorkommen, daß die Ausgaben sehr groß oder

sehr klein werden; dann kann das Netz nur noch sehr langsam lernen, weil die Sigmoidfunktion für sehr große und sehr kleine Werte immer flacher wird.
Deshalb bilde ich die Eingaben mit $x \mapsto \frac{x}{5} - 1$ auf das Intervall $[0, 1]$ ab.
Die so vorbereiteten Eingaben werden über eine verborgene Schicht an die Ausgabeneuronen weitergeleitet. Die Ausgabeschicht wird dann nach dem 1–aus–N–Prinzip ausgewertet: ich betrachte ein Neuron als aktiv, falls seine Ausgabe über einem als Parameter anzugebenden Schwellwert liegt. Sind mehrere Neuronen so stark aktiviert, so bestimmt das am stärksten aktivierte die Ausgabe des Netzes. Zeigt kein Neuron eine Aktivierung in der erforderlichen Höhe, so wird dies als das Ergebnis *Kein Phonemanfang* interpretiert. Alternativ ist auch eine k–aus–N–Auswertung möglich, bei der dann die k oberhalb des Schwellwertes am höchsten aktivierten Neuronen als Kandidaten für die Ausgabe betrachtet werden.
Für die Zielausgabe lasse ich in JANUS einen *path* durch *forced alignment* berechnen und verwende die dort angegebenen Phoneme mit den Indizes der *frames*, wo sie laut JANUS beginnen und enden.

Kapitel 5

Ergebnisse

Ich werde jetzt beispielhaft einige Lernkurven vorstellen und deuten. Ich habe Netze für drei verschiedene Aufgaben trainiert: einmal sollten nur in drei Klassen (Vokale, Konsonanten und nicht–phonemische Laute) eingeteilte Phonemanfänge voneinander und von nicht–Phonemanfängen unterschieden werden, einmal sollten 20 Klassen von Phonemen erkannt werden und einmal alle 68 in der Trainingsmenge vorgesehenen Phoneme.
In Abschnitt 5.1 beschreibe ich sprecherabhängiges Training, in Abschnitt 5.2 sprecherunabhängiges.

5.1 Sprecherabhängiges Training

In den ersten Trainingsversuchen wurde das Netz nur mit jeweils einer Äußerung trainiert. Ich zeige hier die Ergebnisse für eine Trainingsmenge mit 338 und eine Validierungsmenge mit 38 *frames*.

5.1.1 Phonemanfänge erkennen

Im ersten Versuch ging es um die Erkennung von Phonemanfängen. Das Netz hatte drei Ausgabeneuronen; alle drei sollten bei nicht–Phonemanfängen niedrig aktiviert sein und andernfalls den Beginn eines Konsontants, Vokals oder nicht–phonemischen Lautes anzeigen. Ein Netz mit 20 Neuronen in der verborgenen Schicht konvergierte sowohl auf der Trainings– als auch auf der Validierungsmenge. Das folgende Diagramm zeigt den Anteil richtig erkannter Muster nach jedem der 20 Trainingsschritte.

Die Entwicklung des Fehlers auf der Trainingsmenge:

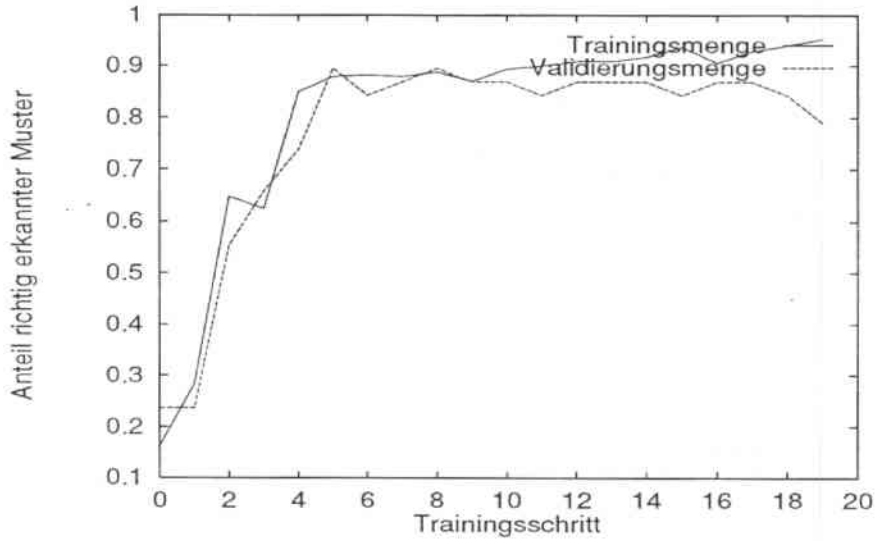

Abbildung 5.1: Anteil richtig erkannter Muster nach jedem Trainingsschritt

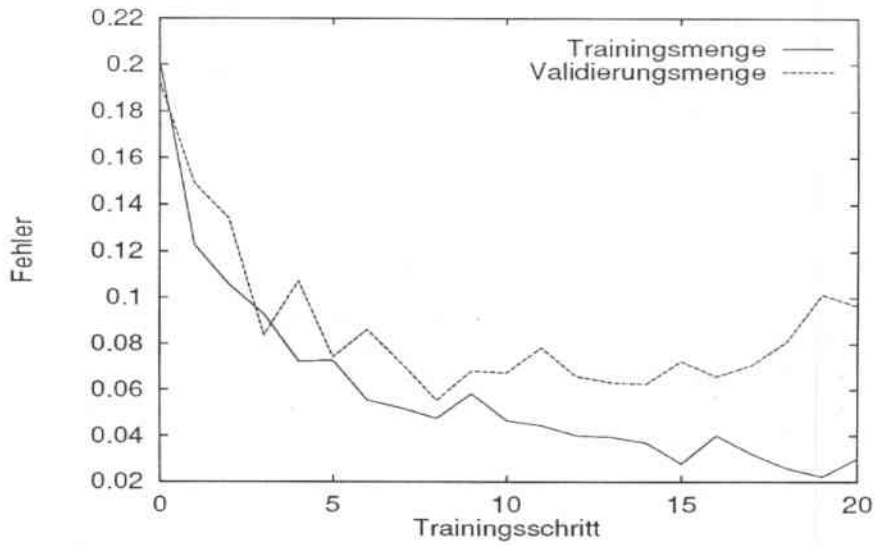

Abbildung 5.2: Entwicklung des Fehlers (CFM) auf einer Trainingsmenge von 338 Mustern bei 3 Ausgabe- und 20 verborgenen Neuronen

Auch bei 100 Neuronen in der verborgenen Schicht konvergiert das Netz nicht schneller, es erreicht (auch bei längerem Training) keine höheren Maximalwerte. Andererseits sind auch keine Overfitting–Effekte zu beobachten: Die Leistung auf der Validierungsmenge bleibt prozentual im Bereich derer auf der Trainingsmenge.
An folgender Tabelle kann man sehen, welche Fehler das Netz im letzten Durchgang auf der Trainingsmenge machte. Der Muster sind zeilenweise, die Ausgabe des Netzes spaltenweise eingetragen.

