Phonemerkennung mittels Time Delay Neural Networks

Abteilung Neuroinformatik
Fakultät für Informatik
Universität Ulm

Benedikt Delker

Zusammenfassung

Time Delay Neural Networks sind neuronale Netzwerke, die in der Lage sind Beziehungen zwischen zeitversetzten Ereignissen zu erkennen. Sie sind somit in der Lage auch nicht exakt ausgerichtete Muster richtig zu klassifizieren. Dieses Verfahren bietet sich damit insbesondere für die Erkennung von Sprache, bei der eine genaue Ausrichtung der Muster nur schwer zu erreichen ist. Diese Ausarbeitung befaßt sich mit dem von Waibel et al. vorgestellten Konzept der TDNNs, was sowohl die prinzipielle Struktur wie auch zwei Beispiele beinhaltet.

1 Einleitung

Motiviert durch die Arbeit von Waibel, Hanazawa, Hinton, Shikano and Lang über Erkennung von Phonemen mittels Time Delay Neural Networks im Jahre 1989, befaßt sich diese Ausarbeitung ebenfalls mit gleicher Thematik. Dabei wird inhaltlich häufig Bezug auf genanntes Werk genommen, aber ebenso Parallelen zu späteren Arbeiten mit gleicher oder ähnlicher Thematik hergestellt werden. Hintergrund zur Phonemerkennung bildet das höhere Ziel gesprochene Worte automatisch richtig klassifizieren zu können.

7. Dezember 2004

2 Phoneme

Ein Phonem ist die kleinste bedeutungsunterscheidende Einheit [3,4,6]. Wählen wir als Beispiel die Wörter "Gang" und "Hang", so sind hier "H" und "G" Phoneme. Das Beispiel erweitert sich fast automatisch und gibt auch "D" als Phonem Preis ("Hand"). Von besonderem Interesse für die Spracherkennung sind Phoneme mit ähnlichem Klang, die diesem entsprechend in Gruppen eingeteilt werden. Waibel et al. haben sich in ihrer Grundsatzanalyse auf die Erkennung der Phoneme "B", "D" und "G" beschränkt.

3 Künstliche Neuronale Netzwerke

Künstliche Neuronale Netzwerke, im folgenden als KNN bezeichnet, sind der Versuch das Gehirn auf Maschinenebene nachzubilden. Prinzipiell sind neuronale Netzwerke in der Lage jede beliebige, berechenbare Funktion zu berechnen. Ein KNN besteht aus Einheiten u, die untereinander über Eingabeleitungen und Ausgabeleitungen vernetzt sind. Jede Einheit u verfügt über k_u Eingänge und einen Ausgang. Die Einheit berechnet intern eine Funktion $F : A^{k_u} \rightarrow A$, wobei A die Menge aller möglichen Ein– und Ausgaben ist. Der Funktionswert von F wird auf den Ausgang von u gelegt, der wieder als Eingang für eine andere Einheit dienen kann. Die Einheiten werden unterschieden in Eingabeeinheit, Ausgabeeinheit und versteckte Einheit. Eine Eingabeeinheit bekommt seine Eingabe von keiner Einheit des gleichen Netzes. Eine Ausgabeeinheit hingegen gibt seine Ausgabe an keine Einheit des gleichen Netzes. Für versteckte Knoten gilt, dass sie Ihre Eingabe von Einheiten des gleichen Netzes bekommen und ihre Ausgabe an Einheiten des gleichen Netzes abgeben. Man spricht in diesem Zusammenhang von mehreren Schichten. Die Eingabeeinheiten bilden die Eingabeschicht, usw. Jede der Eingaben ist gewichtet und im Normalfall wird die gewichtete Summe über den Eingängen berechnet und auf diesem Wert anhand einer einstelligen Funktion entschieden, ob und was die jeweilige Einheit ausgibt. Ein typische KNN ist das Single–Layer Perzeptron, das lediglich aus Ausgabeeinheiten besteht, die ihre gewichtete Eingabe direkt beziehen. Das Multi–Layer Perzeptron dagegen ist ein KNN, das aus mehreren Schichten besteht, die sich wie oben beschrieben verhalten. Im Allgemeinen sind Multi–Layer Perzeptrone feed–foreward Netzwerke, jede Schicht i bezieht ihre Eingaben aus Schicht $i-1$ oder vorhergehenden, und reicht ihre Ausgaben an Schicht $i+1$ oder nachfolgende weiter. Im Kontrast dazu stehen die rückgekoppelten Netzwerke, in denen Schlingen und Kopplungen in schon besuchte Schichten vorkommen. Es gibt noch eine Vielzahl weiterer KNNs, auf die wir nicht weiter eingehen werden. In Abbildung 1 ist ein mögliches Multi–Layer Perzeptron zu sehen, allerdings ohne die Gewichte oder Funktion der Einheiten explizit zu nennen.

