Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für Biomedizinische Technik und Medizinische Informatik

Diplomarbeit

Entwicklung eines Praktikumsversuches zu Neuronalen Netzen auf der Basis von Borland Delphi

vorgelegt von:	Christian Heller
Studiennummer:	20887
Immatr.-Jahr:	1990
Seminargruppe:	BMT-91
Studienrichtung:	Biomedizinische Technik und Medizinische Informatik
Studiengang:	Elektrotechnik
Betreuer:	Dipl.-Ing. D. Steuer
verantwortlicher Hochschullehrer:	Prof. Dr. rer. nat. habil. G. Grießbach
eingereicht am:	16.07.1997

Danksagung

Ich möchte mich ganz herzlich bei meiner Mentorin Dunja Steuer bedanken, die mir, ob bei der Beschaffung notwendiger Literatur oder durch das Geben nützlicher Hinweise, stets hilfreich zur Seite stand.

Nicht unerwähnt bleiben sollte auch die gute Zuarbeit und konstruktive Mithilfe von Andreas Heyn, durch dessen Beitrag die Qualität der Arbeit stark beeinflusst wurde. Außerdem sei gedankt allen Freunden, die durch ihre Ratschläge zum Gelingen des Projekts beitrugen sowie Herrn Dr. H. J. Böhme für die hilfreichen Tips zur Daten-Codierung bei der Arbeit mit Neuronalen Netzen.

Schließlich gebührt großer Dank meiner Familie und Freundin Kas´ka, die sich während der vergangenen Monate um mein körperliches und geistiges Wohlergehen besonders gekümmert haben.

Vorwort

Die vorliegende Diplomarbeit ist das Ergebnis einer mehrmonatigen, intensiven Beschäftigung mit dem Aufgabengebiet „Neuronale Netzwerke“, welche die Entwicklung eines entsprechenden Versuches für den Einsatz im Laborpraktikum „Medizinische Informatik“ der TU Ilmenau zum Ziel hatte.

Dazu war, neben der oben genannten, auch eine Einarbeitung in Themen wie „Klassische Problemtypen für Neuronale Netzwerke“, die Grundlagen der Programmierung unter dem Entwicklungssystem „Delphi 2.0“, die Arbeitsweise von Spracherkennungs-Programmen und die Hilfeprogrammierung als eigenständiges Gebiet erforderlich.

Bei der Programmierung des Simulators wurde besonderes Augenmerk auf seinen modularen Aufbau, die anschauliche Darstellung der Zusammenhänge zwischen einzelnen Netzwerkparametern und auch die Erstellung eines Hilfesystems gelegt. Eine innerhalb einer kurz zuvor abgeschlossenen Studienjahresarbeit entstandene Delphi-Komponente konnte erfolgreich in das Programm integriert werden. Als zusätzliche Möglichkeit stand die Implementierung eines Moduls zur Spracherkennung, welches als Einzelworterkenner realisiert werden konnte.

Die Auswahl der Praktikumsaufgaben erfolgte nach Gesichtspunkten eines didaktisch möglichst sinnvollen Versuchsablaufs.

Diesen Punkten folgend, spiegelt die Gliederung der Arbeit im Wesentlichen drei Entwicklungsabschnitte wider: eine zweimonatige Einarbeitungsphase, die Programmierung der Simulationssoftware und das Aufstellen des Laborversuchs.

In Kapitel 1 werden – mit Blick auf die zu erstellende Praktikumsanleitung – Aufbau und Funktion Neuronaler Netze näher beschrieben. Der zweite Abschnitt belegt die Suche nach geeigneten Anwendungs- und Demonstrationsbeispielen für den Laborversuch. In der in Kapitel 3 folgenden Evaluierung werden die dann neu gesteckten Ziele beschrieben. Teil 4 legt daraufhin die Grundlagen der Spracherkennung dar. Das 5. Kapitel lässt den Leser die einzelnen Etappen der Programmiertätigkeit noch einmal nachvollziehen, bevor in einem

letzten Punkt eine rückblickende Zusammenfassung und Empfehlungen für weitergehende Arbeiten gegeben werden.

Die Bedienung des Programmes ist im beigefügten Handbuch näher beschrieben. Die Anleitung zum Versuch gibt zunächst einen logischen, wenn auch groben Überblick mit dem Ziel, dem unbedarften Praktikanten ein schnelles Eindenken in das Aufgabengebiet zu ermöglichen. Daran schließen sich, wie gewohnt, Vorbereitungs- und Versuchsaufgaben mit anschließender Auswertung an. Eine einführende Literaturliste ist dem letzten Kapitel zu entnehmen. Zur Hilfestellung für den Betreuer wurde ein Musterprotokoll angefertigt.

Hinweis: Die Arbeit wurde verfasst nach dem neuen amtlichen Regelwerk zur deutschen Rechtschreibung [Dude], welches seit Juli 1996 in einer Übergangsphase und ab dem 1. August 1998 seine volle Gültigkeit haben wird.

Hiermit versichere ich, die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel geschrieben zu haben.

Stützerbach, den 16.07.1997 _____

– Christian Heller –

Inhaltsverzeichnis

1 Neuronale Netzwerke 7
1.1 Einführung 7
1.2 Das menschliche Nervensystem 7
1.3 Modellierung eines Neurons 8
1.4 Erstellen Künstlicher Neuronaler Netze 10
1.4.1 Grundsätzliches 10
1.4.2 Systematisierung 11
1.4.3 Fehlerrückführungsnetz 13
1.5 Funktion 14
1.5.1 Lernvorgang 14
1.5.2 Einfluss der Parameter 16
1.6 Strukturoptimierungsverfahren 18
1.7 Anwendungen und Grenzen 20
1.8 Historie und Ausblick 21

2 Problemtypen 24
2.1 Klassische Schulprobleme 25
2.1.1 XOR-Problem 25
2.1.2 Paritätsproblem 27
2.1.3 Encoder-Decoder-Problem 27
2.2 Andere Anwendungen 27

3 Zwischenresümee 29

4 Spracherkennung 30
4.1 Biologischer Hintergrund 30
4.2 Systematisierung 31
4.3 Verarbeitungsstufen 32
4.3.1 Vorverarbeitung 32
4.3.2 Merkmalsextraktion/-reduzierung 33
4.3.3 Klassifikation/Mustervergleich 35
4.3.4 Kontextprüfung 35
4.4 Probleme 35

4.5 Aktuelle Systeme 36

5 Praktische Umsetzung 37

5.1 Revision 37

5.2 Hard- oder Software 38

5.3 Komponentenentwicklung 38

5.4 Simulator „Res Neuronum“ 40

5.4.1 Hauptprogramm 41

5.4.2 Modul Fehlerrückführung 42

5.4.3 Modul Spracherkennung 43

5.4.4 Hilfe-System 47

5.4.5 Setup-Programm 48

6 Zusammenfassung und Ausblick 49

7 Verzeichnisse 52

7.1 Abkürzungsverzeichnis 52

7.2 Symbolverzeichnis 53

7.3 Gleichungsverzeichnis 54

7.4 Tabellenverzeichnis 54

7.5 Abbildungsverzeichnis 54

7.6 Quellenverzeichnis 55

7.6.1 Literaturzitate 55

7.6.2 Andere Literaturverweise aus dem Text 57

7.6.3 Sonstige Quellen 58

7.6.4 Warenzeichen 60

Anhänge 61

A Thesen 61

B Benutzerhandbuch Res Neuronum 61

C Praktikumsanleitung 61

D Musterprotokoll 61

1 Neuronale Netzwerke

1.1 Einführung

Das Streben der Menschheit nach Wissen und Erkenntnis steht und fällt mit der Entwicklung des Großhirns ihrer Spezies, das sie als eigentliches Unterscheidungsmerkmal von anderen biologischen Formen abhebt.

Zwar mögen mehrere Arten spezielle Fähigkeiten besitzen, die sie gegenüber dem Menschen eindeutig bevorteilen, doch zeichnet sich keine andere, uns bis heute bekannte Lebensform durch eine Intelligenz aus, die mit der des Menschen auch nur annähernd vergleichbar wäre.

Daher ist es nicht verwunderlich, dass einer seiner Träume stets war, sich selbst – und das heißt insbesondere sein Zentralnervensystem – nachzubilden. Diese Aufgabe fällt in der aktuellen Forschung neben Medizinern, Biologen, Mathematikern und Informatikern auch den Biokybernetikern zu.

1.2 Das menschliche Nervensystem

Seit Menschengedenken erfasst unser Gehirn Informationen mit Hilfe der dem Menschen eigenen fünf Sinnesorgane Auge, Ohr, Nase, Mund und Haut. Sie statten den Organismus nicht nur mit den durch ihre Namen assoziierbaren Sinnen Sehen, Hören, Riechen, Schmecken und Tasten aus. Auch Empfindungsqualitäten wie Wärme, Körperlage, Beschleunigung oder Schmerz können wahrgenommen werden. So ist z.B. das Ohr neben dem Gehör- auch für den statokinetischen Sinn von Bedeutung, genau wie die Haut für Temperatur- und Schmerzsinn.

Unabdingbare Voraussetzung für diese Fähigkeiten ist das Vorhandensein von Sensorzellen, genannt Rezeptoren, welche die aus der Umwelt auf uns einströmenden Reize aufnehmen, wonach diese in kodierter Form an das Zentralnervensystem übertragen werden (vom animalen Nervensystem zur Regelung der willkürlichen Funktionen des Organismus oder vom vegetativen zur unwillkürlichen Regelung der Vitalfunktionen). Dort erfolgt die Informationsverarbeitung, welche eine Reaktion nach sich zieht, die meist durch das motorische System weitergegeben und als Muskelbewegung sichtbar wird.

Die angesprochene Informationsverarbeitung erfolgt sowohl im Gehirn als auch im Rückenmark. In ihm können nach neuesten Erkenntnissen z. T. Bewegungsabläufe der Gliedmaßen gespeichert werden. Zentraler Punkt ist und bleibt jedoch das Gehirn, genauer gesagt die Hirnrinde. Hier werden die Informationen erfasst und zueinander in Relation gesetzt. Das spätere Erkennen von wiederkehrenden Mustern und das Erlernen von Assoziationen zwischen ihnen hat hierin seinen Ursprung. Die entstehenden Abbildungen kann man wie folgt einteilen [Kinn]: mathematisch berechenbare, intuitiv erfassbare und chaotische.

Mit zweiterer Gruppe beschäftigen sich derzeit im wesentlichen zwei Wissenschaftsfelder, das der „unscharfen Logik“ (*Fuzzy-Logic*) und jenes der „Künstlichen Neuronalen Netze“. Während die unscharfe Logik Sachverhalte auf symbolische Weise darstellt und Zusammenhänge direkt benennt, lassen Neuronale Netze nur eine verteilte Darstellung zu. Die Informationen über Zustände und Beziehungen sind im ganzen Netz verteilt; ein einzelnes Neuron repräsentiert stets nur einen Bruchteil davon [Hoff].

Wie man Neuronale Netze modelliert, darauf soll im Folgenden näher eingegangen werden. Zuvor sei einschränkend erwähnt, dass, wie fast alle Modelle, auch die hier beschriebenen nur ein grob verallgemeinertes Pendant ihres in der Natur real auftretenden Objektes sein können.

1.3 Modellierung eines Neurons

Betrachtet man den Aufbau einer menschlichen Nervenzelle, so besteht diese – wie gewöhnliche Zellen – aus Zellkern, Membran, Zytoplasma u. a. Zusätzlich enthält sie jedoch Bauteile, die sie zum strukturellen Grundelement des Nervensystems machen. Dazu zählen die Dendriten als kleine Fortsätze zum Empfang nervöser Erregungen, welche ihnen entweder durch elektrische Synapsen (Depolarisation der postsynaptischen Membran) oder chemische Synapsen (Neurotransmitter ändern Permeabilität der Membran für Ionen) unipolar und diskontinuierlich übertragen werden. Synapsen sind quasi Kontaktzellen, deren Wirkung charakteristisch für jedes Neuron ist und Signale exzitatorisch oder inhibitorisch beeinflussen kann. In der Nervenzelle baut sich ein in seiner Größe von der Anzahl der über die Synapsen eingehenden Signale abhängiges Membranpotential auf. Schließlich leitet der Neurit, auch als Axon bekannt, die resultierende Information weiter zur nächsten Zelle oder motorischen Endplatte, indem an den – angenommen chemischen – Synapsen Neurotransmitter ausgeschüttet werden.

Gemäß diesen Überlegungen formulierten McCulloch und Pitts im Jahre 1943 als Erste: „Die Verbindungen eines Neurons nehmen eine Aktivierung e_j mit einer bestimmten Stärke w_j von anderen Neuronen auf, summieren diese und lassen dann am Ausgang a des Neurons eine Aktivität entstehen, sobald die Summe vorher einen Schwellwert ϑ überschritten hat.“ [Brau]

Diese Betrachtung eines Neurons als eine Art Addierer mit Schwellwert gilt bis heute als Grundmodell aller Neuronalen Netzwerke (Abbildung 1.1).

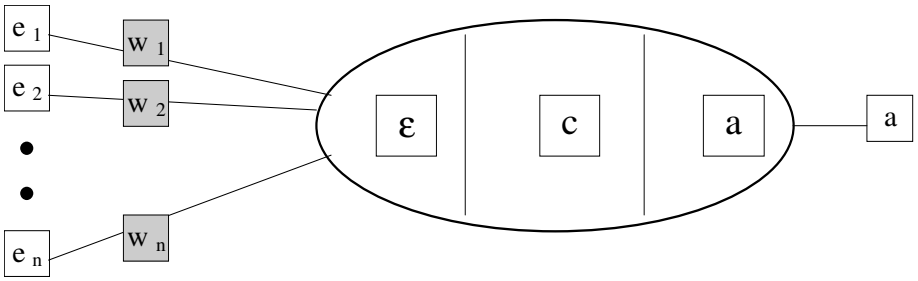

Abbildung 1.1: Struktur eines künstlichen Neurons

Im Modell sind drei Funktionen von Bedeutung, mit deren Hilfe die nötigen Werte berechnet werden. Für den linearen Neuronen-Typ lauten sie:

$$\varepsilon = \vec{w}\vec{e} = \sum_{j=1}^{n} w_j e_j$$ **Gleichung 1.1: Eingangsfunktion (Skalarprodukt)**

$$c = s\varepsilon$$ **Gleichung 1.2: Aktivierungsfunktion (linear)**

$$a = \sigma(c - \vartheta)$$ **Gleichung 1.3: Ausgangsfunktion (linear)**

Der effektive Eingang ε modelliert die Wirkung der Synapsen, wohingegen die Aktivierungsfunktion c das – im häufigsten Falle als proportional steigend angenommene – Membranpotential verkörpert. Für die Weitergabe des Signals ist im künstlichen Neuron die Ausgangsfunktion a zuständig; in der realen Nervenzelle geschieht das, wie schon erwähnt, durch Neurotransmitter. Häufig werden Aktivierungs- und Ausgangsfunktion zur „Transferfunktion“ zusammengefasst.

1.4 Erstellen Künstlicher Neuronaler Netze

1.4.1 Grundsätzliches

Koppelt man nun diese einfachen Neuronen-Elemente, so können sie parallel arbeiten und ein Netzwerk bilden, dessen Funktion größtenteils durch die Verbindungen zwischen den Neuronen bestimmt wird [Matl]. Zur neuronalen Verschaltung kommt zum einen die Divergenz (ein Neuronausgang verzweigt zu mehreren Eingängen anderer Neuronen) und zum anderen die Konvergenz (eine künstliche Nervenzelle erhält Signale von vielen anderen) in Frage.

Im Netz gibt es Eingangs- und Ausgangsneuronen. Die Ausgangsneuronen geben ihre Signale direkt an die Netzausgänge weiter. Bei den Eingangsneuronen unterscheidet man zwei Konventionen. In der ersten verteilt sich jeder Netzeingang auf mehrere Neuroneingänge. Man kann das vermeiden, indem – wie in der zweiten möglichen Konvention – für jeden Netzeingang ein zusätzliches Eingangsneuron, auch „Verteilungsneuron“ genannt, eingeführt wird, dessen einzige Aufgabe es ist, den Netzeingang auf die folgenden Neuronen zu verteilen (Abbildung 1.2). Auf die Funktion des Netzes hat es keinerlei Einfluss.

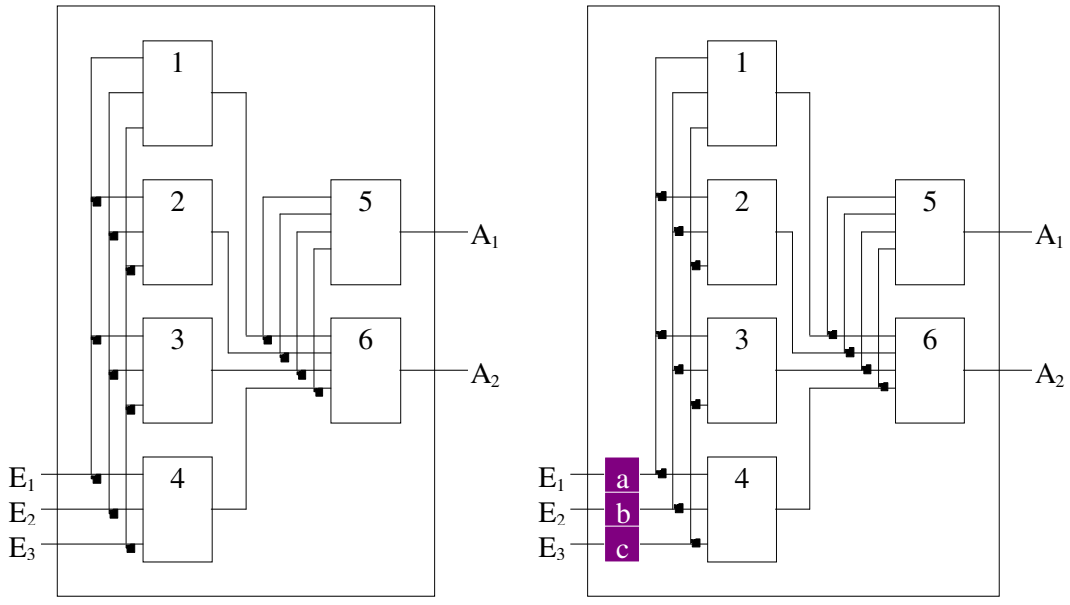

Abbildung 1.2: Netzwerk ohne (2-schichtig) und mit Verteilungsneuronen (3-schichtig)

Weiterhin legt man – eine lineare Aktivierungsfunktion vorausgesetzt – häufig ein sogenanntes „Bias-Neuron“ als zusätzlichen Eingang mit dem festen Wert eins an ein Neuron an mit dem Zweck, die Schwelle der Ausgangsfunktion durch das negative Gewicht dieses Eingangs zu ersetzen. Dadurch lässt sich zum einen die Ausgangsfunktion leichter

implementieren und zum anderen kann die Schwelle ϑ in den Lernvorgang einbezogen und flexibel gestaltet werden. Das ist notwendig z. B. für den Fall, dass ein komplett aus Nullen bestehender Eingangsvektor eine Eins am Ausgang erzeugen soll. Hier kämen die Gewichtsänderungen nicht zum Tragen, da alle mit null multipliziert würden und durch einen festen Schwellenwert größer oder gleich null könnte das Element nicht aktiviert werden.