Sollausgabe	Kein Phonemanfang	Nicht–phonemisch	Vokal	Konsonant
Kein Phonemanfang	169	0	0	0
Nicht–Phonemisch	0	8	0	6
Vokal	0	0	49	6
Konsonant	0	11	17	86

Wie man sieht, erkennt das Netz alle 169 Nicht-Phonemanfänge eindeutig als solche, und es gibt auch nie fälschlicherweise *Kein Phonemanfang* aus. Das wird dadurch erleichtert, daß sowohl in der Trainings– als auch in der Validierungsmenge nur solche nicht-Phonemanfänge vorhanden waren, die aus der Mitte zwischen zwei Phonemanfängen stammten — das heißt, es waren Muster, die wohl besonders leicht zu erkennen sind.

Daher habe ich in einem der folgenden Versuche Validierungsmengen verwendet, die beliebige *frames* enthielten. Dabei zeigte sich, daß es nur auf wenigen Trainingsmengen möglich war, nicht–Phonemanfänge zuverlässig zu erkennen; zumeist wurden nur diejenigen nicht–Phonemanfänge als solche erkannt, die aus der Mitte zwischen zwei Phonemanfängen stammten. Das ist insofern nachvollziehbar, als ein Phonemanfang ein Übergang ist, der sich nicht an einem einzelnen *frame* erkennen läßt, sondern am Vergleich mehrerer. Da ich aber als Eingabe für das Netz jeweils eine Folge von fünf *frames* betrachte, kann zum Beispiel im vierten oder fünften ein Phonemanfang sein, obwohl im dritten (der für die Zielausgabe entscheidend ist) keiner vorliegt. Dazu paßt, daß das Netz fast nie fälschlicherweise *Kein Phonemanfang* ausgab; die folgende Tabelle zeigt die Ausgaben (zeilenweise) und Sollausgaben (spaltenweise) für eine Validierungsmenge mit 831 Mustern, von denen 73 Phonemanfänge waren. Die Trainingsmenge enthielt 861 Muster, von denen die Hälfte Phonemanfänge waren.

Sollausgabe	Kein Phonemanfang	Nicht–phonemisch	Vokal	Konsonant
Kein Phonemanfang	283	1	9	4
Nicht–Phonemisch	111	9	–	1
Vokal	116	1	13	4
Konsonant	247	3	8	20

Diese Ergebnisse besserten sich nicht, wenn in der Trainingsmenge mehr nicht–Phonemanfänge vertreten waren. Sie verschlechterten sich aber auch nicht, wenn das Netz mit 20 anstelle von drei Ausgaben trainiert wurde — das Training dauerte dann allerdings länger; mit drei Ausgaben ließ sich eine Trefferquote von 80 bis 90% auf der Trainingsmenge mit einem Netz mit 30 verborgenen Neuronen in 300 Trainingsschritten erreichen; bei 20 Ausgaben brauchte ein Netz mit 50 verborgenen Neuronen 500 Schritte.
Damit kann ein neuronales Netz Phonemanfänge zwar nicht mit absoluter Sicherheit erkennen, es kann aber etwa ein Drittel der nicht–Phonemanfänge als solche zu erkennen lernen und es gibt fast nie fälschlicherweise *Kein Phonemanfang* aus.
Die folgenden Bilder zeigen die Zielaktivierung und Ausgabe für die ersten 150 *frames* einer Validierungsmenge mit 1422 Einträgen, die alle *frames* enthielt.

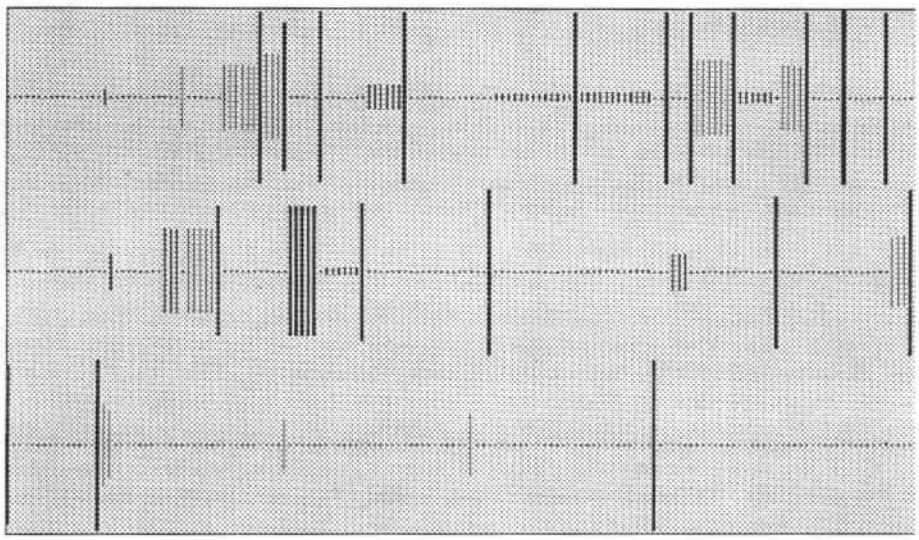

Abbildung 5.3: Ausgabe eines Netzes mit drei Ausgabeneuronen (Kein Phonemanfang, Vokal, Konsonant) auf den ersten 150 *frames* der Validierungsmenge.

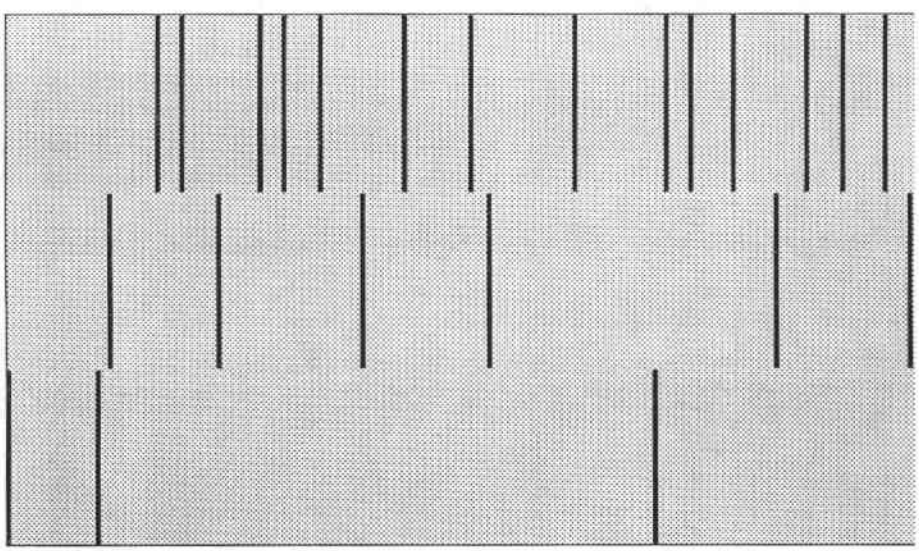

Abbildung 5.4: Zielausgabe eines Netzes mit drei Ausgabeneuronen (Kein Phonemanfang, Vokal, Konsonant) auf den ersten 150 *frames* der Validierungsmenge.

