Interner Bericht

240

Theorie und Praxis neuronaler Netze
- Seminar SS 93 -

Oliver Wendel
Aldo von Wangenheim
(Hrsg.)

Fachbereich Informatik

Universität Kaiserslautern · Postfach 3049 · D-6750 Kaiserslautern

Theorie und Praxis neuronaler Netze
- Seminar SS 93 -

Oliver Wendel
Aldo von Wangenheim
(Hrsg.)

Arbeitsgruppe Künstliche Intelligenz
– Expertensysteme –
Prof. Dr. Michael M. Richter

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern

Tel: 0631-205-3327, 3358
wendel@informatik.uni-kl.de
awangenh@informatik.uni-kl.de

1. Auflage

Juli 1993

Vorwort

The brain happens to be a meat machine.
Marvin Minsky

Neuronale Netze sind ein derzeit (wieder) aktuelles Thema. Trotz der oft eher schlagwortartigen Verwendung dieses Begriffs beinhaltet er eine Vielfalt von Ideen, unterschiedlichste methodische Ansätze und konkrete Anwendungsmöglichkeiten. Die grundlegenden Vorstellungen sind dabei nicht neu, sondern haben eine mitunter recht lange Tradition in angrenzenden Disziplinen wie Biologie, Kybernetik, Mathematik und Physik. Vielversprechende Forschungsergebnisse der letzten Zeit haben dieses Thema wieder in den Mittelpunkt des Interesses gerückt und eine Vielzahl neuer Querbezüge zur Informatik und Neurobiologie sowie zu anderen, auf den ersten Blick weit entfernten Gebieten offenbart.

Gegenstand des Forschungsgebiets Neuronale Netze ist dabei die Untersuchung und Konstruktion informationsverarbeitender Systeme, die sich aus vielen mitunter nur sehr primitiven, uniformen Einheiten zusammensetzen und deren wesentliches Verarbeitungsprinzip die Kommunikation zwischen diesen Einheiten ist, d.h. die Übertragung von Nachrichten oder Signalen. Ein weiteres Charakteristikum dieser Systeme ist die hochgradig parallele Verarbeitung von Information innerhalb des Systems. Neben der Modellierung kognitiver Prozesse und dem Interesse, wie das menschliche Gehirn komplexe kognitive Leistungen vollbringt, ist über das rein wissenschaftliche Interesse hinaus in zunehmendem Maße auch der konkrete Einsatz neuronaler Netze in verschiedenen technischen Anwendungsgebieten zu sehen.

Der vorliegende Report beinhaltet die schriftlichen Ausarbeitungen der TeilnehmerInnen des Seminars *Theorie und Praxis neuronaler Netze*, das von der Arbeitsgruppe Richter im Sommersemester 1993 an der Universität Kaiserslautern veranstaltet wurde. Besonderer Wert wurde darauf gelegt, nicht nur die theoretischen Grundlagen neuronaler Netze zu behandeln, sondern auch deren Einsatz in der Praxis zu diskutieren. Die Themenauswahl spiegelt einen Teil des weiten Spektrums der Arbeiten auf diesem Gebiet wider. Ein Anspruch auf Vollständigkeit kann daher nicht erhoben werden. Insbesondere sei darauf verwiesen, daß für eine intensive, vertiefende Beschäftigung mit einem Thema auf die jeweiligen Originalarbeiten zurückgegriffen werden sollte.

Ohne die Mitarbeit der Teilnehmerinnen und Teilnehmer des Seminars wäre dieser Report nicht möglich gewesen. Wir bedanken uns daher bei Frank Hauptmann, Peter Conrad, Christoph Keller, Martin Buch, Philip Ziegler, Frank Leidermann, Martin Kronenburg, Michael Dieterich, Ulrike Becker, Christoph Krome, Susanne Meyfarth, Markus Schmitz, Kenan Çarki, Oliver Schweikart, Michael Schick und Ralf Comes.

Oliver Wendel & Aldo v. Wangenheim

Inhaltsverzeichnis

I Grundlagen und Paradigmen 1

1 Biologische Grundlagen neuronaler Netze 3
- 1.1 Einleitung . . . 3
- 1.2 Bausteine des Nervensystems . . . 3
  - 1.2.1 Der Aufbau einer Zelle . . . 4
  - 1.2.2 Spezialisierung der Neuronen . . . 6
- 1.3 Signalleitung in der Zelle . . . 8
  - 1.3.1 Das Ruhepotential . . . 8
  - 1.3.2 Die elektrotonische Signalleitung . . . 10
  - 1.3.3 Die Aktionspotentiale . . . 11
- 1.4 Signalübertragung an den Synapsen . . . 15
  - 1.4.1 Elektrische Synapsen . . . 15
  - 1.4.2 Chemische Synapsen . . . 16
- 1.5 Zusammenfassung . . . 20
- 1.6 Literatur . . . 21

2 Frühe Anfänge 23
- 2.1 Einleitung . . . 23
- 2.2 Frühe Modelle . . . 23
  - 2.2.1 Modelle der Psychologie . . . 23
  - 2.2.2 Modelle der physischen Struktur . . . 24
- 2.3 Erste Mathematische Modelle . . . 25
  - 2.3.1 William James . . . 25
  - 2.3.2 Pitts & McCulloch . . . 25
  - 2.3.3 Hebb . . . 27
  - 2.3.4 Rosenblatt . . . 29
  - 2.3.5 Block . . . 31
  - 2.3.6 Minsky & Papert . . . 32
- 2.4 Moderne Varianten . . . 33
  - 2.4.1 Backpropagation . . . 33
  - 2.4.2 Hopfield-Netze . . . 34
  - 2.4.3 Probabilistische Netze . . . 34
- 2.5 Zusammenfassung . . . 34
- 2.6 Literatur . . . 34

3 Hopfield Netze und Boltzmann Maschine 37
- 3.1 Einleitung . . . 37
- 3.2 Das Hopfield Modell . . . 37
  - 3.2.1 Struktureller Aufbau . . . 37
  - 3.2.2 Speichern von Mustern . . . 39
  - 3.2.3 Speicherkapazität eines Hopfield Netzes . . . 41
  - 3.2.4 Die Energiefunktion . . . 42

3.2.5 Herleitung der Theorie von einer Energiefunktion . . . 43
3.2.6 Arbeitsweise des Hopfield Netzes . . . 44
3.2.7 Unerwünschte Muster . . . 44
3.2.8 Erweiterungen des Hopfield Modells . . . 44
3.3 Boltzmann Maschine . . . 45
3.3.1 Struktureller Aufbau . . . 45
3.3.2 Arbeitsweise der Boltzmann Maschine . . . 45
3.3.3 Das Lernen der Gewichte . . . 47
3.3.4 Beispiel einer gelernten 7 Segment Anzeige: . . . 51
3.4 Zusammenfassung . . . 52
3.5 Literatur . . . 53

4 Feedforward–Perzeptronen und Backprop 55
4.1 Begriffe und Vorbemerkungen . . . 55
4.1.1 Struktur des Feedforward-Netzes . . . 55
4.1.2 Aufgaben eines Feedforward-Netzes . . . 56
4.1.3 Mathematische Beschreibung der Aufgabe eines Perzeptrons . . . 57
4.2 Simple Perceptrons . . . 57
4.2.1 Schwellwertneuronen . . . 58
4.2.2 Lineare Neuronen . . . 60
4.2.3 Nichtlineare Neuronen . . . 63
4.3 Mehrschichtige Feedforward–Netze . . . 64
4.3.1 Backpropagation — Theoretische Grundlagen . . . 64
4.3.2 Backpropagation–Algorithmus . . . 67
4.3.3 Bemerkungen zum vorgestellten Algorithmus . . . 68
4.3.4 Probleme des BP–Algorithmus, Lokale Minima . . . 69
4.3.5 Konvergenzbeschleunigung . . . 70
4.4 Zusammenfassung . . . 71
4.5 Literatur . . . 71

5 Rekursive Neuronale Netze 73
5.1 Einleitung . . . 73
5.2 Rekursive Neuronale Netze . . . 73
5.2.1 Definition . . . 73
5.2.2 Dynamische Entwicklungsregeln . . . 74
5.2.3 Rekursives Backpropagation . . . 75
5.2.4 Vergleich von Backpropagation in Feedforward- und Rekursiven Netzen . . . 78
5.3 Zeit in Rekursiven Neuronalen Netzen . . . 78
5.3.1 Einführung dynamischer Vorgänge . . . 78
5.3.2 Trainingsmethoden für Zeitfunktionen . . . 79
5.4 Reinforcement Learning . . . 86
5.4.1 Aufgabenstellungen . . . 86
5.4.2 Stochastische Neuronen . . . 86
5.4.3 Trainingsmethoden . . . 87
5.5 Literatur . . . 87

6 ART und Competitive Learning 91
6.1 Einleitung . . . 91
6.2 Einfaches Wettbewerbslernen . . . 91
6.2.1 Grundlagen . . . 91
6.2.2 Geometrische Veranschaulichung . . . 93
6.2.3 Dead units . . . 93
6.3 Beispiele und Anwendungen von Wettbewerbslernen . . . 94
6.3.1 Bipartionierung von Graphen . . . 94

6.3.2 Vektorquantisierung . . . 95
6.3.3 Mehrstufige Netze . . . 96
6.4 Adaptive Resonance Theory(ART) . . . 97
6.4.1 Das Stabilitäts-Plastizitäts-Dilemma . . . 97
6.4.2 Der ART1-Algorithmus . . . 98
6.4.3 Ein Beispiel für den Algorithmus . . . 100
6.4.4 Implementierung des ART1-Netzwerks . . . 102
6.4.5 ART2 und ART3 . . . 103
6.5 Zusammenfassung . . . 104
6.6 Literatur . . . 104

7 Kohonen Netzwerke und Semantische Karten 107
7.1 Motivation und grundlegende Ideen selbstorganisierender Karten . . . 107
7.2 Das Modell von Kohonen . . . 108
7.2.1 Der Algorithmus von Kohonen . . . 109
7.2.2 Ein erstes Beispiel . . . 114
7.3 Anwendung auf das Problem des Handlungsreisenden . . . 116
7.3.1 Eindimensionale Karten . . . 116
7.3.2 Der Weg des Handlungsreisenden als eindimensionale Karte . . . 116
7.4 Semantische Karten . . . 118
7.4.1 Problematik diskreter Symbole . . . 118
7.4.2 Lösungsvorschlag . . . 118
7.4.3 Beispiel einer selbstorganisierenden semantischen Karte . . . 119
7.5 Mathematische Betrachtung des Kohonen-Modells . . . 121
7.5.1 Problematik: "Verborgene Parameter" . . . 121
7.5.2 Faktoranalyse, ein linearer Lösungsansatz . . . 121
7.5.3 Hauptkurven und Kohonen Algorithmus . . . 122
7.6 Schlußbemerkung . . . 123
7.7 Literatur . . . 123

II Anwendungen 125

8 Neuronale Netze in der Robotik 127
8.1 Einleitung . . . 127
8.2 Selbstorganisierende Karten . . . 127
8.2.1 Anpassung von Kohonens Modell an Steuerungsaufgaben in der Robotik – Motorische Karten . . . 127
8.2.2 „Stabbalance-Problem" als technisches Beispiel . . . 128
8.2.3 Okulomotorik als biologisches Beispiel . . . 129
8.3 Anwendungen in der Robotik . . . 131
8.3.1 Probleme der Robotersteuerung . . . 131
8.3.2 Visuomotorische Koordination eines Roboterarms . . . 131
8.3.3 Manipulatorsteuerung mit hierarchischen Netzen . . . 137
8.3.4 Lernen ballistischer Bewegungen . . . 143
8.4 Zusammenfassung . . . 144
8.5 Literatur . . . 145

9 Bildverarbeitung und Mustererkennung 147
9.1 Einleitung . . . 147
9.2 Das Neocognitron-Modell . . . 147
9.3 Lernen und Wiedererkennen von 3D-Objekten . . . 156
9.4 Zusammenfassung . . . 159
9.5 Literatur . . . 161

10 Sprachverarbeitung mit neuronalen Netzen **163**
10.1 Einleitung . . . 163
10.2 Beschreibung des NETtalk - Projektes . . . 164
10.2.1 Aufgabe des Netzwerkes . . . 164
10.2.2 Topologie des Netzes . . . 164
10.2.3 Training von NETtalk . . . 165
10.3 Spracherkennung: Ansatz von P.C. Woodland . . . 165
10.3.1 Aufgabe des Netzwerkes . . . 165
10.3.2 Erstellung der Datenbasis zum Training des neuronalen Netzes . . . 166
10.3.3 Aufbau des Netzwerkes . . . 166
10.3.4 Training des Netzwerkes . . . 166
10.3.5 Leistungen des neuronalen Netzes im Vergleich mit anderen Methoden . . . 167
10.3.6 Anwendung des neuronalen Netzes für eine Telefonansage mithilfe eines Digitalen Signal Prozessors (DSP) . . . 168
10.4 Spracherkennung: Ansatz von S.G. Smyth . . . 168
10.4.1 Aufgabe des Netzwerkes . . . 169
10.4.2 Aufbau des Netzwerkes . . . 169
10.4.3 Training des neuronalen Netzes . . . 169
10.4.4 Zusammensetzung der Subwords durch "Dynamisches Programmieren" . . . 170
10.4.5 Resegmentierung der Trainingsdaten . . . 170
10.4.6 Beispiel für die Anwendung von Dynamischem Programmieren . . . 170
10.4.7 Erweiterung auf fließende Sprache . . . 171
10.5 Zusammenfassung . . . 171
10.6 Literatur . . . 172

11 Neuronale sematische Netze **175**
11.1 Einleitung . . . 175
11.2 Was sind und was können semantische Netze? . . . 175
11.2.1 Darstellung semantischer Netze . . . 175
11.2.2 Interpreter und Anfragen . . . 176
11.3 Aufbau eines statischen, neuronalen Netzes . . . 176
11.3.1 Prinzipielle Darstellung der semantischen Netze durch neuronale Netze . . . 176
11.3.2 Ein konkretes neuronales Netz . . . 178
11.4 Das Lernen eines semantischen, neuronalen Netzes . . . 181
11.4.1 Vorgehensweise und Ziel dieses Ansatzes . . . 182
11.4.2 Architektur des Netzes . . . 182
11.4.3 Das Lernen . . . 185
11.4.4 Ein Beispiel . . . 186
11.5 Zusammenfassung . . . 188
11.6 Literatur . . . 189

12 Neuronale Netze in Hybriden Systemen **191**
12.1 Einleitung . . . 191
12.2 Bekannte Ansätze hybrider Systeme [Goo] . . . 193
12.3 Symbolic Connectionist Net: SC-Net . . . 193
12.3.1 Einführung von Symbolverarbeitung in ein connectionistisches Modell . . . 193
12.3.2 Übersetzung von Regeln in ein SC-Netz . . . 194
12.3.3 Lernen und Modifizieren . . . 195
12.3.4 Implementierung von Variablen . . . 195
12.3.5 Fuzzy und relationale Operatoren . . . 195
12.3.6 Fazit für SC-Netze . . . 196
12.4 *echte* hybride Systeme . . . 196
12.4.1 HSHSP - Hybrides System für High School Physics . . . 196
12.4.2 NESSY - Neural Network and Symbolic Reasoning System . . . 198

12.5 Fazit für Hybride Systeme 200
12.6 Literatur 200

13 Wissensbasierte neuronale Netze **205**
13.1 Einleitung 205
13.2 KBANN-Knowledge-Based Artificial Neural Networks 205
13.2.1 Einfügung von Wissen in ein neuronales Netz 206
13.2.2 Die Verbesserung der KBANN-Netze 209
13.2.3 Die Extraktion von Regeln aus einem KBANN-Netz 209
13.3 Anmerkungen zu KBANN 212
13.4 Zusammenfassung 213
13.5 Literatur 213

14 Modellierung biologischer Systeme **219**
14.1 Einleitung 219
14.1.1 Motivation: Warum und wofür ? 219
14.1.2 Wege zu einem Modell 220
14.2 Dendritenmodell 220
14.2.1 Kabeltheorie 220
14.2.2 Compartmental Model 225
14.3 Simulatoren 226
14.4 Zusammenfassung 228
14.5 Literatur 228

Teil I

Grundlagen und Paradigmen

Kapitel 1

Biologische Grundlagen neuronaler Netze

Frank Hauptmann

Übersicht

Die Beschreibung der biologischen Grundlagen neuronaler Netzwerke beginnt mit einem kurzen Überblick über die Bestandteile und die Funktionsweise einer Zelle. Danach werden die Eigenschaften beschrieben, die eine Nervenzelle gegenüber den anderen Zellen besonders auszeichnen. Da Neuronen elektrische Signale leiten müssen, folgt nun ein Einblick über die elektrischen Eigenschaften der Zellmembran, bevor mit dem Aktionspotential die effektivste Form der Signalleitung beschrieben wird. Den Abschluß bildet eine Beschreibung der Vorgänge, die an den Übergangsstellen zu anderen Neuronen, den Synapsen, vor sich gehen.

1.1 Einleitung

Neuronale Netzwerke haben sich in der Natur so gut bewährt, daß man nun beginnt, diese für Anwendungen auf digitalen Rechnern nutzbar zu machen. Zu diesem Zweck wird versucht, solche Netzwerke in der Natur möglichst genau zu untersuchen und zu verstehen, um die Funktionsweise in allen Einzelheiten zu erfassen. Im folgenden werden die bisher erzielten Erkenntnisse beschrieben, wobei versucht wurde, auf komplizierte biologische, chemische und biochemische Feinheiten zu verzichten.

In der Natur sind neuronale Netzwerke immer identisch mit den Begriffen ‚Gehirn' und ‚Nervensystem', womit sich auch schon der Inhalt der nun folgenden Seiten festlegen läßt.

1.2 Bausteine des Nervensystems

Unser Nervensystem (wie übrigens auch das aller anderen Tiere) besteht aus einzelnen Zellen, die über verschiedene Fortsätze miteinander verbunden sind. Lediglich die Anzahl dieser Nervenzellen (Neuronen) ist bei den verschiedenen Tieren verschieden, ebenso wie die Organisation der Neuronen zu übergeordneten Einheiten (z. B. Gehirn). So ist das menschliche Gehirn mit seinen 100 Milliarden Neuronen – das entspricht etwa der Anzahl der Sterne in der Milchstraße – mit Sicherheit wesentlich komplexer aufgebaut als das der Meeresschnecke *Aplysia* mit seinen nur ca. 20 000 Neuronen – die man dafür aber auch alle kennt. Ob der Mensch allein dadurch schon als ‚schlauer' eingestuft werden kann, sei hier einmal dahingestellt.

Bevor die einzelnen Bestandteile des Nervensystems und ihre Funktionsweise näher betrachtet werden, folgt nun ein kurzer Exkurs in die Biologie, um den grundsätzlichen Aufbau aller Zellen zu beleuchten.

1.2.1 Der Aufbau einer Zelle

(Dieser Abschnitt wurde in großen Teilen nach [1] gestaltet.)
In der folgenden Abbildung ist eine idealisierte Zelle dargestellt, die die allen höheren Zellen gemeinsamen Organellen zeigt.

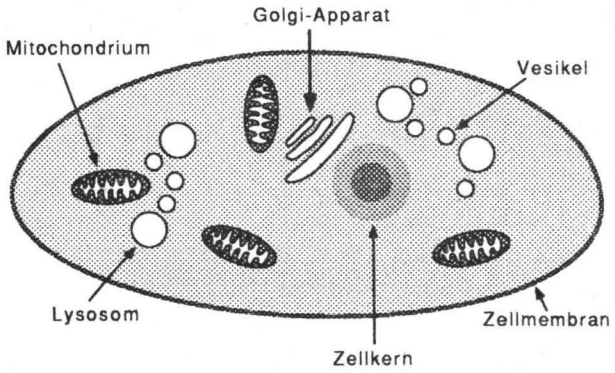


Abbildung 1.1: Grundbauplan einer Zelle

Jede Zelle enthält einen **Zellkern**, in dem die **Erbinformation**, kodiert in der **DNS** (*Desoxyribonucleinsäure*), abgelegt sind. In jeder Zelle eines Lebewesens ist die gleiche Erbinformation enthalten, auch wenn die Zellen sich noch so sehr unterscheiden. Teilt sich die Zelle, wird die DNS kopiert und diese Kopie in die neu entstandene Zelle übernommen.

Die **Lysosomen**, von denen es über hundert geben kann, stellen gewissermaßen die ‚Mägen' der Zelle dar. Hier finden der Abbau der von der Zelle aufgenommen Nährstoffe sowie die Trennung von für die Zelle verwendbarem Material von unbrauchbarem statt. Aber nicht nur von außen eingeführte Teilchen wie Proteine werden hier zerlegt, auch die Innereien der Zelle selbst verschwinden nach einigen Tagen in den Lysosomen und werden hier zerlegt. Dieser *Autophagie* genannte Vorgang sichert, daß durch äußere Einflüsse beschädigte Organellen nach einiger Zeit ersetzt werden. Autophagie kann aber auch lebensrettend sein, da ein Organismus bei Nahrungsentzug zuerst nicht unbedingt notwendige Teile seiner eigenen Zellen abbaut. Zu den wiederverwendbaren Bausteinen gehören z. B. die Moleküle der Zellmembran, die dieser auch wieder zugeführt werden, um andere zu ersetzen. Aber auch die hier gewonnen Verdauungsprodukte verbleiben in der Zelle, lediglich die unverwertbaren Teile werden von der Zelle ausgeschieden.

Die Zelle besitzt zur Nahrungsaufnahme und zur Ausscheidung ein ausgefeiltes Transportsystem, denn die Moleküle werden in kleinen Bläschen, sogenannten **Vesikeln** aufbewahrt, während sie das Zellinnere durchqueren. Diese Vesikel entlassen ihren Inhalt in das Lysosom oder aus der Zelle, indem sie mit der entsprechenden Membran verschmelzen, und sie entnehmen ihren Inhalt aus den Lysosomen oder dem extrazellulären Raum, indem sie sich aus der jeweiligen Membran herauslösen. Die Nahrungsaufnahme an der Zellmembran wird übrigens **Endocytose**, die Ausscheidung **Exocytose** genannt.

Die Produktion der für die Verdauung in den Lysosomen erforderlichen Enzyme[1] erfolgt über

[1] Enzyme sind so etwas wie Katalysatoren in der Chemie; sie beschleunigen einen Reaktionsprozeß und gehen anschließend unverändert wieder daraus hervor. Daher heißen sie auch ‚Biokatalysatoren'.

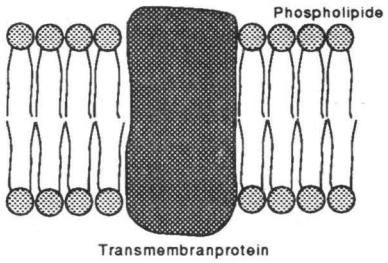


Abbildung 1.2: Ausschnitt aus der Zellmembran

den **Golgi-Apparat**, wo diese produziert und in Vesikeln den Lysosomen zugeführt werden.

Die **Mitochondrien** stellen das Kraftwerk der Zellen dar, hier wird durch *Oxydation* („Verbrennung“[2]) energiereicher organischer Stoffe wie Fette und Kohlenhydrate **ATP** (*Adenosintriphosphat*), der eigentliche Energielieferant für fast alle Reaktionen, die in einer Zelle ablaufen, gewonnen. Wird ATP unter Abspaltung einer Phosphatgruppe in ADP (*Adenosindiphosphat*) umgewandelt, wird genug Energie frei, um alle anderen Zellprozesse durchzuführen, sei es eine Ionenpumpe an einer Membran in Aktion zu halten oder die Geißel eines Einzellers zu Bewegungen zu veranlassen, um diesem die Fortbewegung zu ermöglichen.

Alle diese Organellen sind von einer Haut, einer sogenannten **Membran**, umgeben. Da dieser auch bei den Neuronen eine besondere Funktion zukommt, sei sie hier genauer beschrieben. Eine Membran besteht zunächst einmal aus einer sogenannten **Lipid-Doppelschicht**. Lipide sind lange Moleküle, die sich durch eine besondere Eigenschaft auszeichnen: Sie haben an einem Ende einen großen hydrophilen [3] Kopf mit einer negativ geladenen Phosphorsäure-Gruppe, die noch mit einer anderen, oft positiv geladenen Gruppe verknüpft ist. Von diesem Kopf gehen zwei hydrophobe [4] Fettsäure-Ketten aus. Über elektrostatische Anziehungskräfte (sogenannte Van-der-Vaals-Kräfte) kann die stark geladene Seite mit der intra- und der extrazellulären Flüssigkeit in Wechselwirkung treten, während sich die Moleküle mit den hydrophoben Fettsäure-Ketten aneinanderlagern. Auf diese Weise ist eine für geladene wie ungeladene Moleküle fast vollkommen undurchlässige Membran entstanden.

Die Membran sorgt für die Aufrechterhaltung der verschiedenen Ionenkonzentrationen, die in der intra- und extrazellulären Flüssigkeit vorliegen. Eine Möglichkeit, Moleküle in das Innere der Zelle zu transportieren, sind die Vesikeln. Aber in der Membran eingelagerte Proteine können ebenfalls zum Austausch von Ionen und Molekülen zwischen beiden Seiten der Membran dienen.

Hier wären zunächst einmal die **Ionenkanäle** zu nennen. Hierbei handelt es sich um spezielle Transmembranproteine, die einen Kanal durch die Membran legen. Meist ist ein solcher Kanal hochselektiv, d. h. er läßt nur eine Ionensorte durch. Z. B. läßt ein K^+-Kanal fast ausschließlich Kalium-Ionen passieren, während er für Natrium-Ionen fast undurchlässig ist. Im folgenden spielen Kalium- und Natrium-Kanäle bei der Betrachtung der Neuronen die wohl wichtigste Rolle.

Da für die meisten Ionen solche Kanäle in der Membran vorhanden sind, würde der osmotische Druck nach einiger Zeit zum Ausgleich der unterschiedlichen Ionenkonzentrationen auf beiden Seiten der Zellmembran führen. Um nun für die durch die Ionenkanäle diffundierenden (wandernden) Ionen einen Ausgleich zu schaffen, sind in der Membran auch sogenannte **Ionenpumpen** eingelagert. Diese sorgen für die Aufrechterhaltung der unterschiedlichen Ionenkonzentrationen, indem

[2] Verbrennung ist ein äußerst unzutreffender Ausdruck dieser Vorgänge, da mit diesem Begriff immer eine entsprechende Hitze assoziiert wird – eine Zelle ist allerdings kein Ofen oder etwas ähnliches!

[3] hydrophil: wasserliebend

[4] hydophob: wasserabweisend

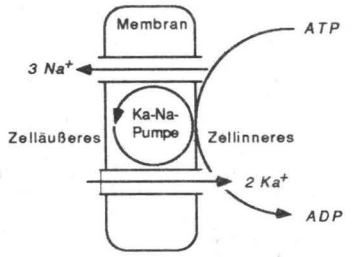


Abbildung 1.3: Schema der Natrium-Kalium-Ionenpumpe

sie unter Verwendung von ATP Ionen durch die Zellmembran befördern. So transportiert beispielsweise die Na^+-K^+-Ionenpumpe unter Verbrauch eines ATP-Moleküls jeweils drei Natrium-Ionen aus der Zelle und zwei Kalium-Ionen hinein.

Eine weitere Art von Transmembranproteinen sind die Rezeptoren. Dabei handelt es sich um Moleküle, die eine spezielle ‚Andockstelle' für bestimmte Moleküle haben. Lagert sich ein solches Molekül an seinen zugehörigen Rezeptor an, können im Innern der Zelle bestimmte Funktionsabläufe in Gang gesetzt werden. So gibt es spezielle Ionenkanäle, die mit einem Rezeptor gekoppelt und nur dann durchlässig sind, wenn sich das entsprechende Molekül an den Rezeptor angelagert hat. Ansonsten sind diese Kanäle undurchlässig.

Neuronen sind nun besonders spezialisierte Zellen, und dementsprechend ausgerüstet. Was Neuronen von anderen Zellen unterscheidet, ist Thema das folgenden Abschnitts.

1.2.2 Spezialisierung der Neuronen

Neuronen unterscheiden sich in einer ganz bestimmten Hinsicht von anderen Zellen: Sie verarbeiten Informationen, und dazu sind sie mit ganz speziellen Anpassungen versehen. So gehen von ihrem Zellkörper verschiedene Nervenfortsätze aus, von denen meist einer besonders lang ist. Dieser besondere Ast heißt auch **Axon**, während sich für die anderen, meist wesentlich kürzeren Fortsätze der Begriff **‚Dendriten'** eingebürgert hat. Da es auch Neuronen ohne oder mit sehr kurzem Axon gibt, faßt man oft alle Fortsätze unter der Bezeichnung **‚Neuriten'** zusammen. Früher hat man angenommen, daß die Signale an den Dendriten zum Neuron gelangen und dieses ausschließlich über das Axon wieder verlassen – diese Ansicht wurde allerdings durch neuere Forschungen widerlegt. Signalübergänge sind bekannt von Axon auf Dendrite, Zellkörper und Axone, sowie von Dendriten auf Dendrite, Axone und Zellkörper. Es gibt sogar Hinweise für eine Übertragung direkt von Zellkörper auf Zellkörper. Axone können übrigens beachtliche Längen erreichen. So entsenden die großen Pyramidenzellen beim Menschen, deren Zellkörper in der Großhirnrinde liegen, ihre Axone 1,20 Meter tief hinunter in das Rückenmark. Bei der Giraffe sind die entsprechenden Axone immerhin 4,50 Meter lang.

Die Übertragung der Signale auf eine andere Nervenzelle geschieht an speziellen Kontaktstellen, den **Synapsen**. Meist hat ein Neuron zwischen 100 und 10000 Synapsen. Diese sitzen allerdings nicht direkt an der Zellmembran das postsynaptischen Neurons, stattdessen klafft eine 0,02 Mikrometer breite Lücke, der **synaptische Spalt**. Daher kann ein Signal auch nicht direkt an die nächste Zelle weitergegeben werden. An elektrischen Synapsen sorgt ein besonders großes Protein für die Übertragung des elektrischen Potentials, während an chemischen Synapsen für diesen Zweck spezielle Botenstoffe, sogenannte Neurotransmitter, freigesetzt werden, die sich an Rezeptoren der postsynaptischen Zellmembran heften und dort für spezielle Funktionen sorgen können, die meist wieder ein elektrisches Potential in dieser Zelle zur Folge haben.

Der neuronale Zellkörper hat gegenüber einer normalen Zelle wesentlich höhere Versorgungsleistungen zu erbringen, er muß nicht nur für den gesamten Zellkörper mitsamt seiner Neuriten

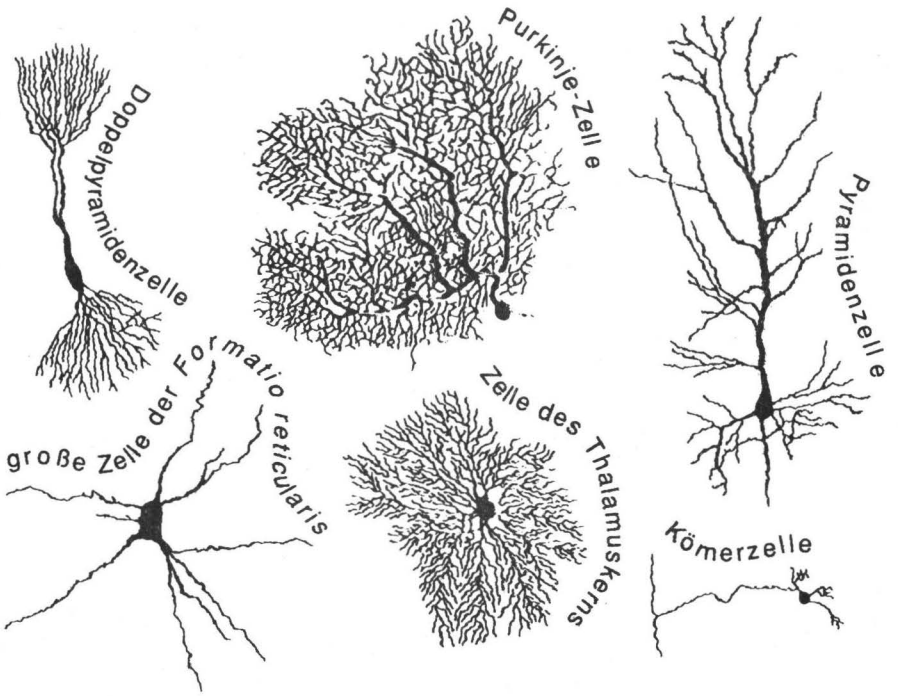


Abbildung 1.4: verschiedene Neuronentypen (aus [3])

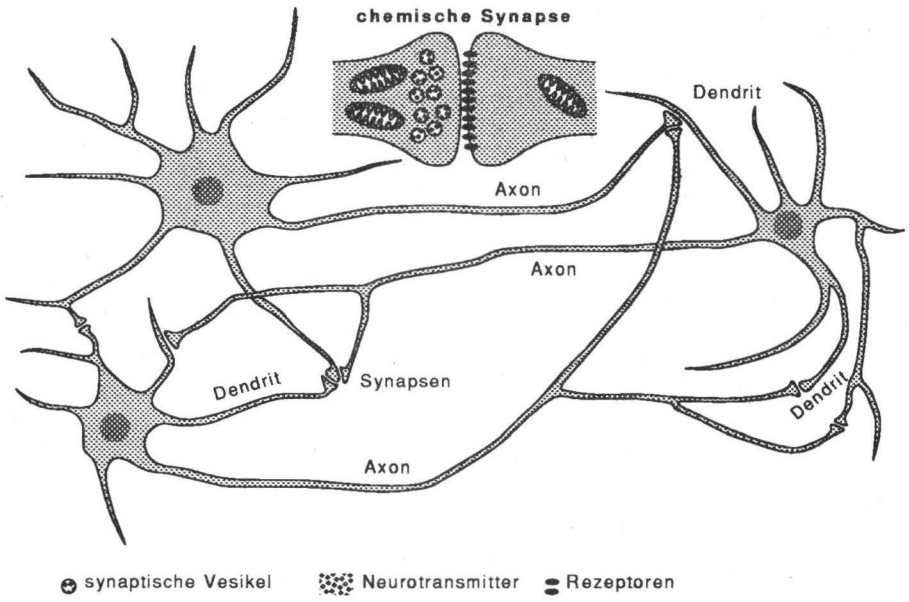


Abbildung 1.5: Schema der Vernetzung der Neuronen

	Intrazelluläre Konzentration ($mmol$)	Extrazelluläre Konzentration ($mmol$)
K^+	124	2
Na^+	10	145
Ca^{2+}	5	2
Cl^-	2	77
A^-	74	13

Tabelle 1.1: Ionenkonzentrationen

die Membranproteine produzieren und diese an die vorgesehenen Stellen transportieren, sondern er ist im allgemeinen auch für die Produktion der Neurotransmitter zuständig. Und auch die Abbauprodukte müssen über einen weiten Bereich zum Zellkörper, der im Durchschnitt nur $^1/_{10}$ des Volumens der gesamten Zelle ausmacht, befördert werden. So ist es denn auch nicht weiter verwunderlich, daß ein großer Teil der Stoffwechselenergie im Körper zur Aufrechterhaltung der Funktion des Nervensystems dient. Außerdem werden im Nervensystem mehr genetische Informationen genutzt als an sonst irgendeiner Stelle des Körpers.

Aber die weitreichendsten Anpassungen an die Aufgaben der Neuronen findet man in deren Zellmembran. Da Neuronen elektrische Reize oder Potentiale weiterleiten müssen, ist die Membran für diese Aufgabe mit zusätzlichen Strukturelementen wie Ionenpumpen und -kanälen ausgestattet.

1.3 Signalleitung in der Zelle

1.3.1 Das Ruhepotential

Wie bei allen anderen Zellen auch sind in einem Neuron die Ionenkonzentrationen auf beiden Seiten der Zellmembran verschieden groß. Ist die Zelle gerade inaktiv, d. h. sie ist nicht gerade für die Signalübertragung herangezogen, kann man ein sogenanntes **Ruhepotential** messen, indem man mit einer Mikroelektrode die Zellmembran durchsticht und das dort gemessene Potential mit dem in der extrazellulären Flüssigkeit gemessenen in Beziehung setzt. Für ein in Ruhe befindliches Neuron mißt man in diesem Fall ein negatives Potential, welches meist zwischen -50 und $-80mV$ liegt. In der folgenden Tabelle sind die Konzentrationen für verschiedene Ionen innerhalb und außerhalb der Zelle aufgeführt.

An der Entstehung dieser Konzentrationsunterschiede sind wieder die bereits beschriebenen Ionenpumpen beteiligt. Zur Aufrechterhaltung des Natrium- Kalium-Ionengefälles beiderseits der Zellmembran dient unter anderem die bereits beschriebene Na^+-K^+-Ionenpumpe. Um diese Ionenpumpen in Gang zu halten, benötigt eine Nervenzelle ca. 70 % der gesamten ihr zur Verfügung stehenden Energie.

In diesen Konzentrationsgradienten der verschiedenen Ionen beiderseits der Membran verbirgt sich leicht zugängliche Energie, die insbesondere der Erzeugung und Erhaltung der elektrischen Potentialdifferenz zwischen beiden Seiten der Membran dient. Zu diesem Zweck sind die unterschiedlichen Ionenkanäle in die Membran eingelagert. Normalerweise ist eine Lipid- Doppelschicht für Ionen fast undurchlässig, da die elektrisch geladenen Ionen nur sehr schwer durch die stark hydrophoben Lipidschichten diffundieren können. Daher sind zur Beschleunigung der Diffusion durch die Membran *Ionenkanäle* in derselben eingebettet. Bei diesen handelt es sich um röhrenförmige Polypeptide[5], die durch die Membran hindurchreichen und so in der Membran eine Pore bilden, die die Ionen passieren können.

[5] Polypeptide sind lange Molekülketten aus Aminosäuren, das sind Moleküle mit der allgemeinen Strukturformel $NH_2 - CHR - COOH$, wobei R eine unter mehreren möglichen Seitengruppen ist; Aminosäuren stellen auch die Bausteine der Proteine dar

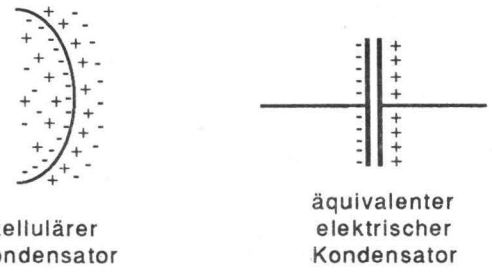


Abbildung 1.6: Membran als Kondensator (aus [4])

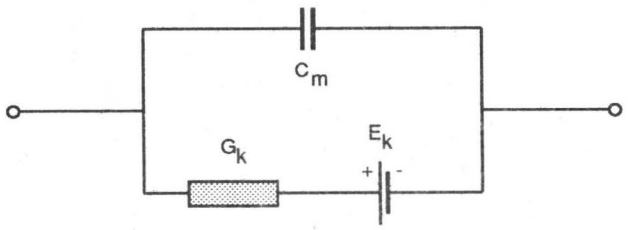
Fast alle Ionenkanäle arbeiten selektiv, d. h. sie lassen nur eine Sorte an Ionen durch die Pore wandern, zum einen abhängig von der Größe der Pore, zum anderen abhängig von der Ladung der Moleküle. Denn der Durchmesser der Pore ist recht klein und beträgt an den engsten Stellen nur wenige Ångström. Daher werden größere Ionen und Moleküle an der Passage gehindert, gewissermaßen ‚ausgesiebt'. Die Polypeptidketten des Ionenkanals weisen zudem oft elektrostatisch geladene Bereiche auf, durch die je nach Ladung positive oder negative Ionen zurückgehalten werden. Zwar ist kein Kanal vollkommen selektiv und läßt auch Ionen passieren, für die er eigentlich undurchlässig sein sollte, aber deren Anzahl ist vernachlässigbar gering. So läßt der Kalium-Kanal 100mal mehr K^+-Ionen passieren als Na^+-Ionen. Die Durchlaßrate solcher Ionenkanäle ist recht beachtlich und beträgt meist mehr als 10^6 Ionen pro Sekunde pro Kanal. Da jedes Neuron viele solcher Ionenkanäle besitzt, fließt so ein beträchtlicher Ionenstrom durch die Membran.

Aufgrund des Konzentrationsgefälles, welches durch die Ionenpumpen aufrecht erhalten wird, fließen, bedingt durch den osmotischen Druck, immer Ionen durch diese Ionenkanäle, wodurch elektrische Ladungen auf die jeweils andere Seite der Membran transportiert werden. Betrachten wir zum Verständnis im folgenden eine Membran mit Ionenkanälen, die ausschließlich von Kalium-Ionen passiert werden können. Getrieben durch die höhere Konzentration der Ionen innen diffundieren Kalium-Ionen durch die Kanäle an die Außenseite der Zellmembran. Da allerdings mit jedem Ion, welches die Zelle verläßt, auch eine positive Ladung von innen nach außen gelangt, für die kein Ausgleich geschaffen werden kann (die Membran ist ja nur für Kalium-Ionen durchlässig), findet so eine Aufladung des Äußeren gegenüber dem Zellinneren statt. Die dadurch entstehende Potentialdifferenz steht aber einer weiteren Diffusion von Kalium-Ionen durch die Membran nach außen entgegen, so daß sich an der hypothetischen Membran bald ein *Gleichgewichtszustand* herausbildet, der durch ein gleichbleibendes *Gleichgewichtspotential* gekennzeichnet ist. In diesem Zustand gelangen gleich viele K^+-Ionen von innen nach außen wie von außen nach innen. Die in diesem Zustand vorliegende Potentialdifferenz kann berechnet werden und beträgt $-58mV$.

Da auf beiden Seiten einer Zellmembran nun unterschiedliche Ladungen vorhanden sind, die durch eine dünne Isolierschicht getrennt sind (nämlich die Membran), kann man diese auch als Kondensator auffassen.

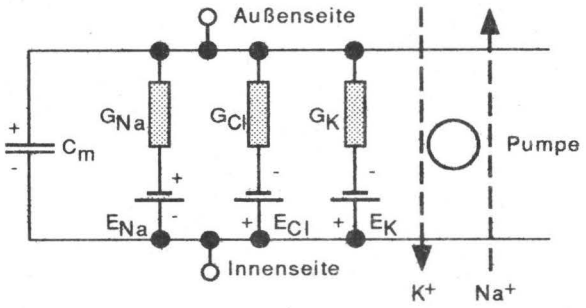
Da die Zellmembran durch die Ionenkanäle für Ladungen durchlässig ist, hat sie auch einen gewissen Widerstand G_k, so daß man das elektrische Verhalten einer Zellmembran auch durch ein Ersatzschaltbild darstellen kann.

Dabei stellt C_m die Kapazität der Membran dar, während das Gleichgewichtspotential E_k als Spannungsquelle dient. In einer realen Zelle sind allerdings nicht nur Kanäle für Kalium-Ionen eingelagert, sondern auch für Chlor-, Natrium-, Calcium- und einige weitere Ionen. Daher wechseln auch mehrere Ionen permantent die Seiten der Membran, bis sich das Gleichgewicht einstellt, welches als *Ruhepotential* des Neurons bezeichnet wird. Es handelt sich dabei allerdings nicht um ein Gleichgewichtspotential wie bei der nur mit Kalium-Kanälen ausgerüsteten Membran, sondern um ein *Fließgleichgewicht*, da die abwandernden Ionen zur Aufrechterhaltung des Potentials mittels



E_k Spannungsquelle, Spannung entspricht Gleichgewichtspotential
G_k Gesamtleitfähigkeit der Membran für K^+
C_m Kapazität der Membran

Abbildung 1.7: Ersatzschaltbild für Membran mit Kalium-Kanälen (aus [4])



E_i Spannungsquelle, Spannung entspricht Gleichgewichtspotential
G_i Gesamtleitfähigkeit der Membran für das entsprechende Ion
C_m Kapazität der Membran

Abbildung 1.8: Ersatzschaltbild für Fließgleichgewicht (aus [4])

Ionenpumpen permanent ersetzt werden müssen, weil sonst die Konzentrationen der Ionen nach einiger Zeit ausgeglichen werden würden. In dem dazugehörenden Ersatzschaltbild ist daher die Kalium-Natrium-Pumpe zur Aufrechterhaltung der Konzentrationen mit berücksichtigt.

1.3.2 Die elektrotonische Signalleitung

Die elektrischen Signale der Neuronen sind zeitliche Änderungen im Potential der Zellmembran, die als Folge von Ionenströmen in die Zelle hinein oder aus der Zelle hinaus entstehen. Sie entstehen entweder in speziellen Rezeptorzellen wie z. B. Sinneszellen oder sie werden über Synapsen auf das Neuron übertragen. An diesen Synapsen können die Signale **inhibitorisch**, d. h. hemmend, oder **exhibitorisch**, d. h. verstärkend, wirken. Diese **elektrotonischen** Signale breiten sich dabei passiv und ohne Verstärkungsmechanismen entlang der Zellmembran aus, wobei der Signalverlauf durch die elektrischen Widerstands- und Kapazitätseigenschaften der Zellmembran verlangsamt und in der Amplitude deutlich abgeschwächt wird. Diese Form der Signalleitung ist mithin also nicht verzerrungsfrei.

Auch mit der Leitung in metallischen Leitern wie Stromkabeln läßt sich die Leitung entlang der Zellmembran nur bedingt vergleichen, da sie sich in wesentlichen Punkten unterscheiden. Zunächst

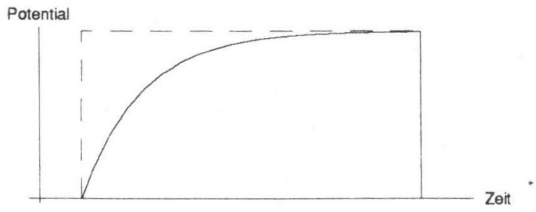


Abbildung 1.9: Verzerrung eines Signals durch die Membran

einmal ist die Leitfähigkeit in der Zellmembran etwa 10^7-fach geringer als die eines metallischen Leiters, denn in der Membran dienen Ionen zur Übertragung von Strömen, und die sind größer und damit unbeweglicher als die Elektronen in einem Metallkabel. Außerdem liegen sie in geringerer Konzentration vor als die vergleichbaren Elektronen. Dann ist der Stromfluß wesentlich verlustreicher als in einem Kabel, denn die Membran der Nervenzelle ist ein ziemlich schlechter Isolator (man denke an die zahlreichen Ionenkanäle, durch die ein Teil der Ionen hindurchtritt und damit die Potentialdifferenz ausgleicht). Zusammen mit der großen Kapazität der Membran tritt also eine eklatante Abschwächung des Signals ein, denn zur Umladung der Kapazität werden vom sich ausbreitenden Ionenstrom zusätzliche Ladungen abgezweigt. Vor allem schnelle Signale schwächen sich so sehr schnell ab.

Zum besseren Verständnis ziehen wir wieder das obige Ersatzschaltbild heran. Widerstand und Kondensator liegen im Ersatzschaltbild parallelgeschaltet vor, weshalb Spannungsänderungen auf beiden Seiten der Bauelemente zu jeder Zeit gleich sein müssen. Wird nun ein rechteckiger, konstanter Stromimpuls angelegt, dient dieser zunächst zum Umladen des Kondensators. Mit dem Umladen wird aber aufgrund der auftretenden Spannungsänderung ein Teil des Stroms über den Widerstand abfließen, der nicht mehr für den Umladevorgang zur Verfügung steht. Und je mehr Strom über den Widerstand abfließt, desto weniger steht zum Umladen des Kondensators zur Verfügung. Schließlich fließt der gesamte Strom über den Widerstand ab. Daher wird ein Signalpuls beim Fluß entlang der Zellmembran nicht unerheblich verzerrt.

Aber auch die Reichweite des Impulses ist auf wenige Millimeter begrenzt, da ein Teil des Stroms durch die Ionenkanäle verlorengeht. Dieser Verlust nimmt exponentiell zur zurückgelegten Entfernung zu, da auf jeder Strecke entlang der Membran ein etwa gleicher Anteil an Ladungen für die weitere Signalausbreitung nicht mehr zur Verfügung steht, abhängig von der Anzahl der Ionenkanäle auf dieser Strecke.

Solange nun ein Neuron klein ist und die zu übertragenden Signale langsam sind, reicht die elektrotonische Signalleitung aus, und man findet Neuronen, die auf dieser Basis ihre Signale verarbeiten, z. B. in den Horizontalzellen und den Bipolarzellen. Allerdings ist diese Art der Signalleitung schon dann unbrauchbar, wenn es um die schnelle Informationsverarbeitung geht. So können bei einem Tier, welches auf der Flucht vor einem Räuber ist, schon Sekundenbruchteile über dessen Überleben entscheiden. Auch ist aufgrund der Abschwächungseigenschaften der elektrotonischen Signalleitung diese nicht für längere Wege wie die oben erwähnten Axone der Pyramidenzellen geeignet. Und so gibt es im Nervensystem auch noch eine wesentlich effektivere Möglichkeit, Impulse weiterzuleiten, die zudem noch schneller, über weite Entfernungen konstant und eindeutig als An-oder-Aus-Impulse übertragen werden. Diese **Aktionspotentiale** werden im folgenden beschrieben.

1.3.3 Die Aktionspotentiale

Neurone, die ihre Signalpulse schneller befördern können, sind mit bestimmten Mechanismen ausgestattet, die eine sehr schnelle Depolarisierung und Repolarisierung der Membran gewährleisten. In weniger als einer Millisekunde kann vom negativen Ruhepotential ausgehend der Nullwert er-

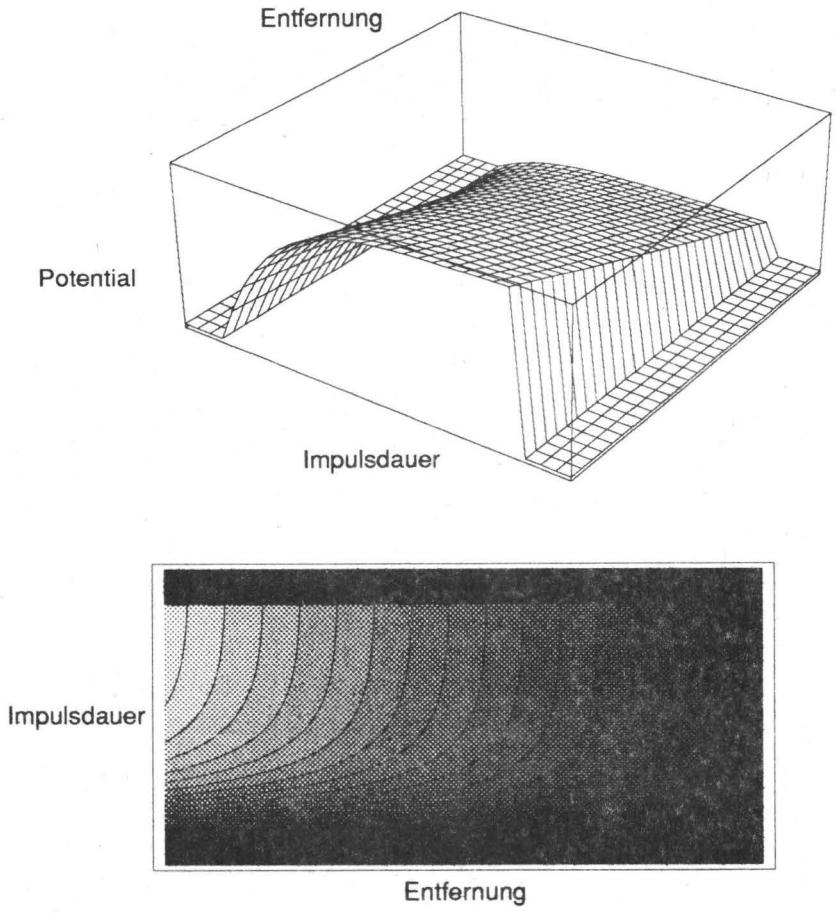


Abbildung 1.10: Abschwächung eines Signals durch die Membran

reicht und meist auch überschritten und in ebenso kurzer Zeit wieder erreicht werden. Diese *Aktionspotentiale* breiten sich ohne Verzerrung und Abschwächung auch entlang eines sehr langen Nervenfortsatzes aus. Um diese Leistungen zu vollbringen, ist die Membran mit speziellen *spannungsabhängigen Ionenkanälen* ausrüstet. Dabei handelt es sich um Transmembranproteine, die verschiedene quasistabile Zustände einnehmen und bei einer bestimmten Potentialänderung an der Zellmembran zwischen diesen hin- und herschalten können, um z. B. den Kanal für bestimmte Ionensorten zu öffnen oder zu schließen. Ein solcher Ionenkanal kann aber auch durch die zellinterne Ionenkonzentration beeinflußt werden.

