Spracherkennung mit Hardware Neuronalen Netzen

Diplomarbeit
von
Michael Berthold

Betreuung:
Prof. Dr. Alex Waibel
Dipl. Inform. H. Hild

Bearbeitungszeitraum:
1. April 1992 — 30. September 1992

Institut für Logik, Komplexität und Deduktionssysteme
Fakultät für Informatik
Universität Fridericiana (TH) Karlsruhe
D-7500 Karlsruhe 1

Ich versichere hiermit, die vorliegende Arbeit selbständig und ohne unzulässige Hilfsmittel angefertigt zu haben. Die verwendeten Quellen sind im Literaturverzeichnis aufgeführt.

Pittsburgh, den 30. September 1992

Vorwort

Diese Diplomarbeit entstand im Frühjahr und Sommer 1992 während meines Aufenthaltes an der Carnegie Mellon University, Pittsburgh, USA.
An dieser Stelle möchte ich mich bei Alex Waibel bedanken, der mir die Möglichkeit gab, diese Arbeit in seiner Gruppe an der CMU anfertigen zu können. Meinem Betreuer Hermann Hild verdanke ich interessante Diskussionen im "Murray Street Cafe", die noch ein paar der berühmten Prozente einbrachten.
Torsten Zeppenfeld, der mir geduldig die kleinen, aber tückischen "Hacks" erklärte, die nötig sind, damit der Wordspotter diesem Namen auch gerecht wird, möchte ich ganz besonders Dank sagen.
Mark Holler und Simon Tam von Intel danke ich für die großzügige Unterstützung dieser Arbeit und der immer vorhandenen Bereitschaft, auftretende Schwierigkeiten gemeinsam zu lösen.
Dank schulde ich auch all denen, die mir bei der Beseitigung von Tipp– (und anderen) Fehlern halfen. Nennen möchte ich hier stellvertretend Rainer („Bei dem Hardware–Kauderwelsch fallen einem ja die Zähne aus!") und Petra, die als Germanistin mit den Tücken der Informatikersprache gekämpft hat.
Für die moralische Unterstützung beim Kampf gegen die Technik (warum ein "floating point error: precision" zum kompletten Absturz führen muß, ist mir bis heute nicht ganz klar!), möchte ich mich bei Barbara und Ingrid vom CMT und bei Peter, der per eMail seine Kommentare beisteuerte, bedanken.
Und nicht zuletzt danke ich meinen Eltern, die mir dieses Studium ermöglicht haben.

Zusammenfassung

Diese Arbeit beschreibt die Realisierung eines Spracherkenners mit Hilfe von spezieller Hardware. Da herkömmliche Erkenner meist Software Simulationen von Neuronalen Netzen verwenden, was dazu führt, daß keine Echtzeiterkennung möglich ist, ist eine Implementierung auf spezialisierter Hardware von großem Interesse. Die vorliegende Arbeit vergleicht einige vorhandene Systeme in Hinsicht auf die Möglichkeit der Realisierung eines Wordspotters und stellt schließlich die Implementierung dieses Spracherkenners auf einem der Systeme, dem ETANN (Electrically Trainable Analog Neural Network) von Intel vor.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Vor nicht allzu vielen Jahrzehnten war der Glaube, daß die Fähigkeiten von Computern bald von denen des menschlichen Gehirns nicht mehr zu unterscheiden sind, noch weitverbreitet. Mittlerweile ist aber sehr deutlich geworden, daß „gnadenlose Rechenpower“ selbst bei so logisch erscheinenden Aufgaben wie dem Schachspiel dem Menschen immer noch unterlegen ist. Wie sollen dann aber erst die vielen so menschlichen, gefühlskoordinierten Aufgaben dem Computer „beigebracht“ werden? Eines der großen Probleme ist hierbei sicherlich die Spracherkennung. Wäre es nicht schön, nicht mehr alle Kommandos mühsam eintippen zu müssen, sondern einfach mit seinem Computer zu reden? Oder seiner Waschmaschine nur noch sagen zu müssen, was zu tun sei? Ein maschineller Butler, der mit einem „einen Martini. Gerührt, nicht geschüttelt!“ etwas anfangen kann, auch wenn es im Tumult einer Bar untergeht, das Auto mit dem man sich streiten kann, wie ein Stau am Besten zu umfahren ist, oder einfach nur die berühmte “black box”, die alle Fremdsprachen dieser Welt versteht und in die eigene Sprache übersetzt (was auch einige Fallen der Logik vermeiden helfen würde und den Babel Fisch erübrigen könnte[1]) sind weitere Beispiele. Betrachtet man sich dieses Szenario ein wenig näher, stellt man bald fest, daß man hier mit strenger Logik sehr schnell an die Grenzen stößt. Schon wenn es darum geht, einzelne Kommandos zu erkennen, ist man vor undeutlicher Aussprache, Akzenten, Geschwindigkeitsunterschieden und vielen anderen Eigenschaften der men-

[1] siehe auch Douglas Adams: „Per Anhalter durch die Galaxis“

schlichen Stimme nicht gefeit.

Für diese Problematik wurden erst in den letzten Jahren viele neue Lösungsideen vorgestellt. Es gibt Methoden, die versuchen, den Verschiedenheiten der Menschen mit Hilfe von statistischen Mitteln beizukommen. Auch wurden Versuche unternommen, die Spracherzeugung des Menschen zu simulieren, um damit herauszufinden, welches die wichtigen Kriterien sind, zu erkennen, was er nun wirklich gesagt hat. Die in letzter Zeit sehr populär gewordenen Neuronalen Netze, die versuchen, sich an die Funktionsweise des menschlichen Gehirns anzulehnen (allerdings stark abstrahiert und vereinfacht!), waren zu Beginn nicht sehr erfolgreich, da durch ihre statische Natur keine Möglichkeit bestand, Zeitabhängigkeiten festzustellen. Erst durch Modifikationen der originalen Struktur, wie zum Beispiel durch das "Time Delay Neural Network", das ermöglicht, Abhängigkeiten zwischen verschiedenen Zeitpunkten festzustellen und zudem auch noch in bestimmten Grenzen zeitinvariant ist, wurde diese Technik „salonfähig".

Eines der großen Probleme all dieser Ansätze ist aber immer der gewaltige Aufwand an Daten, die benötigt werden, das jeweilige System dazu zu bringen, möglichst gut zu erkennen. Insbesondere bei Neuronalen Netzen ist der Aufwand an Trainingszeit oftmals gewaltig. Das führt dazu, daß viele neue Ideen nicht vernünftig erprobt werden können, da ein Trainingslauf nicht selten mehrere Tage benötigt. Wie soll da der Einfluß eines Parameters festgestellt werden? Da niemand warten will, bis die nächsten, noch schnelleren Maschinen verfügbar sind und sich anscheinend für immer schnellere Maschinen immer zeitfressendere Aufgaben finden lassen, liegt die Lösung hier in schneller, spezialisierter Hardware.

Grundidee dieser Arbeit ist zum einen, vorhandene Hardware für Neuronale Netze auf ihre Verwendbarkeit in der Spracherkennung zu untersuchen, und zum anderen, auf einem geeigneten System die Implementierung eines Spracherkenners vorzunehmen. Anhand eines schon existierenden Systems, eines Wordspotters, der in der Lage ist, in sprecherunabhängiger Sprache Schlüsselwörter zu erkennen, wird dargestellt, wie eine solche Portierung abläuft.

Die vorliegende Arbeit kann grob in zwei Abschnitte aufgeteilt werden. In

den folgenden Kapiteln werden verschiedene Hardware–Lösungen für die schnelle Simulation von Neuronalen Netzen vorgestellt und — soweit möglich — miteinander verglichen. Das zweite Kapitel gibt dabei einen groben Abriß über momentan verwendete Methoden (Stand: Mitte 1992), um verschiedene Ziele bei der Simulation zu erreichen. Kapitel 3 beschreibt dann einige der Systeme näher und geht dabei auch auf technische Einzelheiten ein. Eine Gegenüberstellung der Systeme rundet das dritte Kapitel ab. Die Gründe für die endgültige Entscheidung für eines dieser Systeme legt Kapitel 4 dar, und das fünfte Kapitel beschreibt das ausgewählte System näher.

Der zweite Abschnitt beschreibt die Realisierung eines Spracherkenners auf einer VLSI–Implementierung eines Neuronalen Netzes und beginnt mit einem Kapitel, das den hier verwendeten Erkenner beschreibt. Darauf folgt ein Kapitel, das vorstellt, wie Intels ETANN zur Berechnung einer Diskreten Fourier Transformation verwendet werden kann, wie Time Delay Neural Networks auf dem ETANN realisiert werden können und wie sich eine DTW–Routine durch ein Linear Time Alignment ersetzen läßt. Daran anschließend wird die tatsächliche Implementierung des Erkenners beschrieben und einige Resultate vorgestellt. Das letzte Kapitel faßt die Arbeit schließlich kurz zusammen.

Im Anhang wird der bekannte Back–Propagation Trainings Algorithmus für Neuronale Netze kurz wiederholt und zwei Erweiterungen dieses Algorithmus beschrieben. Zusätzlich wird im zweiten Teil des Anhangs der während dieser Arbeit entwickelte Simulator für Neuronale Netze vorgestellt, mit dem alle in dieser Arbeit erwähnten Experimente durchgeführt wurden.

So eine Arbeit wird eigentlich nie fertig,
man muß sie für fertig erklären,
wenn man nach Zeit und Umständen
das möglichste getan hat.
(J. W. Goethe, Italienische Reise)

Kapitel 2

Hardware–Implementierungen für Neuronale Netze

Die Entwicklung von Chips, die versuchen, einige oder alle Aspekte von Neuronalen Netzen abzudecken, findet schon seit längerer Zeit statt. Zu Beginn war den Entwicklern meist selbst nicht klar, daß sie eigentlich eine Art Neuronales Netz bauten, wie z.B. C. Carroll, der versuchte, ein Pfadsuchproblem mit Hilfe von Hardware schneller zu lösen (siehe [3]).

Seitdem aber offensichtlich ist, daß die Simulation Neuronaler Netze eine Aufgabe ist, die auf einem Rechner, also als reine Softwaresimulation, gewaltige Rechenzeit erfordert, werden regelmäßig neue Konzepte vorgestellt. Grob unterscheiden kann man all diese Entwürfe in folgende Kategorien:

- Digitale Signal–Prozessoren
  Hier wird einfach versucht, die nötige Rechenleistung durch Verwendung spezialisierter Prozessoren zu erreichen. Die Berechnung erfolgt aber nach wie vor rein digital, d.h. meistens in Form von Fließkomma–Arithmetik. Ein schönes Beispiel hierfür ist die in Kapitel 3 vorgestellte RAP. DSPs sind auch deshalb gut geeignet, weil die schnelle Simulation von Neuronalen Netzen hauptsächlich die Berechnung von Additionen und Multiplikationen erfordert.

- neuronale Co–Prozessoren
  Dieser Ansatz versucht die zeitkritischen Operationen auf einen entsprechend spezialisierten Co–Prozessor auszulagern. Ein gutes Beispiel ist der Lneuro–Chip (siehe [2]), der versucht, die Leistung eines Transputer–Systems durch die Verwendung von Co–Prozessoren zu verbessern. Diese Lneuro–Chips berechnen parallel die gewichtete Summe der Eingänge und wenden auf das Ergebnis noch ein Sigmoid an. Das Ganze geschieht digital und ermöglicht damit eine einfache Verwendung als „Rechenknecht“ für die zeitkritischen Teile einer Simulation Neuronaler Netze.

- neuronale Prozessoren
  Hier muss man zumindest unterscheiden in:

  - rein digitale Realisierung
    Hier wird konventionelle Entwurfstechnik, wie sie vom Mikroprozessor–Entwurf bekannt ist, verwendet, um einen spezialisierten Prozessor zu konstruieren, der sich besonders für die Simulation von Neuronalen Netzen eignet. Die Berechnung geschieht dabei rein digital, meist in Form von Fließkommazahlen, manchmal auch als 16 bit Integers. Beispiel hierfür ist ein System, welches von Hitachi[1] entwickelt wurde. Es besteht aus einem 5 Zoll–Wafer, auf dem 144 digitale Neuronen untergebracht werden, die vollständig verbunden sind. Durch Kombination von 8 dieser Wafer kommt das gesamte System auf 1000 Neuronen (durch Fehler auf den einzelnen Wafern können nicht jeweils alle 144 Neuronen verwendet werden) und erreicht damit eine zehnmal höhere Rechenleistung als ein Hitachi S–820 Supercomputer bei der Software–Simulation des entsprechenden Netzes (siehe hierzu auch [6], [7]).
  - digital/analoge Realisierung
    In diese Rubrik fallen alle Chips, auf denen die Darstellung der Gewichte und Ein–/Ausgänge teils digital, teils analog ist. Meist werden hier die Gewichte digital gespeichert und die Ein–/Ausgänge analog dargestellt. Der später, in Kapitel 3,

[1] Central Research Laboratory, Hitachi, Tokyo, Japan

vorgestellte CCD–Chip basiert auf diesem Prinzip.

- rein analoge Darstellung
  Hier existieren viele unterschiedliche Varianten. Unterscheiden kann man hauptsächlich die Art der Repräsentation der Gewichte auf dem Chip (als Ladungen in Kondensatoren, in EEPROM-Zellen, ...). Diese Darstellung wird unter anderem bei Intels ETANN verwendet, der in Kapitel 3 ausführlich vorgestellt wird.

Diese Aufteilung ist natürlich willkürlich. Wichtiges Unterscheidungsmerkmal sind auch die möglichen Netzstrukturen, die mit Hilfe des entsprechenden Chips realisiert werden können. Einige Exemplare erlauben vollständig verbundene Netze, die dann aber meistens nur einschichtig sind, andere ermöglichen die freie Verbindung beliebiger Neuronen, erlauben dann aber nur eine sehr begrenzte Anzahl von Verbindungen (siehe z.B. [4], 256 Neuronen, die mit einem beliebigen anderen Neuron verbunden werden können, es sind aber jeweils nur 128 Verbindungen gestattet.)

Erwähnt werden sollen auch noch Ansätze, ein Neuronales Netz optisch zu realisieren. Hier wurden einige interessante Entwicklungen vorgestellt, die die Durchführung einer rein optischen Matrix–Vektor–Multiplikation ermöglichen. Allerdings sind solche Projekte bisher noch auf große Aufbauten mit optischen Bänken im Labor angewiesen und daher für diese Arbeit also nicht weiter von Interesse.

Bei den meisten bisher realisierten Systemen wurde vor allem Wert darauf gelegt, eine schnelle Berechnung des Neuronalen Netzes im Vorwärtsbetrieb zu ermöglichen. Die wenigsten Chips gestatten auch ein Trainieren des Neuronalen Netzes ohne Unterstützung durch einen Computer. Meist wird davon ausgegangen, daß die Daten für das trainierte Netz (Verbindungsstruktur und Gewichte) durch eine vorausgegangene Software–Simulation erzeugt wurden. Das liegt unter anderem daran, daß durch die Integration der zusätzlichen Logik, die für ein "on–chip"–Lernen erforderlich wäre, die erreichbare Arbeitsgeschwindigkeit und/oder die Anzahl der Synapsen/Neuronen pro Chip deutlich niedriger ausfallen würden.

Es gibt aber auch in dieser Richtung schon Ansätze, wie zum Beispiel einen Chip, der in Anlehnung an den Algorithmus der Boltzmann–Maschine ein

stochastisches Lernen ermöglicht (siehe hierzu [5]). Auf dem ersten Exemplar waren daher auch nur 6 Neuronen und 15 Synapsen vorhanden. Der Chip ermöglichte aber sowohl überwachtes als auch unüberwachtes Lernen und konnte damit das bekannte XOR–Problem auch tatsächlich in wenigen Millisekunden lernen. Auch die weiter oben erwähnte Wafer Scale Integration–Lösung von Hitachi ermöglicht ein Trainieren des Netzes on line. Hier wurde als Trainingsverfahren der Back–Propagation Algorithmus gewählt.

Kapitel 3

Verfügbare Systeme

Nach diesem kleinen Rundblick über verschiedene Ansätze, ein Neuronales Netz in Hardware zu realisieren, sollen die folgenden Abschnitte Vertreter dieser Art von Hardware ein wenig näher vorstellen. Alle Systeme vertreten verschiedene Philosophien, wie man ein Neuronales Netz in Hardware „gießen“ kann, und die jeweiligen Abschnitte versuchen, auf die Besonderheiten der entsprechenden Realisierung einzugehen.

Am Anfang steht ein sogenannter „CCD programmierbarer Signalprozessor“, bei dessen Entwicklung das Hauptaugenmerk auf der schnellen Berechnung einer gewichteten Summe lag. Das darauf folgende System legt mehr Wert auf eine möglichst modulare Konzeption, so daß auf verschiedene Ansprüche bezüglich Genauigkeit und Komplexität Rücksicht genommen werden kann. Das führt zu einer Art „Baukasten“, mit dem sich unterschiedliche Konfigurationen realisieren lassen. Der dritte und letzte hier vorgestellte Einzelchip ist vermutlich der praxisnahste und realisiert ein komplettes Neuronales Netz auf einem Chip, ermöglicht dem Benutzer aber Einflußnahme auf viele der Parameter und den Ablauf bei der Berechnung. Dadurch wird der Chip sehr flexibel und läßt sich vielfältig einsetzen.

Die letzten zwei vorgestellten Systeme sind schließlich mehr mit Blick auf die enormen Trainingszeiten bei Neuronalen Netzen konzipiert. Beide Systeme versuchen, durch die Zusammenschaltung spezieller Prozessoren eine geeignetere Struktur für die Simulation von Neuronalen Netzen bereitzustellen als die herkömmlichen „von Neumann“–Rechner; und dies, im Gegen-

satz zu massiv parallelen Großrechnern, zu einem erschwinglichen Preis.

Den Abschluß dieses Kapitels bildet eine kurzer Vergleich der Daten aller Systeme.

3.1 Ein CCD–programmierbarer Signalprozessor

Der von Dr. Alice Chiang[1] entwickelte „CCD[2]–programmierbare Signalprozessor“ ermöglicht die Berechnung von 14 verschiedenen gewichteten Summen mit jeweils bis zu 144 Elementen. Bild 3.1 zeigt das verwendete Prinzip auf.