Ausgabeneuronen sind häufig unmittelbar vor oder nach einem tatsächlichen Phonemanfang aktiviert. Die häufigen falsch erkannten Phonemanfänge sind

damit erklärbar.

5.1.2 20 Phonemklassen erkennen

Bei 20 Ausgabeneuronen (Klasseneinteilung siehe Anhang) konvergierte ein Netz mit 100 Neuronen in der verborgenen Schicht ebenfalls gut. Das Netz erkannte von 338 *frames* in der Trainigsmenge nach 20 Trainingsschritten 307 korrekt, also 90.8%. Insbesondere erkannte es alle 169 nicht-Phonemanfänge eindeutig als solche. Auf den verbleibenden 169 *frames* erzielte es somit immer noch eine Trefferquote von 81.66%, was bei 20 Ausgabeneuronen deutlich besser als geraten ist. Auf der Validierungsmenge zeigte das Netz ähnlich gute Ergebnisse, brauchte allerdings etwas mehr Trainingsiterationen.
Insgesamt habe ich dieses Netz in etwa 200 Durchgängen trainiert. Verwendet man 300 Neuronen in der verborgenen Schicht, so wird der Maximalwert von 317 auf der Trainingsmenge und 30 (von 38) auf der Validierungsmenge schon nach etwa 20 Trainingsdurchläufen erreicht, danach verbesserte sich die Leistung allerdings nicht mehr. Eine Vergrößerung der verborgenen Schicht über 300 Neuronen hinaus änderte nichts an Trainingsgeschwindigkeit oder Maximalwerten.
Bei diesem Trainingsversuch war die Klasseneinteilung nicht besonders gut und die Trainingsmenge mit 338 Beispielen recht klein; einige Phonemklassen waren deutlich unterrepräsentiert, was sich auch in der Erkennungsleistung niederschlug.
Hier sind beispielhaft einige Aktivierungen der Ausgabeneuronen für die Klassen 6 (E, E2, EH, EHR, ER, ER2), 10 (I, IE, IHR, IR) und 13 (M, N, NG), außerdem für die Ausgabe *Kein Phonemanfang* aufgeführt. Die Bilder zeigen für jedes Ausgabeneuron seine Aktivierung bei verschiedenen Beispieleingaben.

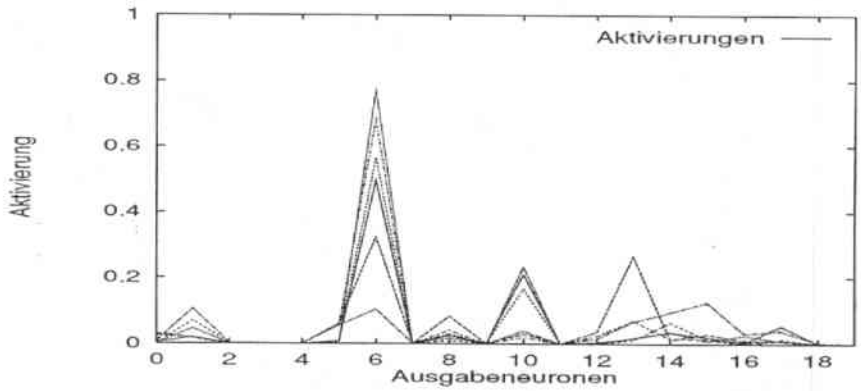

Abbildung 5.5: Aktivierungen für die Klasse 6 (E, E2, EH, EHR, ER, ER2)

Die Aktivierung für *Kein Phonemanfang* ist fast konstant 0, während bei den

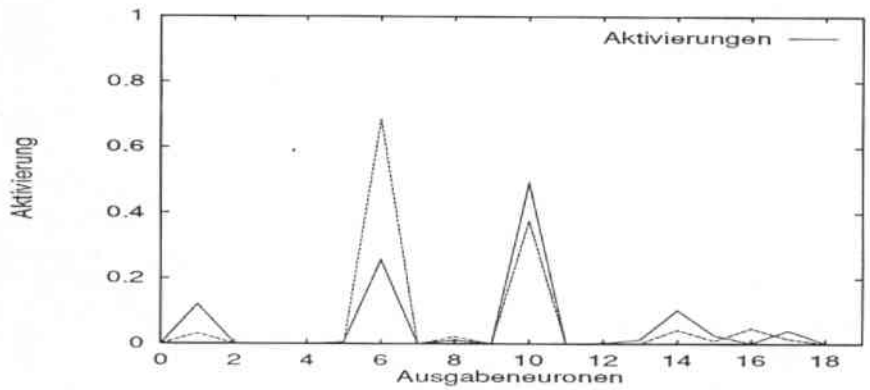

Abbildung 5.6: Aktivierungen für die Klasse 10 (I, IE, IHR, IR)

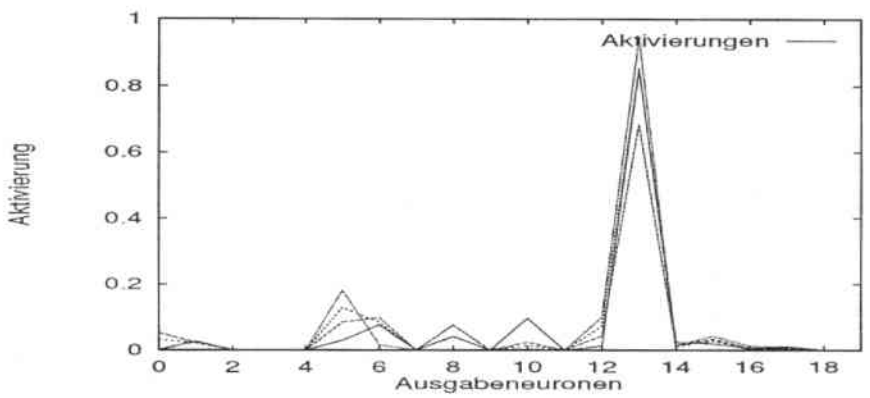

Abbildung 5.7: Aktivierungen für die Klasse 13 (M, N, NG)

echten Phonemanfängen immer mindestens ein Neuron eine Aktivierung über 0.01 zeigt — wenn es auch nicht immer das richtige ist. Einige Verwechslungen, wie zum Beispiel die der Klassen 6 und 10, sind leicht zu erklären — manchmal ist ein EH auch für einen Menschen kaum von einem I zu unterscheiden. Daher habe ich im nächsten Versuch eine andere Klasseneinteilung gewählt und nach Möglichkeit diejenigen Phoneme zu einer Klasse zusammengefaßt, die besonders häufig verwechselt wurden. Außerdem hatte die richtige Ausgabe einige Male immerhin die zweit- oder dritthöchste Aktivierung hatte. Es kann sich also lohnen, statt einer 1-aus-n- eine 3-aus-n-Bewertung durchzuführen. Das habe ich in den folgenden Versuchen berücksichtigt.