1.4.2 Systematisierung

Es ist möglich, dass Netze ihre Struktur selbst bilden. Manche der dazugehörigen Algorithmen sind eng verwandt mit den „Strukturoptimierungsverfahren“ (1.6), zu denen man sie z. T. zählt. Im Weiteren soll jedoch über Netzwerke gesprochen werden, deren Struktur unveränderlich vorgegeben ist, die also lediglich Kennwerte lernen.

Um die räumliche Organisation der menschlichen Großhirnrinde nachzubilden, kann jedem Neuron ein Ort in einem fiktiven Raum zugeordnet werden. Das wird möglich, wenn man die Gewichte der Verbindungen je nach Entfernung der Neuronen voneinander setzt, also je weiter desto kleiner. Man erhält räumlich organisierte Netze. Sie kommen dem Aufbau des menschlichen Hirns momentan am nächsten. Allerdings sind sie relativ kompliziert zu bilden, weswegen sich in der jüngeren Vergangenheit hauptsächlich die semantisch organisierten Netze verbreitet haben.

Erfüllen Neuronen innerhalb eines Netzes gemeinsam eine bestimmte Aufgabe, so kann man diese zu Schichten zusammenfassen. Man unterscheidet Eingangs-, Ausgangs- und Verborgene Schichten.

Gewöhnliche Netzmodelle sind meist vorwärtsgekoppelt. Führt man hingegen Signale von Neuronausgängen zurück auf vom Signalfluss bereits berührte Neuronen, so sagt man, das Netz habe Rückkopplungsschleifen, man spricht vom „rückgekoppelten Netz“. Dazu meint Hoffmann [Hoff] in seinem Buch:

> Grundsätzlich kann jeder Neuronausgang mit jedem Neuron des Netzes verbunden sein und zugleich einen Netzausgang bilden. Außerdem kann jeder Netzeingang an alle Neuronen des Netzes angeschlossen werden. Es ist sogar möglich, dass ein Neuron seinen Ausgang auf sich selbst zurückführt (Selbstrückkopplung). Solche Netze heißen ‘vollständig verbunden’.

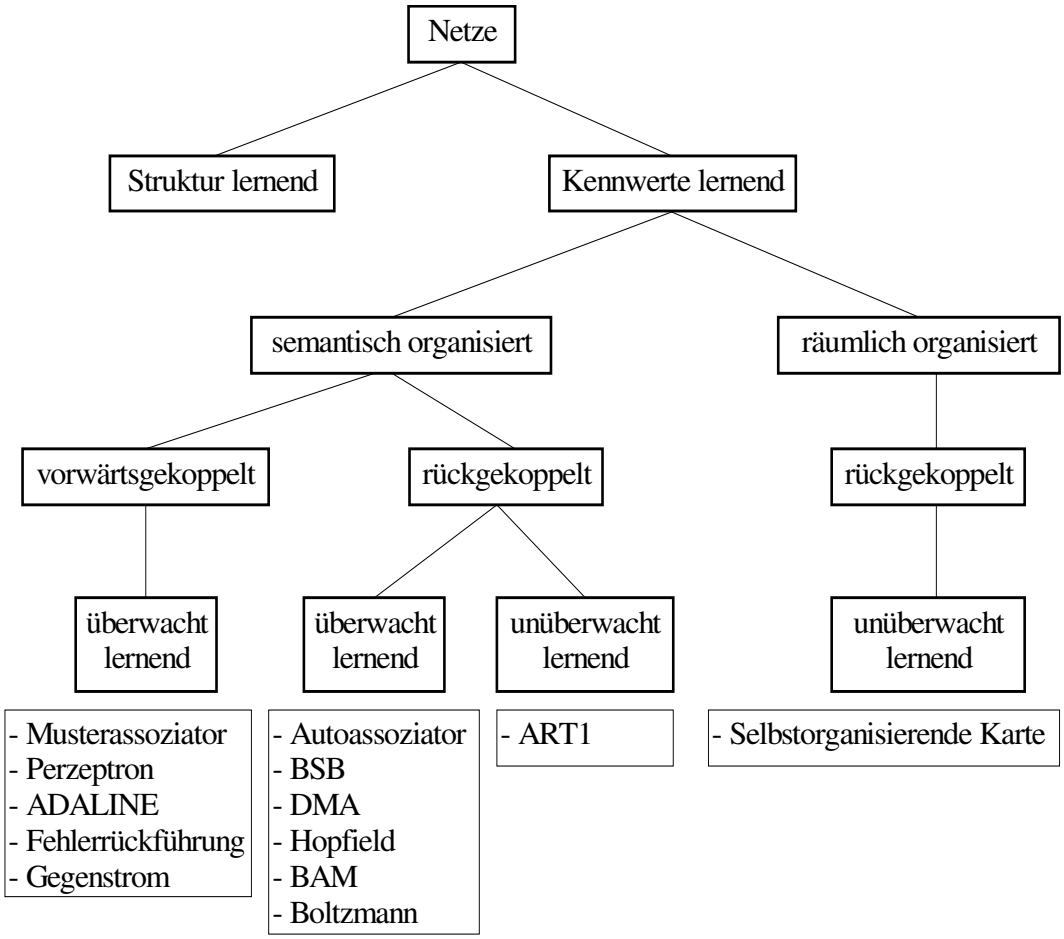

Abbildung 1.3: Arten Neuronaler Netze – klassifiziert nach Aufbau und Lernmethode

Ihre eigentliche Aufgabe des unter 1.2 angesprochenen Erkennens von wiederkehrenden oder ähnlichen Mustern (Reproduktion) können die Netze nur lösen, wenn dem ein Erlernen von Assoziationen zwischen den Mustern vorausgegangen ist (Lernen). Für beide Vorgänge existieren Methoden, die je nach Art des angewandten Netzes differieren.

Beim Belehren unterscheidet man zusätzlich zwischen den Methoden für überwachtes und für unüberwachtes Lernen (Abbildung 1.3). Um ein Netz überwacht zu belehren, wird ein bekanntes Muster an den Netzausgang angelegt. Hingegen erwartet man von einem ohne Lehrer lernenden Netz, dass es selbständig Ähnlichkeiten in unbekannten Mustern herausfindet und diese klassifiziert.

Abbildung 1.3 gibt einen auszugsweisen Überblick über die während der letzten Jahrzehnte entstandene Fülle unterschiedlichster Netzwerk-Strukturen. Das zum heutigen Zeitpunkt populärste Modell ist das „Fehlerrückführungs-Netz“, welches in 80 % aller Problemstellungen zur Anwendung kommt.

1.4.3 Fehlerrückführungsnetz

Häufig werden vorwärtsgekoppelte Netze ohne verdeckte Schichten als „Perceptron“ bezeichnet. Die Fehlerrückführungsnetze (Abbildung 1.4) sind Vertreter dieser Gruppe und werden auf Grund ihrer Struktur oft als „Mehrschichtiges Perceptron“ bezeichnet. Man kann sie mit der entsprechenden Lernregel trainieren, um in der Reproduktionsphase erstens Funktionen zu approximieren, zweitens spezifische Ein- und Ausgangsvektoren zu assoziieren oder drittens Eingangsvektoren in einer gegebenen Form zu klassifizieren [Matl].

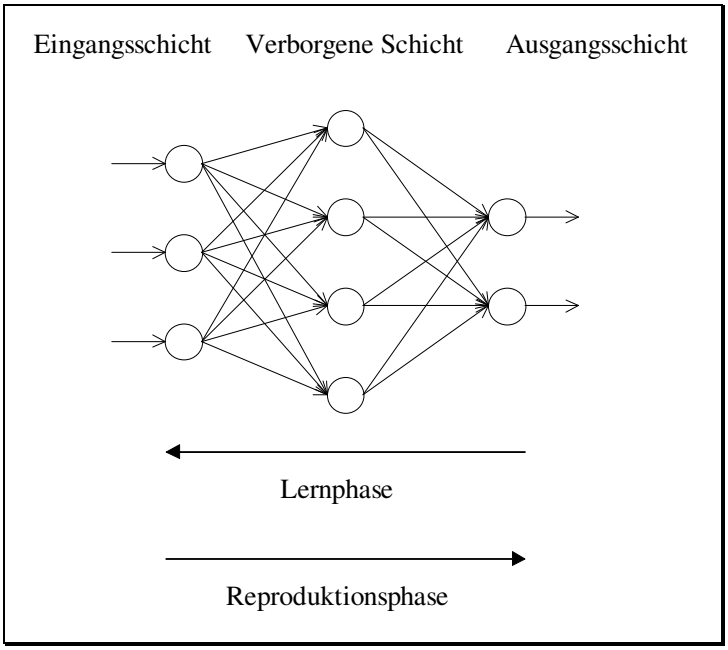

Abbildung 1.4: Struktur und Arbeitsphasen eines Fehlerrückführungsnetzes

Auf Grund der genannten Fähigkeiten, weiten Verbreitung und auch wegen der guten Anschaulichkeit sei dieses Modell für die folgenden Erklärungen zu Grunde gelegt.

1.5 Funktion

1.5.1 Lernvorgang

Drei Voraussetzungen werden angenommen: Effektiver Eingang für alle Neuronen ist das Skalarprodukt, Aktivierungsfunktion die Identität. Das Netz kann nichtlineare[1], aber differenzierbare[2] Ausgangsfunktionen verwenden; meist nimmt man die Fermi-Funktion (Gleichung 1.4). Hier ist *m* das Minimum und *M* das Maximum der Ausgangsfunktion. (Zu beachten ist, dass *M* mindestens genauso groß wie – besser größer als – der höchste Wert im gewünschten Zielvektor sein muss; sonst kann das Netz keine Lösung erreichen! Entsprechendes gilt für *m*.)

$$a(c) = m + \frac{M - m}{1 + e^{-4\sigma\frac{c-\sigma}{M-m}}}$$

Gleichung 1.4: Erweiterte Fermi-Funktion

Legt man eines der zu lernenden Musterpaare μ an Ein- (E) und Ausgang (A) eines Fehlerrückführungsnetzes an, so wird in den Neuronen der ersten Schicht mit Hilfe der Eingangsfunktion aus den Werten des Eingangsmusters e_j und den entsprechenden Gewichten w_j der effektive Eingangswert ε berechnet. Dieser fließt als Parameter in die Aktivierungsfunktion der Neuronen ein, welche die Aktivität c als Ergebnis liefert. Schließlich erhält man durch Anwendung der Ausgangsfunktion auf die Aktivität den Ausgangswert a eines jeden Neurons. Besteht das Netz aus mehreren Schichten, so pflanzt sich das Signal in ihnen auf die gleiche Art und Weise fort, wobei jeweils die Ausgänge der Vorgängerschicht zu den Eingangswerten der Nachfolgerschicht werden.

Die endgültigen Ausgangswerte a_i der Neuronen der letzten Schicht werden nun verglichen mit jenen des angelegten Trainingsmusters. Liegt eine Abweichung von diesen Sollwerten S_i vor, so werden (mit Hilfe der Ableitung a'_i der Ausgangsfunktion des Neurons und dessen Aktivität) zunächst die Fehlermaße δ_i^μ für die Neuronen der Ausgangsschicht (Gleichung 1.5) und mit ihnen die Fehlermaße der jeweiligen Vorgängerschichten (Gleichung 1.6) berechnet. Daher der Name Fehlerrückführungsnetz.

[1] Sinnvoll, da ein mehrschichtiges Netz aus linearen Neuronen immer durch ein einschichtiges ersetzt werden kann.

[2] Notwendig, weil in der Lernregel die Ableitung der Ausgangsfunktion vorkommt.

Bei Aved'yan [Aved] heißt es dazu weiter:

> The learning process of multilayer neural networks is carried out by comparison of the network's outputs (i. e., the outputs of the neurons of the last layer) with the corresponding teacher's instructions. Information about the ideal outputs of the neurons of hidden layers is absent. Despite this, knowledge of the structure of a MNN [Multilayered Neural Network] allows us to calculate the correcting signals not only for the neurons of the last layer but also for the neurons of the other layers.

$$\delta_i^\mu = (S_i^\mu - a_i^\mu)a_i'(c_i^\mu)$$ **Gleichung 1.5: Fehlermaß für Neuron i der Ausgangsschicht**

$$\delta_i^\mu = a_i'(c_i^\mu)\sum_k \delta_k^\mu w_{ki}$$ **Gleichung 1.6: Fehlermaß in verborgenen Schichten**

Die Summe in Gleichung 1.6 erstreckt sich über die Neuronen *k* der jeweils folgenden Schicht; die w_{ki} sind die Gewichte der folgenden Schicht vor der Gewichtsanpassung.

Nun ist man imstande, mit der für das Netz typischen Lernregel (Gleichung 1.7) die notwendigen Änderungen der Lernparameter zu bestimmen. In der Regel werden lediglich die Gewichte neu gesetzt (Gleichung 1.8); modifiziert werden können aber auch andere Parameter wie beispielsweise die Schwelle oder auch Steigung in der Ausgangsfunktion der Neuronen. Mit den neuen Werten erfolgt dann der nächste Lernschritt usw., bis letztendlich Ist- und Sollwerte übereinstimmen oder der Fehler δ_i eine zuvor angegebene Grenze unterschreitet.

$$\delta w_{ij} = \eta\sum_\mu \delta_i^\mu e_j^\mu$$ **Gleichung 1.7: Fehlerrückführungs-Lernregel**

$$w_{ij}^{neu} = w + \delta w_{ij} = w + \eta\sum_\mu \delta^\mu e_j^\mu$$ **Gleichung 1.8: Neuberechnung der Gewichte**

In der Praxis fallen natürlich nicht nur einzelne Muster, sondern ganze Mustersets zur Verarbeitung und Auswertung an. Demnach ergeben sich zwei Möglichkeiten für einen Lernschritt: Beim „einfachen" (direkten) wird jeweils ein Muster nach obigem Schema abgearbeitet, dann werden die Gewichte neu gesetzt und das nächste Muster kann folgen. Demgegenüber reproduziert man das Netz beim „kumulativen" (epochenweisen) Lernschritt für alle Muster des Sets, speichert die Zwischenergebnisse, berechnet dann alle Fehlermaße und ändert schließlich in einem Zuge alle Gewichte. Vorteilhaft bei diesem Vorgehen ist, dass der tatsächliche Fehler über alle Muster berücksichtigt wird. Allerdings steigt mit der Anzahl von Mustern auch die Menge der notwendigen Rechenoperationen und der Speicherbedarf.

Um das Netz mit einfachen Lernschritten zu belehren, ist die Reihenfolge, in der die Muster dem Netzwerk angeboten werden, zufällig zu wählen. Bei einer geordneten Abfolge der Muster treten nach jedem Lernschritt wiederkehrende, zyklische Gewichtsänderungen ein. Das ist für einen optimalen Lernschritt nicht sinnvoll und verlängert die Lernzeit.

1.5.2 Einfluss der Parameter

Besonderes Augenmerk ist zu richten auf die Initialisierung der Gewichte. Die Fehlerrückführungs-Lernregel verlangt, falls alle Neuronen dieselbe Ausgangsfunktion haben (was meist stillschweigend vorausgesetzt wird), Gewichte, die nicht nur ungleich null, sondern sogar untereinander verschieden sein müssen. Wäre das nicht der Fall, würden alle Neuronen einer Schicht den gleichen Ausgang liefern und man könnte alle – bis auf eines – weglassen, ohne dass sich an den Reproduktionseigenschaften des Netzes etwas änderte. Die Initialisierung aller Gewichte mit demselben Wert führt also zu einer wesentlichen Einschränkung der Möglichkeiten des Netzes; gewöhnlich wählt man sie deshalb zufällig.

Zur anschaulichen, graphischen Darstellung einer Gewichtsmatrix kann man ihre Komponenten in einer Ebene anordnen. Positive Werte werden durch weiße, negative durch schwarze Quadrate repräsentiert; die Größe der Quadrate gibt den Absolutwert an [Hoff]. Diese Darstellung ist unter der Bezeichnung „Hinton-Diagramm“ verbreitet.

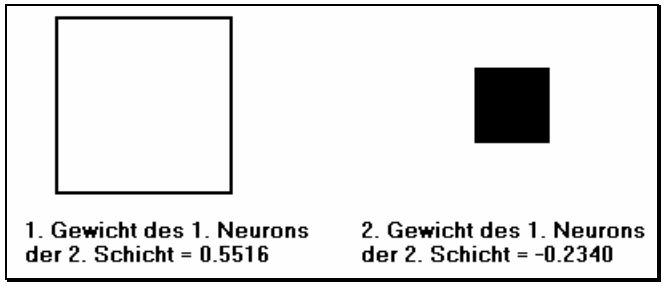

Abbildung 1.5: Hinton-Diagramm mit zwei Gewichten

Das Training führt häufig zu einem lokalen, statt globalen Minimum in der Kostenfunktion. Unter dieser, auch „Fehlergebirge“ genannten Funktion versteht man die Darstellung des Fehlermasses über dem hochdimensionalen Gewichtsraum. Sie ist also ein Mass für die Abweichung des tatsächlichen Ausgangs vom gewünschten. Zusätzliche Neuronen vergrößern die Freiheit des Netzwerkes (Variablenanzahl in der Fehlerfunktion) und damit die Chance, dass selbst ein lokales Minimum nur einen kleinen Fehler liefern wird. Allerdings steigt mit der Zahl zusätzlicher, verborgener Neuronen auch die der Nebenminima und somit die Wahrscheinlichkeit, dass der Lernvorgang in ein solches Nebenminimum läuft.

Implementiert man ein sogenanntes „Momentum“ in das Netzwerk, so verringert das die Wahrscheinlichkeit, dass ein Netzwerk in einem flachen, lokalen Minimum (kurzfristige Schwankung) des Fehlergebirges hängenbleibt und somit auch die Trainingszeiten. Der Momentumterm wird in die Lernregel eingefügt und berücksichtigt die Änderung der Gewichte vom vorhergehenden Lernschritt (Gleichung 1.9). Dies gleicht der Wirkung eines Tiefpasses. Die aktuell berechnete Gewichtsänderung kann so in die längerfristig optimalere Richtung verlaufen. Für den Parameter μ ist ein möglichst großer Wert zwischen 0 und 1 zu wählen.

$$\delta w_{ij} = \eta \delta_i e_j + \mu \delta w_{ij}^{alt}$$ **Gleichung 1.9: Lernregel mit Momentumterm**

Die Grösse der aktuellen Gewichtsänderung δw_{ij} wird jedoch in erster Linie beeinflusst von der Lernrate η. Sie wird der Lernregel als Faktor zugeordnet, um eine schnelle Konvergenz zu erreichen und darf nicht null betragen. Ein zu großer Wert für die Lernrate resultiert in instabilem Lernen. Mit einem kleinen Wert wird die Genauigkeit sehr hoch, doch benötigt das Netz viele Iterationsschritte und verursacht so zu lange Trainingszeiten, wie in [Matl] zu lesen steht. Die ebd. vorgeschlagene, rekursive Lernrate verringert die Trainingszeit, indem sie die Lernrate ausreichend hoch hält, während die Stabilität gewahrt bleibt. Dazu wird die Entwicklung des globalen Fehlers während der letzten Lernschritte betrachtet. Verlief sie kontinuierlich in eine Richtung, d. h. erhöhte oder verringerte sie sich über mehrere Lernschritte hinweg, so wird die neue Lernrate bestimmt durch Multiplikation mit einem entsprechenden Faktor. Dieser hat einen Wert größer als eins (idealerweise 1.04), wenn sich der Fehler verringerte, das Netz also „auf dem richtigen Wege“ ist, was eine Beschleunigung des Lernprozesses zur Folge hat. Erniedrigt wird die Lernrate im Falle eines größer werdenden Fehlers, indem sie mit einem Faktor kleiner als eins (empfohlen 0.7) multipliziert wird.