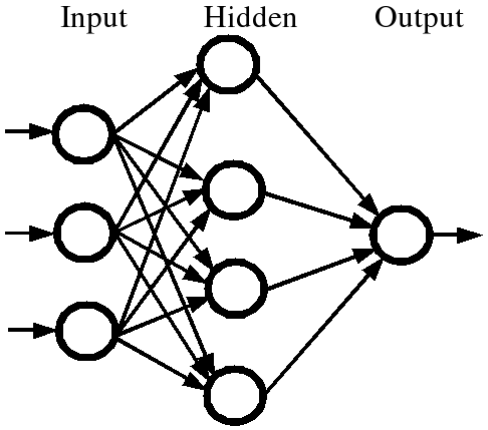

Abbildung 1. Multi Layer Perceptron mit drei Neuronen in der Eingabeschicht, 4 Neuronen in der Ausgabeschicht und einem Neuron in der Ausgabeschicht

3.1 Lernen von KNNs

Der wohl größte Vorteil von KNNs ist ihre Fähigkeit Muster zu lernen. Die Wahl eines geeigneten Lernalgorithmus ist aus diesem Grund eben so wichtig für die Funktionalität des Netzes wie der Entwurf des Netzwerkes selber. Die verschiedenen Konzepte lassen sich prinzipiell grob gliedern in überwachtes und unüberwachtes Lernen. Beim überwachten Lernen wird das KNN auf eine Eingabemenge angesetzt. Das gewünschte Ergebnis, das für jedes Element der Eingabemenge bekannt ist, wird mit der vom KNN erzeugten Ausgabe verglichen und bei Fehlern werden dementsprechende Änderungen an den Gewichten vorgenommen. Auf diese Weise sollte das KNN zu einem Zustand konvergieren in dem es in der Lage ist, jede Eingabe der Aufgabe des KNN entsprechend korrekt zu erkennen. Als Beispiele für überwachte Lernverfahren kann man die Delta–Regel, Backpropagation und Hebbsches Lernen nennen. Als Beispiel für unüberwachtes lernen kann man Self Organizing Maps aufführen. Im Rahmen dieser Ausarbeitung wird später grob auf den Backpropagation–Algorithmus eingegangen werden. Lernen nach der Delta–Regel berechnet den Unterschied zwischen der Ausgabe und der erwünschten Ausgabe und passt die Gewichte diesem Unterschied entsprechend mittels dem Gradienten an. Das Hebbsche Lernen bestimmt die Art und Weise wie Gewichte zwischen Neuronen geändert werden. Demnach werden Gewichte zwischen Neuronen erhöht, wenn sie gleichzeitig aktiviert werden und erniedrigt, wenn sie zu unterschiedlichen Zeitpunkten aktivert werden [2]. Zu dieser Methode gibt es verschiedene mathematische Variationen, welche die Entwicklung von Beziehungen fördern sollen. Self Organizing Maps sind neuronale Netze, die sich selber verwalten. Sie gewichten die Eingänge ihrer Neuronen nach folgendem Verfahren: Finden des Neurons, dessen Ausgabe am ehesten der gewünschten Ausgabe entspricht. Anpassen der Gewichte des gefundenen Neurons. Anpassen der Gewichte der Neuronen, die um das gefundene Neuron herum liegen.

Diese Anpassung geschieht abhängig von der Distanz zum gefundenen Neuron [2].