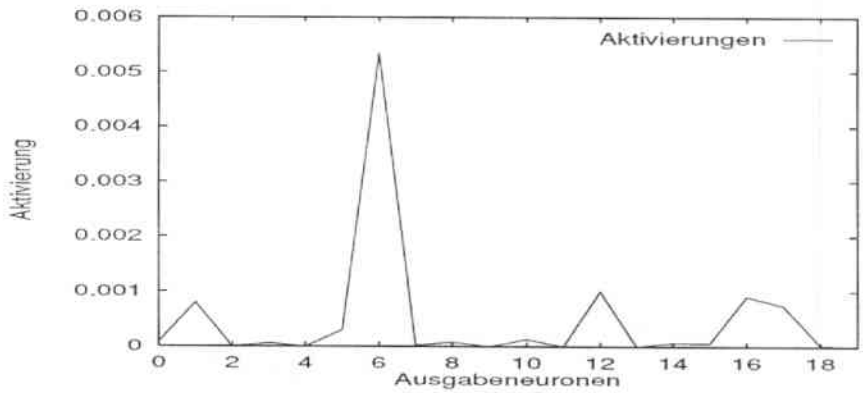

Abbildung 5.8: Aktivierungen für nicht-Phonemanfänge

Im nächsten Versuch enthielt die Trainingsmenge 1000 Beispiele, die Validierungsmenge 116. Die Klasseneinteilung erfolgte diesmal nach *classes3* (Seite 41); beim ersten Trainingsversuch hatte das Netz 100 Neuronen in der verborgenen Schicht. Der Anteil korrekt erkannter Phoneme in beiden Mengen ist dem folgenden Diagramm zu entnehmen:

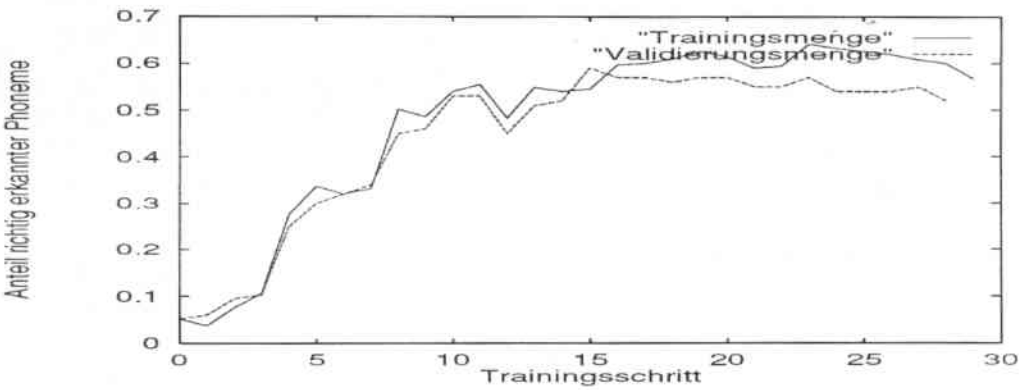

Abbildung 5.9: Anteil richtig erkannter Phoneme bei 20 Ausgabeneuronen

Die Beobachtung der Ausgaben zeigte, daß bestimmte Phonemklassen häufig verwechselt wurden. Daher habe ich bei einer Wiederholung des Versuchs gezählt, wie oft die richtige Ausgabe unter den drei am höchsten aktivierten Phonemen war; dadurch stieg die Trefferrate um 10%.
Mit 200 Neuronen in der verborgenen Schicht lernte das Netz etwa doppelt so schnell, und auch die Erkennungsleistung ließ sich bis auf etwas mehr als 70% steigern.

Bei diesen Versuchen fiel auf, daß das Netz meist zuerst nur einzelne Phonemklassen zu erkennen lernte und diese Menge dann nach und nach erweiterte. War jedoch eine Phonemklasse in der Trainingsmenge überrepräsentiert, so lernte das Netz manchmal, nur diese zu erkennen. Das ist wohl auch der Grund, warum die Erkennungsleistung manchmal leicht absank, obwohl der mittlere Fehler immer noch fiel: die Ausgabe für einzelne Phoneme rutschte dabei genau unter den Schwellwert, während sich die für andere von unten an ihn annäherte.

5.1.3 68 Phoneme unterscheiden

Als nächstes habe ich versucht, ein Netz mit 68 Ausgabeneuronen zu trainieren. Im ersten Versuch hatte das Netz 300 Neuronen in der verborgenen Schicht. Die Trainingsmenge hatte 810, die Validierungsmenge 96 Einträge. Nach einigen

Iterationen zeigte sich, daß dieses Netz nur die am stärksten repräsentierten Phonemklassen zu erkennen lernte. Die Erkennungsleistung blieb unter 50%. Bei einer Wiederholung des Versuchs mit 500 verborgenen Neuronen verstärkte sich diese Tendenz, bis das Netz nur noch lernte, grundsätzlich *Kein Phonemanfang* auszugeben. Bei Wiederholung des Versuches mit anderen Initialisierungswerten für die Gewichte lernte das Netz dann beispielsweise, nur das Phonem mit dem Index 54 zu erkennen — die Ergebnisse waren nie brauchbar. Ich nehme an, daß die Größe der Trainingsmenge nicht ausreichte, um die gewünschte Anzahl an Ausgaben zu trainieren. Um das Netz davon abzuhalten, einzelne Phoneme bis zur Perfektion zu erlernen, während es andere vernachlässigte, habe ich die Neuronenzahl bis auf 30 verringert. Damit wurden dann tatsächlich einige Phoneme erlernt, wenn auch nur etwa die 20 häufigsten. Daher habe ich dieses Training auf den nächsten Abschnitt (sprecherabhängiges Training) verschoben.

5.1.4 Zusammenfassung

Es ist ziemlich schwierig, Phonemanfänge als solche zu erkennen; ich vermute, daß ein Großteil der benötigten Neuronen dafür verwendet wird. In den Versuchen wurden nur sehr eindeutige nicht–Phonemanfänge zuverlässig identifiziert. Da die neuronalen Netze Phonemanfänge eigentlich nicht unbedingt zu erkennen brauchen — sie sind auch dann nützlich, wenn sie nur eine Einschätzung über die möglicherweise vorliegende Phonemklasse geben — habe ich auch Netze mit Trainingsmengen ohne nicht-Phonemanfänge trainiert. Dabei stellte sich heraus, daß schon ein Netz mit 20 verborgenen Neuronen in der Lage war, bei 20 Phonemklassen eine Trefferquote von gut 40% zu erzielen. Ab etwa 50 Neuronen allerdings lernte es nur noch die am stärksten repräsentierten Klassen zu erkennen, dort jedoch erzielte es Aktivierungen nahe bei 1.0. Ich nehme an, daß die Negativbeispiele dazu beitragen, Aktivierungen allgemein niedrig zu halten, so daß die Überzahl an Mustern für beispielsweise die Klasse 0 weniger stark ins Gewicht fällt. Auch das Netz mit 20 Neuronen erkannte nicht alle Klassen, dafür war es zu klein, aber auf den in der Trainingsmenge hinreichend stark (also mit mehr als 50 Mustern) vertretenen Klassen erreichte es Trefferquoten von 60% und mehr. Überraschenderweise konvergierte das Netz nicht schneller, wenn nur Phonemanfänge in der Trainingsmenge vorkamen. Die Netze gerieten viel leichter in lokale Minima, häufig wurde die Fehlerfunktion konstant oder begann zu oszillieren. Daher habe ich auch in den folgenden Versuchen Trainingsmengen sowohl mit als auch ohne Phonemanfänge betrachtet.