Nicht zu verwechseln mit der rekursiven Lernrate ist das Verfahren des adaptiven Lernens. Es basiert auf den adaptiven Schätzverfahren, die eine rekursive Methode der Signalanalyse zur Beschreibung von Zeitreihen darstellen. Sie sind relativ robust gegenüber verschiedenen Signalstrukturen, stellen also gleichermaßen für deterministische und stochastische Signale einen algorithmischen Lösungsansatz dar [Steu]. Wegen ihrer rekursiven Gestalt ermöglichen sie eine Online-Auswertung der Zeitreihe und aufgrund der Adaptionseigenschaften eine schnelle Anpassung an Veränderungen. Mathematischer

Hintergrund der adaptiven Verfahren sind über Methoden der stochastischen Approximation gewonnene, rekursive, konsistente Schätzfunktionen für Kenngrößen stochastischer Prozesse. Die Schätzfunktion zur Bestimmung des adaptiven Mittelwertes beispielsweise hat die in Gleichung 1.10 gegebene Form. Anhand des adaptiven Mittelwertes ist nun eine Beurteilung des Lernverhaltens Neuronaler Netze möglich (Näheres dazu in [Steu2]). Im vorliegenden Fall verkörpert δw_n die zu berechnende Schätzgröße „Gewichtsänderung“ und δw_{n-1} enthält den Wert dieser Schätzgröße vom vorhergehenden Lernschritt. Durch δw_n^{Signal} wird der aktuelle Wert im Signalverlauf der Gewichtsänderung vertreten; c wird als Adaptionskonstante bezeichnet. Anstelle der Gewichte könnten natürlich auch andere, lernbare Parameter wie die Schwelle σ oder Steigung s stehen. Ergebnis des adaptiven Lernens ist ein stabilisierter Verlauf der Fehlerkurve, in welcher große Sprünge vermieden werden, wodurch häufig erst ein erfolgreiches Lernen möglich wird.

$$\delta w_n = \delta w_{n-1} + \frac{1}{c}(\delta w_n^{Signal} - \delta w_{n-1})$$ **Gleichung 1.10: Adaptiver Mittelwert**

Präsentiert man dem Netz jetzt Muster, die ihm völlig unbekannt sind, muss es nicht neu trainiert werden. In einem solchen Fall antwortet es mit einem Ausgangswert, dessen zugehöriger, vorher erlernter Eingang dem gerade angelegten Eingangswert am ähnlichsten ist. Diese Möglichkeit, ein Netzwerk auf repräsentative Eingangs-/Zielpaare zu trainieren, ist als seine „Generalisierungsfähigkeit“ oder Eigenschaft der Verallgemeinerung bekannt.

Zusammenfassend kann man also sagen, dass sowohl das Neuron-Modell als auch die Architektur eines Neuronalen Netzes beschreiben, auf welche Art und Weise dieses Netzwerk seine Eingangs- in Ausgangsgrößen transformiert. Mit anderen Worten erlangt das Netz eine Lösung, indem es seine Eingangs- und Ausgangswerte einander zuordnet, wobei die jeweils zu lösende Zuordnungsaufgabe die Anzahl der Netzwerkein- wie auch -ausgänge bestimmt.

1.6 Strukturoptimierungsverfahren

Seit erwiesen ist, dass für viele, mit Neuronalen Netzen lösbare Aufgaben eine optimale Struktur ermittelt werden kann, gehen die Bestrebungen zunehmend dahin, diese Suche einer Idealstruktur zu automatisieren [Hell]. Doch was bedeutet der Begriff „Optimale Struktur“ in diesem Zusammenhang? Wie schon in Kapitel 1.5.2 erwähnt wurde, kann durch

die Erhöhung der Anzahl der Neuronen bzw. Schichten erreicht werden, dass ein Fehlerrückführungsnetz bessere Ergebnisse liefert. Allerdings können sich die so erworbenen Vorteile schnell umkehren in einen Nachteil, genannt „Overfitting-Phänomen“. Dieses Überladen mit Neuronen führt letztendlich zu einem schlecht verallgemeinernden (nicht generalisierungsfähigen) Netzwerk.

Deswegen verwendet man Optimierungsmethoden – um die Netzwerkstruktur zu vereinfachen und das Lernen zu beschleunigen. Dazu merken Jutten und Fambon [Jutt] an:

> While constructive methods try to build up a minimal model by adding successive parameters (or units), the pruning approach starts off with an already built model, and tries to extract its essence by pruning useless parameters.

Daraus lässt sich eine erste Systematisierung herleiten. Man unterscheidet also zwischen Verfahren, die dem Netz schon in der Konstruktionsphase die optimale Struktur verleihen und solchen, die aus einem fertiggebauten Netzwerk unwichtige Parameter heraustrennen (*Pruning*). Letztere kann man weiter untergliedern in die Gruppe der indirekten (Gewichte werden entfernt) und direkten Methoden (Neuronen werden entfernt). Der Ausdünnungs-Prozess erfolgt bei den indirekten Methoden während oder nach dem Lernen.

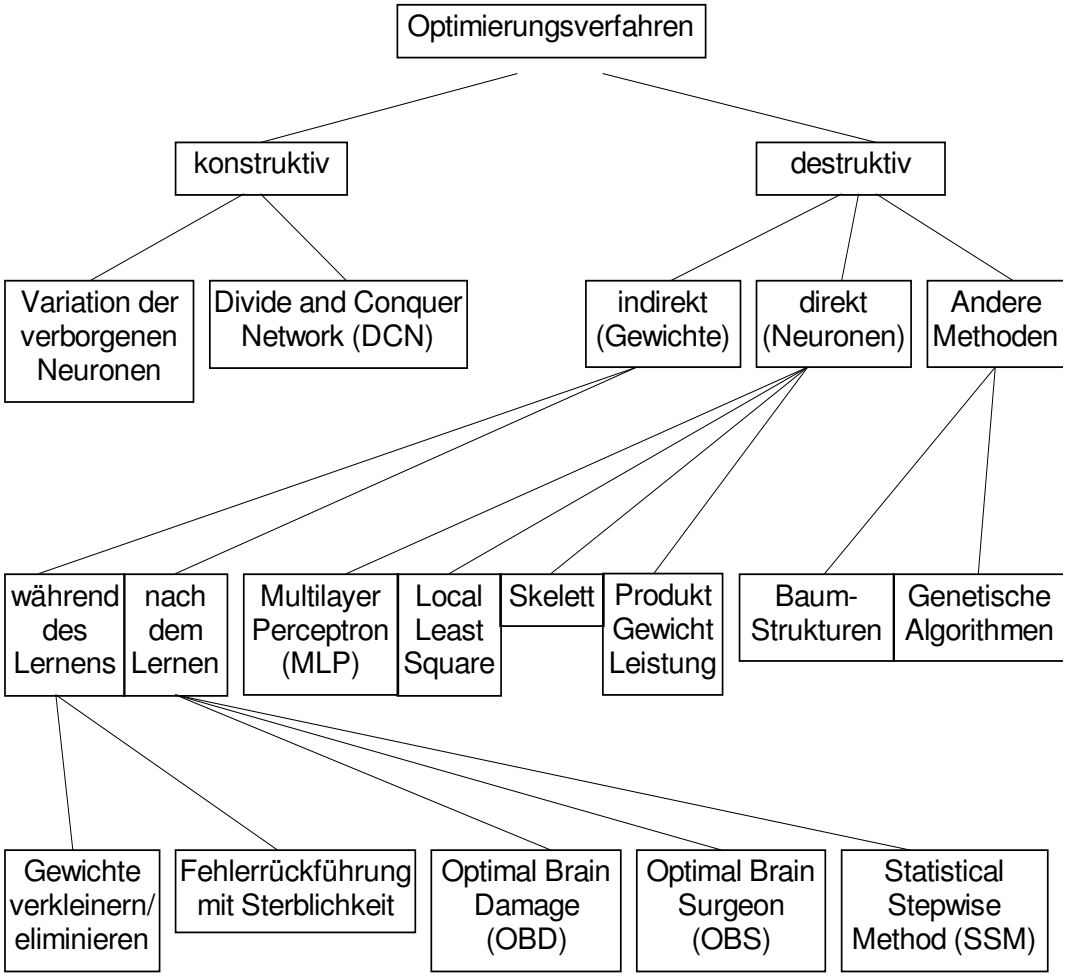

Abbildung 1.6: Methoden zur Strukturoptimierung

1.7 Anwendungen und Grenzen

Mit dem Aufkommen der Rechentechnik wurde die Möglichkeit gegeben, die Funktion Neuronaler Netzwerkmodelle in Simulationen nachzuvollziehen und zu prüfen. Damit war auch der Weg frei für ihre praktische Anwendung.

Die unter 1.2 angesprochenen, mit konventionellen Ansätzen nicht lösbaren Zuordnungsaufgaben der lediglich intuitiv erfassbaren Abbildungen (häufiges Beispiel: Beurteilen der Qualität einer Schweißnaht durch einen Fachmann) findet man in allen Bereichen der Gesellschaft. Dementsprechend universell ist auch der Anwendungsbereich Neuronaler Netze.

Zu den wichtigsten Gebieten zählen heute [Kinn] die Regelungstechnik, Robotik, Medizin, Meteorologie, Chemie, Wirtschaft und das Bankenwesen, die Sprachwissenschaft und die Informatik. Innerhalb dieser gilt es, Aufgaben wie die Steuerung von Prozessen,

Muster- und Zeichenerkennung, Klassifikation und Qualitätskontrolle, Erstellung von Prognosen, Sprachsynthese und -analyse, Datenkompression oder auch das Herleiten von Spielstrategien und musikalische Komponieren zu lösen.

Von dem ursprünglich verfolgten Ziel der Imitation des menschlichen Gehirns mit seinen 10^{11} Neuronen ist man jedoch, nicht zuletzt durch die immer noch unzureichenden Hardwaremöglichkeiten[3], bis auf unbestimmte Zeit noch weit entfernt. Die Forschung beschränkt sich daher zunächst auf die Nachbildung bescheidenerer Nervenstrukturen, beispielsweise von Lebewesen wie Schnecken, Blutegeln oder Tintenfischen mit nur etwa 10000 Neuronen bzw. im Menschen selbst auf die Bewältigung von Teilaufgaben wie z. B. die Unterstützung des Gehörs tauber Menschen [Frey].

Außerdem wird der Einsatz Künstlicher Neuronaler Netze für Aufgabenstellungen, die ein deterministisches Verhalten benötigen – etwa die Überwachung von Kenndaten in Kernkraftwerken –, zunehmend in Frage gestellt. Da die Netze ausschließlich über Beispieldaten lernen, lassen sich über ihr Verhalten bei unbekannten Eingabemustern lediglich statistische Aussagen treffen.

1.8 Historie und Ausblick

Die Entwicklung Künstlicher Neuronaler Netze erstreckt sich mittlerweile über fünf Jahrzehnte, geht man davon aus, dass McCulloch und Pitts mit ihrer Aussage (1.3) im Jahre 1943 den Grundstein legten [McCu].

Der Psychologe Donald Hebb stellte in seinem Buch [Hebb] 1949 die berühmte Hypothese auf: „Die synaptische Eigenschaft (Verstärken oder Hemmen) ändert sich proportional zum Produkt von prae- und postsynaptischer Aktivität.“, woraus die „Hebbsche Lernregel“ abgeleitet wurde, die in dieser Form allerdings noch nicht zu den gewünschten Ergebnissen führte.

$$\delta w_{ij} = \eta S_i E_j$$ **Gleichung 1.11: Hebbsche Lernregel**

Eine modifizierte Lernregel (Delta-Lernregel) wandte F. Rosenblatt [Rose] 1958 in seinem „Perceptron“ genannten Netzwerk an. Dieses enthielt lernfähige Gewichte, das

[3] Zwar liegen Simulationsprogramme auf Rechnerbasis in ihrer Verarbeitungsgeschwindigkeit um ein Vielfaches höher als reale biologische Nervenstrukturen, doch arbeiten letztere massiv parallel durch ihre Milliarden von Prozessoren (Neuronen).

Lernverfahren konvergierte in endlich vielen Schritten und außerdem war das Netz fehlertolerant und lieferte sogar für leicht modifizierte Eingaben richtige Ausgabewerte.

Basierend auf dem Perceptron entwickelten B. Widrow und M. Hoff 1960 das „ADALINE“ (*Adaptive Linear Element*). Es handelte sich um ein einstufiges Netz, an dessen Ein- und Ausgängen die binären Werte -1 und +1 anliegen konnten [Widr]. Die Wissenschaftler formulierten die Delta-Lernregel in der folgenden Form, die häufig auch „Widrow-Hoff-Lernregel“ genannt wird:

$$\delta w_{ij} = \eta(S_i - A_i)E_j$$ **Gleichung 1.12: Delta-Lernregel**

Mit M. Minski und S. Paperts Nachweis, dass sich einige wichtige logische Aussagefunktionen (z. B. die Bool’sche XOR-Funktion) mit dem Formalismus dieser Netztypen nicht beschreiben ließen [Mins], erfuhr die Theorie der Neuronalen Netze ab 1969 einen zunehmenden Bedeutungsverlust.

Trotzdem erstellte man weitere Modelle zum besseren Verständnis der Vorgänge im Gehirn. So sind die von T. Kohonen in dieser Zeit entwickelten, selbstorganisierenden Netze [Koho, Ritt] ein Beispiel für Netzwerke mit kollektiv reagierenden Neuronen. Auch für spezielle Anwendungen wie die Mustererkennung wurden Netze entworfen. Das 1975 von Fukushima vorgestellte und 1980 zum Neokognitron [Fuku80] erweiterte Kognitron [Fuku75] war bereits ein Mehrschichtennetz.

Erst viele Jahre später entdeckte man, dass Minski und Paperts Aussage für mehrstufige Netzwerke nicht galt. Solche mehrschichtigen Netze konnten alle möglichen Funktionen darstellen. Zwei neue Ansätze riefen ein wieder regeres Interesse an Neuronalen Netzen hervor. Dazu zählen die Arbeiten von Hopfield [Hopf], dessen Beschäftigung mit gewissen Verhaltensweisen von Festkörpern (Spingläsern) einen mathematischen Formalismus lieferte, der letztendlich zu den „Hopfield-Netzen“ führt. Eine noch größere Resonanz fanden die Arbeiten von Hinton, Rumelhart und Williams [Hint] über das Lernverfahren der „Fehlerrückführung“.

Zahlreiche Weiterentwicklungen stellte man in den folgenden Jahren vor. Aus dem Jahr 1985 stammt das Boltzmann-Netz von Ackley, Hinton und Sejnowski [Ackl, Aart]; es basiert auf dem Hopfield-Netz mit Rückkopplung, nur erfolgt die Reproduktions-Phase hier durch ein Verfahren namens „Simuliertes Kühlen“ und die Gewichte werden gelernt. 1987 publizierte B. Kosko den „Bidirektionalen Assoziativspeicher“ in Anlehnung an

Hopfield-Netze [Kosk]. Eine Variation eines vorwärtsgekoppelten Netzes stellt das im gleichen Jahr publizierte „Gegenstrom-Netz“ von R. Hecht-Nielsen [Hech] dar. Es ist zweistufig und enthält in der ersten Stufe eine Kohonen-Schicht. Ist diese berechnet, kann man aus ihr die zweite, sogenannte Grossberg-Schicht ermitteln.

Mittlerweile hat die Neuroinformatik also ihre Anlaufschwierigkeiten (die für neue Wissenschaftsdisziplinen typische Euphorie, gefolgt von starker Ernüchterung) überwunden und sich durch zahlreiche, nützliche Anwendungen als ein Teilgebiet der Künstlichen Intelligenz etabliert. In Zukunft ist mit einer noch stärkeren Integration, auch in andere Aufgabenfelder, zu rechnen. Dabei wird die Weiterentwicklung der Möglichkeiten der Rechentechnik – wie massiv parallele Rechner oder opto-elektronische Systeme – eine große Rolle spielen.

Für das Gebiet der Künstlichen Intelligenz im Ganzen und zukünftige Forschungen – evtl. sogar bis hin zum Künstlichen Leben – fordert Louis Bec [Bec]:

> Mais l’émergence des sciences de l’artificiel, des sciences de la communication, l’explosion des technosciences et des sciences du vivant, ainsi que la transformation sous toutes ses formes, des domaines artistiques, proposent à nouveau un champ fusionel.

Es wird wohl nicht ausbleiben, dass im Zuge der angesprochenen Entwicklung weitere, neue Wissenschaftsfelder gegründet werden, die herkömmliche und neue Technologien vereinen und heute noch nicht abzugrenzen sind.