4 Time Delay Neural Network

Allerdings brauchen künstliche neuronale Netze Muster, die einen gewissen Grad an Statik besitzen. Sprache an und für sich jedoch ist etwas sehr dynamisches. Phoneme können beispielsweise unterschiedlich lang ausgesprochen werden. Deswegen ist es erforderlich, dass die Muster auch dann richtig erkannt werden, wenn sich ein Teil davon zeitlich verschoben hat. Prinzipiell sollte ein KNN folgende Eigenschaften haben um Phoneme, und insbesondere auch Sprache, vernünftig erkennen zu können[1]:

(1) Das KNN sollte mehrschichtig sein um Muster für nicht linear separierbare Mengen lernen zu können.
(2) Das KNN sollte in der Lage sein, Beziehungen zwischen zeitlich versetzten Ereignissen herstellen zu können.
(3) Die tatsächlich gelernten Features bzw. Abstraktionen sollten nicht durch zeitliche Größen beeinflußbar sein.
(4) Die zu lernenden Muster sollten nicht notwendiger Weise zeitlich präzise ausgerichtet sein müssen.
(5) Um das KNN zu motivieren Regelmäßigkeiten zu lernen sollte die Anzahl der Gewichte im Verhältnis zur Anzahl der zu lernenden Daten möglichst klein sein.

Eine Möglichkeit um diese Ziele zu erreichen ist die Konstruktion von Time Delay Neural Networks, im folgenden mit TDNN abgekürzt. Von Waibel et al. wird ein TDNN beschrieben, das alle fünf Punkte erfüllt und in der Lage ist die Phoneme "B", "D" und "G" zu erkennen. Auf die konkrete Realisierung wird später genauer eingegangen. Prinzipiell kann man ein TDNN dadurch erhalten, dass man eine Einheit eines KNN dahingehend erweitert, dass jeder der n Eingänge wieder in $k+1$ Leitungen aufgespalten wird, k von ihnen mit verschidenen Delays $(D_1, \ldots, D_k)$ und eine, die einfach so durchgeschalten wird. Diese $n \cdot (k+1)$ Leitungen haben jeweils wieder ihre eigenen Gewichte. Über das Auslösen eines Signals entscheidet dann wieder wie üblich eine Funktion. Auf diese Art und Weise kann die Einheit eines TDNN die momentane Eingabe in Beziehung zu jeder anderen Eingabe bis zum Zeitpunkt k vor der aktuellen setzen. In Abbildung 2 sehen wir den strukturellen Aufbau einer Einheit eines TDNNs.