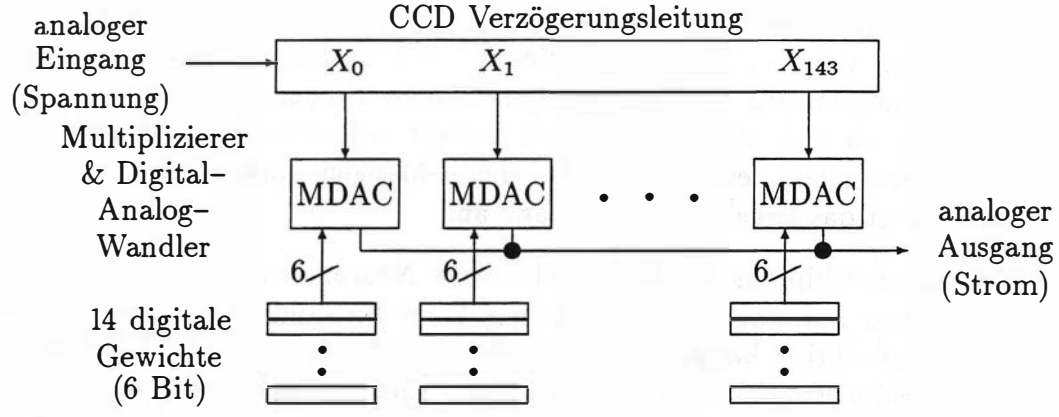


Abbildung 3.1: Blockschaltbild des CCD–programmierbaren Signalprozessors

Die analogen Eingangssignale werden in einer Verzögerungsleitung mit 144 Anschlüssen gehalten. Diese werden dann mit einem der 14 6–bit–Gewichte multipliziert und aufsummiert. Das Ergebnis liegt am Ausgang als (analoger)

[1] Lincoln Laboratory, Massachusetts Institute of Technology
[2] CCD = Charge–Coupled Device

Strom vor. Entscheidend sind die hierbei verwendeten Multiplizierer. Da die Berechnung möglichst schnell durchgeführt werden soll, sind spezielle Verfahren nötig, um eine analoge Spannung mit einem digitalen Gewicht zu multiplizieren. Das Prinzip der hier verwendeten MDAC's (Multiplizierender Digital–Analog Wandler) beruht (bei einem N–bit Gewicht) auf N parallelen Eingängen, deren Eingangs–Gates eine Fläche proportional zu 2^n haben, $0 \leq n \leq N-1$ (siehe auch [8]). Dadurch werden die Transistoren, die zu den jeweiligen Eingangsleitungen gehören, entsprechend gewichtet, und der Ausgangsstrom ist proportional zum Ergebnis der Multiplikation.
Der tatsächlich realisierte Chip erreicht damit eine Rechenzeit von 100 Nanosekunden für eine dieser Summen. Das bedeutet, daß dieser Chip $1,44*10^9$ Multiplikationen und $1,43*10^9$ Additionen[3] pro Sekunde berechnet. Das führt zu einer Rechenleistung von etwa 2,8 Milliarden arithmetischen Operationen pro Sekunde bei einer Leistungsaufnahme von ungefähr 2 Watt.

Die Bedienung des Chip ist denkbar einfach. Die Initialisierung erfordert nur das Laden der 144*14 digitalen 6–bit–Gewichte über einen Adreß– und Datenbus; vor den Rechenoperationen wird zusätzlich noch die analoge Verzögerungsleitung mit den entsprechenden Eingangswerten geladen (je nachdem, ob man alle Eingänge neu belegen will, oder teilweise alte Werte wiederverwendet), eine der 14 Gewichte–Mengen ausgewählt, und 100ns später liegt das Ergebnis am Ausgang an.

Um diesen Chip für die Simulation eines Neuronalen Netzes einzusetzen, würde also nur noch die Berechnung eines Sigmoids oder einer anderen Ausgangsfunktion benötigt werden. Für den Einsatz als TDNN (Time Delay Neural Network) wäre die verwendete Verzögerungsleitung sehr günstig. Allerdings kann die gesamte Rechenleistung nur dann wirklich ausgenutzt werden, wenn man einen Eingang zu 144 verschiedenen Zeitpunkten betrachten will. Ansonsten ist man gezwungen, die Verzögerungsleitung als Schieberegister zu verwenden und verliert dadurch Zeit, um dieses zu laden. Bei z.B. vier Eingängen zu je 36 Zeitpunkten würde nur noch ein Viertel der Rechenleistung erreicht.

Dieser Chip existiert bisher nur als Labormuster und wurde auch nur "stand alone" getestet.

[3] eine 144–fache Summe benötigt 143 Additionen.

3.2 Der JPL–Chipsatz

Der in der Gruppe von Dr. Anil Thakoor[4] entwickelte Chipsatz besteht aus Chips, die Synapsen zur Verfügung stellen, und Chips, die die dazugehörigen Neuronen enthalten. Das vereinfacht die Auswahl verschiedener Neuronen und der Genauigkeit und Implementierung der Synapsen. Bisher realisierte Chips enthalten 1024 Synapsen bzw. 32 Neuronen.

Die Synapsen–Chips existieren in verschiedenen Ausführungen, haben aber alle gemeinsam, daß die analogen Eingangssignale als Spannungen und die Ausgangssignale als Ströme vorliegen. Unterschiede bestehen nur in der Repräsentation der Gewichte. Hier existieren digitale Versionen mit Auflösungen von 1,5,6 und 7 Bit und eine analoge Version, die die Gewichte in Sample&Hold–Gliedern speichert. Die Auflösung soll hier fast nur vom externen D/A–Wandler abhängen, der benötigt wird, um die Gewichte zu laden.

Neuronen–Chips existieren bisher nur in einer Ausführung, die 32 Neuronen bereitstellt. Über je einen stromgesteuerten Widerstand läßt sich die Steigung des jeweiligen Sigmoids verändern.

Diese Chips sind in gewissen Grenzen kaskadierbar, man muß aber immer beachten, daß die Neuronen hier R–C–Glieder beinhalten, was dazu führt, daß man an bestimmte Maximalgeschwindigkeiten gebunden ist. Bei den bisher verwendeten Chips liegen diese im Rahmen einiger Mikrosekunden für die Berechnung eines Neuronen–Ausgangs. Das führt zu einer Rechenleistung in der Größenordnung von $1 * 10^8$ arithmetischen Operationen pro Sekunde, bei 32 Eingängen und 32 Neuronen (also jeweils ein Chip). Bild 3.2 zeigt die Kaskadierung mehrerer Chips um ein Neuronales Netz mit einem Hidden Layer zu realisieren.

Für den Einsatz als Simulator für neuronale Netze sind diese Chips also gut geeignet. Durch die Kaskadierungsmöglichkeit ist man nicht auf eine feste Netz–Topologie festgelegt, und durch die Auswahl an Synapsen–Chips kann auch die Auflösung der Gewichte leicht geändert werden.

[4]Jet Propulsion Laboratory, California Institute of Technology

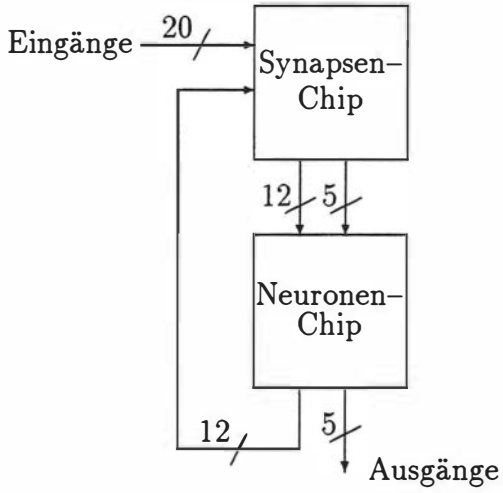


Abbildung 3.2: Zwei Chips, die zusammen ein Netz mit 20 Eingängen, 12 Hidden Units und 5 Ausgängen berechnen. Durch die Wahl eines anderen Synapsen–Chips kann die Auflösung der Gewichte gewählt werden (siehe Text).

3.3 Intels ETANN

Der von der "Novel Device Group"[5] entwickelte Chip ETANN (Electrically Trainable Analog Neural Network) bietet 64 Neuronen und 10240 Synapsen Platz. Interessant an diesem Chip ist die chipinterne Darstellung der Gewichte dieser Synapsen (siehe auch [11]). Um eine analoge Darstellung zu erhalten, die aber nicht–flüchtig sein soll, werden vom EEPROM (Electrically Erasable PROM) bekannte Techniken verwendet. In einem sogenannten "differential floating gate" werden zwei EEPROM–Zellen zur Speicherung eines Gewichtes benutzt.

Bild 3.3 zeigt zusätzlich zu den 64 Ein– und Ausgängen, dem 64*64–Eingangs–Synapsenfeld noch ein sogenanntes "feedback–array". Dieses kann dazu verwendet werden, die Eingangsanzahl zu verdoppeln, indem die Eingangswerte in zwei Blöcken angelegt werden und beide Synapsenfelder zusammen genutzt werden. Man kann den Chip aber auch in einem rückgekoppelten Modus betreiben oder ein Netz mit mehr als zwei Layern realisieren, in-

[5]Intel Corporation, Technology Development

dem man die Ausgangswerte wieder in das untere Synapsenfeld zurückführt. So besteht die Möglichkeit, ein Netz mit bis zu 64 Ein– und Ausgängen zu realisieren, das theoretisch bis zu 64 Layer haben kann. Allerdings ist die Gesamtzahl der Neuronen auf 64 beschränkt, in diesem Fall wäre also pro Layer nur ein Neuron vorhanden.

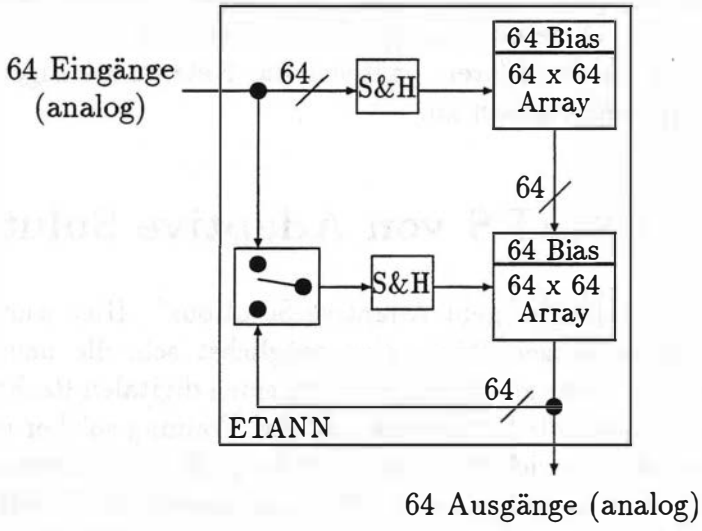


Abbildung 3.3: Blockschaltbild des ETANN–Chips von Intel.

Um nach Anlegen der Eingänge ein korrektes Ergebnis am Ausgang bereitzustellen, benötigt der Chip etwa 20 Mikrosekunden, und erreicht damit eine Rechenleistung von ca. $5 * 10^9$ Multiplikationen pro Sekunde. Das führt zu etwa 10 Milliarden arithmetischen Operationen pro Sekunde.

Zu diesem Chip werden im Moment schon komplette Entwurfswerkzeuge angeboten. Damit ist es möglich, auf einem IBM PC/AT mit einem entsprechenden Adapter Chips zu programmieren. Die dafür benötigten Daten werden vorher in einem Simulationslauf berechnet, wobei hier verschiedene Lernverfahren einsetzbar sind. Selbstverständlich kann man auch schon vorhandene Werte für die Gewichte verwenden. Eine interessante Möglichkeit

ist, daß der Chip sozusagen „nachtrainiert“ werden kann. Da die Gewichte analog gespeichert werden, haben Fertigungstoleranzen einen entscheidenden Einfluß darauf, daß sich zwei Chips nie exakt gleich verhalten, d.h. es wäre extrem aufwendig, die Eigenheiten des jeweiligen Chips bei der Simulation zu berücksichtigen. Das führt natürlich zu Fehlern, die in Kauf genommen werden müßten. Durch das sogenannte “chip in loop”–Training können diese Effekte allerdings vermieden werden. Hier werden bei dem normalen Software–Trainingslauf die Ergebnisse des Forward–Passes nicht berechnet, sondern vom ETANN selber erzeugt. Verwendet man nun diese Ergebnisse, um das Training durchzuführen, paßt sich das Netz an die Eigenschaften des jeweiligen Chips automatisch an.

3.4 Die CNAPS von Adaptive Solutions

Einen völlig anderen Weg geht Adaptive Solutions[6]. Hier wurde nicht versucht, mit Hilfe analoger Tricks eine möglichst schnelle neuronale Netz–Simulation auf die Beine zu stellen, sondern einen digitalen Rechner zu bauen, der sich besonders für die Simulation und das Training solcher und ähnlicher Strukturen eignet. Entwickelt wurde einer der größten im Moment hergestellten Chips, der 64 Processing Units (PU) und insgesamt 256KByte Speicher beherbergt. ASI bietet einen kompletten Rechner (CNAPS = Connected Network of Adaptive Processors, siehe auch [9]) an, der bis zu vier dieser Chips enthält und an vorhandene Netzwerke (Ethernet) angeschlossen werden kann. Mit der vorhandenen Software ist diese Maschine für schnelle Neuronale Netz–Simulationen und –Trainingsläufe einsetzbar. Die Programmierung mit einem angepassten C ermöglicht aber auch die Implementierung anderer hochparalleler Strukturen.

Grundidee dieser Maschine ist es, das Netz layerweise auf die PU's zu verteilen (siehe Bild 3.4). Es wird also zuerst die Eingabe in das Netz auf den Eingangsbus gelegt, danach an dem Ausgangsbus die Ausgabe dieses ersten Layers ausgelesen und evtl. (falls noch mehr Layer folgen sollen) als Eingabe wieder auf den Eingangsbus gelegt. Falls mehr Units in diesem Layer enthalten als PU's vorhanden sind, werden einfach mehrere Units auf

[6] Adaptive Solutions INC. , Portland, Oregon

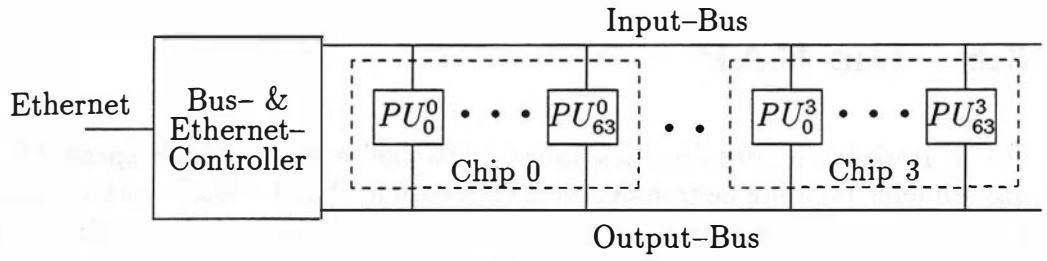


Abbildung 3.4: Blockschaltbild der CNAPS

einer PU berechnet. Durch genügend Speicher (4KByte pro PU) können mehrere Sätze von Gewichten auf der jeweiligen PU gehalten werden. Adaptive Solutions gibt für das 4–Chip–System eine Spitzenleistung von 5 Milliarden Connections pro Sekunde an, überflügelt damit also problemlos momentane Simulationen auf Großrechnern (CM–5, Cray). Durch die Verwendung universell programmierbarer PU's ist es auch möglich, auf der Maschine Netze zu trainieren. Mit einer Spitzenleistung von etwa einer Milliarde Connection–Updates pro Sekunde[7] ist die CNAPS auch hier zur Zeit konkurrenzlos schnell.

Ein Problem dieses Systems ist die interne Darstellung der Gewichte, die hier als 16bit–Zahlen repräsentiert werden, was gegenüber den üblichen Software–Simulationen, die meistens Fließkommazahlen verwenden, zu einem großen Handikap werden könnte. Das scheint allerdings im Moment keine weiteren Probleme zu bereiten. Untersuchungen über den Einfluß beschränkter Auflösung an den Gewichten eines Netzes (siehe [22]) haben gezeigt, daß ab einer Auflösung von 12–13 Bit keine merklichen Einflüsse auf das Trainings–Verhalten mehr feststellbar sind. Im einfachen Vorwärtsbetrieb ist sogar schon eine Auflösung von 5–6bit ausreichend. Großes Problem ist nur, daß sich die üblichen, mit Software–Simulationen trainierten Netze, nicht ohne Anpassungen auf die CNAPS übertragen lassen.

Mit einem Preis einer 4–Chip–Maschine von ca. $50.000 bietet die CNAPS aber auf jeden Fall im Moment ein konkurrenzlos niedriges Preis/Leistungs–Verhältnis.

[7]Diese Zeit bezieht sich wieder auf das 4–Chip–System auf dem mit dem Back–Propagation–Algorithmus trainiert wird.

3.5 Die RAP

Die in Berkeley innerhalb eines Jahres entwickelte RAP[8] wurde speziell für das schnelle Training neuronaler Netze entworfen. Auf der Basis vorhandener Komponenten sollte innerhalb kürzester Zeit eine funktionsfähige Maschine gebaut werden. Das gelang durch Verwendung von digitalen Signalprozessoren TMS320C30 von Texas Instruments, die durch programmierbare Gate–Arrays von Xilinx miteinander verbunden sind. Die DSPs werden als Rechenelemente verwendet, um schnelle Fließkommaoperationen durchzuführen. Durch die Gate–Arrays wird die Übertragung der Daten zwischen ihnen gesteuert. Auf 4 Karten werden jeweils 4 DSPs, 16MByte dynamisches RAM (erweiterbar auf 64MB) und 1MByte schnelles, statisches RAM untergebracht. Die Ankopplung an das Ethernet besorgt ein Host–Rechner, der auf einem Motorola 68020 basiert. Bild 3.5 zeigt ein Blockschaltbild der RAP.

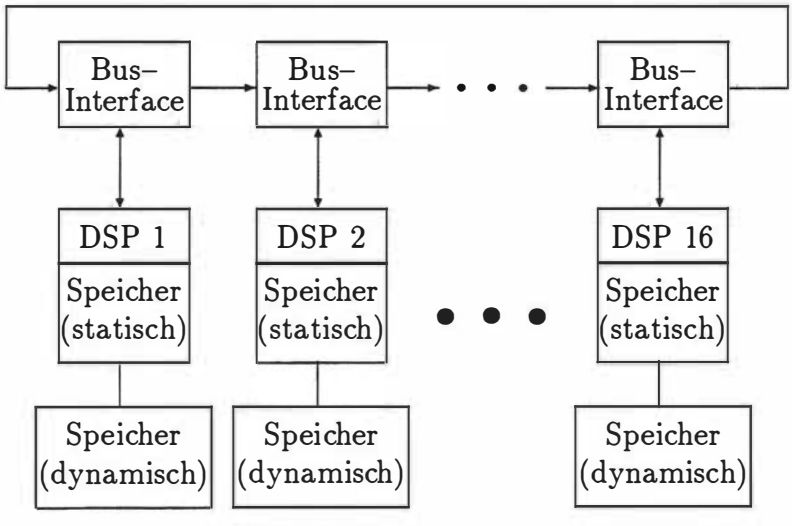


Abbildung 3.5: Blockschaltbild der RAP (16 Knoten)

Die theoretische Rechenleistung beträgt 128MFLOPs pro Karte und damit

[8]RAP = Ring Array Processor, siehe auch [10]

ein halbes GigaFLOP für die RAP selbst. In der Praxis erreicht man dann etwa ein Drittel bis die Hälfte dieser Leistung, abhängig vom verwendeten Algorithmus, da sich die Netzwerktopologie meistens nicht optimal auf die Architektur der Maschine abbilden läßt.

Ein großer Vorteil der gewählten Architektur besteht darin, daß man relativ unproblematisch Änderungen am Datenübertragungsprotokoll durchführen kann, weil nur die Gate–Arrays neu programmiert werden müssen. Da hier Xilinx Gate–Arrays verwendet werden, genügt es sogar, nur die jeweiligen seriellen PROMs neu zu programmieren.