2 Problemtypen

Gemäß den unter 1.6 angesprochenen Anwendungsgebieten und -aufgaben sowie der Vielzahl von Netzstrukturen ist anzunehmen, dass Neuronale Netzwerke ganz unterschiedliche Problemtypen bearbeiten können. Um eine Verbindung zwischen Netz- und Problemtyp herzustellen, sei es nach Hoffmann [Hoff] nützlich, die verschiedenen Möglichkeiten des Reproduktionsverhaltens eines Netzes zu klassifizieren und jedem Verhaltenstyp einen Problemtyp zuzuordnen. So könne eine Einteilung aussehen wie folgt:

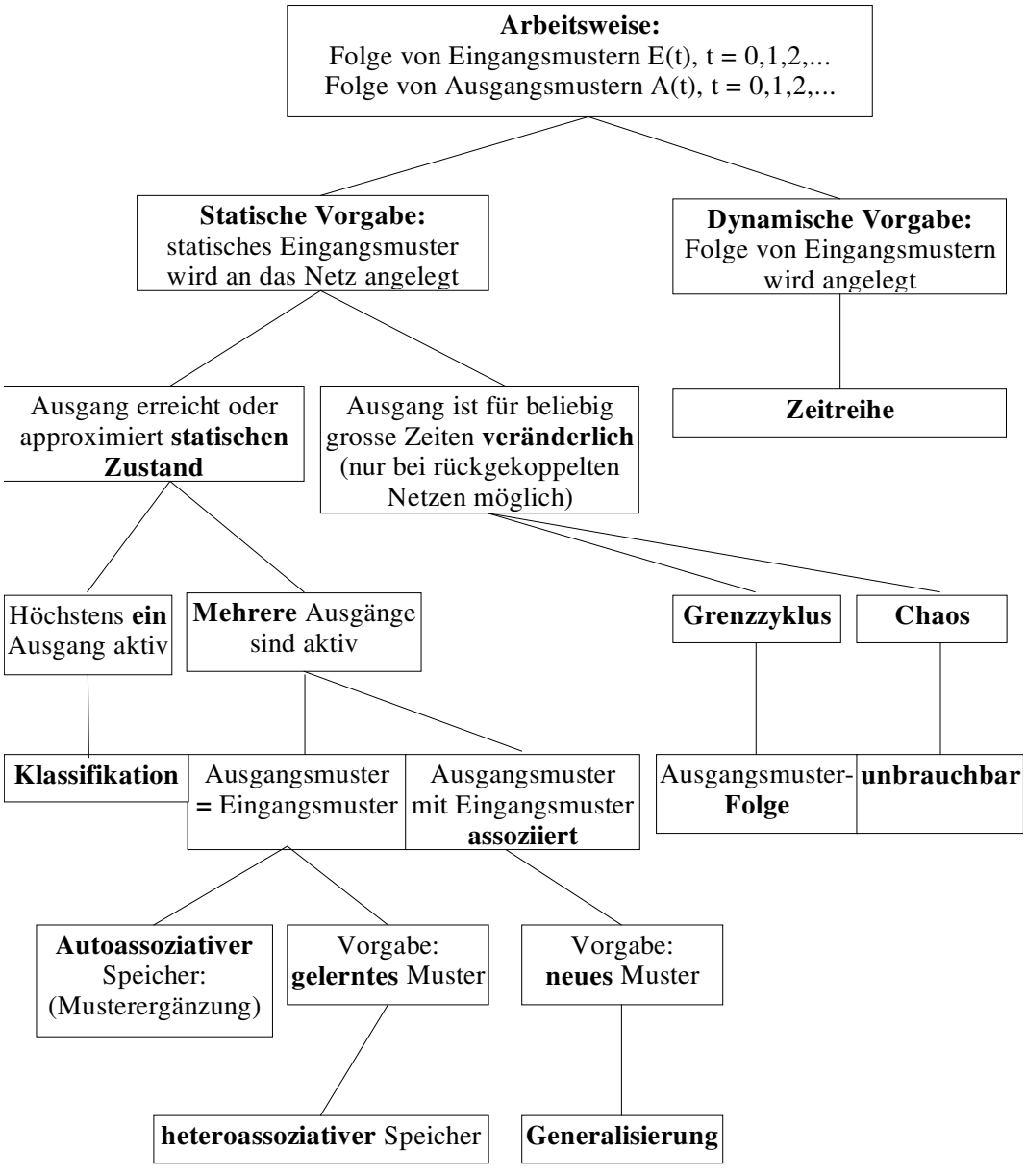

Abbildung 2.1: Arbeitsweisen der Netze – klassifiziert nach ihrem Reproduktionsverhalten

Die dynamische Vorgabe, bei der am Netzeingang eine Zeitreihe (unendliche Folge reeller Zahlen) angelegt wird, stellt die allgemeinste Form der Arbeitsweise Neuronaler Netze dar. Meist präsentiert man dem Netz jedoch unveränderliche Muster (statische Vorgabe).

Natürlich wird mit einer statischen Eingabe ein nach einer endlichen Zahl von Reproduktionsschritten fest definierter oder approximativ erreichter Ausgangszustand erwartet. Allerdings gibt es auch den Fall, dass am Ausgang Musterfolgen oder Zeitsequenzen entstehen. Ebenfalls möglich sind chaotische Folgen, die aber weitgehend unerwünscht sind.

Im Falle eines statischen Ausgangs kann das Netz zur Klassifikation verwendet werden, wenn höchstens je ein Ausgang aktiv ist. Andernfalls unterscheidet man zwischen auto- und heteroassoziativem Verhalten.

Ein autoassoziativer Speicher besitzt die Fähigkeit, gelernte Muster, die an seinen Eingang angelegt werden, zu reproduzieren (ergänzen). Dagegen lassen sich heteroassoziative Speicher charakterisieren als Netze, deren Ein- und Ausgangsmuster miteinander assoziiert sind. Legt man hier ein gelerntes Muster an, so reproduziert das Netz das zugehörige Ausgangsmuster. Beim Anlegen eines neuen, ungelernten Musters liefert das Netz jenes Ausgangsmuster als Ergebnis, dessen Eingangsmuster dem neuen Eingang am ähnlichsten ist.

2.1 Klassische Schulprobleme

Zur besseren Veranschaulichung der Arbeitsweise Neuronaler Netzwerke werden in einführenden Literaturquellen immer wieder die sogenannten „Schulprobleme“ zitiert. So soll auch hier kurz auf sie eingegangen werden.

2.1.1 XOR-Problem

Bei dieser Aufgabenstellung wird die logische XOR-Verknüpfung („Antivalenz“ bzw. „exklusives ODER“) durch ein Netz dargestellt, wobei die Wahrheitswerte „wahr“ und „falsch“ durch „1“ und „0“ repräsentiert werden. Die in Tabelle 2.1 aufgeführten Musterpaare richtig zuzuordnen, ist also Aufgabe der Reproduktionsphase des Netzes.

Eingang 1	Eingang 2	Ausgang
0	0	0
0	1	1
1	0	1
1	1	0

Tabelle 2.1: Logische XOR-Verknüpfung

Typischerweise zeigt man anhand dieses Problems, dass Netze zur Bearbeitung bestimmter Aufgaben eine Minimalstruktur benötigen. Im vorliegenden Fall bräuchte man z. B. entweder ein heteroassoziatives, zweischichtiges Netz (ohne Verteilungsschicht) mit mindestens zwei verborgenen und einem Ausgangsneuron (wobei diese eine nichtlineare Übgertragungsfunktion verwenden) oder ein Netz mit Neuronen höherer Ordnung. Zu beachten ist, dass sich mehrschichtige Netze, deren Neuronen eine rein lineare Ausgangsfunktion besitzen, zu einem einschichtigen Netz mit linearem Übertragungsverhalten zusammenfassen lassen. Durch einschichtige Netze aus Neuronen erster Ordnung ließe sich für das XOR-Problem (Gleichung 2.1) aber keine Lösung erreichen.

$$XOR(x_0, x_1) := x_0\bar{x}_1 \vee \bar{x}_0 x_1$$ **Gleichung 2.1: Logisches XOR-Problem**

Zur besseren Anschauung sei auf Abbildung 2.2 verwiesen. Dargestellt sind die Entscheidungsregionen für das XOR- bzw. – zum Vergleich – das AND-Problem (Abszisse: Wert der Eingangskomponente x_0; Ordinate: Komponente x_1). Die beiden linksseitigen Entscheidungsregionen haben die Form einer Halbebene und werden von einem einschichtigen Netz gebildet. Im Falle des linear separierbaren AND-Problems reicht eine Trenngerade aus, um eine Lösung zu erreichen; die logische Verknüpfung XOR bleibt ungelöst. Rechterhand sieht man die Entscheidungsregionen für das mit einem zweischichtigen Netz gelöste XOR-Problem. Die Lösung kann man sich hier wie folgt vorstellen [Lern]: „Jeder Knoten (Neuron) der verdeckten Schicht bildet eine Trenngerade aus, die im Anschluss im Ausgangsknoten verrechnet wird. Beispielsweise erzeugt ein Knoten der verdeckten Schicht den Ausgang $x_0\bar{x}_1$, der andere den Ausgang $\bar{x}_0 x_1$. Der Ausgangsknoten agiert als OR-Operator.“ Ein dreischichtiges Netz schließlich kann die Entscheidungsregionen beliebig eingrenzen.

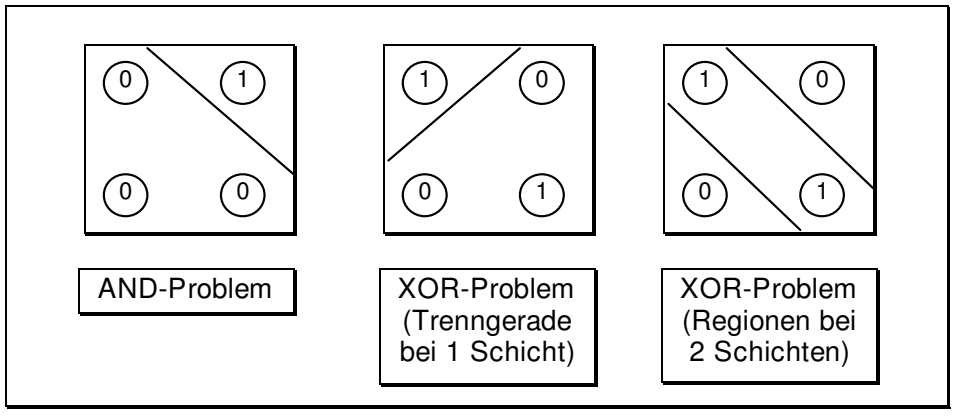

Abbildung 2.2: Entscheidungsregionen für AND- und XOR-Problem

2.1.2 Paritätsproblem

Eine Verallgemeinerung des XOR-Problems stellt das „Paritätsproblem“ dar. Hier soll ein gegebener Vektor aus Nullen und Einsen auf die Anzahl seiner Einsen hin untersucht werden. Ist diese gerade, so nimmt der Netzausgang den Wert null, andernfalls eins an.

Dafür benötigt man ebenfalls ein zweischichtiges, heteroassoziatives Netz. Die Anzahl der verborgenen Neuronen ist gleich der Dimension des Netzeingangs; die Ausgangsschicht besteht aus lediglich einem Neuron.

2.1.3 Encoder-Decoder-Problem

Soll das Netz seinen Eingang unverändert auf den Ausgang durchschalten, spricht man vom „Encoder-Decoder-Problem“. Die verborgene Schicht kodiert die Eingangsvektoren; die Ausgangsschicht dekodiert sie wieder.

Lösen lässt sich dies wiederum mit einem zweischichtigen Netzwerk, wobei die Neuronenzahl der verborgenen Schicht kleiner als die Netzeingangszahl sein muss. Alle Verbindungen sind vorwärtsgerichtet und die Ausgangsschicht ist ausschließlich mit der verborgenen Schicht verbunden.

Genau zuordnen lässt sich dieser Problemtyp weder den autoassoziativen, noch heteroassoziativen Speichern. Da Eingangs- und Ausgangsmuster gleich sein sollen, arbeitet das Netz eigentlich autoassoziativ; doch findet keine Musterergänzung statt. Heteroassoziative Charakterzüge werden ihm verliehen durch die Vorwärtskopplung, Mehrschichtigkeit und durch die meist verwandte Fehlerrückführungs-Lernregel.

2.2 Andere Anwendungen

Beim „Problem des Handlungsreisenden“ (Vertreterproblem) wird versucht, die kürzeste Verbindungslinie zwischen in einer festen Anzahl vorgegebenen Punkten einer Ebene zu finden. Für diese Aufgabe kommen Selbstorganisierende Karten oder auch Hopfield-Netze in Frage.

Muster will man in der Regel nach gewissen Kriterien klassifizieren oder aus verrauschten Fragmenten rekonstruieren. Dazu bedient man sich ganz unterschiedlicher Netze wie des Muster-Assoziators oder ART-Netzes bzw. des Auto-Assoziators oder Hopfield-Netzes.

An Bedeutung gewonnen hat auch der Einsatz Neuronaler Netze zur Sprachanalyse und -synthese. Derzeit konkurrieren sie mit klassischen Methoden wie der „dynamischen Programmierung“ (*Dynamic Time Warping*, DTW) [Itak] oder Klassifizierern auf der Basis von „Hidden-Markov-Modellen“ (HMM) [Krau]. Es hat sich aber schon gezeigt, dass die erstgenannte Gruppe zu langsam und der Speicherbedarf zu groß ist, um die schwierigen Probleme der Spracherkennung zu behandeln. Die Hidden-Markov-Modelle hingegen werden heute in vielen Systemen eingesetzt. Kurzfristig versprechen wohl vor allem Kombinationssysteme (*Hybrid Models*) Erfolg, die die Vorteile der Neuronalen Netze mit den klassischen Verfahren zu kombinieren versuchen [Huan, Sako]. Hier übernehmen die Netze nur einen Teil der Erkennung, meist die Durchführung der Klassifikationsaufgabe. Auch zu Verfahren, die ausschließlich mit Neuronalen Netzen arbeiten, gibt es seit einiger Zeit interessante Fortschritte [Gram]. Obwohl diese Techniken noch relativ jung sind, kann man ihre Ergebnisse bei einfacheren Aufgaben – wie der Einzelworterkennung – mit denen herkömmlicher Techniken vergleichen.

Zum Einsatz in der Spracherkennung kommen am häufigsten die mehrschichtigen Perceptrons. Da diese aber in ihrer Standardform unfähig sind, die Zeit als Parameter einzubeziehen, wurden verschiedene Lösungen vorgeschlagen, um sie zu erweitern. Eine erste sind die Netze aus Neuronen mit Zeitverzögerungs-Gliedern (*Time-Delay-Neural Networks*, TDNN), eine weitere die kontextuellen Netze und eine dritte Lösung bilden die rückgekoppelten Netze, die genauso gut wie klassische stochastische Modelle arbeiten [Guyo]. Boltzmann-Maschinen mit dem Lernverfahren des „Simulierten Kühlens“ wurden ebenfalls untersucht und es konnte festgestellt werden, dass sie leicht bessere Ergebnisse bei der Vokalerkennung zeigen als Perceptrons; jedoch ist ihre Rechendauer inakzeptabel. Sensorische Karten werden inzwischen für vielfältige Aufgaben der Merkmalsquantifizierung eingesetzt [Zell]. Sie lassen sich dazu verwenden, einen Sprecher einer bestimmten Gruppe zuzuordnen oder Phoneme zu erkennen.

3 Zwischenresümee

Die beiden einführenden Kapitel haben u. a. gezeigt, dass Neuronale Netzwerke aus Wissenschaft und Forschung, aber auch der Wirtschaft nicht mehr wegzudenken sind. Sie stellen mittlerweile ein problemlos zu implementierendes Hilfsmittel zur Lösung verschiedenster Aufgaben dar. Obwohl sie nicht ihre Überlegenheit gegenüber klassischen Methoden beweisen konnten, besitzen konnektionistische Modelle doch Charakteristika, die ein Interesse wert sind. Nur einige seien hier nochmals stichpunktartig zusammengefasst:

- Lernfähigkeit
- Rausch-Unempfindlichkeit
- Fähigkeit zur Abstraktion
- Fehlertoleranz bei Netzschäden
- Informations-Verarbeitung parallel
- verständlicher Formalismus

Innerhalb der großen Palette zur Verfügung stehender Netzmodelle bilden die mehrschichtigen Perceptrons mit dem Lernverfahren der Fehlerrückführung nach wie vor den Teil mit der höchsten Anwendungsrate. Sie sind universell einsetzbar, können mit einem sehr verständlichen Formalismus beschrieben werden und scheinen daher prädestiniert für Unterrichtszwecke zu sein.

Die pädagogisch und didaktisch sinnvoll erscheinende Methodik, nach erklärender Einleitung über Sinn, Aufbau und Funktion Neuronaler Netze das gelernte Wissen anhand von Anwendungsbeispielen zu festigen, soll dem Praktikumsversuch zu Grunde liegen. Daher steht als nächste Überlegung die Frage nach geeigneten Problemstellungen.

Von den Arbeitsweisen eines Netzes werden in einer Demonstration aus Praktikabilitätsgründen nur jene in Frage kommen, die einen statischen Eingang erfordern und einen solchen Ausgang erreichen. Dazu zählen nach 2 die Klassifikation, wenn nur ein Ausgang aktiv ist und die Mustererkennung, wenn mehrere Ausgänge aktiv sind. Hier eignen sich besonders logische Zuordnungsaufgaben wie AND oder XOR zur Verdeutlichung der Netzfunktion und verschiedener Abhängigkeiten.

Weiterhin ist es wünschenswert, neben diesen eher theoretischen Aufgabenstellungen auch eine echt praktische Applikation des neu Gelernten zu ermöglichen. Auf Grund der zunehmenden Bedeutung und großen Popularität fällt die Entscheidung hier zu Gunsten einer Simulation zur Spracherkennung.

4 Spracherkennung

4.1 Biologischer Hintergrund

Menschliche Sprache ist nichts anderes als in Schwingungen versetzte Luft, genauer periodische Luftdruckschwankungen, die sich kugelförmig ausbreiten. Der zu deren Erzeugung notwendige Luftstrom wird von der Lunge bereitgestellt und durch die engste Stelle des Sprechapparates (Artikulationstrakt) so beeinflusst, dass unterschiedliche Laute (Phoneme[4]) entstehen. Vokale bilden sich mit Hilfe der Stimmbänder, wobei die Stellung der Zunge und die Öffnung des Mundes als Filter wirken, so dass sich die Vokale anhand der Energiekonzentrationen in einigen wenigen Frequenzbereichen unterscheiden lassen. Konsonanten dagegen engen den Luftkanal so stark ein, dass bei ihnen gar keine oder diffus verteilte Energiekonzentrationen auszumachen sind. Eine Sonderstellung nehmen die Nasale ein: hier kann keine Luft durch den Mund entweichen, es bleibt nur der Ausweg durch die Nase [Schu].

Zum Empfang von Sprache dient das Ohr, welches sich in Außen-, Mittel- und Innenohr gliedert. Schallwellen werden vom äusseren Trommelfell über das Mittelohr zum inneren Trommelfell geleitet. Von größter Bedeutung ist die Schnecke (*Cochlea*) im Innenohr, die ankommende Signale (Schall) in ihre Frequenzbestandteile (bis zu 600 verschiedene) zu zerlegen vermag. Das geschieht durch winzige, feinste Härchen, die von den Schallwellen – je nach Tonhöhe – in Bewegung gesetzt werden und dann über das Gewebe und chemische Reaktionen, die auf die Nervenenden wirken, ein winziges elektrisches Signal zum Hirn leiten [Bone]. Phonetische Experimente haben gezeigt, dass die Cochlea Frequenzen auf nicht-lineare Weise[5] detektiert und dass sie Vorzugs-Bänder (*critical bands*) besitzt, welche die Bedeutung niedriger Frequenzen gegenüber hohen relativ verstärken [Guyo]. Der *Thalamus* (größte graue Kernmasse des Zwischenhirns) als „Umschaltstation für optische und akustische Bahnen“ [Psch] verstärkt die ankommenden Signale, bevor sie dann zur Hirnrinde weitergeleitet und dort ausgewertet werden. Das menschliche Ohr ist in der Lage, Frequenzen

[4] Def.: Kleinste bedeutungsdifferenzierende sprachliche Einheit [Dude].

[5] Mittenfrequenz bis ca. 1000 Hz linear, darüber annähernd logarithmisch verteilt (Mel- bzw. Bark-Skalierung).

zwischen ungefähr 16 und 20000 Hz zu empfangen. Der Bereich menschlicher Sprache (100 Hz bis etwa 7 kHz) ist damit vollständig abgedeckt.