Insgesamt ergaben diese Versuche also folgende Ergebnisse: ein neuronales Netz war mit hinreichend langem Training in der Lage, Phonemklassen zu erkennen, Phonemanfänge allerdings nur in besonders eindeutigen Fällen. Dabei reagierte es allerdings sehr empfindlich gegenüber Variationen in der Blockgröße und der Anzahl verborgener Neuronen: bis zu einem gewissen Grad läßt sich durch Vergrößerung der verborgenen Schicht die Trainingsgeschwindigkeit steigern, aber eine zu große verborgene Schicht erhöht offenbar die Wahrscheinlich-

keit, in ein lokales Minimum zu geraten. Die Blockgröße dagegen war vor allem deshalb wichtig, weil einzelne Phonemklassen in den Trainingsmengen überrepräsentiert waren; waren sie zusätzlich in einzelnen Blöcken überrepräsentiert, so hatten sie auch bei der Gewichtsanpassung zu viel Einfluß. Zusammenfassend war es also auch im Interesse der Ausführungsgeschwindigkeit besser, die Anzahl verborgener Neuronen niedrig zu halten, auch wenn das ein längeres Training erforderlich machte.

5.2 Sprecherunabhängiges Training

Die Netze sind im Gebrauch nur dann interessant, wenn sie zu sprecherunabhängigen Generalisierungen in der Lage sind. Daher habe ich Trainingsmengen zusammengestellt, die Muster verschiedener Sprecher enthielten. Auf den Versuch, Phonemanfänge zu erkennen, habe ich dabei verzichtet, weil sich beim sprecherabhängigen Training schon gezeigt hatte, daß diese nicht zuverlässig genug erkannt werden konnten.
Wie zu erwarten war, ist die Leistung eines Netzes, das auf genau einen Sprecher trainiert wurde, schlecht auf Validierungsmengen, die von einem anderen Sprecher stammen. Die Leistung für untrainierte Sprecher bessert sich deutlich, wenn mehrere Sprecher in der Trainingsmenge vertreten sind.

5.2.1 Wechsel des Sprechers (20 Klassen)

Um zu sehen, wie gut die Generalisierungsfähigkeit der Netze ist, habe ich in einem Versuch mit zehn Sätzen mit jedem Satz 20 mal trainiert und bin dann zum nächsten Satz übergegangen. Folgendes Diagramm zeigt den Anteil richtig erkannter Phoneme in jedem der ersten zehn Trainingsschritte. Das Netz hatte 100 verborgene und 20 Ausgabeneuronen.

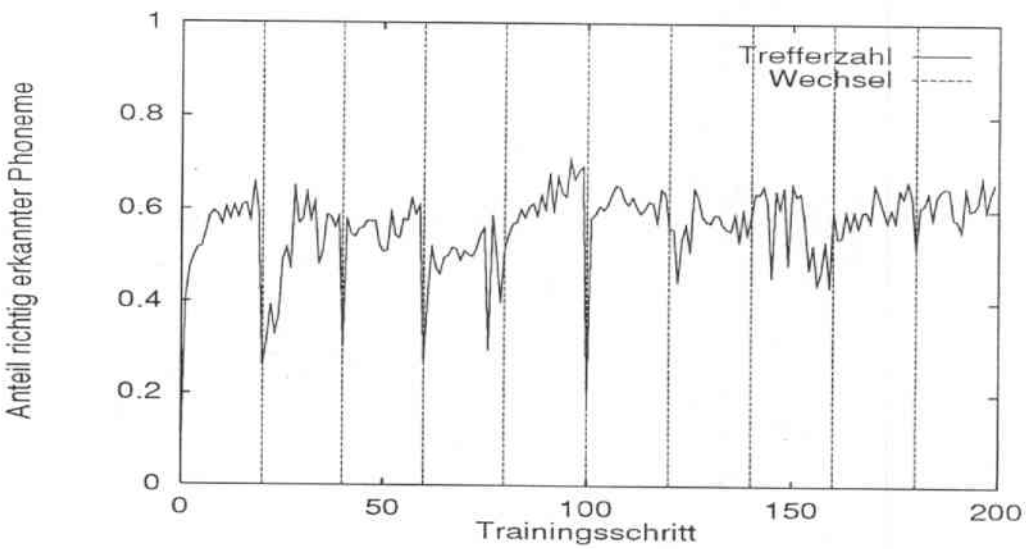

Abbildung 5.10: Anteil richtig erkannter Muster nach jedem Trainingsschritt

Vor dem Training erreicht das Netz eine Trefferquote von 5%, das entspricht bei 20 Ausgaben genau der Wahrscheinlichkeit, bei zufälligem Raten das richtige Ergebnis zu finden. Nach 20 Trainingsschritten erreicht das Netz 66% Treffer, dann wechselte die Trainingsmenge und die Trefferquote sank kurzfristig auf 25.96%. Bei den nächsten Wechseln der Trainingsmenge sank die Trefferquote immer seltener ab, bis der Wechsel zum Schluß gar nicht mehr am Diagramm zu erkennen ist. Das so trainierte Netz erreichte dann auch auf den zuerst trainierten Mengen Trefferquoten von etwa 60%.
Damit hat sich das Netz als überraschend stabil erwiesen, eine höhere Trefferquote wäre allerdings wünschenswert. Da die Fehlerkurve gegen Ende des Trainings stark oszillierte, habe ich den Versuch mit 200 Neuronen wiederholt, dabei sank jedoch die Trefferquote auf nicht trainierten Mengen. Wenn man aber mit den einzelnen Mengen häufiger trainiert, lassen sich durchschnittliche Trefferquoten von etwa 70% erzielen.

5.2.2 Gemischte Trainingsmengen

Bei dem vorangegangenen Versuch war in jeder Trainingsmenge nur jeweils ein Sprecher vertreten; sehr wahrscheinlich lernt das Netz beim Training individuelle Eigenschaften dieses Sprechers, die später die Anpassung an unbekannte Sprecher erschweren. Aus diesem Grund habe ich im nächsten Versuch gemischte Trainingsmengen verwendet. Die Muster stammten aus den gleichen 20 Sätzen wie im letzten Versuch, waren aber in 20 Trainingsmengen zu je 1000 Mustern zufällig gemischt.
Bei diesem Versuch waren die Wechsel der Trainingsmenge von Anfang an nicht an der Trefferquote abzulesen; die Trefferquote auf der Trainingsmenge stieg insgesamt langsamer als beim letzten Versuch, schwankte aber dafür weniger. Bei wiederholtem Training (insgesamt wurde jede Trainingsmenge drei mal verwendet) konnte ein Netz mit 200 verborgenen Neuronen auf untrainierten Mengen im Durchschnitt 76.15% der Phoneme der richtigen Klasse zuordnen.
Folgende Tabelle zeigt durchschnittliche Fehlerwahrscheinlichkeiten erster und zweiter Art (die Fehlerwahrscheinlichkeit erster Art für ein Phonem x gibt an, wie oft im Durchschnitt fälschlicherweise ein anderes Phonem als x erkannt wurde, die Fehlerwahrscheinlichkeit zweiter Art gibt an, wie oft im Durchschnitt *nicht* x ausgegeben wurde, obwohl x vorlag.) Die Klassen 6, 12 und 19 kamen in den Trainingsmengen zu selten vor, um aussagekräftige Ergebnisse zu erhalten.