In der jetzigen Konfiguration kommt die RAP bei der Simulation Neuronaler Netze auf eine Leistung von ca. 50 Millionen Connections pro Sekunde (MCPS), bzw. 12 Millionen Connection–Updates pro Sekunde (MCUPS) beim Trainieren des Netzes mittels Back–Propagation bei einem Preis von ca. $100.000 (allerdings ist diese Maschine nicht kommerziell erhältlich, sondern wird nur zum Selbstkostenpreis an andere Universitäten abgegeben).

Für die Programmierung der RAP steht ein komplettes Entwicklungssystem auf dem SUN Host–Rechner zur Verfügung. Ein angepasster C–Compiler mit erweiterter Bibliothek sorgt dafür, daß entsprechender Code für die einzelnen DSPs und den 68020 VME–Host generiert wird. Die entsprechenden Laufzeit–Routinen laden diese Programme dann auf die RAP und kontrollieren den Ablauf. Zusätzlich existiert sogar ein Simulator für die RAP selbst, so daß man auch ohne die entsprechende Hardware Programme entwickeln und testen kann.

3.6 Gegenüberstellung

Im Folgenden soll versucht werden, die spezifischen Daten der in diesem Kapitel vorgestellten Systeme einander gegenüberzustellen. Dies ist nicht ganz einfach, da sich die Systeme teilweise in ihrem Operationsprinzip stark unterscheiden. Eine analoge Darstellung der Gewichte mag zu einer theoretisch höheren Genauigkeit und größeren Geschwindigkeit führen, hat aber gegenüber der digitalen Repräsentation den großen Nachteil, daß sich die Gewichte mit der Zeit verändern (zumindest ist das bei den hier vorgestellen Systemen der Fall). Die folgende Übersicht ist also mehr dazu gedacht, einige wichtige Daten noch einmal kurz zusammenzufassen.

Die erste Tabelle stellt die Daten der drei Chips gegenüber:

	CCD–Chip	JPL–Chipsatz[9]	Intel–Chip
Eingänge	144 [10]	32	64
Ausgänge	14 [10]	32	64
Neuronen	keine	32	64
Synapsen	$14 * 144$	1024	10240
Auflösung	6 bit	1–7 bit[11]	ca. 6–7 bit[12]
arithm. Op./sek.	$2,8 * 10^9$	$0,4 * 10^9$	$6,8 * 10^9$
Connections/sek.	$1,4 * 10^9$	$0,2 * 10^9$	$1,4 * 10^9$
Berechnung eines Neurons	100ns [13]	ca. 5μsec	3 μsec
Darst. Gewichte	digital	digital/analog	analog
Darst. Eingang	Spannung	Spannung	Spannung
Darst. Ausgang	Strom	Spannung	Spannung

Es sollte beachtet werden, daß einige der Angaben Maximalwerte darstellen. Insbesondere bei den Ausführungsgeschwindigkeiten führt das dazu, daß der CCD–Chip eine verhältnismäßig hohe Anzahl arithmetischer Operationen pro Sekunde aufweist, die aber tatsächlich nur dann erreicht werden kann, wenn er optimal betrieben wird, also wenn ein einzelner Eingang zu 144 verschieden Zeitpunkten betrachtet werden soll.

Die nächste Tabelle stellt drei der Systeme gegenüber, mit denen auch tatsächlich eine Simulation neuronaler Netze ohne „Lötarbeiten“ möglich ist:

[9] Für diesen Vergleich wird je ein Synapsen– und ein Neuronenchip verwendet.

[10] Alle Ein–/Ausgänge werden über einen physikalischen Anschluß gemultiplext.

[11] Die Auflösung der analogen Variante hängt maßgeblich vom verwendeten D/A–Wandler ab. Es sollen bis zu 16 bit möglich sein.

[12] Die mögliche Auflösung hängt sowohl von der Dauer des geplanten Einsatzes (Informationsverlust), als auch vom Trainingsverfahren ab.

[13] Der CCD–Chip berechnet kein Sigmoid oder eine andere Ausgangsfunktion, sondern nur eine gewichtete Summe.

	ETANN EMB (8 Chips, Intel)	CNAPS (Adaptive Solution)	RAP (Berkeley)
Darstellung der Gewichte	analog (ca. 8 bit)	16 bit Integer	float
Ein-/Ausgabe	analog	digital	digital
Millionen Connections pro Sekunde (forward)	$13,7 * 10^3$ MCPS	$5 * 10^3$ MCPS	50 MCPS
Millionen Connections pro Sekunde (learn)	—	$1 * 10^3$ MCUPS	12 MCUPS
Preis des Systems	\$17.000	\$50.000	\$100.000
Preis pro MCPS	\$1,24	\$10,—	\$2.000,—
Preis pro MCUPS	—	\$50,—	\$8.300,—

Auch bei dieser Tabelle ist zu beachten, daß alle Angaben Spitzenwerte sind. So kann es bei einer — für die jeweilige Maschine — ungünstigen Netzstruktur dazu kommen, daß das System nicht optimal ausgenutzt werden kann, was natürlich zu Leistungseinbußen führt.

Die letzte Tabelle versucht schließlich, einige Daten aller hier betrachteten Systeme einander gegenüberzustellen:

	CCD-Chip	JPL-Chips	ETANN (EMB)	CNAPS	RAP
MCPS	1.400	200	13.700	5.000	50
MCUPS	—	—	—	1.000	12
Gewichte	6 bit	1–7 bit	analog	16 bit	Fließkomma
Taktfrequenz	10MHz	0,2MHz	0,3MHz	20MHz	16MHz
Hostrechner	—	—	PC/AT	SUN	68020
Netzanschluß	Nein	Nein	über PC	Ethernet	Ethernet
Training online	Nein	Nein	Nein	Ja	Ja

Kapitel 4

Gründe für die Wahl des Intel–Chips

In diesem Abschnitt werden die Gründe für die Wahl des schließlich verwendeten Systems vorgestellt. Ein Aspekt ist die Frage, wie gut sich die im vorigen Abschnitt vorgestellten Systeme für die Implementierung des Wordspotters eignen. Weiterhin war die Verfügbarkeit der Systeme sowie Unterstützung durch vorhandene Software von Interesse.

Es fiel schnell auf, daß der CCD–Chip von Alice Chiang nicht sehr gut geeignet ist, um ein TDNN mit nur wenigen time–delays zu realisieren. Durch die Verzögerungsleitung mit 144 Stufen wäre es wesentlich günstiger, auch in dieser Größenordnung time–delays zu verwenden. Andernfalls wird viel Zeit benötigt, um die Verzögerungsleitung — die dann ja nur als Schieberegister verwendet wird — zu laden. Dieser „Flaschenhals“ würde die Rechenleistung in unserem Fall um einen Faktor 30–40 senken[1]. Zweitens ist in diesem System noch kein Neuron enthalten, man müßte hier also noch zusätzliche Hardware verwenden, um ein Sigmoid oder Ähnliches zu berechnen.

Der Chip des Jet Propulsion Laboratory hat diese Nachteile nicht. Hier wäre es leicht möglich, durch Kaskadierung mehrerer Chips ein mehrschichtiges Netz zu realisieren. Allerdings ist die Verwendung von time–delays nicht vorgesehen, es wären also externe, analoge Schieberegister nötig.

[1] Da für den Wordspotter nur max. 5 time–delays benötigt werden

Für beide Systeme exisitert bisher auch keine unterstützende Software, die eine einfache Benutzung erlauben würde.

All diese Probleme beseitigt der Intel–Chip. Durch die Verwendung des "feed–back"–Synapsenfeldes und der Sample&Hold–Glieder kann man ohne weiteres Signale zwischenspeichern und später weiterverwenden. Durch die flexible Struktur ist man sogar in der Lage, mehrschichtige Netzwerke in einem Chip unterzubringen. Ein weiterer Pluspunkt für dieses System ist die Möglichkeit, mehrere Chips auf einem "Multi–Chip–Board" zusammenzufassen. Damit lassen sich auch grössere Netzwerke ohne Probleme simulieren und austesten. Nicht unerwähnt bleiben sollte auch die vorhandene Software, die einen Netzentwurf mit dem Multi–Chip–Board unterstützt.

Die beiden Komplettsysteme, die CNAPS von Adaptive Solutions und die RAP aus Berkeley, waren zum Zeitpunkt dieser Arbeit entweder noch nicht marktreif oder nicht lieferbar. Beide Maschinen wären für den Einsatz als schnelle Trainingsplattform für Intels ETANN geeignet gewesen. Auch eine direkte Gegenüberstellung der Systeme zum ETANN wäre interessant gewesen.

So blieb als einziges System, das sowohl für diese Arbeit geeignet ist als auch erhältlich war, der ETANN–Chip von Intel mit seinem zugehörigen Entwicklungssystem.

Die Hauptgründe für die Entscheidung, den Intel–Chip zu verwenden, waren also:

1. Time–delay–Möglichkeit durch integrierte Sample&Hold–Glieder, die eine externe Speicherung der Signale erübrigen,
2. flexible Struktur, die auch mehrschichtige Netz–Topologien ermöglicht,
3. vorhandenes Multi–Chip–Board, das Kaskadierung mehrerer Chips vereinfacht,
4. vorhandene Software, die die Programmierung des ETANN auf hohem Abstraktions–Niveau ermöglicht,
5. Verfügbarkeit.

Kapitel 5

Der ETANN und seine Entwicklungsumgebung

In diesem Kapitel soll der für diese Arbeit verwendete Chip, der ETANN[1] von Intel, näher vorgestellt werden. Dazu werden zunächst einige interne Block–Strukturen des Chips und deren Zusammenwirken im gesamten System erläutert. Abschließend folgt ein Überblick über die verwendeten Entwurfswerkzeuge.

5.1 Die Synapsen und die Darstellung der Gewichte

Wie schon im vorigen Kapitel kurz angesprochen, werden die Gewichte für die Synapsen in EEPROM ähnlichen Strukturen gespeichert. Wie das Blockschaltbild einer Synapse zeigt, werden pro Gewicht zwei EEPROM–Zellen verwendet (siehe Bild 5.1).

Das hat den Vorteil, daß man ein differentielles Gewicht speichern und damit Temperatur– und andere Umwelteinflüsse einschränken kann. Das

[1]ETANN = Electrically Trainable Analog Neural Network

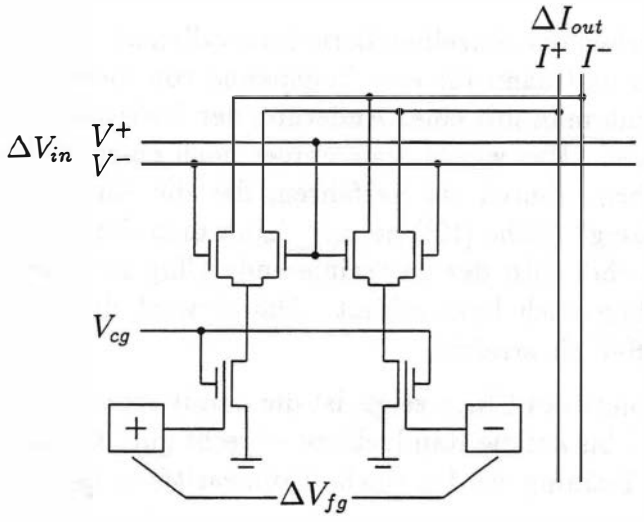


Abbildung 5.1: Gilbert–Multiplizierer wie in Intels ETANN verwendet

tatsächliche Gewicht ist also proportional zur Ladungsdifferenz in den sogenannten "floating gates". Als Eingang dient wieder eine Spannungsdifferenz. Am Ausgang liegt eine Strom–Differenz vor.

Die Programmierung erfolgt ähnlich wie bei einem EEPROM. Durch das Anlegen hoher Spannungen werden Elektronen dazu gebracht, auf oder von einem floating gate zu tunneln. Das gewünschte Gewicht ist damit als Differenz beider Ladungsmengen gespeichert. Ansonsten entspricht die Schaltung einer NMOS–Version des bekannten Gilbert–Multiplizierers (siehe [15]). Einziges Problem dieses Verfahrens ist, daß die Programmierung verhältnismäßig langsam ist. Das liegt in diesem Fall am verwendeten Prinzip, dem Fowler–Nordheim–Tunnel–Effekt, der von Natur aus relativ langsam ist. Daher benötigt man in diesem Falle für die Änderung eines Gewichtes eine Zeit von einigen Mikrosekunden bis zu einer Millisekunde.

Die theoretische Auflösung eines solchen Gewichts liegt bei ca. 30 bit, da durch das Hinzufügen eines Elektrons schon eine Änderung möglich wäre. Dies ist aber in der Praxis nicht möglich — dazu müßte die Dauer und Spannung der Programmierpulse hochpräzise kontrollierbar sein — und so erreicht man nur eine Auflösung von ca. 7 bit. Das entspricht einer Änderung der Threshold–Spannung der floating–gate–Transistoren von etwa 22mV.

Die Langzeitgenauigkeit dieser Gewichte hängt natürlich auch von den

Abständen zwischen den einzelnen Lern–Intervallen ab. Soll der Chip einmal trainiert werden und dann für eine Zeitspanne von mehreren Jahren eingesetzt werden, muß man mit einer Änderung der Gewichte in einem Rahmen von 6–7% rechnen. Das würde aber immer noch einer Auflösung von etwa 4 bit[2] entsprechen. Durch ein Verfahren, das die Autoren treffenderweise "Bake Re–Training" (siehe [12]) nennen, kann man die Auflösung allerdings verbessern. Hierbei wird der zu trainierende Chip zwischen den einzelnen Trainigsintervallen noch kurz erhitzt. Damit wird dann eine tatsächliche Auflösung von 6–7 bit erreicht.

Wie das Datenblatt des Chips zeigt, ist die damit erreichte Linearität dieser Multiplizierer — bis auf die Randgebiete — recht gut. Durch das verwendete "chip in loop"–Training werden solche Unlinearitäten darüberhinaus wieder ausgeglichen.

5.2 Die Neuronen

Die Neuronen bestehen aus einem Addierer, der die differentiellen Ströme der Synapsen aufsummiert und daraus eine differentielle Spannung erzeugt, und einer elektronischen Emulation eines Sigmoids. Die Steigung des Sigmoids ist durch eine externe Referenzspannung in bestimmten Grenzen beeinflußbar, wie das entsprechende Schaubild des Datenblattes zeigt.

5.3 Ein–/Ausgänge

Sowohl die Ein– als auch die Ausgänge sind analog ausgeführt, um einen Informationsverlust zu vermeiden. Um die Zahl der Anschlüsse niedrig zu halten, sind sie asymmetrisch gehalten. Damit sind dann Buffer erforderlich, die die nötigen Umwandlungen durchführen. Am Eingang muß die Spannung in eine Differenz–Spannung umgewandelt werden und am Ausgang entsprechend umgekehrt. Das hat den Vorteil, daß man über einen externen Referenz–Anschluß den Nullpunkt sowohl für die Ein– als auch für die Ausgänge festlegen kann. Damit kann man nur positive Eingänge realisieren

[2] 6–7% Toleranz führen zu 14–17 unterscheidbaren Zuständen

oder TTL–kompatible Eingänge erhalten. Und durch einen zusätzlichen Eingang, der an die Neuronen führt, ist es sogar möglich, den Nullpunkt der Neuronen mit dem Nullpunkt der Eingänge abzugleichen.

5.4 Der Chip im Überblick

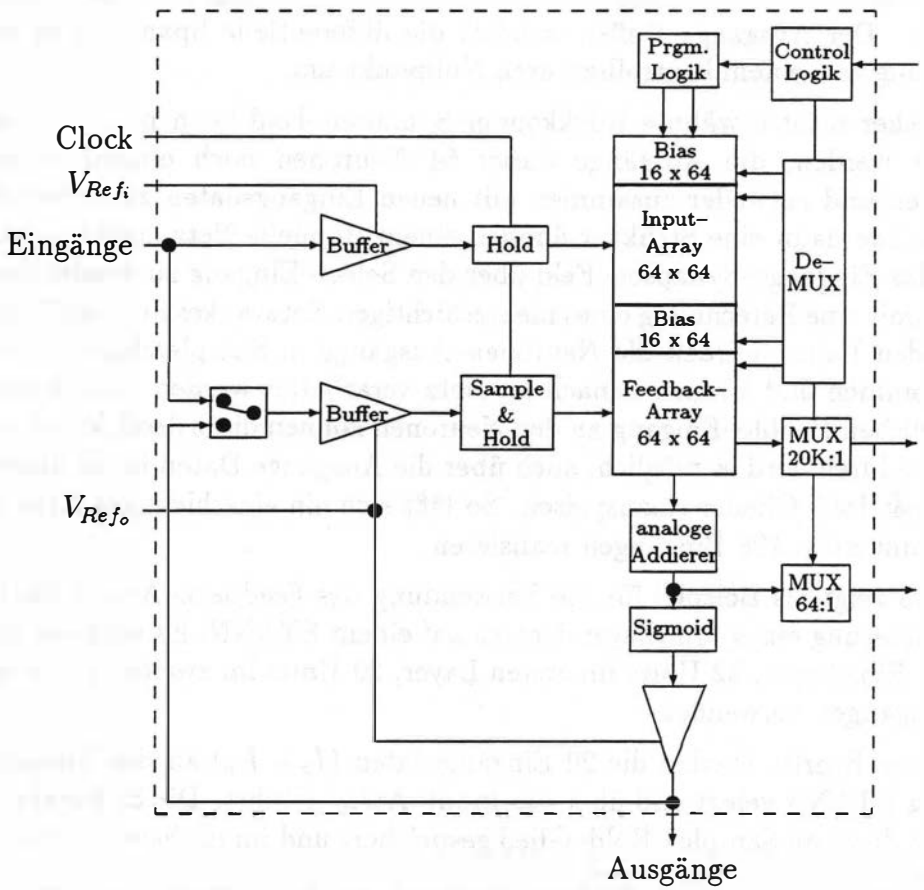


Abbildung 5.2: Blockschaltbild des ETANN

Bild 5.2 zeigt das Zusammenspiel der vorgestellten Komponenten. Die 64 analogen Eingänge werden in den Eingangs–Buffern in differentielle Spannungen umgewandelt (mit V_{Ref_i} als Referenzspannung) und in einem Sample&Hold–Glied zwischengespeichert. Danach werden diese Eingänge an das erste Synapsen–Feld weitergeleitet. Hier sind zusätzlich zu den 4096 Synapsen, die benötigt werden, um 64 Eingänge auf 64 Ausgänge vollständig zu verbinden, noch weitere 16 Synapsen pro Ausgang vorhanden, die auf einem festen Potential liegen. Damit lassen sich Bias–Eingänge für die Neuronen realisieren[3]. In den Synapsen werden als Ergebnis Ströme erzeugt, die danach aufsummiert und wieder in eine Spannung umgewandelt werden. Und zum Schluß durchlaufen diese Spannungen das Ausgangs–Sigmoid, dessen Steigung und Nullpunkt durch zwei externe Anschlüsse kontrolliert werden. Der Ausgangs–Buffer wandelt die differentielle Spannung in eine Spannung mit einem kontrollierbaren Nullpunkt um.