4.2 Systematisierung

Bei der digitalen Verarbeitung akustischer Sprachsignale unterscheidet man mehrere Teilgebiete. Die wohl am häufigsten zur Anwendung kommenden Systeme sind die „Einzelworterkenner“. Sie haben sich in vielen Gebieten, wo bisher nur die Tastatur in Frage kam, einen festen Platz erobert. Bei der Einzelworterkennung müssen die Worte mit deutlichen Pausen zwischen den einzelnen Wörtern – abhängig vom verwendeten System und der Computerleistung – ausgesprochen werden [Bone].

Kontinuierliche Spracheingabe bzw. das Erkennen fliessender Rede ist eines der interessantesten, aber auch am schwierigsten zu realisierenden Probleme. Solche ohne Wortpausen miteinander verbundenen, aufeinanderfolgenden Wörter verbergen ihre Grenzen teilweise oder sogar vollständig.

Das Gebiet der Sprechererkennung könne laut Mak u. a. [Mak] eingeteilt werden in „speaker-verification“ und „speaker-identification“. Das Ziel des ersteren Teilgebietes sei es, „…to verify whether an unknown voice matches the voice of a speaker whose identity is being claimed.“ Bei der Sprecheridentifizierung indes komme es darauf an, eine unbekannte Stimme aus einem Set bekannter Stimmen herauszufinden. Zur besseren Anschauung wird in Abbildung 4.1 ein baumartiger Überblick gegeben.

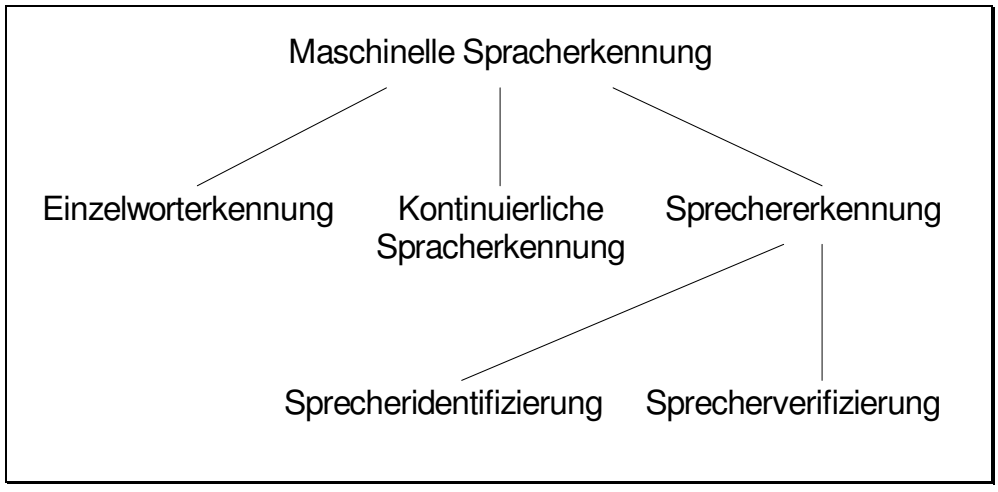

Abbildung 4.1: Teilgebiete der Maschinellen Spracherkennung

4.3 Verarbeitungsstufen

Die unter 4.2 bereits zitierten Mak u. a. [Mak] meinen weiterhin, dass die Datenverarbeitung eines Spracherkennungs-Systems im Wesentlichen in zwei Schritten erfolge: Extraktion der Kennwerte und Klassifikation der Muster. Dem pflichten andere Autoren [Bone, Gram, So, Zhu] bei, wobei hier z. T. noch weitere Unterteilungen vorgenommen werden. Boner [Bone] beispielsweise zählt auch Ein- und Ausgabe der Informationen sowie die Signalvorverarbeitung zum Erkennungsvorgang hinzu. Am sinnvollsten erscheint die Gliederung in [So], nach welcher vier Verarbeitungsstufen bei der Erkennung kontinuierlicher Sprache unterschieden werden, wie im Folgenden schematisch verkürzt zu sehen ist.

4.3.1 Vorverarbeitung

Ähnlich dem Trommelfell im Ohr des Menschen (4.1) empfängt eine Mikrofonmembran Schwingungen, die sie in elektrische Signale umwandelt und so für eine weitere elektronische Verarbeitung zur Verfügung stellt.

Die jetzt einsetzende Vorverarbeitung hat die Aufgabe, das Sprachsignal so aufzubereiten, dass die für den Erkennungsprozess relevanten Informationen leicht und sicher gewonnen werden können. Doch stellt sich zunächst die Frage, welches die optimalsten Merkmale (Informationen) für die Spracherkennung sein werden. Vorläufig besteht unter den Spracherkennungsforschern noch keine Einigkeit darüber. Die meisten Systeme stützen sich auf eine zeitliche Analyse des Sprachsignals und auf das daraus ableitbare Frequenzverhalten [Bone].

Mit diesem Ziel vor Augen wird das Signal vom Analog-Digital-Wandler der Soundkarte zunächst digitalisiert, d. h. in Abschnitten von 20 bis 40 Millisekunden (so lang ist ein Sprachsignal stationär, ändert sich also nicht) abgetastet. Dann müssen störende Signalanteile entfernt und signaltypische herausgehoben werden.

Vorhandene Gleichanteile beispielsweise können durch das Zentrieren des Signals (Subtrahieren des Mittelwertes von jedem Signalwert) herausgerechnet werden. Zur Kompensation unterschiedlicher Aufnahmelautstärken wurden in [Krau] alle Werte auf maximale Lautstärke normiert. Im nächsten Schritt erfolgt eine Fensterung mit anschließender

Fourier-Transformation. Aus jedem der Zeitfenster wird dabei ein Leistungsspektrum mit mehreren Koeffizienten berechnet.

Um die nichtlineare Transformation (siehe 4.1) von Frequenz zu psychoakustischer Tonheit (gemessen in Bark) zu berücksichtigen und um die Information pro Zeitfenster zu reduzieren, wird das Frequenzspektrum in ein Barkspektrum mit einer kleineren Anzahl von *K* Kanälen umgewandelt [Gram]. Die Transformation von Frequenz *f* nach Tonheit *z* gleicht einer Frequenzverzerrung und findet statt nach Gleichung 4.1 [Volk]. Die Bark-Frequenzen verkörpern dabei die Mittenfrequenz der gebildeten Frequenzbänder.

$$z = \frac{26.81 f}{1960 Hz + f} - 0.53$$ **Gleichung 4.1: Berechnung der Bark-Frequenzen**

Jeder der *K* Leistungswerte $\bar{b}_i$ eines Barkspektrums wird anschließend in eine Lautheit umgewandelt (Gleichung 4.2). Dieser Schritt dient der Nachbildung der Vorzugs-Frequenz-Bänder des Innenohrs (Kapitel 4.1).

$$b_i = \bar{b}_i^{\,p} \qquad (i = 1...K;\ 0 < p < 1)$$ **Gleichung 4.2: Transformation Leistung → Lautheit**

Als weiterhin mögliche Stufe der Vorverarbeitung beschreibt [Gram] ein Modell, das er „Kontrastieren der Spektrogramme“ nennt. Es könne die dynamische Anpassung der Lautheit im menschlichen Gehör sehr gut nachbilden und trüge zur deutlichen Erhöhung der Erkennungsraten bei.

4.3.2 Merkmalsextraktion/-reduzierung

Während über den Ablauf der Vorverarbeitung weitgehend Einigkeit herrscht, existieren für die Merkmalsextraktion annähernd so viele Lösungsansätze wie sich Autoren zu diesem Thema geäußert haben. Eine Kombination statischer (*Mel-Frequency scaled Cepstral Coefficients*, MFCC) mit dynamischen (*Dynamic MFCC*) Merkmalen schlägt Zhu [Zhu] vor. Die Lokalisierung von Silbenkernen mit anschließender Trennung von Halbsilben durch Hidden-Markov-Modelle steht im Vordergrund des Verfahrens in [Rusk]. In [Gram] wurde ein eigenes Neuronales Netz (*Feature Finding Neural Network*) konzipiert, das u. a. die bei Säugetieren gefundenen Merkmalszellen durch Sigma-Pi-Neuronen[6] modelliert und so relativ

[6] Typ eines künstlichen Neurons, dessen Eingangsfunktion nicht als gewichtete Summe der Eingangswerte, sondern auf komplizierte Weise berechnet wird (Neuron höherer Ordnung).

invariante Merkmale extrahiert. Bei den meisten Verfahren werden typische akustische Merkmale zusammengefasst und so auf Phoneme, die Grundlaute einer Sprache (im Deutschen 40 bis 50), reduziert. Es entsteht für jedes diktierte Wort eine Lautfolge.

Erwähnenswert ist auch ein in [Krau] beschriebenes Modell aus Kohonen-Netz zur Merkmals-Quantifizierung und Fehlerrückführungs-Netz zur Klassifizierung, das allerdings nur als Einzelworterkenner fungiert. Ebenda wird (mit Verweis auf [Kell]) ein Verfahren zur Wortgrenzendetektion störungsfreier Signale erläutert (Abbildung 4.2). Dazu berechne man die Gesamtleistung für jeden gebildeten *Frame* (Zeitfenster) eines Signals über alle Kanäle. Beim Analysieren der zeitlichen Leistungsänderung gelten dann alle Werte über einer vorher festgelegten Schwelle S als Sprache. Ein Zähler Z wird inkrementiert, wenn ein Frame als Sprache erkannt wird. Liegt der Zählerwert über einer Toleranzschwelle W_a, so gilt der Wortanfang als erkannt. W_a verhindert, dass kurzzeitige Störungen als Wortanfang detektiert werden. Eine weitere Toleranzschwelle W_i ($W_i > W_a$) bewirkt, dass etwaige Innerwortpausen nicht sofort als Wortende erkannt werden. Der Zähler Z erhöht sich schrittweise bis maximal zur Schwelle W_i, nachdem ein Wortanfang detektiert wurde. Während des Wortes bewegt sich Z zwischen W_i und W_a. Erst, wenn der Zähler wieder null erreicht hat, gilt das Wortende als erkannt.

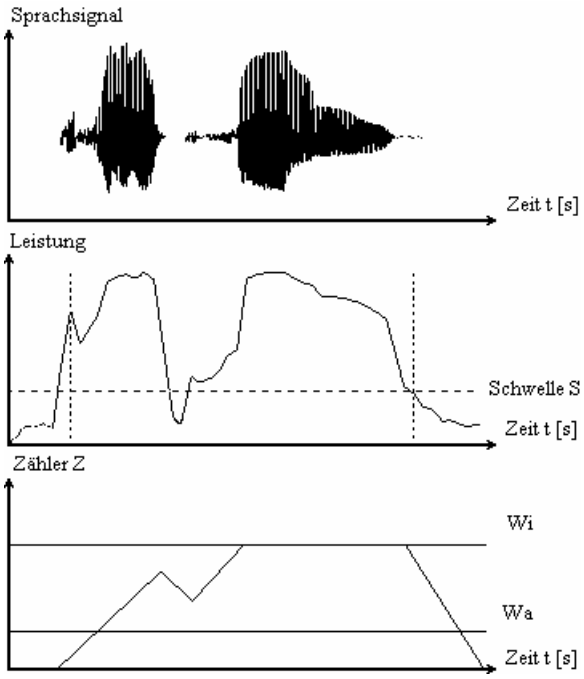

Abbildung 4.2: Wortgrenzendetektion

Die Suche nach idealen Merkmalen, die ein Wort relativ invariant beschreiben, wird auch in Zukunft weitergehen. Hier hat vor allem die Biologie ihren Beitrag zu leisten, denn im

Gegensatz zum Sehsystem ist das Hörsystem noch weitgehend unerforscht. So ist leider bis heute nicht bekannt, welche Merkmale der Mensch aus dem Sprachsignal extrahiert, um sein internes Bild von Wörtern und Bedeutungen aufzubauen.

4.3.3 Klassifikation/Mustervergleich

Die bestimmten Merkmale werden im Lexikon, der Datenbank des Spracherkennungs-Systems, mit den dort gespeicherten Frequenzmustern verglichen. Das gesprochene Wort, dessen Muster der Phonem-, Silben- oder anderen Folge des gespeicherten Wortes am nächsten kommt, wird vorgemerkt.

Es ist zu unterscheiden zwischen sprecherabhängigen und -unabhängigen Systemen. Um letztere zu realisieren, gibt es im Wesentlichen zwei Möglichkeiten: einen sprecherunabhängigen und einen sprecheradaptierenden Ansatz. Weiter heißt es in [Zhu]:

> Der erste Ansatz hat den Vorteil, dass das Erkennungssystem durch Generalisierung einer großen Sprachdatenbank realisiert wird, so dass die statistischen Eigenschaften der zu erkennenden Sprache ausgenutzt werden können … Allerdings hat diese Methode den Nachteil, dass das Erkennungssystem wegen der sich phonetisch überlappenden Sprachsegmente unterschiedlicher Sprecher erhöhte Fehlerraten aufweist. Das sprecheradaptierende Verfahren kann dagegen die Leistung eines Erkennungssystems durch beaufsichtigte oder unbeaufsichtigte Trainingsalgorithmen verbessern.

4.3.4 Kontextprüfung

Bevor das als wahrscheinlich erkannte Wort übernommen wird, erfolgt meist eine Kontextprüfung, etwa mit Triagrammstatistiken, wo verzeichnet ist, mit welchen zwei anderen Wörtern ein Wort häufig kombiniert wird. Das als wahrscheinlich erkannte Wort wird dann in die Textdatei übernommen.

4.4 Probleme

Wie schwer es für ein Computersystem ist, gesprochene Wörter zu erkennen und in geschriebene umzusetzen, wird durch folgende Probleme offensichtlich [So]:

- Die akustischen Klangvariationen sind groß – zwischen Sprechern (Dialekt, Sprachfehler) und auch je nach physischer (Erkältung!) und psychischer Verfassung des Sprechers.
- Semantik, also Inhalt und Bedeutung von Zeichen („Der gefangene Floh“ oder „Der Gefangene floh“).
- Wortschatz: Jede Wortform (Flexion) erfordert einen Eintrag im Lexikon, also in der Datenbank des Spracherkennungs-Programmes. Während es beispielsweise im Englischen durchschnittlich 2,2 Flexionen je Grundform gibt, sind es im Deutschen 5.

- Homophone, Wörter, die gleich ausgesprochen, aber verschieden geschrieben werden (mehr/Meer, Lärche/Lerche, viel/fiel, wieder/wider).
- Komposita, zusammengesetzte Worte (Steuerlüge).

4.5 Aktuelle Systeme

Zur Untermauerung der Aktualität der maschinellen Spracherkennung konnten drei repräsentative Beispiele recherchiert werden. Diese nicht ausschließlich mit Neuronalen Netzen realisierten Systeme zur Sprachsteuerung haben auf dem Markt mittlerweile einige Bedeutung erlangt und erhalten von [Spra] folgende Einschätzungen.

Von Dragon-Systems kommt die Software „Dragon-Dictate“ in Version 2.2 für Windows-Rechner: Bereits nach einer kurzen Trainingszeit (man muss dem Rechner bestimmte Vokabeln vorsprechen, damit er sie versteht) sei die Software funktionstüchtig. Die Erkennungsrate des Systems sei verhältnismäßig schnell und man könne auch recht flüssig diktieren.

Vom Soundkartenhersteller Terratec kommt das „Smart-Office“, das sich aus Soundkarte, hochwertigem Headset (Mikrofon plus Kopfhörer) und der Software „Dragon Dictate“ in der Version 1.4 (gegenüber der Vollversion von Dragon-Dictate etwas reduziert) zusammensetzt: Nach einiger Zeit könne man mit dieser Ausrüstung wirklich schnell den Rechner steuern. Dank der Qualität der Soundkarte (die Qualität der Digitalisierung ist ein nicht zu unterschätzender Faktor) und des Headsets laufe die Spracherkennung mit einiger Übung richtig flüssig.

Innerhalb der neuen Version des Betriebssystems OS/2 Warp (4.0 – Merlin) von IBM wird das Sprach-Diktierprogramm „Voice Type 3.0“ von IBM bereits mitgeliefert: Auch hier sei festzustellen, dass nur Training zum Erfolg führe. Mit Voice Type könne man richtig schnell die Aktionen des Rechners steuern und auch Text diktieren. Von dem Programm existiere mittlerweile auch eine Light-Version, genannt „Simply Speaking“.

Die Tatsache, dass diese Systeme heute auf dem Markt sind, stellt einen bedeutenden Fortschritt für die Anwendung Neuronaler Netze dar. Noch vor fünf Jahren waren die Methoden zur akustischen Kommunikation für den praktischen Einsatz nicht tauglich. Bis dahin gab es lediglich einige erste Versuche wie das „TRACE-Netz“ von McClelland zum Erkennen gesprochener Wörter als Spracherkennungs-Programm oder das Fehlerrückführungs-Netz „NETtalk“ von Brause u. a. als Sprachsynthese-Programm zur Umwandlung von ASCII-Text in Phoneme.

5 Praktische Umsetzung

5.1 Revision

Es wurden in obigen Kapiteln die Grundlagen Neuronaler Netze erarbeitet und im Hinblick auf den Einsatz in einem Praktikumsversuch ebenfalls mögliche Problemstellungen. Außerdem wurden für die optionale Erweiterung des Laborversuches um einen echt praktischen Teil die Grundlagen der Spracherkennung in Computern beschrieben.

Da die Praktikanten – geht man nach dem aktuellen Lehrplan – zum ersten Mal mit dem Aufgabengebiet „Neuronale Netze“ konfrontiert werden, wird es zunächst darauf ankommen, grundlegende Begriffe des umfangreichen, neuen Vokabulars einzuführen und Funktionsweisen verständlich zu machen. Das könnte im Rahmen der Grundlagenbeschreibung und der Vorbereitungsaufgaben geschehen, wobei der Schwerpunkt auf das Fehlerrückführungs-Netz zu setzen wäre, wie in Kapitel 3 festgestellt wurde.

Dann wird eine erste Aufgabe gestellt werden können, mit Hilfe derer der grundsätzliche Ablauf zur Erstellung des Netzes, eines passenden Muster-Sets und schließlich der Anlernvorgang veranschaulicht wird. Als relativ einfache und außerdem häufig zur Einführung zitierte Problemstellung böte sich dafür beispielsweise die logische Aussagefunktion AND an.

Sind die Verfahrensweisen dann bekannt, wird ein näheres Eingehen auf einzelne Parameter möglich sein. So könnte die unterschiedliche Arbeitsweise verschiedener Neurontypen, d. h. im Besonderen die Wirkung ihrer Ausgangsfunktion anhand einfacher Probleme betrachtet werden. Weiterhin erlaubte die Variation einzelner Parameterwerte wie Lernrate, Momentum u. a., deren Auswirkungen auf den Lernprozess zu untersuchen. Wichtig wäre die Demonstration der Entscheidungsfindung eines Netzes und der Forderung nach einer Mindestkonfiguration, wofür das XOR-Problem geeignet erscheint.