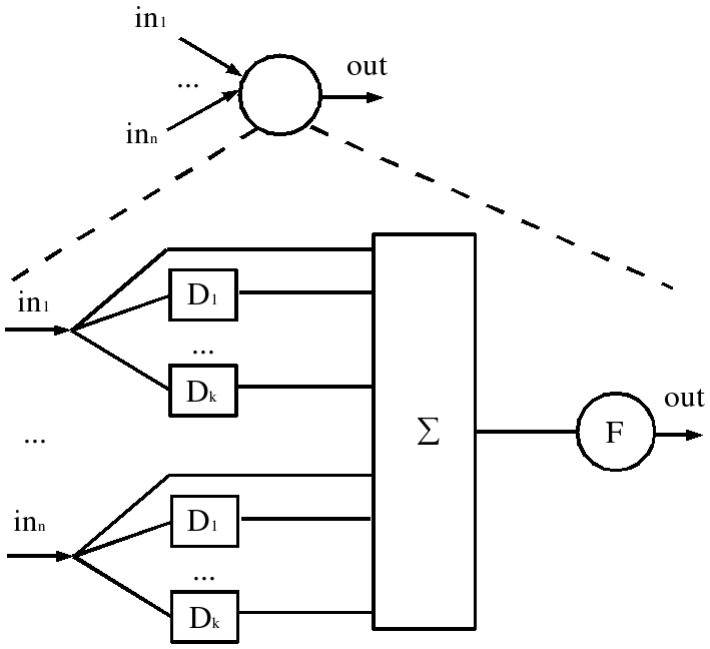

Abbildung 2. Aufbau der Einheiten im TDNN

5 Test mit TDNN, Waibel

Waibel hat mit seiner Arbeit über TDNNs eine konkrete Implementierung zur Erkennung der Phoneme "B", "D" und "G" vorgestellt. Das Grundprinzip wird hier nun vorgestellt. Es sei angemerkt dass die nachfolgende Beschreibung direkt aus [1] entnommen ist und dort präziser und detailierter nachgelesen werden kann, da hier auf die exakten technischen Details nicht eingegangen wird. Das in [1] beschriebene TDNN besteht aus drei Schichten. Die erste Schicht besteht aus 16 mel-skalierten Spektralkoeffizienten, die dem Netz als Eingabe dienen[1]. Die Spracheingabe wird in 15 Zeitfenster zerlegt, wobei eine Zeiteinheit 10 ms entspricht. Die zweite Schicht besteht aus acht Einheiten eines TDNN mit 16 Eingängen und je Eingang 2 Delays. Also läuft ein Fenster von drei Zeiteinheiten über die Eingabe, welche aus 15 Blöcken besteht, die jeweils wieder 16 Signale enthalten. Bei der Wahl der 3 Zeiteinheiten als betrachtetes Zeitfenster stützt sich Waibel auf die Erfahrungen früherer Arbeiten anderer Autoren [1]. In der nächsten Schicht, bestehend aus 3 Einheiten, wird das Zeitfenster auf 5 Zeiteinheiten vergrößert. Die Erweiterung des Zeitfensters soll das Lernen von Zusammenhängen über eine größere Reichweite fördern. Die Eingabe wird ausschließlich aus der vorhergehenden Schicht bezogen. Als letztes wird zeitabhängig die gewichtete Summe über der Ausgabe der dritten Schicht gebildet. Als Ausgabefunktion für die einzelnen Einheiten wurde die Sigmoid–Funktion gewählt.

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

Die Ableitung der Sigmoid–Funktion, welche für das Lernen mit dem Backpropagation – Algorithmus gebraucht wird, ist somit gegeben durch

$$\frac{d}{dx}s(x) = \frac{e^{-x}}{(1+e^{-cx})^2} = s(x)(1-s(x))[10].$$

Wegen dieser und weiterer schöner Eigenschaften bietet sich die Sigmoid–Funktion als Ausgabefunktion an.

5.1 Lernen im TDNN

Für den Lernprozess des TDNN wird der Backpropagation – Algorithmus eingesetzt [10]. Grob gesprochen wird zu einer Trainingsmenge T für jedes Element die erzeugte Ausgabe o mit dem gewünschten Ziel t verglichen und entsprechend die Gewichte geändert. Das Ziel ist es die Fehlerfunktion

$$E = \frac{1}{2}\sum_{i=1}^{|T|} \|o_i - t_i\|^2 \tag{1}$$

zu minimieren. Dies erreicht man durch iterative Anpassung der Gewichte mit Hilfe des Gradienten der Fehlerfunktion. Der Algorithmus besteht also aus zwei wesentlichen Schritten: Zuerst werden bei der Eingabe beginnend jede einzelne Ausgabe der Einheiten berechnet, sowie die Ableitung der Funktion an der Stelle des Eingabewertes. Durch die Backpropagation, die den zweiten Schritt darstellt, erhält jede Einheit ihren individuellen Fehler und passt nach diesem ihre Gewichte an. Dazu wird die Eingabe aus der Ausgaberichtung mit der Ableitung und einer Konstanten multipliziert. Das Ergebnis wird dann zu dem Gewicht addiert[10]. Ein TDNN kann äquivalent als statisches KNN dargestellt werden, indem die Knoten dupliziert werden und die Eingabe erweitert wird[1,8]. Um nun Muster auch unter zeitlicher Verschiebung zu erkennen, hat Waibel jede TDNN Einheit so oft dupliziert, wie es Zeitfenster gibt. In der ersten Schicht sind das also 15 Duplikate. Weiterhin wurde die Beschränkung eingeführt, dass die Gewichte der duplizierten Einheiten jeweils zu dem ihrer korrespondierenden Einheit gleich sein müssen, da sie ja jeweils nach dem gleichen Muster, jedoch zeitversetzt, Ausschau halten sollen. Zum besseren Verständnis betrachte man Abbildung 3. Dort sehen wir den Versuchsaufbau von Waibel et al. noch einmal in schematischer Form. Die Anordnung der Neuronen in einer Matrix dient dabei der Veranschaulichung. Das Neuron (k,l) in der ersten verdeckten Schicht bezieht seine Eingaben also von allen Neuronen (i,j) für die gilt $1 \leq i \leq i_{max}$ und $l \leq j \leq l + r^{(1)} - 1$. In der ersten verdeckten Schicht ist somit Anzahl aller Gewichte das Produkt von $j_{max}, i_{max}, k_{max}$ und $r^{(1)}$. In unserem Beispiel also 4992. Die einzelnen Gewichte bezeichnen wir mit w_{ijkl}. Für die zweite verdeckte Schicht wird analog verfahren. Die Ausgabeschicht summiert dann die Quadrate der einzelnen Ausgaben aus der