Meist besitzt ein solcher Kanal drei Zustände, die in einer bestimmten Reihenfolge durchlaufen werden. Zu Beginn befindet sich der Kanal im Zustand *geschlossen*, d. h. er ist für Ionen undurchlässig. Durch das Aktionspotential wird er in den Zustand *offen* versetzt und somit für Ionen geöffnet. Nach kurzer Zeit geht er automatisch in den Zustand *inaktiviert* über, in dem er einige Zeit verbleibt, bevor er den Zustand *offen* wieder erreichen kann. Für das grundlegende Verständnis des Aktionspotentials ist die Kenntnis der Funktionsweise nur zweier Ionenkanäle, des spannungsabhängigen Kalium-Kanal und des spannungsabhängigen Natrium-Kanals, ausreichend.

Befindet sich die Zelle im Ruhezustand, d. h. liegt das Ruhepotential am Kanalprotein an, ist der Na^+-Kanal im Zustand geschlossen, da diese Form dann energetisch am stabilsten ist. Wird das Neuron depolarisiert, geht der Kanal von der geschlossenen in die offene Form über und läßt

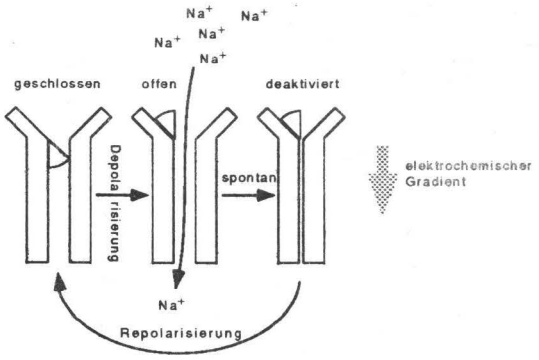


Abbildung 1.11: Schema eines spannungsabhängigen Ionenkanals (aus [4])

in selektiver Weise fast ausschließlich Na^+-Ionen passieren, abhängig vom osmotischen Druck auf beiden Seiten der Membran. Damit diffundieren überdurchschnittlich viele Natrium-Ionen in das Innere der Zelle. Nach kurzer Zeit jedoch geht der Kanal spontan in den deaktivierten Zustand über, indem er für Ionen undurchlässig ist. Von der geschlossenen Konfiguration unterscheidet sich der deaktivierte Kanal dadurch, daß er nicht ohne weiteres wieder direkt in die offene Form übergehen kann, sondern zunächst in die geschlossene Konformation versetzt werden muß. Dies geschieht im allgemeinen erst dann, wenn das Potential wieder Werte in der Nähe des Ruhepotentials erreicht hat.

In den für die Ausbildung eines Aktionspotentials relevanten Membranbereichen liegen besonders viele Na^+-Kanäle (zwischen 35 und 500 pro Quadratmikrometer). Einige dieser Kanäle werden bei einer ausreichenden Depolarisierung der Membran in die offene Konformation übergehen, wodurch Natrium-Ionen wegen des osmotischen Drucks in die Zelle einströmen. Bevor der jeweilige Kanal wieder in die inaktive Phase übergeht, ist ein minimaler Strom im Pikoamperebereich geflossen. Durch die hohe Kanaldichte in diesen Bereichen reicht der insgesamt durch alle bisher offenen Kanäle eingeflossene Strom für die Öffnung weiterer geschlossener Kanäle aus, so daß die Membran weiter depolarisiert wird, wodurch weitere Natrium-Ionen für eine weitere Depolarisation sorgen...

Innerhalb von weniger als einer Millisekunde gehen durch diesen *selbstverstärkenden Vorgang* die meisten Na^+-Kanäle in ihren offenen Zustand über, wodurch sich die Permeabilität der Membran für Na^+- Ionen gegenüber der für K^+-Ionen drastisch erhöht. Da sich die Kanäle durch den Übergang in die deaktivierte Phase kurz schließen, ist der maximal fließende Na^+-Einstrom begrenzt, wodurch das Potential Werte von ca. $+40mV$ erreichen kann. Ist der maximale Wert erreicht, fällt das Potential nach ca. $2ms$ wieder auf den Wert des Ruhepotentials zurück.

Allerdings ist für das Auslösen eines Aktionspotentials das Überschreiten eines bestimmten Schwellenwertes notwendig, unterhalb dieses Wertes kann sich kein Aktionspotential ausbilden, da nur wenige Natrium- Kanäle in den offenen Zustand übergehen. Weil nur wenige Natrium-Ionen so für eine Depolarisierung sorgen, andererseits immer eine Ruheleitfähigkeit für Kalium-Ionen vorhanden ist, kann diese Depolarisierung durch den Ausstrom von K^+-Ionen abgefangen werden. Daher bewegt sich das Potential wieder auf das Ruhepotential zu, ohne den Schwellenwert, der bei etwa $-50mV$ liegt, zu überschreiten. Erst beim Überschreiten kommt es zu dieser selbstverstärkenden Reaktion, die das Aktionspotential hervorbringt.

Bei den meisten Neuronen sind aber neben den Natrium-Kanälen auch spannungsabhängige Kalium-Kanäle vorhanden. Mit diesen läßt sich der Zeitverlauf eines Aktionspotentials weiter beschleunigen. Wie der Na^+- Kanal liegt auch der K^+-Kanal im Ruhezustand des Neurons in

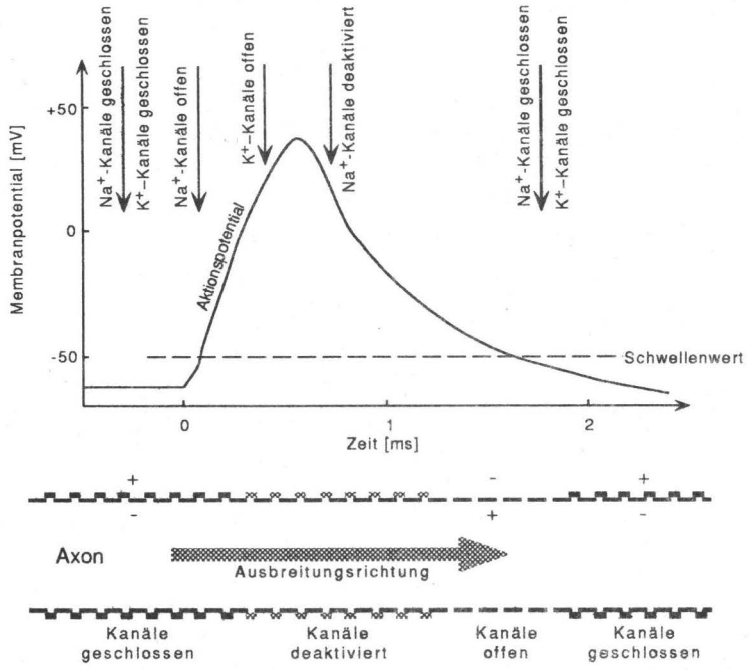


Abbildung 1.12: Ablauf eines Aktionspotentials

geschlossener Konformation vor und geht bei der Depolarisierung der Membran in den offenen Zustand über. Dieser Übergang geht gegenüber dem Na^+- Kanal langsamer, und der K^+-Kanal geht nicht spontan in eine inaktive Form über. Stattdessen wird der Kanal erst mit der Annäherung des Potentials an das Ruhepotential wieder geschlossen. Dies führt zu folgendem Ablauf des Aktionspotentials:

Zunächst werden sich die Natrium-Kanäle öffnen und für die Depolarisation der Zellmembran führen, bis das Potential positive Werte erreicht. Bevor der Maximalwert erreicht ist, sperren die Natrium-Kanäle, wodurch sich die Membran langsam repolarisieren würde. Kurz nach dem Erreichen des Maximalwertes der Depolarisation öffnen sich jedoch die Kalium-Kanäle, woraus eine starke Erhöhung des Kalium-Ionen-Flusses resultiert. Dadurch wird die Membran sehr schnell wieder repolarisiert, und die Kalium-Kanäle gehen wieder in die geschlossene Form über, womit der Ausgangszustand, das Ruhepotential, wieder erreicht ist. Der ganze Vorgang dauert nur etwa eine Millisekunde. Allerdings kann nun nicht gleich ein neues Aktionspotential ausgelöst werden, da sich viele Natrium-Kanäle noch in der deaktivierten Konformation befinden, ein Zustand, der einige Millisekunden andauern kann. Erst nach dieser *Refraktärzeit* kann erneut ein Aktionspotential ausgelöst werden. Zusätzlich zu den Natrium- und Kalium-Kanälen sind noch eine Reihe anderer Ionenkanäle an der Ausbildung eines Aktionspotentials beteiligt.

Da es sich bei dem Aktionspotential um ein Alles-oder-Nichts-Signale mit nur zwei unterscheidbaren Zuständen handelt, und auch der Zeitverlauf im wesentlichen konstant ist, bleibt für die Kodierung der Informationen nur das Intervall zwischen zwei Aktionspotentialen übrig. Da sich das Aktionspotential nicht abschwächt und zudem noch um ein vielfaches über dem Schwellenwert liegt, ist diese Art der Signalleitung sehr sicher. Durch die Refraktärzeit kann das Potential nicht rückwärts laufen, denn dort ist noch keine Depolarisierung möglich. Ferner ist die maximale Frequenz der Aktionspotentiale durch die Summe aus Dauer eines Signals und Refraktärzeit begrenzt, was eine maximale Frequenz von nur $500Hz$ bedingt.

Die Geschwindigkeit des Aktionspotentials ist allerdings durch die elektrotonischen Eigenschaften der Membran begrenzt, denn ein vor einem aktiven Abschnitt liegender Bereich muß zunächst einmal depolarisiert werden. Eine schnellere Leitung kann nur durch eine Vergrößerung des Widerstandes der Membran (weniger Ionenkanäle) oder durch eine Verringerung des inneren Widerstandes (durch Vergrößerung des Durchmessers des Nervenfortsatzes) erreicht werden. Mit der letzten Strategie erreichen Axone bei Wirbellosen Leitungsgeschwindigkeiten von ca. $20m/s$, was nur mit Axondurchmessern von $1mm$ erreicht werden kann. Wirbeltiere haben hier mehr auf die erste Strategie der Heraufsetzung des Membranwiderstandes gesetzt, indem die Nervenfortsätze von einer Markscheide umgeben ist, die in kurzen Abständen eingeschnürt ist.

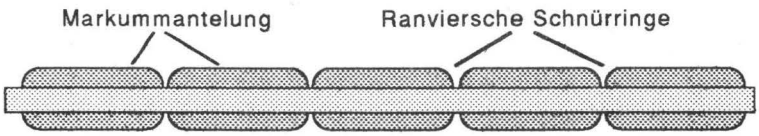


Abbildung 1.13: Markscheide

Diese Markscheide besteht zu 70 % aus Phospholipiden und Glykolipiden, die dafür sorgen, daß eine Markscheide ein äußerst schlechter Leiter ist. Auch befinden sich in der umhüllten Membran keine Ionenkanäle. Diese kommen nur an den Einschnürungen, den sogenannten Ranvier-Schnürringen, vor, dort allerdings stark angereichert. Das Aktionspotential springt bei dieser *saltatorischen Erregungsleitung* von Schnürring zu Schnürring und erreicht dadurch Geschwindigkeiten von etwa $100m/s$. Bestünde das Rückenmark des Menschen nur aus Neuronen ohne Markscheide, müßte es einen Durchmesser von mehreren Metern haben, um die gleiche Ausbreitungsgeschwindigkeit zu gestatten. Daher ist diese Umhüllung sehr platzsparend. Ein Verlust dieser Markscheide hat gravierende Defekte der Funktion des Nervensystems zur Folge, wie sich bei der Krankheit Multiple Sklerose in dramatischer Weise zeigt.

1.4 Signalübertragung an den Synapsen

Als man erkannte, daß Neuronen elektrische Signale generieren und weiterleiten, vermutete man, daß auch die Übertragung auf andere Neuronen elektrisch vor sich ging. Doch bevor sich diese These betätigen ließ, folgte die Entdeckung der Übertragung durch Botenstoffe, sogenannte Neurotransmitter, und durch diese Entdeckung geriet die Idee der elektrischen Signalweitergabe fast genau zu dem Zeitpunkt ins Hintertreffen, als sie zum ersten Mal in Laborversuchen nachgewiesen wurde. Zwar herrschen in fast allen Nervensystemen chemische Synapsen vor, in bestimmten Bereichen des Nervensystems sind daneben aber auch meistens elektrische Synapsen anzutreffen.

1.4.1 Elektrische Synapsen

Liegen zwei Neuronen ohne eine spezielle Kontaktstelle sehr dicht zusammen, kann ein Teil des Stroms aus dem ersten Neuron für eine Beeinflussung des zweiten Neuron sorgen, die jedoch im allgemeinen viel zu schwach ist, um dort ein Aktionspotential auszubilden. Denn der Strom müßte die Membranen beider Zellen durchdringen, und dazu ist der Membranwiderstand wesentlich größer als der Widerstand durch den Extrazellularraum zurück.

Um diese Leitfähigkeit zu erhöhen, gibt es an elektrischen Synapsen sogenannte **Connexon-Moleküle**, die durch beide Zellmembranen hindurch einen leitenden Kanal ausbilden. Dabei handelt es sich um recht große Transmembranproteine, die eine Röhre mit einem zentralen Hohlraum von etwa $2nm$ bilden. Für die Bildung eines leitenden Kanals müssen in beiden Membranen diese Connexon-Moleküle an derselben Stelle liegen.

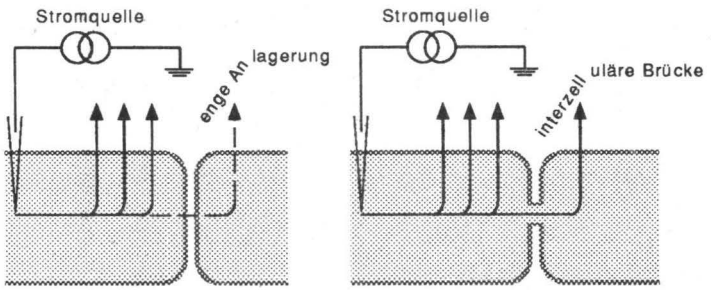


Abbildung 1.14: elektrische Verbindung zwischen zwei Neuronen

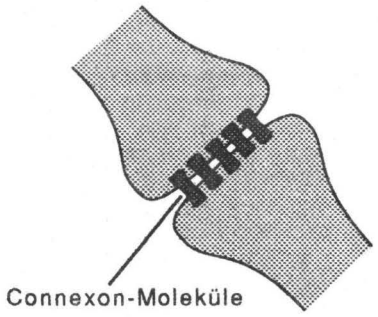


Abbildung 1.15: elektrische Synapse

Die Vorteile der elektrischen Synapsen liegen in der Übertragungsgeschwindigkeit, da der Strom ohne Umwandlung vom präsynaptischen auf das postsynaptische Neuron übergehen kann. Daher findet man elektrische Synapsen in der Natur immer dort, wo es auf besonders schnelle Signalübertragung ankommt, z. B. an den Gehirnstellen, die für das Fluchtverhalten zuständig sind, da in solchen Situationen Millisekunden über Leben und Tod des Lebewesens entscheiden können.

Die meisten elektrischen Synapsen verhalten sich symmetrisch, d. h. ein Potential kann die Synapse aus beiden Richtungen passieren. Im Falle eines Aktionspotential ist ein Rücklauf wegen der Refraktärzeit allerdings nicht möglich. Bekannt sind auch *spannungsabhängige elektrische Synapsen*, deren Kanal je nach Spannung leitet oder sperrt, wodurch der Strom wie bei einer elektrischen Diode nur in einer Richtung durchgelassen wird. Der größte Nachteil der elektrischen Synapsen ist das Fehlen jeglicher Signalverstärkung, weshalb prä- und postsynaptische Neuriten aufeinander abgestimmt sein müssen. Der Strom an einer kleinen synaptischen Endigung von $1-2\mu m$ Durchmesser kann eine große Faser mit $100\mu m$ Durchmesser kaum beeinflussen. Die Connexon-Moleküle sind deshalb an elektrischen Synapsen immer gruppenweise vorhanden, um den größtmöglichen Stromfluß zu gewährleisten. Bei den im folgenden Abschnitt beschriebenen chemischen Synapsen gibt es solche Einschränkungen nicht.

1.4.2 Chemische Synapsen

Bei einer chemischen Synapse erfolgt die Übertragung des Signals durch sogenannte **Neurotransmitter**. Diese werden an der präsynaptischen Endigung bei Eintreffen eines entsprechenden Potentials in den synaptischen Spalt freigesetzt, diffundieren über diesen hinweg, um sich an bestimmte **Rezeptorstellen** der postsynaptischen Endigung anzuheften. Diese Rezeptormoleküle verändern bei Anheftung eines solchen Neurotransmitters ihre Konformation und sorgen so direkt oder indi-

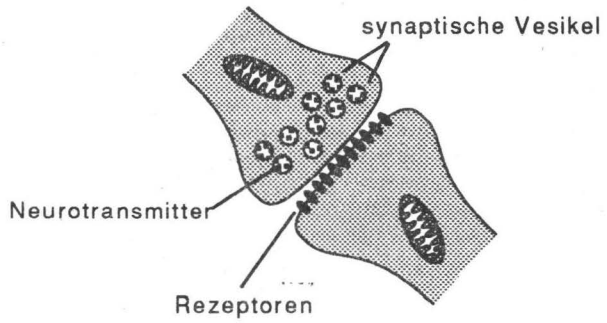


Abbildung 1.16: chemische Synapse

rekt für eine Veränderung des Potentials in der postsynaptischen Membran.

In einer präsynaptischen Endigung befinden sich die Neurotransmitter in den Vesikeln, und werden bei Eintreffen eines Potentials mit der zellulären Membran im Bereich der Synapse exozytotisch verschmolzen, woraus eine Freisetzung der in den Vesikeln enthaltenen Botenstoffe in den synaptischen Spalt resultiert. Die in der Endigung vorhandenen Mitochondrien sorgen für die Energieversorgung dieses Vorgangs.

Die Entdeckung der chemischen Signalübermittlung über Transmitterstoffe geht auf das Jahr 1920 zurück, als als erster Neurotransmitter das *Acetylcholin* am Vagusnerv des Froschherzens entdeckt wurde. Seither wurden immer weitere Transmitter entdeckt, deren Zahl mittlerweile etwa fünfzig betragen dürfte, während mindestens dieselbe Anzahl an Stoffen im Verdacht steht, ebenfalls Transmitterfunktionen übernehmen zu können. Dennoch muß ein solcher Stoff bestimmten Anforderungen genügen, um als Neurotransmitter im engeren Sinne zu gelten. Zunächst einmal sollte er im Neuron synthetisiert werden und in der präsynaptischen Endigung in den Vesikeln vorliegen. Ferner muß der Stoff, wenn er in den synaptischen Spalt injiziert wird, dieselben Wirkungen entfalten wie der Stoff aus der Zelle selbst. Und letztendlich muß er auf demselben Weg abgebaut werden können wie das Originalprodukt.

Im Nervensystem unterscheidet man zwei Gruppen von Neurotransmittern. Zunächst einmal sind dies kleine neuroaktive Moleküle, die fast ausschließlich Aminosäuren sind und als klassische Neurotransmitter gelten. Gesichert ist hier das Vorhandensein von mindestens acht verschiedenen Stoffen, die allesamt unter Einwirkung von Enzymen in der Zelle, oft sogar in der synaptischen Endigung selbst, produziert werden.

Drei dieser gesicherten acht Botenstoffe werden nun kurz vorgestellt:

Acetylcholin
wurde in den zwanziger Jahren als erster Neurotransmitter entdeckt. Der Stoff kommt in den Endplatten der Skelettmuskulatur sowie in vielen Zellen des autonomen Nervensystems vor. Über seine genaue Wirkung bestehen immer noch Zweifel.

Noradrenalin
war in den dreißiger Jahren nach Acetylcholin der zweite entdeckte Neurotransmitter. Er sorgt beispielweise dafür, daß sich bei Streß der Herzschlag beschleunigt, die Bronchien erweitern sowie der Blutdruck erhöht.

Dopamin
ist erst 1958 im motorischen Kontrollzentrum des Gehirn entdeckt worden. Wird dieser Neurotransmitter z. B. durch Psychopharmaka in seiner Funktion gestört, stellen sich die Symptome

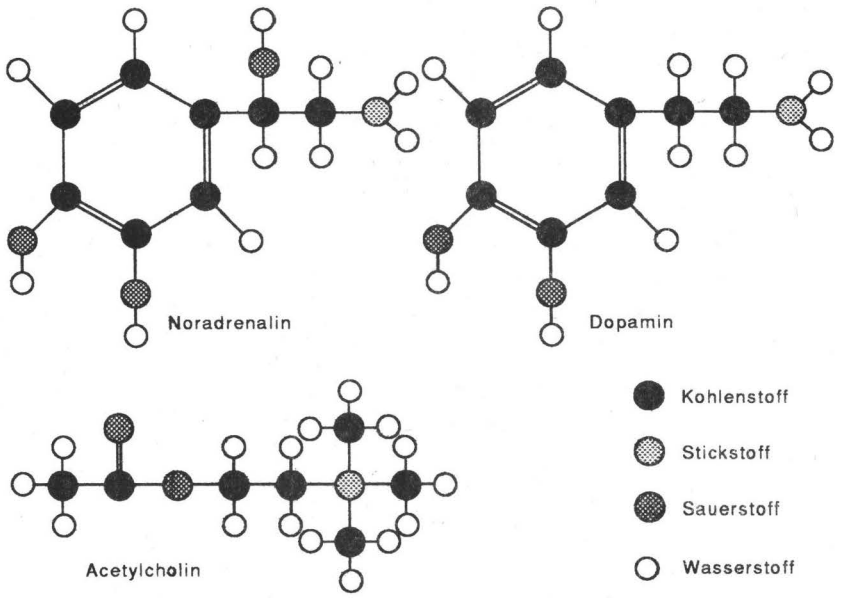


Abbildung 1.17: Strukturformeln klassischer Neurotransmitter

der Parkinson-Krankheit wie Starrsinn und Muskelzucken ein. Medikamente zur Behandlung schizophreniekranker Patienten, sogenannte Antischizophrenika, hindern diesen Transmitter an der Entfaltung seiner Wirkung, weshalb diese Patienten dann häufig die Parkinson-Syndrome ausbilden.

Bei der zweiten Gruppe der Neurotransmitter handelt es sich um die sogenannten Neuropeptide, das sind Ketten von zwei bis fünfzig Aminosäuren. Dieser Gruppe gehören die meisten der heute bekannten Transmitterstoffe an, und es gibt Forscher, die deren Anzahl auf mehrere Tausend schätzen. Von diesen betrachten wir nun kurz die Gruppe der **Enkephaline**. Diese Moleküle rufen im Gehirn ähnliche Wirkungen hervor wie Opiate, sie wirken also schmerzlindernd. Opiate wurden noch vor 100 Jahren als allgemein schmerzlindernde Medikamente anerkannt und eingesetzt. Zu dieser Gruppe gehört auch das Rauschgift Heroin, welches anfangs als nicht-süchtig machendes, hochwirksames Schmerzmittel gepriesen wurde, und das, obwohl mit Opium (Heroin ist chemisch bis auf zwei Seitengruppen mit Opium identisch) behandelte Soldaten aus dem Deutsch-französischen Krieg 1870/71 mit als ‚Soldatenkrankheit' bezeichneten Suchterscheinungen zurückkehrten. Enkephaline wurden entdeckt, als man untersuchte, an welche Rezeptoren der postsynaptischen Membran sich die Opiatmoleküle binden, denn sowohl Enkephaline als auch Opiate besetzen die gleichen Rezeptoren. Diese Rezeptoren sind im Gehirn im Limbischen System, welchem die Kontrolle über das emotionale Verhalten zugeschrieben wird, und im sogenannten periaquaeductalen Grau, durch welches die Schmerzsignale verlaufen, lokalisiert worden, wodurch sich die schmerzlindernde Wirkung erklärt [2, 5].

Nach diesem kleinen Abstecher nun zurück zu den Synapsen. Die Forschung hat ergeben, daß sich die spannungsabhängigen Calcium-Kanäle an der Synapse bei einem in der Membran ankommenden Potential öffnen und so Ca^{2+}-Ionen in die Zelle gelangen. Indem man die in der Membran enthaltenen Kalium- und Natrium-Kanäle blockiert hat, konnte man zeigen, daß diese bei den biochemischen Vorgängen hier keinen Einfluß haben, sondern das ausgelöste postsynaptische Potential in der Stärke ausschließlich von den einströmenden Calcium-Ionen abhängt. Je mehr Calcium-Ionen in die Zelle gelangen, desto stärker fällt das Potential in der postsynaptischen Nervenzelle aus. Allerdings ist für die postsynaptische Signalauslösung wieder ein bestimmer Schwellenwert zu

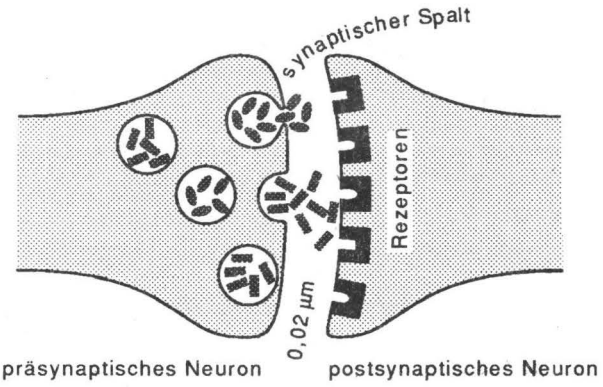


Abbildung 1.18: Vesikeln verschmelzen mit der Membran

überschreiten. Zudem ist die chemische Übertragung mit etwa $0,5ms$ recht langsam. Obwohl man die genauen Mechanismen dieses Vorgangs noch nicht in allen Einzelheiten verstanden hat, führt dieser Calcium-Einstrom zum Verschmelzen der synaptischen Vesikeln mit der präsynaptischen Membran und damit zur Freisetzung der jeweiligen Neurotransmitter.

Da mit jedem Vesikel eine bestimmte, pro Vesikel relativ konstante Anzahl an Transmittermolekülen in den synaptischen Spalt ausgeschüttet wird, und diese an der postsynaptischen Membran nur eine entsprechende Anzahl Rezeptoren besetzen können, wird pro Vesikel nur ein geringes Potential ausgelöst. Erst wenn ausreichend Vesikel ihren Inhalt ausgeschüttet haben, erreicht die Summe der einzelnen Potentiale eine ausreichende Größe, um auch im postsynaptischen Neuron ein eindeutiges Potential zu erzeugen. Jedes Vesikel enthält etwa 5000–10000 Transmittermoleküle, und bei jeden Aktionspotential werden an der präsynaptischen Membran zwischen 10 und 200 solcher Quanten freigesetzt.

In der postsynaptischen Membran sind Rezeptormoleküle eingebettet, an die sich die verschiedenen Transmittermoleküle anheften können. Diese Rezeptoren können im allgemeinen nicht nur einen bestimmten Molekültyp anlagern, sondern erlauben oft auch anderen Molekülen das Anbinden. So akzeptieren die Rezeptoren für die körpereigenen Enkephaline auch die körperfremden Opiatmoleküle. Die Umsetzung in ein elektrisches Potential kann dabei auf mehreren Wegen erfolgen. Einige Rezeptoren sind mit beweglichen Natrium- oder Kalium-Kanälen verbunden, die sich durch das Anbinden des Transmitters öffnen, und so bei ausreichender Anzahl ein Aktionspotential in dieser Zelle hervorbringen können.

Möglich ist aber auch bei inhibitorisch wirkenden Transmittern, daß ein Aktionspotential von der postsynaptischen Zelle für eine gewisse Zeit unterbunden wird. Manche Transmitter bleiben über Stunden oder gar Tage an ihren spezifischen Rezeptoren gebunden. Auch können die Rezeptoren biochemische Veränderungen in der Zelle hervorrufen und z. B. die Produktion eines bestimmten Neurotransmitters anregen oder blockieren, sie müssen also nicht zwangsweise ein elektrisches Potential auslösen. Bisher hat man allerdings nur einen unvollständigen Überblick über alle Funktionsweisen der Neurotransmitter.

Um nun das postsynaptische Neuron erneut aufnahmebereit zu machen, müssen zunächst die an den Rezeptoren gebundenen Moleküle entfernt werden. Diese lösen sich nach einiger Zeit von ihren Rezeptoren, und diffundieren durch den synaptischen Spalt. Während einige Neurotransmitter aus diesem herausdiffundieren, werden andere durch spezielle Enzyme abgebaut. Meist allerdings werden diese von den Gliazellen, das sind spezielle, nicht direkt an der Signalleitung beteiligte Nervenzellen, die lediglich Stützfunktionen für das Gewebe haben, oder von den Neuronen, von

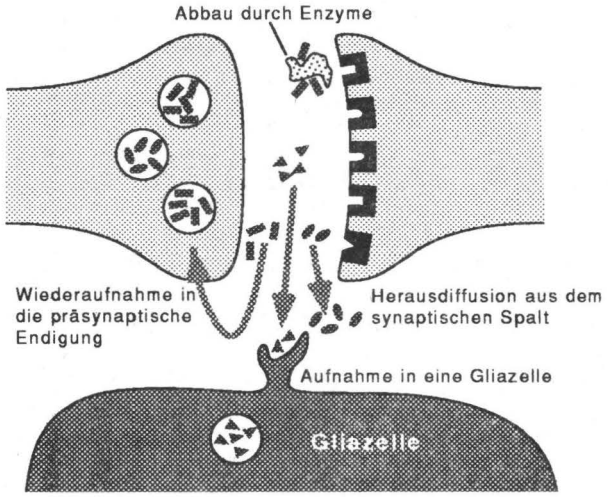


Abbildung 1.19: Entfernung der Neurotransmitter

denen sie erzeugt wurden, wieder aufgenommen. In ihren Ursprungsneuronen stehen sie nun wieder in den Vesikeln zur Verfügung.

Ein postsynaptisches Neuron benötigt sehr viele kleine Potentialschübe, um selbst darauf zu reagieren. Meist reicht ein an einer Synapse eingehendes Potential nicht zur Ausbildung eines Aktionspotentials aus, so daß das Neuron auf die Signale von möglichst vielen Eingängen angewiesen ist, um selber zu ‚Feuern'. Reicht das an einem Eingang angelangte Signal nicht zur Ausbildung eines Aktionspotentials aus, pflanzt es sich über kurze Entfernungen elektrotonisch fort. Sind diese von den verschiedenen Eingängen des Neurons kommenden elektrotonischen Signale stark genug und erreichen noch den Anfang eines Axon, so kann es hier zur Ausbildung eines Aktionpotentials kommen. Lange Axone haben nämlich an dieser Stelle den Axonhügel, der für diese Umwandlung und Ausprägung des Aktionspotentials dient.

1.5 Zusammenfassung

In diesem Kapiteln haben wir Einblicke in den hochkomplexen Aufbau und die Funktionsweise von in der Natur vorkommenden neuronalen Netzen erhalten. So sind Ionenflüsse durch eine elektrisch isolierende Zellmembran für die Signalleitung verantwortlich. Befindet sich ein Neuron in Ruhe, so bildet sich durch das Zusammenspiel von Ionenpumpen und Ionenkanälen ein Ruhepotential von etwa $-40mV$ aus. Die Zellmembran selbst ist eher ungeeignet zur elektrischen Potentialfortpflanzung, da sich elektrotonische Signale bereits nach kurzer Zeit verzerren und abschwächen. Nur das Aktionspotential gewährleitet eine schnelle, unverzerrte, verlustfreie und sichere Transportleistung, da dieser Prozeß ein selbstverstärkender Alles-oder-Nichts-Prozeß ist. Der Übergang der Signale auf eine andere Zelle erfolgt an den Synapsen. Dabei kann man zwischen elektrischen und chemischen Synapsen unterscheiden. An den schnellen elektrischen Synapsen springt das elektrische Signal direkt auf die nächste Zelle über, allerdings mit dem Nachteil, daß diese Art der Übertragung nicht selbstverstärkend ist. Im Gegensatz dazu setzen chemische Synapsen Botenstoffe, sogenannte Neurotransmitter ein, die eine einwandfreie Übertragung auf die postsynaptische Zelle garantieren. Dort heften sich diese Transmitter an Rezeptoren und sorgen direkt oder indirekt für die Entstehung eines geringen Potentials. Erst das Zusammenspiel vieler solcher Mini-Potentiale führt aber zu der Ausbildung eines Aktionspotentials am Ausgang der Zelle.

Obwohl man viele der Vorgänge im Nervensystem bereits kennt und verstanden hat, ist man

noch weit von einem tatsächlichen Verstehen der Funktionsweise entfernt. Zum einen gibt es noch viel zu viele unbekannte Faktoren, zum anderen glaube ich nicht, daß allein die Kenntnis aller dieser Vorgänge für ein Verständnis des Geschehens ausreicht.

1.6 Literatur

[1] Christian de Duve. *Die Zelle.* Spektrum der Wissenschaft, Heidelberg, 1986

[2] Elliot S. Gershon und Ronald O. Rieder. *Molekulare Grundlagen von Geistes- und Gemütskranheiten.* Spektrum der Wissenschaft, Heidelberg, Heft 11/1992

[3] Gerald Fischbach. *Gehirn und Geist.* Spektrum der Wissenschaft, Heidelberg, Heft 11/1992

[4] H. Reichert. *Neurobiologie.* THIEME, 1990

[5] Solomon H. Snyder. *Chemie der Psyche.* Spektrum der Wissenschaft, Heidelberg, 1989

Kapitel 2

Frühe Anfänge

Peter Conrad

Übersicht

Im folgenden Kapitel wird versucht, einen Überblick über die Geschichte der neuronalen Netze zu geben, angefangen bei den ersten 'Denkmodellen' von Aristoteles ca. 400 v.Chr. bis hin zu modernen Algorithmen wie dem Backpropagation.

2.1 Einleitung

"Cogito ergo sum - Ich denke, also bin ich." [1] Diesen ebenso einfachen wie beeindruckenden 'Existenzbeweis' lieferte Rene Descartes der Philosophie vor ca. 350 Jahren. Die Frage, die viele Wissenschaftler aber seit fast zweieinhalbtausend Jahren beschäftigt, lautet: "Sum quomodo cogito? - Ich bin, aber wie denke ich?" Antworten zu dieser Frage gab es im Laufe der Zeit viele. Der Fortschritt der Technik war oft Voraussetzung für ein neues, genaueres Bild des menschlichen Hirns, und mit der detaillierteren Kenntnis der Physis ist es dem Menschen immer besser gelungen, ein Bild der Psyche, also der Semantik der Physis, zu entwerfen. Ein Nebenprodukt dieser Forschung ist in gewisser Hinsicht der von Neumann-Rechner, dessen 'Evolution' Ende des zweiten Weltkriegs eine andere Richtung einschlug als die der massiv parrallelen Systeme. Der vorläufige Höhepunkt dieser langen Entwicklung ist die Theorie der Neuronalen Netze, der sich dieses Seminar widmet.

2.2 Frühe Modelle

2.2.1 Modelle der Psychologie

Bereits Aristoteles entwarf in seinem Buch "De memoria et reminiscentia" [2] eine Theorie, um das menschliche Denken zu erklären. Er unterschied dabei zwei wesentliche Vorgänge: konkrete Gedanken und das Denken als kreativer Vorgang.

Gedanken sind immer an ein konkretes Bild gebunden, das der Mensch über seine Sensoren aufnimmt und in irgendeiner Form festhält (er vergleicht dieses 'Festhalten' mit dem Abdruck eines Siegelringes in Wachs). Der Gedanke bzw. die Erinnerung ist also die Folge einer Wahrnehmung. Indem sich der Mensch an eine Situation erinnert, 'betrachtet' er das festgehaltene Bild der vergangenen Situation. Um nicht das Bild mit der Wirklichkeit zu verwechseln, gehört zu jeder Erinnerung eine Art 'flag', das angibt, ob die Situation in der Vergangenheit liegt oder tatsächlich gerade wahrgenommen wird. Geisteskrankheit rührt daher, daß der Kranke dieses 'flag' nicht (oder falsch) interpretiert.

Im Gegensatz dazu ist das Denken nicht die direkte Folge von sensorischem Input, sondern die Verbindung vorhandener Gedanken zu neuem Wissen. Diese Verbindungen werden vor allem nach zwei Gesetzmäßigkeiten aufgebaut: einerseits zwischen gleichen, gegensätzlichen oder ähnlichen Gedanken, andererseits nach der Gewohnheit des Denkenden. Dies ist der Grund, warum der Mensch assoziativ denkt.

Dieses sehr einfache Modell enthält bereits einige wesentliche Punkte, die wir auch heute bei neuronalen Netzen kennen. Gedanken manifestieren sich nach heutigem Wissen tatsächlich in einer Art 'Bild' in der Struktur des Hirns, bei Versuchen hat man direkte Verbindungen zwischen Punkten auf der Netzhaut und Punkten auf der Großhirnrinde festgestellt. Der Ausdruck 'Gewohnheit des Denkenden' drückt bereits genau das aus, was Hebb 1949 konkreter formulierte: je häufiger eine Verbindung (zwischen Gedanken bei Aristoteles, zwischen Neuronen bei Hebb) genutzt wird, desto effektiver wirkt sie.

Doch trotz dieser Aussagen zielte Aristoteles in vielerlei Hinsicht weit daneben, z.B. wenn er schließt, das Denken sei ein körperlicher Vorgang, weil erfolgloses Nachdenken sich in Wut und körperlicher Aggression äußern kann, und weil Denkprozesse oft im Unterbewußtsein weiterlaufen, ohne daß man sie anhalten kann, ähnlich einer Flüssigkeit, die, einmal in Fluss geraten, schwer aufzuhalten ist.

2.2.2 Modelle der physischen Struktur

Auch zur Struktur des Denkorgans hat Aristoteles eine Theorie entworfen. Er wurde dabei stark von Plato's Ideen beeinflußt, der das Denken für einen Vorgang in der Seele hielt. Da die Seele perfekt ist, mußte sie sich in einem perfekten Organ befinden, und für den perfektesten Körperteil hielt Plato den Schädel mit seiner (fast) perfekten Kugelform. Aristoteles modifizierte diese Theorie, indem er den Sitz der Seele in das Herz verlegte und dem Hirn die Aufgabe zuschrieb, das Blut zu kühlen.

Das erste nennenswerte Forschungsergebnis in Bezug auf das menschliche Nervensystem lieferte 130 n.Chr. der römische Arzt Claudius Galenus [3]. Schwer verletzte und gefallene Gladiatoren lieferten ihm reichlich Anschauungsmaterial, und so konnte er feststellen, daß ein Mensch noch Gefühl in einem Körperteil haben kann, obwohl er ihn nicht mehr bewegen kann. Aufgrund derartiger Beobachtungen entwickelte er ein Modell eines Systems von Nerven, durch die, ähnlich den Blutbahnen, eine sehr feine Flüssigkeit floß. In Anlehnung an Platos Theorie glaubte er, daß aus der Nahrung in der Leber eine Flüssigkeit gewonnen wurde, die von Herz und Hirn weiter verfeinert und schließlich in die Nervenbahnen geleitet wurde.

Diese Theorie hielt sich, zunächst mangels technischer Hilfsmittel, später mangels einer besseren Erklärung, über 1600 Jahre. Nach der Erfindung des Mikroskops 1660 n. Chr. erkannte man zwar schnell, daß die Nerven nicht hohl sind und Flüssigkeiten leiten, sondern daß sie mit einer weichen Substanz gefüllt sind. Jedoch erst im Jahre 1791 konnte Luigi Galvani experimentell nachweisen, daß die Nerven elektrische Impulse leiten. In seinem berühmten Experiment regte er Froschschenkel durch elektrische Schocks zu Bewegungen an. Auf welche Art und Weise diese Elektrizität entstand, konnte Volta durch seine Batterie erklären.

Wiederum bedurfte es einer technischen Weiterentwicklung, bevor die elektrischen Signale der Nerven genauer entschlüsselt werden konnten. 1850 gelang es Emil DuBois-Reymond nicht nur, die von Galvani vorausgesagte elektrische Aktivität der Nerven nachzuweisen, sondern er zeigte auch, daß diese Elektrizität kein konstanter Strom ist, wie man bisher angenommen hatte, sondern sich vielmehr in kurzen, spitzen Impulsen äußert.

Wenig später gelang es dem Engländer Richard Canton, mit noch empfindlicheren Instrumenten den Unterschied zwischen motorischen und sensorischen Neuronen nachzuweisen, den Galen vorausgesagt hatte. Außerdem lokalisierte er Signale im Affenhirn, die durch Licht auf der Netzhaut ausgelöst werden konnten.

Nachdem die Elektrizität als Informationsträger in den Neuronen nachgewiesen war, wurde das Nervensystem gerne mit dem in den Vereinigten Staaten entstehenden Telegraphennetz verglichen. Experimente von Samuel Morse hatten gezeigt, daß die Elektrizität sich sehr schnell über die Telegraphenleitungen ausbreitete. Entsprechend gingen einige Spekulationen davon aus, daß die

Impulsgeschwindigkeit in den Nervenbahnen bei etwa zehn Millionen Meilen pro Sekunde lag. Es gelang jedoch Hermann von Helmholtz, die Impulsgeschwindigkeit in den Nerven zu messen. Das Ergebnis, etwa 90 Meilen pro Stunde, warf eine neue Frage auf: warum ist der Telegraph so viel schneller als die Nervenbahnen?

Diese Frage konnte nicht so bald beantwortet werden. Nachdem Jan Purkinie 1838 den Aufbau der Nervenzellen und die Unterteilung in Axon und Zellkörper entdeckt hatte, ging man immer noch davon aus, daß das Nervensystem ein zusammenhängendes Ganzes ist. Erst die Entwicklung sehr feiner Kontrastmittel erlaubte es etwa vierzig Jahre später dem Spanier Santiago Ramon Cajal die Synapse zu entdecken. Jedoch war die damalige Technik nicht in der Lage, die an Synapsen entstehenden Spannungen von wenigen Millivolt exakt zu messen. Erst mit der Entwicklung der Vakuumröhren konnte man diese Spannungen verstärken und messen. In den ersten Jahren dieses Jahrhunderts wurde mit solchen Hilfsmitteln die Informationsübertragung am synaptischen Spalt erforscht. Dabei fand man auch die Antwort auf die Frage, warum die Nerven Informationen so langsam übertragen: am synaptischen Spalt wird die Information nicht elektrisch sondern auf chemischem Weg weitergegeben. Bei der genaueren Untersuchung der beteiligten Chemikalien stieß man auf viele Substanzen, die die Übertragung beeinflußen, wie z.B. Nervengas und Drogen, aber auch Heilmittel für verschiedene Nervenkrankheiten.

Mit allen diesen Entdeckungen und dem immer weiter verbesserten Verständnis der Arbeitsweise der Nervenzellen wurde eines klar: Um zu verstehen, wie der Mensch denkt, genügt es nicht, die Funktionsweise der einzelnen Zellen zu entschlüsseln. Viel wichtiger erscheint es, das Zusammenwirken der einzelnen Nerven im ganzen Nervensystem zu verstehen. Zu diesem sehr komplexen Ziel führt nur ein enges Zusammenarbeiten verschiedener Wissenschaften wie Mathematik, Psychologie, Physiologie und heutzutage auch der Informatik.

2.3 Erste Mathematische Modelle

2.3.1 William James

William James gilt als der bedeutendste amerikanische Psychologe des 19. Jahrhunderts. Eine gekürzte Ausgabe seiner "Principles of Psychology" [4] gehört auch heute noch zur Pflichtlektüre amerikanischer Psychologiestudenten. In diesem Werk finden sich neben detaillierten Beschreibungen diverser anatomischer Untersuchungen an Tieren (bezeichnend ist wohl der Satz "The way really to understand the brain is to dissect it") auch einige sehr konkrete Ansätze zur Beschreibung der Hirnfunktion mit mathematischen Mitteln. Als Psychologe kam James vermutlich nicht auf die Idee, diese Ansätze auf der Ebene der Mathematik weiterzuverfolgen. Aber viele Leute haben auf dieser Basis weiterdenken können, und sind auch zu sinnvollen Ergebnissen gekommen.

Ähnlich wie Aristoteles beschreibt er die Denkvorgänge mit Hilfe von Begriffe wie 'Gedanken' oder 'Vorstellungen' von Objekten einerseits bzw. Assoziationen zwischen den Objekten andererseits. Dabei benutzt er auch mathematische Ausdrücke. So schreibt er z.B. daß die Aktivität in einem Punkt der Hirnrinde der Summe der Aktivitäten aller Punkte entspricht, die ihre Aktivität an diesen Punkt weitergeben, wobei die Wahrscheinlichkeit, daß der Aktivierungszustand weitergegeben wird, proportional ist zur Intensität der Aktivierung und zur Häufigkeit der Weitergabe eines aktiven Zustands. Diese Formulierung deutet bereits auf die Neuronen heutiger mathematischer Modelle hin.

2.3.2 Pitts & McCulloch

1943 stellten Walter Pitts und Warren McCulloch [5] ein 'echtes' mathematisch-logisches Modellneuron vor. Es handelte sich dabei um ein einfaches Schwellwertneuron mit absolut inhibitorischen Synapsen, d.h. wenn eine inhibitorische Synapse aktiviert ist, kann das Neuron auf keinen Fall feuern. In einem Zeitintervall dt wird die gewichtete Summe aller Synapsen berechnet, und wenn die Summe einen Grenzwert überschreitet, ist das Neuron im nächsten Zeitintervall selbst aktiviert. Dieses Modell ist als 'McCulloch-Pitts-Neuron' in sehr vielen Netzen verwendet worden.

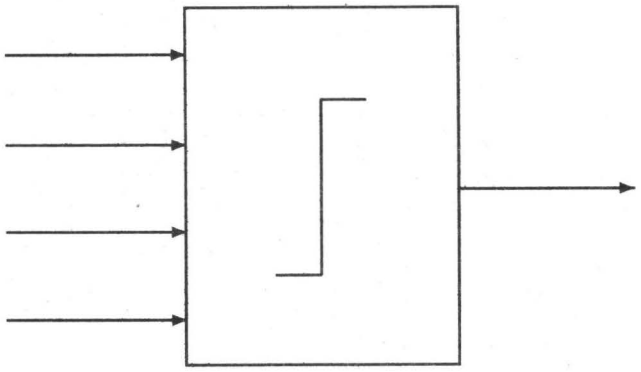


Abbildung 2.1: Graphische Darstellung des McCulloch-Pitts-Neurons

Was bei Erscheinen des Artikels großes Aufsehen verursachte, war, daß Pitts und McCulloch bewiesen, daß jede logische Funktion mit Hilfe solcher Neuronen berechnet werden kann. Diese wurden durch 5 einfache Annahmen beschrieben:

1. Neuronen haben binären Charakter, d.h. entweder sie feuern, oder sie feuern nicht.
2. Eine bestimmte, feste Anzahl von Synapsen muß während der Phase der Aufsummierung aktiviert sein. Diese Zahl ist unabhängig von der Position des Neurons im Netz und vorhergegangenen Aktivierungszuständen.
3. Die einzige signifikante zeitliche Verzögerung ist die am synaptischen Spalt hervorgerufene.
4. Die Aktivierung irgendeiner inhibitorischen Synapse verhindert die Aktivierung des Neurons.
5. Die Struktur des Netzes verändert sich nicht im Laufe der Zeit.

Eine graphische Darstellung ist in Abbildung 2.1 zu sehen.

Auf dieser Basis ließ sich beweisen, daß zu jeder endlichen logischen Funktion ein äquivalentes Netz aus solchen Neuronen aufgebaut werden kann. Desweiteren untersuchten Pitts und McCulloch Netze mit Schleifen. In solchen Netzen können Impulse prinzipiell beliebig lange eine Schleife immer wieder durchlaufen. Auf diese Art und Weise läßt sich ein einfacher Gedächtnismechanismus konstruieren.

Mit ihrem Artikel lieferten McCulloch und Pitts den ersten Versuch, zu erklären, wie das menschliche Hirn tatsächlich funktioniert: der Mensch denkt, indem sehr viele, sehr einfache Bausteine zusammen komplexe logische Funktionen berechnen. Auf die Psychologie hatten ihre Ausführungen sehr wenig Einfluß, da die logischen Funktionen zur Beschreibung des Denkens zwar im Prinzip sehr einfach waren, aber eine viel zu große Komplexität besaßen, wenn man sie in der Praxis anwenden wollte. Jedoch die beginnende Wissenschaft der automatischen Rechenmaschinen eröffnete Mathematikern die Möglichkeit, kompliziertere Netze zu modellieren und sehr einfache 'denkende' Maschinen zu bauen.

Nach damaligem Wissensstand entsprach dieses Modell den 'echten' Neuronen im menschlichen Gehirn. Das Verständnis der elektrochemischen Vorgänge in Nervenzellen reichte nicht aus, um zu erkennen, daß die Zellen keineswegs nur zwei mögliche Zustände besitzen, sondern fließende Übergänge haben. Ebensowenig erkannten Pitts und McCulloch, daß durch Verändern der synaptischen Verbindungen ein Netz 'lernen' kann. Heute ist klar, daß das menschliche Hirn keine ideale Rechenmaschine ist, wie dieser Artikel vermuten lassen könnte.

Vier Jahre später, 1947, veröffentlichten die beiden Wissenschaftler einen weiteren Artikel [6], in dem sie versuchen, die assoziativen Fähigkeiten des Hirns auf der Basis einfacher Modellneuronen zu erklären. Tatsache ist, daß der Mensch einen Gegenstand aus sehr vielen verschiedenen

Blickwinkeln, bei unterschiedlicher Beleuchtung usw. erkennen kann. Ebenso ist es möglich, verschiedene Gegenstände als zu einer Klasse gehörig zu identifizieren, z.B. erkennt man Quadrate als Quadrate weitgehend unabhängig von Lage, Größe oder Farbe. Bilder einer Person, mit unterschiedlichen Gesichtsausdrücken oder unterschiedlicher Körperhaltung aufgenommen werden als ein und dieselbe Person identifiziert, usw. usf.