Eine Möglichkeit zur Auswertung wichtiger Parameter wird geschaffen werden müssen. Als – neben dem Anlernen – zweite mögliche und für Anwendungen eigentlich interessante Arbeitsphase eines Neuronalen Netzes wird die Reproduktion nicht fehlen dürfen. Hier könnte einfacherweise ein vorher gelerntes Muster reproduziert und mit den gewünschten Zielvektor-Werten verglichen werden, um die Eignung des Netzes für dieses

spezielle Problem festzustellen. Die Darstellung der Arbeitsweise eines Optimierungsalgorithmus' wird wünschenswert sein.

Zur optionalen Erweiterung des Laborversuches wird ein mögliches Verfahren zur Spracherkennung in Frage kommen. Wie unter 4 gesehen werden konnte, könnte es sich dabei, um im Rahmen des Machbaren zu bleiben, lediglich um eine Demonstration zur Erkennung einzelner, gesprochener Worte handeln, die offline arbeitet.

Eine während des Versuches schnell und einfach zu handhabende Möglichkeit zur Hilfestellung wird gegeben werden müssen.

5.2 Hard- oder Software

Zunächst stellte sich jetzt die Frage, wie die in Kapitel 1.3 und 1.4 beschriebenen Modelle biologischer Nervenstrukturen technisch zu realisieren sind. Dazu gibt es heute (sieht man einmal von den noch in der Entwicklung steckenden Verfahren zur Produktion natürlicher Netze ab) zwei Möglichkeiten: Hard- oder Software.

Den Laborversuchen gemein ist die Anwendung theoretisch erworbenen Wissens in der Praxis. Nun lässt sich über den praktischen Gehalt einer computergestützten Simulation streiten. Tatsache jedoch ist, das Neuronale Netze in Industrie und Wirtschaft derzeit überwiegend auf Computern simuliert werden. „Das hat den Nachteil", schreibt Hoffmann [Hoff] dazu, „dass die einzelnen Neuronen der Reihe nach abgearbeitet werden müssen. Im Prinzip gilt das auch bei Multiprozessorsystemen; da ein einzelnes Neuron sehr einfach ist, wäre es Verschwendung, für jedes Neuron einen eigenen Prozessor vorzusehen." Doch erweist sich die große Flexibilität von Computern als Vorteil, besonders wenn verschiedene Netzmodelle erprobt werden sollen. Die Herstellung neuer Hardware für jede Aufgabe in einem Praktikumsversuch wäre schlicht unangemessen. Daher basiert die Entwicklung des Versuches auf einem Computerprogramm.

5.3 Komponentenentwicklung

Ausgangspunkt waren hier Überlegungen zur Struktur des zu erstellenden Programmes, dessen interner Realisierung, optischer Aufmachung nach außen und zur Beachtung gängiger Normen bei der Programmierung, wie z. B.: die klare Gliederung einzelner Programmteile, um die Wiederverwendbarkeit von Programmcode zu ermöglichen;

ausreichende Erläuterungen und Quelltext in einheitlicher Sprache, um die Übersichtlichkeit zu erhöhen; Definition fester Schnittstellen zwischen einzelnen Programmteilen.

Für die Realisierung dieser Forderungen schien Borland's Entwicklungssystem „Delphi" in der Version 2.0 für 32-Bit-Windows am besten geeignet. Es verschmilzt den visuellen Entwurf von Fenstern mit dem Entwickeln des Programmcodes, stellt dem Benutzer ein völlig ereignisgesteuertes Programmiermodell zur Verfügung und erlaubt die Entwicklung eigener Komponenten, die nahtlos in die Klassenhierarchie der Visuellen Komponentenbibliothek eingebunden werden [Wark2].

Dieser letzte Punkt war ausschlaggebend für die Entscheidung, die interne Funktionalität des Neuronalen Netzes als Delphi-Komponente zu realisieren. Es wurden zunächst alle notwendigen Parameter gesammelt und geklärt, an welcher Stelle die Komponente in die Klassenhierarchie einzubinden ist.

Von TObject, der Wurzel des Hierarchiebaumes, wurde eine neue Klasse TNeuron abgeleitet. Sie kapselt alle, für das Neuron-Modell typischen Eigenschaften und Methoden und stellt so eine allgemeine Basisklasse dar, die in verschiedenen Netzkomponenten Verwendung finden kann. Erwähnt werden muss, dass durch sie lediglich Neuronen erster Ordnung dargestellt werden, was aber in den meisten Simulationen der Fall ist. „Durch die Verwendung von Neuronen höherer Ordnung[7] kommt man in einem Netz zwar mit weniger Neuronen aus", wie Hoffmann [Hoff] meint, doch sei das bei computersimulierten Netzen nicht unbedingt von Vorteil, da man wesentlich mehr Gewichte speichern müsse und außerdem komplizierte Berechnungen auszuführen habe.

TLayer kann mehrere Instanzen der Klasse TNeuron in einer Liste speichern; sie stellt einen Nachfolger von TPersistent dar. Diese Wahl wurde getroffen, um mit Hilfe der TStream-Klassen eine polymorphe Speicherung ihrer Objektinstanzen zu ermöglichen, die so bei jedem erneuten Programmlauf wieder eingelesen werden können.

Schließlich wurde TBackPropNet definiert als Nachfolger von TComponent und verkörpert somit die eigentliche, eine nicht-visuelle Komponente, d.h. sie wird zur Entwurfszeit nur als Icon dargestellt und kann lediglich verschoben, nicht aber anderweitig manipuliert werden. So entfällt aber jeglicher Ballast, den eine visuelle, von TWinControl abgeleitete Komponente benötigt, um echte Windows-Fenster zu verwalten. Die Oberfläche des Programmes soll ja sowieso unabhängig von der Komponente programmiert werden und

[7] Auch „Sigma-Pi-Neuronen" genannt.

nur über einige Schnittstellen erreichbar sein. In einer Liste wiederum kann TBackPropNet mehrere Instanzen der Klasse TLayer speichern.

Eine weitere Klasse zur unabhängigen Verwaltung der nicht zum eigentlichen Netz gehörenden Muster- und Reproduktionsvektoren, TSet, ist als Nachfahre von TList definiert worden. Gespeichert werden können in ihr mehrere Instanzen der ebenfalls neu erstellten Klasse TPattern, die Nachfahre von TObject ist.

5.4 Simulator „Res Neuronum“

Im Vordergrund zunächst stand bei der Oberflächen-Entwicklung wieder das Bestreben, alle Programmteile so zu realisieren, dass sie eine möglichst hohe Eigenständigkeit aufweisen. Daher wurden größere Programm-Module als Dynamische Link-Bibliotheken angelegt.

DLLs (Dynamic Link Libraries) sind eines der wichtigsten Grundelemente von PC-Betriebssystemen wie Windows oder OS/2. Sie stellen – ähnlich wie EXE-Dateien – fertig übersetzte Module dar, die jedoch nicht eigenständig ausgeführt werden, sondern anderen Modulen (EXE-Dateien und DLLs) ihre Dienste in Form von Funktionsbibiliotheken anbieten [Wark2]. Im vorliegenden Simulationsprogramm werden sie bei Bedarf zur Laufzeit (dynamisch, also explizit) zum Programm hinzugebunden, wodurch zum einen eine immense Platzersparnis im Arbeitsspeicher möglich ist (im Gegensatz zu den implizit oder statisch gebundenen DLLs). Zum anderen können die vorhandenen DLLs in andere Programmierumgebungen wie C/C++ eingebunden werden; auch der umgekehrte Weg ist natürlich möglich.

Die Namen von Dateien beginnen grundsätzlich mit einem Kürzel des Projektes, zu dem sie gehören. Darauf folgt eine nähere Bezeichnung ihrer Funktion. Wenn in der Datei ein Formular definiert ist, folgt als nächstes das Wort „Form“. Am Namensende steht „File“. Die Bezeichnungen der zugehörigen Units sind identisch. Streicht man das Wort „File“, so bleibt der Formular-Name übrig. Dazu ein Beispiel: Die Datei „BackPropEvalDiagFormFile.pas“ beinhaltet die gleichnamige Unit, in welcher wiederum das Formular „BackPropEvalDiagForm“ definiert ist. Die ersten zwei Teilwörter „BackProp“ lassen erkennen, dass die Unit zum Projekt „BackPropagation.dpr“ gehört. Ihre Funktion wird durch „EvalDiag“ näher erläutert – im vorliegenden Fall also ein Diagramm-Formular, das von der Seite „*Evaluation*“ (Auswerten) des BackProp-Formulars aufgerufen wird.

Alle verwendeten Variablen wurden an Klassen gebunden, um „wilde“ Definitionen innerhalb der Units zu vermeiden. Es wurden fast ausschließlich generische Typen (Integer; Character; String) verwandt, um bestmögliche Ergebnisse mit dem zu Grunde liegenden Prozessor (CPU) und Betriebssystem zu erzielen. Die in ihrem Bereich und Format vom Betriebssystem unabhängigen Fundamental-Typen wurden lediglich für einige Dateioperationen benutzt, damit die Speicherformate auch auf zukünftigen Systemen mit der Object-Pascal-Anwendung in Einklang stehen. Die alphabetische Anordnung der Funktionen und Prozeduren erleichtert das Auffinden gesuchter Stellen. Neben der Programmiersprache sind auch sämtliche Variablen- und Methodennamen sowie die Erläuterungen in englischer Sprache gehalten. Lediglich die auf der Oberfläche erscheinenden Begriffe wurden in Deutsch eingegeben.

5.4.1 Hauptprogramm

Das Hauptformular „BasicForm“ verwaltet u. a. die unter 5.4 beschriebenen DLLs, zu denen die unterschiedlichen Netzwerktypen[8] zählen. Seine Erweiterung ist jederzeit problemlos möglich. Es besteht, um sinnvolle Aufgaben mittels der eingebundenen Netze lösen zu können, zum Beispiel die Möglichkeit, verschiedene Anwendungen ebenfalls modular in das Programm aufzunehmen.

Ein hier zu nennender Nachteil der Verwendung von DLLs ist, dass die durch sie gekapselten Formulare nur modal ausgeführt werden können. Dadurch bleibt die Möglichkeit verwehrt, mehrere Instanzen eines Netzes oder einer Anwendung innerhalb des Programmes parallel laufen zu lassen und zwischen ihnen beliebig zu wechseln. Um das zu erreichen, wäre eine Umstellung der Eigenschaft „FormStyle“ in Hauptformular und Modulen auf „fsMDIForm“ bzw. „fsMDIChild“ vonnöten. Doch würde das schon beim Formular „Fehlerrückführung“ (siehe Kapitel 5.4.2), welches eigene Unterfenster verwaltet, nicht funktionieren. Außerdem müsste das Hauptformular beträchtlich vergrößert werden, da MDI-Kind-Formulare nur innerhalb ihres Eltern-Fomulars geöffnet werden können. Schließlich gingen alle durch die Verwendung von DLLs gewonnenen Vorteile verloren. Daher schied diese Form des Programmaufbaus aus.

Das Erscheinungsbild des Hauptprogrammes ist im Windows-Stil gehalten; dazu gehören Hauptmenü, Status- und Hinweisfenster sowie die SpeedButtons; sie gelten als

[8] In der Version 1.2 des Simulationsprogrammes lediglich Modul „Fehlerrückführung“.

hilfreich und nutzerfreundlich, da sie im Besonderen ein schnelleres Arbeiten ermöglichen. Das Formular ist kaum größer als eine Statusleiste, wodurch eine gute Übersichtlichkeit auf dem Bildschirm gewahrt bleibt.

Die Definition des Hauptformulars erfolgt in der Unit „ResNeuroBasicFormFile“, welche in gleichnamiger Datei abgelegt ist. Weiterhin zum Hauptprogramm gehören die Dateien mit folgenden Units: „ResNeuroLogoFormFile“, „ResNeuroSettingsFormFile“ und „ResNeuroAboutFormFile“. Sie enthalten die Formulare mit dem Titel-Logo, den Einstellungs-Optionen und dem Info-Dialog. Die Bedienung des Simulators ist im Handbuch (Anhang B) näher beschrieben.

5.4.2 Modul Fehlerrückführung

In Anlehnung an die unter 1.4 - 1.6 erarbeiteten Punkte wurde das Formular durch die Komponente BackPropPageControl gegliedert in sechs Seiten: *Configuration*, *PatternSet*, *Learn*, *Evaluation*, *RecallSet* und *Pruning*. Diese Trennung soll dem Nutzer die einzelnen Schritte bei der Arbeit mit Neuronalen Netzen stets klar vor Augen halten.

Auch bei der weiteren Gestaltung des *Graphical User Interface* (GUI) wurde versucht, alle visuellen Komponenten auf dem Formular klar und übersichtlich anzuordnen und Abläufe logisch zu gestalten, um die vielzitierte „intuitive Bedienbarkeit“ zu ermöglichen. Bei Abwahl einer bestimmten Option beispielsweise werden alle mit ihr in Verbindung stehenden Felder automatisch deaktiviert. So werden Zusammenhänge noch besser dargestellt und Fehleingaben vermieden. Zum besseren Nachvollziehen des Lernablaufs und für die Auswertung der Ergebnisse wurden mit Hilfe von PaintBoxes mehrere graphische Möglichkeiten geschaffen.

Die Programmierung des Zusammenspiels der einzelnen Komponenten gestaltete sich schwieriger als gedacht. Häufig müssen nach der Änderung eines Parameters auch alle anderen, von ihm abhängigen, aktualisiert werden. Die dafür aufgerufenen Prozeduren sind meist Reaktionen auf „OnChange“- oder „OnExit“-Ereignisse. Wird nun ein anderer Wert neu gesetzt, tritt für seine Komponente ebenfalls wieder ein Ereignis „OnChange“ ein und kann eine Methode aufrufen. Die so entstehende Kettenreaktion darf den Wert der ursprünglich geänderten Komponente nicht nochmals erreichen und ändern, sonst sind Endlosschleife und damit Fehler vorprogrammiert.

Zum Formular „BackPropForm“ gehören vier weitere, modale Dialog-Formulare zur Einstellung von Parametern für die Algorithmen der Datenauswertung. Außerdem wird mit dem Öffnen von „BackPropForm“ ein unsichtbares Formular erstellt, das die Diagramme zur Datenauswertung enthält. Jedes dieser fünf Formulare kann von der Seite „Auswerten“ aus aufgerufen werden.

Bei der Programmierung des Strukturoptimierungsverfahrens (Seite „Pruning“) wurden die Empfehlungen früherer Arbeiten [Trau, Hell] der TU Ilmenau zu diesem Thema beachtet und eine Algorithmus-Erweiterung durchgeführt. So basiert er jetzt auf Listen als dynamischen Speicherstrukturen, statt, wie früher, auf statischen Arrays. Dadurch können Netze beliebiger Größe (im gültigen Bereich) optimiert werden. Außerdem ist es nun möglich, die Optimierung nicht nur auf die Neuronen, sondern auch die Schichten anzuwenden.

Eine Aufsplittung der Formular-Unit „BackPropFormFile“ in Unter-Units wurde trotz ihres relativ großen Umfangs vermieden, um das Nachvollziehen des Codes durch die Trennung der Formular-Klasse nicht unnötig zu erschweren. Helfen soll hier die Bezeichnungsweise der Variablen und Methoden, die stets mit dem Namen der PageControl-Seite, zu der sie gehören, beginnt.

5.4.3 Modul Spracherkennung

Die als zusätzliche Erweiterung des Praktikumsversuchs gedachte Applikation ist in der Lage, nach entsprechender Belehrung des enthaltenen Netzes mit einem begrenzten Wortschatz, einzeln gesprochene Wörter wiederzuerkennen. Zur Arbeit mit dem Modul „SpeechRecognition.dll“ ist neben einer Soundkarte, die in der Lage ist, Dateien mit 11025 Hz, 16 Bit aufzunehmen/wiederzugeben (z. B. SB 16), auch ein Mikrofon erforderlich.

```
typedef struct{
        FOURCC      ckID;
        DWORD       ckSize;
        BYTE        ckData[ckSize];
} CK
```

Abbildung 5.1: Aufbau eines Chunks

Für die Erstellung der Sprach-Samples bot sich – als gängigste Form zur Speicherung von Audio-Daten unter Windows – das WAV-Format an. Es unterliegt dem RIFF-Standard,

der von Microsoft für die meisten unter Windows benutzten Datei-Typen als Grundstruktur definiert wurde. In [Stol] ist zu Abbildung 5.1 erklärend zu lesen:

> Eine RIFF-Datei baut sich aus mehreren kleinen Daten-Einheiten auf, den sogenannten Chunks. Ein Chunk … hat immer denselben Aufbau, egal, was für Daten darin stehen. … Die Typdefinition FOURCC steht für 'Four Character Code'. Sie zeigt an, dass ein Chunk [CK] immer mit einer Sequenz von vier ASCII-Zeichen beginnt, um den Chunk identifizieren zu können [Struktur-Element ckID]. … Das Element ckSize beinhaltet die Größe des Chunks in Form einer positiven 32-Bit-Zahl. Dafür wurde der Typ DWORD, also Double-Word definiert. Die enthaltene Größe beinhaltet jedoch nicht die vier Bytes von ckID und ckSize. Hier steht also nur die Anzahl von Bytes, die nach ckSize beginnen. Die Daten des Chunks stehen dann in ckData.

WAV-Dateien setzen sich also aus mehreren Chunks zusammen. Ihr schematischer Aufbau ist in Abbildung 5.2, welche aus [Ryan] bzw. [Stol] entnommen ist, dargestellt. Im besprochenen Modul werden Samples im PCM-Format (*Pulse Code Modulation*) mit einer Auflösung von 16 Bit, einem Kanal (mono) und einer Samplingrate von 11025 Hz verarbeitet. Diese Abtastfrequenz gestattet (bei Vermeidung von Überlagerungseffekten) eine maximal mögliche Signalfrequenz von 5,5 kHz. Minimal auflösbar sind mit 256 Stützstellen (siehe weiter unten) Signalfrequenzen von 43 Hz, womit der Bereich normal gesprochener, menschlicher Sprache abgedeckt ist.

```
<WAVE-form>   -> RIFF (      'WAVE'                          // Form-Typ
                             <fmt-ck>                        // Waveform Format
                             <data-ck>                       // Waveform Daten

<fmt-ck>      -> fmt (       <wave-format>                   // Datenstruktur WaveFormat
                             <format-specific> )             // abhängig von Datenformat

<wave-format> -> struct {
                             WORD      formatTag;            // Format-Kategorie
                             WORD      nChannels;            // Anzahl der Kanäle
                             DWORD     nSamplesPerSec;       // Sample-Rate
                             DWORD     nAvgBytesPerSec;      // Puffer-Info
                             WORD      nBlockAlign;          // Block Alignment
                             }
```

Abbildung 5.2: Beschreibung einer WAVE-Datei

Zur Verwaltung eines geladenen Samples wurde die Klasse „TWaveFile" definiert. In ihr wird die Datei-Struktur abgelegt, der Name der aktuellen Datei festgehalten, die Anzahl der Abtastwerte angegeben u. a. m. Als wichtige Methoden definiert diese Klasse die Prozeduren zur Signalverarbeitung. Auf sie wird im Anschluss noch näher eingegangen. Nicht minder wichtig sind die Methoden zur Durchführung der notwendigen Dateioperationen. Sie

deklarieren objektorientierte Streams, mit deren Hilfe die Daten geladen bzw. gesichert werden.