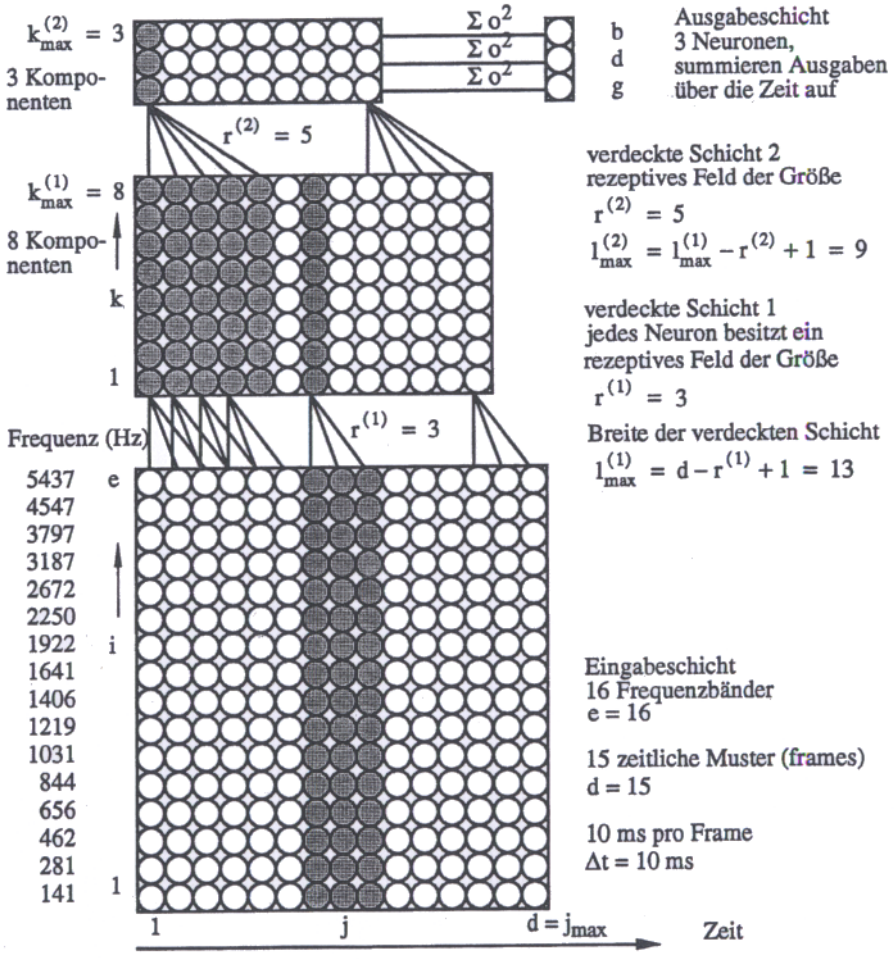

Abbildung 3. Schematischer Aufbau eines TDNNs [11]

zweiten verdeckten Schicht über der Zeit ungewichtet auf. Um nun mit dem Backpropagation – Algortihmus zu lernen, was nichts anderes bedeutet als die Gewichte anzupassen, wird die Gleichung (1) minimiert, d. h. sie wird nach dem anzupassenden Gewicht differenziert. Die Änderung des Gewichts für eine Eingabe p ist dann nach [11]

$$\Delta_p w_{ijkl} = \eta o_{p,kl} \delta_{p,kl},$$

wobei

$$\delta_{p,kl} = \begin{cases} f'(net_{p,kl}) 2 o_{p,kl} (t_{p,k} - \sum_{l=1}^{l_{max}} o_{p,kl}^2) & \text{falls } (k,l) \text{ Ausgabezelle} \\ f'(net_{p,kl}) \sum_{m=1}^{m_{max}} \sum_{n=l-r+1}^{l} (\delta_{p,mn} w_{klmn}) & \text{falls } (k,l) \text{ verdeckte Zelle} \end{cases}.$$

Dies bedeutet praktisch nichts anderes als dass der Fehler von der Ausgabe durch das Netz bis hin zur Eingabe zurückpropagiert wird. Dabei wird der propagierte Wert jeweils mit dem individuellen Fehler $\delta_{p,kl}$ des Neurons (k,l) multipliziert und dann an alle Vorgänger (i,j) weiterpropagiert. Auf diese