Angeregt durch neurophysiologische Untersuchungen, u.a. von Julia Apter [7, 8], präsentierten sie einen Vorschlag, diese Frage zu beantworten. J. Apter hatte festgestellt, daß ein Lichtreiz an einer ganz bestimmten Stelle der Netzhaut ein Signal an einer bestimmten Stelle der Hirnrinde auslöst. Auf diese Weise ließ sich ein verzerrtes Bild der gesamten Netzhaut auf die Hirnrinde projizieren. In nachfolgenden Untersuchungen stellte sie eine Bewegung der Augen an eine bestimmte Stelle fest, wenn bestimmte korrespondierende Punkte der Hirnrinde elektrisch gereizt wurden. Auf diese Weise konnte sie ein ähnlich verzerrtes Bild des Sichtfeldes auf die Hirnrinde projizieren, das fast deckungsgleich mit der Projektion der früheren Versuche war.

Dies führte McCulloch und Pitts zu der Annahme, daß aufgenommene Bilder als Punktmatrix der Netzhaut auf eine Matrix von Neuronen abgebildet werden. Diese Neuronen können dann das Bild als ein ganz bestimmtes, durch die synaptischen Verbindungen vorgegebenes, identifizieren. Weitere Neuronen dienen dazu, endlich viele verschiedene Transformationen der Signale von der Netzhaut durchzuführen, z.B. Translationen und Rotationen, und die Ergebnisse ebenfalls der Matrix zu präsentieren. Dieser Mechanismus erlaubt es, verschiedene Ansichten eines Gegenstandes zu erkennen. Mit mehreren Matrizen ist es entsprechend möglich, mehrere Bilder zu erkennen. Ein Problem dieses Ansatzes liegt im immensen Aufwand zur Realisierung aller möglicher Transformationen für jede dieser Matrizen. Zur Lösung desselben schlugen McCulloch und Pitts einen Zeit-für-Platz Ansatz vor, in die transformierten Bilder nacheinander den verschiedenen Matrizen präsentiert werden, und die Ergebnisse der Matrizen selektiv nacheinander abgefragt werden.

Zur Unterstützung dieser These präsentierten sie ein einfaches Modell des menschlichen Gehörs. Der Vorteil dieses Beispiels lag darin, daß das Frequenzspektrum eines Klanges als ein Vektor dargestellt werden kann, entsprechend können verschiedene Klänge durch eine Matrix repräsentiert werden. Das Resultat wurde mit einer mikroskopischen Aufnahme eines für das Gehör zuständigen Bereichs der Hirnrinde verglichen, wobei ähnliche Strukturen erahnt werden konnten.

Nach heutigem Wissen ist dieser etwas andere Ansatz als ein Schritt in die richtige Richtung zu sehen. Es wurde dabei weniger das Hirn als ein massiver logischer Rechenapparat betrachtet, sondern die wesentlich deutlicher vorhandenen assoziativen Fähigkeiten zu erklären versucht.

2.3.3 Hebb

1949 veröffentlichte Donald O. Hebb sein Buch "The Organization of Behaviour" [9]. Seine besondere Bedeutung liegt darin, daß Hebb synaptische Veränderungen vorschlägt, mit deren Hilfe 'gelernt' werden kann. Synapsen, die dieses Verhalten zeigen, werden seither als 'Hebb'sche Synapsen' bezeichnet. Die allgemeine Formulierung dieser Idee führte dazu, daß diese Bezeichnung auf sehr viele unterschiedliche Lernalgorithmen zutrifft. Hebb führt aus, daß das Kurzzeitgedächtnis sehr wohl durch Signale in einem Ring von Neuronen repräsentiert werden kann. Da aber solche Schleifen sehr empfindlich auf äußere Störungen reagieren können, muß es für das Langzeitgedächtnis einen anderen Mechanismus geben. Wenn z.B. jemand eine Zahl mit vielen Stellen, sagen wir Pi, liest, wird er nicht in der Lage sein, sich an die letzten zehn oder 15 gelesenen Stellen zu erinnern. Jedoch jemand, der oft mit einer Näherung von Pi rechnet, wird ohne zu zögern die ersten 5-10 Nachkommastellen aufzählen. Dies führte Hebb zu der Erkenntnis, daß die häufige Wiederholung eines Gedankens zu einer permanenten Veränderung im Hirn führen muß. "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" dürfte der meistzitierte Satz in Hebb's Buch sein.

Es gibt aber noch weitere interessante Aspekte, die in diesem Buch angesprochen werden. Zum einen stellt Hebb fest, daß das Hirn nicht nur eine Verbindung zwischen sensorischen und motorischen Nerven darstellt. Selbst in einfachen Hirnen wie dem der Ratte ist die Motorik nicht

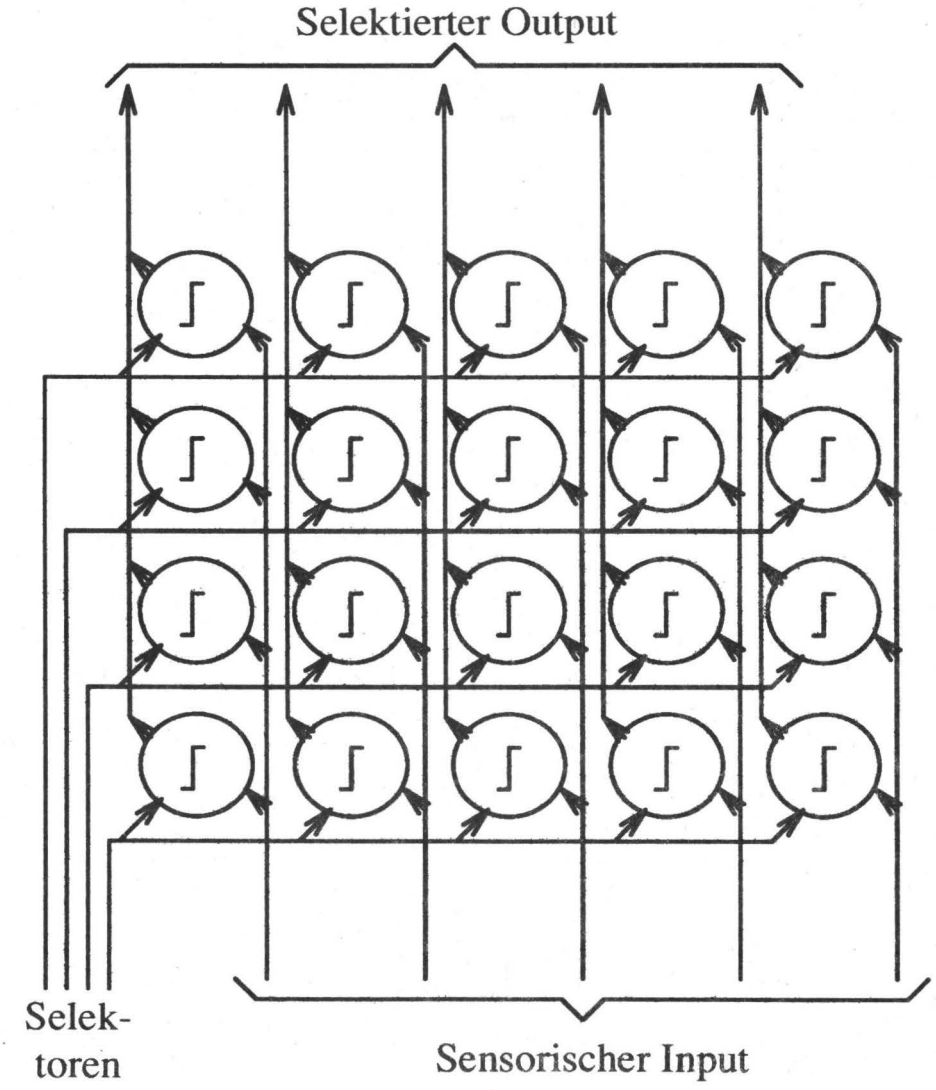


Abbildung 2.2: Beispiel für ein einfaches Netz

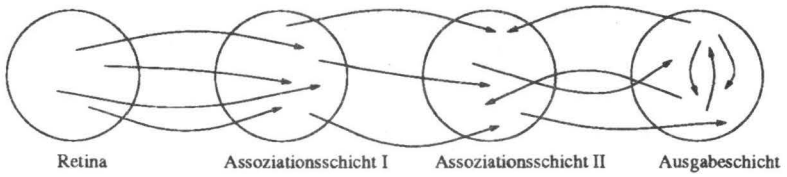


Abbildung 2.3: Schemazeichnung eines Perceptrons

ausschließlich als eine Funktion der sensorischen Eingaben zu beschreiben, die ähnlich einer Telefonzentrale arbeitet. Die Eingaben werden vielmehr mit Erinnerungen, Wissen oder Gedanken kombiniert und dann in einem komplexen, nicht völlig überschaubaren Rahmen in motorische Signale umgesetzt. Das Wissen und die Erinnerungen sind dabei nicht streng lokalisierbar, sondern sind über die ganze Hirnrinde verteilt gespeichert. Zum anderen spricht er von sehr komplexen dreidimensionalen Zellstrukturen, in denen komplexere Signalmuster längere Zeit bestehen können (mit längerer Zeit ist hier ein Vielfaches der synaptischen Verzögerungen gemeint, also eine Zeitspanne von bis zu einer Sekunde etwa). Dies entspricht auch in etwa der Zeitspanne, die für eine komplexere Wahrnehmung nötig ist.

Hebbs Idee der synaptischen Veränderungen war eine bahnbrechende Entdeckung auf dem Gebiet der neuronalen Netze. Sie ermöglichte die Modellierung einfacher, lernender Systeme, die durch Rechenmaschinen experimentell untersucht werden konnten.

2.3.4 Rosenblatt

Ein Modell eines neuronalen Netzes, das 1958 von Frank Rosenblatt in seinem Artikel "The Perceptron" [10] vorgestellt wurde, erregte großes Aufsehen. Es war ein im Aufbau sehr einfaches Netz, daß in der Lage war, Muster zu erlernen. Basierend auf einfachen McCulloch-Pitts-Neuronen und einer Hebbschen Lernregel war es in der Lage, einige psychologische Fähigkeiten des Menschen nachzubilden. Umfangreiche statistische Untersuchungen und Experimente mit simulierten Netzen belegten die erstaunliche Leistungsfähigkeit der Perceptrons.

Perceptrons bestehen im wesentlichen aus drei, manchmal auch aus vier, Schichten von Neuronen:

1. Die 'Retina'; bestehend aus Sensoren, die ein 'Bild' in ein Muster aus feuernden und nicht feuernden Neuronen umwandelt.

2. Eine Assoziationsschicht A_I; die Neuronen der Retina feuern auf die Neuronen der A_I-Schicht über anregende oder inhibitorische Synapsen, wobei die Zuordnung Retinaneuronen-A_I-Neuronen zufällig aber fest ist; ein A_I-Neuron feuert, wenn die Summe der aktiven anregenden oder inhibitorischen Synapsen seinen Schwellwert überschreitet.

3. Eine zweite Assoziationsschicht A_{II} mit Verbindungen von A_I-Neuronen analog den Verbindungen von der Retina zu den A_I-Neuronen.

4. Eine Ausgabeschicht R mit Verbindungen sowohl von A_{II} nach R als auch umgekehrt, außerdem verhindert ein feuerndes Neuron in R das Feuern weiterer Neuronen in R.

In Modellen mit nur drei Schichten fehlt die A_I-Schicht und die Retina ist direkt mit A_{II} verbunden. Eine graphische Darstellung ist in Abbildung 2.3 zu sehen.

Perceptronen lassen sich auf zwei Arten trainieren: mit überwachtem Lernen und mit nichtüberwachtem Lernen. Bei der letztgenannten Variante wird dem Netz ein zu lernendes Muster präsentiert. Daraufhin wird sich in der Ausgabeschicht ein Neuron gegen die anderen durchsetzen. Der Lernalgorithmus sieht nun vor, die synaptischen Gewichte der aktiven Synapsen an diesem Neuron zu verstärken (eine Variante wäre, die Gewichte der inaktiven Synapsen zu verringern). In

Zukunft wird dieses Neuron sich bei Präsentation des gleichen Musters noch besser durchsetzen können, daß Muster wird 'wiedererkannt'. Bei überwachtem Lernen wird 'von außen' vorgegeben, welches Ausgabeneuron aktiv zu sein hat. Durch das anschließende Verändern der Gewichte wird dafür gesorgt, daß das Neuron das Muster in Zukunft selbst erkennt.

Rosenblatt experimentierte mit verschiedenen Arten von Mustern: zum einen mit Mustern aus einer 'idealen' Umgebung, zum anderen mit Mustern aus einer 'differenzierten' Umgebung. Damit meinte er im ersten Fall Muster, die keine erkennbaren Ähnlichkeiten besaßen, z.B. zufällige Verteilungen von Punkten, und im zweiten Fall Klassen von ähnlichen Mustern, wie sie in der Realität des Menschen vorkommen, also z.B. ähnliche geometrische Figuren.

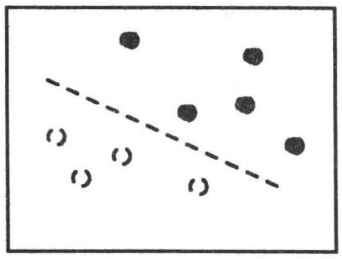
Die wesentlichen Ergebnisse seiner Untersuchungen faßt Rosenblatt wie folgt zusammen:

- Das Perceptron ist in der Lage, Zufallsmuster in bestimmten Umfang zu lernen. Selbst wenn viele unterschiedliche Muster derselben Klasse zugeordnet werden, liefert das Perceptron öfter ein richtiges Ergebnis, als wenn man 'raten' würde.
- In einer solchen 'idealen' Umgebung nimmt bei zunehmender Anzahl gelernter Muster das Verhältnis Treffer/Fehler ab bis zu der Trefferquote bei zufälligem Raten.
- In einer 'idealen' Umgebung gibt es keinen Ansatz zur Verallgemeinerung.
- In einer 'differenzierten' Umgebung nimmt die Trefferquote mit der Anzahl gelernter Muster asymptotisch zu bis zu einem Wert, der über der Zufallstrefferquote liegt. Durch Erhöhen der Anzahl der Neuronen in den Assoziationsschichten läßt sich die Trefferquote beliebig nahe an 100% annähern.
- Die Wahrscheinlichkeit, daß das Perceptron in einer solchen Umgebung ein Muster richtig klassifiziert, das es vorher nicht gelernt hat, nähert sich derselben Asymptote wie die Wahrscheinlichkeit daß bekannte Muster richtig klassifiziert werden, falls das unbekannte Muster 'ähnlich genug' einem bekannten Muster ist (die Umschreibung 'ähnlich genug' wird bei Rosenblatt durch eine mathematische Ungleichung genauer definiert).
- Die Performance des Systems kann verbessert werden, in dem der Retina ein Präprozessor vorgeschaltet wird, z.B. um Konturen eines Musters herauszuarbeiten.
- Das Gedächtnis des Perceptrons ist über das ganze Netz verteilt in dem Sinne, daß ein Entfernen einiger Assoziationsneuronen keinen Einfluß auf das Erkennen einzelner Muster hat, sondern auf die Gesamtperformance des Systems.
- Obwohl das Perceptron gut in der Lage ist, Eigenschaften von Mustern zu klassifizieren, kann es keinen Zusammenhang zwischen zeitlich auseinanderliegenden Mustern erkennen.

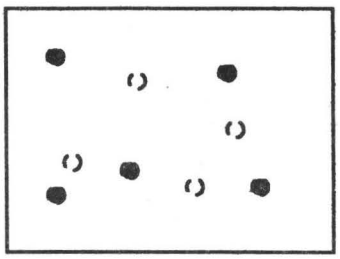
Als weiteren Vorteil sieht er die Tatsache, daß das Verhalten des Systems aufgrund von physikalischen Größen voraussagen läßt. Es handelt sich dabei um

- die Anzahl anregender Verbindungen pro Assoziationsneuron,
- die Anzahl inhibitorischer Verbindungen pro A-Neuron,
- den mittleren Schwellwert der A-Neuronen,
- den Anteil an Ausgabeneuronen, mit denen ein A-Neuron verbunden ist,
- die Anzahl A-Neuronen im System, sowie
- die Anzahl der Ausgabeneuronen im System.

Obwohl das Perceptron eine Euphorie unter Mathematikern, Hirnforschern und Psychologen auslöste, war Rosenblatt sich einiger Unzulänglichkeiten seines Modells durchaus bewußt. Auf jeden Fall war das Perceptron die erste erfolgreiche Anwendung von Hebb's Lernregel, und die statistisch verifizierten und vorhersagbaren Ergebnisse, die Rosenblatt's Experimente lieferten, ließen auf eine große Zukunft der Perceptrons hoffen.



linear trennbare Punkte



nicht linear trennbare Punkte

2.3.5 Block

In den Jahren nach Rosenblatt's Artikel über Perceptrons [11] wurde eifrig mit verschiedenen Methoden nach Weiterentwicklungen geforscht. Dabei wurden große Fortschritte in der mathematisch-theoretischen Analyse solcher Netze gemacht. Außerdem wurden Perceptrons aus Hardware aufgebaut, sowie umfangreiche Computersimulationen von Perceptrons durchgeführt. Eine der Hardwarelösungen war das "MarkI"-Perceptron, das von Rosenblatt und einigen Mitarbeitern aufgebaut wurde. Es handelte sich dabei um ein Modell mit 400 photoempfindlichen Zellen in einer 20x20-Matrix, die über 512 A-Neuronen acht Ausgabeneuronen erreichten. Einige der Ergebnisse dieser Versuche wurden 1962 von Block vorgestellt.

Allem voran stellt er dabei die Erkenntnis, daß das Perceptron keinen Anspruch darauf erhebt, die Arbeitsweise des Gehirns widerzuspiegeln. Es ist lediglich ein hinreichend einfaches Modell, das aus Komponenten ähnlich den Nervenzellen aufgebaut ist. Desweiteren gab es bis dato keine befriedigenden Ergebnisse mit anders aufgebauten Modellen, es konnte kein 'Speicherorgan' im menschlichen Körper gefunden werden.

Dieser weisen Erkenntnis folgt eine mathematische Analyse der Lernphase eines Perceptrons, die in dem bekannten 'Perceptron Convergence Theorem' gipfelt. Dieses Theorem war der erste mathematische Beweis dafür, daß ein Perceptron eine Repräsentation tatsachlich lernt, falls es eine solche gibt.

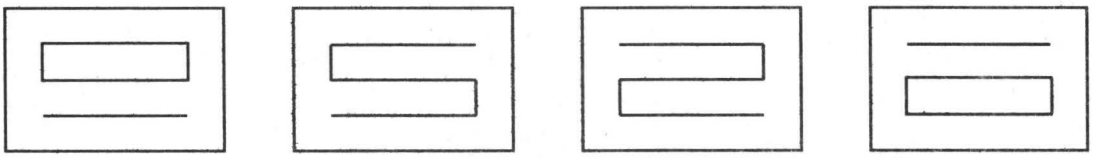
Die Bedingung 'falls es eine solche gibt' basiert dabei auf der ebenfalls sehr wesentlichen Erkenntnis, daß die zu einer Klasse gehörenden Muster linear trennbar sein müssen von den nicht zu dieser Klasse gehörenden (dies ist einer der Punkte, auf die Minsky und Papert später in ihrem Buch 'Perceptrons' genauer eingingen, s.u.).

Zu den experimentell ermittelten Ergebnissen gehören unter anderem Statistiken zur Performance bei 'verrauschten' Mustern oder bei einem 'Lehrer', der Fehler macht. Es wurde z.B. Mark-I mit den 26 Buchstaben des Alphabets in einer festen Position trainiert. Das Perceptron erreichte dabei nach etwa 15 Präsentationen pro Buchstabe eine Erfolgsquote von fast 100%.

In einem weiteren Versuch wurde auf unterschiedliche Lernverfahren eingegangen: das Netz wurde mit jeweils acht Buchstaben trainiert, einmal mit überwachtem, einmal mit nicht-überwachtem Lernen. Im ersten Fall näherte sich die Erfolgsquote einem Wert unter 75%, im zweiten Fall den gewünschten 100%.

Bei Versuchen, Mark-I auf die Unterscheidung von 'X' und 'E' zu trainieren, wobei zum einen ein 'Störungsrauschen' in Form zufällig belichteter Punkte die Muster veränderte, zum anderen gab der Trainer in 30% falsche Antworten vor. Erwartungsgemäß lag die Erfolgsquote bei verrauschten Bildern unter 100%. Erstaunlicherweise übertraf das Verhalten des Netzes bei weitem die Performance des falsch arbeitenden Trainers selbst, es wurde sogar fast ein ebensogutes Ergebnis erzielt wie mit einem perfekten Trainer.

In einem nächsten Schritt wurden aus dem trainierten Netz A-Neuronen entfernt. Die Performance von Mark-I veränderte sich selbst bei einem Ausfall von 50% der A-Neuronen nicht. Auch als nur noch 1/8 der ursprünglichen Zahl vorhanden war, lag die Trefferquote des Perceptrons noch deutlich über der Zufallstrefferquote von 50%.



C-00 C-01 C-10 C-11

Abbildung 2.4: Beispiele für 'Connectedness'

Die Experimente mit Mark-I bewiesen die erstaunlichen Fähigkeiten von Perceptrons in Bezug auf Fehlertoleranz und Verallgemeinerung. Mittels der verbesserten mathematischen Analyse und des Konvergenztheorems wurde eine Basis für weitere Untersuchungen über das Lernverhalten von neuronalen Netzen geschaffen.

2.3.6 Minsky & Papert

Mit dem 1969 von M. Minsky und S. Papert erschienenen Buch "Perceptrons" [12] gab es einen entscheidenden Knick in dem steilen Aufstieg der neuronalen Netze. Sicherlich ist dieses Buch nicht allein verantwortlich dafür gewesen. Vielmehr wurde seit etwa zehn Jahren intensiv im Bereich der Perceptrons und verwandter Netze geforscht, jedoch ohne daß entscheidend neue Resultate erbracht worden wären. Die Presse (und auch einige Wissenschaftler) warteten mit immer neuen Sensationsmeldungen und Voraussagen auf, die von der Forschung nicht eingehalten werden konnten. Die bereits von Rosenblatt und Block erkannten Unzulänglichkeiten der Perceptrons in Hinsicht auf die Mächtigkeit der mit ihnen berechenbaren Probleme erwiesen sich als schwerwiegend. Minsky und Papert gelang es oft, mit einfachen, geometrischen Argumenten nachzuweisen, daß Perceptrons nicht in der Lage sind, Probleme zu berechnen, die für das menschliche Hirn trivial sind. Auf der anderen Seite erwiesen sich Ansätze der Künstlichen Intelligenz (KI) im Bereich der Expertensysteme als überaus erfolgreich. Der Kosten für praktisch verwendbare Systeme dieser Art waren wesentlich geringer, als der gewaltige Aufwand, der für die vielen Verbindungen selbst kleiner Perceptrons betrieben werden mußte.

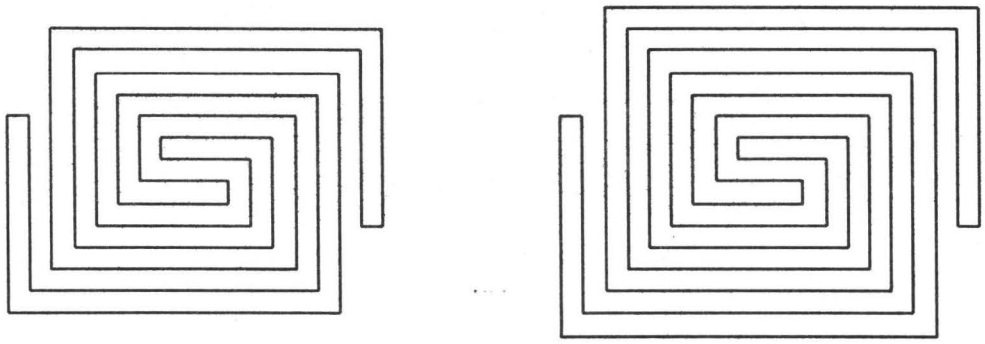
Minsky und Papert betrachten Perceptrons nicht als Musterklassifikatoren, sondern einfach als Prädikate, die einem Muster einen Ausgabewert wahr oder falsch zuordnen. Aufbauend auf dieser Betrachtungsweise unterteilten sie zunächst das mehrschichtige Perceptron in zwei Schichten: einer Schicht von lokalen Prädikaten (entsprechend der Assoziationsschicht) und der Ausgabeschicht, die aus den lokalen Prädikaten ein zusammengesetztes berechnet. Ein Perceptron, das ein beliebiges Prädikat berechnet, könnte auf triviale Weise derart gebaut werden, daß jedes Muster, auf das das Prädikat zutrifft, durch ein globales Prädikat in der Assoziationsschicht identifiziert wird. Dieses Verfahren würde jedoch den Aufwand für das Perceptron mit der Größe der Retina exponentiell wachsen lassen. Aus diesem Grund ist es sinnvoll, sich in der Assoziationsschicht auf lokale Prädikate zu beschränken.

Diese Einschränkung ist jedoch sehr essentiell: Minsky und Papert zeigten, daß ein derart eingeschränktes Perceptron bestimmte Prädikate nicht berechnen können, z.B. das Prädikat 'Connectedness' (d.h. ob eine Figur gezeichnet werden kann ohne den Stift abzusetzen).

Wie man an den Beispielen leicht sieht, ist dieses Problem äquivalent mit dem logischen Exklusiv-Oder (XOR). Zusammenfassend stellen die Autoren vier Thesen auf:

1. Die Idee, geometrische Objekte als n-dimensionale Vektoren zu beschreiben, führt zum Verlust der geometrischen Eigenheiten dieser Objekte. Eine darauf aufbauende Theorie kann nicht viel mehr tun, als die mit einem Perceptron berechenbaren Prädikate aufzuzählen.

2. Die Größe der Gewichte der einzelnen Verbindungen führt dazu, daß zur Speicherung aller Gewichte eines Perceptrons mehr Speicher notwendig sein kann, als zur Speicherung aller



A B

Abbildung 2.5: Nichttriviale Beispiele für 'Connectedness'

von diesem Perceptron richtig klassifizierbaren Muster. Dies gilt in besonderem Maße für große Perceptrons.

3. Die für die Lernphase benötigte Zeit wächst überexponentiell mit der Größe des Perceptrons.

4. Um ein zusammengesetztes Prädikat aus vielen lokalen Prädikaten zu berechnen wird ein immenser Overhead produziert. Sehr viele der lokalen Prädikate müßten nicht berechnet werden, wenn aufgrund einiger weniger berechneter Prädikate entschieden werden könnte, welche weiteren Prädikate noch bis zur Entscheidungsfindung herangezogen werden müssen.

Nach dem Erscheinen von "Perceptrons" verwandelte sich die Hoffnung, die viele Wissenschaftler in neuronale Netze gesetzt hatten, in Resignation. Verbunden damit war natürlich eine Kürzung der entsprechenden Forschungsetats, und das allgemeine Interesse an 'künstlichen Gehirnen' ging verloren.

Doch trotz des niederschmetternden Erfolges von Minsky's und Papert's Thesen dienen Perceptrons auch heute noch zur Erklärung bestimmter psychologischer Phänomene. So ist z.B. auch der Mensch nicht immer in der Lage, das Prädikat 'Connectedness' mit einem Blick zu berechnen. In Abbildung 2.5 kann man erst durch Verfolgen der Linien feststellen, welche der Figuren 'Connected' ist.

2.4 Moderne Varianten

2.4.1 Backpropagation

Der 1980 entwickelte Backpropagation-Algorithmus [3] verhalf den neuronalen Netzen zu neuem Ansehen. Er ermöglichte es, auch die synaptischen Gewichte in den über der Ausgabeschicht liegenden Schichten gezielt zu verändern um ein verbessertes Verhalten des Netzes zu erreichen (beim Perceptron wurden die Gewichte der Assoziationsschicht zufällig festgelegt und nicht verändert). Dafür konnten allerdings keine einfachen Schwellwertneuronen mehr eingesetzt werden, sondern es war eine stetige Übergangsfunktion notwendig.

Eines der bekanntesten Beispiele für ein Backpropagation-Netz dürfte das 1986 entwickelte NETtalk sein, das in wenigen Stunden lernte, Wörtern Phoneme zuzuordnen. Auf diese Weise lernte es, Wörter zu 98% richtig auszusprechen. Auch Wörter, die NETtalk nie zuvor gesehen

hatte, wurden mit dieser Trefferquote richtig ausgesprochen. Damit übertraf dieses einfache Netz aus nur etwa 300 Neuronen das DECtalk-System, das mit erheblich höherem Aufwand aus einer Reihe von linguistischen Regeln und einem umfangreichen Lexikon an Ausnahmen Phoneme mit einer Trefferquote von 95% generierte.

Der Algorithmus wurde und wird in etlichen Varianten in sehr vielen Anwendungen erfolgreich eingesetzt.

2.4.2 Hopfield-Netze

Diese von John Hopfield Anfang der 80er Jahre entwickelten Netze unterscheiden sich von den bisher betrachteten durch ihre Struktur und ihr Verhalten. Sie bestehen im wesentlichen aus nur einer Schicht von Neuronen, die jeweils paarweise symmetrisch miteinander verbunden sind. Diese Verbindungen lassen sich einfach durch eine Gewichtsmatrix beschreiben, deren Diagonale nur Nullen enthält. Wird dem Netz ein Muster präsentiert, so werden iterativ über die Verbindungen neue Muster berechnet, bis ein stabiler Zustand im Netz erreicht ist. Ein Lernen durch eine Hebbsche Lernregel findet nicht statt, vielmehr muß die Gewichtsmatrix von außen so vorgegeben werden, daß die gewünschten stabilen Zustände existieren.

Das Prinzip basiert auf der Idee, daß das Netz einen Zustand minimaler Energie annimmt, indem es in jedem Iterationsschritt seine Energie verringert, ähnlich einem physikalischen System.

Hopfield-Netze lassen sich gut als Assoziativspeicher verwenden: ein Teil des gewünschten Musters wird vorgegeben, und das Netz reproduziert mit mehreren Iterationen das komplette Muster. Das Problem, die richtige Gewichtsmatrix vorzugeben, ist durch entsprechende einfache Algorithmen gelöst. Das Hinzufügen weiterer Klassen von Mustern ist iterativ möglich.

2.4.3 Probabilistische Netze

Ein weiterer Ansatz wurde 1984 formuliert. Dabei feuert ein Neuron nicht deterministisch nach fest vorgegebenen Regeln, sondern mit einer bestimmten Wahrscheinlichkeit [13]. Diese Wahrscheinlichkeit hängt wiederum davon ab, wie viele andere Neuronen dieses Neuron anregen.

Der Lernvorgang benötigt hier zwei Schritte: in einer ersten Phase wird das Netz mit einer Hebbschen Lernregel trainiert, ähnlich wie bereits besprochene Netze. Im zweiten Schritt läuft das Netz ohne Eingaben (Neuronen können auch ohne äußere Reize feuern), wobei auftretende Reize durch eine Anti-Hebb-Lernregel unterdrückt werden. Schon Hopfield hat eine Art 'Traumphase' verwendet, um in seinen Netzen unerwünschte lokale Energieminima zu unterdrücken. Dieses Prinzip wird hier ebenfalls verwendet, es ist sogar nötig um akzeptable Ergebnisse zu erzielen.

2.5 Zusammenfassung

In den vorangegangenen Abschnitten wurde versucht, anhand einiger markanter Punkte in der Geschichte der Hirnforschung und der Forschung an künstlichen neuronalen Netzen die Gesamtentwicklung deutlich zu machen. Natürlich waren es nicht nur Hebb, Rosenblatt und Minsky, die diese Entwicklung vorantrieben. Alle Namen zu nennen und alle Modelle vorzustellen reicht wohl der Rahmen dieses Seminars nicht aus. Aber ich hoffe, der Leser kann sich ein Bild davon machen, wie umfangreich dieses Gebiet ist, und wie wenig wir bisher davon wissen. Es bleibt abzuwarten, ob die relativ neuen Ansätze sich durchsetzen können, oder ob es eine erneute Schaffenspause geben wird wie 1969 nach Minky's und Papert's Buch.

2.6 Literatur

[1] Rene Descartes. *Meditationes de prima Philosophia.* 1641.

[2] Aristoteles. *De Memoria et Reminiscentia.* (Translated by R.Sorabji). Dawn University Press, Providence, RI. 400 v. Chr.

[3] Colin Johnson und Chappell Brown. *Cognizers.* John Wiley and Sons, New York. 1988.

[4] William James. *Principles of Psychology.* Holt, New York. 1890.

[5] Warren S. McCulloch und Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5:115–133.* 1943.

[6] Warren S. McCulloch und Walter Pitts. How we know universals. *Bulletin of Mathematical Biophysics, 9:127–147.* 1947

[7] Julia Apter. The projection of the retina on the superior colliculus of cats. *Journal Neurophysiological, 8:123–134.* 1945.

[8] Julia Apter. Eye movements following strychninisation of the superior colliculus of cats. *Journal Neurophysiological, 9:73-85.* 1946.

[9] Donald O. Hebb. *The Organization of Behaviour.* John Wiley, New York. 1949.

[10] Frank Rosenblatt. The perceptron. *Psychological Review, 65:386–408.* 1958.

[11] H. D. Block. The perceptron: a model for brain functioning. *Reviews of Modern Physics, 34:123–135.* 1962.

[12] Marvin Minsky und Seymour Papert. *Perceptrons.* MIT Press, Cambridge, MA. 1969.

[13] Rüdiger Brause. *Neuronale Netze.* Teubner-Verlag, Stuttgart. 1991.

Kapitel 3

Hopfield Netze und Boltzmann Maschine

Christoph Keller

Übersicht

Im folgenden sollen zwei rückgekoppelte neuronale Netzwerke vorgestellt werden und Unterschiede sowie Gemeinsamkeiten zwischen den Modellen aufgezeigt werden. Es handelt sich hiebei um das Hopfield Modell und die Boltzmann Maschine. Bei beiden handelt es sich um rückgekoppelte Netzwerke, wobei sich die Boltzmann Maschine mehr auf stochastischen Prinzipien basiert.

3.1 Einleitung

Im folgenden sollen zwei rückgekoppelte, neuronale Netze vorgestellt werden, das Hopfield Modell und die Boltzmann Maschine. Beide Modelle sind sich sehr ähnlich, wobei die Boltzmann Maschine mehr auf statistischen Methoden basiert. Das Hopfield Modell resultierte aus dem verstärkten Interesse der theoretischen Physik nach mathematischen Modellen, um ihre physikalisch-mathematischen Modelle besser zu behandeln. John Hopfield war einer der ersten, der den Zusammenhang zwischen magnetischen Anomalien (Spingläsern) und neuronalen Netzen aufzeigte und das Verhalten solcher magnetischen Effekte in einem neuronalen Netz mit einer Energiefunktion modellierte. Er bediente sich dabei eines sehr einfachen Neuronenmodells, das schon von Little und Shaw 1978 eingefhrt wurde, vereinfachte es an einigen Stellen aber. Im folgenden wird zunächst das sequentielle Hopfield Modell vorgestellt.

3.2 Das Hopfield Modell

3.2.1 Struktureller Aufbau

Hopfield bediente sich bei seinem Modell sehr einfacher Neuronen. Diese Neuronen können nur zwei Zustände annehmen, nämlich -1 und 1. Diese Zustandwerte kommen zum einen aus dem physikalischen Ansatz, den Hopfield seinerzeit verfolgte, zum anderen vereinfachen sie auf die mathematische Betrachtung bisweilen. Die Eingabe eines jeden Neurons N_i setzt sich zusammen aus extrenen Eingabe I_i und Eingaben von anderen Neuronen (Vergleiche Abbildung 3.1) .

Die Neuronen sind also vollständig untereinander vernetzt. Die Verbindungen zwischen den Neuronen sind gewichtet, das Gewicht der Verbindung zwischen Neuron i und j trage die Bezeichnung w_{ij}. Die Gesamteingabe eines Neurons N_i läßt sich also mit der folgenden Gleichung beschreiben:

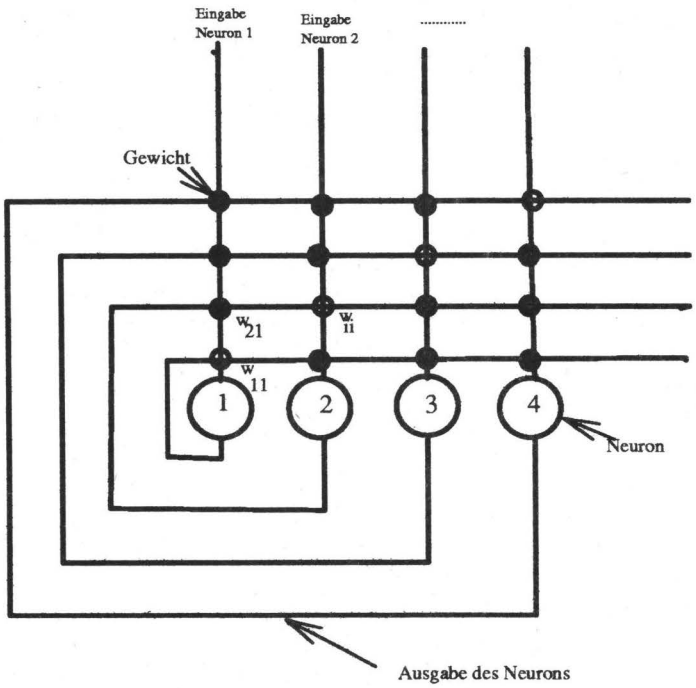


Abbildung 3.1: Hopfield Modell

$$H_i = \sum_{ij} w_{ij} S_j + I_i$$

wobei $S_j \in \{-1, 1\}$ den Zustand des j - ten Neurons bezeichne. Das Neuron N_i ermittelt nun seinen eigenen neuen Zustand nach der folgenden Bedingung :

$$S_i = \begin{cases} +1 & H_i \geq \text{Schwelle}_i \\ -1 & H_i < \text{Schwelle}_i \end{cases}$$

Setzt man die Schwelle für jedes Neuron gleich 0, so läßt sich die obige Gleichung etwas einfacher als

$$S_i = sgn(\sum_j w_{ij} S_j + I_i)$$

In Abbildung 3.2 ist das Ausgabeverhalten eines Neurons dargestellt. Die gepunktete Linie sei ein anderes Ausgabeverhalten mit einem Schwellenwert. Man kann in diese Gleichung noch einen Schwellenwert θ miteinbauen, der in bestimmten Fällen die Speicherkapazität des Netzes verbessern kann. Jedes Neuron berechnet seinen Zustand zu einer zufälligen Zeit neu. Es nimmt die aktuellen Eingaben und berechnet daraus seinen Zustand entsprechend der Gleichung aus und wechselt entsprechend. Diesen stochastischen Prozeß führt jedes Neuron im Mittel W-mal durch, wobei die Zeit in der einen Neuron andere Neuronen abfragt unabhängig ist von der Zeit in der es selbst abgefragt wird. Daraus resultiert natürlich ein asynchroner Mechanismus bei der Zustandsänderung des gesamten Netzes. Hopfield vereinfachte hier das echt parallele Modell von Little und Shaw dahingehend, daß er nur einen Zustandsübergang zu einer Zeit für ein Neuron zuließ. Dies läßt sich natürlich besser auf sequentiellen Rechner modellieren, beeinflußt aber das Verhalten des Modells erheblich.

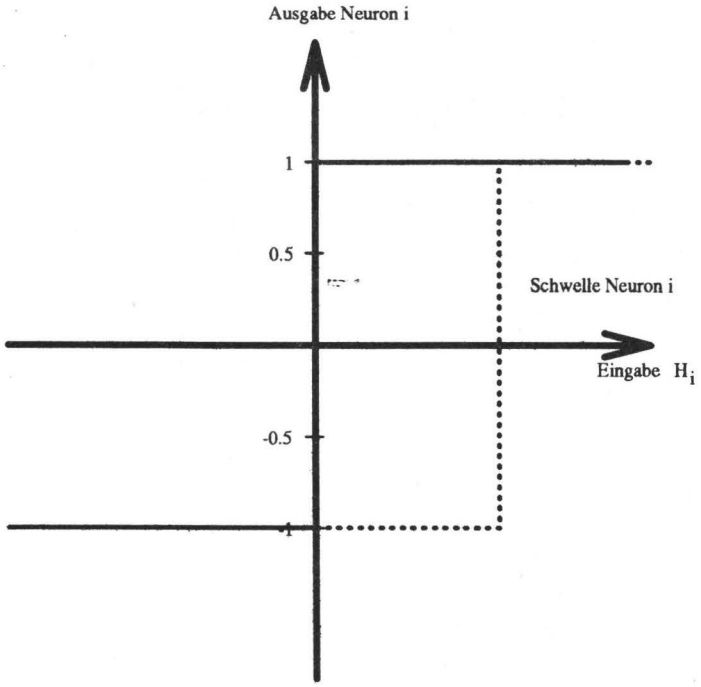


Abbildung 3.2: Ausgabeverhalten eines Neurons

3.2.2 Speichern von Mustern

Hopfield Netze eignen sich hervorragend als Assoziativspeicher. Um ein oder mehrere Muster zu speichern, müssen die Gewichte so eingestellt werden, das sich das Netz bei gegebener (gestörter) Eingabe an sie erinnert. Die Gewichte müssen immer vorher, passend zu den zu speichernden Mustern berechnet werden. Um die Wahl der Gewichte besser zu verstehen, nehmen wir an, wir wollen ein Muster x in einem Netz speichern. Damit das Muster stabil gespeichert ist müssen die Gewichte offensichtlich folgender Bedingung genügen:

$$sgn(\sum_j w_{ij} S_j) = S_i \qquad \forall i \in \{[1, \ldots, n\}$$

Dies ergibt sich aus der Tatsache, daß dann im Netz von den Neuronen keine Zustandswechsel mehr vorgenommen werden, was sich leicht anhand der Zustandswechselfunktion für die Neuronen zeigen läßt. Wobei der Zustand des Netzes, also die S_i die x_i natürlich wiederspiegeln sollen. Die w_{ij} müssen proportional zu $S_i S_j$ sein, setzt man

$$w_{ij} = \frac{1}{N} S_i * S_j$$

wobei N die Anzahl der Neuronen im Netz sei, so erhält man Gewichte, die das gewünschte leisten, daß heißt das Netz speichert ein beliebiges Muster. Man kann sich die zu speichernden Muster als Attraktoren vorstellen, die eine gegebene Eingabe zu sich hinziehen. In Abbildung 3.3 ist dies exemplarisch für ein Muster und sein Inverses dargestellt. Zu jedem gespeicherten Muster existiert immer noch ein sogenanntes inverses Muster, das sich von dem gewünschten Muster dadurch unterscheidet, daß bei ihm alle Bits das entgegengesetze Vorzeichen tragen.

Es bleibt noch anzumerken, daß bei Netzeingaben, bei denen weniger als die Hälfte der Bits vom Orginalmuster abweichen, vom Netz richtig erkannt werden, da die negativen Bits von den anderen, bei der Eingabefunktion $H_i = \sum_j w_{ij} S_j$ überstimmt werden. Das Netz wird also zu

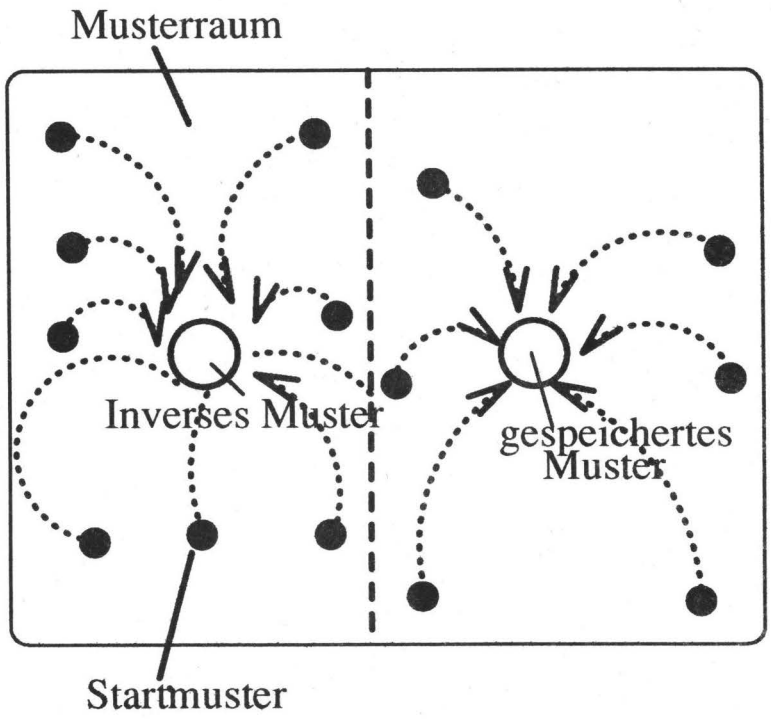


Abbildung 3.3: Muster als Attraktor im Raum der möglichen Muster

dem richtigen Muster hingezogen. Sind hingegen mehr als die Hälfte falsch, so landet das System bei dem sogenannten inversen Muster, welches gerade die umgekehrten Vorzeichen hat. Um nun mehrere Muster zu speichern, summiert man einfach bei jedem Gewicht über alle Muster und erreicht so das gewünschte Verhalten.

$$w_{ij} = \frac{1}{N} \sum_{\chi=1}^{p} S_i^{(\chi)} S_j^{(\chi)}$$

wobei p hier die Anzahl aller zu speichernden Muster bezeichne.

Das heißt, man superponiert alle Netzeingaben. Damit erhält man ein Gleichungssystem, das mindestens so viele Gleichungen wie Unbekannte hat, welches auch mit konventienellen Mitteln gelößt werden kann. Die obige Gleichung wird auch als generalisierte Hebb'sche Lernregel bezeichnet.

Praktisch kann man die Gewichte folgendermaßen berechnen. Sei $x = \{x_1, x_2, ..., x_m\}$ eine Menge mit m n-dimensionalen binären Vektoren, die gespeichert werden sollen. Um ein neuronales Netz zur Speicherung der Vektoren aufzubauen, gehen wir wie folgt vor: Zu jedem Vektor x_i bilden wir eine (n x n) - Matrix W_i nach folgendem Schema :

$$W_i = x_i * (x_i)^T - I_n$$

, wobei I_n die Einheitsmatrix ist. Um die Hopfield Matrix für die Verbindungsgewichte zu erhalten werden alle W_i 's aufsummiert, daß heißt

$$W = \sum_{i=1}^{n} W_i = \sum_{i=1}^{n} (x_i * (x_i)^T - I_n)$$

dabei sind die Einträge w_{ij} dieser Matrix die Gewichte der Verbindungen zwischen den i - ten und j - ten Neuron. Das heißt also, die zu speichernden Muster sind die Eigenvektoren der Gewichtsmatrix W.

Beispiel:

Sei $x = \{(1,1,1,1,1),(1,-1,-1,1,-1),(-1,1,-1,-1,-1)\}$ es ergeben sich somit die W_i als

$$\mathbf{W_1} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\mathbf{W_2} = \begin{pmatrix} 0 & -1 & -1 & 1 & -1 \\ -1 & 0 & 1 & -1 & 1 \\ -1 & 1 & 0 & -1 & 1 \\ 1 & -1 & -1 & 0 & -1 \\ -1 & 1 & 1 & -1 & 0 \end{pmatrix}$$

$$\mathbf{W_3} = \begin{pmatrix} 0 & -1 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 & -1 \\ 1 & -1 & 0 & 1 & 1 \\ 1 & -1 & 1 & 0 & 1 \\ 1 & -1 & 1 & 1 & 0 \end{pmatrix}$$

$$\text{und somit } \mathbf{W} = \sum_{i=1}^{3} W_i = \begin{pmatrix} 0 & -1 & 1 & 3 & 1 \\ -1 & 0 & 1 & -1 & 1 \\ 1 & 1 & 0 & 1 & 3 \\ 3 & -1 & 1 & 0 & 1 \\ 1 & 1 & 3 & 1 & 0 \end{pmatrix}$$

Damit haben wir jetzt die Gewichte der Verbindungen zwischen den Neuronen berechnet. Mit der Matrix W kann man aber auch zu einer gegebenen Eingabe berechnen in welche Zustände das Netz schließlich konvergieren wird. Man multipliziert dazu einfach einen n - dimensionalen Eingabevektor mit der Matrix W und betrachtet dann die Vorzeichen der Einträge im Ergebnisvektor, daß heißt man transformiert den Ergebnisvektor e wieder in die Form $e_i \in \{-1,1\} \quad \forall i$. Dann entscheidet man, welches Bit gegenüber dem alten Vektor geändert wird und wiederholt dieses bis keine Änderungen mehr eintreten.

Nehmen wir beispielsweise den Vektor x'= (1 -1 -1 1 1) der den Hamming - Abstand 1 zum Vektor x_2 hat. Aus Wx' erhalten wir $x'' = (+4,-3,4,4,-2) \Rightarrow sgn(x'') = (1,-1,1,1,-1)$ gegenüber dem Ausgangsvektor x' haben sich die Bits 3 und 5 geändert. Noch dem asynchronen Updatemechanismus können jetzt entweder das 3. oder das fünfte Neuron Ihren Zustand wechseln. Wechselt das 3. , so erreichen wir im nächsten Schritt den Vektor x_1 als stabilen Zielzustand, wohingegen ein Wechsel des 5. in den Zielzustand x_2 führt. Hier liegt eines der Probleme des Hopfield Ansatzes.

3.2.3 Speicherkapazität eines Hopfield Netzes

Die Speicherkapazität eines Hopfield Netzes hängt von vielen Faktoren ab. Zum einen natürlich stark von den zu speichernden Mustern. Hat man nur orthogonale Muster zum Abspeichern, so kann man mit m Neuronen gerade m Muster der Länge m abspeichern. Wobei man maximal 1 Bit pro unabhängiges Gewicht speichern kann. Für einen Abruf eines gespeicherten Musters gilt, daß das System nur dann zuverlässig zu ihm konvergiert, wenn die Abweichung zur Eingabe nicht größer als 13,8 % ist, dh wenn nicht mehr als 13,8 % der Bits unterschiedlich sind.

Für nicht orthogonale Muster gelten weit schlechtere Werte. hat man m Muster zum Speichern, so braucht man wesentlich mehr Neuronen als beim orthogonalen Fall. Man kann zeigen, daß $m = \frac{n}{(4*ln(n))}$, wobei n die Anzahl der Neuronen ist.