Das bisher nicht erwähnte Rückkoppel–Synapsen–Feld kann nun dazu verwendet werden, die Ausgänge dieser 64 Neuronen noch einmal zurückzuführen und entweder zusammen mit neuen Eingangsdaten zu verwenden (das würde dann eine Struktur ähnlich einem Hopfield–Netz ergeben), oder auch das Eingangs–Synapsen–Feld über den Select–Eingang zu deselektieren und damit eine Berechnung eines mehrschichtigen Netzwerkes durchzuführen. In beiden Fällen werden die Neuronen–Ausgänge in Sample&Hold–Glieder übernommen und können danach im Netz verarbeitet werden. Durch einen zusätzlichen Enable–Eingang an den Neuronen können diese deselektiert werden. Dadurch wird es möglich, auch über die Ausgänge Daten in die unteren Sample&Hold–Glieder einzuspeisen. So läßt sich ein einschichtiges Netz mit insgesamt max. 128 Eingängen realisieren.

Bild 5.3 zeigt als Beispiel für die Verwendung des Feedback–Arrays die Implementierung eines Multilayer–Netzes auf einem ETANN. Es wird ein Netz mit 20 Eingängen, 32 Units im ersten Layer, 20 Units im zweiten Layer und 12 Ausgängen verwendet.

Im ersten Schritt werden die 20 Eingangsdaten ($I_0 - I_{19}$) an den Eingangsbus des ETANN gelegt und über das Input–Array geführt. Die 32 Ergebnisse werden dann im Sample&Hold–Glied gespeichert und im nächsten Schritt als

[3]Für einen Bias werden 16 Synapsen verwendet, um den verwendbaren Bereich der Bias–Units zu vergrößern, die damit im Intervall $[-40, +40]$ statt nur $[-2.5, +2,5]$ liegen können.

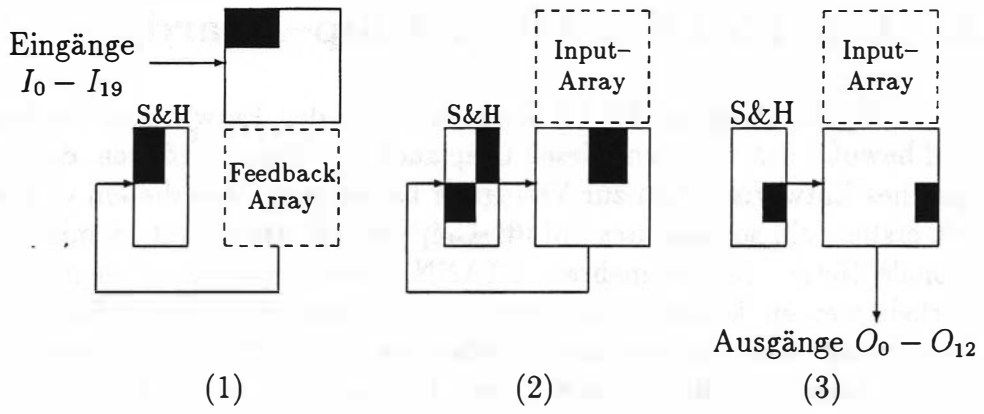


Abbildung 5.3: Ein 20–32–20–12 Netz auf einem ETANN

Eingabe für das Feedback–Array benutzt, dessen 20 Ausgaben wieder zwischengespeichert werden. Im letzten Schritt berechnet das Feedback–Array schließlich die 12 Ausgaben ($O_0 - O_{11}$). Der besseren Übersichtlichkeit halber wurde in diesem Beispiel darauf verzichtet, ein Pipelining der Eingaben darzustellen. Selbstverständlich ist es aber möglich, schon im zweiten Schritt wieder neue Eingaben an den Eingangsbus anzulegen, um den Chip besser auszunutzen.

Erwähnenswert ist auch die Tatsache, daß sämtliche Gewichte einzeln auslesbar und auch einzeln programmierbar sind. Damit wird erst das „Nachlernen" der Synapsen möglich, welches durch die Software unterstützt wird. Dabei werden nach einer Initialisierung der Gewichte (die Daten dafür stammen aus einer Software–Simulation) die fertigungsbedingten Toleranzen durch einen Test der Funktion des Netzes und gegebenenfalls durch ein Nachprogrammieren der Gewichte ausgeglichen. Dazu werden weitere Trainingsläufe durchgeführt, bei denen die Daten des Forward–Passes von dem jeweiligen Chip berechnet werden. Der verwendete Trainingsalgorithmus (in diesem Falle Back–Propagation, siehe auch Anhang A) paßt die Gewichte des Netzes dann an die Unregelmäßigkeiten des Chips an.

Die Programmierlogik funktioniert ähnlich wie bei einem EEPROM. Durch Anlegen kurzer Impulse mit variabler Spannung wird erreicht, daß die gewünschte Ladungsmenge auf oder von einem floating gate fließt.

5.5 Das ETANN Multi–Chip–Board

Nach der Entwicklung der ETANN–Chips wurde den Entwicklern bei Intel schnell bewußt, daß man, um diesen Chip auch einsetzen zu können, ein umfangreiches Entwurfssystem zur Verfügung haben muß. Aus diesem Grunde wurde erstens ein sogenanntes "Multi–Chip–Board" entwickelt, damit auch Neuronale Netze, die auf mehrere ETANN–Chips verteilt werden müssen, entwickelt werden können. Zweitens wurden zwei bestehende Software–Pakete zur Entwicklung Neuronaler Netze erweitert, um die Eigenarten des ETANN–Chips zu berücksichtigen. Diese Programme werden in einem eigenen Abschnitt näher vorgestellt.

Das ETANN Multi–Chip–Board (kurz EMB) bietet bis zu 8 ETANN–Chips Platz. Durch eine geschickte Bus–Logik wird erreicht, daß diese Chips beliebig miteinander verbunden werden können, ohne daß diese Verbindungen tatsächlich physikalisch vorhanden sein müssen. Das ermöglicht den einfachen Test Neuronaler Netze, die auf mehrere Chips verteilt sind. Nachdem sich ein solches Netz bewährt hat, kann es auch fest verdrahtet werden. Dafür existieren spezielle "jumper"–felder, die es erlauben, jeden beliebigen dieser 8 Chips mit jedem anderen zu verbinden. Bei Intel wurde mit diesem Konzept ein Neuronales Netz aufgebaut, welches mit 8 ETANN–Chips Buchstaben erkennen kann (siehe hierzu [13]). Insgesamt können so Netze mit bis zu 512 Neuronen und 64k Verbindungen aufgebaut werden. Bis zum jetzigen Zeitpunkt sind aber noch keine Untersuchungen über Einflüsse des Bus–Systems auf die übertragenen Daten bekannt.

5.6 Programmieren des ETANN

Als Entwicklungsumgebung für den ETANN ist von Intel ein IBM kompatibler PC/AT vorgesehen, der mit der "PCPP Universal Programmer"–Karte ausgerüstet ist. Daran wird durch ein Flachbandkabel das entsprechende Programmiergerät angeschlossen, auf das ein ETANN–spezifischer Aufsatz mit einem 208–poligen PGA–Nullkraft–Sockel gesteckt ist. Das Programmiergerät stellt die benötigten hohen Spannungen zur Verfügung, um die Synapsen des ETANN zu programmieren und erlaubt gleichzeitig die Verwendung von 64 Digital/Analog–Wandlern und einem Analog/Digital–Wandler.

Die Wandler werden für den Funktionstest des ETANN verwendet um Eingangsdaten an den Chip anzulegen und die Ausgänge einzulesen. Zusätzlich kann so auch das "chip in loop"–Training durchgeführt werden, durch das eine Anpassung der Gewichte des Chips an seine speziellen Eigenheiten erreicht wird. Dabei werden die Daten der Software–Simulation an den Chip angelegt und die Ausgaben des Chips mit den gewünschten Werten verglichen. Liegen Abweichungen vor, kann mit einem Trainingsschritt die Gewichtsmatrix des Chips so verändert werden, daß interne Ungenauigkeiten kompensiert werden. In der Praxis wird dazu das verwendete Trainingsverfahren, nach zufriedenstellender Konvergenz der Software Simulation, auf die Gewichte des Chips angewendet, bis auch dieser entsprechend gute Ergebnisse liefert.

Klar ist, daß durch die verwendeten Wandler die volle Leistung des ETANN nie ausgenutzt werden kann (insbesondere müssen die Ausgänge nacheinander eingelesen werden, da nur ein Analog/Digital–Wandler vorhanden ist) solange die Programmierumgebung verwendet wird. Will man die Geschwindigkeit des ETANN also ausnutzen, muß man ihn in einer entsprechenden Umgebung einsetzen. Trotz dieses Flaschenhalses ist bei größeren Netzen die Software Simulation[4] teilweise langsamer, als bei einer Verwendung des ETANN für den Forward–Pass.

5.7 Die Software

Die von Intel mitgelieferte Software besteht aus zwei kommerziellen Entwurfssystemen für Neuronale Netze, die um einige Funktionen erweitert wurden, um spezielle Eigenschaften des ETANN und des Multi–Chip–Boards zu unterstützen. Dazu gehört die Möglichkeit das "chip in loop"–Training durchzuführen, was zu einer besseren Anpassung an die Toleranzen der verschiedenen Chips führt.

Das bekanntere der beiden Systeme ist sicherlich BrainMaker, das auf dem PC–Markt als Simulator für Neuronale Netze schon lange angeboten wird. In der Version, die den ETANN unterstützt (das Programm heißt dann „iBrainMaker"), ist noch ein zusätzlicher Menüpunkt hinzugekommen, der es er-

[4]Verwendet wurde ein 50Mhz PC mit einem Intel 486 Prozessor

laubt, trainierte Netze auf den Chip zu laden. Zusätzlich kann nun ausgewählt werden, ob man ein normales Training durchführen will, oder ob das "chip in loop"–Training die Anpassung an den jeweiligen Chip durchführen soll. Das Feedback–Array des ETANN wird bei iBrainMaker nur zur Realisierung weiterer Layer eines Netzes verwendet. Die Möglichkeit von Rückkopplungen ist bei iBrainMaker nicht vorgesehen.

Dynamind (oder in der ETANN–Version „iDynamind") bietet ähnliche Möglichkeiten wie iBrainMaker, unterstützt aber zusätzlich auch noch rekurrente Netze, die auf einen Layer beschränkt sind. Hier wird Truetime als Traingsverfahren eingesetzt, es besteht so auch die Möglichkeit, das Feedback–Array des ETANN anderweitig zu verwenden.

Beide Programme ermöglichen jedoch keine Verwendung der eingebauten Sample&Hold–Glieder des ETANN um Time–Delays zu realisieren. Eine Verwendung des Feedback–Arrays als Zwischenspeicher ist nicht vorgesehen.

Kapitel 6

Der Word Spotter

Die Wahl des zu realisierendes Systems fiel auf den "word spotter" von Zeppenfeld&Waibel, der an der CMU entwickelt wurde. Dieses System kann in kontinuierlicher Sprache sprecherunabhängig einzelne Schlüsselwörter erkennen (siehe hierzu auch [17]). Die Grundidee besteht darin, die Wörter als Sequenz von akustischen Zuständen zu modelieren, die mit Hilfe eines neuronalen Netzes erkannt werden können. Dann wird versucht, die Folge dieser Zustände als Wort zu erkennen. Da der Datensatz, der zum Training verwendet wird, nur eine Kennzeichnung des Anfangs und des Endes eines Wortes enthält, kann diese Unterteilung nicht auf einer Phonem–Ebene geschehen, sondern es wird versucht, die Grenzen zwischen verschiedenen Zuständen innerhalb eines Wortes auch zu trainieren. Im Folgenden wird die Struktur und das Trainingsverfahren dieses Systems näher erläutert.

6.1 Die Signalverarbeitung

Eingabe für das gesamte System ist ein analoges Signal, das von einem Mikrophon aufgenommen wird. Dieses wird zuerst digitalisiert und quantisiert, so daß in diesem Fall 16–bit–Worte mit einer Frequenz von 10kHz vorliegen. Das geschieht mit Hilfe eines AD–Wandlers[1]; die digitalen Werte wer-

[1] Verwendet wurde ein Desklab von Gradient Tech.

den dann im Computer weiterverarbeitet. Zunächst werden jeweils 256 Werte mit einem Hamming–Window multipliziert, und das Ergebnis einer diskreten Fast–Fourier–Transformation unterworfen. Die Ausgabe dieser Stufe sind jeweils 128 Frequenz–Amplituden (und Phasen), die in einem Abstand von 12,8 msec aufeinander folgen[2]. Aus diesen Amplituden und Phasen wird ein sogenanntes Leistungsspektrum berechnet, wobei die Phasen–Information wegfällt (die keine Bedeutung für phonetische Erkennung hat) und nur noch die Amplitude der verschiedene Frequenzen vorliegt. Danach werden die Frequenzen in einer dem menschlichen Ohr angepaßten Weise zu 16 sogenannten Melscale–Koeffizienten zusammengefaßt. Ein solcher Koeffizient besteht dabei aus dem Zehner–Logarithmus der Summe mehrerer Frequenzamplituden:

$$mel_i = \log_{10}\Big(\sum_{\omega=min_i}^{max_i} \beta_{\omega,i} \sqrt{\mathcal{F}(\omega)^2} \quad \Big)$$

$$\text{mit} \quad \beta_{\omega,i} = \begin{cases} \frac{1}{2} & : \quad \omega = min_i \quad \text{und} \quad i \neq 1 \\ \frac{1}{2} & : \quad \omega = max_i \quad \text{und} \quad i \neq 16 \\ 1 & : \quad sonst \end{cases}$$

$$\text{und} \quad min_{i+1} = max_i \quad , \quad 1 \leq i < 16$$

Der Logarithmus wird verwendet, um das logarithmische Lautstärke–Empfinden des Menschen zu berücksichtigen. Die Frequenzbereiche werden zu höheren Frequenzen hin immer breiter, was der Frequenz–Empfindlichkeit des Ohres entspricht. $\beta_{\omega,i}$ wird verwendet, um eine gleichmäßige Gewichtung aller Frequenzen zu gewährleisten, da die Frequenzen am Rande eines Bereiches jeweils zu zwei Melscale–Koeffizienten gehören. Die jetzt vorhandenen Daten bestehen also aus 16 derartigen Koeffizienten, die mit einer Frequenz von 100Hz vorliegen. Diese werden direkt als Eingabe in ein neuronales Netz verwendet.

[2]Das Fenster wird jeweils um 128 Werte verschoben.

6.2 Das Neuronale Netz

Eine Besonderheit dieses Netzes ist die Verwendung von "time delays", sprich, es werden nicht nur Koeffizienten zu einem, sondern zu mehreren Zeitpunkten betrachtet. Die Struktur dieses Netzes zeigt Bild 6.1.

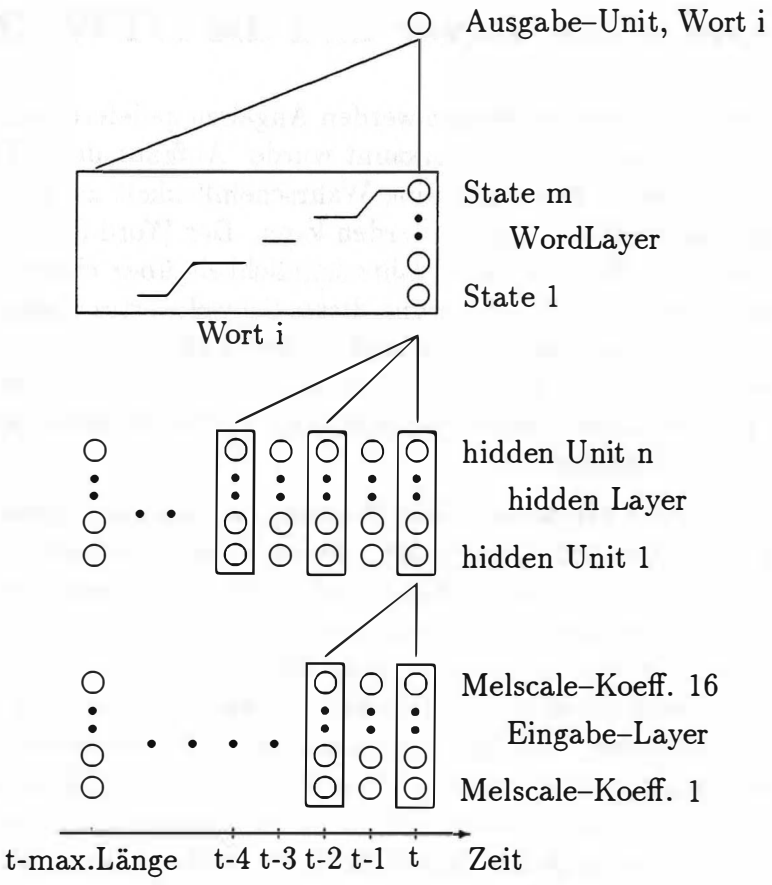


Abbildung 6.1: Die Struktur des WordSpotters

Die 16 Melscale–Koeffizienten werden also zu zwei, um zwei Abtastzeitpunkte versetzten Zeitpunkten als Eingabe für das Netz verwendet. Die Ausgabe des ersten Layers wird dann zu drei Zeitpunkten, wieder jeweils um zwei

Einheiten versetzt, an den zweiten Layer weitergereicht.

Darauf setzt für jedes zu erkennende Wort ein sogenannter "state layer" auf. Dieser Teil besteht aus einem letzten neuronalen Netz–Layer, der die für dieses Wort wichtigen Zustände erkennt, und einer DTW–Routine[3], die versucht, eine korrekte Folge der Zustände zu erkennen.

6.3 Die State–Layer und die DTW–Routine

Von dem letzten Layer des Netzes werden Angaben geliefert, wie gut ein Zustand eines bestimmten Wortes erkannt wurde. Aufgabe der DTW–Routine ist es nun, zu jedem Zeitpunkt eine Wahrscheinlichkeit zu liefern, mit der das entsprechende Wort erkannt werden kann. Der Word Spotter „erkennt" dann einfach das Wort, dessen Wahrscheinlichkeit über einem bestimmten Schwellwert liegt. Je nach Justierung dieses Schwellwertes erkennt der Spotter zwar viele Wörter richtig, aber auch relativ viele „Wörter", die gar nicht vorhanden waren. Oder man erkennt nicht alle Wörter, dafür aber fast nur richtige. Die Wahl des Schwellwertes ist also (unter anderem) entscheidend für eine gute Erkennungsrate.