Weise kann es passieren, dass die Neuronen einer Zeile unterschiedliche Gewichtsänderungen erhalten. Um Translationsinvarianz weiterhin zu erhalten wird der Mittelwert der Summe aller Änderungen als neues $\Delta_p w_{ijk}$ verwendet. Damit sieht die Anpassung folgendermaßen aus:

$$w_{ijkl} = w_{ijkl} + \Delta w_{ijk} = w_{ijkl} + \frac{1}{l_{max}} \sum_{l=1}^{l_{max}} \Delta_p w_{ijkl}$$

De facto wurde nach obiger Beschreibung für jede Einheit schrittweise einzeln gelernt und die Gewichte je Schritt als Durchschnitt der einzelnen, neu gelernten Gewichte gewählt. Auf diese Weise war es möglich auch sehr verzerrte Muster richtig zu erkennen. Allerdings ist diese Art und Weise des Lernens recht teuer. Waibel hat für seine Trainingsmenge bestehend aus 800 Beispielen zwischen 20000 und 50000 Iterationen je Beispiel benötigt (grob über den Daumen gepeilt insgesamt etwa $28 \cdot 10^6$ Iterationen). Trotz einiger Optimierungen dauerte die Lernphase einige Tage, wobei nicht alle Optimierungsmöglichkeiten ausgeschöpft wurden.

5.2 Vergleich : TDNN und Hidden Markov Model

Um eine geeignete Aussage über die Leistungsfähigkeit von TDNNs treffen zu können, hat Waibel sein TDNN mit den besten ihm zur Verfügung stehenden HMMs verglichen. Zu diesem Test wurden 5240 Japanische Worte von drei Männern gesprochen, deren Muttersprache Japanisch ist. Jeweils die Hälfte davon bildeten die Trainings– sowie Testmenge für das HMM. Der Versuch war ausgerichtet auf die Phoneme "B", "D" und "G". Die Netzwerke wurden jeweils sprecherabhängig trainiert. Durch die Einbettung der Phoneme in Worte ist gewährleistet, dass das Netzwerk die typischen Muster lernen muss, die ein Phonem hat, auch wenn es in unterschiedlichen Färbungen vorkommt, wie es zum Beispiel bei "BI" und "BA"der Fall ist.
Die Ergebnisse sind zu sehen in Tabelle 1. Sie beziehen sich ausschließlich auf die Testmenge, welche für das Netzwerk völlig unbekannt war. Für alle drei Sprecher ist die Erkennungsrate bei dem TDNN jeweils höher als für das HMM.

Drei wichtige Eigenschaften des TDNN konnten beobachtet werden. Zuallererst war das Netzwerk in der Lage ohne menschliche Interaktion (von der einmaligen Sprachaufnahme abgesehen) wichtige sprachliche Abstraktionen zu lernen. Weiterhin konnte es Darstellungsformen lernen, die verschiedene akustische Ereignisse zueinander in Relation setzen, und war somit in der Lage seine Entscheidung besser zu fundieren. Als drittes schließlich ist das Netzwerk tatsächlich in der Lage auf Verschiebungen in der zeitlichen Anordnung richtig zu reagieren. Das TDNN war in der Lage durchschnittlich $98,5\%$ aller

Sprecher	Anzahl Eingaben	Erkennungsrate	TDNN	Erkennungsrate	HMM
MAU	b(227)	98,2	98,8	92,1	92,9
	d(179)	98,3		96,7	
	g(252)	99,6		90,6	
MHT	b(208)	99,0	99,1	96,2	97,2
	d(170)	100		98,2	
	g(254)	98,4		97,2	
MNM	b(216)	94,9	97,5	87,5	90,9
	d(178)	99,4		92,7	
	g(256)	98,4		92,6	

Tabelle 1
Ergebnisse TDNN – HMM

Testeingaben richtig zu erkennen, wohingegen das beste, verwendete HMM lediglich $93,7\%$ aller Testeingaben richtig erkannt hat. Ein weiterer sehr interessanter Aspekt ist, dass das TDNN es dazu tendiert keine Aussage über ein Phonem zu treffen, anstatt eventuell eine Falschaussage von sich zu geben.