Phonemklasse	FW 1. Art	FW 2. Art
0	0.06	0.14
1	0.24	0.29
2	0.52	0.47
3	0.31	0.26
4	0.18	0.36
5	0.17	0.31
7	0.1	0.29
9	0.37	0.28
10	0.03	0.4
11	0.26	0.37
13	0.55	0.11
14	0.42	0.44
15	0.13	0.31
16	0.27	0.15
17	0.03	0.52
18	0.28	0.43
Durchschnitt	0.245	0.314

Es war also möglich, einzelne Phonemklassen mit großer Zuverlässigkeit zu erkennen. Die hohen Fehlerwahrscheinlichkeiten speziell für die Klassen 13 und 2 sind darauf zurückzuführen, daß diese beiden Klassen sehr oft verwechselt wurden. Phonetisch ist das überraschend, denn Klasse 2 enthält die Phoneme AEH, AEHR, AI, E, E2, EH, EHR, ER, ER2, OE, OEH und Klasse 13 M, N, NG; sie sollten leicht zu unterscheiden sein, zumal es sich um zwei der am häufigsten vorkommenden Klassen handelt.
Eine 3–aus–n–Auswertung ergab Trefferquoten um 78%. Die folgende Tabelle gibt zu jedem Phonem an, wie wahrscheinlich es unter den drei am höchsten aktivierten ist und wie zuverlässig man aus der Tatsache, daß es unter den drei am höchsten aktivierten ist, schließen kann, daß es gesprochen wurde.

Phonemklasse	Unter den ersten 3	Zuverlässigkeit
0	0.94	0.91
1	0	–
2	0.9	0.92
3	0.81	0.86
4	0.78	0.74
5	0.5	0.5
7	0.92	0.92
9	0.86	0.86
10	0.36	0.36
11	0.51	0.54
13	0.65	0.65
14	0.63	0.63
15	0.94	0.94
16	0.97	0.97
17	0.67	0.62
18	0.33	0.33
Durchschnitt	0.67	0.72

Damit lassen sich einige Klassen bei einer 3–aus–n–Auswertung mit großer Sicherheit erkennen. Eine k–aus–n–Auswertung mit $k > 3$ ergab jedoch keine weitere Steigerung der Trefferrate. Auch eine Beschränkung auf nur 10 Klassen erhöhte die Trefferrate nicht.

5.2.3 68 Phoneme erkennen

Im letzten Versuch habe ich Netze mit 68 Ausgaben trainiert. Wiederum wurde nach jeweils 20 Trainingsschritten die Trainingsmenge gewechselt. Ein Netz mit 300 Neuronen in der mittleren Schicht pendelte sich bei einer Trefferquote von 60% auf der Trainingsmenge und 50 bis 55% auf der Validierungsmenge ein; dabei war es gegenüber Wechseln der Trainingsmenge weniger stabil als ein Netz mit 20 Ausgabeneuronen. Das folgende Diagramm zeigt den Anteil richtig erkannter Phoneme nach jedem Trainingsschritt (eine senkrechte Linie markiert jeweils den Wechsel der Trainingsmenge):

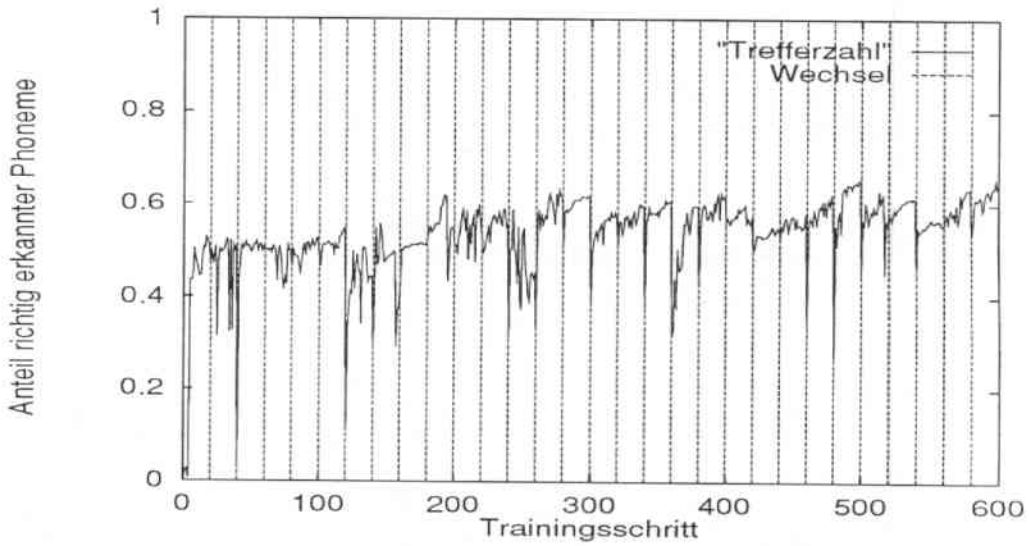

Abbildung 5.11: Anteil richtig erkannter Muster nach jedem Trainingsschritt

Fehlerwahrscheinlichkeiten erster und zweiter Art für die erkannten Phoneme:

Phonem	FW 1. Art	FW 2. Art
1	0.5	0
2	0	0
12	0.29	0.22
15	0.06	0.3
16	0.5	0
17	0.26	0.25
20	0.52	0.21
23	0.37	0.49
24	0.54	0.5
25	0.58	0.43
26	0.3	0.23
27	0.2	0.07
28	0.5	0
29	0	0.35
33	0.67	0.71
34	0.44	0.39
36	0.4	0.45
38	0.67	0.5
42	0.29	0.13
43	0.92	0
44	0.24	0.28
45	0.2	0.05
48	0.68	0
49	0	0
50	0.38	0
53	0.43	0.25
58	0.25	0.67
59	0	0.38
60	0.75	0
61	0	0
62	0.39	0.07
63	0	0.5
65	0.53	0.4
66	0.17	0.3
67	0.5	0.17
Durchschnitt	0.315	0.259

Auch bei einer k–aus–n–Auswertung stieg die Trefferquote nicht über 65%. Die Auswertung mit einem Netz mit 68 Ausgaben bleibt also relativ unzuverlässig. Um zu sehen, ob es sich lohnt, mit einer Einteilung der Phoneme in Klassen dem Netz einen Teil der Klassifizierungsarbeit abzunehmen, habe gezählt, wie oft die Ausgabe des Netzes in der richtigen Klasse lag, obwohl das

Netz mit 68 Ausgaben trainiert worden war.