Läßt man zu das die Zustände die sich im System nach einer Zeit einschwingen nur im Mittel den gewünschten Zuständen entsprechen, daß heißt man erlaubt auch fehlerhafte Zustände (übersprechen der Neuronen), so verschlechtert sich das Speicherverhalten weiter.. Dies tritt vor allem beim Speichern zufälliger Muster auf. Man kann zeigen, daß die Speicherbelegung $\alpha = \frac{m}{n}$ für solche Muster bei $\alpha = 0.138$ eine 50 % Fehlerrate beim Auslesen produziert und es somit sinnlos ist so viele Muster zu speichern. Desweiteren hängt die Kapazität auch von der Kodierung des Problems ab. Man kann zeigen, daß der symmetrische Wertebereich $\{-1, 1\}$ im allgemeinen bessere Resultate liefert als der unsymmetrische mit $\{0, 1\}$. Damit die Muster auch zuverlässig erkannt werden, dürfen die gestörten Eingaben auch nur maximal die halbe Länge des Bitvektors als Hamming - Abstand von dem gewünschten Zielvektor entfernt liegen.

3.2.4 Die Energiefunktion

Die wesentliche Neuerungen die Hopfield einführte war die Idee einer Energiefunktion. Er betrachte den Zustand des System als Energie, die es dann, um bestimmte Zustände zu erreichen (zum Beispiel ein gespeichertes Muster zu erkennen), zu minimieren gilt. Dabei kann man sich die zu speichernden Muster als (lokale) Minima dieser Funktion vorstellen. Eine Funktion, die die "Energie" eines Netzes beschreibt ist zum Beispiel

$$H = -\frac{1}{2}\sum_{ij} w_{ij} S_i S_j$$

Diese Doppelsumme beschreibt den Energiezustand des Systems in sehr einfacher und effektiver Weise. Der Faktor $\frac{1}{2}$ kommt daher zustande, daß die Eingaben aufgrund der symmetrischen Verbindungstruktur doppelt gezählt werden. Die Terme für i = j tragen nach der Definition der Zustände ($S_i^2 = 1$) nur einen konstanten Faktor zur Summe bei und um die Rechnung zu vereinfachen setzt man $w_{ii} = 0$, so daß der konstante Beitrag verschwindet. Desweiteren haben die Rückkopplungen der Neuronen auf sich selbst noch andere negative Wirkungen, doch dazu später mehr.

Die zentrale Eigenschaft einer Energiefunktion ist, daß sie in einem dynamischen Prozeß ständig kleiner wird oder gleich bleibt. Ändert sich die Energie nicht mehr, so hat das System einen stabilen Zustand erreicht. Man kann mittels der Energiefunktion zeigen, daß das Netz immer in einen stabilen Zustand konvergiert.

Da die Energiefunktion immer konvergiert, kann man die gespeicherten Muster als Attraktoren interpretieren, die einen gegebenen Eingabevektor zu sich hinziehen. Diese Attraktoren kann man als Täler im verlauft der Enegiefunktion (siehe Abbildung 3.4) anssehen in die die Eingabemuster hineinlaufen. Es existiert zu einem gespeicherten Muster wie schon erwähnte immer noch das sogennante inverse Muster im Netz, welches dann in einem anderen Tal angesiedelt ist. Muster sind also so etwas wie lokale Energieminima, in die eine gegebene Eingabe hineinläuft. In neuronalen Netzten existiert immer eine solche Enegiefunktion, wenn die Verbindungsgewichte symmetrisch, dh $w_{ij} = w_{ji}$ sind. Desweiteren müssen alle Gewichte w_{ii} einen Wert größer oder gleich Null tragen. Für den symmetrischen Fall kann man die Funktion auch als folgende Gleichung schreiben

$$H = \sum_i w_{ii} S_i^2 - \sum_{i \neq j} w_{ij} S_i S_j$$

wobei hier $i \neq j$ alle unterschiedlichen Paare von i und j bedeuten soll. Man kann jetzt leicht zeigen, daß die dynamische Regel, nach der die Neuronen ihren eigenen Zustand ändern, diese Enegiefunktion in jedem Schritt nur verkleinern oder gleich lassen kann. Ist nämlich $S_i' = sgn(\sum_j w_{ij} S_j)$ der neue Zustand des Neurons i. Es sind nun zwei Fälle möglich

1. Fall : $S_i' = S_i$ In diesem Fall bleibt die Gesamtenergie gleich.

2. Fall : $S_i' = -S_i$ Es ergibt sich

$$H' - H = \left(\sum_i w_{ii} S_i^2 - \sum_{i \neq j} w_{ij} S_i' S_j\right) - \left(\sum_i wii S_i^2 - \sum_{i \neq j} w_{ij} S_i S_j\right)$$

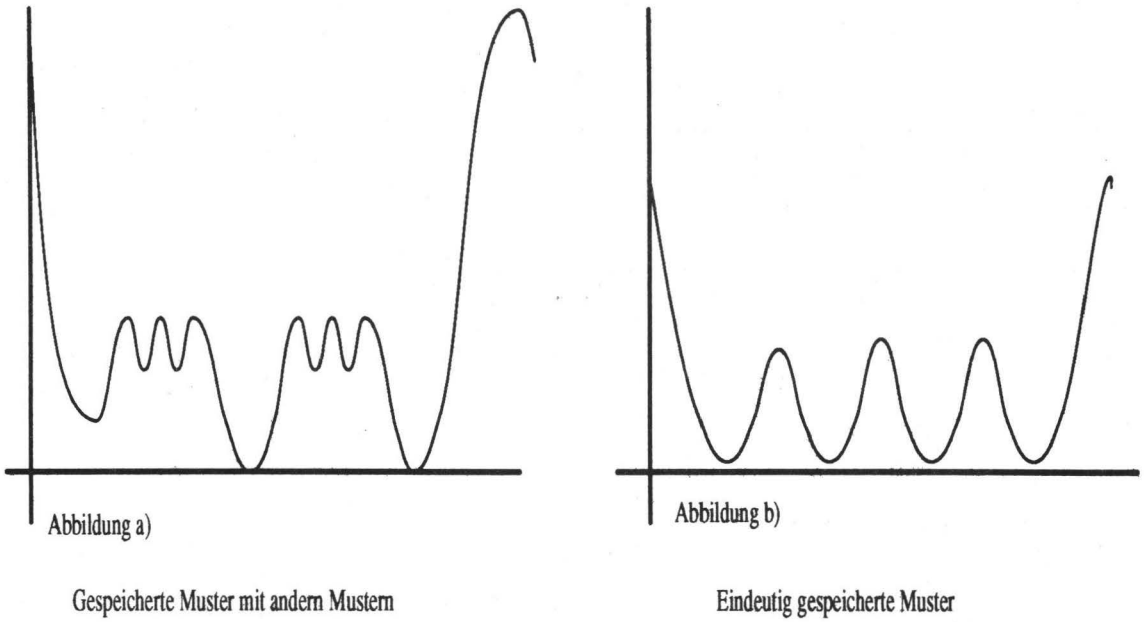


Abbildung 3.4: Energieminima der Enegiefunktion bei gespeicherten Mustern (schematisch)

einsetzen von $-S_i$
für S_i' ergibt dann

$$\begin{aligned} H' - H &= 2S_i \sum_{j \neq i} w_{ij} S_j \\ &= 2S_i \sum_{j} w_{ij} S_j - 2w_{ii} \end{aligned}$$

da $sgn(\sum_j w_{ij} S_j)$ das entgegengesetzte Vorzeichen von S_i hat, ist der erste Teilterm negativ und, da die w_{ii} positiv gewählt sind, ergibt sich eine negative Gesamtdifferenz.

Beide Fälle zeigen also, das die Energie keinesfalls zunimmt. Die w_{ii} sollten außer acht gelassen werden, da sie für Netze mit vielen Neuronen die Stabilität eines gespeicherten Musters nicht beeinflussen. Dies ändert sich jedoch, wenn in spärlich verbundenen oder kleinen Netzen der Beitrag der w_{ii} größer ist als der Beitrag der anderen Neuronen zur Gesamteingabe. Dann wird die Dynamik des Systems ganz erheblich betroffen und es treten mehr unerwünschte Zwischenzustände auf.

3.2.5 Herleitung der Theorie von einer Energiefunktion

Geht man von der Enegiefunktion aus, als etwas, das minimiert werden soll, um in einen stabilen Zustand zu gelangen, so hat man einen alternativen Weg die Hebb'sche Lernregel herzuleiten. Das Ziel ist es, die Energie des Systems zu minimieren wenn die Übereinstimmung mit einem gespeicherten Muster am größten ist. Setzt man $H = -\frac{1}{2N}(\sum_i S_i a_i)^2$, wobei a das gespeicherte Muster sei. Der Faktor $\frac{1}{2N}$ wird heuristisch gewählt. Auf diese Weise erhält man die Energiefunktion für ein Muster. Für alle gespeicherten Muster, summiert man einfach auf

$$\mathbf{H} = -\frac{1}{2N} \sum_{u=1}^{p} \left(\sum_i S_i a_i^u \right)^2$$

$$= -\frac{1}{2N}\sum_{u=1}^{p}\left(\sum_j S_j a_j^u\right)\left(\sum_i S_i a_i^u)\right)$$

$$= -\frac{1}{2}\sum_{ij}\left(\frac{1}{N}\sum_{u=1}^{p} a_i^u a_j^u\right) S_i S_j$$

wählt man die nun die Gewichte $w_{ij} = \frac{1}{N}\sum_{u=1}^{p} a_i^u a_j^u$, so entspricht die obige Gleichung exakt der Gleichung, die vorhin als generallisierte Hebb'sche Lernregel bezeichnet wurde.

3.2.6 Arbeitsweise des Hopfield Netzes

Ein Algorithmus für das Hopfield Modell läßt sich mit der Energiefunktion folgendermaßen definieren. Nach der Wahl der Gewichte, die für das vorliegende Problem zu berechnen sind, geht das Netz wie folgt vor.

Zu einer gegebenen Eingabe wird zuerst die gesamte Systemenergie gemäß den obigen Gleichungen berechnet. Dann wird eine sequentiell eine Zufallsauswahl eines Neurons durchgeführt bis die Energie unter eine vorgegebene minimale Schwelle sinkt. Ist diese minimale Energie nicht bekannt, kann solange iteriert werden bist die Enegiedifferenz unter eine gewisse Schranke ϵ fällt. Der Algorithmus läßt sich also etwa folgendermaßen notieren:

```
wähle w_ij gemäß dem Problem
Eingabe an das Netz anlegen
E = Σ_ij w_ij S_i S_j            /* Gesamtenergie */
repeat
  i := Random(1,n)               /* Wahl eines Neurons i */
  ΔE := E_i(S_i) − E_i(−S_i)     /* Energiedifferenz beim Zustandwechsel des Neurons i */
  if ΔE < 0 then                 /* Energie wird kleiner ? */
    E := E + ΔE                  /* Neue Systemenergie berechnen */
    S_i := −S_i                  /* Neuron i wechselt Zustand */
until E = minimal
```

3.2.7 Unerwünschte Muster

Wir haben bisher gesehen das wir in einem Hopfieldnetz p Muster speichern können und das diese auch wiedererkannt werden, dh die Energiefunktion hat für diese Muster lokale Minima, in die sich für die jeweilige Eingabe hineinläuft. Es stellt sich nun die Frage, ob es noch andere lokale Minima für die Energiefunktion gibt.

Es gibt noch andere, zum Einen sind natürlich alle Inversen Muster auch als lokale Minima vertreten, dies ist aber nicht so schlimm, da die Muster ja dennoch korrekt erkannt werden, nur mit einem anderen Vorzeichen. Desweiteren treten leider auch Linearkombination der zu speichernden Muster als lokale Energieminima auf, und das System kann sich bei diesen stabilen Zuständen fälschlicherweise einpendeln. Dies tritt vor allem bei sich überlappenden Mustern auf. Je stärker sich die Zu speichernden Muster ähneln, umso schlechter ist die Wiedererkennung bei gestörten Mustern und umso mehr Zwischenmuster werden gespeichert.

3.2.8 Erweiterungen des Hopfield Modells

Das bisher vorgestellte Modell war das ursprüngliche sequentielle von John Hopfield. Man kann sich aber auch fragen, was passiert, wenn statt sequentiell die Neuronenzustände zu ändern dies synchron parallel macht. Es zeigt sich dabei, daß keine Änderungen im Verhalten des Netzes eintreten. Es verändert sich weder die Speicherkapazität noch das Verhalten der Mustererkennung.

Eines der Hauptprobleme des Hopfield Modells sind die unerwünschten Muster, die immer dann auftreten, wenn sich die zu speichernden Muster überlappen. Dies führte ja zu den der begrenzten Speicherkapazität der Netze. Man kann die Kapazität aber erhöhen, wenn man spärlich kodierte Muster verwendtet, oder die zu speichernden Muster spärlich kodiert. Spärlich kodiert bedeutet, in diesem Zusammenhang, daß es nur wenige Komponenten im Mustervektor x gibt, die den Wert 1 haben. Man kann zeigen, daß die Speicherkapazität α für spärlich kodierte Muster ungefähr gleich $\frac{1}{2a|log(a)|}$ ist, wobei a die Wahrscheinlichkeit ist, das eine Musterkomponente gleich 1 ist. Bei spärlicher Kodierung der Muster ist es sinnvoll, für nicht zu lange Muster von der symmetrischen Kodierung der Zustände von -1 und 1 zu der unsymmetrischen von 0 und 1 zu wechseln. Es zeigt sich, daß dann nochmals eine geringe Kapazitätsverbesserung erreicht werden kann.

Eine weitere Variante, die auch von Hopfield untersucht wurde, ist ein Netz mit Neuronen, die Werte aus einem Intervall annehmen können, anstatt nur die binären Werte. Daß heißt also die Ausgabe eines Neurons läßt sich schreiben als $S_j = g\left(\sum_j w_{ij} S_j\right)$, wobei g(x) eine signoide Funktion sein muß wie beispielsweis tanh(x). Ein solches Netz arbeitet natürlich wie bisher auch asynchron oder synchron. Ein weiterer Arbeitsmodus bei den kontinuierlichen Neuronen ist allerdings, daß sich die Ausgaben permanet den Eingaben der Neuronen anpassen. Man kann in diesen Netzen zeigen, daß die gespeicherten Muster eindeutig gespeichert werden (man spricht dann von endgültigen Attraktoren), daß heißt es treten keine unbeabsichten Zwischenmuster auf.

3.3 Boltzmann Maschine

Man kann die Boltzmann Maschine als Erweiterung des Hopfield Modells durch wahrscheinlichkeitsgesteuerte Zustandsübergänge, ansehen. Ein Nachteil des Hopfield Modells ist, daß die Gewichte die später die stabilen Zustände garantieren, zuerst berechnet werden müssen. Dieser Nachteil wird bei der Boltzmann Maschine umgangen. Hier werden die Muster durch verändern der Gewichte gelernt.

3.3.1 Struktureller Aufbau

Die Neuronen der Boltzmann Maschine werden im Gegensatz zum Hopfield Netz in verschiedene Schichten eingeteilt. Man unterscheidet zwischen Eingabe-, Hidden- und Ausgabeneuronen (siehe Abbildung 3.5). Die Anzahl der Hidden-Neuronen hängt von dem Problem ab, das die Boltzmann Maschine lernen soll, sie können im Einzelfall auch entfallen. Was noch auffällt ist die nicht Verbindungstruktur. Hier sind die Neuronen nicht mehr alle untereinander verbunden, sondern nur noch innerhalb einer Schicht und zwischen den Schichten. Es gibt also keine vollständige Vernetzung der Neuronen mehr.

Was erhalten bleibt, sind die symmetrischen Verbindungen (dh $w_{ij} = w_{ji}$), da sonst keine Energiefunktion vorhanden wäre.

3.3.2 Arbeitsweise der Boltzmann Maschine

Die Einstellung der Zustände der Neuronen der Boltzmann Maschine kann man auf zwei Arten realisieren.

1. **Sequentiell**
   Die erste, und einfachste Variante, stellt die sequentielle Boltzmann Maschine dar. Hierbei werden die Zustandsübergänge nach der Verfahren ein Übergang zu einer Zeit vorgenommen. Dies läßt sich natürlich bei der Simulation auf einem konventionellen Rechen am einfachsten realisieren.

2. **Parallel**
   Bei der parallel Arbeitenden Boltzmann Maschine kann man wiederum zwei Arten unterscheiden, nämlich den synchronen und den asynchronen Fall.

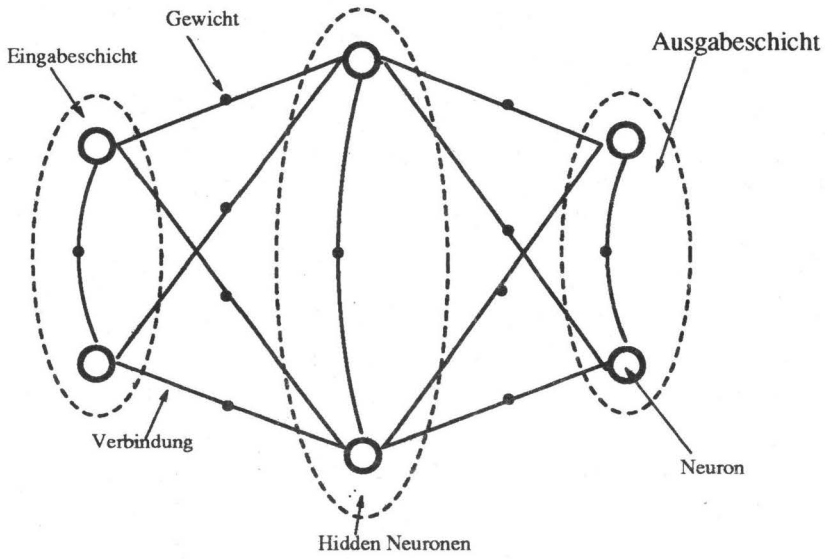


Abbildung 3.5: Struktur einer Boltzmann Maschine

(a) Synchrone Arbeitsweise
Man unterscheidet hiebei die begrenzte und die unbegrenzte Arbeitsweise. Im ersten Fall unterteilt man die Neuronen in Mengen von nicht benachbarten Neuronen. Diese Mengen können nun für sich parallel zur gleichen Zeit ihre Zustandswechsel berechnen. Dadurch kann es also nicht passieren, daß ein Neuron gerade seinen Zustand wechselt, während das benachbarte ebenfalls wechselt. Im zweiten Fall können alle Neuronen synchron ihren Zustand wechseln.

(b) Asynchrone Arbeitsweise
Hierbei arbeiten alle Neuronen unabhängig voneinander. Zustandswechsel werden beliebig berechnet und asgeführt. Diese arbeitsweise ist die problematischste, da hier Zustandwechsel geschehen können während ein Neuron noch bei der Auswertung seiner eigenen Eingaben ist. Durch den Wechsel des anderen Neurons kann für dieses Neuron eine völlig ander Situation entstehen, und es ist nicht mehr garantiert, daß die Systemenergie ständig abnimmt.

Im folgenden wird die Arbeitsweise einer sequentiellen Boltzmann Maschine beschrieben. Eine parallele Variante arbeitet natürlich vom Prinzip her genauso, doch müssen insbesondere bei der mathematischen Betrachtung Schwierigkeiten überwunden werden, die zum Teil noch nicht gelößt worden sind. Die grundlegende Aufgabe einer Boltzmann Maschine besteht im Einstellen eines Gleichgewichts bezüglich bestimmter Nebenbedingungen. Diese Nebenbedingungen können zum Beispiel Vorgabe eines Eingabemusters sein, für die eine Ausgabe gesucht wird, Vorgabe eines Ausgabemusters, um ein passendes Eingabemuster zu finden oder die Vorgabe von Ein- und Ausgaben , die, wenn sie gestört sind, ergänzt werden sollen bzw gelernt werden sollen. Hierbei wird das Vorgeben von Ein- und Ausgaben als gebundener Lauf bezeichnet. Die Suche nach passenden Ein- oder Ausgaben als freier Lauf.

Ein wesentlicher Punkt beim wechseln in einen Systemzustand ist, im Gegensatz zum Hopfield Modell, daß die Wechsel nur mit einer bestimmten Wahrscheinlichkeit (siehe Abbildung 3.6) durchgeführt werden. Diese Wahrscheinlichkeit hängt von der Energiedifferenz $\triangle E$ ab, die sich beim Wechsel einstellt. Der Übergang findet in jedem Fall statt, falls der neue Systemzustand eine geringere Energie aufweist als der Alte. Wird die Systemenegie aber erhöht, so wird der neue

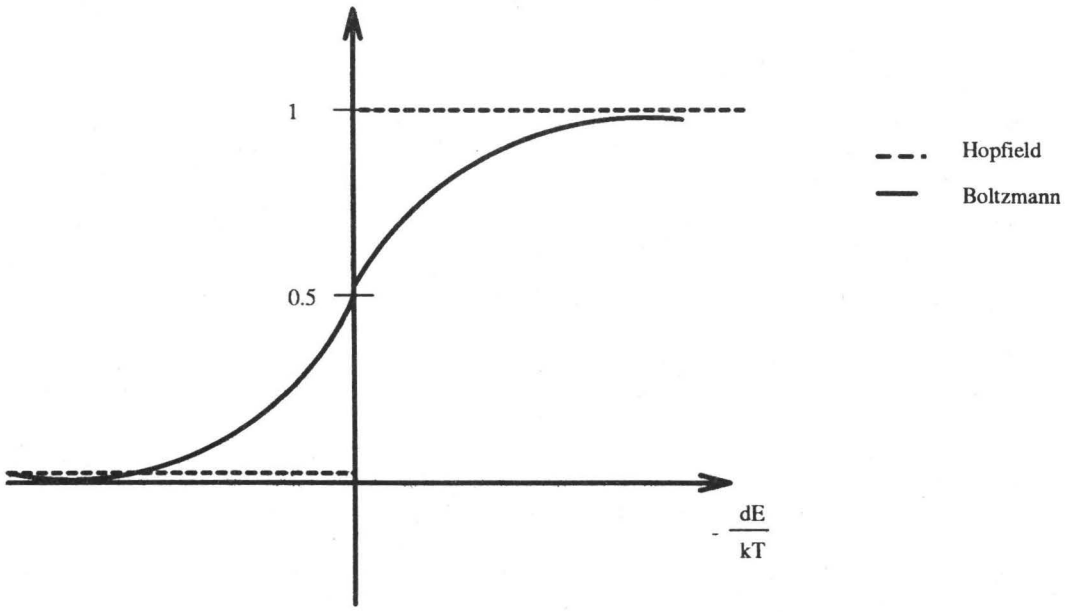


Abbildung 3.6: Annahmewahrscheinlichkeit eines Zustandes

Zustand mit einer exponentiell abnehmenden Wahrscheinlichkeit, nämlich

$$P(S) = \frac{1}{1 + exp(-\frac{\triangle E}{kT})}$$

angenommen. Hierbei bezeichnet k die Boltzmann Konstante und T kann man als Temperatur des Systems ansehen. Die zwei Konstanten kommen aus der Thermodynamik, da die Neuronen sich so Verhalten wie Magnetpartikel in einem Stoff bei gewissen Temperaturen. Dieses Verhalten der Neuronen hat den Vorteil, daß die Maschine aus lokalen Energieminima wieder herauskommen kann, um sich so dem gewünschten Zielzustand zu nähern. Zu bemerken bleibt noch das die Zustandsübergänge für $T \rightarrow 0$ genauso vorgenommen werden wie bei den Hopfield Netzen. Man kann drei verschiedene Funktionsweisen der Boltzmann Maschine unterscheiden:

1. Der freie Lauf :

   Von einem Ausgangszustand S gehen die Neuronen in einen anderen über, wobei die Übergangswahrscheinlichkeiten gelten.

2. Gebundener Lauf : Auslesen

   Der Zustand S einiger Ein- bzw Ausgabeneuronen wird vorgegeben und es stellt sich nach einiger Zeit ein Gleichgewicht im Gesamtsystem ein.

3. Gebundener Lauf : Lernen

   Der Zustände S der Ein- und Ausgabeneuronen wird vorgegeben. Die Gewichte werden nun derart verändert, daß die Zustände der Hidden Neuronen derart einstellen, daß ein stabiler Gesamt- zustand erreicht wird.

3.3.3 Das Lernen der Gewichte

Das Ziel des Trainings einer Boltzmann Maschine ist es, zu gegebenen Ein- und Ausgabemustern passende Zustände der Hiddenneuronen und Gewichte zu finden, so daß die Maschine sich im Gleichgewicht befindet. Um den Lernalgorithmus besser zu verstehen, soll im folgenden kurz der mathematische Hintergrund erläutert werden.

Mathematischer Hintergrund

Eine Konfiguration sei die Menge der Zustände der einzelnen Neuronen in dem System. Die Menge aller Konfigurationen K repräsentiert also alle Möglichen Belegungen der Neuronen im Netz. Sei $k \in K$ der Menge aller möglichen Konfiguration eine Konfiguration des Netzes.

$$E(k) = \sum_{ij \in k} w_{ij} S_i S_j$$

die Energie der Maschine für einen Zustand k. Die Wahrscheinlichkeit einer Konfiguration k für gegebene Gewichte w_{ij} im Gleichgewicht zu sein ist gegeben durch

$$q_k(c) = \frac{1}{N_o(c)} exp\left(\frac{\sum_{ij} w_{ij} S_i S_j}{c}\right) = \frac{exp\left(\frac{E(k)}{c}\right)}{N_o(c)}$$

wobei

$$N_o(c) = \sum_{l \in K} exp\left(\frac{E(l)}{c}\right)$$

$q_k(c)$ stellt die globale Sicht auf das gesamte System dar, wobei $q_k(c)$ beim trainieren des Netzes einer Bolzmann Verteilung folgt. Man unterscheidet zwischen möglichen und unmöglichen Zuständen. Mögliche Zustände sind solche Zustände, die aufgrund der Gewichtsverteilung und der Funktion $q_k(c)$ eine maximale Wahrscheinlichkkeit besitzen. Die Wahrscheinlichkeiten $q_k(c)$ sind so gewählt, daß die Konfigurationen, die die zu lernenden Muster beinhalten, natürlich eine höhere Wahrscheinlichkeit besitzen, als solche die keine oder nicht gewünschte beinhalten. Eine lokale Sicht auf das System geben die Verbindungen, bzw die damit verknüpften Gewichte. Eine Menge von Gewichten wird als möglich bezeichent, wenn sie bei einem Zustand k die Maschine im Gleichgewicht hält. Um ein qualitatives Maß für die Wahrscheinlichkeit einer solchen Gewichtsmenge zu erhalten definiert man die Aktivierungswahrscheinlichkeit einer Verbindung folgendermaßen:

$$p_{ij}(c) = \sum_k q_k(c) S_i S_j$$

Diese Gleichung gibt die Wahrscheinlichkeit an, wie oft zwei benachbarte Neuronen i und j bei einer gegebenen Wahrscheinlichkeit einer Konfiguration k gleichzeitig den Zustand 1 annehmen. Damit hat man ein Mittel an der Hand, daß es ermöglicht, aus lokalen Informationen Rückschlüsse auf die Wahl der Gewichte zu nehmen. Die mittlere Information eines Zustandes k ergibt sich als

$$H = -\sum_k q_k(c) ln\,(q_k(c))$$

diese Gleichung kommt aus der Thermodynamik und bezeichnet die mittle Entropie eines Zustands. Beim Training stellt sich nun ein Zustand k mit einer gewisssen Wahrscheinlichkeit $q'_k(c)$ ein. Die mittlere Information eines tatsächlichen Zustandes bei vorgegebener Wahrscheinlichkeit $q_k(c)$ beim Training läßt sich damit beschreiben als

$$H' = \sum_k (q_k(c) ln\,(q'_k(c)))$$

H' kann man als subjektive Information des Netzes deuten und die Differenz $H' - H = H_m$ als die fehlende Information. Diese fehlende Information sollte null werden, wenn das Netz sich an die vorgaben von außen angepaßt hat. Um nun diese Informationsdifferenz zu minimieren verwendet man die sogenannte Gradientensuche. Etwas mathematischer formuliert, kann man die Divergenz zweier Aktivierungswahrscheinlichkeiten q_k und q'_k folgendermaßen beschreiben.

$$D(q|q') = \sum_k q'_k(c) ln\left(\frac{q'_k(c)}{q_k(c)}\right)$$

Es läßt sich weiterhin zeigen, daß der Gradientenabsbstieg von $D(q|q') == 0$ gdw $p == p'$, daß heißt wenn die tatsächlichen Aktivierungswahrscheinlichkeiten der Verbindungen sich den erwarteten annähern. Dies ist ein wichtiger Punkt der später beim Lernalgorithmus ausgenutzt wird. Die Gradiendensuche läßt sich nun formulieren als die partielle Ableitung von $D(q|q)$ nach w_{ij}, und nach einigen Umformungen gelangt man zu der Form:

$$\frac{D(q|q')}{\partial w_{ij}} = -\frac{p'_{ij}(c) - p_{ij}(c)}{c}$$

p_{ij} und p'_{ij} können aus lokalen Informationen hergeleitet werden. Die Konstante c dient zur verbesserung des Konvergenzverhaltens beim der Arbeit der Maschine. Man kann durch geschickte Wahl des Paramters c eine Verbesserung erhalten. Dies muß jedoch ausprobiert werden.

Abschätzen der Aktivierungswahrscheinlichkeiten p, p'

Während des Trainings der Boltzmann Maschine können die Aktivierungswahrscheinlichkeiten $p_{ij}(c)$ und $p'_{ij}(c)$ für alle Gewichte gleichzeitig folgendermaßen bestimmt werden: Während der Präsentation der Beispiele, , zählt jede Verbindung wie oft sie aktiviert war, daß heißt wie oft die beiden Neuronen i und j gleichzeitig aktiviert waren. Daraus ergibt sich die relative Aktivierungsfrequenz z' welche eine Approximierung von p' darstellt.

Dann bringt sich die Maschine ins Gleichgewicht, daß heißt es finden keine Zustandwechsel der Hidden Neuronen mehr statt. Dabei werden wieder die Aktivierungen der einzelnen Verbindungen mitgezählt und daraus ergibt sich dann die relative Aktivierungshäufigkeit z_{ij}.

Der Lernalgorithmus

Es folgt ein kurzer Algorithmus der in groben Zügen das Trainig einer Boltzmann Maschine beschreibt. Der Algorthmus arbeitet grob etwa so

1. `Initialisierung der Gewichte`
   Zunächst werden allen Gewichten willkürliche Werte zugewiesen. Vorzugweise wird aber den Gewichten am Anfang der Wert 0 zugewiesen.

2. `Gewichtssequenz bilden`
   Dies läuft folgendermaßen ab : Man bildet eine Folge von Gewichten derart, daß

   $$w_{ij}(t+1) = w_{ij}(t) - \beta(z_{ij} - z'_{ij})$$

   gilt, wobei β ein Stabilitätsfaktor, der garantiert, das das System konvergiert. Man kann zeigen, daß das System sicher stabil läuft wenn

   $$\beta \leq \frac{c^2}{\text{Anzahl der Gewichte}}$$

   ist.

Der Lernalgorithmus läßt sich nun folgendermaßen notieren :

```
for alle Verbindungen w_ij do
  w_ij := 0                                   /* Gewichtsinitialisierung */
repeat
  repeat
    Freier Lauf
    berechene die z'_ij
    Gebundener Lauf
    berechne die z_ij
    w_ij := w_ij − β(z_ij − z'_ij)            /* Gewichtsanpassung */
  until Abbruchkriterium
until alle Muster trainiert
```

Beim Lernen passiert nun folgendes: Nachdem die Gewichte initialisiert worden sind, werden nacheinander die zu lernenden Muster angelegt. Dabei wird wie folgt vorgegangen. Zuerst wird ein Eingabemuster angelegt (Freier Lauf) und dann werden die Gewichte K - mal neu berechnet bis sich ein stabiler Gesamtzustand eingestellt hat. Daraus ergeben sich dann die z'_{ij} als

$$z_{ij} = \frac{\text{Anzahl der Aktivierungen}}{K}$$

. Dann werden zusätzlich die Ausgaben vorgegeben (Gebundener Lauf) es wird wiederum eine gewisse Weile iteriert bis sich ein stabiler Zustand eingestellt hat. Nun werden die z_{ij} auf die gleiche weise wie die z'_{ij} berechnet. Dann werden die Gewicht angepaßt. Da die Gewichte ein Maß für die Wahrscheinlichkeit der Aktivierung der Verbindung sind, werden Gewichte von Verbindungen die weniger oft aktiviert wurden erhöht, Gewichte von Verbindungen die öfter aktiviert werden, hingegen vermindert. Führt man diesen Prozeß entsprechend oft für jedes Ein- / Ausgabemuster durch so speichert das Netz diese ab.

Bemerkungen

Der Lernalgorithmus hier konvergiert zwar immer, es dauert aber unter umständen sehr lange bis die Muster gelernt werden. Dies liegt vor allem daran, das es in dem freien Lauf sehr lange dauert bis sich die Zustände stabilisiert haben. Man kann dies Umgehen, indem man mehr Neuronen festhält und nur ein Teil sich in Gleichgewicht bringt. Diese Technik wird vor allem bei Klassifikationsproblemen benutzt, da hier meist viele Eingabeneuronen verwendet werden, die dann entsprechend nicht mehr während des freien Laufs ins Gleichgewicht gebracht werden. Um die Konvergenzeigenschaften des Lernalgorithmus zu verbessern, erhöht bzw vermindert man die Gewichte der Verbindungen meist um eine Konstante γ, daß heißt

$$w_{ij}(t+1) = w_{ij}(t) - f(z_{ij} - z'_{ij})$$

wobei

$$f(x) = \begin{cases} \gamma & x > 0 \\ 0 & x = 0 \\ -\gamma & x < 0 \end{cases}$$

Mit dieser Methode erreicht man aber nicht den besten Fall, daß heißt den tiefsten Punkt eines lokalen Minimes, man erreicht aber meißt eine schneller Konvergenz des Lernalgorithmus. Die Termination des Lernalgorithmus kann nur durch die lokale Information bei den einzelnen Verbindungen sichergestellt werden. Man verwendet meist das Kriterium, das $|z_{ij} - z'_{ij}| < \epsilon$ nach K Iterationen des Lernprozeß.

Bei praktischen Implementation der Boltzman Maschine fällt auf, daß, wenn sie einmal trainiert ist, sie relative unempfindlich ist gegen den ausfall einzelnen Neuronen. Es wird trotzdem noch das erwartete Verhalten gezeigt. Man kann sogar die Gewichte in gewissen Grenzen stören, dann wird jedoch eine, wenn auch sehr kurze, Wiederlernphase nötig, und das Verhalten bleibt relativ stabil.

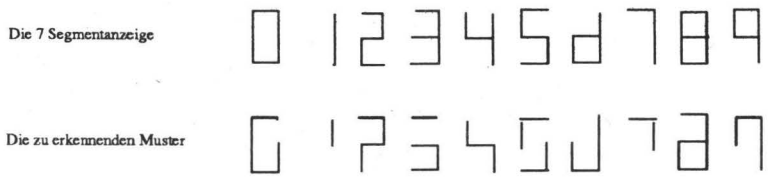


Abbildung 3.7: Die Eingabemuster

Weiterhin ist noch interessant, das bei zu lernende Muster derart in internen (Hidden-Neuronen) Konfigurationen abgelegt werden, daß die einzelnen Muster den größtmöglichen Hamming Abstand voneinander haben.

3.3.4 Beispiel einer gelernten 7 Segment Anzeige:

Im folgenden soll am Beispiel einer zu lernenden 7 Segementanzeige die Mustererkennung einer Boltzmann Maschine gezeigt werden. Zu diesem Zweck wurde eine Boltzman Maschine mit 7 Eingabeneuronen für die 7 Segmente , 10 Hiddenneuronen und 10 Ausgabeneuronen ver wendet. Hierbei sollte jeweils ein Ausgabeneuron für die entsprechende Ziffer aktiviert werden. In Abbildung 3.7 sind sie zu lernenden Ziffern, sowie die späteren zu erkennenden Muster dargestellt. In den Tabellen steht in der horizontalen Spalte immer die erkannte Ziffer und in der vertikalen die Netzeingabe.

Das Abbruchkriterium für den Lernalgorithmus war $|z_{ij} - z'_{ij}| < 0.05$, β wurde auf 1 gesetzt. Nach 870 Iterationen stoppte der Lernalgorithmus mit folgenden Ergebnissen:

Zuerst wurden von dem Netz, von jeder Zahl je 100, zufällig ungestörte Ziffern eingegeben. Die nachfolgende Tabellen zeigen, daß in 972 Fällen die richtige Zahl erkannt wurde.

	0	1	2	3	4	5	6	7	8	9
0	100	-	-	-	-	-	-	-	-	-
1	-	98	-	-	-	-	-	2	-	-
2	-	-	99	1	-	-	-	-	-	-
3	-	-	-	100	-	-	-	-	-	-
4	-	-	-	-	100	-	-	-	-	-
5	-	-	-	-	-	100	-	-	-	-
6	-	-	-	-	-	-	100	-	-	-
7	-	13	-	-	-	-	-	87	-	-
8	7	-	2	-	-	-	-	-	91	-
9	-	-	-	-	3	-	-	-	-	97

Erkennungshäufigkeit der Eingaben bei jeweils 100 Mustern.

	0	1	2	3	4	5	6	7	8	9
0	0	4	3	3	4	3	3	3	1	3
1	4	0	5	3	2	5	5	1	5	3
2	3	5	0	2	5	4	4	4	2	4
3	3	3	2	0	3	2	4	2	2	2
4	4	2	5	3	0	3	3	3	3	1
5	3	5	4	2	3	0	2	4	2	2
6	3	5	4	4	3	2	0	6	2	4
7	3	1	4	2	3	4	6	0	4	2
8	1	5	2	2	3	2	2	4	0	2
9	3	3	4	2	1	2	4	2	2	0

Hamming Abstand der 7 Segment Eingaben untereinander.

Um die asssoziativen Fähigkeiten der Maschine zu testen wurden je 100 Mal gestörte Eingaben vorgegeben. Die folgende Tabelle zeigt die Häufigkeit der erkannten Muster. Betrachtet man die Hamming Abstände der Muster zu den erkannten Ziffern, so sieht man daß die Maschine immer zu den Mustern mit den kleinesten Hamming Abständen, daß heißt zu den nächstliegenden, konvergiert. In 900 von den 1000 Fällen wurde zu einem Muster mit dem Hamming Abstand 1 zur Eingabe konvergiert (vergleiche Abbildung 3.7) .

	0	1	2	3	4	5	6	7	8	9
0	82	-	-	-	-	-	18	-	-	-
1	-	100	-	-	-	-	-	-	-	-
2	-	-	100	-	-	-	-	-	-	-
3	-	-	2	56	-	42	-	-	-	-
4	-	-	-	-	97	-	-	-	-	3
5	30	-	-	7	-	61	2	-	-	3
6	7	-	-	-	-	-	93	-	-	-
7	-	1	-	-	-	-	-	99	-	-
8	4	-	49	35	-	-	-	-	12	-
9	-	26	-	-	3	-	-	64	-	10

Erkennungshäufigkeit der Eingaben bei jeweils 100 Mustern.

	0	1	2	3	4	5	6	7	8	9
0	1	5	4	4	5	2	2	4	2	4
1	5	1	4	4	3	6	6	2	6	4
2	4	4	1	3	4	5	5	3	3	3
3	4	4	3	1	4	1	3	3	3	3
4	5	3	6	4	1	2	2	4	4	2
5	2	4	5	3	4	1	3	3	3	3
6	2	4	5	5	4	3	1	5	3	5
7	4	2	5	3	4	3	5	1	5	3
8	2	4	1	1	4	3	3	3	1	3
9	2	2	5	3	2	3	5	1	3	1

Hamming Abstände der gestörten zur ungestörten Eingabe

Es sei noch angemerkt, daß man dieses Problem auch ohne Hidden Neuronen lösen kann, und zwar lediglich mit 7 Eingabeneuronen und 10 Ausgabeneuronen.

3.4 Zusammenfassung

Es wurden zwei neuronale Netzwerke vorgestellt, die sich sehr ähnlich sind. Beim Hopfield Modell fällt die einfache Struktur und die einfache Implementierung auf herkömmlichen Rechnern auf. Wohingegen die Boltzmann Maschine durch ihre komplexere Struktur bessere Speicherkapazitäten aufweißt. Bei Konzepte eignen sich gut als Assoziativspeicher, wobei das Hopfield Modell aufgrund seiner schlechten Kapazitätsverhältnisse natürlich hinter der Boltzmann Maschine zurückbleibt. Die Boltzmann Maschine hat aber noch einen anderen Vorteil. Man kann sie nämlich als bidirektionalen Assoziativspeicher verwenden, daß heißt man kann zu einer gegebenen Ausgabe ein passendes Eingabemuster suchen. Desweiteren ist die Boltzmann Maschine unempfindlicher gegen Störungen innerhalb ihrer Struktur, daß heißt gegen den Ausfall einzelner Neuronen oder Verbindungen innerhalb der Maschine. Durch Ihren stochastischen Ansatz wird auch die Speicherung von Mustern wesendlich stabiler und damit die Mustererkennung verbessert. Desweitern erlaubt die geschichtete Struktur eine wesendlich einfachere Problemmodellierung als beim Hopfield Netz, wo zu einer gegebenen Eingabe immer ebensoviele Ausgabeneuronen vorhanden sein müssen. Man kann aber beim Hopfield Modell auf effiziente Weise die Gewichte berechnen und muß nicht wie

bei der Boltzmann Maschine ein unter umständen langwieriges Training der Neuronen vornehmen. Bei der Boltzmann Maschine hingegen kann durch geschicktes anpassen der Gewichte auch ein vergessen von Informationen realisiert werden. Eindeutige stärken der beiden Ansatze sind wie schon erwähnt Mustererkennung oder etwas allgemeiner Klassifikationsaufgaben. Hierbei ist noch zu bemerken, daß mit der Boltzmann Maschine auch solche Probleme, wie zum Beispiel eine XOR Funktion, realisiert werden können. Desweitern eignet sich die Boltzmann Maschine auch hervoragend zur Lösung von kombinatorischen Problemen, wie das Einfärben von Graphen oder das Problem eines Handlungsreisenden, was nicht zuletzt an der flexiblen Schichtenstruktur liegt. Bei einem Hopfield Netz muß man, um solche Probleme zu lösen, einen nicht unbeträchtlichen Aufwand in die Problemkodierung stecken. Beide Ansätze verfügen also über ihre Vor- und Nachteile, wobei es im gegebenen Problemfall abzuwägen gilt für welches Modell man sich entscheidet.

3.5 Literatur

[1] John Hertz. *Introduction to the theory of neural computation* Addison-Wesley Publishing Company, 1991

[2] Emilie Aarts. *Simulatet Anealing and Boltzmann machines* Wiley-Interscience Series, 1989

[3] Rüdiger Brause. *Neuronale Netze* Teubner Verlag, 1986

[4] V. Vemuri. *Artificial neural networks: Theoretical concepts* Computer society press, No 855

Kapitel 4

Feedforward–Perzeptronen und Backprop

Martin Buch

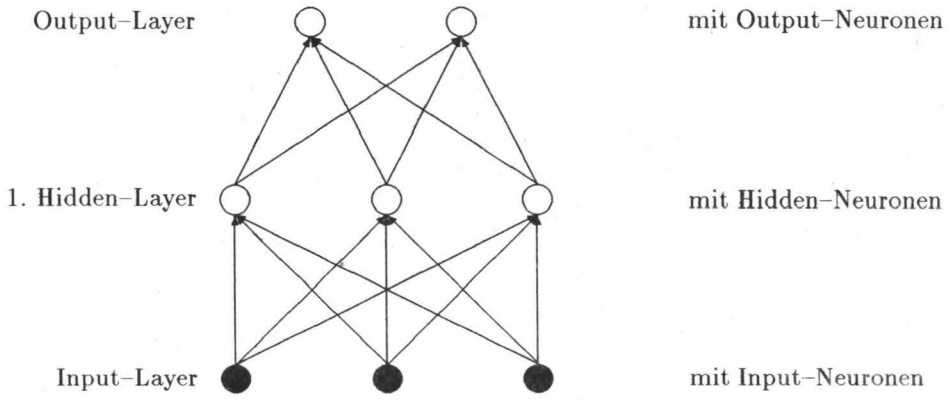
Übersicht

Feedforward–Perzeptronen sind Neuronale Netze aus einer oder mehreren Schichten gleichartiger Neuronen, die für praktische Anwendungen eine wichtige Rolle spielen. Im Rahmen dieses Kapitels sollen Perzeptronen daraufhin untersucht werden, ob und wie man sie darauf trainieren kann ein gegebenes Assoziationsproblem (Zuordnung von Ein– zu Ausgabewerten) zu lösen. Der Backpropagation-Algorithmus ist dabei von zentraler Bedeutung für die Beherrschung mehrschichtiger Feedforward-Netze.

4.1 Begriffe und Vorbemerkungen

4.1.1 Struktur des Feedforward-Netzes

In diesem Kapitel werden neuronale Netze behandelt, die eine "innere Struktur" haben. Daß heißt, die Neuronen sind schichtenweise hintereinandergeschaltet, beginnend mit einer Schicht aus Eingabeneuronen (Input-Layer, 0. Schicht), eventuell Schichten aus "verborgenen" Neuronen (Hidden-Layer) und einer Schicht aus Ausgabe-Neuronen (Output-Layer, n. Schicht). Jedes Neuron der i-ten Schicht ist mit allen Neuronen der i+1-ten Schicht verbunden. Über mehrere Schichten hinweggehende oder rückwärts gerichtete Verbindungen unter den Neuronen gibt es nicht. Aufgrund dieser Struktur spricht man auch von **Feed-Forward-Netzen** oder **Perzeptronen**.
Ein Netzwerk mit nur einer Ein– und einer Ausgabeschicht bezeichnet man als **Simple Perceptron**.



Zur Erinnerung: Prinzipielle Funktionsweise eine Neurons
Aus den Eingangsaktivitäten ζ_k wird eine gewichtete Summe gebildet:

$$h = \sum_k w_k \zeta_k$$

wobei w_k das Gewicht der k-ten Eingangsleitung darstellt.
Das Neuron erzeugt eine **Aktivität** O ("feuert"), wenn diese Summe h einen bestimmten Wert θ (**Schwellwert**, Erregungsschwelle) überschreitet:

$$O = g(h - \theta) \qquad \text{mit} \qquad g(x) = \begin{cases} 1 & : \quad x \geq 0 \\ 0 & : \quad x < 0 \end{cases}$$

g bezeichnet man auch als **Aktivierungsfunktion**.
Für die Anwendung ist man aber nicht an bestimmte Aktivierungsfunktionen gebunden. Man unterscheidet, je nach Art der Aktivierungsfunktion, verschiedene Klassen von Neuronen, die in den nächsten Abschnitten noch detailliert untersucht werden. Folglich kann für die Neuronenaktivitäten (Eingangs– wie Ausgangsaktivitäten) ein diskreter Wertebereich oder kontinuierlicher Wertebereich (z.B. $R, R^+,]-1;1[, \ldots$) angenommen werden.

4.1.2 Aufgaben eines Feedforward-Netzes

Die Neuronen des Input-Layers sind eigentlich keine Neuronen im Sinne der Defintion. Sie haben lediglich die Aufgabe, die Komponenten eines vorgegebenen Eingabevektors an alle Neuronen der darauffolgenden (ersten) Schicht weiterzuleiten. Erst ab hier beginnen die Neuronen, gemäß ihrer Aktivierungsfunktion, diesen "Eingabevektor" Schicht für Schicht durch das Netz zu propagieren bis schließlich am Output–Layer ein Ausgabevektor erscheint.
Hat man nun eine Menge von sogenannten **Musterpaaren** (Pattern) mit Eingabe– und erwünschten Ausgabe–Vektoren (= **Trainingsset**) vorgegeben, so ist es das Ziel die Gewichte der Neuronenverbindungen im Netz so zu wählen, daß propagierte Ausgabe und erwünschte Ausgabe für alle Paare übereinstimmen (bzw. möglichst kleine Differenzen aufweisen). Man spricht dabei auch vom Prinzip des **überwachten Lernens**.
Für die praktische Anwendung wird es wichtig sein, Bedingungen für die Lösbarkeit eines solchen Zuordnungsproblems und vor allem Lösungsmöglichkeiten zu ergründen. Bevor wir darauf eingehen, sollen die für ein Feedforward–Netz im folgenden verwendeten mathematischen Symbole erklärt und eine mathematische Formulierung der Aufgabe und Funktionsweise erreicht werden.

4.1.3 Mathematische Beschreibung der Aufgabe eines Perzeptrons

Die folgenden mathematischen Symbole werden uns im Laufe des Kapitels noch häufig begegnen. Deren Bedeutung soll deshalb vorab schon einmal angegeben werden:

Trainingsset: $\{(\xi^\mu, \zeta^\mu) | \mu = 1, \ldots, p\}$

ξ^μ	Eingabe-Muster für den Input–Layer
ξ_k^μ	Eingabewert für das Input–Layer–Neuron k oder allgemein: Eingangsaktivität eines Neurons
ζ^μ	Ziel–Muster (–Pattern) = gewünschte Ausgabe des Output–Layers
ζ_i^μ	gewünschter Ausgabewert des Output–Layer–Neuron i
i, j, k	Indizes für Neuronen
w_{ij}	Gewicht der gerichteten Verbindung von Neuron j zu Neuron i
h_j^μ	(Gesamt–) Eingabe des Neurons j
V_j^μ	Ausgangsaktivität des (Hidden–) Neurons j
O_i^μ	Ausgabe von Output–Neuron i
θ_i	"Erregungsschwelle" des Neurons i
g	Neuronen-Aktivierungsfunktion (gleich für alle Neuronen im Netz)

Gegebenenfalls wird die o. a. Symbolik an Ort und Stelle erweitert oder modifiziert.
Die Neuronen–Erregungsschwelle θ, die in dem eingangs vorgestellten Neuronenmodell (Schwellwertneuron) benutzt wurde, verliert ihre Bedeutung für kontinuierliche oder gar differenzierbare Aktivierungsfunktionen. Trotzdem ist es sinnvoll für jedes Neuron eine Konstante (in der Literatur häufig 'bias' genannt) zu definieren, die der gewichteten Summe über die Eingangsaktivitäten hinzuaddiert wird, ehe man die Aktivierungsfunktion darauf anwendet. Formelmäßig läßt sich das am besten realisieren, wenn man jedem Neuron eine weitere Eingangsverbindung mit konstantem Gewicht $w_{i0} = \theta_i$ und der konstanten Eingabe $\zeta_0 = -1$ hinzufügt:
Für die Ausgabe eines Output–Neurons gilt dann z. B. :

$$O_i = g(h_i) = g\left(\sum_{k=1}^{N} w_{ik}\xi_k - \theta_i\right) = g\left(\sum_{k=0}^{N} w_{ik}\xi_k\right) \tag{4.1}$$

mit $w_{i0} = \theta_i, \xi_0 = -1$
Wenn keine Unklarheiten bestehen, so verzichten wir in Zukunft auf die Summationsgrenzen.
Hat man nun ein Zuordnungsproblem in Form einer Menge von Ein–/Ausgabevektoren gegeben, so sollen die Gewichte im Feedforward-Netz so gewählt werden, daß für alle Output–Layer–Neuronen und alle Musterpaare μ gilt:

$$O_i^\mu = \zeta_i^\mu \qquad \text{bei Eingabe} \qquad \xi_\mu \tag{4.2}$$

Es ergeben sich nun z.B. folgende Fragen:

- gibt es überhaupt passende Gewichte w_{ik}, die ein gegebenes Assoziationsproblem lösen ?
- wenn ja, wie finde ich diese am schnellsten. Gibt es eine möglichst einfache **Lernregel** um, ausgehend von einer zufälligen Wahl, diese Gewichte zu ermitteln (in endlich vielen Schritten ?)