Wie in den unter 4 besprochenen Grundlagen gesehen werden konnte, müssen die vorhandenen Daten der Sprachsamples zunächst präpariert werden für die weitere Verarbeitung durch das Netzwerk. In Zimmermann [Zimm] steht dazu zu lesen:

> Der Aufwand, der in eine sorgfältige Vorbereitung der Daten gesteckt wird, ist jedoch entscheidend für die Lerngeschwindigkeit des Netzes und auch für die Qualität der Approximation durch das Netzwerk. Jede Stunde, die in die Aufbereitung der Daten gesteckt wird, kann beim Training des Neuronalen Netzes Tage einsparen.

Bei der im Modul verwendeten Vorverarbeitung des Audio-Signals wurden übliche Algorithmen der Signalverarbeitung verwendet. Es wird zunächst ein eventuell vorhandener (z.B. vom Mikrofon verursachter) Gleichspannungsanteil (*Offset*) herausgerechnet, das Sample also zentriert. Anschließend erfolgt durch einen Skalierungsfaktor die Transformation der Daten in den gewünschten Wertebereich[9]. Diese Normierung auf maximale Lautstärke dient der Kompensation unterschiedlicher Aufnahmelautstärken (Prozedur „Normalize“).

Das so bearbeitete Sample wird danach in Segmente (*Frames*) von 256 Punkten zerlegt und mit einem der in der Oberfläche wählbaren Algorithmen gefenstert[10] (Prozedur „ApplyWindow“). Von dem nach der anschließenden Fourier-Transformation (Prozedur „ApplyFFT“) für jedes Fenster erhaltenen Leistungsspektrum werden die ersten 128 Kanäle (ohne redundante Information) aufsummiert, so dass man für jeden dieser Frames einen repräsentativen Wert seiner Gesamtleistung erhält. Aus der nun möglichen Abbildung der Signalleistung auf die Zeit kann mit dem unter 4.3.2 beschriebenen Verfahren das eigentliche Wort detektiert werden (Prozedur „DetectWord“).

Innerhalb der Wortgrenzen werden die Frequenzanteile der Frames Bark-skaliert, d. h. zu Bändern zusammengefasst, wie in 4.3.1 anklang. Die gewünschten Frequenzgruppen sind als Bandfilter approximiert worden [Volk]. Im letzten Schritt werden die 18 Leistungswerte des Barkspektrums in eine entsprechende Lautheit umgewandelt (Prozedur „ApplyBark“).

[9] Bei Auflösung von 16 Bit: Max. Lautstärke=2^{16}=65536. Für WAV-Dateien: Integerwerte zwischen -32768 und +32767.

[10] Voreinstellung: Hammingfenster; Breite des Fensters: 256 Abtastwerte; keine Überlappung (der Parameter Überlapp zwischen den Fenstern hat sich nach [Gram] als relativ unkritisch erwiesen).

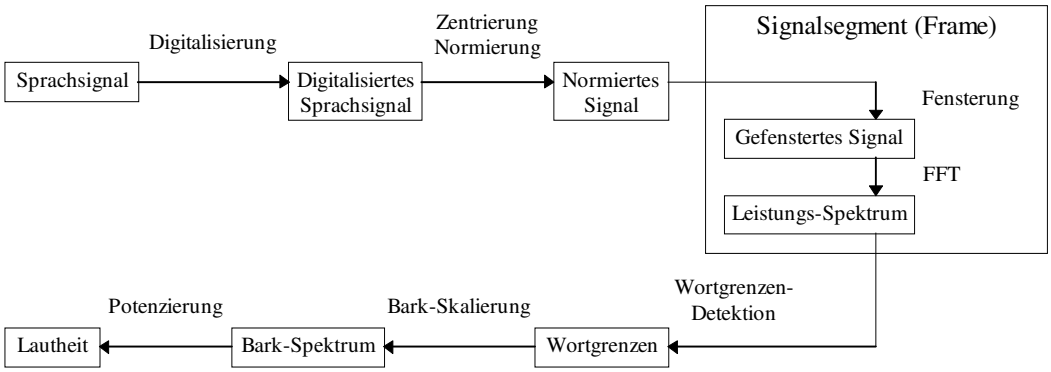

Abbildung 5.3: Signal-Vorverarbeitung

Damit das Netz in der Lage ist, die z. T. stark abweichenden (4.4) Sprachsamples zu erlernen, wird der aus 18 Lautheiten zwischen 0 und 3000 bestehende Eingangsvektor umcodiert auf Binärwerte.

Die Oberfläche des Moduls „SpeechRecognition.dll“ besteht, neben der üblichen Statusleiste, aus drei TabSheets. Ihre Definition erfolgt in der Klasse „TSpeechRecogForm“, welche Teil der Unit „SpeechRecogFormFile“ ist. Außerdem enthalten sind hier mehrere, nur zur Entwicklungszeit sichtbare, nicht-visuelle Komponenten. Dazu zählt eine Instanz von TBackPropNet, welche die Netzfunktionalität nachbildet, ein Timer zur Steuerung der Aufnahme, zwei ImageLists, welche die Bilder zur Ergebnispräsentation speichern und auch ein NetFileDialog, der für die Dateiarbeit zuständig ist.

Zur graphischen Darstellung des Signals in Zeit- und Frequenzebene wurde auf der Seite „AudioDatei“ des Formulars eine „TTabControl“-Komponente eingefügt. Mit ihrer Hilfe kann zwischen den unterschiedlichen Darstellungsweisen des Signals gewählt werden, wobei in den dazugehörigen Panels jeweils auch unterschiedliche Datei- und Signalverarbeitungs-Parameter zur Anzeige kommen. Die Werte für die Wortgrenzendetektion und der Fensterungs-Algorithmus können in SpinEdits bzw. ComboBoxes eingegeben werden. Der MediaPlayer dient der Aufzeichnung und Wiedergabe von Wave-Dateien, die mit dem BitBtn „Öffnen“ auch geladen werden können.

Die Seite „MusterSet“ dient, wie im Modul „BackPropagation“, der Zusammenstellung mehrerer Muster für das Anlernen des Netzes. Ihre Bedienung erfolgt weitgehend analog, jedoch werden Audio-Dateien als Eingangsvektoren bestimmt, denen ein Binärcode zugeordnet wird. Diese binär-codierten Zielvektoren haben die Dimension 16 und können ebensoviele, verschiedene Samples klassifizieren. Neben seiner Funktion als

Klassifizierer kann das Modul auch als Mustererkenner genutzt werden, wenn mehr als 16 Worte (maximal 2^{16}) mit Hilfe der Ausgangswerte codiert werden. Doch steigt mit dem Grad der Codierung auch die Wahrscheinlichkeit, dass das Netz keine Lösung mehr erreichen kann.

Das Ziel der Arbeit auf der dritten Seite ist es schließlich, eine angegebene Audio-Datei zu erkennen. Der Erfolg/Misserfolg wird dem Anwender mitgeteilt durch die Bewegung des Objektes im Ergebnis-Panel. Zur Kontrolle des erhaltenen Ergebnis-Strings wird dieser zusätzlich in einem Panel ausgegeben.

Das in der RadioGroup „ErkennungsTyp“ wählbare Verfahren „Online“ bezieht sich nicht auf einen echtzeitfähigen Algorithmus. Eine Bearbeitung erfolgt also nicht schon während, sondern erst nach dem Sprechen. Die Bezeichnung bezieht sich lediglich auf den automatisierten Ablauf der einzelnen Schritte zur Einzelworterkennung, wonach eine frisch aufgezeichnete Datei sofort der Signalverarbeitung unterzogen und dann durch das Neuronale Netz reproduziert wird, das ein Ergebnis liefert.

5.4.4 Hilfe-System

In das Programm „Res Neuronum“ wurde ein relativ umfangreiches Hilfe-System integriert, das nicht nur Erläuterungen zur Bedienung gibt, sondern auch einiges Basiswissen zur Verfügung stellt. Das System setzt sich zusammen aus mehreren Hilfe-Dateien und es können auch hier Hilfe-Projekte neuer Module problemlos eingebunden werden.

Zur Erstellung des Hilfe-Systems wurde der „Microsoft Help Workshop“ in der Version 4.00.0950 verwendet. Bevor dieser jedoch zur Anwendung kommen konnte, mussten die Hilfe-Themen in einem Textverarbeitungs-Programm („Word für Windows 7“) verfasst werden. Die entsprechenden Dateien wurden im RTF-Format (*Rich-Text-File*) abgelegt, da im Text wichtige Fußnoten, unterstrichene und verborgene Kennwörter enthalten sind. Jetzt konnten die Hilfe-Projekte (.hpj) kreiert werden und im Anschluss daran die Inhalts-Dateien (.cnt). Schließlich wurde das Hilfe-Projekt durch den Help Workshop compiliert, wozu er alle oben beschriebenen Dateien miteinander verband. Als Resultat entstanden die Hilfe-Dateien im herkömmlichen Format (.hlp).

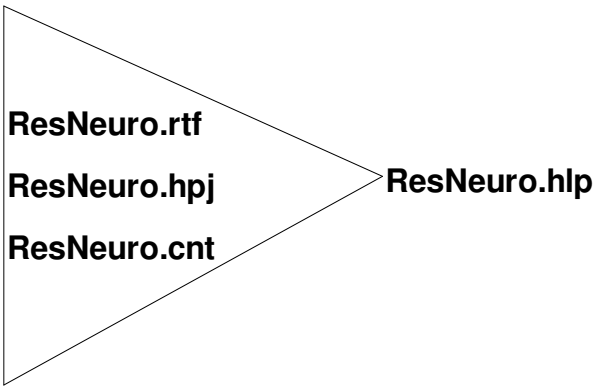

Abbildung 5.4: Erstellung einer Hilfe-Datei

5.4.5 Setup-Programm

Um dem Anwender eine komfortable Installation neuer Software zu ermöglichen, werden zu dieser gewöhnlich eigene Setup-Programme mitgeliefert. Zur Generierung einer solchen Setup-Routine für „Res Neuronum“ wurde der „Install Shield Express“ von Stirling Technologies, Version 1.0 verwendet. Das entstandene Programm leitet den Anwender sicher durch alle Schritte der Installation.

6 Zusammenfassung und Ausblick

Das Ziel der Aufgabenstellung zur Diplomarbeit war es, Wissen über Künstliche Neuronale Netzwerke innerhalb eines zu erstellenden Praktikumsversuches auf anschauliche Weise vermittelbar zu machen. Die Vorgehensweise bei der Lösung des gestellten Problems richtete sich nach einem unter dem englischen Begriff „*Value Engineering*“ bekannten Verfahren. Wie die zu ihm gehörenden fünf Schritte in dieser Diplomarbeit umgesetzt wurden, sei hier noch einmal kurz zusammengefasst.

Zu Beginn stand die Sammlung unterschiedlichster Informationen zum Thema (*Information Phase*), wobei sich Grundlagenwissen aus zahlreichen Quellen angeeignet wurde. Dies umfasste neben dem Gebiet der Neuronalen Netze im Allgemeinen auch jene der Spracherkennung und Programmierung unter Delphi. Im zweiten Arbeitsgang (*Function Phase*) kam es darauf an, wichtige von unwichtigen Materialien zu trennen und nicht realisierbare Ideen von vornherein auszuschließen. So wurde festgestellt, dass sich Fehlerrückführungsnetze auf Grund ihrer einfachen Struktur und leichten Implementierbarkeit besonders zu Demonstrationszwecken eignen; nicht zuletzt für sie sprach auch ihre weite Verbreitung. Als häufig verwandte, anschauliche Aufgabenstellungen konnten logische Probleme wie AND und XOR identifiziert werden. Weiterhin wurde erkannt, dass für die Zusatzaufgabe lediglich ein Algorithmus zur Einzelworterkennung in Frage käme. Das Erkennen fließender Sprache würde neben dem Erstellen umfangreicher Wort-/Phonem-Datenbanken incl. Suchverfahren auch in dieser kurzen Zeit nicht zu realisierende Algorithmen zur Merkmalsextraktion und -klassifikation beinhaltet haben. Eine Hardware-Realisierung schied nicht nur wegen des vorgegebenen Themas sondern auch aus Gründen der Flexibilität und Praktikabilität von vornherein aus. Während der nun folgenden Phase (*Speculation Phase*) wurde ein *Brainstorming*, d. h. eine zunächst völlig unbewertete Sammlung von Ideen zur Realisierung des Projektes durchgeführt. Zu den Vorschlägen zählten: *Multi-Document-Interface*-Programm, einfache, kompakte Simulation und modularer Ansatz, der auf Dynamischen Bibliotheken basiert. Das Programm sollte einfach zu bedienen sein und auf anschauliche, wissenschaftlich korrekte Weise die Arbeit mit Neuronalen Netzen demonstrieren. Vor- und Nachteile der gesammelten Möglichkeiten mussten in einer vierten Phase (*Evaluation Phase*) klar gegenübergestellt werden. Hier kristallisierte sich als bester der

modulare Ansatz heraus. Im letzten Schritt (*Implementation Phase*) erfolgte die Übertragung des Projektes in die Praxis.

Als Ergebnis kann eine allgemein verwendbare, computergestützte Simulation zu Künstlichen Neuronalen Netzwerken präsentiert werden, die nicht nur im Laborversuch des Praktikums „Medizinische Informatik“ einsetzbar, sondern auch für andere Aufgabengebiete erweiterbar sein sollte. In den Aufgabenstellungen zum Praktikums-Versuch werden besonders die grundlegenden Fragen des Aufbaus, der Funktion und der Arbeitsweise mit Neuronalen Netzen behandelt. Mit der Applikation „Spracherkennung“ konnte zudem eine Erweiterung des Versuchs um einen wirklich praktischen Teil geschaffen werden.

Für die Zukunft ergibt sich zunächst die Möglichkeit, weitere Arten Neuronaler Netzwerke oder Anwendungen für diese in den Simulator zu integrieren. Durch dessen modularen Aufbau steht dabei genügend Gestaltungsfreiraum zur Verfügung. Um die Portabilität in andere Programme und Sprachen zu gewährleisten, ist eine Konzipierung der Module als Dynamische Bibliotheken anzustreben.

Denkbar wäre auch eine Erweiterung des Programms zu einem Datenanalyse-System allgemeiner Natur, ähnlich dem Software-Tool „DataEngine“ (siehe dazu [Zimm]). Dazu könnten neben Neuronalen Netzen auch die mit ihnen verwandten Methoden aus dem Bereich der Fuzzy-Technologien eingebaut werden.

Naheliegender erscheint indes die Empfehlung, je nach aktuellem Stand der Forschung und abzusehendem Trend, im Tempo der Weiterentwicklung des Programms auch den Praktikumsversuch zu aktualisieren. Konkrete Möglichkeiten zur Verbesserung des Programms sind gegeben durch:

- Steigerung der Performance des Simulators „Fehlerrückführung“ durch das Ersetzen der Listen als dynamische Speicherstrukturen (zur Speicherung der Netzdaten während des Lernens und Optimierens) mit dynamischen Arrays.
- Implementierung weiterer Datenverarbeitungs-Verfahren auf der Seite „Auswerten“ des Moduls „Fehlerrückführung“ und somit Schaffung zusätzlicher Werkzeuge zur Evaluierung der Lernparameter für die Entdeckung neuer Zusammenhänge.
- Aus solcherart Zusammenhängen und Abhängigkeiten könnten zusätzliche Kriterien für den Strukturoptimierungs-Algorithmus auf der Seite „Optimieren“ des

Moduls „Fehlerrückführung“ gefunden werden, wodurch eine nochmals verbesserte, automatisierte Suche der optimalen Netzkonfiguration ermöglicht würde.

- Aufnahme von Hinweisen, die an bestimmten Stellen der Programmausführung erscheinen und Hilfestellungen zur Findung einer effektiven Netzstruktur anbieten (z. B. abgestimmte Anzahl an Neuronen und Mustern, günstige Lernrate etc.). Diese anwendungsgerechten Empfehlungen können nach allgemein bekannten, heuristischen (Faust-) Regeln berechnet werden. (Bei der Suche einer günstigen Netzkonfiguration für das Modul „Spracherkennung“ wurde festgestellt, dass ein solches Hinweissystem durchaus zur Beschleunigung der Arbeit beitragen könnte.)
- Ausbau des Einzelworterkenners, beispielsweise durch Hinzunahme einer Kohonenkarte zur besseren Merkmalsbildung; Verbesserung der Daten-Aufbereitung durch neue Codierungs-Verfahren; Anwendung des Cross-Validation-Lernverfahrens, um eine zu große Spezialisierung des Netzes zu vermeiden; langfristig evtl. Erweiterung auf Online-Spracherkennung.

Nicht zuletzt bietet sich der Einsatz des Netzes in modifizierter Form für verschiedene, praktische Aufgabenstellungen an. Grundsätzlich kommen alle Probleme, bei denen scharfe/unscharfe Objekte/Klassen aufeinander abgebildet werden, in Betracht. Im Rahmen von Prognosen für Börsenkurse konnte die Tauglichkeit des Mehrschichtigen Perceptrons mit Fehlerrückführungs-Lernregel für Zeitreihenvorhersagen bereits bestätigt werden. Es bleibt zu untersuchen, in wie weit eine Eignung für den Einsatz im Medizintechnik-Sektor, beispielsweise die EEG- /EKG-Analyse besteht.

7 Verzeichnisse

7.1 Abkürzungsverzeichnis

ADALINE	ADAptive LINear Element
AI	Artificial Intelligence (dt: KI)
ANN	Artificial Neural Network (dt: KNN)
ART 1-3	Adaptive Resonance Theory 1/2/3
ASCII	American Standard Code for Information Interchange
BAM	Bidirectional Associative Memory
BP	BackPropagation network
BSB	Brain State-in-a-Box
CK	Chunk
CNT	CoNTents file
CPU	Central Processing Unit
DCN	Divide and Conquer Network
DLL	Dynamic Link Library
DMA	Distributed Memory and Amnesia
DMFCC	Dynamic Mel Frequency scaled Cepstral Coefficient
DTW	Dynamic Time Warping
DWORD	Double WORD
EEG	ElectroEncephaloGram
EKG	ElectroKardioGram
EXE	EXEcutable file
FOURCC	FOUR Character Code
FFNN	Feature Finding Neural Network
FFT	Fast Fourier Transformation
GUI	Graphical User Interface
HLP	HeLP file
HMM	Hidden Markov Model
HPJ	Help ProJect
INI	INItialisation file
MDI	Multi Document Interface
MLP	Multi Layer Perceptron
OBD	Optimal Brain Damage
OBS	Optimal Brain Surgeon
PCM	Pulse Code Modulation
RIFF	Resource Interchange File Format
RTF	Rich Text File format
SB 16	SoundBlaster 16
SSM	Statistical Stepwise Method
TDNN	Time Delay Neural Network
VCL	Visual Component Library

7.2 Symbolverzeichnis

Das im Folgenden aufgeführte Verzeichnis erläutert die Symbole, welche in dieser Arbeit oder in der dazugehörigen Simulations-Software benutzt werden. Es hält sich – was Neuronale Netze betrifft – an die von Hoffmann [Hoff] verwendete Bezeichnungsweise. (Ebenda sind sehr anschauliche, vergleichende Übersichten für andere Symboliken zu finden.) Alle weiteren Symbole wurden aus den im Kontext angegebenen Quellen entnommen.