Phonemklasse	FW 1. Art
0	0.25
1	0.37
2	0.2
3	0.88
4	0.64
5	0.23
7	0.39
9	0
10	0.34
11	0.67
13	0.32
14	0.33
15	0.13
16	0.5
17	0.32
18	0.28

Der Fehler ist hier durchschnittlich etwas größer, als wenn das Netz mit 20 Klassen trainiert worden ist; daraus kann man schließen, daß die Klasseneinteilung dem Netz die Erkennung erleichtert.
Insgesamt ist der Fehler bei 68 Ausgabeneuronen fast immer zu groß. Da die Anzahl der richtig erkannten Muster von Trainingsschritt zu Trainingsschritt schwankte, habe ich den Versuch mit mehr verborgenen Neuronen wiederholt, die Ergebnisse besserten sich dadurch jedoch nicht.

5.3 Fazit

Neuronale Netze waren in meinen Versuchen in der Lage, Phoneme mit etwa 55% Genauigkeit und Phonemklassen mit etwa 75% Genauigkeit zu erkennen. Mit einer Wahrscheinlichkeit von 78% und mehr ist die richtige von 20 Phonemklassen unter den 3 am höchsten aktivierten Ausgaben beziehungsweise das richtige von 68 Phonemen unter den 6 am höchsten aktivierten Ausgaben.
Dabei sind einige Phonemklassen offenbar leichter zu erkennen als andere; zum Teil hängt dies davon ab, wie stark einzelne Klassen in der Trainingsmenge vertreten sind, zum Teil aber auch von anderen Faktoren: zum Beispiel wurde die Klasse der Nasale (N, M, NG) wesentlich leichter gelernt als (S, SCH, TS, TSCH), obwohl letztere ungefähr gleich häufig vorkam wie erstere. Einige Klassen werden häufig verwechselt; hier ließe sich durch eine geschicktere Einteilung und durch ausgewogenere Trainingsmengen die Zuverlässigkeit sicher noch steigern.

Auffallend ist das Verhalten der Netze bei Vergrößerung der verborgenen

Schicht: die Erkennungsleistung wird ab einer bestimmten Neuronenzahl (zum Beispiel 300 für Netze mit 80 Eingaben und 20 oder 68 Ausgaben) nicht mehr wesentlich gesteigert, wohl aber die Trainingsgeschwindigkeit. Das ist einleuchtend, denn jedes zusätzliche Neuron in der verborgenen Schicht bedeutet einen Freiheitsgrad bei der Berechnung der gesuchten Funktion — je mehr verborgene Neuronen es gibt, desto mehr 'gute' Gewichtsverteilungen sind möglich und desto leichter wird es, schnell relativ gut Werte zu erreichen. Gleichzeitig steigt mit der Zahl verborgener Neuronen aber auch die Gefahr, in ein lokales Minimum zu geraten, aus dem das Netz sich dann selten innerhalb vertretbarer Zeit befreien konnte. Ein Beispiel dafür sieht man in einigen Trainingstabellen, wo nach wenigen Iterationen Trefferquoten um 50% erreicht wurden, die darauf zurückzuführen waren, daß das Netz einzelne Klassen mit absoluter Sicherheit erkannte und die Aktivierungen für die entsprechenden Ausgabeneuronen bis nahe an 1.0 steigerte, während es sämtliche anderen Klassen vernachlässigte. In diesen Fällen half manchmal nur ein Neustart des Versuchs mit veränderten Anfangsgewichten. Netze mit weniger Neuronen waren hier gerade wegen ihrer langsameren Konvergenz weniger gefährdet. Hier könnte es helfen, eine niedrigere Zielaktivierung vorzugeben oder ab einer bestimmten Aktivierung keinen Fehler mehr zu generieren. Dadurch wird allerdings wahrscheinlich die Konvergenzgeschwindigkeit beeinträchtigt.

Ein zweites wichtiges Kriterium ist die Geschwindigkeit, mit der ein Netz eine Aussage machen kann; hier ist eine kleine Tabelle mit Geschwindigkeiten für unterschiedlich große Netze und Eingabemengen. Die Messungen erfolgten auf einer DEC Alpha mit 122 Megabyte Hauptspeicher, auf der allerdings zum Zeitpunkt der Messungen noch andere Prozesse die CPU belasteten, so daß die Zeiten bestenfalls eine grobe Abschätzung liefern können.

Eingaben	verb. Neuronen	Ausgaben	Länge d. Eingabe	Zeit
80	300	20	31.1s	6s
80	200	20	31.1s	4.5s
80	200	20	3.5s	1s
80	100	20	31.1s	3s
80	100	20	3.5s	<1s
80	300	68	31.66s	10s
80	300	68	3.78s	2s
80	500	68	31.66s	13s
80	500	68	3.78s	7s

Kapitel 6

Ausblick

6.1 Mögliche Verbesserungen

6.1.1 Bessere Parameter

Durch Variation der Parameter bei Training und Auswertung läßt sich die Leistung der Netze sicher noch steigern; zum Beispiel könnte es sich lohnen, alternative Aktivierungs– oder Fehlerfunktionen auszuprobieren. Auch die Anpassung der Schritt– weite ließe sich geschickter vornehmen; sie sollte mit fortschreitendem Training generell kleiner werden, da zu vermuten ist, daß man sich dem Ziel nähert. Bis jetzt haben sich jedoch dank der sehr flach werdenden Fehlerkurven daraus keine Probleme ergeben.

6.1.2 Auswahl der Daten

Einige Phonemklassen konnten nicht gelernt werden, oder sie wurden nur unzureichend gelernt, weil sie in der Trainingsmenge nicht in hinreichender Anzahl vertreten waren; idealerweise sollten alle Phonemklassen gleich häufig in der Trainingsmenge vorkommen.

6.1.3 Vorverarbeitung der Daten

Man könnte die Anzahl der Eingabeparameter verringern, indem man auf dem FeatureSet eine LDA (Linear Discriminant Analysis) durchfürht. Man geht dabei von n-dimensionalen Vektoren aus, die Klassen zugeordnet sind und bildet sie so in einen m-dimensionalen Vektorraum ab ($m < n$), daß Vektoren, die in der gleichen Klasse sind, auch im Bildraum nahe beieinander liegen, während Vektoren aus verschiedenen Klassen weit entfernt sind. Dabei werden n-dimensionale Feature-Vektoren auf m Dimensionen reduziert, und zwar mit minimalem Informationsverlust. Neben der Reduktion der Eingabedimension, die sich unmit-

telbar in der Größe des benötigten Netzes und damit im Aufwand für Training und Auswertung niederschlägt, erleichtert man so auch dem neuronalen Netz die Erkennung, denn die transformierten Vektoren sind leichter voneinander zu unterscheiden.