4.2 Simple Perceptrons

In diesem Abschnitt werden die einfachsten Feedforward–Netze behandelt, die mit einem Input– und einem Output–Layer. Wir untersuchen deren Verhalten für verschiedene Arten von Neuronen: Schwellwert–, lineare und nichtlineare Neuronen:
Diese unterscheiden sich im wesentlichen in der Aktivierungsfunktion.

4.2.1 Schwellwertneuronen

Einfache Beispiele, Lösbarkeitskriterien

Wir beginnen mit den einfachsten Neuronen, den Schwellwertneuronen. Ein Schwellwertneuron kennt für seine Ausgangsaktivität nur zwei Zustände, abhängig von einem Schwellwert (Erregungsschwelle).

Beispiele

$$g(h) = \mathrm{sgn}(h) = \begin{cases} +1 & : \quad h \geq 0 \\ -1 & : \quad h < 0 \end{cases} \tag{4.3}$$

oder

$$g(h) = \begin{cases} 1 & : \quad h \geq \theta \\ 0 & : \quad h < \theta \end{cases} \tag{4.4}$$

Die angegebenen Aktivierungsfunktionen sind durchaus typisch für Schwellwertneuronen. So können beide benutzt werden um z.B. Boolesche Funktionen zu realisieren — wir werden das noch sehen.

Wir müssen natürlich annehmen, daß die vorgegebenen Zielvektoren ebenfalls nur solche Werte haben wie die Aktivierungsfunktion sie annehmen kann, wenn das Perzeptron seine Aufgabe korrekt erfüllen soll.
Der Einfachheit halber beschränken wir unsere Untersuchungen auf das einzelne Output-Neuron (und nicht das Ausgabeverhalten des ganzen Layers). Dies ist möglich, da das Verhalten der Output-Neuronen voneinander unabhängig ist. Auf eine Indizierung der Output–Neuronen sei daher verzichtet.
Die Gewichte schreibt man zweckmäßigerweise in Vektorform (erscheinen fettgedruckt), ebenso die Werte der Eingabe–Muster:

$$\begin{aligned} \mathbf{w} &= (w_1, w_2, \ldots, w_N) \\ \xi^\mu &= (\xi_1^\mu, \ldots, \xi_N^\mu) \end{aligned}$$

dabei bezeichne N die Anzahl der Inputneuronen.
Als Aktivierungsfunktion $g(h)$ wollen wir die Signumfunktion nehmen (Beispiel, s. o.). Aus (1) ergibt sich dann:

$$O = \mathrm{sgn}\left(\sum_{k=1}^{N} w_k \xi_k^\mu - w_0\right) = \mathrm{sgn}(\mathbf{w} \cdot \xi^\mu - w_0) \tag{4.5}$$

und als Ziel (siehe (2))

$$O = \zeta^\mu \qquad \text{(gewünscht)}$$

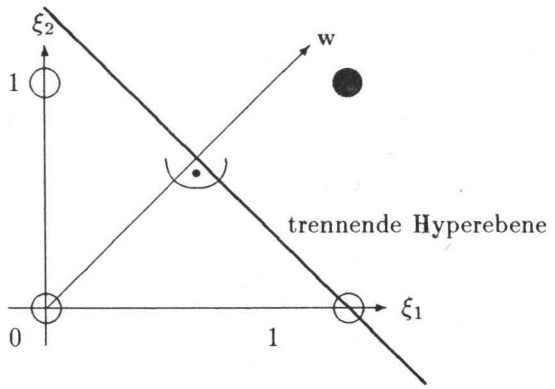
Anders ausgedrückt

$$\begin{aligned} \mathbf{w} \cdot \xi^\mu &\geq w_0 \qquad \text{falls } \zeta^\mu = +1 \\ \mathbf{w} \cdot \xi^\mu &< w_0 \qquad \text{falls } \zeta^\mu = -1 \end{aligned} \tag{4.6}$$

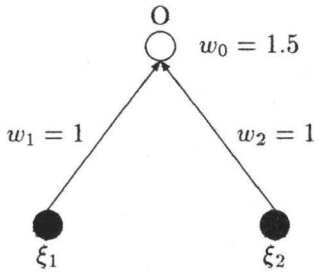
Geometrisch bedeutet dies, daß im N–dimensionalen Raum der Eingabevektoren eine N-1–dimensionale Hyperebene $\mathbf{w} \cdot \xi = w_0$ so gelegt werden kann, daß sie die Eingabe–Muster mit unterschiedlichen Zielzuständen $(+1/-1)$ voneinander trennt.

Beispiel: (N=2), Boolesche AND-Funktion (+1 = TRUE, -1 = FALSE)

μ	ξ_1	ξ_2	ζ
1	0	0	−1
2	0	1	−1
3	1	0	−1
4	1	1	+1



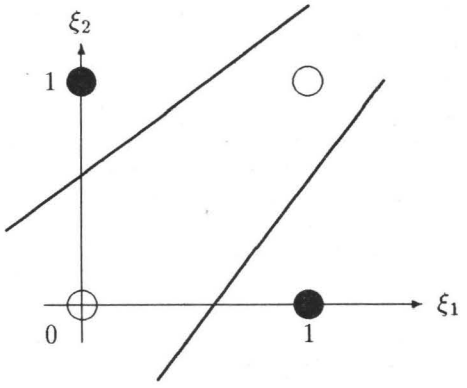
Ein Simple Perceptron mit Schwellwertneuronen, daß die AND–Funktion realisiert, wäre z. B.



(6) bedeutet also, daß die Existenz einer solchen trennenden Hyperebene notwendige Bedingung für die Lösbarkeit eines gegebenen Zurordnungsproblems ist. Man nennt diese Eigenschaft auch **lineare Trennbarkeit** .

Gegenbeispiel: (N=2), Boolesche XOR–Funktion

μ	ξ_1	ξ_2	ζ
1	0	0	−1
2	0	1	+1
3	1	0	+1
4	1	1	−1



hier kann es aufgrund der Verteilung der Eingabevektoren keine die TRUE– und FALSE–Punkte trennende Linie geben. Also läßt sich XOR nicht mit einem Simple Perceptron aus Schwellwertneuronen realisieren.
Dies läßt sich auch erkennen, wenn die Gleichungen (6) explizit notiert werden:

$$\begin{aligned} 0 \cdot w_1 + 0 \cdot w_2 &= 0 < w_0 \\ 0 \cdot w_1 + 1 \cdot w_2 &= w_2 \geq w_0 \\ 1 \cdot w_1 + 0 \cdot w_2 &= w_1 \geq w_0 \\ 1 \cdot w_1 + 1 \cdot w_2 &= w_1 + w_2 < w_0 \end{aligned}$$

Die Addition der 2. und 3. bzw. der 1. und 4. Gleichung ereben den Widerspruch:

$$2w_0 \leq w_1 + w_2 < 2w_0$$

Auch die allgemeine Paritätsfunktion (mit N Eingabewerten), als Verallgemeinerung des XOR, läßt sich aus Gründen der nicht–linearen Trennbarkeit mit Simple Perceptrons nicht verwirklichen.

Ein einfacher Lernalgorithmus

Hat man nun die lineare Trennbarkeit für sein Problem, so möchte man die Gewichte der Neuronenverbindungen so verändern, daß das Netzwerk seine Aufgabe erfüllt (d. h. $O_i^\mu = \zeta_i^\mu \quad \forall i$). Die Paare von Ein–/Ausgabemustern legen einen iterativen Ansatz nahe:

$$w_{ik}^{neu} = w_{ik}^{alt} + \Delta w_{ik} \qquad \text{mit} \tag{4.7}$$

$$\Delta w_{ik} = \begin{cases} 2\,\eta\,\zeta_i^\mu \xi_k^\mu & : \ \zeta_i^\mu \neq O_k^\mu \\ 0 & : \ \text{sonst} \end{cases} \tag{4.8}$$

Dies ist die sogenannte **Hebb'sche Lernregel**. Sie beruht auf den Erkenntnissen des Psychologen Hebb (1949), daß die durch eine Synapse bewirkte Verbindung zwischen zwei Neuronen sich proportional zur Aktivität vor und hinter der Synapse ändert. (8) spiegelt diesen Sachverhalt nicht ganz exakt wieder (ζ_i^μ statt O_i^μ !), hat sich jedoch als sinnvolle Lernregel für Simple Perceptrons herausgestellt und läßt sich durch einen anderen, mathematischen Ansatz bestätigen. Dazu mehr im nächsten Abschnitt.
Die Wahl des Proportionalitätsfaktors (2η) hat eine entscheidende Bedeutung für die Effektivität des Lernalgorithmus. η bezeichnet man daher auch als **Lernrate**.

Im Fall der Schwellwertneuronen (also z. B. $\zeta_i^\mu, O_i^\mu = \pm 1$) läßt sich (7) umschreiben zu

$$\Delta w_{ik} = \eta\,(1 - \zeta_i^\mu O_i^\mu)\,\zeta_i^\mu \xi_k^\mu \qquad \text{oder} \tag{4.9}$$

$$\Delta w_{ik} = \eta\,(\zeta_i^\mu - O_i^\mu)\,\xi_k^\mu \tag{4.10}$$

Gleichung (10) wird uns in ähnlicher Form auch noch bei anderen Lernregeln für Feedforwardnetze begegnen.
Die **Konvergenz** (d. h. das Finden einer Lösung in endlich vielen Schritten) der Perzeptronen-Lernregel (8) ist, unter gewissen Voraussetzungen (lineare Trennbarkeit, alle Musterpaare beim Trainieren berücksichtigen) mathematisch nachweisbar. Dabei ist es interessant, daß die Anzahl der tatsächlich durchgeführten Gewichtsanpassungen wohl von der Anzahl der Input-Neuronen (N), der Lernrate η, aber **nicht** von der Anzahl der zu lernenden Musterpaare abhängt. In der Praxis müssen aber alle Musterpaare daraufhin überprüft werden, ob und welche Gewichte zu verändern sind. Daher ist die Anzahl der benötigten Lernschritte natürlich auch von der Größe des Trainigssets abhängig.

4.2.2 Lineare Neuronen

Schwellwertneuronen haben den Nachteil, daß sie sich im Hinblick auf die Qualität eines bestimmten Lernalgorithmus einer genauen mathematischen Analyse entziehen. Grund dafür ist die

Nicht-Differenzierbarkeit der Aktivierungsfunktion.
Von nun an sei also die Aktivierungsfunktion **stetig und differenbzierbar**. Dann läßt sich eine sogenannte Kostenfunktion $E(\mathbf{w})$ konstruieren, mit der man die Performance eines Feedforward-Netzes "messen" kann. Diese Kostenfunktion wird die Fehler bei den Output-Neuronen ($|\zeta_i^\mu - O_i^\mu|$) quantitativ erfassen und eine differenzierbare, von den Gewichten der Neuronenverbindungen ($\mathbf{w} = \{w_{ij}\}$) abhängige Funktion sein.
Ziel ist es, mit Optimierungsstrategien die Gewichte so zu verändern, daß sie die Kostenfunktion minimieren, d. h. das Neuronale Netz seine Aufgabe fehlerminimal lösen kann.

In diesem Abschnitt beschäftigen wir uns mit dem einfachsten Fall differenzierbarer Aktivierungsfunktion, den linearen Funktionen. Daher bezeichnet man Neuronen, die eine lineare Beziehung zwischen ihrer Ein- und Ausgangsaktivität haben als **Lineare Neuronen**.
Im folgenden sei also

$$g(h) = h$$

Es ist vollkommen ausreichend sich auf die Identität zu beschränken, denn:
Ein Proportionalitätsfaktor ist überflüssig, ein entsprechender Effekt würde sich durch Veränderungen der Gewichte der Neuroneneingangsleitungen erzielen lassen. Eine additive Konstante wurde generell für jedes Neuron implizit vereinbart (siehe Vorbemerkungen).
Da wir uns immer noch mit Simple Perceptrons auseinandersetzen, ergibt sich:

$$O_i^\mu = \sum_k w_{ik}\xi_k^\mu$$

bei der Eingabe des Patterns ξ^μ.
Wir wollen erreichen, daß

$$\zeta_i^\mu = O_i^\mu \qquad \text{also} \qquad \zeta_i^\mu = \sum_k w_{ik}\xi_k^\mu \tag{4.11}$$

gilt. Hierzu gibt es verschiedene Lösungsansätze:

Explizite Lösung

(11) ist nichts anderes als ein lineares Gleichungssystem in den Variablen w_{ik}. Diese lassen sich, im Fall der Lösbarkeit, z. B. mit Hilfe der **Pseudoinversen** der folgenden Matrix Q bestimmen:

$$Q_{\mu\nu} = \frac{1}{N}\sum_k \xi_k^\mu \xi_k^\nu = \frac{1}{N}\xi^\mu \cdot \xi^\nu \qquad \mu,\nu = 1,\ldots,p \tag{4.12}$$

Q bezeichnet man in diesem Zusammenhang auch als Überlappungsmatrix der Eingabe–Pattern.
Man kann schnell nachprüfen, daß

$$w_{ik} = \frac{1}{N}\sum_{\mu\nu} \zeta_i^\mu (Q^{-1})_{\mu\nu}\xi_k^\nu \tag{4.13}$$

eine Lösung für (11) ist, falls Q^{-1} existiert. Und dies ist genau dann der Fall, wenn die Eingabemuster ξ^μ **linear unabhängig** sind, also

$$a_1\xi^1 + a_2\xi^2 + \ldots + a_p\xi^p = 0$$

nur die triviale Lösung $a_1 = a_2 = \ldots = a_p = 0$ besitzt.
Die lineare Unabhängigkeit ist hinreichend, aber nicht notwendig für die Lösbarkeit einer Netzwerk–Aufgabe mit linearen Neuronen. Selbst bei linear abhängigen Eingabe–Vektoren kann es Gewichte geben, so daß (11) erfüllt ist. Dies hängt zusätzlich von einer speziellen Wahl der vorgegebenen Ausgabe–Vektoren ab, welche in (11) die 'rechte Seite' des Gleichungssystems bilden.
Man beachte auch den Unterschied zwischen linearer Abhängigkeit und linearer Trennbarkeit. Lineare Unabhängigkeit impliziert lineare Trennbarkeit, aber nicht umgekehrt. So sind z. B. die Eingabemuster für die logischen Funktionen AND, OR linear trennbar aber wegen $p > N$ (d. h. die Anzahl der Pattern ist größer als die Anzahl der Input-Neuronen) linear abhängig.

Gradientenabstiegverfahren

Der direkte Weg über Gleichung (13) zur Bestimmung der Gewichte ist für praktische Zwecke ungeeignet. Zum einen ist das Berechnen der Inversen einer i. a. sehr großen Matrix zu aufwendig und algorithmisch oft zu ungenau, andererseits weiß man nicht von vornherein ob es überhaupt eine perfekte Lösung gibt, möchte aber doch die Gewichte so bestimmen, daß das Netz minimale Ausgabefehler produziert. Es ist besser eine Lernregel zu finden, die ausgehend von frei wählbaren Startwerten, die Gewichte iterativ bestimmt.
Hier kommt die eingangs erwähnte Kostenfunktion $E(\mathbf{w})$ zum Tragen. Wir definieren sie wieder als Summe der Fehlerquadrate $(\zeta_i^\mu - O_i^\mu)^2$ über alle Output–Neuronen und alle Musterpaare:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - O_i^\mu)^2 = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - \sum_k w_{ik}\xi_k^\mu)^2 \tag{4.14}$$

Es ist $E(\mathbf{w}) = 0$ genau dann, wenn unsere Forderung (11) erfüllt ist, also es eine exakte Lösung gibt.
Der naheliegende Weg zur Berechnung des Minimums von E ist, die partiellen Ableitungen von E zu ermitteln und deren Nullstellen zu berechnen. Dies würde wieder zu einem linearen Gleichungssystem führen und eine solche Lösungsmöglichkeit wurde bereits bei der expliziten Lösung verworfen.
Nun hat daber der **Gradient** $\frac{\partial E}{\partial w_{ik}}$, der Vektor der partiellen Ableitungen, die Eigenschaft stets in Richtung des steilsten Anstiegs von E zu zeigen (Differenzierbarkeit vorausgesetzt). Um E so stark wie möglich zu verkleinern verändert man die Gewichte wiefolgt:

$$\Delta w_{ik} = -\eta\frac{\partial E}{\partial w_{ik}} \tag{4.15}$$

Das negative Vorzeichen macht deutlich, daß wir uns in entgegengesetzter Richtung (die des steilsten Abstiegs) bewegen. Der Erfolg der Gewichtsanpassung hängt wesentlich von der Wahl des Parameters η (Lernrate) ab.
Aus (14) folgt damit:

$$\Delta w_{ik} = -\eta\frac{\partial E}{\partial w_{ik}} = \eta\sum_\mu(\zeta_i^\mu - O_i^\mu)\,\xi_k^\mu \tag{4.16}$$

Man kann die Gewichte auch für jedes Eingabe-Pattern einzeln anpassen:

$$\Delta w_{ik} = \eta\,(\zeta_i^\mu - O_i^\mu)\,\xi_k^\mu \tag{4.17}$$

oder

$$\Delta w_{ik} = \eta\,\delta_i^\mu\xi_k^\mu \quad \text{mit den Fehlern (Delta's)} \tag{4.18}$$

$$\delta_i^\mu = (\zeta_i^\mu - O_i^\mu) \tag{4.19}$$

(17) nennt man auch **Delta-Regel** und hat exakt diesselbe Gestalt wie die Lernregel (10) bei den Netzen mit Schwellwertneuronen, was man auch als Bestätigung der Hebb-Regel deuten kann. Der Ansatz mit dem Gradientenabstiegverfahren ist jedoch viel allgemeiner und wird auch später bei den mehrschichtigen Netzen angewendet.

Man beachte die Unterschiede zu den Neuronalen Netzen mit Schwellwertneuronen:

- das Gradientenabstiegsverfahren benötigt eine differenzierbare Aktivierungsfunktion
- Bedingungen für die Lösbarkeit (lineare Trennnbarkeit vs. lineare Unabhängigkeit)
- der Lernalgorithmus für Schwellwertneuronen–Netze findet eine Lösung in endlich vielen Schritten während ein Algorithmus mit dem Gradientenabstiegsverfahren theoretisch unendlich lange läuft

Die **Konvergenz** des Gradientenabstiegsverfahrens ist gesichert, wenn die Lernrate η einen bestimmten Wert nicht überschreitet (für Insider: $\eta < 1/a_\lambda^{max}$, wobei a_λ^{max} der größte Eigenwert der diagonalisierten quadratischen Form von E ist). Im Fall der linearen Unabhängigkeit der Eingabe–Muster hat die Kostenfunktion nur das eine Minimum mit $E = 0$, d. h. die zugehörigen Gewichte würden das Problem exakt lösen.

4.2.3 Nichtlineare Neuronen

Die linearen Neuronen haben den Nachteil, daß ihre Aktivierungsfunktion für $x \to \pm\infty$ kein asymptotisches Verhalten zeigen, sondern ebenfalls gegen $\pm\infty$ divergieren. Dies verlangsamt u. U. die Konvergenz doch erheblich. Die Aktivierungsfunktion der Schwellwertneuronen dagegen nimmt nur zwei Zustände (z. B. $0/1, +1/-1$) an. Man ist daher an differenzierbaren Aktivierungsfunktionen interessiert, die für $x \to \pm\infty$ gegen zwei verschiedene Werte konvergieren und "dazwischen" trotzdem noch ein quasi–lineares Verhalten zeigt.
Wir geben hier zwei wichtige differenzierbare Aktivierungsfunktionen mit ihren Ableitungen an:

$$\begin{aligned} g(h) &= \tanh(\beta h) \qquad \text{Wertebereich }]-1;+1[\\ g'(h) &= \beta(1-g^2(h)) \end{aligned}$$

$$\begin{aligned} g(h) &= f_\beta(h) = \frac{1}{1+\exp^{-2\beta h}} \qquad \text{Wertebereich }]0;1[\\ g'(h) &= 2\beta g(h)(1-g(h)) \end{aligned}$$

mit einem wahlfreien Parameter $\beta > 0$.
Diese Funktionen haben den großen praktischen Vorteil, daß man ihre Ableitungen, die man ja in jedem Lernschritt auch bestimmen muß, aus der Funktion selbst berechnen kann.
Der gravierendste Nachteil linearer Neuronen zeigt sich erst beim Einsatz in mehrschichtigen Feed–Forward–Netzen. Da jede Schicht eine lineare Transformation ihrer Eingaben durchführt und eine lineare Transformation davon wieder eine lineare Transformation ist, können mehrschichtige Netze mit linearen Neuronen nicht mehr Probleme lösen als Simple Perceptrons.
Für Simple Perceptrons mit differenzierbarer Aktivierungsfunktion ergibt sich

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i\mu}(\zeta_i^\mu - O_i^\mu)^2 = \frac{1}{2}\sum_{i\mu}\left(\zeta_i^\mu - g\left(\sum_k w_{ik}\xi_k^\mu\right)\right)^2 \tag{4.20}$$

und daraus die partiellen Ableitungen (mit Kettenregel):

$$\frac{\partial E}{\partial w_{ik}} = -\sum_\mu(\zeta_i^\mu - g(h_i^\mu))g'(h_i^\mu)\xi_k^\mu \tag{4.21}$$

und der Gewichtsveränderung

$$\Delta w_{ik} = -\eta\frac{\partial E}{\partial w_{ik}} = \eta\sum_\mu(\zeta_i^\mu - g(h_i^\mu))g'(h_i^\mu)\xi_k^\mu \tag{4.22}$$

wenn man die Gewichte nach jeder Einzelpräsentation eines Eingabe–Musters anpassen will ergibt sich:

$$\begin{aligned} \Delta w_{ik} &= \eta\delta_i^\mu\xi_k^\mu \qquad \text{mit} \\ \delta_i^\mu &= (\zeta_i^\mu - O_i^\mu)g'(h_i^\mu) \end{aligned} \tag{4.23}$$

eine Delta–Regel mit dem zusätzlichen Faktor der Ableitung $g'(h)$ gegenüber der Delta–Regel für lineare Neuronen (18).
Die allgemeine Bedingung für die Existenz einer perfekten Lösung ist dieselbe wie im linearen

Fall, nämlich die lineare Unabhängigkeit der Eingabe-Vektoren. Man erhält die gleiche Lösung auch mit einem Simple Perceptron aus linearen Neuronen, wenn man statt ζ_i^μ, als Zielwerte der Output-Neuronen $g^{-1}(\zeta_i^\mu)$ vorgibt (die Aktivierungsfunktion soll streng monoton wachsend und daher umkehrbar sein).
Das Gradientenabstiegsverfahren kann jedoch nur dann eine perfekte Lösung ($E = 0$) finden, wenn die Zielwerte ζ_i^μ im Wertebereich der Aktivierungsfunktion liegen, also $g^{-1}(\zeta_i^\mu)$ überhaupt existiert. Ansonsten wird es Konvergenzprobleme geben oder nur ein **lokales Minimum** (mit $E > 0$) erreichbar sein.

4.3 Mehrschichtige Feedforward-Netze

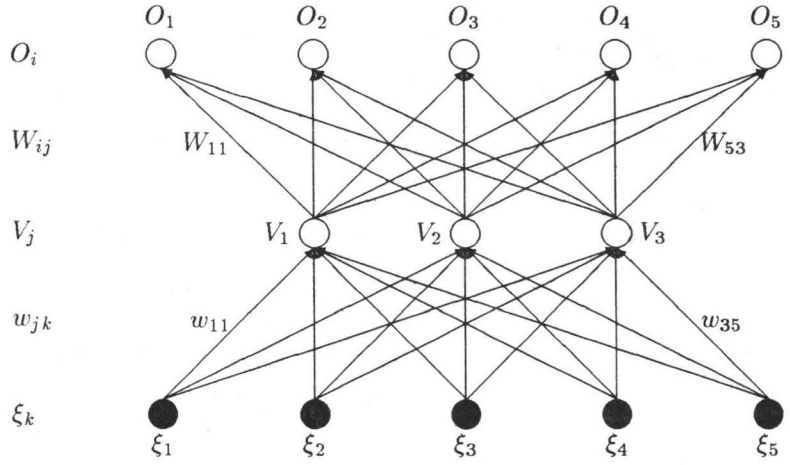
Die lineare Trennbarkeit schränkt die Verwendung von Simple Perceptrons sehr stark ein. Es ist, wie wir gesehen haben, nicht einmal möglich alle logischen Funktionen damit exakt zu realisieren. Linear nicht trennbare Zurordnungsprobleme lassen sich mit mehrschichtigen Feedforward-Netzen lösen, sofern die Aktivierungsfunktion differenzierbar und nicht linear ist.

4.3.1 Backpropagation — Theoretische Grundlagen

Obwohl man sich der Mächtigkeit von Multi-Layer-Netzwerken bewußt war, gab es lange Zeit keine Lernregel für die Gewichte der "inneren" Neuronenverbindungen.
Der bis heute wichtigste Lernalgorithmus, von 1969-85 mehrfach unabhängig voneinander entwickelt, ist der sogenannte Fehler-Rückpropagierungsalgorithmus (Error-**Backpropagation**-Algorithmus). Sein Name verrät schon, wie die Anpassung der Gewichte erfolgt. Nachdem man einen Eingabevektor durch das Netz durchpropagiert und die Fehler bei den Output-Neuronen berechnet hat, erfolgt die Gewichtsanpassung schichtenweise in rückwärtiger Richtung. Einen detaillierten Algorithmus werden wir später angeben.
Mathematische Grundlage der Backpropagation ist das Gradientenabstiegverfahren (für eine Kostenfunktion), so wie es im letzten Abschnitt schon eingeführt wurde.
Für die folgenden mathematischen Untersuchungen werden wir uns der Einfachheit halber auf ein zweischichtiges Feedforward-Netz (jeweils ein Input-, Hidden- und Output-Layer) beschränken. Der allgemeine Fall, also Netze mit beliebig vielen Hidden-Layern, läßt sich daraus problemlos ableiten.

Einige Bezeichungen:
Die Ausgangsaktivitäten der Output-Neuronen bezeichen wir mit O_i, die der Hidden-Neuronen mit V_j und die Eingabe über die Input-Neuronen mit ξ_k.
Die Gewichte der Verbindungen zwischend den Input- und Hidden-Neuronen seien mit w_{jk}, die zwischen den Hidden- und Output-Neuronen mit W_{ij} bezeichnet. Der Index i benennt die Output-Neuronen, j die Hidden- und k die Input-Neuronen, von denen es N gebe.

Beispiel: 2-schichtiges Netz mit 5 Input-, 3 Hidden- und 5 Output-Neuronen (5-3-5-Enkoder)



Es sei ein Trainingsset mit p Ein–/Ausgabe–Mustern (ξ^μ, ζ^μ), $(\mu = 1, \ldots, p)$ gegeben.
Mit einer Kostenfunktion E messen wir den Gesamtfehler:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)^2 \geq 0 \tag{4.24}$$

mit dem Gewichtsvektor $\mathbf{w} = (w_{11}, \ldots, W_{11}, \ldots)$, der alle Gewichte im Netz enthält. Diese sollen so gewählt werden, daß E minimal wird.

Berechnung der Netz–Ausgabe, bei Eingabe des Vektors ξ^μ aus dem Trainingsset:

nachdem man ξ^μ dem Input–Layer präsentiert hat, erhält das Hidden–Neuron j die Eingabe

$$h_j^\mu = \sum_k w_{jk} \xi_k^\mu \tag{4.25}$$

und erzeugt als Ausgabe

$$V_j^\mu = g(h_j^\mu) = g\left(\sum_k w_{jk} \xi_k^\mu \right) \tag{4.26}$$

Das Output–Neuron i empfängt nun die Eingabe

$$h_i^\mu = \sum_j W_{ij} V_j^\mu = \sum_j W_{ij}\, g\left(\sum_k w_{jk} \xi_k^\mu \right) \tag{4.27}$$

und erzeugt als Netz–Ausgabe

$$O_i^\mu = g(h_i^\mu) = g\left(\sum_j W_{ij} V_j^\mu \right) = g\left(\sum_j W_{ij}\, g\left(\sum_k w_{jk} \xi_k^\mu \right) \right) \tag{4.28}$$

Es sei noch bemerkt, daß jedes Neuron eine implizite 'Erregungsschwelle', so wie früher, in Form einer zusätzlichen Verbindung w_{j0}, W_{i0} mit konstanter Eingabe -1 besitzt.
Mit (28) folgt für die Kostenfunktion (24):

$$E(\mathbf{w}) = \frac{1}{2} \sum_{\mu i} \left[\zeta_i^\mu - g\left(\sum_j W_{ij}\, g\left(\sum_k w_{jk} \xi_k^\mu \right) \right) \right]^2 \tag{4.29}$$

Dies ist eine nach allen w_{jk}, W_{ij} differenzierbare Funktion, so daß man auch hier das Gradientenabstiegverfahren zur Konstruktion eines Lernalgorithmus heranziehen kann.

Betrachten wir zunächst den einfacheren Fall der Hidden–Output–Neuronenverbindungen W_{ij}. Die benötigten partiellen Ableitungen bestimmen wir mit der Kettenregel der mehrdimensionalen Analysis aus den Gleichungen (24), (27) und (28):

$$\begin{aligned}
\frac{\partial E}{\partial W_{ij}} &= \sum_\mu \sum_l \frac{\partial E}{\partial O_l^\mu} \cdot \frac{\partial O_l^\mu}{\partial W_{ij}} \\
&= \sum_\mu \frac{\partial E}{\partial O_i^\mu} \cdot \frac{\partial O_i^\mu}{\partial W_{ij}} && \text{da } W_{ij} \text{ nur in } O_i^\mu \text{ vorkommt} \\
&= -\sum_\mu (\zeta_i^\mu - O_i^\mu) \cdot \frac{\partial O_i^\mu}{\partial W_{ij}} && \text{mit (24)} \\
&= -\sum_\mu (\zeta_i^\mu - O_i^\mu) g'(h_i^\mu) \frac{\partial h_i^\mu}{\partial W_{ij}} && \text{mit (28)} \\
&= -\sum_\mu (\zeta_i^\mu - O_i^\mu) g'(h_i^\mu) V_j^\mu && \text{mit (27)}
\end{aligned}$$

und damit

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} = \eta \sum_\mu (\zeta_i^\mu - O_i^\mu) g'(h_i^\mu) V_j^\mu = \eta \sum_\mu \delta_i^\mu V_j^\mu \tag{4.30}$$

wobei

$$\delta_i^\mu = g'(h_i^\mu)(\zeta_i^\mu - O_i^\mu) \tag{4.31}$$

Dieses Ergebnis entspricht genau dem bei den Simple Perceptrons (Gl. (23)), wenn man die Ausgaben V_j^μ der Hidden–Neuronen dort als Eingaben für den Output–Layer interpretiert.
Nun zum Fall der Input–Hidden–Neuronenverbindungen w_{jk}. Hier brauchen wir die Kettenregel auch für die innerenn Ableitungen. Aus (24)–(28) folgt

$$\begin{aligned}
\frac{\partial E}{\partial w_{jk}} &= \sum_\mu \sum_l \frac{\partial E}{\partial V_l^\mu} \cdot \frac{\partial V_l^\mu}{\partial w_{jk}} \\
&= \sum_\mu \frac{\partial E}{\partial V_j^\mu} \cdot \frac{\partial V_j^\mu}{\partial w_{jk}} && \text{da } w_{jk} \text{ nur in } V_j^\mu \text{ vorkommt} \\
&= \sum_\mu \left(\sum_i \frac{\partial E}{\partial O_i^\mu} \cdot \frac{\partial O_i^\mu}{\partial V_j^\mu} \right) \cdot \frac{\partial V_j^\mu}{\partial w_{jk}} && \text{mit Kettenregel} \\
&= \sum_\mu \left(\sum_i -(\zeta_i^\mu - O_i^\mu) g'(h_i^\mu) \frac{\partial h_i^\mu}{\partial V_j^\mu} \right) \cdot g'(h_j^\mu) \frac{\partial h_j^\mu}{\partial w_{jk}} && \text{mit (24),(28),(26)} \\
&= -\sum_{\mu i} (\zeta_i^\mu - O_i^\mu) g'(h_i^\mu) W_{ij} \cdot g'(h_j^\mu) \xi_k^\mu && \text{mit (25),(27)}
\end{aligned}$$

und damit

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} = \eta \sum_\mu \delta_i^\mu \xi_k^\mu \tag{4.32}$$

wobei

$$\delta_j^\mu = g'(h_j^\mu) \cdot \sum_i W_{ij} \delta_i^\mu \tag{4.33}$$

und δ_i^μ wie in (31).

Die Gleichungen (30) und (32) haben also die gleiche Form, nur die δ's werden unterschiedlich berechnet.

Während die δ's für die Output–Neuronen (31) direkt über deren Fehlverhalten berechnet werden, sind die δ's der Hidden–Neuronen rekursiv zu ermitteln. Die Berechnung der Gewichtsveränderungen Δw erfolgt also zunächst für die Verbindungen zu den Output–Neuronen und, im allg. Fall mehrschichtiger Netze, dann Schicht für Schicht rückwärts in Richtung der Eingabeschicht. Dieses Vorgehen gab dem hier besprochenen Algorithmus seinen Namen:

Fehler-Rückpropagierungsalgorithmus

Mit den Gleichungen (32) und (33) läßt sich auch die allgemeine Delta–Regel für beliebige Feedforward–Netze herleiten. Dazu nummerieren wir die Layer von 0 (Input–Layer) bis M (Output–Layer) durch und versehen die davon abhängigen Größen mit einem entsprechenden Layer–Index m.
Für das Gewicht w_{pq}^m der Verbindung von Neuron q im m-1–ten Layer zum Neuron p des m–ten Layers ergibt sich

$$\Delta w_{pq}^m = \eta \sum_\mu \delta_p^{m,\mu} \cdot V_q^{m-1,\mu} \tag{4.34}$$

mit

$$\delta_p^{m,\mu} = \begin{cases} g'(h_p^{M,\mu}) \cdot (\zeta_p^\mu - O_p^\mu) & , \text{ für Output–Layer } m = M \\ g'(h_p^{m,\mu}) \cdot \sum_r w_{rp}^{m+1} \delta_r^{m+1,\mu} & , \text{ sonst } \quad (m < M) \end{cases} \tag{4.35}$$

dabei setzt man $V_k^{0,\mu} \leftarrow \xi_k^\mu$ für den Input–Layer
Mathematisch gelingt der Nachweis der Rekursionsformel für die δ's durch mehrfaches Anwenden der Kettenregel.

4.3.2 Backpropagation–Algorithmus

Die Kostenfunktion E, so wie sie bisher betrachtet wurde, berücksichtigt alle Ein–/Ausgabe–Muster des Trainingssets. Entsprechend ist die allgemeine Delta–Regel (34) definiert.
Für die folgende algorithmische Formulierung der Backpropagation ist es jedoch zweckmäßig die Anpassung der Gewichte Pattern für Pattern vorzusehen. Der andere Fall, also die Veränderung der Gewichte erst nach Präsentation aller Elemente des Trainingssets, läßt sich leicht auf den vorgenannten Fall zurückführen.

Algorithmus:

Gegeben:

- ein Feedforward–Netz mit M Layern
- mit den gewichteten Neuronenverbindungen von Layer m-1 zu Layer m: w_{ij}^m
- eine differenzierbare Aktivierungsfunktion g
- eine Lernrate η
- ein Trainingsset: ξ^μ Netzeingaben, ζ^μ erwartete Ausgaben

Ablauf:

1. Initialisieren aller Gewichte $w_{ij}^m, m = 0 \ldots M$ mit (kleinen) Zufallswerten
2. Wahl eines Musterpaares μ aus dem Trainingsset:
   Netzwerk–Input: ξ^μ
   erwünschte Ausgabe: ζ^μ
   und dem Input–Layer das Eingabe–Pattern präsentieren:

$$V_k^0 = \xi_k^\mu \qquad \text{für alle Input–Neuronen k}$$

3. Durchpropagieren des Eingabevektors durch das Netz:
   für $m = 1, \ldots, M$:

$$V_i^m = g(h_i^m) = g\left(\sum_j w_{ij}^m V_j^{m-1}\right) \qquad \text{für alle Neuronen i des m–ten Layer}$$

   (bis die Netzwerkausgaben $O_i^\mu = V_i^M$ berechnet sind)

4. Berechnung der δ's für den Output–Layer:

$$\delta_i^\mu = g'(h_i^\mu)(\zeta_i^\mu - O_i^\mu) \qquad \text{für alle i}$$

   (Vergleich der aktuellen Ausgabe O_i^μ mit der erwünschten Ausgabe ζ_i^μ)

5. Berechnung der δ's der vorangehenden Layer durch Rückwärtspropagierung des Fehlers:
   für $m = M-1, \ldots, 1$:

$$\delta_i^m = g'(h_i^m) \sum_j w_{ji}^{m+1} \delta_j^{m+1} \qquad \text{für alle Neuronen i des m-1–ten Layers}$$

6. Bestimmung der Gewichtsveränderungen:

$$\Delta w_{ij}^m = \eta\, \delta_i^m V_j^{m-1} \qquad \text{für alle Neuronenverbindungen}$$

7. und Setzen der neuen Gewichte:

$$w_{ij}^m \leftarrow w_{ij}^m + \Delta w_{ij}^m \qquad \text{für alle Layer m, Neuronen i,j}$$

8. weiter bei Schritt 2 mit dem nächsten Musterpaar

4.3.3 Bemerkungen zum vorgestellten Algorithmus

Wie oben erwähnt kann man die Anpassung der Gewichte (Schritt 6) in zweierlei Weise vornehmen:

a) man verändert die Gewichte, so wie angegeben, nach jeder Präsentation eines Musters

b) man paßt die Gewichte erst an, wenn man alle Paare des Trainingssets genau einmal verarbeitet hat. Dazu müssen in Schritt 6 die Gewichtsveränderungen Δw_{ij} für alle Musterpaare μ gesammelt werden (zusätzlicher Speicheraufwand nötig), ehe die Gewichtsanpassung in Schritt 7 in Kraft tritt — effektiv also
   $\Delta w_{ij}^m = \eta \sum_\mu \delta_i^m V_j^{m-1}$ gilt.

In beiden Fällen a) und b) sind die Muster (ξ^μ, ζ^μ) in zufälliger Weise dem Algorithmus zuzuführen (Schritt 2), da es sonst zu einer erheblichen Konvergenzverlangsamung kommen kann.
Eine Abbruchbedingung für den Algorithmus wurde nicht formuliert, sie hängt von der Anwendung ab (Hintergrund: Minimierung der Kostenfunktion, wie genau arbeitet das Netz).

Welches der o. a. Verfahren effizienter ist, hängt natürlich im einzelnen von der Anwendung ab. Meistens jedoch ist der inkrementelle Ansatz (a) vielversprechender.

Komplexitätsbetrachtungen

- die Delta–Regel (Schritt 6) für die Gewichte benötigt nur 'lokale', den Neuronenverbindungen zugeordnete Daten (die Aktivitäten V_j,die δ's). Dies macht den Backpropagation–Algorithmus interessant für parallele Verarbeitung
- hat man ein Netz mit etwa n Neuronenverbindungen insgesamt und wollte man die δ's direkt aus den Ableitungen der Kostenfunktion ermitteln, so braucht man $O(n^2)$ Rechenoperationen. Die rekursive Bestimmung vermindert diese Berechnungskomplexität auf $O(n)$.
- es gibt "gute" Aktivierungsfunktionen, deren Ableitungen sich direkt aus den Funktionswerten selbst berechnen lassen. Zwei solcher Funktionen wurden schon vorgestellt

4.3.4 Probleme des BP–Algorithmus, Lokale Minima

Konvergenz

Die Konvergenz des Gradientenabstiegverfahrens ist gesichert, wenn die gewünschten Werte der Output–Neuronen (Zielpattern ζ_i^μ) im Wertebereich der Aktivierungsfunktion liegen. Doch mehr noch als bei Simple Perceptrons können lokale Minima der Kostenfunktion das Finden einer optimalen Lösung ($\rightarrow E = 0$) verhindern. Dies hängt sehr stark von der Größe der Anfangsgewichte, als auch der Lernrate η ab. Es kann nicht einmal eine Aussage darüber getroffen werden, wann und unter welchen Bedingungen lokale Minima überhaupt auftreten.
Wann soll man das Verfahren abbrechen, wenn man über die (Güte der) Konvergenz nichts genaues weiß ?

Lokale Minima

Es gibt kein Patentrezept um lokale Minima algorithmisch in den Griff zu bekommen, doch gibt es einige grundsätzliche Maßnahmen die helfen, das "Hängenbleiben" in lokalen Minima zu vermeiden:

- Initialisierung der Gewichte mit kleinen Zufallszahlen (< 0.3)
- Initialisierung der Gewichte so, daß die Netzeingabe h_i eines Neurons klein, aber nicht viel kleiner als 1 ist, z. B. $w_{ij}^{init} = 1/\sqrt{k_i}$, wenn k_i die Anzahl der in Neuron i eingehenden Verbindungen ist
- Verarbeitung der Musterpaare in zufälliger Reihenfolge und sofortige Anpassung der Gewichte
- unter Umständen explizit, von Zeit zu Zeit kleinste zufällige Veränderungen der Gewichte oder der Eingabe–Pattern vonehmen

Beispiel: 8–3–8–Enkoder

Ein Feedforwardnetz, daß einen 8–3–8–Enkoder realisiert hat die Aufgabe die Eingabevektoren bei denen nur jeweils eine der Komponenten einen Wert 1 hat (der Rest ist 0) durch einen "Flaschenhals" von 3 Hidden–Neuronen an den Output–Layer durchzuschalten (vgl. 5–3–5–Enkoder in der Abbildung zu Beginn des Abschnitts).

Testdaten und Parameter:

- Eingabe– und Zielpattern: die 8 1–aus–8-Kodes
- nichtlineare Neuronen mit $g(h) = 1/(1 + \exp^{-8h})$
- Lernrate: $\eta = 0.2$
- 600 Lernepochen (d. h. 600 × Gewichtsanpassung / gesamtes Trainingsset)

Nach der Trainingsphase wurde das Netz getestet, mit folgenden Ergebnissen:

Dargestellt sind jeweils die Aktivierungen der Input– und Output–Neuronen bzw. Input– und Hidden–Neuronen:

Eingabemuster	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8
(10000000)	0.932025	0.0000004	0.0274214	0.0340411	0.0000018	0.0000618	0.0003381	0.0100843
(01000000)	0.0000515	0.948701	0.0000666	0.0000869	0.0415415	0.0240926	0.0252561	0.00703439
(00100000)	0.042101	0.0000144	0.946569	0.00344849	0.0305793	0.0000000	0.0338305	0.0000041
(00010000)	0.0338927	0.0000329	0.00264335	0.935431	0.0298745	0.0442375	0.0000002	0.0000115
(00001000)	0.0000418	0.0315029	0.0393059	0.0464123	0.941522	0.0001314	0.0002387	0.0000008
(00000100)	0.00260164	0.0260671	0.0000034	0.0434301	0.0001935	0.941102	0.0000113	0.0520025
(00000010)	0.00252659	0.0225344	0.037343	0.0000015	0.0001029	0.0000005	0.950327	0.045795
(00000001)	0.0558462	0.00633619	0.0000254	0.0000604	0.0000002	0.031258	0.0295593	0.936167

Eingabemuster	V_1	V_2	V_3	Kodierung
(10000000)	0.99347	0.814259	0.989049	(111)
(01000000)	0.0192114	0.0103515	0.117895	(000)
(00100000)	0.922209	0.994625	0.0236821	(110)
(00010000)	0.0813686	0.993597	0.886491	(011)
(00001000)	0.0132483	0.759521	0.00903372	(010)
(00000100)	0.0062582	0.261739	0.989445	(001)
(00000010)	0.966943	0.173325	0.0115075	(100)
(00000001)	0.724423	0.00466681	0.8844475	(101)

Wenn man sich die resultierenden Aktivierungen der 3 Hidden–Neuronen ansieht, erkennt man, daß das Netz im Prinzip den Binärcode "entdeckt" hat, um die 8 Eingabevektoren durch den Engpaß des Hidden–Layers durchzuschleusen. Man hat also eine Art von interner Datenspeicherung in einem Feedforwardnetz mit verborgenen Schichten.

4.3.5 Konvergenzbeschleunigung

Zum Abschluß des Kapitels sollen noch zwei allgemeine Möglichkeiten zur Konvergenzbeschleunigung (Varianten) des Backpropagation–Algorithmus diskutiert werden

Momentum–Parameter

Die Größe der Lernrate η bestimmt wesentlich die Güte der Konvergenz des Gradientenabstiegverfahrens. Ist η zu klein, so konvergiert das Verfahren unnötig langsam, andererseits neigt das Verfahren zum Oszillieren wenn η zu groß ist. Um beiden Effekten entgegenzuwirken führt man einen sogenannten Momentum–Parameter α ein. Bei jeder Gewichtsveränderung berücksichtigt man zusätzlich noch einen Anteil (α) der vorangegangenen Gewichtsveränderung:

$$\Delta w_{pq}(t+1) = -\eta \frac{\partial E}{\partial w_{pq}} + \alpha \cdot \Delta w_{pq}(t)$$

Dies hat einen langfristig dämpfenden Einfluß auf das Verfahren. α muß einen Wert zwischen 0 und 1 haben; oft bewährt sich ein Wert $\alpha = 0.9$.
Für eine Phase konstant bleibender Gewichtsanpassungen (d. h. $\Delta w_{pq}(t+1) \approx \Delta w_{pq}(t)$) folgt aus obiger Gleichung:

$$\Delta w_{pq} \approx -\frac{\eta}{1-\alpha} \cdot \frac{\partial E}{\partial w_{pq}}$$

was eine (höhere) effektive Lernrate von $\frac{\eta}{1-\alpha}$ bedeutet. Insgesamt kann man eine Beschleunigung des Verfahrens um den Faktor $\frac{1}{1-\alpha}$ erwarten, ohne den Einfluß von Oszillationen zu vergrößern.

Adaptive Parameter

Der Einfluß von Lernrate (η) und Momentum (α) auf das Verfahren ist erheblich. Beide Grösen bleiben während des Verfahrens konstant. Stellt sich aber deren anfängliche Wahl nach einigen Schritten als ungünstig heraus, ist es ratsam auch sie ggf. zu verändern.

Beispiel: Lernrate η
Man sollte η erhöhen, wenn sich die Kostenfunktion E über längere Zeit ständig verringert. Sollte sich E andererseits vergrößern, so muß man η verkleinern:

$$\Delta\eta = \begin{cases} +a & : \quad \Delta E < 0 \quad \text{während der letzten k Schritte} \\ -b\eta & : \quad \Delta E > 0 \\ 0 & : \quad \text{sonst} \end{cases}$$

wobei ΔE die Änderung der Kostenfunktion während der letzten Schritte bzw. des letzten Schrittes darstellt; a und b sind bestimmte, positive Konstanten.

4.4 Zusammenfassung

Wir haben in diesem Kapitel Feedforward–Perzeptronen daraufhin untersucht, ob und wie sie ein beliebiges vorgegebenes Assoziationsproblem lösen können, d. h. ob und wie man geeignete Gewichte der beteiligten Neuronenverbindungen findet. Anhand der Simple Perceptrons haben wir dann verschiedene Arten von Neuronen und ihren Einfluß auf die Lösbarkeit des Zuordnungsproblems kennengelernt:

Schwellwertneuronen haben nur zwei diskrete Zustände und können verwendet werden um z. B. Boolesche Funktionen zu realisieren. Doch dabei wurde deutlich, daß nur solche Probleme vernünftig gelöst werden konnten, bei denen die beiden Mengen der Musterpaare mit gleichen Zielzuständen **linear trennbar** waren. Dies trifft z. B. auf die XOR–Funktion nicht zu. Auch haben wir eine erste **Lernregel** zur Justierung der Gewichte kennengelernt: die **Hebb–Regel**. Im Fall der linearen Trennbarkeit ist damit eine Lösung in endlich vielen Schritten erreichbar.
Für **lineare Neuronen** gibt es, ähnlich der linearen Trennbarkeit, eine noch stärkere Einschränkung für die Klasse der lösbaren Probleme: die **lineare Unabhängigkeit** der Eingabemuster. Eine ideale Lösung schon für boolesche Funktionen ist also undenkbar. Trotzdem kam zum ersten Mal eine auch für andere Neuronenarten vielversprechende Methode zum Einsatz: das **Gradientenabstiegverfahren**. Mit ihm war eine genauere mathematische Analyse des Verhaltens von Perceptrons möglich. Mit einer **Kostenfunktion** kann man den Fehler und damit die Leistungsfähigkeit des Netzes messen. Darauf aufbauend konnte dann, im Gegensatz zur Hebb–Regel, eine Lernregel hergeleitet werden.
Nichtlineare Neuronen haben eine differenzierbare, quasilineare Aktivierungsfunktion. Für Simple Perceptrons hatten wir gegenüber den linearen Neuronen zwar nichts gewonnen — doch war das Gradientenabstiegverfahren eine Wegbereitung für die Handhabbarkeit mehrschichtiger Feedforwardnetze.

Mit **mehrschichtigen Feedforward–Perzeptronen** können auch schwierigere Assoziationsprobleme gelöst werden. Mit dem Gradientenabstiegverfahren konnte eine (etwas kompliziertere) Lernregel hergeleitet werden. Diese war die Grundlage für die Formulierung des Backpropagation–Algorithmus, der heute in zahlreichen Varianten Verwendung findet. Die Konvergenz des Verfahrens macht im allgemeinen nur insofern Schwierigkeiten, daß lokale Minima in der Kostenfunktion das Finden einer optimalen (fehlerminimalen) Lösung verhindert, obwohl es eine gibt. Es gibt verschiedene Möglichkeiten um zu versuchen lokale Minima zu umgehen, doch im einzelnen hängt dies von der Anwendung ab.

Zum Abschluß des Kapitels wurden noch zwei Verfahren vorgestellt, die die Konvergenz der Backpropagation beschleunigen:
Mit dem **Momentum–Parameter** wird in jedem Lernschritt berücksichtigt, wie die Gewichte in den vorangegangenen Lernschritten verändert wurden. Man führt also keine ad–hoc–Gewichtsanpassung durch sondern beobachtet langfristig die vorgenommenen Gewichtsveränderungen. Momentum und Lernrate kann man zu **adaptiven Parametern** machen. D. h. man kontrolliert den Fortgang des Verfahrens und paßt diese dafür kritischen Parameter nach Bedarf an. Eine anfänglich falsch getroffene Wahl dieser Angaben muß daher nicht dazu führen das Verfahren zu wiederholen.

4.5 Literatur

[1] Hertz, John et ali; *Introduction to the Theory of Neural Computation*, Santa Fé Institute — Studies in Sciences of Complexity Series, Addison–Wesley, Redwood City, CA, USA, 1990.