Die DTW–Routine (Dynamic Time Warping) ist demnach dafür zuständig, eine Folge von Zuständen zu finden, die zu diesem Zeitpunkt einen optimalen Pfad darstellen, d.h. in einem optimalen Score resultieren. Um die Erkennungsrate des Spotters zu verbessern, gibt es für das Auffinden dieses Pfades einige Restriktionen. So ist zum Beispiel vorgeschrieben, daß in jedem Zustand zumindest drei Zeitpunkte verweilt werden muß. Weiterhin werden während des Trainings die minimalen und maximalen Längen der Worte festgestellt und als weitere Vorschriften in die DTW–Routine eingebracht. Und schließlich wird bei der Suche nach einem günstigen Pfad noch berücksichtigt, wie lange man schon in einem Zustand war ("state duration penalty") und ob der Übergang in einen anderen Zustand sehr ungünstig ist ("state transition penalty"). Der folgende Algorithmus soll das Vorgehen zum Finden des Scores für ein Wort zusammenfassen:

[3]DTW = Dynamic Time Warping

```
 1: frames X: 0 ... max.length
 2: states i: 1 ... m
 3:   score_A = score[state i, frame X-1]
                + activity[state i, frame X]
 4:   score_A += length_penalty * state_length[state i]
 5:   score_B = score[state i-1, frame X-1]
                + activity[state i, frame X]
 6:   pen = activity[state i, frame X] * transition_penalty
 7:   if (pen > 0) pen = 0
 8:   score_B += pen
 9:   if (score_A > score_B)
10:      score[state i, frame X] = score_A
11:      state_length[state i] ++
12:    else
13:      score[state i, frame X] = score_B
14:      state_length[state i] = 1
15: return
      max{score[last_state, frame X] | min.length < X < max.length}
```

Zu Beginn werden die Scores der in Frage kommenden Vorgänger„pfade“ aus der Scorematrix geholt und zu beiden wird die aktuelle Aktivität addiert (Zeilen 3 und 5) . Zusätzlich wird zu dem Score des aktuellen Zustands noch die “duration penalty” addiert, um zu verhindern, daß sich der Pfad zu lange in einem Zustand aufhält (Zeile 4). Zum Score des anderen Pfades (der an dieser Stelle den Zustand wechseln würde) wird eine “state transition penalty” addiert (Zeile 8), die sich aus der aktuellen Aktivität des neuen Zustands berechnet (Zeile 6). Allerdings wird diese “penalty” nur addiert, wenn sie negativ ist (Zeile 7). Ein Vergleich der beiden Scores ergibt dann, welcher der bessere ist und aktualisiert die Zähler für die “state durations” entsprechend (Zeilen 9–14). Als Ergebnis der Routine dient dann der maximale Score für eine Pfadlänge, die die Längenrestriktionen des entsprechenden Wortes erfüllt.

Die jeweiligen “penalties” sind für alle Wörter gleich, nur die minimale und maximale Wortlänge wird aus den Trainingsdaten für jedes Wort seperat ermittelt.

6.4 Worterkennung

Die Erkennung einzelner Wörter ist nun kein großes Problem mehr. Es muß nur noch ein geeigneter Schwellwert festgelegt werden, der definiert, ab wann ein Ausgangsscore groß genug ist, um das Wort als „erkannt“ zu melden. Das heißt, im Erkennungsfall wird einfach ein Forwardpass durchgeführt und alle Ausgänge werden auf Überschreitung dieses Grenzwertes überprüft. Um einzelne Fehler auszusortieren, wird als Randbedingung noch gefordert, daß diese Überschreitung zu drei aufeinanderfolgenden Zeitpunkten vorliegen muß. Ist dies der Fall, wurde das entsprechende Wort „erkannt“. Die Wahl des Schwellwertes ist hierbei nicht unkritisch, sie bestimmt schließlich das Verhältnis von „guten“ zu „schlechten“ Erkennungen. Das heißt, ein zu niedriger Schwellwert führt zu vielen “false alarms”, sprich Erkennen von Wörtern, die gar nicht vorliegen. Umgekehrt bedeutet ein zu hoher Schwellwert eine Verschlechterung der Erkennung von tatsächlich vorliegenden Wörtern (siehe hierzu auch den Abschnitt über die Leistungsmessung).

6.5 Das Training des Word Spotters

Wie oben schon erwähnt, sind in den Trainingsdaten nur Markierungen für den Anfang und das Ende der jeweiligen zu lernenden Wörter vorhanden. Deshalb besteht das Training des Spotters aus zwei Teilen, dem sogenannten “bootstrapping” und dem darauf folgenden adaptiven Training.

Während des Bootstrap–Trainings werden die Wörter in gleichlange Stücke aufgeteilt und das TDNN wird so trainiert, daß zu Beginn eines Wortes der erste und am Ende der letzte Zustand erkannt wird. Bild 6.2 zeigt am Beispiel eines 15 Frame langen Wortes, das auf 5 Zustände verteilt werden soll, den entsprechenden Pfad. Es werden also in Trainingsfragmenten, in denen ein Wort vorhanden ist, Ausgänge des Netzes, die nicht auf diesem „Pfad“ liegen negativ trainiert. Die Ausgänge, die der Zustandsfolge dieses Wortes entsprechen, erfahren ein positives Training.

Das adaptive Training erfolgt nun ziemlich analog zum Training eines normalen Netzes. Das Netz wird getestet und an Stellen eines “false alarms”, also einem Erkennen eines Wortes, das aber an dieser Stelle gar nicht vorhanden ist, wird dieser falsche Pfad negativ trainiert. In Bereichen, in denen

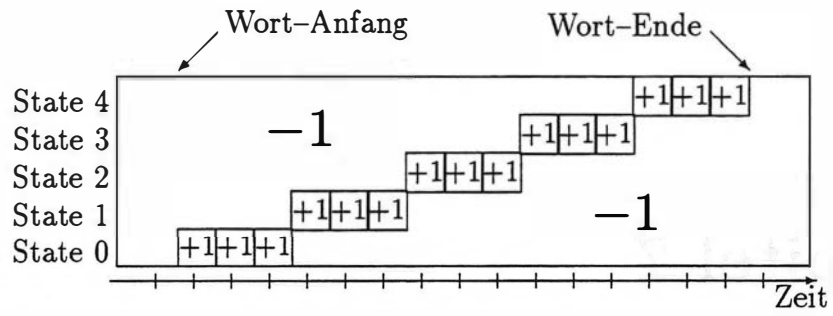


Abbildung 6.2: Bootstrap–Training eines Wortes

das Netz ein Wort erkennen soll, wird der entsprechende Pfad gesucht und diesmal positiv trainiert. Dadurch wird die feste Fixierung der einzelnen Zustände auf bestimmte Wortteile teilweise wieder aufgehoben.

6.6 Leistungsmessung

Die Leistung eines Wordspotters anzugeben ist, wie man sich leicht klar machen kann, keine eindeutige Aufgabe. Was soll man von einem Spotter halten, der zwar 100% aller Worte erkennt, aber dafür auch alle 2 Sekunden ein Wort fälschlicherweise meldet? Ist es besser die Zahl anzugeben, wieviele Wörter erkannt werden, wenn keine einzige falsche Erkennung erlaubt ist?

Für diesen Zweck gibt es ein Verfahren, das versucht, diesem Problem gerecht zu werden. Man ermittelt eine Zahl p_i, die den Prozentsatz der richtig erkannten Wörter angibt, wenn man den Threshold so verschiebt, daß genau i falsche Erkennungen vorliegen. Je größer i, desto „besser“ wird natürlich auch die Zahl p_i. Man kann nun die ersten zehn dieser p_i addieren und noch zusätzlich zeitlich normieren — die Länge des Testdatensets sollte ja auch einfließen — und erhält so die sogenannte “figure of merit” (FOM):

$$FOM = (p_1 + p_2 + \cdots + p_N + a * p_{N+1})/(10 * T)$$

T ist dabei die Dauer des Testsets, angegeben in Vielfachen von Stunden. $N = \lceil 10T - 1/2 \rceil$ bestimmt die Anzahl der Glieder dieser Summe und $a = 10T - N$ ist ein Korrekturfaktor für den letzten Summanden.

Kapitel 7

Ein Word Spotter auf dem ETANN

Bedingt durch die Struktur des Erkenners kann die Aufgabe, dieses System in Hardware zu implementieren in drei Teile aufgeteilt werden. Der erste Schritt besteht darin, ein Time Delay Neural Network auf dem ETANN zu realisieren. Danach muß ein Weg gefunden werden, die DTW–Routine durch ein Neuronales Netz zu ersetzen, um auch diese auf den Chips laufen zu lassen. Eine weitere Aufgabe ist die Berechnung der DFT auf dem ETANN. Die nachfolgenden Abschnitte beschreiben Probleme und Lösungsideen für diese drei Aufgaben und führen schließlich zu einem Kapitel, in dem die tatsächliche Implementierung des Wordspotters beschrieben wird und Resultate aufgeführt sind.

7.1 Diskrete Fourier–Transformation auf dem ETANN

Einer der Gedanken bei einer Implementierung eines Spracherkenners mit Hilfe des ETANN ist es, auch noch Teile der digitalen Signalverarbeitung auf dem ETANN zu erledigen. Ein Kandidat dafür ist sicherlich die Diskrete Fourier–Transformation, die ja auch bei Software–Simulationen einen großen Teil der Rechenzeit verschlingt. In diesem Abschnitt werden einige Mög-

lichkeiten vorgestellt, wie eine Fourier–Transformation mit Hilfe eines Neuronalen Netzes berechnet werden kann. Besondere Aufmerksamkeit wird einer möglichen Implementierung auf Intels ETANN gewidmet. Deshalb werden Methoden, die Netze mit exotischen Ausgangsfunktionen verwenden, hier nicht weiter untersucht.

7.1.1 Theoretische Grundlagen der DFT

Betrachtet man sich die theoretischen Grundlagen der DFT ein bißchen genauer und versucht, die geläufige imaginäre Schreibweise aufzulösen, so daß man nur noch cos– und sin–Terme erhält, ergibt das:
(die Funktion $f(x)$ wird im Intervall $0 \leq x < L$ an N (N gerade) Stellen betrachtet, also $f(x_k)$ mit $x_k = \frac{kL}{N}$, $k = 0, 1, ..., N-1$)

$$f(x_k) = a_0 + \sum_{i=1}^{\frac{N}{2}} \left(a_i \cdot cos\left(2\pi \frac{x_i \cdot k}{L}\right) + b_i \cdot sin\left(2\pi \frac{x_i \cdot k}{L}\right)\right)$$

mit:

$$a_i = \frac{2}{N} \sum_{k=0}^{N-1} \left(f(x_k) \cdot cos\left(2\pi \frac{k \cdot i}{N}\right)\right)$$

und

$$b_i = \frac{2}{N} \sum_{k=0}^{N-1} \left(f(x_k) \cdot sin\left(2\pi \frac{k \cdot i}{N}\right)\right)$$

Umformung in Matrixschreibweise führt für die Koeffizienten a_i (b_i analog) zu:

$$\begin{pmatrix} a_0 \\ \vdots \\ a_{\frac{N}{2}-1} \end{pmatrix} = \frac{2}{N} \begin{pmatrix} cos(2\pi \frac{0 \cdot 0}{N}) & \cdots & cos(2\pi \frac{(N-1) \cdot 0)}{N}) \\ \vdots & \ddots & \vdots \\ cos(2\pi \frac{0 \cdot (\frac{N+1}{2}-1)}{N}) & \cdots & cos(2\pi \frac{(N-1) \cdot (\frac{N}{2}-1)}{N}) \end{pmatrix} \begin{pmatrix} f(x_0) \\ \vdots \\ f(x_{N-1}) \end{pmatrix}$$

Das Ganze ist also nichts weiter als eine Matrix–Vektor Multiplikation (oder eine gewichtete Summe). Und das ist eine der Hauptaufgaben neuronaler Netze — es sollte also keine Probleme bereiten, die Berechnung dieser Koeffizienten einem neuronalen Netz zu überlassen. Die einzige Forderung ist eine lineare Ausgangsfunktion. Zur Berechnung einer 256–Punkt–DFT benötigt

man also eine Netz mit nur einem Layer, 256 Eingängen (für die Funktionswerte $f(x_k)$) und 256 Ausgänge (je 128 für die Koeffizienten a_i und b_i). Die Gewichte an den jeweiligen Verbindungen lassen sich einfach mittels:

$$w_{ij} = \begin{cases} \frac{2}{N} cos(2\pi \frac{ij}{N}) & : \quad 0 \leq j < 256 \\ \frac{2}{N} sin(2\pi \frac{ij}{N}) & : \quad 256 \leq j \end{cases} \quad \text{für } 0 \leq i < 256 \wedge 0 \leq j < 512$$

berechnen.

Ein Problem bei der Übertragung dieser Gewichte auf den ETANN wird schnell klar: Die auf maximale 128 beschränkte Eingangszahl und (in diesem Modus) dann auch nur 64 mögliche Ausgänge. Da für die Berechnung ein vollständig verbundenes Netz erforderlich ist, kann hier durch ein Zusammenschalten mehrerer Chips auch nichts erreicht werden. Man ist also auf eine 128–Punkt DFT beschränkt, muß dann aber 2 Chips verwenden um sowohl die sin– als auch die cos–Koeffizienten zu berechnen. (Man kann die 128–Punkt Einschränkung natürlich umgehen, indem man weitere 128–Punkt DFTs mit anderen „Auflösungen“ berechnen läßt). Da die DFT im Prinzip symmetrisch in ihrer Ausgabe ist, wird nur die Hälfte der Ausgänge benötigt und man kann auf einem ETANN einen kompletten Koeffizientensatz einer 128–Punkt DFT berechnen.

Eine Implementierung auf dem ETANN brachte dann auch erstaunlich gute Ergebnisse. Vor allem die Geschwindigkeit kann sich sehen lassen. Bedingt durch den 128–Input–Modus muß der Chip zwar einmal getaktet werden, bis die Ergebnisse vorliegen, aber es kann immer noch eine Rate von 6μsek pro DFT erreicht werden (das entspricht einer Rechenleistung von etwa 2000 Millionen Fließkomma–Operationen pro Sekunde (2GFLOP), wenn man außer Acht läßt, daß sich die DFT auf einem Rechner mittels z.B. des Butterfly–Verfahrens etwas „günstiger“ berechnen ließe).

Schade ist nur, daß in der Spracherkennung eine reine DFT eigentlich nicht benötigt wird. Meistens wird zumindest ein Leistungsspektrum verlangt (also die Amplituden der verschiedenen Frequenzanteile) was sich aus der normalen DFT ja leicht durch eine (imaginäre) Betragsbildung erhalten läßt ($a \cdot cos^2(\phi) + a \cdot sin^2(\phi) = a$). Nun ist die Verwandtschaft zum neuronalen Netz allerdings nicht mehr so offensichtlich. Eine Multiplikation zweier Eingangsgrößen ist mit normalen Netzen (sprich: keine ausgefallenen Ausgangsfunktionen, sondern nur einfache Sigmoide) aus dem Stegreif nicht möglich.

7.1.2 Leistungsspektrum mit dem ETANN

Man kann versuchen, über Umwege an ein approximiertes Ergebnis zu gelangen, indem man nicht nur die Sinus– und Cosinus–Koeffizienten berechnet sondern noch zusätzliche Koeffizienten, die mit einem anderen Phasenversatz gegenüber den Cosinus–Koeffizienten berechnet werden (dann erhält man die Sinus–Koeffizienten durch eine Phasenverschiebung von $\pi/2$). Durch Addition der Beträge genügend vieler dieser Koeffizienten kommt man auf eine relativ gute Näherung für die Amplitude der entsprechenden Frequenz:

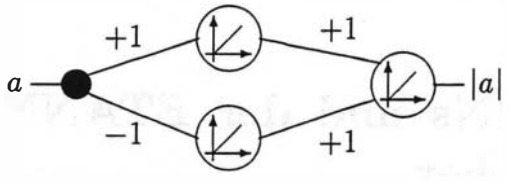
$$c_{i,\phi} = \frac{2}{N} \sum_{k=0}^{N-1} \left(f(x_k) \cdot cos \left(2\pi \frac{k \cdot i}{N} + \phi \right) \right)$$

und

$$\text{Amplitude}_i = \frac{1}{z_M} \sum_{k=0}^{M-1} c_{i,\frac{\pi \cdot k}{2M}}$$

Der Fehler dieser Approximation hängt von der Anzahl M der berechneten phasenverschobenen Koeffizienten ab.

Nun muß also „nur“ noch eine Betragsfunktion auf einem neuronalen Netz implementiert werden. Mit Sigmoid–Funktionen läßt sich das nicht besonders gut realisieren, durch den frei wählbaren Nullpunkt des Sigmoids bei dem ETANN ist allerdings eine geschickte Lösung implementierbar, Bild 7.1 zeigt das entsprechende Netzwerk.



Übertragungsfunktion der Neuronen: $out(x) = \begin{cases} x & : \ x \geq 0 \\ 0 & : \ x < 0 \end{cases}$

Abbildung 7.1: Das Netz zur Berechung des Absolutbetrags

Es wird sofort klar, daß diese Art der Realisierung auf dem ETANN ausscheidet, da der Aufwand für die Betragsbildung viel zu hoch ist. Um eine

einigermaßen vernünftige Genauigkeit zu erreichen, müssen mindestens vier verschiedene Koeffizienten pro Frequenz berechnet werden (der Fehler ist dann unterhalb 5%), d.h. für eine 128–Punkt DFT würden allein 4 Chips für Koeffizientenberechnung und noch einmal 12 Chips (für insgesamt 768 Neuronen) für die Berechnung der Amplituden benötigt.

7.1.3 Alternative Implementierungsmöglichkeiten

Eine interessante Variante wird von J. Brauch beschrieben (siehe [14]). Hier wird der Betrag der imaginären Größe mit einem kleinen Netz berechnet, benötigt werden 2 Eingänge, 4 Hidden Units und 1 Ausgang. Das Netz wird dann darauf trainiert, das Produkt der beiden Eingänge am Ausgang zu erzeugen. Problem dieser Methode ist wieder, daß sich die Anzahl der benötigten Units nahezu vervierfacht und man für eine 128 Punkt DFT sechs ETANNs benötigt. Ein Problem dieser Methode ist auch die mangelnde Genauigkeit der so erzeugten Multiplizierer.

Die Berechnung der DFT auf dem ETANN scheidet demnach aus, solange man keine zusätzliche Hardware zur Erzeugung der Amplituden verwenden will. Da aber für die Berechnung von Fourier–Transformationen bereits genügend echtzeitfähige Module existieren, ist das kein großes Problem. Bei einer echten "stand alone"–Lösung des Wordspotters müßten dann die entsprechenden Verstärker und ein DFT–Modul verwendet werden.

7.2 TDNNs und der ETANN als analoger Speicher

Für Sprachverarbeitung haben normale Neuronale Netze den Nachteil, daß sie statisch sind. Das heißt, Zusammenhänge über der Zeit werden nicht erkannt und können nicht klassifiziert werden. Eine Möglichkeit Zeitabhängigkeiten zu berücksichtigen bietet das sogenannte "Time Delay Neural Network" (kurz TDNN, siehe auch [18]). Zusätzlich ist das TDNN sogar verschiebungsinvariant, es klassifiziert also auch Zeitabhängigkeiten, die zeitlich

verschoben sind. Die Grundidee des TDNN besteht darin, Eingänge zu verschiedenen Zeitpunkten zu betrachten. Das hat zur Folge, daß sich auch Folgen von Eingangssignalen klassifizieren lassen. Bild 7.2 soll das verdeutlichen, es zeigt das altbekannte XOR–Netz. Es wird aber nur ein Signal zu

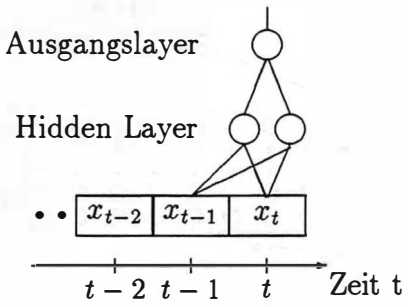


Abbildung 7.2: Ein TDNN berechnet ein sequentielles XOR

zwei aufeinanderfolgenden Zeitpunkten betrachtet. Dadurch „erkennt“ das Netz aufeinanderfolgende, ungleiche Eingangssignale. Das Training eines solchen Netzes unterscheidet sich nicht sonderlich vom Training eines normalen Neuronalen Netzes (siehe auch Anhang A).