$A_1, A_2, \ldots$	Ausgänge eines Netzes
$A(t)$	Ausgangsmusterfolge
a_i	Ausgang des Neurons i, Ausgangsfunktion des Neurons i
$\vec{b}_i$	Leistungswerte des Bark-Spektrums
b_i	Lautheit
c_i	Aktivität des Neurons i, Aktivierungsfunktion des Neurons i
d	Abklingkonstante in der Aktivierungsfunktion
$E_1, E_2, \ldots$	Eingänge eines Netzes
$E(t)$	Eingangsmusterfolge
e_0	Bias-Eingang
$e_1, e_2, \ldots$	Eingänge eines Neurons
f	Frequenz in Hertz
i (Index)	numeriert gewöhnlich die Neuronen eines Netzes
j (Index)	numeriert gewöhnlich die Eingänge eines Neurons
k	Kanäle des Bark-Spektrums
M/m	Maximum bzw. Minimum in der Aktivierungs- und in der Ausgangsfunktion
N_A / N_E	Anzahl der Ausgänge bzw. Eingänge eines Netzes
N_i	Anzahl der Neuronen in der Schicht i
n_i	Anzahl der Eingänge des Neurons i
p	Anzahl der Muster in einem Trainingssatz
S	Schwelle bei der Wortgrenzendetektion
$S_1, S_2, \ldots$	Sollwerte eines Netzes
s	Skalierungsfaktor in der Aktivierungsfunktion
t	Zeit in Sekunden
W_a / W_i	Untere bzw. obere Toleranzschwelle bei der Wortgrenzendetektion
w_0	Bias-Gewicht
w_{ij}	Gewicht des Neurons i am Eingang j
x_0, x_1	Eingangskomponenten beim XOR-Problem
Z	Zähler bei der Wortgrenzendetektion
z	Tonheit in Bark
δ_{ij}	Kroneckersymbol: 1 für i = j, 0 für i ≠ j
δw_{ij}	Änderung des Gewichts w_{ij}
ε_i	effektiver Eingangswert des Neurons i
η	Lernrate in der Lernregel
ϑ	Schwelle der Ausgangsfunktion
μ	Momentfaktor in der Fehlerrückführungs-Lernregel
μ (Index)	numeriert gewöhnlich die Muster eines Trainingssatzes
σ	Steigung der Ausgangsfunktion

7.3 Gleichungsverzeichnis

Gleichung 1.1: Eingangsfunktion (Skalarprodukt) ... 9
Gleichung 1.2: Aktivierungsfunktion (linear) ... 9
Gleichung 1.3: Ausgangsfunktion (linear) ... 9
Gleichung 1.4: Erweiterte Fermi-Funktion ... 14
Gleichung 1.5: Fehlermaß für Neuron i der Ausgangsschicht ... 15
Gleichung 1.6: Fehlermaß in verborgenen Schichten ... 15
Gleichung 1.7: Fehlerrückführungs-Lernregel ... 15
Gleichung 1.8: Neuberechnung der Gewichte ... 15
Gleichung 1.9: Lernregel mit Momentumterm ... 17
Gleichung 1.10: Adaptiver Mittelwert ... 18
Gleichung 1.11: Hebbsche Lernregel ... 21
Gleichung 1.12: Delta-Lernregel ... 22
Gleichung 2.1: Logisches XOR-Problem ... 26
Gleichung 4.1: Berechnung der Bark-Frequenzen ... 33
Gleichung 4.2: Transformation Leistung → Lautheit ... 33

7.4 Tabellenverzeichnis

Tabelle 2.1: Logische XOR-Verknüpfung ... 25

7.5 Abbildungsverzeichnis

Abbildung 1.1: Struktur eines künstlichen Neurons ... 9
Abbildung 1.2: Netzwerk ohne (2-schichtig) und mit Verteilungsneuronen (3-schichtig) ... 10
Abbildung 1.3: Arten Neuronaler Netze – klassifiziert nach Aufbau und Lernmethode ... 12
Abbildung 1.4: Struktur und Arbeitsphasen eines Fehlerrückführungsnetzes ... 13
Abbildung 1.5: Hinton-Diagramm mit zwei Gewichten ... 16
Abbildung 1.6: Methoden zur Strukturoptimierung ... 20
Abbildung 2.1: Arbeitsweisen der Netze – klassifiziert nach ihrem Reproduktionsverhalten ... 24
Abbildung 2.2: Entscheidungsregionen für AND- und XOR-Problem ... 26
Abbildung 4.1: Teilgebiete der Maschinellen Spracherkennung ... 31
Abbildung 4.2: Wortgrenzendetektion ... 34
Abbildung 5.1: Aufbau eines Chunks ... 43
Abbildung 5.2: Beschreibung einer WAVE-Datei ... 44
Abbildung 5.3: Signal-Vorverarbeitung ... 46
Abbildung 5.4: Erstellung einer Hilfe-Datei ... 48

7.6 Quellenverzeichnis

7.6.1 Literaturzitate

Aved'yan, Eduard. *Learning Systems*. London: Springer, 1995.

Bec, L. „Eléments d'Epistemologie Fabulatoire". *Artificial Life II: the proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems held 1990 in Los Alamos, New Mexico*. Reading: Addison-Wesley, 1990. 799-811.

Boner, Andreas. *Spracherkennung mit Computer*. AT Verlag; VDE-Verlag, 1992.

Brause, Rüdiger. *Neuronale Netze: Eine Einführung in die Neuroinformatik*. Stuttgart: Teubner, 1991.

Frey, Hans Georg und Justus Schach. „Modern geknotet: Neuro-Netze: Strukturen, Typen, Anwendungen". *c't*. Heft 2 (Februar 1996): 256-264.

Gramß, Tino. *Worterkennung mit einem künstlichen neuronalen Netzwerk*. Dissertation. Göttingen: Georg-August-Universität zu Göttingen, 1992.

Guyot, Frédéric, Frédéric Alexandre, Catherine Dingeon and Jean-Paul Haton. „The cortical column as a model for speech recognition: principles and first experiments". *Speech Recognition and Understanding: Recent Advances, Trends and Applications – NATO ASI Series*. Vol. F 75. Edited by P. Laface and R. De Mori. Berlin; Heidelberg: Springer-Verlag, 1992.

Heller, Christian. *Implementierung eines Strukturoptimierungsalgorithmus' für neuronale Backpropagation-Netzwerke*. Studienjahresarbeit. Ilmenau: TU Ilmenau, 1996.

Hoffmann, Norbert. *Kleines Handbuch neuronale Netze: anwendungsorientiertes Wissen zum Lernen und Nachschlagen*. Braunschweig; Wiesbaden: Vieweg, 1993.

Huang, W. und R. Lippman. „A Neural Net Approach for Speech Recognition". *Proc. IEEE-ICASSP*. New York, 1988.

Jutten, Christian und Olivier Fambon. *Pruning Methods: A Review*. [Quellzeitschrift lag vor im Bereich Neuroinformatik, Paterre, BMTI-Gebäude, TU Ilmenau].

Kinnebrock, Werner. *Neuronale Netze: Grundlagen, Anwendungen, Beispiele*. München; Wien: Oldenbourg, 1992.

Krause, Thomas. *Realisierung eines Demonstrationsprogrammes zur Anwendung Neuronaler Netze in der Spracherkennung*. Studienjahresarbeit. Ilmenau: TU Ilmenau, 1995.

Lernparadigmen. Anleitung zum Praktikum Neuroinformatik Nr. 1. Hg. Fachgebiet Neuroinformatik. Ilmenau: TU Ilmenau, 1996.

Mak, M. W., W. G. Allen und G. G. Sexton. „Speaker identification using radial basis functions“. *IEE Third Int. Conf. on ANN: Conference Publication*. No. 372: 138-142. London: Institution od Electrical Engineers, 1993.

MATLAB: Neural Network Toolbox-User's Guide. Hg. The Math Works, Inc. Natick, June 1992.

Pschyrembel Klinisches Wörterbuch: mit klinischen Syndromen und Nomina Anatomica. Bearb. von d. Wörterbuchred. d. Verl. unter Leitung von Christoph Zink. 256., neu bearb. Aufl. Berlin; New York: de Gruyter, 1990.

Ryan, Rob. *RIFF WAVE (.WAV) file format*. e-mail: ST802200@brownvm.brown.edu. Dokument „rmrtf.zrt“ im Verzeichnis „vendor/microsoft/multimedia“ bei Adresse „ftp.uu.net“. Brown University, 1997.

Schuster, H. „His Master's Voice: Entwicklungsstand der Spracherkennung mit Computern“. *c't*. Heft 11 (Nov. 1993): 167-176.

„So funktioniert Spracherkennung im Computer“. *Arzt Online: Das Computermagazin der Ärzte Zeitung*. Heft 3 (Mai 1997): 20. Hg. Ärzte Zeitung Verlagsgesellschaft mbH. Neu-Isenburg, 1997.

Sprach-Steuerungs-Software. Fernseh-Dokumentation. Aus: Neues…die Computer-Show. http://www.3sat.com: 3SAT, 1997.

Steuer, Dunja. *Entwicklung Adaptiver Verfahren und deren Anwendung in Problemstellungen der Biosignalanalyse*. http://www-bmti.tu-ilmenau.de, 1997.

Stolz, Axel. Das Soundblaster Buch. 1. Aufl. Düsseldorf: Data Becker, 1992.

Volk, Thomas. *Spracherkennung*. Diplomarbeit. Ilmenau: TU Ilmenau, 1996.

Warken, Elmar. *Delphi 2: Software-Entwicklung für 32-Bit-Windows.* Bonn; Paris [u. a.]: Addison-Wesley, 1996.

Zell, Andreas. *Simulation Neuronaler Netzwerke*. 1. Aufl. Addison-Wesley (D) GmbH, 1994.

Zhu, M. Sc. Ming. *Sprecherunabhängige Erkennung von isoliert gesprochenen Einzelwörtern unter Verwendung der Vektorquantisierung und von Neuronalen Netzen*. Dissertation. Berlin: TU Berlin, 1992.

Zimmermann, Hans-Jürgen. *Datenanalyse: Anwendung von DataEngine mit Fuzzy Technologien und Neuronalen Netzen*. Düsseldorf: VDI-Verl., 1995.

7.6.2 Andere Literaturverweise aus dem Text

Aarts, E. und J. Korst. *Simulated Annealing and Boltzmann Machines*. Chichester, 1990.

Ackley, D. H., G. E. Hinton und T. J. Sejnowski. „A learning algorithm for Boltzmann machines“. *Cognitive Sciences*. 9 (1985).

Duden, Rechtschreibung der deutschen Sprache. 21., völlig neu bearb. und erw. Aufl. Hg. Dudenredaktion [auf der Grundlage der neuen Rechtschreibregeln]. Mannheim; Leipzig; Wien; Zürich: Dudenverl., 1996.

Fukushima, K. „Cognitron: A Self-organizing Multilayered Neural Network“. *Biological Cybernetics*. Nr. 20 (1975): 121-136.

Fukushima, K. Neocognitron: Self-organizing Neural Network Model for Mechanism of Pattern Recognition Uneffected by Shift in Position. *Biological Cybernetics*. Nr. 36 (1980): 193-202.

Hebb, D. *Organisation of Behaviour*. New York: Wiley, 1949.

Hecht-Nielsen, R. „Counterpropagation Networks“. *Proc. of the International Conf. of Neural Networks*. Nr. 2. New York: IEEE Press, 1987. 19-32.

Hinton, G. E., D. E. Rummelhart und J. L. McClelland. *Parallel Distributes Processing. Vol. 1 and 2*. MIT Press, 1987.

Hopfield, J. J. „Neural Networks and physical systems with emergent collective computational abilities“. *Proc. of the Nat. Academy of Sciences*. Nr. 79. Washington, 1982.

Itakura, F. „Minimum prediction residual principle applied to speech recognition“. *IEEE Trans. Acoustics, Speech, Signal Proc*. Nr. 23: 67-72, 1975.

Keller, Michael. *Einzelworterkennung für Sprache in gestörter Umgebung*. Dissertation. Karlsruhe: Universität Fridericiana Karlsruhe, 1991.

Kohonen, T. *Self Organization and Assoziative Memory*. 2. Aufl. Berlin: Springer, 1988.

Kosko, B. „Bidirectional Assoziative Memory“. *IEEE Transactions on Systems. Man and Cybernetics*. Nr. 17 (1987).

McCulloch, W. S., W. Pitts. „A logocal calculus of the ideas immanent in nervous activity“. *Bull. Math. Biophys*. 5 (1943): 115-133.

Minski, M., S. Papert. *Perceptrons*. Cambridge; Massachusetts: MIT Press, 1969.

Ritter, H., Klaus Schulten und Thomas Martinetz. *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke*. Bonn; München; Reading, Mass. [u. a.]: Addison-Wesley, 1991.

Rosenblatt, F. „The Perceptron: a probabilistic model for information storage and organization in the brain“. *Psychological Review*. Nr. 65 (1958).

Ruske, G. and W. Weigel. „Syllable-based stochastic models for continuous speech recognition“. *Speech Recognition and Understanding: Recent Advances, Trends and Applications – NATO ASI Series*. Vol. F 75. Edited by P. Laface and R. De Mori. Berlin; Heidelberg: Springer-Verlag, 1992.

Sakoe, H. et al. Speaker-independent word recognition using Dynamic Programming Neural Networks. *Proc. IEEE-ICASSP*. Glasgow, 1989.

Steuer, Dunja, Gert Grießbach. *Anwendung adaptiver Schätzalgorithmen zur Beurteilung des Lernverhaltens Neuronaler Netze*. Beitrag zur 39. Jahrestagung der Gesellschaft für Biomedizinische Technik. Rostock, September 1994.

Trautwein, Michael. *Anbindung einer Datenbank an einen Netzwerk-Simulator unter Windows und Schaffung von Tools zur Auswertung der Lerndaten*. Diplomarbeit. Ilmenau: TU Ilmenau, 1995.

Widrow, B., M. E. Hoff. *Adaptive switching circuits*. New York: Ire Wescon Convention Record, 1960.

7.6.3 Sonstige Quellen

Bharath, Ramachandran. *Neural Network computing*. New York: Windcrest/McGraw-Hill, 1994.

Bishop, Christopher M. *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995.

Böhme, H.-J. *Künstliche Neuronale Netzwerke*. Arbeitsblätter zur Vorlesung. Ilmenau: TU Ilmenau, Wintersemester 1994/95.

Borland Delphi: Das Buch. Hg. Borland GmbH Deutschland. München: Tewi, 1995.

Braun, H., J. Feulner und R. Malaka. *Praktikum neuronale Netze.* Berlin; Heidelberg: Springer, 1996.

Delphi Benutzerhandbuch. Hg. Borland GmbH. Langen, 1995.

Delphi 2.0: Developer Version. Computer-Software. Programmiersprache. Borland, 1996.

Dorffner, Georg. *Konnektionismus: von neuronalen Netzen zu einer „natürlichen“ KI*. Stuttgart: Teubner, 1991.

Eger, H. Angewandte Anatomie, Physiologie und Strahlenbiologie. Vorlesungsmitschrift. Ilmenau: TU Ilmenau, 1993.

Ehlers, Frank. Hörhilfe: „Spracherkennung aus WAV-Dateien“. *c't*. Heft 11 (Nov. 1993): 186-191.

Grießbach, G. *Digitale Signalverarbeitung*. Vorlesungsmitschrift. Ilmenau: TU Ilmenau, 1993.

Help Workshop 4.00.0950. Computer-Software. Hilfe-Generator für Windows-Hilfe-Dateien. Microsoft, 1994-1995.

Install Shield Express 1.0 für Borland Delphi. Generator für Setup-Programme. Stirling Technologies, 1996.

Johnson, Johnny R. *Digitale Signalverarbeitung*. Übs. Gerhard Schmidt. München; Wien: Hanser; London: Prentice-Hall Internat., 1991.

Krol, Rüdiger. *Implementation eines Neuronalen Backpropagation-Netzes und Erweiterung zur Beschleunigung des Lernprozesses und dessen Stabilisierung*. Studienjahresarbeit. Ilmenau: TU Ilmenau, 1995.

Newsgroups. *comp.lang.pascal.delphi*; *comp.ai*.

Paal, Gerhard. *Hexal-Lexikon Neurologie*. München; Wien; Baltimore: Urban und Schwarzenberg, 1995.

Pacheco, Xavier und Steve Teixeira. *Delphi Developer's Guide*. Hg. Borland Press. Indianapolis: Sams Publishing, 1995.

Peters, Claudia. *Implementation eines objektorientierten Backpropagation-Netzwerkes mit Neuronen höherer Ordnung*. Studienjahresarbeit. Ilmenau: TU Ilmenau, 1995.

Poenicke, Klaus. „Wie verfaßt man wissenschaftliche Arbeiten?: Ein Leitfaden vom 1. Studiensemester bis zur Promotion“. *Die Duden-Taschenbücher*. Bd. 21. 2., neu bearb. Aufl. Mannheim; Wien; Zürich: Dudenverl., 1988.

Rossini, Gioacchino. „Wilhelm Tell“. *Ouvertüren*. Compact Disk. Best.-Nr. 11152. Extra records & tapes, 1995.

Schäpers, Arne. „Ableitungen und Abwege: Listen in Delphi“. *c't*. Heft 5 (Mai 1996): 292-297.

Scholle, Anders und Schumann. *Elektro- und Neurophysiologie*. Vorlesungsmitschrift. Jena; Ilmenau: FSU Jena; TU Ilmenau, 1995.

Warken, Elmar. *Delphi: Entwicklung leistungsfähiger Anwendungen und eigener Komponenten*. Bonn; Paris [u. a.]: Addison-Wesley, 1995.

Windows 95. Computer-Software. Betriebssystem für PC. Microsoft, 1995.

Word for Windows 7. Computer-Software. Textverarbeitungsprogramm. Microsoft, 1995.

7.6.4 Warenzeichen

Folgende, in der Arbeit verwendete Namen sind in einigen Ländern eingetragene Warenzeichen (®, ™) bestimmter Unternehmen:

Warenzeichen	**Eigentümer**
Borland C++	Borland International, Inc.
Delphi	Borland International, Inc.
Dragon Dictate	Dragon Systems, Inc.
Help Workshop	Microsoft Corporation
IBM	IBM Corporation
Install Shield	Stirling Technologies, Inc.
MATLAB	Math Works, Inc.
Microsoft	Microsoft Corporation
OS/2	IBM Corporation
Simply Speaking	IBM Corporation
Smart Office	Terratec, Inc.
Sound Blaster	Creative Labs, Inc.
Voice Type	IBM Corporation
Windows	Microsoft Corporation
Word	Microsoft Corporation

Anhänge

A Thesen

B Benutzerhandbuch Res Neuronum

C Praktikumsanleitung

D Musterprotokoll