6.1.4 Fehlerberechnung

Phonemanfänge sind selten scharf auf einen bestimmten *frame* einzugrenzen; im Training wird aber so verfahren, als wäre das möglich. Sollen beim Training alle *frames* berücksichtigt werden (also nicht nur solche, in denen ein Phonem anfängt und solche aus der Mitte zwischen zwei Phonemanfängen), dann könnte man dies in der Sollausgabe berücksichtigen, indem man für *frames* in der relativen Nähe eines Phonemanfanges Aktivierungen bis zu einer bestimmten Höhe zuläßt.
Außerdem könnte man versuchen, die beobachtete übermäßige Aktivierung bei einzelnen Phonemklassen dadurch zu verhindern, daß man Aktivierungen oberhalb eines gewissen Grenzwertes als 1.0 interpretiert und keinen Fehler mehr generiert. Dadurch ließe sich vielleicht auch die Ungleichverteilung in der Trainingsmenge teilweise ausgleichen.

6.1.5 Einsatz mehrerer Netze für Teilaufgaben

Das Training wird um so schwieriger, je mehr Muster unterschieden werden sollen. Eine Möglichkeit wäre, mehrere kleinere Netze zu verwenden, die jeder nur eine Untermenge der Muster richtig erkennen, also beispielsweise ein Netz, das nur entscheidet, ob es sich bei einem Muster um einen Vokal handelt oder nicht und ein zweites Netz, das zu einem Muster, von dem bekannt ist, daß es ein Vokal ist, erkennen kann, welcher Vokal vorliegt. Man kommt dann zwar mit kleineren Netzen aus, als ich sie im Moment benötige, verliert aber dafür an anderer Stelle Zeit, weil man mehrere neuronale Netze durchlaufen muß.

Anhang A

Phonemklassen

A.1 classes1

Klasseneinteilung für die Unterscheidung von nichtphonemischen Lauten, Vokalen und Konsonanten.

```
Klasse 0: +nKL, +QK, +hBR, +hEH, +hGH, +hHM, +hLG, +hSM,
          +nGN, +nMK, @, 0

Klasse 1:  A, AH, AEH, AEHR, AHR, AI, ANG, AR, AU, E, E2, EH,
           EHR, ER, ER2, EU, I, IE, IHR, IR, O, OE, OEH, OH, OHR,
           OR, U, UE, UEH, UEHR, UH, UHR, UR
Klasse 2:  B, CH, D, F, G, H, J, K, L, M, N, NG, P, R, S, SCH
           T, TS, TSCH, V, X, Z
```

A.2 classes2

Klasseneinteilung mit 20 Klassen in den ersten Versuchen

```
Klasse 0: +nKL, +QK, +hBR, +hEH, +hEM, +hGH, +hHM, +hLG, +hSM, +nGN, +nMK,
          @, SIL
Klasse 1: A, AH, AHR, AR
Klasse 2: AEHR, AEH, AI
Klasse 3: ANG
Klasse 4: AU
Klasse 5: B, D, G
Klasse 6: CH, E, E2, EH, EHR, ER, ER2
```

```
Klasse 7: EU
Klasse 8: F
Klasse 9: H
Klasse 10: I, IE, IHR, IR
Klasse 11: J
Klasse 12: K, P, T
Klasse 13: M, N, NG
Klasse 14: O, OE, OEH, OH, OHR, OR
Klasse 15: L, R
Klasse 16: S, SCH, TS, TSCH
Klasse 17: U, UE, UEH, UEHR, UH, UHR, UR
Klasse 18: V
Klasse 19: X
Klasse 20: Z
```

A.3 classes3

Klasseneinteilung mit 20 Klassen in den späteren Versuchen

```
Klasse 0: +nKL, +QK, +hBR, +hEH, +hGH, +hHM, +hLG, +hSM, +nGN, +nMK, @, SIL
Klasse 1: A, AH, AHR, ANG, AR
Klasse 2: AEH, AEHR, AI, E, E2, EH, EHR, ER, ER2, OE, OEH
Klasse 3: AU
Klasse 4: B, D, G
Klasse 5: CH
Klasse 6: EU
Klasse 7: F
Klasse 8: H
Klasse 9: I, IE, IHR, IR, UE, UEH, UEHR
Klasse 10: J
Klasse 11: K, P, T
Klasse 12: L
Klasse 13: M, N, NG
Klasse 14: O, OH, OHR, OR
Klasse 15: R
Klasse 16: S, SCH, TS, TSCH
Klasse 17: U, UH, UHR, UR
Klasse 18: V
Klasse 19: X
Klasse 20: Z
```

A.4 classes4

Das verwendete *PhonesSet*

```
Klasse 0: +QK
Klasse 1: +hBR
Klasse 2: +hEH
Klasse 3: +hEM
Klasse 4: +hGH
Klasse 5: +hHM
Klasse 6: +hLG
Klasse 7: +hSM
Klasse 8: +nGN
Klasse 9: +nKL
Klasse 10: +nMK
Klasse 11: @
Klasse 12: A
Klasse 13: AEH
Klasse 14: AEHR
Klasse 15: AH
Klasse 16: AHR
Klasse 17: AI
Klasse 18: ANG
Klasse 19: AR
Klasse 20: AU
Klasse 21: B
Klasse 22: CH
Klasse 23: D
Klasse 24: E
Klasse 25: E2
Klasse 26: EH
Klasse 27: EHR
Klasse 28: ER
Klasse 29: ER2
Klasse 30: EU
Klasse 31: F
Klasse 32: G
Klasse 33: H
Klasse 34: I
Klasse 35: IE
Klasse 36: IHR
Klasse 37: IR
Klasse 38: J
Klasse 39: K
```

```
Klasse 40: L
Klasse 41: M
Klasse 42: N
Klasse 43: NG
Klasse 44: O
Klasse 45: OE
Klasse 46: OEH
Klasse 47: OH
Klasse 48: OHR
Klasse 49: OR
Klasse 50: P
Klasse 51: R
Klasse 52: S
Klasse 53: SCH
Klasse 54: SIL
Klasse 55: T
Klasse 56: TS
Klasse 57: TSCH
Klasse 58: U
Klasse 59: UE
Klasse 60: UEH
Klasse 61: UEHR
Klasse 62: UH
Klasse 63: UHR
Klasse 64: UR
Klasse 65: V
Klasse 66: X
Klasse 67: Z
```

Literaturverzeichnis

[1] Herman Ney. A One-stage Dynamic Programming Approach for Conneced Word Recognition. in: Alex Waibel and Kai–Fu Lee (eds.). Readings in Speech Recognition. Morgan Kaufman Publishers, Inc., San Mateo, California, 1990

[2] Rüdiger Brause. Neuronale Netze. B.G. Teubner, Stuttgart, 1991

[3] Timothy Masters. Practical Neural Network Recipes in C++. Academic Press, 1993

[4] John Hertz, Anders Krogh, Richard Palmer. Introduction to the Theory of Neural Computation. Addison–Wesley, 1994

[5] Raul Rojas. Theorie der Neuronalen Netze. Springer Verlag, 1993