Die erste Anwendung des TDNN in der Spracherkennung war 1987 ein Netzwerk zur Phonemerkennung (siehe auch [18]). Bild 7.3 zeigt die Struktur des Netzes und veranschaulicht die verwendeten Time Delays. Die 16 Eingangssignale sind in diesem Falle Melscale Koeffizienten, eine komprimierte Darstellung eines Frequenzspektrums. Diese 16 Eingangssignale werden dann mit drei Time Delays an die nächsten 8 Units weitergereicht, effektiv hat also jedes der Units im ersten Hidden Layer 16*3=48 Eingänge. Genauso gehen dann diese 8 Units, versehen mit diesmal 5 Time Delays in den zweiten Hidden Layer, der drei Units enthält. Der Ausgangs–Layer „integriert“ die drei Hidden Units des zweiten Hidden Layers schließlich noch über 9 Zeitpunkte auf, d.h. in diesem Fall werden sogenannte “shared weights” verwendet, das sind Gewichte die alle den selben Wert haben. Diese “shared weights” führen unter anderem zur Verschiebungsinvarianz des TDNN. Das Netz erkennt schließlich drei akustisch ähnliche Phoneme, „B“, „D“ und „G“.

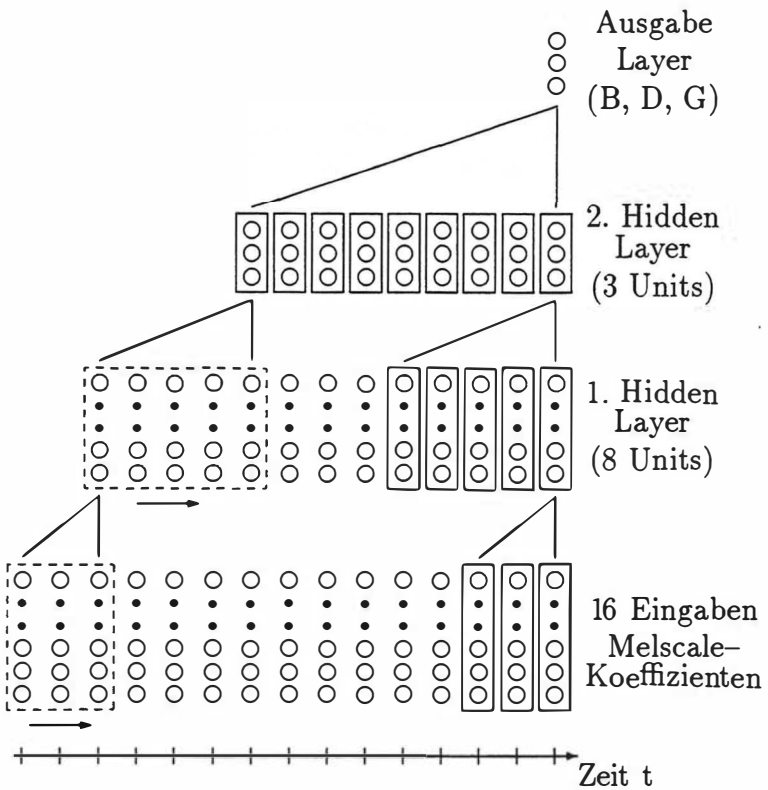


Abbildung 7.3: Die Archtiktur des BDG–TDNNs

Wie werden diese Time Delays nun auf dem ETANN realisiert? Selbstverständlich kann man durch externe Beschaltung Signale zwischenspeichern und so die nötige Vergangenheit der Eingänge zur Verfügung halten. Allerdings bietet es sich durch die Struktur des ETANN auch an, die Signale, die für Time Delays benötigt werden, intern zu speichern. Durch die Möglichkeit, Ausgangssignale in die Sample&Hold–Glieder des Rückkoppel–Arrays zu führen, kann eine Folge von Signalen in dem ETANN gespeichert werden. Bild 7.4 zeigt die Realisierung des zweiten Layers des BDG–Netzes auf einem ETANN. Die 8 Eingangssignale werden viermal über die Ausgänge in das Rückkoppel–Array zurückgeführt und schließlich, zusammen mit den aktuellen Eingangssignalen in die 3 Ausgangsneuronen geführt. Die mit „1“ markierten Matrixbereiche sind als Einheits–Matrix programmiert und dienen nur dazu, die Eingaben unverändert an die Sample&Hold–Glieder weiterzureichen. Die unmarkierten Felder sind gelöscht, tragen

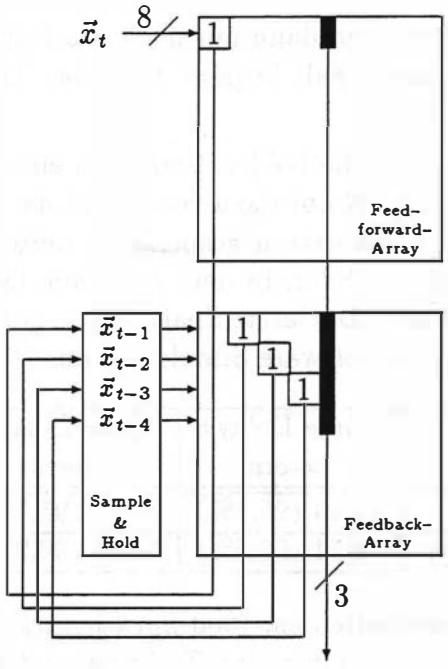


Abbildung 7.4: Der zweite Layer des BDG–Netzes auf einem ETANN

somit zum Ergebnis nicht bei und die ausgefüllten Bereiche enthalten die Gewichte des eigentlichen Netzes. Damit berechnet dieser Teil des so programmierten ETANN einen TDNN–Layer mit 8 Eingängen (zu den Zeitpunkten $t, t-1, \cdots, t-4$) und 3 Ausgängen.

Das Problem bei dieser Art der Realisierung sind die Toleranzen der analogen Multiplizierer auf dem ETANN, Unlinearitäten der Sigmoide sowie Signalverluste bei der Rückführung der Aktivitäten über das Rückkoppel–Array. Durch ein "chip in loop"–Training können diese Ungenauigkeiten kompensiert werden. Man muß in diesem Fall allerdings die besonderen Eigenschaften des Rückkoppel–Arrays berücksichtigen und bei dem "chip in loop"–Training immer wieder die in der Simulation berechneten Ausgangswerte durch die tatsächlichen Werte an den Ausgängen des Chips ersetzen. Dadurch wird erreicht, daß sich während des Trainings die Gewichte des Netzes entsprechend verändern, um die abschwächende Wirkung des ETANN–Feedbackarrays auszugleichen. Durch den großen Einfluß dieses Effekts macht es auch keinen Sinn mehr, das Netz erst in einer Software

Simulation zu trainieren, und dann nur noch das Feintraining "chip in loop" durchzuführen. In diesem Fall beginnt man das Training gleich mit Einbeziehung des ETANN.

Die folgende Tabelle stellt die beiden Verfahren einander gegenüber. Im ersten Fall wurde der ETANN nur dazu verwendet das Netz zu berechnen, die Time Delays müssen dabei extern gespeichert werden. Die nächste Spalte zeigt die Daten bei einem Netz, in dem die Time Delays auf dem ETANN selber gespeichert werden. Die letzte Spalte zeigt schließlich die entsprechenden Zahlen für eine reine Software Simulation des Netzes.

MSE (Erk.rate)	Time Delays extern	Time Delays intern	Software Simulation
Testmenge	14,49 (96,1%)	21,75 (95,7%)	9,04 (96,8%)
Trainingmenge	10,54 (97,5%)	15,37 (96,9%)	6,23 (98,1%)

Die gesamten Daten enthalten ca. 1600 Sprachstücke und wurden aufgeteilt in zwei gleichgroße Mengen für das Training und den Test. Angegeben wurde jeweils der "mean square error" und in Klammern dahinter die Erkennungsrate in Prozent, da der MSE keine besonders gute Aussage über die tatsächlich richtig erkannten Muster bietet. Die etwas höheren Werte für den MSE rühren mit daher, daß ein Sigmoid mit dem Bereich $[-1, +1]$ verwendet wurde, was die Sigmoide des ETANN nicht immer ganz erreichen, hier liegt der Ausgangsbereich des Sigmoids durchschnittlich um $[-0.95, +0,95]$. Dieser Effekt hat aber auf die Erkennungsrate keinen großen Einfluß.

7.3 Linear Time Alignment statt Dynamic Time Warping?

Der letzte Schritt, die "Dynamic Time Warping" Routine auf dem ETANN zu implementieren, ist sicherlich keine leichte Aufgabe. Dadurch, daß dieses Verfahren eine optimale Lösung produziert, kann ein Ersatz durch ein Neuronales Netz nur ebenso gute oder sogar schlechtere Ergebnisse liefern. Ein Ansatz ist das sogenannte „Viterbi–Netz" (siehe hierzu auch [21]), welches den besten Pfad in einer Matrix unter bestimmten Randbedingungen findet. In Anlehnung an den Viterbi–Algorithmus berechnet dieses Netz die Wahrschein-

lichkeit für den optimalen Pfad, wenn Wahrscheinlichkeiten für Übergänge von einem Zustand zum nächsten gegeben sind ($a_{i,i+1}$) und als Eingabe jeweils die Wahrscheinlichkeiten für die entsprechenden Zustände vorliegen (p_{s_i}). Bild 7.5 zeigt die Struktur dieses Netzes. Als Eingang in das Netz dient

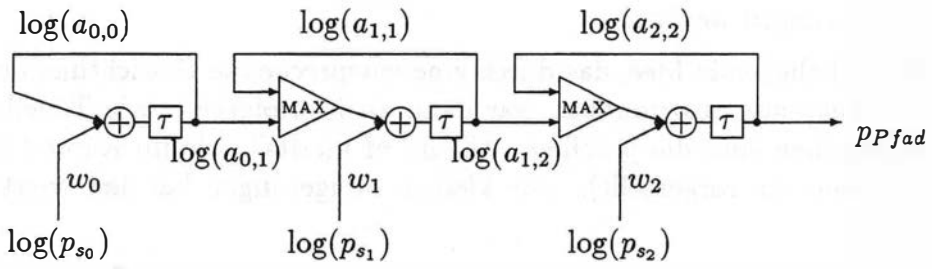


Abbildung 7.5: Das Netz zur Berechnung des Viterbi–Algorithmus

der Logarithmus des Scores eines bestimmten Zustands. Das Netz berechnet zu jedem Zeitpunkt den Logarithmus der höchsten Wahrscheinlichkeit für das Vorhandensein eines Pfades vom ersten bis zum letzten Zustand. Dieses Verfahren ist allerdings für eine Realisierung auf dem ETANN ungeeignet, da es eine große Präzision bei der Aufsummierung der Logarithmen verlangt. Das läßt sich auf dem ETANN — insbesondere bei langen Zustandsfolgen — aufgrund der Ungenauigkeiten nicht erreichen.

Eine zweite Idee versucht die Pfadsuche durch ein Neuronales Netz zu erledigen. Hierbei wird angenommen, daß die optimalen Pfade nicht sehr von der linearen Aufteilung abweichen. Das heißt, alle Zustände kommen in etwa gleichlang vor. Da bei dem Wordspotter die minimalen und maximalen Wortlängen bekannt sind, kann man ein Netz konstruieren, welches die Aktivitäten der Zustände aufaddiert, unter der Annahme, daß alle Zustände gleichlang vorkommen.

Dieser Ansatz wurde schließlich auch gewählt, um die DTW–Routine des Wordspotters zu ersetzen.

Bei ersten Software Simulationen, die ein schon trainiertes Netz mit dieser Art der Score–Berechnung testeten, waren die Ergebnisse schon relativ gut.

Nach einem näheren Blick auf die State–Matrix entstand noch eine Idee, die Ergebnisse zu verbessern. Durch das adaptive Training kann es vorkommen, daß ein Zustand für ein Wort nicht mehr von Bedeutung ist — meistens weil er in verschiedenen Wörtern vorkam und deshalb immer wieder negativ trainiert wurde. Und durch die Verwendung der DTW–Routine kann ein solcher Zustand, ohne allzu negativen Einfluß auf den endgültigen Score, übersprungen werden.

Die naheliegende Idee, das durch eine entsprechende Gewichtung der einzelnen Zustände auszunutzen, war dann auch erfolgreich, wie Tabelle 1 zeigt (angegeben sind die jeweiligen "figure of merits", wie im Kapitel über den Wordspotter vorgestellt). Die kleinen Steigerungen bei dem vierten Wort

Tabelle 1: *Vergleich unterschiedlicher Implementierungen der Linear Time Alignment Routine gegenüber der Standard DTW–Routine. (Verwendete Wörter: "springfield", "secondary", "interstate" und "primary")*

Implementierung	Wort #1	Wort #2	Wort #3	Wort #4	Alle Wörter
DTW–Routine	86,1%	96,1%	76,5%	72,2%	85,2%
LTA, keine Gewichte	75,1%	96,4%	76,2%	58,1%	80,3%
LTA, gewichtet (Ex2)	80,3%	96,4%	76,2%	68,4%	83,6%

kommen dadurch zustande, daß die DTW–Routine "state transition penalties" addiert, was den endgültigen Score des Wortes etwas verschlechtert und den Abstand zu den Fehlalarmen verringert. Wort 2 und 3 ließ sich durch eine Zustandsgewichtung nicht weiter verbessern; bei den anderen Wörtern war aber eine Steigerung möglich, da teilweise Zustände für die Worterkennung nicht relevant waren. Um die Gewichte für das Linear Time Alignment mit gewichteten Zuständen zu erhalten, ist ein Testlauf auf der Trainingsmenge nötig. Dadurch erhält man eine Statistik über die Scores der einzelnen Zustände in den jeweils optimalen Pfaden. Anhand dieser Statistik werden dann Gewichte für die einzelnen Zustände berechnet, die dafür sorgen, daß Zustände, die auch bei einer DTW–Pfadsuche nur „schlechte" Scores beisteuern, beim Linear Time Alignment wenig Einfluß haben.

Das nächste Kapitel zeigt nun die tatsächliche Implementierung des TDNN–

und DTW–Teiles des Wordspotters auf dem ETANN Multi–Chip–Board.

7.4 Resultate der Module und des kompletten Systems

Die vorangegangenen Kapitel haben gezeigt, ob und wie sich die verschiedenen Teile des Wordspotters auf dem ETANN realisieren lassen. Interessant ist nun natürlich, wie gut sich diese Ergebnisse kombinieren lassen. Insbesondere die Auswirkungen der Toleranzen des ETANN sind von Interesse.

Unter Beibehaltung der DTW–Routine wurde zuerst nur der TDNN–Teil des Wordspotters auf den ETANN portiert. Chip Nr. 1 und Nr. 2 auf dem Multi–Chip–Board wurden entsprechend programmiert, so daß der Forwardpass des Wordspotters auf den ETANNs laufen konnte. Die Realisierung des TDNNs auf zwei ETANNs kann, wie Bild 7.6 zeigt, auf zweierlei Art geschehen. Direkt nach dem Programmieren der Gewichte waren die Ergebnisse recht enttäuschend, wurden aber nach einem "chip in loop"–Training schnell besser, wie die Tabelle 1 zeigt (angegeben ist, wie immer, die "figure of merit", *FOM*).

Tabelle 1: *Vergleich unterschiedlicher Implementierungen des Wordspotter TDNNs. (Verwendete Wörter: "springfield", "secondary", "interstate" und "primary")*

Implementierung	Wort #1	Wort #2	Wort #3	Wort #4	Alle Wörter
Software	86,1%	96,1%	76,5%	72,2%	85,2%
ETANN, kein CIL	10,2%	23,4%	15,7%	21,2%	17,5%
ETANN, nach CIL	70,4%	72,0%	60,3%	66,8%	71,8%

Den Einfluß der Unlinearitäten des ETANN erkennt man an dieser Tabelle überdeutlich. Ein einfaches Programmieren der Gewichte mit den Werten, die das Software–Training erzeugt hat, führt zu katastrophalen Ergebnissen. Die Qualität des "chip in loop"–Trainings ist für dieses Netzwerk also entscheidend. Und hier setzt auch eines der größten Probleme ein. Die vielen

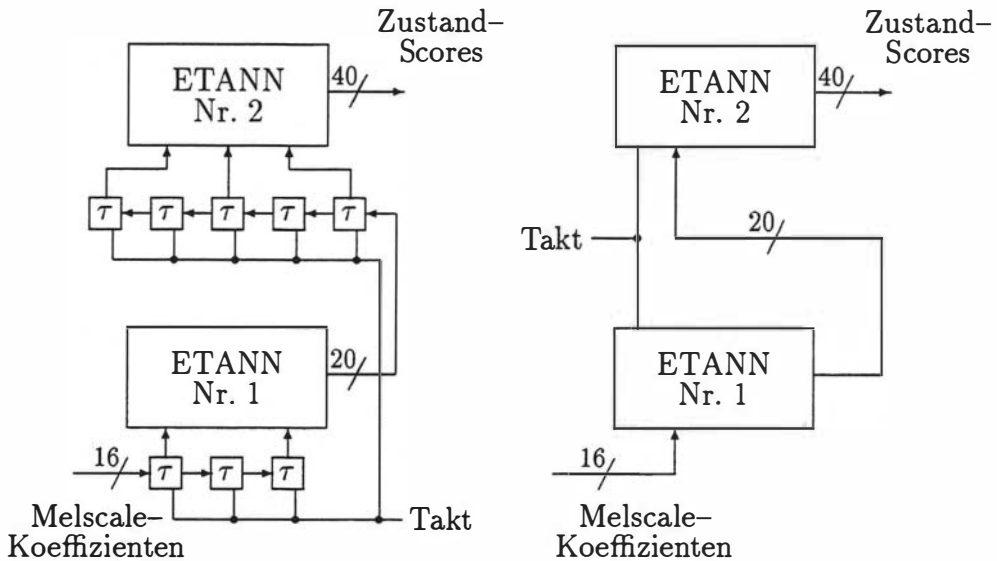


Abbildung 7.6: Realisierung des Wordspotter TDNNs auf zwei ETANNs mit externen Time Delays (links) und interner Speicherung der Time Delays in den Sample&Hold–Gliedern des Feedbackarrays (rechts).

kleinen Hacks, die im Wordspotter dafür sorgen, daß die Erkennungsraten hervorragend ausfallen, lassen sich nicht einfach auf die ETANN–Lösung übertragen. Vielmehr müßte versucht werden, die Hacks so zu übertragen, daß sie auch auf einem Hardware Neuronalen Netz zu brauchbaren Ergebnissen führen. Das "chip in loop"–Training wurde also, mit halbierter Lernrate und in diesem Falle adaptiv, noch viermal durchgeführt[1]. Das führte (bisher) zu den besten, hier angeführten Ergebnissen.

Die Ersetzung der DTW–Routine durch das Linear Time Alignment mit gewichteten States brachte dann, wie zu erwarten war, eine leichte Verschlechterung. Durch die zusätzliche Portierung dieses Netzes auf das Multi–Chip–Board blieb das Ergebnis aber praktisch unverändert. Tabelle 2 stellt die Ergebnisse dieser Schritte vor (das TDNN lief bei allen Experimenten auf den ETANNs).