Kapitel 5

Rekursive Neuronale Netze

Philipp Ziegler

Übersicht

Es werden Definition und Funktion von Rekursiven Neuronalen Netzen vorgestellt und begründet. Dabei wird auf bekannten Vorgehensweisen von Feedforward-Netzen aufgebaut. Einen wichtigen Anteil haben die Dynamischen Entwicklungsregeln ('Dynamic Evolution Rules'), sowie die Herleitung diverser Trainingsverfahren für Rekursive Netze. Es werden zum einen Trainingsverfahren behandelt, die den stationären Zustand trainieren (Rekursives Backpropagation). Zum anderen werden Verfahren vorgestellt, die die Bearbeitung von Zeitreihen trainieren (Real Time Recurrent Learning, Time Dependent Recurrent Backpropagation, Backpropagation Throught Time und Tapped Delay Lines). Desweiteren wird Reinforcement Learning vorgestellt und die Trainingsverfahren der Modellbildung und Associative Reward Penalty betrachtet.

5.1 Einleitung

Die auf Feedforward-Netzen beruhenden Mustererkennungsverfahren zeigen ein rein statisches Verhalten, d.h. sind nur in der Lage, unveränderliche Muster zu erkennen. In vielen Fällen ist es aber gerade interessant, bestimmte Arten von zeitlichen Veränderungen zu erkennen und darauf mit einem eigenen zeitveränderlichen Signal zu reagieren.

Die Einbeziehung der Zeit in Neuronale Netze wird durch die Einführung rekursiver Verbindungen ermöglicht. Dieses führt dazu, daß der Signalfluß nicht mehr nur vorwärts in eine Richtung verläuft, sondern auch "im Kreis" verlaufen kann, was einen zeitabhängigen Zustand des Netzes zur folge hat..

5.2 Rekursive Neuronale Netze

5.2.1 Definition

Bei rekursiven Netzen handelt es sich um eine Verallgemeinerung der beim Backpropagation verwendeten Feedforward Netzen. Rekursive Neuronale Netze werden durch allgemeine gewichtete Digraphen beschrieben, müssen also nicht wie die Feedforward-Netze zyklenfrei sein. Die Gewichtsmatrix $W = (w)_{i,j}$ läßt sich somit nicht mehr durch eine untere Dreiecksform darstellen.

Als Eingabe bzw. Ausgabe Neuronen werden die Neuronen beliebiger Mengen I bzw. O ausgezeichnet. Außerdem wird es sich zeigen, daß es notwendig ist, den Neuronen Anfangszustände zuzuweisen (siehe 5.2.2).

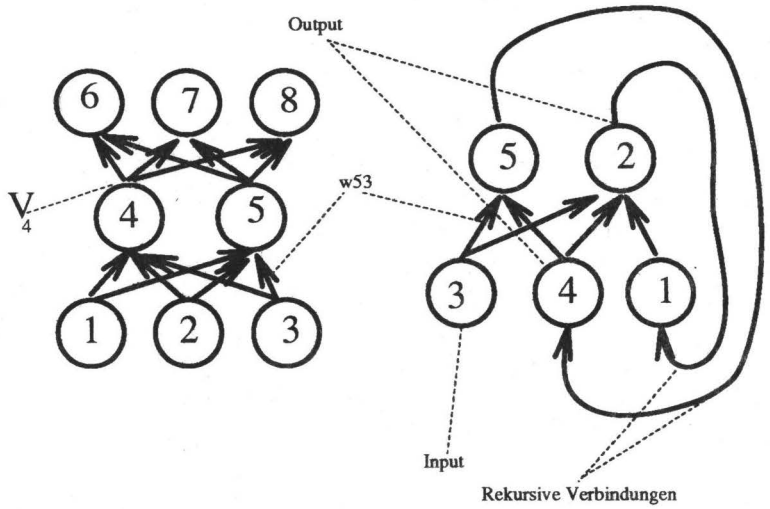


Abbildung 5.1: Beispiel eines Feedforward- und eines Rekursiven-Netzes

Im folgenden bezeichnen Indizes Nummern von Neuronen, V_i die tatsächlichen Ausgangswerte eines Neurons i, ξ den Eingabevektor und ζ den zu lernenden Ausgabevektors eines Neuronalen Netzes. Mit h_i wird im folgenden die gewichtete Eingangssumme der Aktivierungsfunktion bezeichnet, d.h.

$$h_i = \sum_j w_{ij} * V_j$$

5.2.2 Dynamische Entwicklungsregeln

Das durch ein Feedforward Netz definierte Gleichungssystem, läßt sich durch 'Forward-Propagation' lösen, d.h. der Ausgang eines Neurons wird berechnet, falls alle Eingänge schon berechnet wurden. Dieses ergibt bei Feedforward Netzen eine 'Vorwärts-Berechnung' von der Eingabeschicht bis zur Ausgabeschicht.

Daß ein solches Verfahren bei rekursiven Netzen prinzipiell nicht funktionieren kann, wird unmittelbar aus dem Beispiel in Bild 5.2.2. klar. Das durch dieses Netz definierte Gleichungssystem hat unter Voraussetzung einer Aktivierungsfunktion g=id keine Lösung.

$V_1 = -V_1 - V_2$ und $V_2 = -2 * V_1 + 1$ lassen sich umformen zu

$$2 * V_1 + V_2 = 0$$

und

$$2 * V_1 + V_2 = 1,$$

was einen Widerspruch darstellt.

Im physikalischen System führt dieses zu einem instabilen Zustand (d.h. das System schwingt).

Es ist folglich sinnvoll dynamische Entwicklungsregeln zu definieren, die ein Zeitverhalten berücksichtigen. Das dynamische Verhalten eines Rekursiven Netzes wird somit auch von der zugrunde gelegten Entwicklungsregel abhängig.

Bei Feedforward Netzen interessiert ein rein statisches Verhalten. Die Ausgabe V ist einzig und allein von der Eingabe ξ abhängig (d.h. V = F(ξ)). Bei Rekursiven Netzen ist die Ausgabe eine Zeitfunktion V(t), die sowohl von einer Zeitfunktion $\xi(t)$ als auch von der Zeit t, abhänging ist (d.h. V(t) = F(ξ,t)). Die Zeitfunktion F wird hierbei durch die Gewichtsmatrix W und die Dynamische Entwicklungsregel definiert.

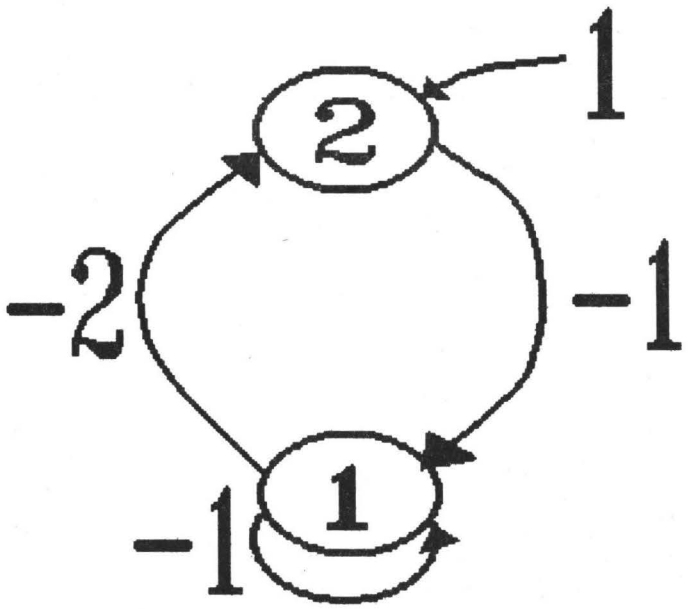


Abbildung 5.2: Ein oszillierendes Rekursives Netz

Bei der asynchronen Entwicklungsregel wird pro Zeitschritt ein einzelnes Neuron i entsprechend der Aktivierungsfunktion g aktualisiert:

$$V_k(t+\Delta t) = \begin{cases} g(\sum_j w_{kj} * V_j(t)) & : k = i \\ V_k(t) & :\text{sonst} \end{cases}$$

Bei der synchronen Entwicklungsregel werden pro Zeitschritt alle Neuronen entsprechend der Aktivierungsfunktion g aktualisiert:

$$V_k(t+\Delta t) = g(\sum_j w_{kj} * V_j(t)) = g(h_k)$$

Eine dritte Möglichkeit, für stetige Aktivierungsfunktionen, ist die stetige Entwicklungsregel, die das zeitkontinuierliche äquivalent zur synchronen Regel darstellt:
Aus dem Differenzenquotient

$$\Delta t * \frac{\partial V_k(t)}{\partial t}(t) \approx V_k(t+\Delta t) - V_k(t)$$

ergibt sich aus der synchronen Regel dann:

$$\Delta t * \frac{\partial V_k}{\partial t} = -V_k + g(h_k)$$

Bei allen aufgeführten Entwicklungsregeln müssen zudem noch die Randbedingungen, z.B. die Anfangszustände $V_k(0)$ definiert werden.

5.2.3 Rekursives Backpropagation

Im folgenden soll das Zeitverhalten der rekursiven Netze vernachläßigt werden. Es soll wie bei den Feedforward Netzen nur das statische Verhalten betrachtet werden. Hierfür werde vorausgesetzt,

daß für endlichen konstanten Eingangswert, die Ausgabe gegen einen endlichen Ausgangswert (Fixpunkt) konvergiert.

$$F(\xi) = \begin{cases} \lim_{t->\infty} F(\xi, t) & \text{:dieser lim exisitiert} \\ undef. & \text{:sonst} \end{cases}$$

Die Aktivierungsfunktion g sei stetig, als Entwicklungsregel werde die stetige Regel gewählt:

$$\Delta t * \frac{\partial V_k}{\partial t} = -V_k + g(\xi_k + h_k)$$

Als Fehlerfunktion wird wie bei den Feedforward Netzen die (zeitunabhängige!) Funktion E gewählt:

$$E = \frac{1}{2} * \sum_k E_k^2$$

$$E_k = \begin{cases} \zeta_k - V_k(\infty) & : k \in O \\ 0 & \text{:sonst} \end{cases}$$

Das Gradienten-Abstiegsverfahren für die Gewichte liefert den Ansatz:

$$\Delta w_{pq} = -\eta * \frac{\partial E}{\partial w_{pq}} = -\eta * \frac{1}{2} * \sum_k 2 * E_k * \frac{\partial E_k}{\partial w_{pq}} = \eta * \sum_k E_k * \frac{\partial V_k(\infty)}{\partial w_{pq}}$$

Durch geschicktes Zusammenfassen, Umformen, Differenzieren ergibt sich zunächst folgendes Gleichungssystem:

$$\Delta w_{pq} = \eta * \delta_p * V_q(\infty)$$

$$\delta_p = g\prime(h_p) * \sum_k E_k * (L^{-1})_{kp}$$

Hierbei ist L eine Matrix, die von den Gewichten w_{ij} und den Neuronen Ausgängen V_k abhängig ist. Abgesehen vom Aufwand für das Invertieren einer Matrix, ist diese Form des Gleichungssystems zum Training eines Neuronalen Netzes denkbar ungeeignet: Der Vorteil der Parallelisierbarkeit der voneinander unabhängigen Neuronen geht hier durch die netzglobale Operation auf L vollständig verloren.

Durch verstärktes Zusammenfassen, Umformen, Differenzieren und die Einführung einer Zwischengröße Y ergibt sich jetzt folgendes Gleichungssystem:

$$\Delta t * \frac{\partial Y_k}{\partial t} = -Y_k + E_k + \sum_p g\prime(h_p) * w_{pk} * Y_p$$

$$\delta_k = g\prime(h_k) * Y_k(\infty)$$

$$\Delta w_{kq} = \eta * \delta_k * V_q(\infty)$$

Die Idee besteht jetzt darin, die oberste Gleichung als stetige Entwicklungsregel für ein weiteres Neuronales Netz zu interpretieren. Mit Aktivierungsfunktion $g_y = id$, Eingängen $\xi_y = E$, Ausgängen $V_y = Y$ und Gewichten $w_{yij} = g'(h_j) * w_{ji}$. Dieses Hilfs-Netz wird auch als Error-Propagation-Netz bezeichnet, weil es als Eingabewerte die Fehler des Trainingsnetzes hat. Es läßt sich zeigen [*HERTZ*], daß dieses Error-Propagation-Netz konvergiert bei Eingaben E_i, falls das Trainingsnetzwerk konvergiert. Letzteres war nach Voraussetzung der Fall.

Es ergibt sich folgende Lernprozedur:

1. Berechne im Trainingsnetzwerk, die Ausgänge V_i (numerische Integration).
2. Bestimme die Fehler E_k.
3. Berechne im Error-Propagation-Netz die Ausgänge Y_k (numerische Integration).

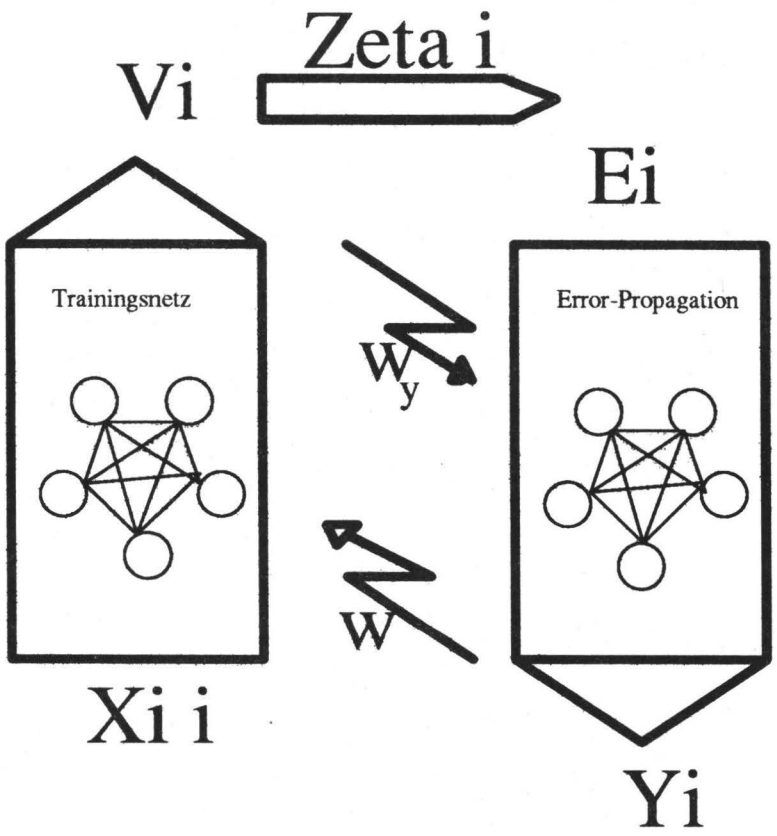


Abbildung 5.3: Trainingsverfahren bei Rekursivem Backpropagation

4. Berechne die Gewichtsänderungen und die neuen Gewichte im Trainingsnetz.

Die neuen Gewichte können im 4. Schritt mit den angegebenen Formeln unabhängig voneinander berechnet werden. Eine Parallelisierung ist deshalb möglich.

5.2.4 Vergleich von Backpropagation in Feedforward- und Rekursiven Netzen

Im Gegensatz zu Backpropagation in Feedforward Netzen, ist Rekursives Backpropagation aufwendiger (Sequentiell: $K*N^2$, bei großem K; Parallele Neuronen: K*N). Zudem hat jedes rekursive Netz, das die Voraussetzungen in 5.2.3 erfüllt und nach endlicher Zeit einen hinreichend stabilen Zustand erreicht, ein äquivalentes Feedforward Netz, das durch 'Auseinanderfalten' bezüglich der Zeit erhalten werden kann (siehe 5.3.2). Durch ein rekursives Netz kann eine Funktion in vielen Fällen allerdings mit wesentlich weniger Neuronen dargestellt werden, so daß sich für den Lernprozeß insgesamt ein starker Performance-Gewinn zeigen kann [*HERTZ*].

5.3 Zeit in Rekursiven Neuronalen Netzen

5.3.1 Einführung dynamischer Vorgänge

Das eigentliche Anwendungsgebiet von rekursiven Netzen liegt im Bereich der Verarbeitung dynamischer Eingaben. Feedforward Netze sind mit ihrem rein funktionalen Charakter, grundsätzlich nicht in der Lage Zeitvorgänge zu behandeln.

Folgende Rechnung soll zeigen, daß Rekursive Netze beispielsweise auch Zeitfunktionen erzeugen können. Hierfür werden Gewichte vorgegeben, so daß am Ausgang die Trajektorie eines Kreises beschrieben wird. Diese Gewichte könnten zum Beispiel in einem Lernprozeß erreicht worden sein.

Bei Betrachtung von Bild 5.2.2 unter Voraussetzung einer Synchronen Entwicklungsregel und einer Aktivierungfunktion g=id ergibt sich folgendes Abbildungsverhalten:

$$V_1(t) = \xi_1(t-1) + w_{11} * V_1(t-1) + w_{12} * V_2(t-1)$$
$$V_2(t) = \xi_2(t-1) + w_{21} * V_1(t-1) + w_{22} * V_2(t-1)$$

Wählen wir:

$$\xi_1 = \xi_2 = 0,$$
$$w_{11} = w_{22} = cos\varphi,$$
$$w_{21} = -w_{12} = sin\varphi,$$

sowie als Anfangswerte:

$$V_1(0) = r$$

(es wird sich zeigen, daß r=Kreisradius)

$$V_2(0) = 0$$

dann ergibt sich:

$$V_1(t) = \cos(\varphi) * V_1(t-1) - \sin(\varphi) * V_2(t-1)$$
$$V_2(t) = \sin(\varphi) * V_1(t-1) + \cos(\varphi) * V_2(t-1)$$

Als Ansatz für die Auflösung der Rekursion:

$$V_1(t) = r * \cos(t * \varphi) \tag{5.1}$$
$$V_2(t) = r * \sin(t * \varphi) \tag{5.2}$$

Die Richtigkeit kann mittels Induktion über t und $\cos((t-1)*\varphi+\varphi) = \cos((t-1)*\varphi) - \sin((t-1)*\varphi)*\sin(\varphi)$ (ergibt sich aus Additionstheorem), gezeigt werden.

Aus 5.1 und 5.2 ist wie gewünscht die Trajektorie eines Kreises erkennbar. Der Radius dieses Kreises ist nur durch den Anfangszustand bestimmt.

Aufgabenstellungen

Die Aufgabenstellungen, die bei der Behandlung von Zeitfunktionen auftreten, lassen sich in drei Kategorien aufteilen:

a) Zeitreihen Erkennung (dynamische Mustererkennung; Als Reaktion auf eine bestimmte Eingabe Zeitreihe liefert das Netz eine bestimmte Ausgabe)

b) Zeitreihen Vervollständigung (dynamische Mustervervollständigung; Das Netz erzeugt nach der Eingabe des Anfangs einer Zeitreihe selbstständig den Rest. Ziel ist, daß gewünschte Eingabezeitreihe und Ausgabezeitreihe übereinstimmen.)

c) Zeitliche Assoziation (als Antwort auf eine bestimmte Eingabezeitreihe, wird eine bestimmte Ausgabe Zeitreihe erzeugt. Im Unterschied zu b) können Eingabe- und Ausgabezeitreihe in beliebiger Beziehung zueinander stehen.)

Welche dieser Aufgabenstellungen ein Neuronales Netz erfüllen kann, ist von der Trainingsmethode abhängig.

5.3.2 Trainingsmethoden für Zeitfunktionen

Das Vorgehen beim Erstellen eines Trainingsverfahrens für ein rekursives Netz entspricht weitgehend dem Vorgehen bei Feedforward Netzen:

Zuerst wird eine Fehlerfunktion aufgestellt. Diese ist abhängig vom gewünschten und vom tatsächlichen Ausgabevektor. Bei rekursiven Netzen kann sie außerdem von der Zeit abhängig sein.

Anschließend wird auf die Fehlerfunktion ein Optimierungsverfahren (z.B. das Gradienten-Abstiegsverfahren) für die Gewichte angewendet. Mittels der sich hieraus ergebenden Gewichtsänderungen Δw_{ij} wird das Netz trainiert. Durch die Fehlerfunktion wird letztendlich bestimmt, woraufhin das Netz trainiert wird, welche Aufgabenstellungen es also erfüllen kann.

Es werden jetzt verschiedene Trainingsmethoden vorgestellt und auf ihre Leistungsfähigkeit untersucht.

Tapped Delay Lines

Bei diesem Verfahren handelt es sich nicht um eine Trainingsmethode für ein rekursives Netz, sondern um einen Versuch, einfachen Feedforward Netzen ein Zeitverhalten beizubringen. Das System besteht aus einer Reihe von hintereinandergeschalteten Verzögerungsgliedern ('delay line'), an die die Eingänge eines Feedforward Netzes angeschaltet ('tapped') werden.

Bei jedem Zeitschritt wird die Eingabe einen Eingang weitergeschaltet. Durch die Verzögerungsglieder wird die Eingangszeitfunktion in ein statisches Eingabemuster transformiert. Das Feedforward Netz wird mit gewöhnlichem Backpropagation auf diese Muster trainiert.

Günstig an diesem Verfahren ist offensichtlich, daß die bekannten Methoden des Backpropagation angewendet werden können. Diesem stehen aber grundsätzliche Beschränkungen gegenüber:

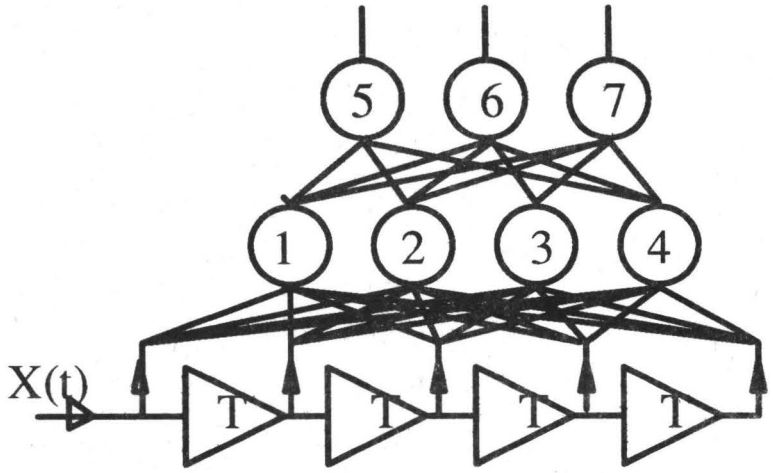


Abbildung 5.4: Ein Tapped-Delay-Line System

Zum einen bleibt das Verfahren auf Zeitreihen bestimmter Längen beschränkt (Anzahl Verzögerungsglieder), zudem muß das Signal schon zeitdiskret und synchronisiert mit der Verzögerungszeit der Verzögerungsglieder vorliegen.

Aufgabenstellungen:

Das Verfahren kann keine Zeitreihen erzeugen oder vervollständigen. Im Grunde handelt es sich eben um ein einfaches Feedforward Netz, das 'von sich aus' keine Ausgabe erzeugen kann. Unter den genannten Voraussetzungen ist das Verfahren für Zeitreihenerkennung recht einfach einsetzbar.

Backpropagation Throught Time (BPTT)

Auch beim BPTT handelt es sich nicht um ein Trainingsverfahren auf einem rekursiven Netz. Hier wird ein rekursive Netz in ein gewissermaßen äquivalentes Feedforward Netz transformiert. Auf diesem wird zum Training wiederum einfaches Backpropagation angewendet. Die somit erzielten Gewichte werden nach dem Lernen in das ursprüngliche rekursive Netz übernommen.

Für dieses Verfahren wird die synchrone Entwicklungsregel vorausgesetzt. Zudem müssen Ausgangsfunktionen nach endlicher Zeit einen annähernd stabilen Zustand erreicht haben.

Die Transformation des rekursiven Netzes in ein äquivalentes Feedforward- Netz, geschieht durch ein Auseinanderfalten des rekursiven Netzen in den verschiedenen Zeitschritten. Für jeden Zeitschritt werden 'neue' Neuronen $V_i^{(t)}$ eingeführt, deren Eingänge jeweils die gewichteten Ausgänge der Neuronen des vorhergenden Zeitschrittes d.h. $V_i^{(t-1)}$ sowie evtl. der Eingang $\xi_i^{(t)}$ sind.

Dieses äquivalente Feedforward-Netz kann jetzt mittels (fast) normalen Backpropagations trainiert werden. Als Ein-Ausgabe Paar werden im μ -ten Trainingsschritt folgende Werte gewählt:

Eingaben $\xi_i^{(t)}$ für $V_i^{(t)}$ sind:

$$\begin{cases} \xi_i(\mu - t) & : \text{falls } t <= \mu \\ beliebig & :\text{sonst} \end{cases}$$

Ausgabezielwerte $\zeta_i^{(t)}$ von $V_i^{(t)}$ sind:

$$\begin{cases} \zeta_i(\mu - t) & : \text{falls } t <= \mu \\ undef. & :\text{sonst} \end{cases}$$

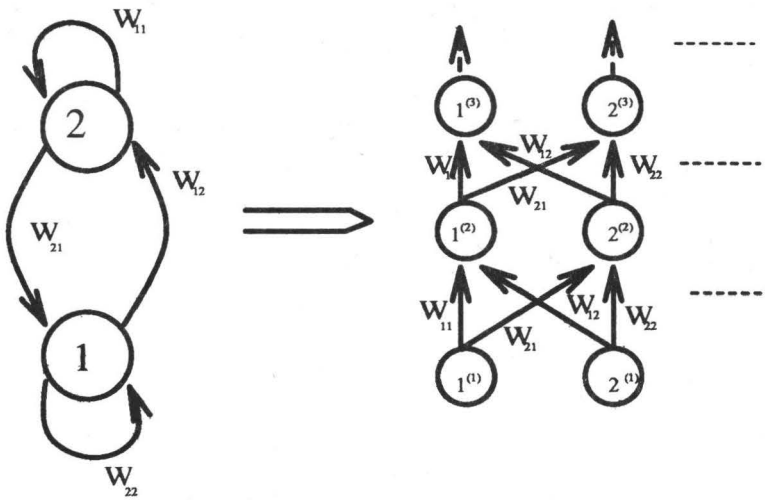


Abbildung 5.5: Transformation eines Rekursiven- in ein vergleichbares Feedforward-Netz

Wichtiger Unterschied zum "normalen" Backpropagation: Die einzelnen Gewichte tauchen im Netzwerk öfter auf, müssen somit alle einen gleichen Wert haben. Da "normales" Backpropagation eigentlich jedes Gewicht einzeln anpaßt, müssen hier Änderungen vorgenommen werden. Eine Möglichkeit wäre beispielsweise auf jede Kopie eines Gewichts, den Durschnittswert der berechneten einzelnen Änderungen anzuwenden. Eine andere Möglichkeit besteht darin, als Gesamtänderung die Gesamtsumme der einzelnen Änderungen zu benutzen.

Nach Abschluß des Trainings werden die erreichten Gewichte in das ursprüngliche 'kleine' rekursive Netz übernommen.

Aufgabenstellung:

Das Verfahren ist prinzipiel nur für Zeitreihen mit fester maximaler Länge geeignet. Mit zunehmender Länge der Zeitreihe steigen Zeit- und Speicheraufwand für das äquivalente Feedforward-Netz und das Backpropagation.

BPTT ist für allgemeine zeitliche Assoziationsaufgaben geeignet, solange Eingabe- und Ausgabezeitreihe eine Länge haben, die nicht größer als eine fester Wert ist. Die Eignung für allgemeine zeitliche Assoziationsaufgaben geht aus der Herleitung des Verfahrens unmittelbar hervor: Dem äquivalenten Feedforward-Netz werden eine Eingabe- und eine Ausgabezeitreihe in "auseinandergefalteter" Form präsentiert. Das Backpropagation ist darauf ausgelegt, für alle Ein-Ausgabe Paare eine minimale Abweichung zu erzielen; nämlich hier für eine Eingabezeitreihe die gewünschte Ausgabezeitreihe zu erzeugen.

Context Units

Bei diesem Verfahren werden nur spezielle eingeschränkte Formen rekursiver Netze behandelt. Diese partiell rekursiven Netze haben einen besonders einfachen Aufbau mit nur wenigen speziellen rekursiven Verbindungen. Die Gewichte der rekursiven Verbindungen werden als fest definiert, müssen dadurch also nicht trainiert werden.

Bild 5.3.2 zeigt gebräuchliche Netze dieser Art. Die eingezeichneten Gewichte sind alle fest, ändern sich während des Trainings also nicht:

Im Gegensatz zu einfachen Feedforward-Netzen sind hier sog. Context-Neuronen dazugekommen. Diese haben die Aufgabe, die Eingabe der Eingangsneuronen in einen zeitlichen Kontext zu setzen. Dieses geschieht durch rückkoppeln von "alten" Ausgangswerten.

Im Falle dieser partiell rekursiven Netze wird eine spezielle Entwicklungsregel zugrunde gelegt. Die rekursiven Verbindungen arbeiten entsprechend der synchronen Entwicklungsregel, d.h. hier

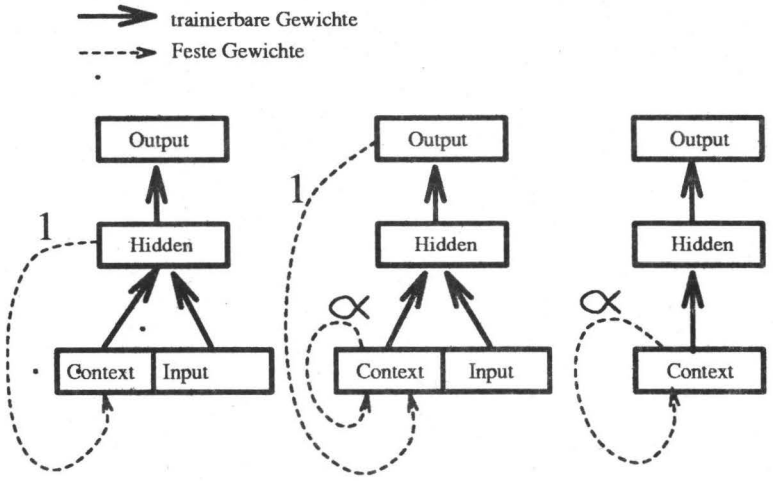


Abbildung 5.6: Verschiedene Möglichkeiten Partiell-Rekursiver Netze

im Fall...

a)

$$C_i(t+1) = 1 * H_i(t)$$

b)

$$C_i(t+1) = \alpha * C_i(t) + 1 * O_i(t)$$

c)

$$C_i(t+1) = \alpha * C_i(t)$$

(C bezeichne Kontext-Neuronen, H Hidden-Neuronen, O Ausgangs-Neuronen)

Die Vorwärts-Verbindungen sollen zeitunabhängig entsprechend einem einfachen Feedforward-Netz entwickeln. Der Faktor $0 < \alpha < 1$ beeinflußt das 'Errinnerungsvermögen' der Kontex-Neuronen. Ein großes α führt dazu, daß alte Werte die Ausgänge der Kontext-Neuronen lange dominieren und somit die Berechnung lange beeinflussen. Dieser Faktor sollte immer an die Länge der zu erkennenden Zeitreihe angepaßt werden.

Als Trainingsmethode wird einfaches Backpropagation verwendet. Dieses verläuft jetzt aber in Zeitschritten. Ein Lernalgorithmus könnte so aussehen:

1. Gib Eingabe $\xi(T)$.
2. Berechne Neuronen mit rekursiven Eingängen (entsprechend a), b) oder c))
3. Berechne Feedforward-Neuronen
4. Vergleiche mit gewünschter Ausgabe $\zeta(T)$
5. Wende Backpropagation an (Die Rückkopplungen der Kontext- Neuronen werden dabei wie Eingabewerte behandelt).
6. Erhöhe T. Weiter bei 1.

Hierbei ist wichtig, daß die Zeitreihe in der richtigen Reihenfolge (d.h. chronologisch) präsentiert wird, da hier jeder Zustand des Netzes von den (zeitabhängigen) Kontext-Neuronen abhängig ist.

Aufgabenstellungen:

Prinzipiell sind diese partiell rekursiven Netze auch für allgemeine zeitliche Assoziationsaufgaben geeignet. In der Praxis scheint es sich allerdings gezeigt zu haben, daß gute Ergebnisse nur bei der Zeitreihenerkennung geliefert werden [$HERTZ$].

Real-Time Recurrent Learning (RTRL)

Beim RTRL handelt es sich um das erste hier vorgestellte Verfahren, daß auf allgemeinen rekursiven Netzen arbeitet. Zudem ist dieses Verfahren so ausgelegt, daß es während der Präsentation der Zeitreihe lernen kann. Die Gewichtsanpassung ist nur vom aktuell präsentierten Ein-/Ausgabe Paar $(\xi(t), \zeta(t))$ und dem Zustand des Netzes abhängig und wird in jedem einzelnen Zeitschritt ausgeführt.

Für RTRL wird die synchrone Entwicklungsregel zugrunde gelegt. Die zu assoziierenden Zeitreihen können beliebige Länge haben.

Als erstes wird eine Fehlerfunktion für die einzelnen Ausgabeneuronen aufgestellt:

$$E_k(t) = \begin{cases} \zeta_k(t) - V_k(t) & : falls \zeta_k(t) definiert \\ 0 & :\text{sonst} \end{cases}$$

Hieraus ergibt sich dann als mögliche Fehlerformel für die gesamte Ausgabe:

$$E(t) = \frac{1}{2} * \sum_k E_k(t)^2$$

Beachtenswert ist, daß diese Fehlerformel von der Zeit t abhängig ist. So kann die Anforderung erfüllt werden, während der Zeitreihen-Präsentation lernen zu können. Das weitere Vorgehen ist Standard. Das Gradienten Abstiegsverfahren liefert:

$$\Delta w_{pq}(t) = -\eta * \frac{\partial E(t)}{\partial w_{pq}} = \eta * \sum_k E_k(t) * \frac{\partial V_k(t)}{\partial w_{pq}}$$

Für die Ableitung von $V_k(t)$ ergibt sich aus der synchronen Entwicklungsregel und der Kettenregel folgendes Ergebnis:

$$\frac{\partial V_k(t)}{\partial w_{pq}} = g'(h_k(t - \Delta t)) * (\gamma_{kp} * V_q(t - \Delta t) + \sum_j w_{kj} * \frac{\partial V_j(t - \Delta t)}{\partial w_{pq}}$$

, wobei

$$\gamma_{ij} = \begin{cases} 1 & :\text{i=j} \\ 0 & :\text{sonst} \end{cases}$$

Somit ist ein induktiver Zusammenhang zwischen der Ableitung zum Zeitpunkt t und zum Zeitpunkt $t + \Delta t$ hergestellt. Als Verankerung wird die plausible Randbedingung

$$\frac{\partial V_k}{\partial w_{pq}}(0) := 0$$

gewählt.

Eine Zeitreihe muß, wie auch beim einfachen Backpropagation, eventuell öfter präsentiert werden, um die gewünschte zeitliche Assoziation zu erreichen.

Aufgabenstellungen:

Der Nachteil des Verfahrens liegt in einem relativ hohen Aufwand. Im schlechtesten Fall (vollst. Graph)) müssen für N Neuronen jeweils N Gewichtsänderungen berechnet werden. Hierfür ist bei N Ausganbeneuronen (schlechtester Fall!) die Summation von N Ableitungen von V_i notwendig. Die Ableitungen berechnen sich wiederum aus einer Summe der N 'vorausgehenden' Ableitungen.

==> Aufwand in der Größenordnung von N^4 im schlechtesten Fall.

Ansonsten ist das Verfahren entsprechend der Fehlerformel für allgemeine zeitliche Assoziationsaufgaben geeignet.

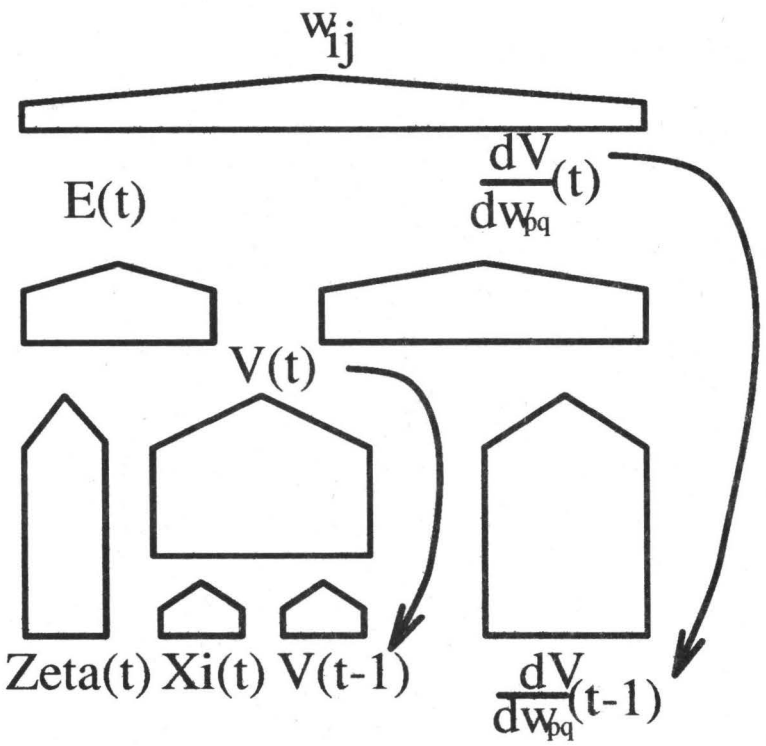


Abbildung 5.7: Trainingsprozedur beim RTRL

Time Dependent Recurrent Backpropagation (TDRBP)

TDRBP ist ein weiteres Trainingsverfahreun für allgemeine rekursive Netze. Im Gegensatz zu RTRL (und auch dem Context-Units Verfahren) muß hier jedoch vor jedem Lernschritt die gesamte Ein- und Ausgabe-Zeitfunktion vorliegen. TDRBP kann als Verallgemeinerung des Rekursiven Backpropagation (RBP) auf Zeitfunktionen betrachtet werden.

Analog zum RBP wird hier die stetige Entwicklungsregel vorausgesetzt, allerdings in leicht abgewandelter Form:

$$\Delta t_i * \frac{\partial V_i}{\partial t} = -V_i + g(h_i) + \xi_i$$

Als Fehlerfunktion wird folgende Formel gewählt:

$$E = 1/2 * \int_0^T E_k(t)^2 \partial t$$

$$E_k(t) = \begin{cases} V_k(t) - \zeta_k(t) & :k \in O und \zeta_k(t) definiert \\ 0 & :\text{sonst} \end{cases},$$

wobei T die Zeit vom Beginn der Eingabe bis zum Ende der Ausgabe ist.

Aus dem Gradienten-Abstiegsverfahren ergibt sich mit einigem mathematischen Aufwand:

$$\frac{\partial Y_i}{\partial t} = -1/\Delta t_i * Y_i + E_i + \sum_j \frac{1}{\Delta t_j} * w_{ij} * g\prime(h_j) * Y_j$$

$$\Delta w_{ij} = -\eta * \frac{\partial E}{\partial w_{ij}} = -\frac{\eta}{\Delta t_i} * \int_0^T Y_i(t) * g\prime(h_i(t)) * V_j(t) \partial t$$

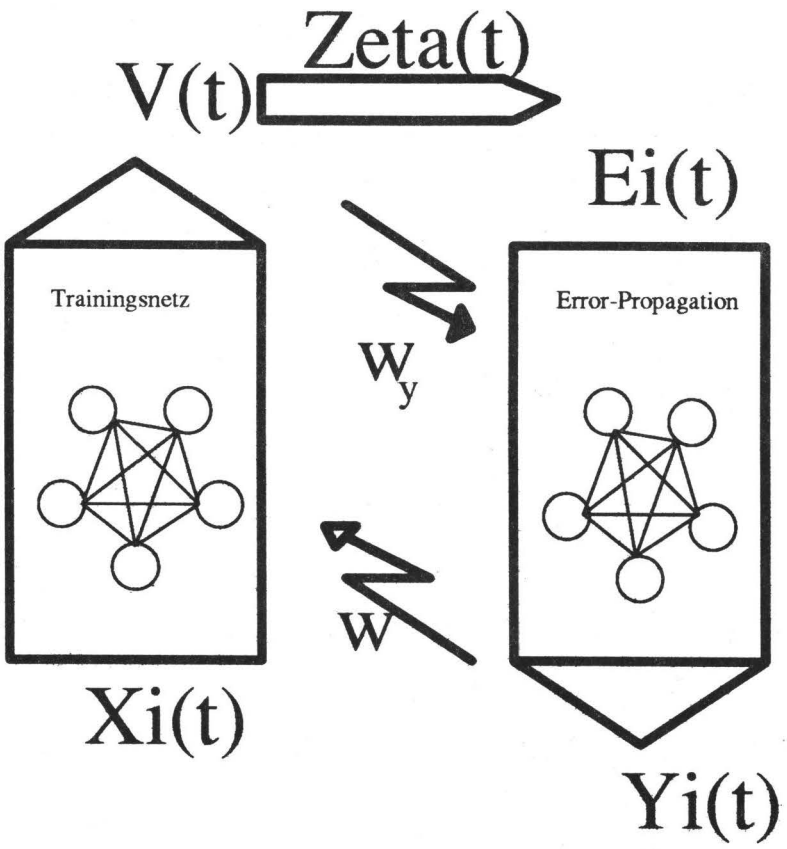


Abbildung 5.8: Trainingsprozedur beim TDRBP

Bei gleichen Δt_i entsprechen diese Formeln weitgehend denen des RBP mit dem Unterschied, daß hier $Y_i, V_i und E_i$ Funktionen der Zeit sind. Im Gegensatz zur Auswertung beim RBP müssen hier allerdings nicht nur einzelne Werte, sondern ganze Zeitfunktionen (Y_i, V_i, E_i sind jeweils abhängig von t) zwischengespeichert werden (z.B. in Form von Stützwerten).

Analog zum RBP ergibt sich hier folgendes Verfahren:

1. Berechne aus der Entwicklungsregel unter geeigneten Anfangszuständen (z.B. $V_i(0) = 0$) die Werte $V_i(t)$ (z.B. in K Stützstellen mittels eines numerischen Verfahrens)

2. Bestimme die Fehlerfunktionen E_i

3. Berechne die Funktionen Y_i (wie 1.)

4. Berechne die Gewichtsänderungen Δw_{ij} (mittels numerischer Integration) und die neuen Gewichte

Aufgabenstellungen:

Auch TDRBP ist für allgemeine zeitliche Assoziationsaufgaben endlicher Zeitfunktionen geeignet. Die Aufwand liegt in einer Größenordnung von $K*N^2$ (K = Anzahl Stützstellen in Zeitfunktionen. K ist im Normalfall wohl größer als N anzunehmen; N = Anzahl Neuronen). Im vollständigen Graphen müssen N^2 Gewichte berechnet werden. Dabei ist für die Integration über K Stützstellen ein Aufwand von K notwendig. Der Aufwand für die Schritte 1.-3. liegt in der selben Größenordnung. Bei Parallel arbeitenden Neuronen reduziert sich der Aufwand auf eine Größnordnung von K*N, wobei der Faktor N durch die Berechnung der Eingabe eines Neurons zustande kommt.

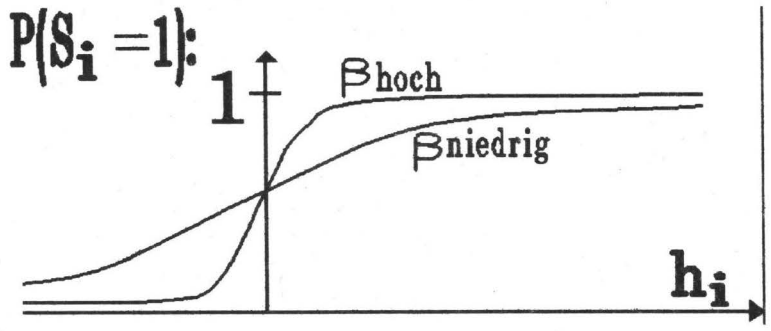


Abbildung 5.9: Verteilungsfunktion f_β

5.4 Reinforcement Learning

Bei den bisher behandelten Lernverfahren bestand die Aufgabenstellung darin, zu einer vorgegebenen Eingabe eine vorgegebene Ausgabe zu lernen. Eine übergangsform zwischen dieser Art des überwachten Lernens und des nicht- überwachten Lernens stellt das Reinforcement Learning dar. Hier wird zu einer vorgegebenen Eingabe nicht mehr die vollständige Ausgabe mitgeliefert, sondern nur noch ein Signal, das anzeigt , ob die Ausgabe des das Neuronalen Netzes richtig war (reward) oder falsch war (penalty). Dieses Signal wird auch als reinforcement Signal bezeichnet.

5.4.1 Aufgabenstellungen

a) Jeder Eingabe ist ein eindeutiger Ausgabewert zugeordnet. Ziel ist es das Netzwerk so zu optimieren, daß zu einem gegebenen Eingabewert der richtige Ausgabewert erzeugt wird.

b) Eine Erweiterung der Aufgabenstellung besteht jetzt darin, nicht nur eindeutige Beziehungen zwischen Eingabe und Ausgabe zuzulassen, sondern lediglich von bestimmten Wahrscheinlichkeiten auszugehen, mit denen eine Ausgabe in Beziehung zu einer Eingabe steht. Ziel ist es jetzt, daß zu einer Eingabe möglichst häufig der richtige Ausgabewert erzeugt wird, bzw. der Ausgabewert mit der höchsten Wahrscheinlichkeit.

5.4.2 Stochastische Neuronen

Das prinzipielle Vorgehen bei dieser Art der Aufgabenstellung besteht darin, verschiedene Eingabewerte durchzuprobieren und die Gewichte "entsprechend" zu ändern. Um einen möglichst breiten Bereich an Ausgabewerten mit möglichst wenigen Eingaben abzudecken und außerdem die Aufgabenstellung b) behandeln zu können, ist es sinnvoll in den zu trainierenden Netzen sogenannte stochastische Neuronen einzusetzen. Diese Neuronen liefern zu einer gegebenen Eingabe nicht eine feste Ausgabe, sondern nur noch eine Ausgabe mit einer bestimmten Wahrscheinlichkeit. Im folgenden sollen die Neuronen Si diskrete Ausgabewerte +/- 1 haben. Als Verteilungsfunktion wird folgende Formel vorausgesetzt:

$$P(S_i = +1) = f_\beta(+h_i) := \frac{1}{1 + \exp^{-2*\beta*h_i}}$$

$$P(S_i = -1) = f_\beta(-h_i) := \frac{1}{1 + \exp^{+2*\beta*h_i}}$$

Der Parameter β wird hierbei als Temperatur bezeichnet.

5.4.3 Trainingsmethoden

Associative Reward/Penalty (ARP)

Bei diesem Verfahren wird mit den Ausgangswerten S_i und dem Reinforcement Signal r zuerst eine Fehlerabschätzung durchgeführt. Mittels dieser Abschätzung wird das Netz dann durch Backpropagation trainiert.

Das ARP-Verfahren rechnet auf folgende Zielwerte zurück:

$$\zeta_i = \begin{cases} S_i & :r = +1 \\ -S_i & :r = -1 \end{cases}$$

Die Motivation für diese Wahl der Zielwerte ist klar: Im Fall, daß die S_i richtig war, war auch der Zielwert S_i. Im anderen Fall war irgendein S_i falsch. ARP geht jetzt einfach davon aus, daß alle S_i falsch waren, setzt die Zielwerte genau auf das Gegenteil von S_i. Da ARP nicht "wissen" kann, welches S_i tatsächlich falsch war, ist dieses Vorgehen durchaus plausibel.

Aus der Verteilungsfunktion der stochastischen Neuronen ergibt sich ein Erwartungswert für S_i und daraus die Fehlerabschätzung E_i:

$$[S_i] = tanh(\beta * h_i)$$

$$E_i = \zeta_i - [S_i]$$

Mit der üblichen Fehlerformel und Gradienten-Abstiegsverfahren ergibt sich dann als Lernregel für die Ausgangsneuronen:

$$\Delta w_{ij} = \begin{cases} \eta * [S_i - [S_i]] * V_j & :\text{r=+1} \\ \eta * [-S_i - [S_i]] * V_j & :\text{r=- 1} \end{cases}$$

Die Gewichtsänderungen für die anderen Neuronen ergeben sich durch Backpropagation.

Modellbildung

Bei diesem Verfahren verläuft das Training in 2 Phasen. In der ersten Phase wird ein Neuronales Netz (Model) so trainiert, daß es abhängig von der gegebenen Eingabe und der vom eigentlich zu trainierendem Netz erzeugten Ausgabe, das entsprechende reinforcement Signal generiert. So wird die Umgebung (Environment) nachgebildet, die ja für die Erzeugung der Eingaben und die Bewertung r der Ausgabe zuständig ist. Das Ziel der ersten Phase ist es ein Modell zu erhalten, das auf ein gegebenes Eingabe/ Ausgabe-Paar als eigene Ausgabe ein Signal R erzeugt, daß mit dem der Umgebung übereinstimmt.

Da es hier nur ein einzelnes Ausgabe Neuron R gibt und nur einen Trainingswert $\zeta = r$, ist diese Modellbildung mittels konventionellen Backpropagations möglich. Zur Erhöhung der Streubreite der Ausgabewerte des zu trainierenden Netzes, werden die Temperatur-Parameter der Stochastischen Neuronen während der Traininsphase erhöht.

In der zweiten Phase das Modellnetzwerk und das zu trainierende Netzwerk „zusammengeschaltet". Da das Ziel des Trainings richtige Ausgaben (also R=+1) sind , wird der Zielwert dieses Netzwerkes auf $\zeta = +1$ gesetzt. Jetzt werden Eingaben generiert und das Gesamtnetzwerk mittels Backpropagations trainiert. Wichtig ist hierbei, daß im Modellnetzwerk zwar Gewichtsänderungen berechnet, die Gewichte hier aber nie geändert werden. Das Modellnetzwerk dient lediglich dazu, möglichst gute Fehlerabschätzungen zu erreichen, mit denen dann das Netzwerk durch Backpropagation trainiert werden kann.