6 Test mit TDNN, Tan Keng Yan

Colin Tan Keng Yan hat sich im Jahre 2000 im Rahmen einer Masterarbeit für die Universität in Singapur mit einem ähnlichen Thema befasst. Insgesamt hat er 12 TDNNs konstruiert, die jeweils unabhängig voneinander aus zwölf Phonemklassen jeweils eine lernen sollten. Jedes dieser 12 TDNNs ist gleich aufgebaut und wird lediglich zu anderen Zwecken trainiert. Die Eingabe besteht bei ihm aus 4 Eingabeknoten. Die versteckte Schicht besteht aus 36 Knoten und die Ausgabeschicht abhängig von der Klasse aus 2, 3, 4 oder 6 Knoten. Das Zeitfenster für die Eingabeschicht ist 3, während die versteckte Schicht 5 Zeitfenster betrachtet. Für das Lernen des TDNN wurde eine Beispielmenge von durchschnittlich 3825 Eingaben (ungleichmäßig verteilt) je Klasse gewählt. Aus diesen Mengen wurden jeweils 25% zufällig aussortiert, um später eine Kreuzvalidierungsmenge zu haben. Mit dieser soll überprüft werden, ob das Netzwerk die zu lernende Menge auswendig oder, so wie es sein soll, die Struktur lernt. Diese Kreuzvalidierungsmenge wurde zu determinierten Zeitpunkten mit der Trainingsmenge kombiniert und von dieser Menge wurden wieder zufällig 25% als Kreuzvalidierungsmenge aussortiert. Anwendung während des Lernens fand hier die Generalized Delta Rule. Die Ergebnisse reichen von $56,47\%$ bis $90,63\%$ korrekt erkannter Phoneme. Jedoch bleibt anzumerken, dass Tan Keng Yan keine reinen TDNNs konstru-

iert hat, sondern diese in eine Struktur eingebettet hat, die ihm ermöglichen sollte Phoneme sprecheradaptiv zu erkennen. Trotzdem bilden die TDNNs den Kern seines Systems.

7 Zusammenfassung

In dieser Ausarbeitung wurde das TDNN als Möglichkeit zur Phonemerkennung vorgestellt. Es wurde ein kurzer Überblick über künstliche neuronale Netzwerke gegeben und einige Lernverfahren angesprochen. Weiterhin wurde die prinzipielle Struktur von TDNNs zur Sprache gebracht. Wir haben zwei konkrete Realisierungen mittel TDNNs gesehen und einen Überblick über ihre Leistungsfähigkeit erhalten. Besonders die Arbeit von Waible et al. hat gezeigt, dass die Verwendung von TDNNs zur Erkennung von Phonemen geeignet ist. Sie haben im Vergleich zu einer Auswahl von Hidden Markov Modellen wesentlich besser abgeschnitten. Allerdings muss man hierzu noch erwähnen, dass die Lernzeit von TDNNs wesentlich höher ist als die bei HMMs. Die Lernzeit von TDNNs beträgt auch auf heutigen, handelsüblichen Maschinen noch einige Tage [7]. Deshalb muss die Entscheidung über die Verwendung von TDNNs immernoch in Abhängigkeit von der konkreten Anwendung getroffen werden.

Literatur

[1] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, Kevin J. Lang
1989 Phoneme Recognition Using Time Delay Neural Networks, IEEE

[2] http://en.wikipedia.org/wiki/Neural_network

[3] http://de.wikipedia.org/wiki/Phonem

[4] http://en.wikipedia.org/wiki/Phoneme

[5] http://www.informatik.hu-berlin.de/~boettche/sprache/sprache.html

[6] Carstensen, Ebert, Endriss, Jekut, Klabunde, Lange
2001 Computerlinguistik und Sprachtechnologie, Spektrum Lehrbuch

[7] Tan Keng Yan, Colin
2000 Speaker Adaptive Phoneme Recognition Using Time Delay Neural Networks, Thesis at the National University of Singapore

[8] http://bach.ece.jhu.edu/~tim/research/tdnn/tdnn.ps

[9] http://uhavax.hartford.edu/compsci/neural-networks-definition.html

[10] http://page.mi.fu-berlin.de/~rojas/neural/chap7.ps

[11] Andreas Zell
1996 Simulation Neuronaler Netze, Addison-Wesley

[12] Altmann, Hans & Ute Ziegenhain
2002 Phonetik, Phonologie und Graphemik fürs Examen. Westdeutscher Verlag: Wiesbaden