[1] Der „Software"–Wordspotter benötigt 4 Bootstrap– und 4 adaptive Trainingsläufe über die gesamten Trainingssätze, um die erwähnte Performanz zu erreichen.

Tabelle 2: *Vergleich unterschiedlicher Implementierungen des Linear Time Alignments. (Verwendete Wörter: "springfield", "secondary", "interstate" und "primary")*

Implementierung	Wort #1	Wort #2	Wort #3	Wort #4	Alle Wörter
DTW–Routine	70,4%	72,0%	60,3%	66,8%	71,8%
WLTA (Software)	64,4%	72,0%	59,2%	62,1%	65,4%
WLTA (auf ETANN)	61,3%	72,0%	57,6%	61,2%	64,0%

Deutlich erkennbar ist, daß sich die Erkennungsrate nicht wesentlich verschlechert, wenn das WLTA von der Software–Simulation auf den ETANN portiert wird. Das liegt an der einfachen Struktur, die nur ein Aufsummieren der Zustands–Scores erfordert. Dadurch haben Unlinearitäten des ETANN keinen entscheidenden Einfluß.

Kapitel 8

Zusammenfassung

Die Aufgabe der vorliegenden Diplomarbeit war es, zu ergründen, wie sich vorhandene Hardware–Implementierungen von Neuronalen Netzen eignen, um darauf ein Spracherkennungs–System zu realisieren. Als Spracherkenner wurde ein Wordspotter gewählt, da es sich hierbei um ein relativ kleines und abgeschlossenes System handelt.

Die Aufgabe setzte sich also aus mehreren Teilen zusammen:

- Es galt festzustellen, was es zur Zeit (im Frühjahr 1992) überhaupt an Hardware-Neuronalen Netzen gab. Betrachtet wurden im Rahmen dieser Arbeit die CNAPS von Adaptive Solutions, die RAP aus Berkeley, der CCD–Chip von Lincoln Labs, die NN–Chips von JPL und der ETANN von Intel.

- Die Auswahl eines Systems für diese Arbeit: das fiel relativ leicht, da sich bald herausstellte, daß die meisten Systeme entweder noch nicht verfügbar waren, oder für diese Aufgabenstellung offensichtlich nicht geeignet waren. Die Wahl fiel auf den ETANN von Intel, da es sich hierbei um ein komplettes System, einen analogen Neuronalen–Netz–Chip mit Entwicklungsumgebung und Entwurfswerkzeugen handelt, das von Intel zur Verfügung gestellt wurde.

- Einarbeitung in den Wordspotter: hierfür wurde ein kompletter Simulator für Neuronale Netze (NESI) in C implementiert. Dies hatte mehrere Vorteile:

  - Durch die modulare Aufbauweise des Programms war es leicht möglich, dieses von den Workstations auf einen PC zu übertragen, wo die Entwicklungsumgebung des ETANN lief.
  - Durch die komplette Neuimplementierung wurden die Tricks des Wordspotters bewußt.
  - Der Einbau von Routinen, die den ETANN ansprachen und benutzten, wurde möglich.

- Die Umsetzung des Wordspotters auf den ETANN–Chip: Hierfür mußte ein "time delay neural network" (TDNN) und die von Zeppenfeld modifizierte "dynamic time warping"–Routine auf dem Chip implementiert werden.

  Zur Realisierung des TDNNs wurden die auf dem Chip vorhandenen Sample&Hold–Glieder verwendet, da sich dadurch ein zusätzlicher Hardwareaufwand (in Form von analogen Haltegliedern) vermeiden ließ. Dies wurde durch geschickte Wahl der Gewichte möglich, so daß gewisse Teile der ETANN–Gewichte dazu verwendet wurden, anliegende Daten durchzureichen. Das hatte bedingt durch Nichtlinearitäten und Fertigungstoleranzen allerdings Ungenauigkeiten zur Folge, die jedoch durch ein entsprechendes Training des gesamten Netzes auf dem Chip ("chip in loop") reduziert werden konnten.

  Das DTW konnte nicht übernommen werden, da es sich nicht eignet, direkt auf Neuronalen Netzen implementiert zu werden. Es wurde ein Verfahren entwickelt, das dem DTW, wie es im Wordspotter verwendet wird, ähnelt und vergleichbare Ergebnisse liefert. Dieses "weighted linear time alignment" (WLTA) genannte Verfahren konnte dann auf dem ETANN realisiert werden.

Ergebnisse:

Es hat sich gezeigt, daß eine Portierung des Wordspotters auf die Hardware des ETANN grundsätzlich möglich ist. Wie nicht anders zu erwarten war, brachte diese Portierung auf analoge Hardware einen Rückgang in der Erkennungsrate. Doch zeigte sich, daß durch Anpassung des Spotters an die gegebenen Umstände durchaus eine Verbesserung möglich ist.

Mit dem verwendeten WLTA anstelle des DTW stellten sich Ergebnisse ein, die in der Erkennungsrate nur etwa 20 Prozentpunkte unter denen der Software–Simulation lagen. Durch verbesserte Anpassung der Parameter des Wordspotters an die Hardware–Umgebung sind aber noch weitere Steigerungen in der Erkennungsrate zu erwarten.

Grundsätzlich läßt sich sagen, daß eine Realisierung des Wordspotters auf zur Zeit vorhandener Neuronale Netz Hardware gut möglich ist und brauchbare Ergebnisse liefert. Doch stellt sich die Frage, ob dies überhaupt erstrebenswert ist und ob der ETANN die dafür optimale Umgebung darstellt. Es erscheint unangebracht, ein System, das Neuronale Netz–fremde Algorithmen enthält, wie die DTW–Routine bei dem Wordspotter, auf einem NN-Chip implementieren zu wollen. Aber durch die vergleichsweise kleine Implementierung ergeben sich viele Anwendungen im Bereich der portablen Spracherkennung (z.B. Autotelephon) und auch in Massenartikeln (z.B. die berühmte Waschmaschine, die gesprochene Kommandos erkennt).

Auf jeden Fall wurde klar, daß bei der Entwicklung eines Systems, welches später auf Hardware Neuronalen Netzen implementiert werden soll, dieser Aspekt schon früh berücksichtigt werden muß. Andernfalls ist man später gezwungen, große Teile der Arbeit neu zu implementieren, damit das System überhaupt auf die Hardware übertragen werden kann. Behält man während der Entwicklung des Systems die Hardware und die dadurch verursachten Einschränkungen (aber vielleicht auch Möglichkeiten) im Auge, stehen einer Portierung nicht allzu viele Schwierigkeiten im Wege.

Anhang A

Der Error–Back–Propagation Algorithmus

Im Folgenden wird kurz der bekannte Error–Back–Propagation Algorithmus vorgestellt, um die später verwendete Notation einzuführen. Darauf folgend werden die Erweiterungen des Algorithmus für Time–Delay Netze und für Netze mit Einschränkungen an den Gewichten (z.B. beschränkte Genauigkeit, eingeschränkter Bereich) beschrieben.

A.1 Back–Propagation

Idee des BP–Algorithmus ist es, ein Neuronales Netz überwacht zu trainieren. Die Ausgaben, die vom Netz zu bestimmten Eingaben errechnet werden, werden mit den gewollten Ausgaben verglichen. Der dabei festgestellte Fehler wird dann dazu verwendet, die Gewichte des Netzes zu verändern.
Für das Training wird also benötigt:

- eine Trainings–Menge ξ von Ein– und Ausgaben:
  $\xi^P = (X^P, O^P)$
  - Eingaben $X^P = (x_1^P, \cdots, x_i^P, \cdots, x_n^P)$
  - gewünschte Ausgaben $O^P = (o_1^P, \cdots, o_j^P, \cdots, o_m^P)$

- die tatsächlichen Ausgaben des Netzes:
  $A^P = (a_1^P, \cdots, a_j^P, \cdots, a_m^P)$

Damit läßt sich ein Maß für den Fehler definieren, den das Netz auf der gesamten Trainings–Menge ξ hat:

$$E = \sum_{(X^P, O^P) \in \xi} \frac{1}{2} \sum_{j=1}^{m} (a_j^P - o_j^P)^2$$

Dieser Ausdruck ist also abhängig von allen Gewichten. Betrachtet man die Fehlerfunktion als Funktion eines Gewichtsvektors $W = (w_{1,1}^1, \cdots, w_{l,m}^K)$, dann gibt der Gradient von E bezüglich W die Richtung des stärksten Anstiegs von E an. Um den Fehler zu verringern, muß der Gewichtsvektor nur noch proportional zu $-grad_W(E)$ verändert werden:

$$\Delta W = -\gamma grad_W(E)$$

γ ist dabei eine Proportionalitätskonstante, die die „Lern"–Geschwindigkeit beeinflußt. Mittlerweile wird allerdings nicht erst nach dem Anlegen aller Test–Muster gelernt, sondern die Gewichte werden nach Anlegen jedes Musters einzeln verändert. Wie sich herausstellt, funktioniert dieses Verfahren auch und konvergiert deutlich schneller. Mit dem Teilausdruck:

$$E^P = \frac{1}{2} \sum_{j=1}^{m} (a_j^P - o_j^P)^2$$

der den Fehler für ein Ein–/Ausgabemuster P bezeichnet, gilt dann:

$$\Delta W_P = -\gamma \, grad_W(E^P)$$

Im Folgenden wird die Indizierung über der Menge der Trainingsmuster aus Gründen der Übersichtlichkeit weggelassen.

Bild A.1 soll die Bezeichnungen des Netzes deutlich machen und die Struktur veranschaulichen. Das Netz besteht aus K Layern, mit jeweils m_κ Neuronen. Das Gewicht an der Verbindung von Layer $\kappa-1$, Neuron k zu Layer κ, Neuron l wird mit w_{kl}^κ bezeichnet. Das Ergebnis der gewichteten Summe wird durch

$$\Sigma_j^\kappa = \sum_{l=1}^{m_{\kappa-1}} w_{lj}^\kappa a_l^{\kappa-1} + \theta_j^\kappa$$

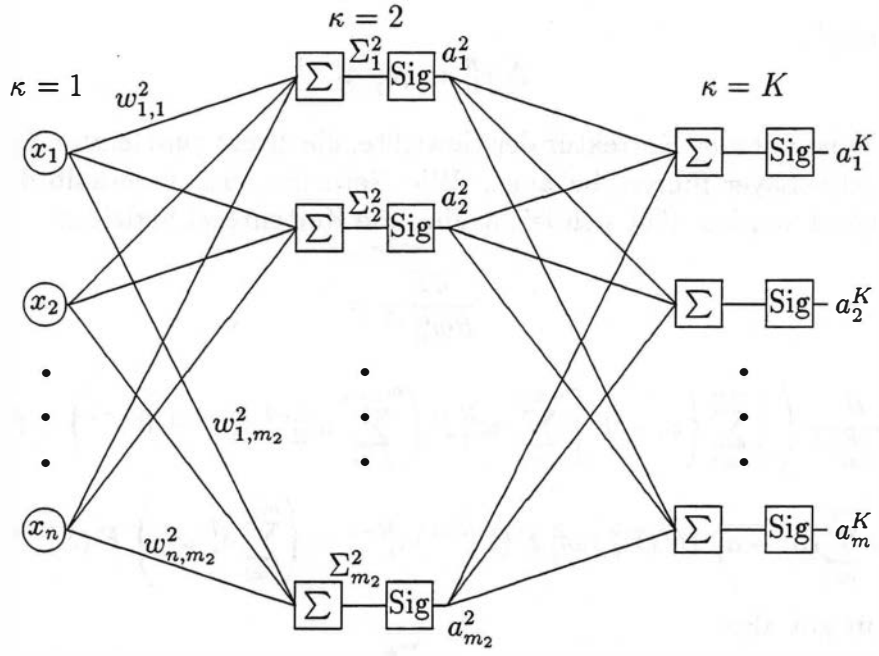


Abbildung A.1: Bezeichnungen des neuronalen Netzes

abgekürzt.

Aus der Formel $\Delta W = -\gamma grad_W(E)$ folgt dann die Änderung der einzelnen Gewichte:

$$\Delta w^{\kappa}_{kl} = -\gamma \frac{\partial E}{\partial w^{\kappa}_{kl}}$$

Da die Ausgabe des Netzes eine differenzierbare Funktion der Aktivitäten des vorletzten Layers ist,

$$a^K_j = F\left(\sum_{l=1}^{m_{K-1}} (w^K_{lj} a^{K-1}_l) + \theta^K_j\right) = F(\Sigma^K_j)$$

läßt sich die partielle Ableitung mathematisch umformen:

$$\frac{\partial E}{\partial w^K_{lj}} = \frac{\partial E}{\partial \Sigma^K_j} \frac{\partial \Sigma^K_j}{\partial w^K_{lj}} = -(o_j - a^K_j) F'(\Sigma^K_j) a^{K-1}_l$$

Mit

$$\delta^K_j = (o_j - a^K_j) F'(\Sigma^K_j)$$

gilt also[1]:

$$\Delta w_{lj}^K = \gamma \delta_j^K a_l^{K-1}$$

Bisher ist nur eine Korrektur der Gewichte, die direkt zum letzten Layer, dem Ausgabe–Layer führen, bekannt. Wie Gewichte auch innerhalb des Netzes korrigiert werden, läßt sich leicht über die Kettenregel herleiten:

$$\frac{\partial E}{\partial w_{kl}^{K-1}} =$$

$$\frac{\partial}{\partial w_{kl}^{K-1}} \left(\frac{1}{2} \sum_{j=1}^{m_K} \left\{ o_j - F \left[\sum_{l=1}^{m_{K-1}} w_{lj}^K F \left(\sum_{k=1}^{m_{K-2}} w_{kl}^{K-1} a_k^{K-2} + \theta_l^{K-1} \right) + \theta_j^K \right] \right\}^2 \right)$$

$$= \sum_{j=1}^{m_K} (o_j - a_j^K) F'(\Sigma_j^K) w_{lj}^K F'(\Sigma_l^{K-1}) a_k^{K-2} = \left(\sum_{j=1}^{m_K} \delta_j^K w_{lj}^K \right) F'(\Sigma_l^{K-1}) a_k^{K-2}$$

Damit gilt also:

$$\delta_l^{K-1} = \sum_{j=1}^{m_K} \delta_j^K w_{lj}^K$$

Dieses Verfahren ist rekursiv anwendbar, um die restlichen δ_l^κ zu berechnen:

$$\delta_l^\kappa = F'(\Sigma_l^\kappa) \sum_{j=1}^{m_{\kappa+1}} \delta_j^{\kappa+1} w_{lj}^{\kappa+1}$$

Diese Formeln können nun verwendet werden, um rekursiv alle Δw_{kl}^κ für jeden Layer des Netzes zu berechnen und die Gewichte so zu verändern, daß der Fehler am Ausgang kleiner wird.
Das führt zu folgendem Verfahren:

[1] Bei Verwendung eines Sigmoids ist zur Berechnung der Ableitung nur die Ausgabe des jeweiligen Neurons nötig:

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

$$F'(x) = F(x)(1 - F(x))$$

Es ist also nicht notwendig, die Eingaben zwischenzuspeichern.

1. Bestimmung von δ_j^κ:

   - Ausgabe Layer ($\kappa = K$):

   $$\delta_j^K = (o_j - a_j^K) F'(\Sigma_j^K)$$

   - Hidden Layer:

   $$\delta_l^\kappa = F'(\Sigma_j^\kappa) \sum_{j=1}^{m_{\kappa+1}} \delta_j^{\kappa+1} w_{lj}^{\kappa+1}$$

2. Veränderung der Gewichte:

$$\Delta w_{kl}^\kappa = \gamma \delta_l^\kappa a_k^{\kappa-1}$$

Beginnend mit dem letzten Layer, werden diese Schritte für jeden Layer des Netzes einmal durchgeführt.

A.2 Variation des BP–Algorithmus für TDNNs

Es gibt zwei Möglichkeiten, den Algorithmus so zu verändern, daß auch Netze mit Time–Delay trainiert werden können. Die erste Variante dupliziert die Teile des Netzes, die eigentlich durch ein Time–Delay erzeugt werden (auch "unfolding in time"), und erzeugt so ein normales Netz. Während des Trainings muß man allerdings immer wieder die — eigentlich ja identischen — Gewichte, die an duplizierten Verbindungen sitzen, mitteln und damit dafür sorgen, daß diese Gewichte gleich bleiben. Das andere Verfahren, das auch hier verwendet wird, besteht darin, daß jeder Layer des Netzes genug der Eingaben des darunterliegenden Layers zwischenspeichert, um das Time–Delay selber berechnen zu können. Das führt dann allerdings zu einem etwas modifizierten Trainings–Algorithmus. Es kann nun nicht mehr davon ausgegangen werden, daß für jeden Ausgang nur ein Fehler vorliegt. Vielmehr kann es auch vorkommen, daß — bedingt durch darüberliegende Time–Delays — Fehler für einen Ausgang zu verschiedenen Zeitpunkten vorhanden sind. Diese dann entsprechend zu kombinieren ist die Aufgabe des hier vorgestellten Algorithmus.

1. Bestimmung von $\delta_j^{\kappa,\tau}$:
   für alle gespeicherten Zeitpunkte τ $(1 \leq \tau \leq \tau_{max})$:

   - Ausgabe–Layer $(\kappa = K)$:

     $$\delta_j^{K,\tau} = (o_j^\tau - a_j^{K,\tau}) F'(\Sigma_j^{K,\tau})$$

   - hidden Layer $(\kappa < K)$:
     Ebene $\kappa + 1$ hat die time–delays $\epsilon \in (\epsilon_1, \cdots, \epsilon_{max})$:

     $$\forall \tau' : 1 \leq \tau' \leq \tau_{max} + \epsilon_{max} : \; \delta_j^{\kappa,\tau'} = \sum_{\tau+\epsilon=\tau'} F'(\Sigma_j^{\kappa,\tau+\epsilon}) \sum_{l=1}^{m_{\kappa+1}} \delta_l^{\kappa+1,\tau+\epsilon} w_{lj}^{\kappa+1}$$

2. Veränderung der Gewichte:

   $$\Delta w_{kl}^\kappa = \sum_{\tau'=1}^{\tau_{max}+\epsilon_{max}} \gamma \delta_l^{\kappa,\tau'} a_k^{\kappa-1,\tau'}$$

Es muß bei diesem Trainingsverfahren also sichergestellt sein, daß die Ausgaben der einzelnen Neuronen mit ausreichender „Vergangenheit“ gespeichert werden.

A.3 Der BP–Algorithmus und beschränkte Gewichte

Ein Problem beider vorgestellter Verfahren ist, daß die Gewichte beliebige Werte annehmen können. Das ist aber in der Praxis, z.B. bei vielen Hardwarelösungen für neuronale Netze, nicht immer möglich. Deshalb wird eine Änderung des Back–Propagation–Algorithmus vorgestellt, die verhindert, daß die Gewichte bestimmte Grenzwerte über– oder unterschreiten.

Die naheliegende Idee, Gewichte, die während des Trainings unzulässige Werte annehmen, einfach auf den Wert des Grenzwertes zu setzen, hat den Nachteil, daß man eine nicht durchgehend differenzierbare Funktion erhält. Die hier vorgestellte Idee vermeidet dieses Problem und läßt sich daher ohne Probleme in die Herleitung des Back–Propagation–Algorithmus übernehmen.