5.5 Literatur

[**HERTZ**] Hertz, John; Introduction to the Theory of Neural Computation, Santa Fe Institute - Studies in the Sciences of Complexity Series, Addison-Wesley, Redwood City, CA, USA, 1990

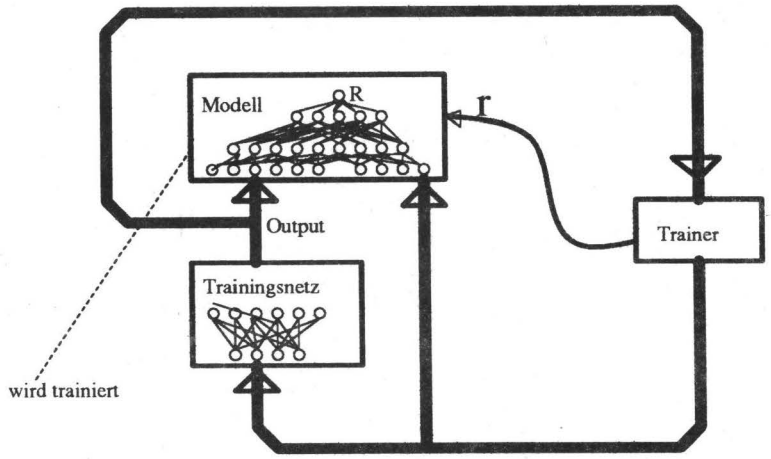


Abbildung 5.10: 1.Phase beim Verfahren der Modellbildung

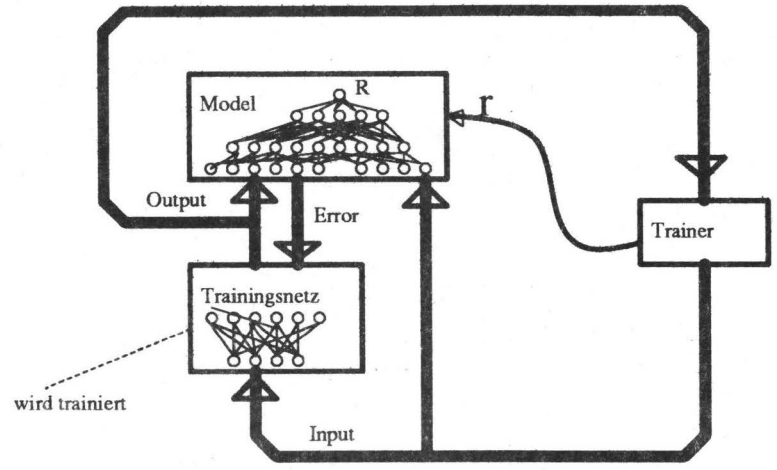


Abbildung 5.11: 2.Phase beim Verfahren der Modellbildung

[**BULSARI**] Bulsari,B. /Saxèn, Henrik; A Partially Recurrent Connectionist Model, Abo, Finland, ECAI 92, John-Wiley and Sons Ltd.,Chichester, England, 1992

Kapitel 6

ART und Competitive Learning

Frank Leidermann

Übersicht

Dieses Kapitel behandelt künstliche neuronale Netze, denen das unüberwachte Wettbewerbslernen ('unsupervised competitive learning') zugrunde liegt. Es werden Grundlagen und Beispiele für solche Netze erläutert.
Danach wird ein konkretes Netz vorgestellt: das ART1-Netz von Carpenter und Grossberg. Es handelt sich hierbei um ein Aufmerksamkeits-gesteuertes System, welches dem Stabilitäts-Plastizitäts-Dilemma Rechnung trägt.

6.1 Einleitung

Die Grundidee des Wettbewerbslernens ist, daß die (Ausgabe-)Units eines neuronalen Netzes bzw. einer Gruppe innerhalb des Netzes darum streiten, wer als einziger feuern darf. Daher nennt man diese Units auch *'winner-take-all units'* oder auch *'grandmother cells'*.
Diese Netze haben die Aufgabe, die Eingabedaten verschiedenen Kategorien zuzuordnen, also *Cluster/Häufungen* zu bilden. Erreicht werden soll dies dadurch, daß bei ähnlichen Eingabedaten dieselbe (Ausgabe-)Unit feuert. Beim unüberwachten Wettbewerbslernen werden die Kategorien oder Klassen jedoch nicht vorgegeben. Das Netzwerk muß diese Einteilung selbst anhand der Korrelation der Eingabedaten durchführen.
Anwendungsbereiche für solche Kategorisierungen sind Bildverarbeitung, statistische Analyse, kombinatorische Optimierung und Funktionsapproximierung. Konkrete Beispiele werde ich noch vorstellen.

6.2 Einfaches Wettbewerbslernen

In diesem Abschnitt werden nur *binäre* Ein-/Ausgabedaten berücksichtigt.

6.2.1 Grundlagen

Abbildung 6.1 zeigt die Architektur eines einfachen Netzes.
Eine einzige Lage von Ausgabe-Units ist vollständig mit den Eingabe-Units verknüpft, d.h. jede Ausgabe-Unit mit jeder Eingabe-Unit. Diese Verbindungen(Gewichte) sind alle anregend, d.h $w_{ij} \geq 0$, wobei w_{ij} das Gewicht zwischen Eingabe-Unit ξ_j und Ausgabe-Unit i ist.
Das Charakteristische dieser Architektur sind die *'lateral connections'* (seitliche Verbindungen) zwischen den Ausgabe-Units. Diese hemmen sich gegenseitig, regen sich selbst aber an. Wenn die Gewichte der seitlichen Verbindungen und die Aktivierungs-/Schwellfunktion richtig gewählt

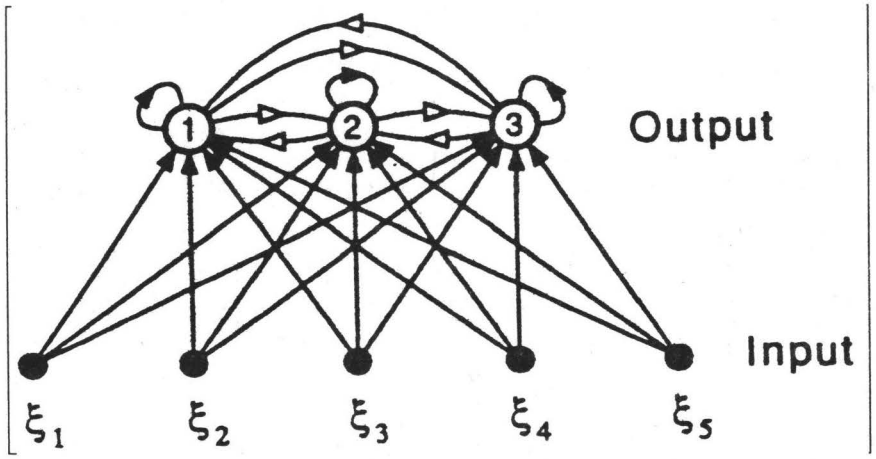


Abbildung 6.1: Einfaches Wettbewerbslernen (vgl. [2])

werden, kann immer nur eine Unit gewinnen. Im Normalfall ist das die Unit mit der maximalen Eingabe

$$h_i = \sum_j w_{ij}\xi_j = w_i \cdot \xi \tag{6.1}$$

('·' steht im gesamten Kapitel für das Skalarprodukt)
D.h. , wenn gilt:

$$\forall i : w_i^* \cdot \xi \geq w_i \cdot \xi \tag{6.2}$$

dann ist der Ausgabewert $O_i^* = 1$, d.h. i^* ist die Gewinner-Unit.
Falls die Gewichts- und Eingabevektoren normiert sind, z.B. $\forall i : |w_i| = 1$ bzw. $|\xi| = 1$, dann ist (6.2) äquivalent zu

$$\forall i : |w_i^* - \xi| \leq |w_i - \xi| \tag{6.3}$$

Dies ist so zu interpretieren, daß der Gewichtsvektor w gewinnt, der dem Eingabevektor ξ am 'nächsten' (bzgl. der herkömmlichen euklidischen Metrik) ist.
Wie erreicht man nun die Kategorisierung?
Zunächst muß man die Gewichte mit kleinen Zufallswerten z_i ($0 < z_i \ll 1$) initialisieren, wobei Symmetrien oder andere Regelmäßigkeiten strikt zu vermeiden sind. Dann wird eine Menge von Eingabedaten ξ^μ eingegeben, z.B. von einer Wahrscheinlichkeitsverteilung. Bei jeder Eingabe wird nun zuerst der Gewinner ermittelt und dann der zugehörige Gewichtsvektor w_i^* *aktualisiert (update).* Die Gewichtsvektoren der anderen Ausgabe-Units werden nicht verändert! Durch die update-Prozedur wird der Abstand des Gewichtsvektors zum aktuellen Eingabevektor $|w_i^* - \xi^\mu|$ verringert, da w_i^* quasi näher an ξ^μ geschoben wird. Konsequenz ist, daß die Wahrscheinlichkeit, daß die Ausgabe-Unit i^* erneut Gewinner wird, wenn diese Eingabe nochmals gemacht wird, steigt.
Doch wie soll man den Gewichtsvektor aktualisieren? Prinzipiell könnte man sich das so vorstellen:

$$w_{i^*j} := w_{i^*j} + \Delta w_{i^*j} \tag{6.4}$$

wobei

$$\Delta w_{i^*j} = \eta \xi_j^\mu \tag{6.5}$$

(η bezeichnet die *Lernrate).*
Dies würde allerdings dazuführen, daß die Gewichte ohne Beschränkung wachsen würden. Möglicherweise würde dann irgendwann eine Unit alle anderen Units dominieren. Das kann man durch

eine Normierung der Gewichtsvektoren verhindern, die folgendermaßen realisiert werden kann:

$$\Delta w_{i^\bullet j} = \eta' \left(\frac{\xi_j^\mu}{\sum_j \xi_j^\mu} - w_{i^\bullet j} \right) \tag{6.6}$$

Der Bruch ist hierbei die normalisierte Version der Eingabe. Falls die Eingabe bereits normalisiert ist, was man (annähernd) durch eine zusätzliche Eingabeschicht erreichen kann, genügt also:

$$\Delta w_{i^\bullet j} = \eta(\xi_j^\mu - w_{i^\bullet j}) \tag{6.7}$$

6.2.2 Geometrische Veranschaulichung

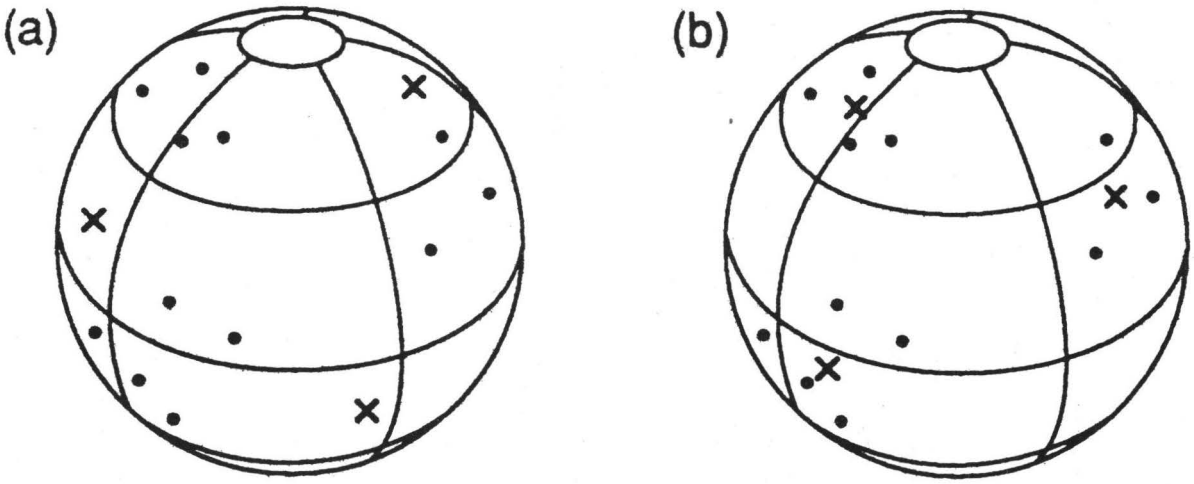


Abbildung 6.2: Wettbewerbslernen (vgl. [2])

Die Eingabedaten seien dreidimensionale Vektoren $\xi^\mu = (\xi_1^\mu, \xi_2^\mu, \xi_3^\mu)$ mit Beschränkung auf binäre Eingabedaten fällt weg. Sowohl die Eingabevektoren als auch die Gewichtsvektoren $w_i = (w_{i1}, w_{i2}, w_{i3})$ seien normiert. Abbildung 6.2.a zeigt die geometrische Veranschaulichung. Die Eingabevektoren sind durch Punkte, die Gewichtsvektoren durch Kreuze dargestellt. Wegen $|\xi^\mu| = |w_i| = 1\,(\forall i)$ liegen sie auf der Oberfläche einer Einheitskugel.
Für jeden Eingabevektor ξ^μ wird nun der Gewinner bestimmt. Nach (6.1) bzw. (6.2) ist dies der Gewichtsvektor w_i, der ξ^μ am *nächsten* liegt. Geometrisch betrachtet gewinnt das Kreuz, dessen Richtung den kleinsten Winkel zu der Richtung des jeweiligen Punktes hat (vom Mittelpunkt der Kugel gemessen). Dann wird das gewinnende Kreuz (Gewicht) in Richtung des Punktes (Eingabewert) geschoben. Die Punkte versuchen also jeweils das nächste Kreuz zu sich zu ziehen.
In Abbildung 6.2.b wird deutlich, was der Lernprozeß bewirkt hat. Jede Ausgabe-Unit hat ein Cluster von Eingabedaten entdeckt und sich ins Zentrum der 'Gravitation' dieser Punkte bewegt. Allerdings müssen die (hier: 12) verschiedenen Eingabedaten genügend oft eingegeben worden sein.

6.2.3 Dead units

Ausgabe-Units, deren Gewichte so 'weit' von den Eingabe-Vektoren 'entfernt' sind, daß sie nie gewinnen, nennt man *'dead units'*, da sie nie lernen.
Es gibt mehrere Möglichkeiten, dead units zu vermeiden:

1. Man *initialisiert* alle Gewichte durch die ersten Eingabedaten.

2. Man aktualisiert auch die Gewichte der *Verlierer*, aber nicht so stark wie die des Gewinners, d.h. mit einem viel kleineren Lernfaktor η. So werden die ständigen Verlierer langsam in Richtung der durchschnittlichen Eingabedaten geschoben. Damit steigt die Wahrscheinlichkeit, daß sie in Zukunft auch einmal gewinnen.

3. Man aktualisiert die Gewichte der dem Sieger *benachbarten* Units. Dies ist die Grundidee für die Kohonen-Netze.

4. Wie oben erläutert (vgl. (6.1)), siegt die Unit mit der maximalen Eingabe h_i.. Man subtrahiert von h_i einen Schwellwert μ_i, der mit der Zahl der Siege ebenfalls steigt. Häufige Gewinner bekommen so quasi ein 'schlechtes Gewissen' und gewinnen folglich nicht mehr so oft (*'conscience mechanism'*).

5. Statt dem Muster ξ^μ gibt man folgendes ein:

$$\tilde{\xi^\mu} := \alpha\xi^\mu + (1-\alpha)v \tag{6.8}$$

   Dabei ist v ein konstanter Vektor, mit dem alle Gewichtsvektoren initialisiert sind, und $\alpha \in [0,1]$.
   Für $\alpha = 0$ ist $\tilde{\xi^\mu} = v$, für $\alpha = 1$ ist $\tilde{\xi^\mu} = \xi^\mu$.
   Läßt man nun α langsam von 0 nach 1 wachsen, so bewegen sich die Eingabevektoren $\tilde{\xi^\mu}$ von v weg und hin zu den tatsächlichen Eingabemustern ξ^μ. Dabei 'ziehen' sie die Gewichtsvektoren mit sich mit.

6. Man fügt zu den Eingabevektoren eine Art *'Rauschen'* hinzu, so daß auch weiter entferntere Units gewinnen können.

6.3 Beispiele und Anwendungen von Wettbewerbslernen

6.3.1 Bipartionierung von Graphen

Das mathematische Problem besteht darin, einen Graphen in zwei Teile mit derselben Knotenanzahl zu teilen, wobei die beiden Teile möglichst wenige Verbindungen zueinander haben sollen.
Man geht das Problem folgendermaßen an:
Jedem Knoten ordnet man eine binäre Eingabe-Unit zu. Man gibt allerdings Kanten ein, indem man die zwei Eingabe-Units der zugehörigen Knoten gleichzeitig auf 1 setzt (*Dipol-Stimulus*). Ausgabe-Units benötigt man zwei, für jede zu suchende Hälfte eine. Das Netz soll also lernen, die Kanten des Graphen in zwei Kategorien einzuteilen. Das eigentliche Problem ist zwar, die Knoten zu kategorisieren, aber man erhält trotzdem oft eine exakte Lösung. In Abbildung 6.3 sieht man links, daß die beiden Ausgabe-Units aufgrund der vollständigen Verknüpfung keine Informationen über die geometrische Anordnung der Knoten haben.
Abbildung 6.3 zeigt rechts drei Experimente A, B und C. Die zufällige, recht gleichmäßige Initialisierung der Gewichte ist links abgebildet, der Zustand der Gewichte nach dem Training (Eingabe von 400 Kanten) rechts. Dabei bedeuten ausgefüllte Punkte, daß Unit 1 das größere Gewicht für diese Eingabe besitzt, während für die Eingabe der unausgefüllte Punkte Unit 2 das größere Gewicht hat. Analog dazu sind die Kanten entweder dick oder dünn gezeichnet, jenachdem, welche Unit das größere Gewicht für die Eingabe dieser Kanten bzw. Knotenpaare besitzt.

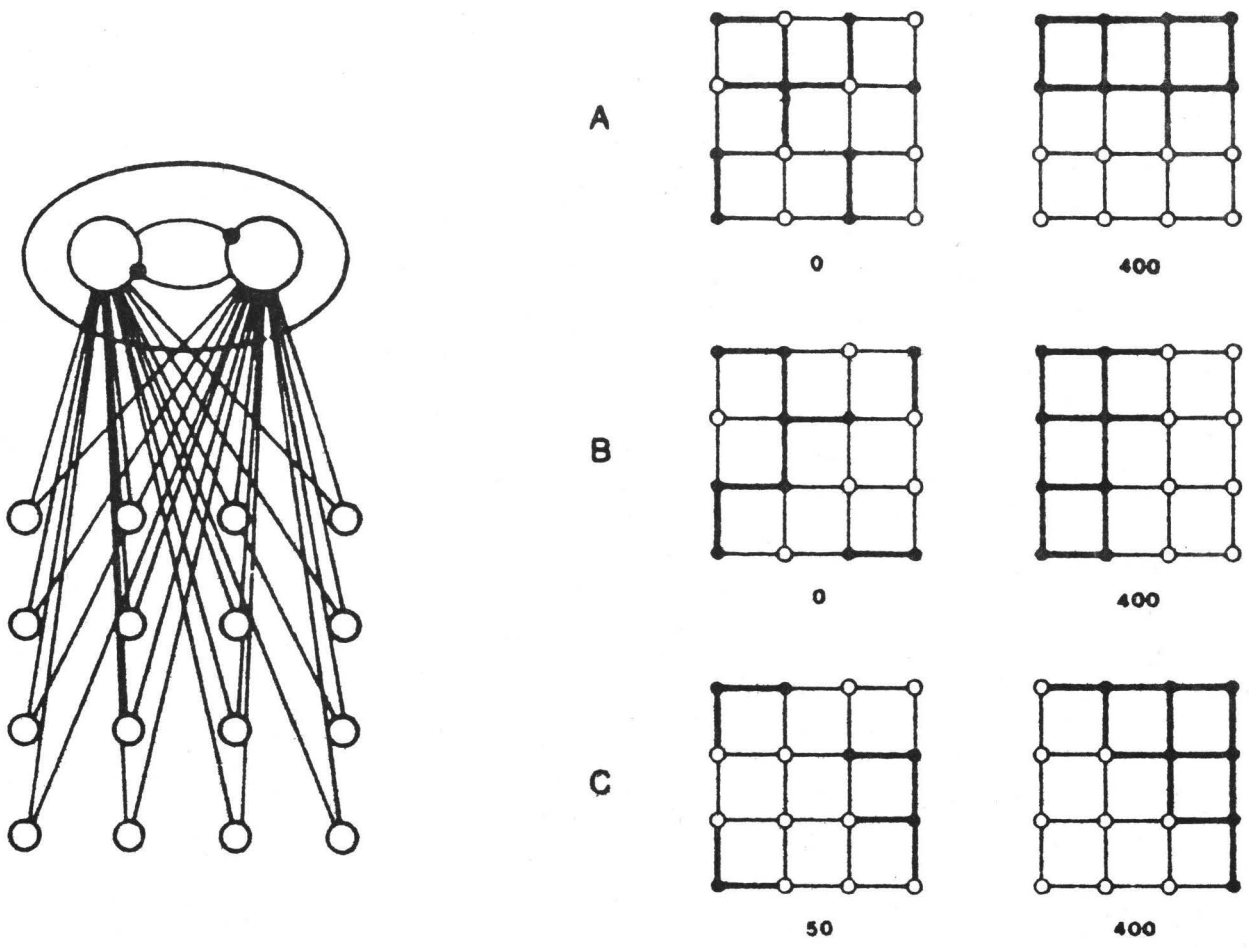


Abbildung 6.3: Das Dipol-Experiment (vgl. [1])

6.3.2 Vektorquantisierung

Vektorquantisierung ist eines der wichtigsten Anwendungsgebiete des Wettbewerbslernens. Ziel ist eine Kompression von Daten, die gespeichert oder übertragen werden sollen. Man kategorisiert eine Menge von Eingabevektoren und repräsentiert diese Vektoren nach dem Lernprozeß durch die Indizes ihrer jeweiligen Klassen. Die Klassen werden meist durch sog. *'Prototyp-Vektoren'* definiert. Man erhält somit eine Art Codebuch. Gibt man dann einen weiteren Eingabe-Vektor ein, so wird er dem Prototyp-Vektor zugeordnet, der ihm am nächsten liegt. Veranschaulicht wird die dadurch entstandene Aufteilung des Eingaberaums in ein *'Voronoir Mosaik/Dirichlet Mosaik'* in Abbildung 6.4.
Man erhält also praktisch eine diskretisierte Karte.
Als Prototyp-Vektoren dienen Gewichtsvektoren, als Gewinnregel (6.3).
Abbildung 6.5 zeigt zwei Beispiele, in denen jeweils 1000 Punkte eingegeben wurden. Das auch hier auftretende Problem der dead units, welches eine gleichmäßige Verteilung der Prototyp-Vektoren verhindern kann, ist mit dem *conscience mechanism* (vgl. 1.2.3.4) bislang am besten gelöst worden.

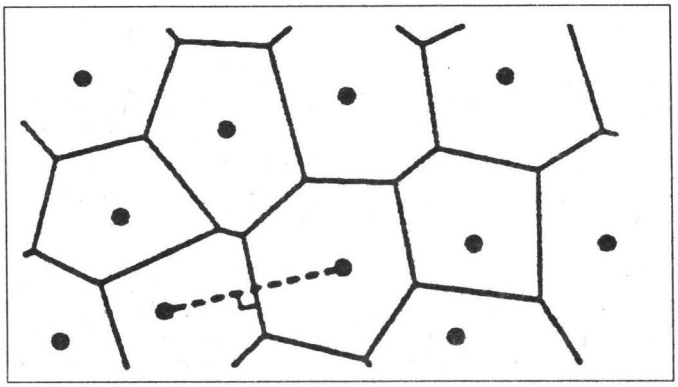


Abbildung 6.4: Voronoir Mosaik (vgl. [2])

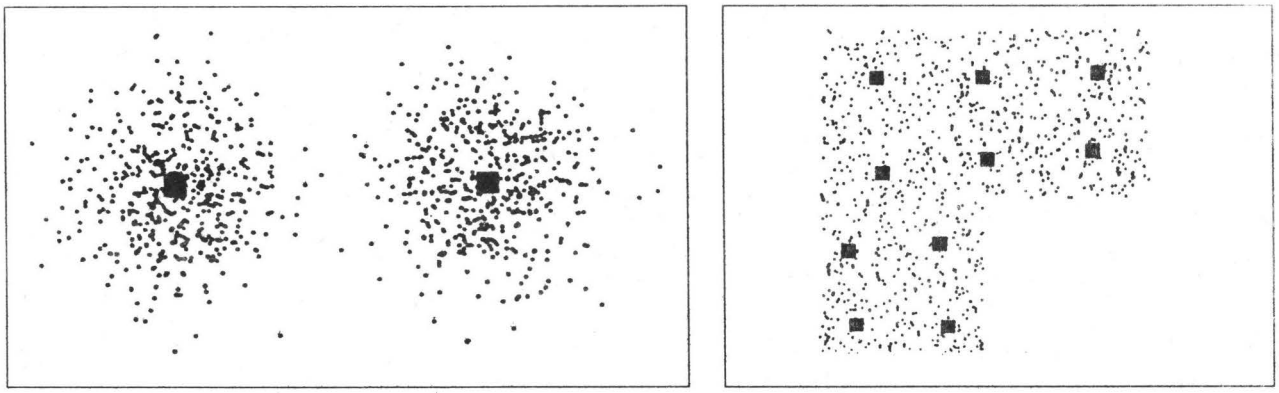


Abbildung 6.5: Zwei Beispiele für Vektorquantisierung (vgl. [2])

6.3.3 Mehrstufige Netze

Bisher haben wir jeweils nur Netze mit einer *einzigen Ausgabeschicht* betrachtet. Man kann dieses Konzept erweitern und *mehrere* Schichten/Stufen verwenden. Sinnvoll ist dies aber nur, wenn in einer zusätzlichen Schicht zwischen Ein- und Ausgabeschicht mehrere Gewinner möglich sind, ansonsten würde man lediglich einen Gewinner zur 'obersten' Ausgabeschicht 'durchreichen'.
Ein Ansatz ist die Aufteilung einer Schicht in mehrere *Gruppen*, die jeweils einen Gewinner haben. (Wenn diese Gruppen unterschiedlich strukturiert sind, sind verschiedene Gewinner möglich und wahrscheinlich. Aber selbst bei gleicher Struktur kann es z.B. aufgrund einer unterschiedlichen (zufälligen) Initialisierung verschiedene Gewinner geben.)
Ein Beispiel für den Sinn mehrstufiger Netzen ist das '*Problem der horizontalen/vertikalen Linien*'. 36 Eingabe-Units sind in einem 6x6-Feld angeordnet. Da sie mit dem restlichen Netzwerk vollständig verknüpft sind, besitzt dieses jedoch keine Information über diese geometrische Anordnung.
Aktiviert werden immer 6 Eingabe-Units, die in einer Linie liegen. Das Problem besteht darin, daß zwei Ausgabe-Units H und V genau dann feuern sollen, wenn die Linie horizontal (H) bzw. vertikal (V) ist.

Für ein einstufiges Netz ist das Problem unlösbar. Denn jeder Punkt auf jeder vertikalen Linie hat die 'gleichen Auswirkungen' auf V, d.h. V erhält von allen 36 Punkten das gleiche Gewicht. Dasselbe gilt für H.
Mit einer zusätzlichen Schicht zwischen Ein- und Ausgabeschicht kann man das Problem jedoch

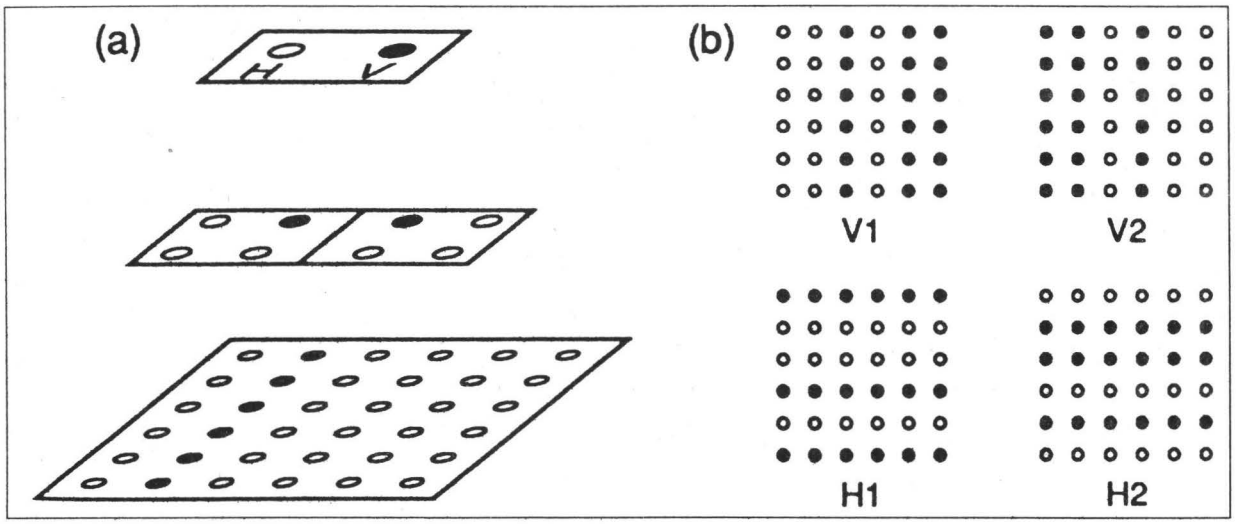


Abbildung 6.6: Das 'Problem der horizontalen/vertikalen Linien' (vgl. [2])

lösen(s. Abbildung 6.6.a). Diese Schicht besteht aus zwei Gruppen mit je vier Units H1, H2, V1, V2. Jede der Units lernt auf 3 vertikale oder 3 horizontale Linien zu reagieren (s. Abbildung 6.6.b). Allerdings darf die Art, wie die 6 Linien in zwei Hälften geteilt werden, nicht dieselbe sein. Dies ist jedoch selten der Fall und kann weitgehend vermieden werden, indem man mehr als zwei Gruppen verwendet. Die beiden Ausgabe-Units lernen die Korrelation zwischen den beiden Gruppen im Normalfall so, daß die Unterscheidung horizontal/vertikal jetzt gelingt. Während des Trainings ist aber noch eine zusätzliche Eingabe, ob es sich um eine horizontale oder eine vertikale Linie handelt, notwendig. Es handelt sich hier also gar nicht um rein unüberwachtes Lernen.

Ein alternativer Ansatz ist, die Zwischenschicht nicht in festgelegte Gruppen zu unterteilen, sondern diese Gruppen *flexibel* zu halten. Man baut hemmende seitliche Verbindungen zwischen zwei Units i und j ein, wenn $|i - j| < d$. Man geht dabei von einer zweidimensionalen Anordnung der Units und einer sinnvollen Abstandsfunktion aus. Diese hemmenden Verbindungen bewirken, daß innerhalb des Radius d nur eine Unit feuern kann, so wie dies innerhalb der oben genannten *starren* Vierergruppe der Fall ist. Dieser flexiblere, aufwendigere Ansatz ist allerdings für das Problem der horizontalen/vertikalen Linien nicht notwendig.

6.4 Adaptive Resonance Theory(ART)

6.4.1 Das Stabilitäts-Plastizitäts-Dilemma

Beim Wettbewerbslernen möchte man beide Ziele erreichen: *Stabilität* und *Plastizität* des Netzes. Die Stabilität bezieht sich auf die erlernten Kategorien. Sie sollen sich langfristig, also nach einer gewissen Lernzeit, nicht mehr verändern. Erreichen kann man das, indem man die Lernrate η langsam gegen 0 gehen läßt. Bei kleinem η kann das Netz aber nicht mehr auf neue Eingabedaten, welche in die bislang erlernten Kategorien gar nicht hineinpassen, reagieren. Die Plastizität des Netzes geht also verloren.

Die beiden Ziele Plastizität und Stabilität sind also *konträr*. Eine Möglichkeit, mit diesem Dilemma umzugehen, führt über die Frage, *wieviele Ausgabe-Units* man benützen soll.
Ist die Anzahl relativ klein und vermeidet man dead units, so führt die dadurch gewonnene langfristige Stabilität zu einem Verlust der Plastizität (wie oben dargelegt). Zu viele Ausgabe-Units zu verwenden, hat hingegen zur Folge, daß eine zu feine und damit sinnlose Kategorisierung entsteht.
Ein Lösung dieses Problems ist folgende Idee: Man besitzt zwar einen großen Vorrat an Units, benützt sie aber erst, wenn sie auch wirklich benötigt werden. Die bis zu einem bestimmten Zeitpunkt benützten Units, d.h. die bis dahin erlernten Kategorien, kann man dann stabil halten. Und für neue Eingabedaten, die in keine bekannte Klasse passen, benützt man einfach eine neue, bislang 'ruhende' Unit. Somit bleibt die Plastizität erhalten, allerdings nur bis der Vorrat an Ausgabe-Units ausgeschöpft ist. Danach wird der Stabilität des Netzes Vorrang gegeben.
Den *ART (adaptive resonance theory)-Netzen* von *Carpenter* und *Grossberg* liegt genau diese Idee zugrunde. Wenn eine Eingabe in eine bekannte Kategorie paßt, d.h. dem entsprechendem Prototyp-Vektor *genügend ähnlich* ist, wird dieser Vektor der Lernregel gemäß verändert, also dem Eingabe-Vektor angepaßt (*Resonanz*). Findet man zu einer gegebenen Eingabe keinen genügend ähnlichen Prototyp-Vektor, so bildet man eine neue Kategorie, wobei der neue Prototyp-Vektor dieser Kategorie mit dem Wert des Eingabe-Vektors initialisiert wird.
Was 'genügend ähnlich' bedeutet, bestimmt ein *'Aufmerksamkeits'-Parameter* ρ mit $\rho \in]0,1]$. Daher gehören die ART-Netze zur Klasse der *'Aufmerksamkeits-gesteuerten' Systeme.*
Für ein großes ρ wird die Ähnlichkeitsbedingung so streng, daß sehr viele feine Kategorien gebildet werden. Eine grobe Kategorisierung wird durch ein kleines ρ erreicht. Da ρ nicht fix sein muß, kann es passieren, daß bei einer Vergrößerung von ρ eine erlernte Klasse in mehrere kleinere aufgeteilt wird.
Abbildung 6.7 verdeutlicht die Funktion des Aufmerksamkeitsparameters ρ.
In diesem Experiment sollen die Buchstaben A bis J kategorisiert werden. Die Buchstaben werden in einer 5x5-Matrix kodiert. Die resultierenden 25 Pixel werden an 25 Eingabe-Units eingegeben. In der ersten Spalte stehen jeweils die Buchstaben, die der alphabetischen Reihenfolge nach eingegeben werden. Die von 1 bis 10 numerierten Spalten enthalten die bis zum jeweiligen Zeitpunkt bekannten Kategorien, repräsentiert durch Prototyp-Vektoren.
In Experiment (a) ist die Aufmerksamkeit wegen $\rho = 0.5$ relativ gering. Daher werden nur drei Klassen gefunden. In Experiment (b) ist die Ähnlichkeitsbedingung dann strenger ($\rho = 0.9$) und es werden fünf Klassen gefunden. Der Buchstaben C paßt beispielsweise in (a) noch in die Kategorie 1, die A und B enthält, während in (b) eine neue Kategorie gebildet werden muß.

Das ART1-Netzwerk werde ich in 1.4.2 – 1.4.4 ausführlich vorstellen. In 1.4.5 werden die Grundideen von ART2 und ART3 dargelegt.

6.4.2 Der ART1-Algorithmus

Um das ART1-Netz verstehen zu können, ist es hilfreich, den zugrundeliegenden Algorithmus zu kennen. Wir betrachten jeweils n-stellige *binäre* Eingabevektoren ξ und Prototyp-Vektoren w_i. Zu Beginn sind alle n Komponenten aller w_i mit 1 initialisiert. Dieser Vektor repräsentiert noch keine Kategorie, sondern markiert den jeweiligen Prototyp-Vektor als *'ungebunden'*, d.h. 'noch nicht benutzt'. Für jede Eingabe ξ wird nun folgender Algorithmus, den ich an einem Beispiel verdeutlichen werde, durchlaufen.

- Schritt 1

  Aktiviere alle Ausgabe-Units (d.h. Prototyp-Vektoren). Das bedeutet alle Prototyp-Vektoren nehmen am Wettbewerb teil.

- Schritt 2

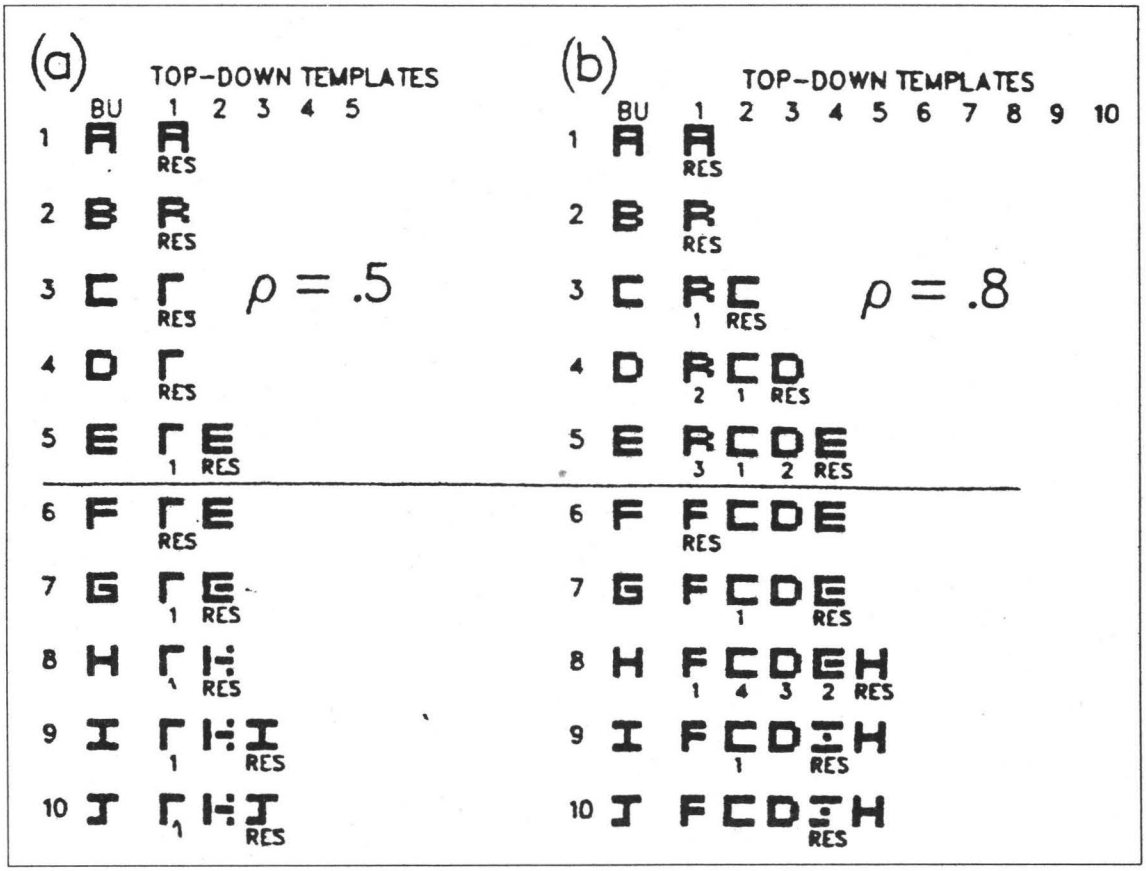


Abbildung 6.7: Buchstabenlernen mit verschiedener Aufmerksamkeit (vgl. [1])

Unter allen aktivierten Ausgabe-Units *gewinnt* die mit dem maximalen $\bar{w}_i \cdot \xi$, wobei $\bar{w}_i$ die normalisierte Form von w_i ist:

$$\bar{w}_i = \frac{w_i}{\varepsilon + \sum_j w_{ji}} \tag{6.9}$$

d.h.

$$\bar{w}_i \cdot \xi = \frac{\text{'Trefferzahl'}}{\text{'Bits von } w_i mbox'} \tag{6.10}$$

wobei mit 'Bits' hier 'Einsen' gemeint ist. ε $(0 < \varepsilon \ll 1)$ dient dazu, Unentschieden zu vermeiden.

- Schritt 3

  Zuerst berechnet man das Verhältnis

  $$r = \frac{w_i^* \cdot \xi}{\sum_j \xi_j} \tag{6.11}$$

  r entspricht im Gegensatz zu $\bar{w}_i \cdot \xi$ der

  $$\frac{\text{'Trefferzahl'}}{\text{'Bits von } \xi_i\text{'}} \tag{6.12}$$

  Jetzt wird getestet, ob der siegreiche Prototyp i^* der Eingabe ξ *genügend ähnlich* ist. Dies ist dann der Fall, wenn $r \geq \rho$ gilt, wobei ρ der Aufmerksamkeits-Parameter ($\rho \in]0, 1]$) ist.

  Wenn $r \geq \rho$ dann gehe zu Schritt 4, sonst 'blockiere', d.h. 'entaktiviere' Unit i^* und gehe zu Schritt 2.

- Schritt 4

  wi^* wird *aktualisiert (update)*:

  $$w_i^* := w_i^* \text{ AND } \xi \tag{6.13}$$

  D.h. alle Bits von w_i^*, die nicht in ξ enthalten sind, werden auf 0 gesetzt.

- Ende

6.4.3 Ein Beispiel für den Algorithmus

Ausgangsposition:
$n = 10$
Aufmerksamkeits-Parameter: $\rho = 0.9$
Vernachlässigung von ε
5 Ausgabe-Units

Eingabe:		ξ	$= 1110011101$
Gewichtsvektoren:	gebunden:	w_1	$= 0011100111$
		w_2	$= 1101111101$
	ungebunden:	w_3	$= 1111111111$
		w_4	$= 1111111111$
		w_5	$= 1111111111$

Ausführen des Algorithmus:

- (Schritt 1)

  Aktiviere alle 5 Units

- (Schritt 2)

  $\bar{w_1} \cdot \xi = \frac{w_1 \cdot \xi}{\sum_j w_{j1}} = \frac{3}{6} = 0.5$

  $\bar{w_2} \cdot \xi = \frac{w_2 \cdot \xi}{\sum_j w_{j2}} = \frac{6}{8} = 0.75$

  $\bar{w_3} \cdot \xi = \frac{w_3 \cdot \xi}{\sum_j w_{j3}} = \frac{7}{10} = 0.7$

  $\bar{w_4} \cdot \xi = \frac{w_4 \cdot \xi}{\sum_j w_{j4}} = \frac{7}{10} = 0.7$

  $\bar{w_5} \cdot \xi = \frac{w_5 \cdot \xi}{\sum_j w_{j5}} = \frac{7}{10} = 0.7$

  $\rightarrow i^* = 2$

- (Schritt 3)

  $r = \frac{w_i^* \cdot \xi}{\sum_j \xi_j} = \frac{w_2 \cdot \xi}{\sum_j \xi_j} = \frac{6}{7} \approx 0.85 < 0.9 = \rho$

  w_2 ist ξ also nicht genügend ähnlich und somit wird Ausgabe-Unit 2 blockiert und zu Schritt 3 zurückgegangen.

- (Schritt 2)

  $\bar{w_1} \cdot \xi = 0.5$

  $\bar{w_3} \cdot \xi = 0.7$

  $\bar{w_4} \cdot \xi = 0.7$

  $\bar{w_5} \cdot \xi = 0.7$

  $\rightarrow i^* = 3$ (je nach Implementierung eine der drei ungebundenen Units)

- (Schritt 3)

  $r = \frac{w_i^* \cdot \xi}{\sum_j \xi_j} = \frac{w_3 \cdot \xi}{\sum_j \xi_j} = \frac{7}{7} = 1 \geq 0.9 = \rho$

  w_3 ist ξ also genügend ähnlich. Da bei ungebundenen Gewichtsvektoren alle Bits gesetzt sind,

  ist deren 'Trefferzahl' immer identisch mit der Anzahl der Bits von ξ, und daher gilt immer $r = 1 \geq \rho$, da $\rho \in]0, 1]$.

- (Schritt 4)

  $w_3 := w_3$ AND $\xi = \xi$

  Der neue Prototyp-Vektor w_3, der bis dahin ungebunden war, ist jetzt also mit dem 'neuartigen' Eingabe-Vektor initialisiert.

- (Ende)

Der Algorithmus kann also auf drei verschiedene Arten stoppen:

1. Der Eingabe-Vektor paßt in eine bekannte Kategorie; Ausgabe: die (angepaßte) Kategorie.
2. Der Eingabe-Vektor paßt in keine bekannte Kategorie (s. Beispiel). Eine neue Kategorie wird gebildet, indem ein ungebundener Vektor auf ξ gesetzt wird: Ausgabe: die (neue) Kategorie.
3. Der Eingabe-Vektor paßt in keine bekannte Kategorie, aber es gibt keine ungebundenen Vektoren mehr. Alle Ausgabe-Units werden blockiert; Ausgabe: keine.

Der Algorithmus besitzt also *Plastizität*, solange noch ungebundene Vektoren existieren. *Stabil* ist er aufgrund der Tatsache, daß beim Update in Schritt 4 immer nur Bits auf 0 gesetzt werden, d.h. ein Gewichtsvektor wird nie größer, d.h. nach maximal n Updates, die ihn wirklich verändern, entspricht er dem Nullvektor und kann nicht mehr kleiner werden. Die relativ langsame Suche nach der genügend ähnlichen Kategorie entfällt, sobald der Algorithmus stabil ist.

6.4.4 Implementierung des ART1-Netzwerks

Wie implementiert man diesen Algorithmus als Netzwerk Ich werde eine vereinfachte Version vorstellen, bei der u.a. der Zeitfaktor verschiedener Vorgänge vernachlässigt wird. Abbildung 6.8 zeigt die Struktur dieses Netzwerks.

Die Eingabeschicht enthält Units V_j, die mit den Units O_i der Ausgabeschicht vollständig ver-

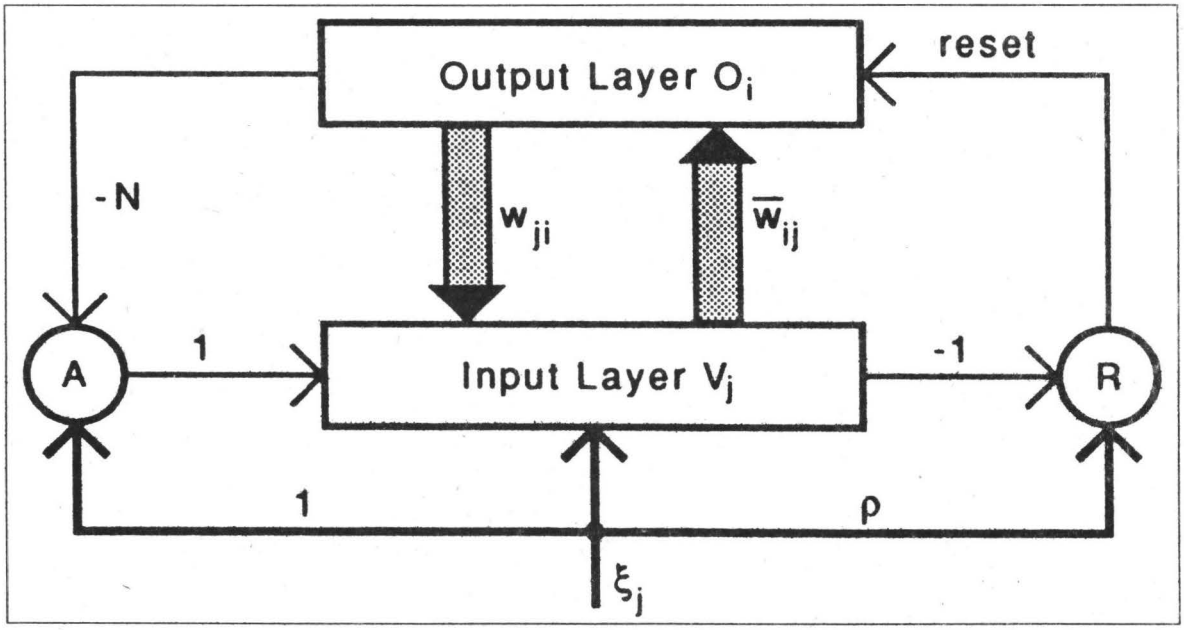


Abbildung 6.8: Das ART1-Netzwerk (vgl. [2])

knüpft sind, wobei $\bar{w_{ij}}$ wie im oben beschriebenen Algorithmus die normalisierte Ver sion von w_{ji} darstellt. Die Eingabe-Units V_j sind so konstruiert, daß

$$V_j = \begin{cases} \xi_j & : \quad O_i = 0 \qquad (\forall i) \\ \xi_j \wedge \sum_i w_{ji} O_i & : \quad \text{sonst} \end{cases} \tag{6.14}$$

Der *erste* Fall tritt ein, wenn ein neuer Vektor ξ eingegeben wird. Da zu diesem Zeitpunkt noch keine Ausgabe-Unit gewonnen hat, ist $O_i = 0$ für alle i. Die *Netzeingabe* in die Ausgabeschicht ist also

$$e_o = \sum_j \bar{w_{ij}} \xi_j = \bar{w_{ij}} \cdot \xi_j \tag{6.15}$$

Dann wird *intern* unter allen aktivierten Ausgabe-Units der Gewinner i^* mit dem maximalen e_o ermittelt und O_i^* auf 1 gesetzt.

Jetzt tritt also der *zweite* Fall ein. Da $\sum_i w_{ji} \cdot O_i = w_{ji^*}$ gilt, gilt nun $V_j = 1$ genau dann wenn $\xi_j = w_{ji^*} = 1$, d.h. genau dann wenn es sich um einen 'Treffer' handelt. Diese Information braucht man für Schritt 3 des Algorithmus, den Ähnlichkeitstest.

Zunächst möchte ich aber erklären, wie (6.14) konstruiert wird. Dazu benötigen wir eine *Hilfs-Unit (auxiliary unit)* A. A ist eine binäre Schwellen-Unit mit Schwellwert 0.5 und (vgl. Abb. 6.8) Eingabe

$$e_a = \sum_j \xi_j - n \sum_i O_i \tag{6.16}$$

Wenn ξ nicht den Nullvektor darstellt (dann läge aber gar keine Eingabe vor, da *Muster* eingegeben werden), ist $e_a \geq 1$, wenn $\forall i : O_i = 0$. Sobald jedoch eine Ausgabe-Unit auf 1 gesetzt wird, ist $e_a \leq 0$. Somit ist in (6.14) im ersten Fall $A = 1$, im zweiten Fall $A = 0$.
Wie aus Abbildung 6.8 ersichtlich, ist die *Gesamteingabe in die Eingabe-Units* V_j folgende: $h_j = \xi_j + \sum_i wjiO_i + A$, wobei V_j auf 1 gesetzt wird (feuert), wenn ein Schwellwert von 1.5 überschritten wird. Also müssen zwei der drei Summanden auf 1 gesetzt sein. Es handelt sich hier um eine '2/3-Regel'.
Gilt $A = 1$, so muß der zweite Summand 0 sein (nach Konstruktion von A). Somit ist der erste Fall von (6.14) realisiert, denn wenn $\xi_j = 1$, wird $V_j = 1 (= \xi_j)$, wenn $\xi_j = 0$ gilt, wird $V_j = 0 (= \xi_j)$.

Gilt $A = 0$, so gilt $V_j = 1$, wenn der erste und der zweite Summand beide auf 1 gesetzt sind. Der zweite Fall von (6.14) ist also auch realisiert.
Kommen wir nun zum *Ähnlichkeitstest*, dem Schritt 3 des Algorithmus. Im Fall $\rho > r$ soll die binäre Unit *R (Reset)* feuern und damit O_i^* auf 0 setzen, also den Gewinner blockieren. Realisiert wird dies dadurch, daß (vgl. Abb. 6.8) R eine Gesamteingabe $e_r = \rho * \sum_j \xi_j - \sum_j V_j$ erhält, wobei der Schwellwert 0 ist.
Denn:
Behauptung: $e_r \begin{cases} > 0 & : \quad \rho > r \\ \leq 0 & : \quad \rho \leq 0 \end{cases}$
Beweis:

$$r = \frac{w_i^* \cdot \xi}{\sum_j \xi_j} \longrightarrow r * \sum_j \xi_j = w_i^* \cdot \xi = \quad \text{'Anzahl der Treffer' von } i^* \tag{6.17}$$

Während des Ähnlichkeitstests ist $O_i^* = 1$. Daraus folgt

$$\sum_j V_j = \sum_j \left(\xi_j \wedge \sum_i w_{ji} O_i \right) = \sum_j (\xi_j \wedge w_{ji^*}) = \quad \text{'Anzahl der Treffer ' von } i^* \tag{6.18}$$

Aus 6.17 und 6.18 folgt

$$\left(r \sum_j \xi_j - \sum_j V_j \right) = 0 \tag{6.19}$$

Daraus folgt

$$\left(\rho \sum_j \xi_j - \sum_j V_j \right) \begin{cases} > 0 & : \quad \rho > r \\ \leq 0 & : \quad \rho \leq 0 \end{cases} \tag{6.20}$$

Dies war zu zeigen.
Das Netzwerk ist also völlig *autonom*, benötigt keinerlei Steuersignale und ist bzgl. der Architektur *parallel*.