Damit ist sichergestellt, daß auch weiterhin ein mathematisch korrekter Weg beschritten wird. Grundgedanke ist es, auf die Originalgewichte eine differenzierbare, monoton steigende Funktion anzuwenden, die innerhalb bestimmter Grenzen bleibt. Naheliegend ist deshalb ein Sigmoid. Bei geeigneter Wahl der Steigung entspricht die Funktion um den Nullpunkt nahezu der identischen Funktion, verhindert aber ein Überschreiten der Grenzwerte. Dadurch ändert sich nur die Berechnung der gewichteten Summe

$$\Sigma_j^\kappa = \sum_{l=1}^{m_{\kappa-1}} W(w_{lj}^\kappa) a_l^{\kappa-1} + \theta_j^\kappa$$

mit

$$W(w) = \frac{1}{1+e^{-w}},$$

und die Backpropagation–Regel muß modifiziert werden:

$$\Delta w_{lj}^K = \gamma \delta_j^K W'(w_{lj}^K) a_l^{K-1}$$

In der Praxis scheint es allerdings keinen Unterschied zu machen, ob Gewichte, die einen Grenzwert überschreiten, einfach auf diesen Wert gesetzt werden, oder ob man den aufwendigen Weg geht, eine differenzierbare Funktion anzuwenden. Implementiert wird daher meistens[2] die einfachere Variante.

[2]Zumindest iBrainmaker, iDynamind und NESI verwenden diese Methode.

Anhang B

NESI, ein Simulator für Neuronale Netze

Da die meisten Simulatoren für Neuronale Netze zwar einige, aber nicht alle benötigten Eigenschaften vorweisen können, die für die vorliegende Arbeit benötigt wurden, wurde ein eigener Simulator entwickelt. Ziele bei der Entwicklung von „NESI[1]“ waren unter anderem:

- Modulkonzept, um einfache Umstrukturierungen zu ermöglichen.
- Unterstützung von Time Delays.
- Möglichkeit, eine Dynamic Time Warping Routine einzubinden.
- Unterstützung der Programmierung des ETANN, insbesondere des “chip in loop”–Trainings.
- Anpassung an gegebene Hardware durch Beschränkung der Gewichte auf ein bestimmtes Intervall und einen entsprechend modifizierten Trainingsalgorithmus.

[1] Another Neural Network Simulator

B.1 Module in NESI

Ein Modulkonzept wurde gewählt, um verschiedene Layer miteinander verbinden zu können, so daß Experimente mit verschiedenen Kombinationen von Layern erleichert werden. Bild B.1 zeigt ein Schema eines solchen Moduls. Jedes Modul besitzt also eine bestimmte Anzahl Ein– und

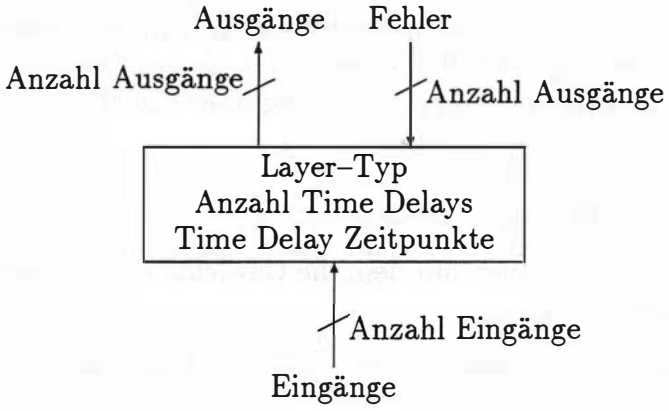


Abbildung B.1: Ein Modul von NESI als "black box"

Ausgänge. Zusätzlich sind für ein Training noch Fehler–Eingänge vorhanden, die einen am Ausgang festgestellten Fehler in das Modul führen. Um auch Time Delays realisieren zu können, wird für jedes Modul noch festgehalten, wie viele Time Delays benötigt werden und zu welchen Zeitpunkten diese verwendet werden. Layer–Typ gibt schließlich für jedes Modul noch den Typ des entsprechenden Layers an. Für jeden Typ schließen sich dann noch weitere Informationen an, die den entsprechenden Layer näher spezifizieren. Bisher wurden die folgenden Modul–Typen implementiert:

- BP–Layer: Ein normaler Neuronaler Netz Layer, der aber Time Delays erlaubt und durch Back Propagation trainiert wird (siehe Anhang A). Parameter:
  - Minimal und maximal erlaubte Werte der Gewichte.
  - Minimal und maximal erlaubte Werte der Biasgewichte.

  - Minimum, Nullpunkt und Maximum der Sigmoide.
  - "1", wenn Gewichte über Time Delays verbunden sind, "0" sonst. Diese Option wird zum Beispiel beim letzten Layer des BDG–Netzes verwendet, um die Integration über die 9 Time Delays des vorherigen Layers zu ermöglichen.

- ETANN–Layer: Ein Layer, der die Programmierung des ETANN erlaubt. Verhält sich wie ein normaler BP–Layer, erlaubt aber "chip in loop" Training und Berechnung des Netzes über den ETANN. (Unterstützt auch die Programmierung über das Multi–Chip–Board)
  Parameter:
  - siehe BP–Layer, zusätzlich
  - Chip–Nummer, auf dem die Gewichte dieses Layers abgelegt werden sollen.
  - Index des ersten zu verwendenden Eingangs.
  - Index des ersten zu verwendenden Neurons bzw. Ausgangs.

- TTW–Layer: Ein modifizierter "Dynamic Time Warping Algorithmus", der die Anpassungen an den Zeppenfeld'schen Wordspotter enthält.
  Parameter:
  - Maximale Länge der Wörter.
  - Anzahl der Wörter.
  - Für jedes Wort:
    * Wort (String zur Kennzeichnung)
    * Anzahl der States
    * Minimale und maximale Länge des Wortes

- LTA–Layer: Ein Layer, der ein Linear Time Alignment durchführt. Ähnlich dem TTW–Layer, nur wird die Aufteilung der States eines Wortes starr durchgeführt. D.h. alle States sind gleichlang.
  Parameter: siehe TTW–Layer.

- Window–Layer: Dieser Layer reicht die Eingaben direkt an die Ausgaben weiter, zeigt sie aber graphisch in einem Fenster an. Dieser Layer läßt sich zwischen zwei Module einschieben und man kann dann die Funktion des Netzes on–line am Bildschirm verfolgen.
  Parameter:
  - Ausschnitt der Eingänge der angezeigt werden soll (von—bis).
  - Breite des Zeitfensters.
  - Minimaler und maximaler Wert der Eingänge (für die Normierung auf 64 Graustufen).

Jedes Modul stellt bestimmte Routinen zur Verfügung, die dann vom Hauptprogramm je nach Konfiguration aufgerufen werden. Folgende Routinen sind in der momentanen Version von NESI nötig:

- Einlesen des eigenen, modulspezifischen Teiles der Konfigurationsdatei.
- Speicherreservierung (sofern nötig) und Freigeben des reservierten Speichers.
- Initialisieren des Moduls.
- Berechnung der Ausgaben bei vorgegebenen Eingaben (Forward–Pass).
- Justierung der Modul–internen Parameter bei vorgegebenen Eingaben, Ausgaben und des zugehörigen Fehlers (Training).
- Abspeichern und Einlesen der Modul–internen Parameter (beim Back–Propagation Layer sind das unter anderem die Gewichte)

B.2 Die Konfigurationsdatei

Um einen Wordspotter zu realisieren, der fünf Schlüsselwörter erkennen soll, müßte die Konfigurationsdatei für NESI demnach etwa wie folgt aussehen:

```
V0.0                % Versionsnummer
WordSpotter         % Beschreibung
3                   % Anzahl der Module
```

```
0                   % Modul Nr. 0
16                  % Anzahl der Eingaenge
20                  % Anzahl der Ausgaenge
2                   % zwei Time Delays ...
0 2                 % zur Zeit  (t) und (t-2)
BP_LAYER            % Layertyp (Backprop)
-2.5 +2.5           % Bereich der Gewichte
-10.0 +10.0         % Bereich der Bias Units
-1.0 0.0 +1.0       % Spez. des Sigmoid
0                   % keine 'connected weights'

1                   % Modul Nr. 1
20                  % Anzahl der Eingaenge
40                  % Anzahl der Ausgaenge
3                   % drei Time Delays
0 2 4               % (t), (t-2), (t-4)
BP_LAYER            % Layertyp (Backprop)
[...]

2                   % Modul Nr. 2
40                  % Anzahl Eingaenge
5                   % Anzahl Ausgaenge
1                   % Ein Time Delay
0                   % Zum Zeitpunkt (t)
TTW_LAYER           % Layertyp (Time Warping)
400                 % max. Wortlaenge
5                   % Anzahl der Woerter
springfield         % 1. Wort
8                   % Anzahl der States
42 168              % min., max. Laenge
[...]
secondary           % 5. Wort
8 43 92             % #States, min/max Laenge

END                 % Ende des Konfigurationsfiles
```

Während der Initialisierungsphase reserviert NESI den entsprechenden Speicherplatz für die Gewichte der ersten beiden Backprop–Layer und für die DTW–Matrizen der 5 Wörter des letzten Layers.

Soll der Wordspotter nach erfolgreichem Training dann auf dem Multi–Chip–Board laufen, muß nur ein anderes Konfigurationsfile geladen werden, in dem

die Eintragungen der BP–Layer durch ETANN–Layer ersetzt wurden, und die Zuordnung zu Chip–Nummern und –Pins festgelegt ist.

B.3 Bedienung

Die Bedienung von NESI erfolgt über eine kleine kommandoorientierte Oberfläche, die auch einen Batch–Betrieb gestattet. Damit können mehrere Trainingsläufe automatisch gestartet und die Gewichte immer wieder automatisch abgespeichert werden. Durch eine "on line" Hilfe werden die Befehle und ihre Syntax kurz erklärt. Die bis jetzt implementierten Kommandos werden im Folgenden, zusammen mit ihrer Syntax vorgestellt und kurz erläutert:

- `loadconf <filename>`: Lädt die entsprechende Konfigurationsdatei, reserviert Speicher für die angegebenen Module und initialisiert alle Speicherbereiche.

- `loadweights <filename>`: Lädt die Parameter aller Module, die zu diesem Zeitpunkt definiert sind (bei dem Backprop–Modul sind das unter anderem die Gewichte). Diese Datei muß zu der vorher geladenen Konfiguration passen.

- `saveweights <filename>`: Speichert die Parameter aller Module, die zu diesem Zeitpunkt definiert sind in die Datei `filename`.

- `initnet`: Initialisiert alle Module unter Berücksichtigung der Variablen `rmin`, `rmax`, `slope`. Je nach Modul werden zum Beispiel Gewichte zufällig initialisiert (BP–Layer) oder Score–Matrizen gelöscht (TTW–Layer).

- `set <var> <value>`: Setzt die angegebene Variable auf den gewünschten Wert. Vorhandene Variablen sind weiter unten aufgeführt.

- `show vars`: Zeigt die Werte aller Variablen an.

- `train [-f filename] [-c #cycles] [-g gamma]`: Trainiert die geladene Konfiguration mit der angegebenen Trainingsdatei und der gewünschten Lernrate Gamma. Ist der Parameter `-c` angegeben wird das Training entsprechend oft durchgeführt. Die Datei muß als erstes die Anzahl der Ein- und Ausgänge enthalten (die zur geladenen Konfiguration passen müssen!) und danach entweder den Buchstaben `i` bzw. `o`, gefolgt von der entsprechenden Anzahl Eingabe- bzw. Ausgabewerte. Das Ende der Datei markiert ein `e`.

- `test [-f filename] [-o outputfile] [-s]`: Führt einen Test mit der angegebenen Datei durch (Format dieser Datei: siehe `train`). Ist der Parameter `-o` angegeben wird das Testprotokoll in die entsprechende Datei geschrieben. `-s` erzwingt eine Ausgabe des ausführlicheren Protokolls auf dem Bildschirm.

- `ttwtrain [-f filename] [-p protocolfile] [-g gamma]`: (Parameter siehe `train`), ermöglicht Training eines MSTDNNs mit der entsprechenden Datei. Zusätzlich kann über den Parameter `-p` eine Protokoll-Datei angegeben werden in der die trainierten Pfade festgehalten werden.

- `ttwtest [-f filename] [-o outputfile]`: (Parameter siehe `test`), ermöglicht den Test eines MSTDNNs.

- `ttwload <filename>`: Lädt die Gewichte eines MSTDNNs. Das Dateiformat entspricht dem Format des Zeppenfeldschen Spotters ohne Vorspann.

- `connect [ETANN | EMB]`: Versucht den iNNTS-Adapter zur Programmierung des ETANN anzusteuern. Nur funktionsfähig bei einer Version, die auf einem PC compiliert wurde.

- `disconnect`: Koppelt den mit `connect` angeschlossenen iNNTS-Adapter wieder ab.

- `downloadnet`: Programmiert die Gewichte des ETANN entsprechend den momentan im Speicher befindlichen Gewichten der ETANN-Layer.

- `uploadnet`: Lädt die Gewichte vom ETANN in den Speicher.

- `do <filename>`: Führt eine Befehlsdatei aus. Alle Kommandos (außer `do` selber) werden genauso ausgeführt, als ob sie von der Tastatur eingegeben werden. Das Ende der Datei wird durch `end` markiert.

- `cd <directory>`: Wechselt das Arbeitsverzeichnis. Je nach Rechner müssen hierbei die entsprechenden UNIX oder DOS Konventionen für die Angabe von Verzeichnisnamen eingehalten werden.

- `! <command>`: Leitet `command` als Kommando an das Betriebssytem weiter.

- `quit`, `exit`: Verlassen von NESI.

Folgende durch `set <var> <value>` veränderbaren Variablen sind im Moment implementiert:

- `slope`: Der Wert für die Steigung der Sigmoide, der bei der Initialisierung des BP– und des ETANN–Layers verwendet wird.

- `rmin`, `rmax`: Bereich für die zufällige Initialisierung der Gewichte.

- `ttw`: Variable die das Trainingsverfahren des Kommandos `ttwtrain` auswählt. Möglichkeiten:
  `BOOTSTRAP`: Bootstrap–Training, die Pfade werden gleichmäßig aufgeteilt.
  `ADAPTIVE`: adaptives Training, Feintuning der Pfade.

- `pos_th`: positiver Schwellwert für eine Erkennung eines Wortes.

- `neg_th`: negativer Schwellwert, ab dem ein "false alarm" erkannt wird, der zu einem negativen Training führt.

- `use_etann` (`YES` oder `NO`): wählt aus, ob der ETANN für die Berechnung der ETANN–Layer verwendet wird (Forwardpass auf dem ETANN).

- `update_interval`: Anzahl der Trainingsdurchläufe, die zwischen dem Verändern der Gewichte auf dem ETANN liegen.

- `td_impl` (`EXTERN`, `INTERN`): Art der Implementierung von Time Delays auf dem ETANN. Die Wahl der Option `EXTERN` erfordert später externe Zwischenspeicher, bei `INTERN` werden die Time Delays in den Sample&Hold–Gliedern des Rückkoppelarrays gespeichert.

- `v_gain`: Spannung, die an V_{Gain} des ETANN angelegt wird und die Steigung des Sigmoids beeinflußt.

Literaturverzeichnis

[1] I. Bronstein, K. Semendjajew: Taschenbuch der Mathematik
Verlag Harri Deutsch, Thun und Frankfurt/Main.

[2] N. Mauduit, M. Duranton, J. Gobert, J. Sirat: Lneuro 1.0: a piece of hardware LEGO for building neural network systems
IEEE Neural Networks, März 1992.

[3] C. Carroll: A Neural Processor for Maze Solving
Analog VLSI Implementation of Neural Systems, edited by C. Mead, M. Ismail. Kluwer Academic Publishers.

[4] H.P. Graf, D. Henderson: A Reconfigurable CMOS Neural Network
IEEE International Solid–State Circuits Conference, 1990.

[5] J. Alspector, B. Gupta, R.B. Allen: Performance of a Stochastic Learning Microchip
Advances in Neural Information Processing Systems I, Morgan–Kaufmann, 1989.

[6] M. Yasunaga, N. Masuda, M. Yagyu, M. Asai, M. Yamada, A. Masaki: Design, Fabrication and Evaluation of a 5–inch Wafer Scale Neural Network LSI composed of 576 Digital Neurons
ICJNN 90, San Diego, Vol. 2, pp.527—535

[7] M. Yasunaga, N. Masuda, M. Yagyu, M. Asai, K. Shibata, M. Ooyama, M. Yamada, T. Sakaguchi, M. Hashimoto: A Self–Learning Neural Network Composed of 1152 Digital Neurons in Wafer–Scale LSIs
IJCNN 91, Singapore, Vol. 3, pp.1844—1849

[8] Alice M. Chiang: A CCD Programmable Signal Processor
IEEE Journal of Solid–State Circuits, Vol.25, No. 6, Dezember 1990, S.1510 ff.

[9] D. Hammerstrom: A VLSI Architecture for high–performance, low–cost, on–chip learning
Proc. IJCNN, San Diego, Juni 1990.

[10] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, J. Beer: The Ring Array Processor: A Multiprocessing Peripheral for Connectionist Applications
Journal of parallel and distributed computing, Vol. 14, 1992.

[11] Mark Holler, Simon Tam, Hernan Castro, Ronald Benson: An Electrically Trainable Artificial Neural Network (ETANN)
Proc. Int. Joint Conf. Neural Networks, 1989.

[12] Simon Tam, Bhusan Gupta, Herman Castro, Mark Holler: Learning on an Analog VLSI Neural Network Chip
IEEE, 1990.

[13] S. Tam, M. Holler, J. Brauch, A. Pine, A. Peterson, S. Anderson, S.Deiss: A Reconfigurable Multi–Chip Analog Neural Network; Recognition and Back–Propagation Training
IJCNN, 1992.

[14] J. Brauch: Neural Network Recognition of Objects based on Impact–Dynamics
Intel, 80170NX Neural Network Technology & Applications.

[15] D. Soo, R. Meyer: A Four–Quadrant NMOS Analog Multiplier
IEEE JSSC, Vol. SC–17, No. 6, Dez. 1882.

[16] Intel Datenblatt: 80170NX Electrically Trainable Analog Neural Network
Intel Corporation, 1991.

[17] T. Zeppenfeld, A. Waibel: A Hybrid Neural Network, Dynamic Programming Word Spotter

[18] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. Lang: Phoneme Recognition Using Time–Delay Neural Networks
IEEE Trans. on Acoustics, Speech and Signal Processing, Vol 37, No. 3. März 1989.

[19] B. Kröse, P. von der Smagt: An Introduction to Neural Networks
Vorlesungsskript Universität Amsterdam, September 1991.

[20] A. Waibel, K. Lee: Readings in Speech Recognition
Morgan Kaufmann Publishers, 1990.

[21] R. Lippmann, B. Gold: Neural–Net Classifiers Useful for Speech Recognition
ICNN, San Diego, 21–24 Juni 1987.

[22] M. Hoehfeld, S. Fahlman: Learning with Limited Numerical Precision Using the Cascade–Correlation Algorithm
CMU technical report CMU–CS–91–130.