6.4.5 ART2 und ART3

Die beiden Netzwerke ART2 und ART3 sind Erweiterungen von ART1.
Bei ART2 fällt die Einschränkung auf binäre Eingabemuster weg. Auch *analoge* Muster können klassifiziert werden. Dabei wurde eine gewisse *Rauschunempfindlichkeit* erreicht.
Während ART1 und ART2 *einstufige* Architekturen besitzen, wird mit ART3 der Ansatz verfolgt,

mehrstufige Systeme zu erstellen, wobei die einzelnen Schichten sich wechselseitig beeinflussen sollen. Außerdem hat nicht jede Schicht F_i eine eigene Reset-Unit R_i, sondern es gibt eine einzige Unit R, die die Aktivität der Ausgabe-Units in allen Schichten kontrolliert.

6.5 Zusammenfassung

Die Idee der 'winner-take-all units' hat drei prinzipielle *Nachteile*:

1. Man benötigt für n verschiedene Kategorien n verschiedene Units. Würde man hingegen die binäre Repräsentation wählen, könnte man 2^n verschiedene Kategorien darstellen.
2. Sobald eine Unit versagt, geht die zugehörige Kategorie verloren. Dieses Problem tritt allerdings nicht nur bei solchen Netzen auf.
3. Wenn man daran festhält, daß immer nur eine Unit feuern darf, ist es nicht möglich, Unterklassen zu bilden, also eine Kategorie innerhalb einer Kategorie. Eine mögliche Hierarchie der Klassen ist also nicht darstellbar.

Während die ersten beiden Kritikpunkte *Komplexität* und *Robustheit* betreffen, zielt Punkt 3 auf grundsätzliche Einschränkungen des Wettbewerbslernens ab. Wettbewerbslernen kann man nur dann anwenden, wenn eine *eindeutige* Entscheidung für eine Lösung gefragt ist.
Zwei potentielle Anwendungen, bei denen dies der Fall ist, sind zum einen *Klassifizierung*, zum anderen das *'Schedulingproblem'* (auf welchem Prozessor soll ein bestimmtes Programmstück ablaufen?). In der Praxis werden solche Probleme jedoch in der Regel mit 'allgemein rückgekoppelten *Hopfield-Netzen*' gelöst.
Das Modell des Wettbewerbslernens hat aber dennoch eine Bedeutung als *Studienobjekt*, um das Konkurrenzprinzip, also die wechselseitige Hemmung der Ausgabe-Units, mit der das winner-take-all-Prinzip realisiert wird, isoliert zu untersuchen.
Zum Abschluß möchte ich noch eine *Abgrenzung* zum folgenden Kapitel machen, welches *Kohonen-Netze* behandelt.
In Abbildung 6.9 werden die verschiedenen seitlichen Verbindungen zwischen den Ausgabe-Units

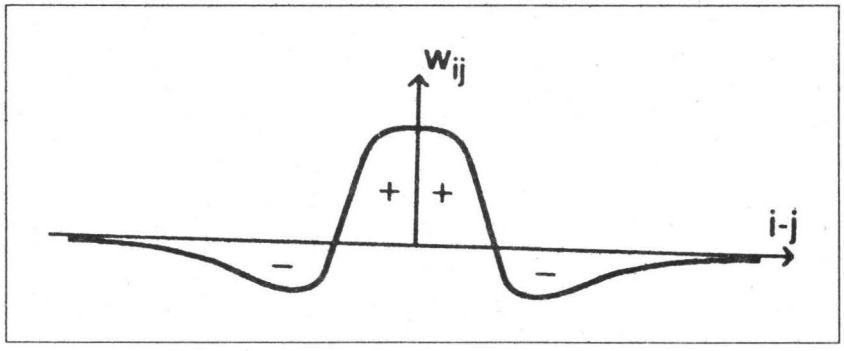


Abbildung 6.9: Die 'Mexican hat'-Form der seitlichen Verbindungen (vgl. [2])

dargestellt. Units, die nahe zusammenliegen, regen sich gegenseitig an, während sich Units, die weit voneinander entfernt liegen, sich gegenseitig hemmen. Diese wechselseitige Hemmung gibt es bei Kohonen-Netzen nicht, die daher kein reines Wettbewerbslernen durchführen.

6.6 Literatur

[1] Rüdiger Brause. *Neuronale Netze: eine Einführung in die Neuroinformatik* Stuttgart 1991.

[2] John A. Hertz. *Introduction to the theory of neural computation* Santa Fe 1991.

Kapitel 7

Kohonen Netzwerke und Semantische Karten

Martin Kronenburg

Übersicht

Das Modell von Kohonen ist ein allgemeiner und abstrakter Ansatz zur Darstellung selbstorganisierender Karten. Dies sind neuronale Netzwerke, bei denen die Aufgabe eines Neurons eng an dessen Position gekoppelt ist. In diesem Kapitel wird der Algorithmus von Kohonen vorgestellt und dessen Leistungsfähigkeit an einigen sehr unterschiedlichen Beispielen demonstriert. Der hohe Abstraktionsgrad des Modells erlaubt schließlich eine mathematische Interpretation seiner Eigenschaften.

7.1 Motivation und grundlegende Ideen selbstorganisierender Karten

Bei vielen typischen Modellen für neuronale Netze, wie zum Beispiel dem Perzeptron- und dem Hopfield-Modell sowie dem Backpropagation-Algorithmus, werden vor allem die Verbindungen zwischen den einzelnen Neuronen und die zur Außenwelt, d.h. die Ein- und Ausgabeleitungen betrachtet und analysiert. Obwohl es beim Backpropagation-Algorithmus bereits eine Aufteilung des Neuronennetzes in mehrere Schichten gibt, existiert auch dort wie bei den anderen erwähnten Ansätzen keine Strukturierung der Neuronen innerhalb einer Schicht. Die Position eines Neurons in solchen neuronalen Netzwerken hat also keinen Einfluß auf dessen Funktion bzgl. der empfangenen und ausgesandten Nachrichten.

Diese Unabhängigkeit der Lage eines Neurons in einem Netzwerk von seiner Aufgabe in diesem System ist bei selbstorganisierenden Karten nicht mehr gegeben. Vielmehr ist gerade der logische Zusammenhang zwischen Position und Funktion eines Neurons ein Kennzeichen, das selbstorganisierende Karten charakterisiert. Ein wesentliches Ziel einer solchen nicht wahllosen, sondern einer bestimmten Strukturierung unterworfenen Anordnung der Neuronen besteht darin, die Kommunikationswege zwischen Neuronen, die häufig miteinander in Verbindung treten, zu verkürzen. Die Reduktion des Kommunikationsaufwandes ist eine wichtige Eigenschaft, die man von parallelen Systemen fordert.

Ein intensiver Daten- und Informationsaustausch findet hauptsächlich zwischen Neuronen statt, die ähnlich gelagerte Aufagben im System erfüllen. Dies hat zur Folge, daß bei selbstorganisierenden Karten Neuronen, die auf ähnliche Signale ansprechen, auch räumlich eng benachbart sind. Konsequenz einer derartigen Anordnung ist eine topographische Karte der Eingangssignale. Diese innere Strukturierung eines Netzwerkes kann schließlich auch als Ergebnis eines Abstraktionsprozesses interpretiert werden, bei dem unbedeutende Einzelheiten weggelassen und wichtige

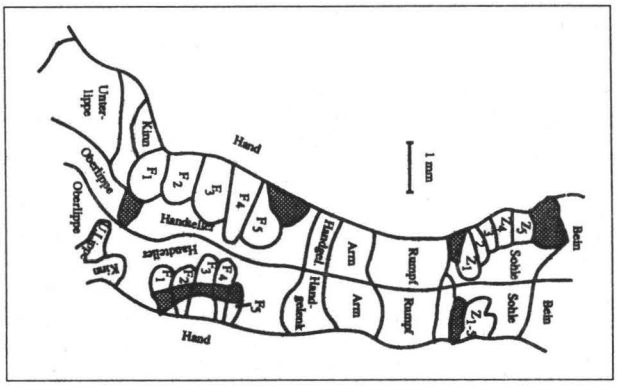


Abbildung 7.1: Karte eines Teils der Körperoberfläche im somatosensorischen Rindenfeld eines Affens

bzw. übergeordnete Gemeinsamkeiten betont werden. Dies geschieht, indem Repräsentanten gemeinsamer Merkmale lokal benachbart im Netzwerk angeordnet werden.

Wie Abbildung 7.1 zeigt, existieren die oben beschriebenen selbstorganisierenden Karten auch im Bereich der natürlichen neuronalen Netzwerke. Man erkennt, daß die enge Nachbarschaft bestimmter Körperteile auch durch die Karte widergegeben wird. Abbildung 7.1 macht aber auch noch ein weiteres wesentliches Merkmal selbstorganisierender Karten sichtbar. Bei der Projektion der Eingabesignale auf das Neuronenfeld findet in der Regel keine maßstabsgerechte Übertragung der Rezeptoren (in diesem Fall der Körperoberfläche) statt, sondern Regionen, die intensivere und zahlreichere Signale aussenden, werden dementsprechend ausgeprägter abgebildet, z.B. die Finger F1 bis F5 oder die Unterlippe im Vergleich zum Rumpf.

In Paragraph 2 wird zunächst das Modell von Kohonen vorgestellt. Es handelt sich hierbei um einen abstrakten und allgemeinen Ansatz zur Modellierung selbstorganisierender Karten.

In den Paragraphen 3 und 4 wird anhand zweier Beispiele, die aus sehr verschiedenen Bereichen der Daten- und Informationsverarbeitung gewählt sind, die Leistungsfähigkeit des Kohonen-Modells unter Beweis gestellt.

Paragraph 5 soll auf einen weiteren Vorteil des Modells hinweisen. Der dem Modell zugrundeliegende hohe Abstraktionsgrad erlaubt nämlich, dieses nicht nur durch Computersimulationen zu untersuchen sondern auch mit mathematischen Mitteln zu analysieren und zu interpretieren.

7.2 Das Modell von Kohonen

Das Modell von Kohonen, das in diesem Paragraph vorgestellt wird, ist ein allgemeiner Ansatz für selbstorganisierende Karten. Im Mittelpunkt werden dabei Überlegungen stehen, die es erlauben, daß sich einzelne Neuronen durch konkurrierendes, unüberwachtes Lernen auf bestimmte Eingabesignale spezialisieren. Diese Spezialisierung soll jedoch keine Isolation des Neurons gegenüber seinen unmittelbaren Nachbarneuronen nach sich ziehen; stattdessen wird eine auf einen Eingangsreiz positiv ansprechende Zelle auch ihre räumliche Nachbarschaft aktivieren, um diese so für ähnliche Reize zu sensibilisieren. Ergebnis eines solchen gemeinsamen, auf gegenseitiger Abhängigkeit beruhenden Lernprozesses des gesamten neuronalen Netzwerks soll sein, daß sich verschiedene Zellen in geordneter Art und Weise auf verschiedene Reize einstellen und so eine semantische Karte der Eingangssignale widergeben.

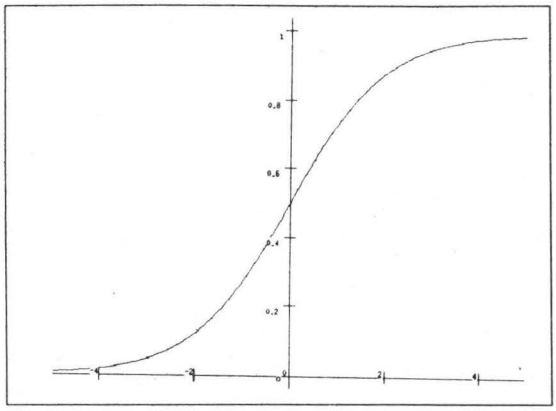


Abbildung 7.2: Graph der Fermitfunktion

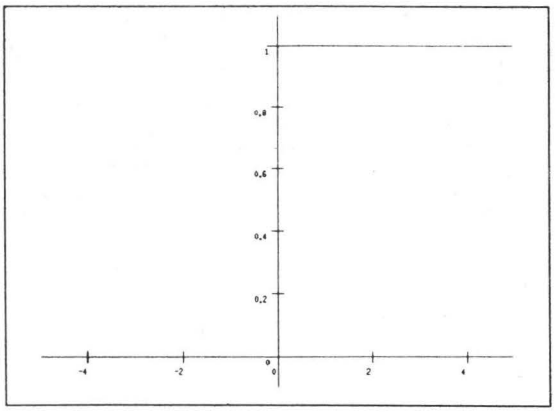


Abbildung 7.3: Graph der Stufenfunktion

7.2.1 Der Algorithmus von Kohonen

Es werden zunächst die wichtigsten Größen eingeführt, die das Modell beschreiben. In der Regel wird eine zweidimensionale **Neuronenschicht A**, ein diskretes periodisches Gitter von Neuronen, zugrundegelegt. Ein **Neuron** $r \in A$ wird dabei durch seinen zweidimensionalen **Ortsvektor r** beschrieben. Ein **einlaufendes Signal v** besteht aus **d Eingangsfasern**, die den Axonen der Rezeptoren, z.B. der Haut, entsprechen und durch die Neuronenschicht A verlaufen. Charakterisiert wird ein Signal v durch die **mittleren Aktivitäten** v_l, $l = 1, ..., d$, der Eingangsfasern. Es wird nun schrittweise beschrieben, wie sich aus diesen Daten und bestimmten Rückkopplungsmechanismen schließlich die Neuronenaktivität f_r eines Neurons $r \in A$ ergibt.

Aus den d Eingabegrößen v_l bildet jedes Neuron $r \in A$ eine gewichtete Summe $S_r = \sum_{l=1}^{d} w_{rl} v_l$, wobei w_{rl} die Stärke der Synapse zwischen Axon l und Neuron r beschreibt. Ist $w_{rl} > 0$, so hat die Synapse anregenden Einfluß auf das Neuron r, für $w_{rl} < 0$ wirkt sie hingegen hemmend.

Für den nächsten Schritt benötigt man den Begriff der **sigmoiden** Funktion. Darunter versteht man eine einstellige, auf $\mathbb{R}$ totale Funktion σ mit folgenden Eigenschaften :

1. σ ist monoton steigend
2. $\sigma(x) \geq 0$ für alle $x \in \mathbb{R}$
3. $\lim_{x \to \infty} \sigma(x) = 1$ und $\lim_{x \to -\infty} \sigma(x) = 0$

Beispiele für sigmoide Funktionen, die im weiteren auch verwendet werden, sind :

- Fermitfunktion :
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
- Stufenfunktion :
$$\psi(x) = \begin{cases} 1 & : \quad x \geq 0 \\ 0 & : \quad x < 0 \end{cases}$$

Aus der gewichteten Summe S_r eines Neurons r erhält man nun mit einer sigmoiden Funktion σ die sogenannte **mittlere Spikefrequenz** f_r^0

$$f_r^0(v) = \sigma(\sum_{l=1}^{d} w_{rl} v_l - \Theta) \tag{7.1}$$

θ übernimmt hierbei - ähnlich wie in anderen Modellen - die Aufgabe einer Erregungsschwelle, unterhalb der das Neuron nur schwach oder gar nicht reagiert. Die mittlere Spikefrequenz f_r^0 eines

Neurons r beschreibt also dessen Erregungszustand in alleiniger Abhängigkeit von den Eingangssignalen v_l.

Im letzten Schritt werden nun noch zusätzlich Verbindungen der Neuronen untereinander ins Auge gefaßt, welche letztendlich von großer Bedeutung für den selbstorganisierten Aufbau semantischer Karten in der Neuronenschicht A sind. Allgemein betrachtet ist also die zum Schluß resultierende Neuronenaktivität f_r eines Neurons r ebenfalls abhängig von den Neuronenaktivitäten $f_{r'}$ aller Neuronen $r' \in A$, insbesondere auch von seiner eigenen. Dadurch ergibt sich eine Rückkopplung der Schicht auf sich selbst.

Wie die Eingangssignale kann auch ein Wert $f_{r'}$ auf ein anderes Neuron r sowohl erregend als auch hemmend wirken. Modelliert wird dies wieder mit Hilfe eines Gewichtsfaktors $g_{rr'}$, der den Einfluß des Neurons r' auf das Neuron r beschreibt. Die Beiträge $g_{rr'}f_{r'}$ aller Schichtneuronen r' addieren sich zu der Summen S_r der äußeren Eingangssignale v_l hinzu. Damit sind im stationären Fall, d.h. würde man eine Momentaufnahme des neuronalen Netzwerkes mit allen Größen betrachten, die Neuronenaktivitäten f_r die Lösungen des nichtlinearen Gleichungssystems

$$f_r = \sigma(\sum_{l=1}^{d} w_{rl}v_l + \sum_{r' \in A} g_{rr'}f_{r'} - \Theta) \tag{7.2}$$

Oft sind die Faktoren $g_{rr'}$ so gegeben, daß die Aktivität $f_{r'}$ auf kurze Distanzen anregend, d.h. $g_{rr'} > 0$, und auf lange Distanzen hemmend, d.h. $g_{rr'} < 0$, wirken. Ist die Rückkopplung in der Neuronenschicht A von diesem Typ, so nennt man dies **Umfeldhemmung** oder **laterale Inhibition**

Man kann nun zeigen, daß eine genügend starke Umfeldhemmung durch die $g_{rr'}$ zur Folge hat, daß die Lösungen von (2) wie folgt aussehen : Neuronen, die in der Nähe des Neurons mit maximaler Erregung liegen, befinden sich ebenfalls in einem angeregten Zustand. Verwendet man eine sigmoide Funktion, wie zum Beispiel die Fermit-Funktion, so nimmt der Grad der Erregung mit zunehmender Entfernung vom Maximum ab. Es bilden sich also räumlich lokalisierte Erregungsantworten.

Diese Tatsache wird hier nicht allgemein bewiesen, sondern anhand einer vereinfachten Version von (2) vorgeführt. Die Einschränkungen lauten :

- Als sigmoide Funktion wird die Stufenfunktion ψ gewählt.
- Die Neuronenschicht A ist nur noch eindimensional, insbesondere ist daher der Abstand zweier Neuronen r, r' gegeben durch den Betrag ihrer Differenz $\mid r - r' \mid$.
- Es liegen keine Eingangssignale vor, d.h. $v_l = 0$ für alle $l = 1, ..., d$.
- Die Erregungsschwelle Θ wird auf 0 gesetzt.
- Die Gewichtsfaktoren $g_{rr'}$ sind gegeben durch

$$g_{rr'} = \begin{cases} 1 & : \ |r - r'| \leq a \\ -g & : \ |r - r'| > a \end{cases}$$

  Dabei ist a eine beliebige positive Zahl, die die durch Neuron r' zu erregende Umgebung abgrenzt, und $g > 2a + 1$ wird vorausgestzt. Dadurch wird die laterale Inhibition geeignet nachgebildet.

Mit den oben angegebenen Einschränkungen erhält man dann aus (2) :

$$\begin{aligned} f_r &= \psi(\sum_{r' \in A} g_{rr'}f_{r'}) \\ &= \psi(\sum_{r'=r-a}^{r+a} 1 f_{r'} + \sum_{r'<r-a, r'>r+a} -g f_{r'}) \end{aligned}$$

$$= \psi(\sum_{r'=r-a}^{r+a} f_{r'} - g \sum_{r'<r-a, r'>r+a} -f_{r'} - g \sum_{r'=r-a}^{r+a} f_{r'} + g \sum_{r'=r-a}^{r+a} f_{r'})$$

also

$$\psi((1+g) \sum_{r'=r-a}^{r+a} f_{r'} - g \sum_{r' \in A} f_{r'}) \tag{7.3}$$

Führt man die Größen

$$M := \sum_{r' \in A} f_{r'} \tag{7.4}$$

und

$$m_r = \sum_{r'=r-a}^{r+a} f_{r'} \tag{7.5}$$

ein und beachtet man, daß bei der Stufenfunktion ψ für $g > 0$ gilt $\psi(x) = \psi(gx)$, so erhält man durch Ausklammern von g in (3) schließlich

$$f_r = \psi((g^{-1}+1)m_r - M) \tag{7.6}$$

Die Gleichungen (4), (5), (6) bilden ein Gleichungssystem für die Neuronenaktivitäten f_r.

Gezeigt wird nun, daß die laterale Inhibition bewirkt, daß dieses Gleichungssystem nur Lösungen besitzt, die wie folgt aussehen: Die Neuronen, die sich im angeregten Zustand $f_r = 1$ befinden, bilden einen zusammenhängenden **"Cluster"**, der aus $a+1$ aufeinanderfolgeden Neuronen besteht. Alle Neuronen, die nicht in diesem Cluster liegen, befinden sich im Ruhezustand $f_r = 0$.

Der Beweis erfolgt in zwei Schritten. Zuerst wird gezeigt, daß es maximal $a+1$ aktive Neuronen in einer Lösung von (6) geben kann. Anschließend wird bewiesen, daß im Falle einer nichttrivialen Lösung von (6), d.h. wenigstens ein $f_r = 1$, mindestens $a+1$ zusammenhängende Neuronen erregt sein müssen.

Bei beiden Beweisen ist zu beachten, daß wegen der Stufenfunktion nur die Zustände $f_r = 0$ oder $f_r = 1$ möglich sind, insbesondere gilt $0 \leq m_r \leq M$ für alle $r \in A$.

Satz 1 :
Sind die f_r eine Lösung von (6) und gilt $g > 2a+1$, so folgt aus $f_r = 1$ stets $f_s = 0$ für alle $s > r+a$ und alle $s < r-a$; insbesondere also $M \leq a+1$.

Beweis :
Es gilt folgende Ungleichskette :

$$m_r \underset{(1)}{\leq} M \underset{(2)}{\leq} m_r + \frac{m_r}{g} \underset{(3)}{\leq} m_r + \frac{2a+1}{g} \underset{(4)}{<} m_r + 1$$

1. wegen der oben angegebenen Bemerkung
2. wegen $f_r = 1$ folgt aus (6) : $(1+g^{-1})m_r - M \geq 0 \Longrightarrow M \leq m_r + \frac{m_r}{g}$
3. wegen $m_r = \sum_{s-a}^{s+a} f_r \leq 2a+1$
4. wegen der lateralen Inhibitionsbedingung $g > 2a+1$

Da m_r und M natürliche Zahlen sind, folgt aus $m_r \leq M < m_r + 1$ schließlich $m_r = M$. Also muß für alle $s \notin [r-a, r+a]$ gelten $f_s = 0$, denn sonst wäre $M > m_r$.

q.e.d.

Satz 2 :
Die f_r bilden eine Lösung von (6) und es gelte $g > 2a + 1$. Weiter gebe es wenigstens ein r' mit $f_{r'} = 1$. Ist nun s das am weitesten linksbefindliche erregte Neuron r, d.h. $s \leq r$ für alle erregten Neuronen, so gilt für jedes der rechts angrenzenden a Neuronen $r \in [s, s + a]$: $f_r = 1$.

Beweis :
Wegen $f_s = 1$ gilt

$$(1 + g^{-1})m_s - M \geq 0$$

Wegen Satz 1, $M \leq a+1$, gibt es nur endlich viele aktiven Neuronen, also gibt es auch ein kleinstes, d.h. ein am weitesten links liegendes. Für $r \in [s, s + a]$ gilt nun :

$$(1 + g^{-1})m_r - M = (1 + g^{-1}) \sum_{r'=r-a}^{r+a} f_{r'} - M$$

Wegen der Wahl von s gilt für alle $r' < s$: $f_{r'} = 0$. Also kann man die unteren Summationsgrenzen bereits bei $r' = s - a$ starten lassen. Nach Satz 1 gilt für alle r' mit $r' > s + a$: $f_{r'} = 0$. Also kann man die obere Summationsgrenze bereits bei $s + a$ abschneiden. Man erhält :

$$\begin{aligned}
(1 + g^{-1}) \sum_{r'=r-a}^{r+a} f_{r'} - M &= \\
(1 + g^{-1}) \sum_{r'=s-a}^{s+a} f_{r'} - M &= \\
(1 + g^{-1})m_s - M \geq 0 &
\end{aligned}$$

$\Longrightarrow f_r = \psi((1 + g^{-1})m_s - M) = 1$ für alle $r \in [s, s + a]$

q.e.d.

Diese beiden Sätze erlauben also eine genaue Beschreibung der Lösungen von (6). Sind nicht alle $f_r = 0$, d.h. liegt nicht die triviale Lösung vor, so bilden die erregten Neuronen einen Cluster von $a + 1$ zusammenhängenden Neuronen.

Wie bereits erwähnt, gilt dieses Ergebnis auch analog im allgemeinen Fall, und bei stetigen Funktionen σ ergibt sich für den Erregungsverlauf eine Art spitze Glockenform. Die Position r_{max} der Spitze der Glocke, d.h. der Maximalwert der Erregung hängt dabei auch von den bisher außer acht gelassenen Eingangsimpulsen v_l ab. Diesen Spitzenwert könnte man zwar durch numerisches Lösen des Gleichungssystems (2) ermitteln, dies wäre jedoch recht aufwendig. Daher macht Kohonen in seinem Modelll den Vorschlag, r_{max} nur aufgrund der äußeren Reize v_l zu bestimmen, d.h. für r_{max} muß gelten

$$\sum_{l=1}^{d} w_{r_{max}l} v_l = \max_{r' \in A} \sum_{l=1}^{d} w_{r'l} v_l \tag{7.7}$$

Einen mathematisch noch einfacheren Ausdruck zur Festlegung von r_{max} erhält man, wenn man voraussetzt, daß die Gesamtsynapsenstärke pro Neuron, $\sqrt{\sum_l w_{rl}^2}$, konstant und für alle Neuronen gleich groß ist, und alle Eingangssignale v gleiche Intensität $\| v \| = 1$ haben. Man erhält dann die zu (7) äquivalente Bedingung für r_{max} :

$$\| w_{r_{max}} - v \| = \min_{r' \in A} \| w_{r'} - v \| \tag{7.8}$$

$\| x \|$ bedeutet in diesem Zusammenhang die euklidische Vektornorm von x, und $w_r = (w_{r1}, ..., w_{rd})^T$ ist der Vektor der Synapsenstärken.

Man schenkt diesem Erregungsmaximum besondere Beachtung, weil es gestattet, jedem Eingangssignal $v = (v_1, ..., v_d)$ einen "sinnvollen" Ort r in A zuzuordnen, nämlich das zu v gehörende Neuron r_{max} gemäß (8). Damit gelingt also eine Projektion des Raumes V der Eingangssignale auf die Neuronenschicht A. Diese Projektion wird im folgenden mit

$$\Phi_w : V \rightarrow A \qquad \text{und} \qquad \Phi_w(v) = r_{max} \tag{7.9}$$

bezeichnet. Der Index w zeigt an, daß $\Phi_w(v)$ stets von den Synapsenstärken w_{rl} abhängt und man für unterschiedliche Werte w_{rl} auch verschiedene Projektionen Φ_w erhält.

Das Problem, welche Synapsenstärken am besten geeignet sind, um "brauchbare" Karten des Signalraumes V in A zu erzeugen, wird im folgenden diskutiert. Man erkennt schnell, daß die Vorgabe fester Werte für w_{rl} nicht zweckmäßig ist; denn wäre dies der Fall, so müßte man a priori die genaue Beschaffenheit der Eingabesignale kennen und wäre auch später nicht in der Lage, die Karte in A möglichen Veränderungen des Signalraums V anzupassen. Die erste Bedingung ist im allgemeinen nicht gegeben und die Adaptionsfähigkeit ist eine wesentliche Eigenschaft, die biologische Nervensysteme auszeichnet, welche ja stets als Vorbild dienen.

Man kommt also schnell zu der Folgerung, daß die Neuronen die geeigneten Synapsenstärken w_{rl} selbst lernen müssen. Dieser unüberwachte Lernprozeß hat als einzige äußere Informationsquelle eine Reihe von Eingangssignalen v. Wie oben geschildert überlagern sich diese Eingabewerte mit den rückgekoppelten Neuronenaktivitäten, so daß man das Gleichungssystem (2) erhält. Die Werte f_r lassen sich bekanntlich nur durch aufwendige numerische Verfahren bestimmen, weswegen Kohonen in seinem Modell vorschlägt, daß die Erregungsantwort eines Neurons r auf ein Eingabesignal v durch eine (Nachbarschafts-) Funktion $h_{rr_{max}}$ gegeben ist, wobei r_{max} wiederum das Erregungszentrum für v darstellt. Als Folge eines einzelnen Reizes v sollten sich dann die Synapsenstärken w_{rl} um den Wert

$$\delta w_{rl} = \varepsilon(h_{rr_{max}} v_l - h_{rr_{max}} w_{rl})$$

korrigieren.

Der erste Term $h_{rr_{max}} v_l$ entspricht dabei der "Hebbschen Lernregel" und besagt, daß sich die Synapsenstärke um so mehr vergrößert, je mehr die durch die Eingabe v_l erzielte Erregung $h_r r_{max}$ mit dieser Eingabe übereinstimmt. Der zweite Term $h_{rr_{max}} w_{rl}$ hemmt die Veränderung δw_{rl} der Synapsenstärke proportional zur bereits existierenden Synapsenstärke w_{rl}. Dadurch wird verhindert, daß w_{rl} unbegrenzt wächst. Der Faktor ε , $0 < \varepsilon < 1$, regelt die Schrittweite eines einzelnen Lernprozesses. Dabei ist es sinnvoll, mit einem großen Wert für ε , d.h. nahe bei 1 zu starten und diesen dann mit wachsender Anzahl t der durchgeführten Lernschritte in Richtung auf 0 zu verringern, ohne diese jedoch jemals zu erreichen. Große ε zu Beginn der Lernphase erlauben ein grobes Herausbilden der Karte. Das stetige Abfallen der Werte von ε garantiert anschließend, daß sich diese Karte in einem relativ stabilen Zustand einpendelt, ohne jedoch jemals völlig statisch zu werden. Denn eine gewisse Adaptionsfähigkeit der Neuronen wird durch $\varepsilon > 0$ gesichert.

Kohonen empfiehlt, für die Nachbarschaftsfunktion $h_r r_{max}$ eine Funktion zu wählen, die ihr Maximum bei r_{max} hat und mit wachsendem Abstand $\| r - r_{max} \|$ abnimmt. Konkret schlägt er die Funktion

$$h_{rr_{max}} = \exp\left(-\frac{\| r - r_{max} \|^2}{2\sigma^2}\right) \tag{7.10}$$

vor.

Die Bedeutung von σ in (10) zeigt Abbildung 7.4. σ setzt den Radius des Wirkungskreises fest, innerhalb dessen $h_{rr_{max}}$ eine Veränderung der w_{rl} bewirken kann. Um eine Grobstrukturierung der Karte zu Beginn des Lernprozesses und eine Spezialisierung der Neuronen am Ende des Lernprozesses zu fördern, sollte sich σ analog zu ε bei wachsender Anzahl der Lernschritte t verringern.

Das letzte Problem, das noch zu lösen ist, ist die Frage, wie die Eingangsreize $v = (v_1, ...v_d) \in V$ gewählt werden. Es wird gefordert, daß diese Signale unabhängig voneinander sind und ihr Auftreten einer Wahrscheinlichkeitsdichte $P(v)$ unterliegt. Die einzelnen Wahrscheinlichkeiten der Eingabevektoren v ermöglichen es, das Aussehen der Karte zu manipulieren. Denn ein Vektor v_1,

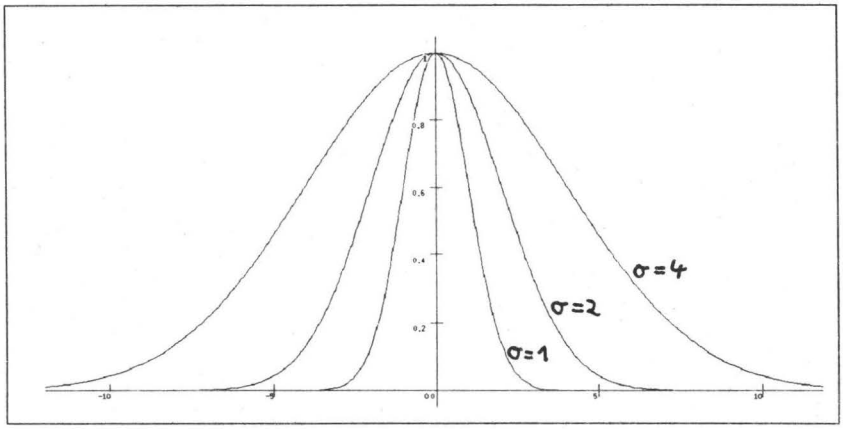


Abbildung 7.4: Graph der Nachbarschaftsfunktion $h_{rr_{max}}$ in Abhängigkeit von σ

der eine höhere Auftrittswahrscheinlichkeit $P(v_1)$ hat als ein Vektor v_2, wird häufiger ausgesendet. Konsequenz ist, daß Vektor v_1 in der Karte auf einem größeren Gebiet und damit detaillierter abgebildet wird als v_2.

Algorithmus des Modells von Kohonen

1. **Initialisierung :** Belege die Synapsenstärken w_{rl} geeignet, falls a-priori-Informationen vorliegen. Ansonsten wähle zufällige Werte.

2. **Stimuluswahl :** Wähle gemäß der Wahrscheinlichkeitsdichte $P(v)$ einen zufälligen Vektor v, der ein "sensorisches Signal" repräsentiert.

3. **Response :** Bestimme das Erregungszentrum r_{max} durch

$$\| v - w_{r_{max}} \| \leq \| v - w_r \| \tag{7.11}$$

   für alle $r \in A$.

4. **Adaptionsschritt :** Führe einen Lernschritt durch Veränderung aller Synapsenstärken gemäß

$$w_r^{neu} = w_r^{alt} + \varepsilon h_{rr_{max}}(v - w_r^{alt}) \tag{7.12}$$

   durch.

   Gehe zu Schritt 2.

Aufgrund der Schritte 2 bis 4 erhält man die in (9) erwähnte Projektion $\phi_w : V \rightarrow A$, welche die neuronale Karte des Eingangssignalraumes V auf dem Gitter A repräsentiert. Dabei werden die wesentlichsten Nachbarschaftsbeziehungen zwischen den Eingabereizen, d.h. die Topologie von V abgebildet. Deshalb prägte Kohonen dafür den Namen "topologieerhaltende Merkmalskarte".

7.2.2 Ein erstes Beispiel

Der Algorithmus des Modells von Kohonen soll zunächst an einem einfachen Simulationsbeispiel veranschaulicht werden. Abbildung 7.5a zeigt die Versuchssituation. Gegeben ist ein krummlinig begrenztes Gebiet G, in dem sich eine Schallquelle bewegt, die zufällig von Zeit zu Zeit ein Signal aussendet. Dieses wird von zwei Mikrophonen am unteren Rand von G empfangen und über

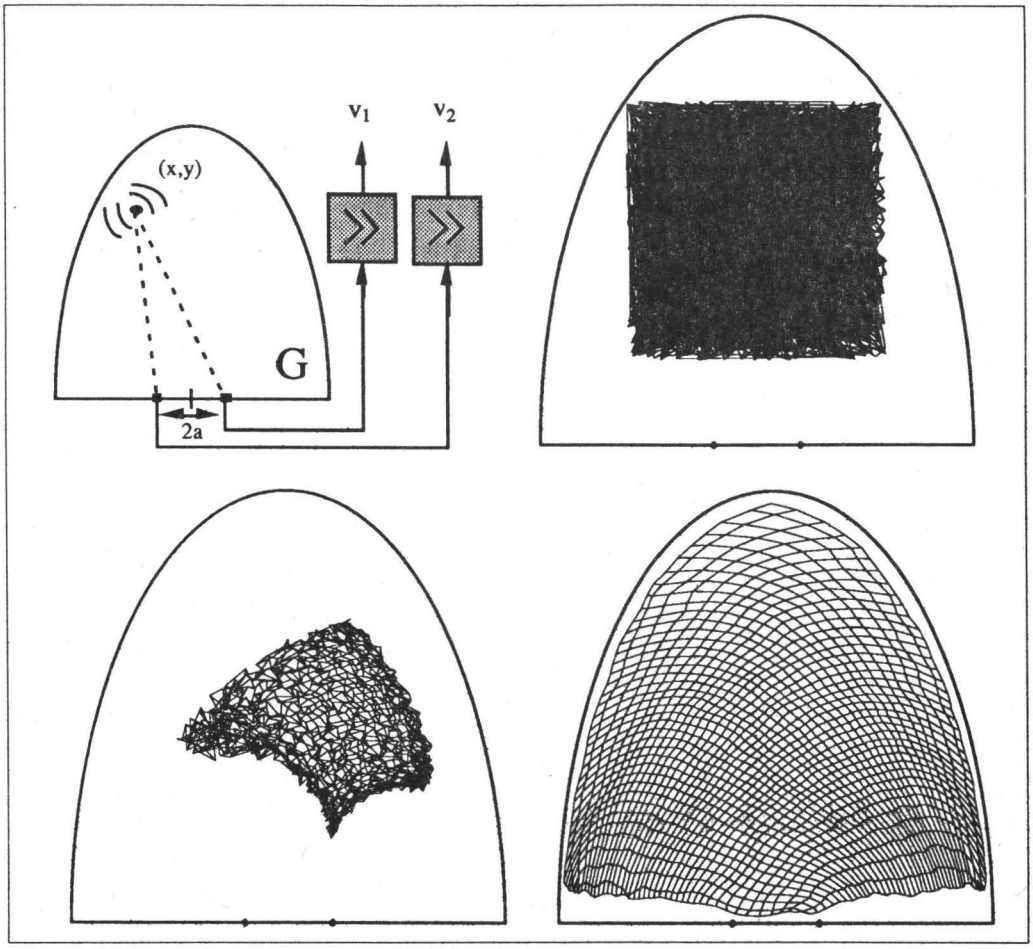


Abbildung 7.5: Versuchsaufbau; Zuordnungen der Neuronen und Ortspositionen : Ausgangssituation; nach 100 Lernschritten; nach 40000 Lernschritten

zwei Verstärker schließlich als "sensorische Signale" v_1 und v_2 an das Neuronengitter A weitergeleitet. Dieses Gitter besteht aus 1600 Neuronen, die in einer 40×40-Matrix angeordnet sind. Charakterisiert wird jedes Neuron r durch einen Vektor $w_r = (w_{r1}, w_{r2})$.

Ein einzelner Lernschritt besteht nun im Verarbeiten eines Verstärkersignals $v = (v_1, v_2)$ gemäß dem Algorithmus von Kohonen. Ziel des Lernprozesses ist, daß sich jedes Neuron anhand der Eingabesignale $v = (v_1, v_2)$ auf ein begrenztes Aufenthaltsgebiet der umherwandernden Schallquelle spezialisiert. Dies sollen die Neuronen so koordinieren, daß das Gitter A schließlich ein Abbild von G wird, d.h. in der Matrix benachbarte Neuronen sollen auch auf benachbarte Gebiete in G ansprechen.

Die Abbildungen 7.5b bis 7.5d dokumentieren den Lernprozeß. Es wurden dabei jedem Neuron $r \in A$ die Koordinaten (x, y) zugeordnet, die der Position der Schallquelle in G entsprechen, wenn diese durch ein Schallsignal den Vektor $v = (v_1, v_2)$ auslöst, auf den sich das Neuron r am stärksten ausgerichtet hat. Die Punkte zweier benachbarten Neuronen wurden mit einer Linie verbunden. Ausgehend von einer zufälligen Zuordnung zwischen Punkten (x, y) in G und Neuronen $r \in A$ (Abb. 7.5b) hat der Algorithmus nach 40000 Lernschritten das krummlinig begrenzte Gebiet G auf das quadratische Gitter A abgebildet.

7.3 Anwendung auf das Problem des Handlungsreisenden

Bei dem Simulationsbeispiel am Ende von Paragraph 2 waren sowohl der Raum der Eingangssignale als auch das Neuronennetz zweidimensional. In einem solchen Fall erwartet man, daß der Algorithmus von Kohonen nach genügend vielen Lernschritten eine adäquate Karte des Ursprungraumes der Signale im Neuronennetz entstehen läßt.

Wenn jedoch die Dimension des Netzwerks kleiner ist als die des Raumes der Eingangssignale, ist es nicht unmittelbar ersichtlich, daß der Algorithmus ein "sinnvolles" Ergebnis liefert. Klar ist, daß die kleinere Dimensionalität des Abbildraumes einen Informationsverlust nach sich zieht. Dennoch können solche Karten zur Lösung höchst nichttrivialer Aufgaben eingesetzt werden. Dies soll in diesem Paragraphen anhand des "Problems des Handlungsreisenden" gezeigt werden.

Bemerkt sei an dieser Stelle noch, daß auch zum Beispiel die Abbildung eines eindimensionalen Raums von Eingangsreizen auf einen zweidimensionalen Raum von Neuronen möglich ist und sogar bei biologischen neuronalen Netzen vorkommt.

7.3.1 Eindimensionale Karten

In diesem Abschnitt werden zunächst einige allgemeine aber nur qualitativen Charakter besitzende Überlegungen bzgl. eindimensionaler "Karten" angestellt, bevor diese dann im nächsten Abschnitt am Beispiel des Problems des Handlungsreisenden illustriert werden.

Es wird folgende Situation zugrundegelegt. Der Raum V der Eingangssignale ist mindestens zweidimensional. Die N Neuronen werden in einer eindimensionalen Kette A angeordnet vorausgesetzt. Dadurch ist eine Reihenfolge unter den Neuronen gegeben, d.h. jedes Neuron erhält aufgrund seiner Position in der Kette eine Nummer. Desweiteren beinhaltet jedes Neuron r einen Vektor w_r, der einen konkreten Punkt im mehrdimensionalen Eingaberaum V repräsentiert. Durchläuft man nun die Neuronenkette A gemäß ihrer Durchnummerierung, so wird durch die den Neuronen zugeordneten Vektoren ein Weg im Raum V beschrieben. Mathematisch formuliert ist dieser Weg das Bild der endlichen Folge

$$A \rightarrow V \qquad \text{mit} \qquad r \mapsto w_r$$

Auf dieser Betrachtungsebene kann man den Algorithmus von Kohonen als ein Verfahren interpretieren, mit dem ein Weg im Raum V schrittweise optimiert wird. Der Lernprozeß läuft dabei wie folgt ab.

Ausgangspunkt ist eine eventuelle zufällige Belegung der Vektoren w_r, d.h. der Weg durchläuft zunächst N beliebige Punkte im Raum V. Bei jedem einzelnen Lernschritt sucht ein Eingangssignal $v \in V$ zunächst das Neuron $s \in A$ heraus, dessen Vektor w_s den Punkt in V repräsentiert, der am nächsten bei v liegt. (Schritt 3, Response)

Der Adaptionsschritt in Kohonens Algorithmus bewirkt anschließend eine Verschiebung der Vektoren w_r, $r \in A$, in Richtung des Eingabevektors v. Die Änderung ist um so stärker, je dichter Neuron r bei Neuron s liegt. Für den durch die Vektoren w_r gegebenen Weg in V bedeutet ein solcher Lernschritt folglich ein Verformung in Richtung auf v hin. Das Ergebnis genügend vieler Lernschritte ist ein Weg in V, der vor allem die Gebiete in V besucht, von denen relativ viele Eingabesignale ausgesendet werden.

Dies ermöglicht es, den Weg in bevorzugte Gebiete von V zu lenken, indem bei der Stimuluswahl (Schritt 2) eine geeignete Wahrscheinlichkeitsdichte $P(v)$ vorausgesetzt wird.

Das Kriterium, nach dem der Weg während des Lernprozesses optimiert wird, ergibt sich unmittelbar aus der dem Algorithmus zugrundeliegenden zentralen Idee: Neuronen r_1 und r_2, die auf der Kette A eng beieinander liegen, sollen Vektoren w_{r1} und w_{r2} zugeordnet werden, die im Raum eng benachbart sind. Der resultierende Weg wird also möglichst kurz sein. Dies führt direkt zum Problem des Handlungsreisenden.

7.3.2 Der Weg des Handlungsreisenden als eindimensionale Karte

Das Problem des Handlungsreisenden (Travelling Salesman Problem) besteht bekanntlich darin, die kürzeste Rundreise durch l vorgegebene Städte S_i zu finden; ausgehend von einer bestimmten

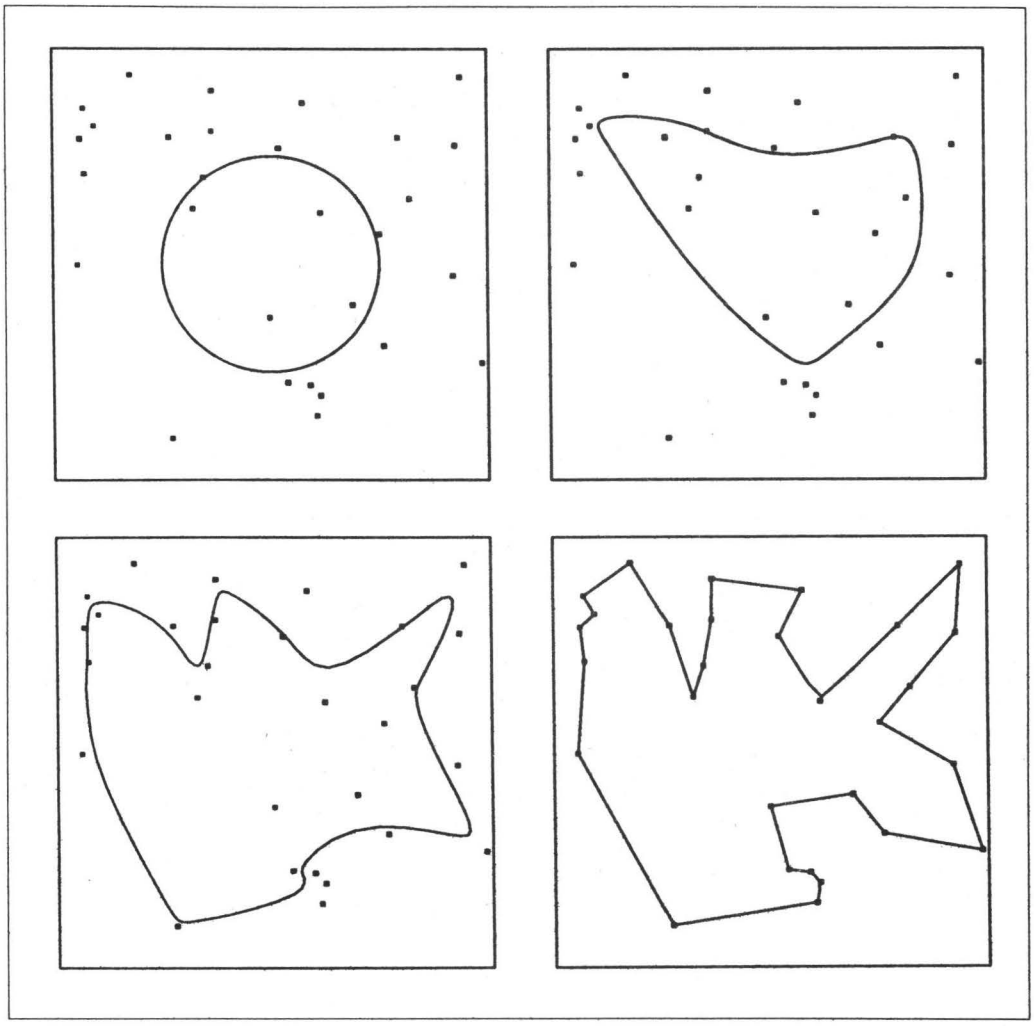


Abbildung 7.6: Ausgangssituation; nach 5000 Lernschritten; nach 7000 Lernschritten;nach 10000 Lernschritten

Stadt soll also jede Stadt genau einmal besucht werden, und der Weg soll am Startpunkt wieder enden. Es handelt sich hierbei um ein sogenanntes **NP**-hartes Problem, d.h. jeder zur Zeit bekannte Algorithmus zur Lösung dieser Aufgabe besitzt exponentielle Laufzeit $O(2^n)$. Das exponentielle Wachstum hat zur Folge, daß das Bestimmen einer optimalen Lösung bereits bei 30 vorgegebenen Städten praktisch nicht mehr möglich ist. Bei NP-harten Problemen sucht man daher stets nach Algorithmen, die bei polynomialem Zeitaufwand Näherungslösungen bestimmen.

Die im folgenden beschriebene Computersimulation zeigt, daß auch der Algorithmus von Kohonen geeignet ist, Näherungslösungen zu bestimmen. Dabei wurden in einem Einheitsquadrat l=30 "Städte" zufällig ausgewählt. Es waren $N = 800$ Neuronen vorhanden, die in einer ringförmig geschlossenen Kette A angeordnet wurden. Die den Neuronen $r \in A$ zugeordneten Vektoren w_r werden gemäß Gleichung (12) iterativ verändert. Bei jedem einzelnen Lernschritt wird zufällig eine der l Städte S_i ausgewählt, indem ihr Ortsvektor bestimmt wird. Dabei ist die Wahrscheinlichkeit p_i, daß die Stadt S_i ausgewählt wird, stets gleich groß; es gilt also für alle $i \in \{1, ..., l\}$: $p_i = \frac{1}{l}$. Für die Funktion $h_{rr_{max}}$ wurde die Gaußglocke, Gleichung (10), gewählt. Die restlichen Simulationsdaten waren $\varepsilon = 0,8$, $\sigma(t) = 0,02^{\frac{t}{t_{max}}}$ und $t_{max} = 10000$.

Im Initialisierungsschritt wurden die 800 Vektoren w_r mit den Ecken eines im Quadrat zentrierten regelmäßigen 30-Ecks belegt. (Abb. 7.6a). Die Abbildungen 7.6b bis d zeigen den erreichten Polygonzug nach 5000, 7000 sowie 10000 Lernschritten. Man erkennt, wie sich der Weg zunächst grob deformiert (Abb. 7.6b), um dann im Zuge der Verringerung der Reichweite von $h_{rr_{max}}$ detailliertere Formen anzunehmen (Abb. 7.6c), bis der Weg schließlich nach 10000 Schritten durch die vorgegebenen "Städte" läuft. Die Länge des Weges beträgt 4,5888 Längeneinheiten und ist in diesem Fall sogar der kürzestmögliche.

Dieses Beispiel zeigt eindrucksvoll, daß Kohonens Modell bereits in Form der recht einfachen eindimensionalen Karten zur Lösung nicht-trivialer Probleme eingesetzt werden kann.

7.4 Semantische Karten

7.4.1 Problematik diskreter Symbole

Bei den bisher vorgestellten Ideen und Beispielen waren entscheidende Voraussetzungen stets gewesen, daß der Raum V der Eingangssignale ein kontinuierliches Spektrum besaß und man auf V problemlos eine Metrik definieren konnte. Ohne eine Metrik auf V ist zum Beispiel der Schritt 3 (Response) in Kohonens Algorithmus nicht durchführbar und damit natürlich der gesamte Algorithmus hinfällig. Auf diese notwendigen Bedingungen wurde bisher nicht näher eingegangen, weil die bisherigen Beispiele diese Eigenschaften von sich aus lieferten.

Betrachtet man jedoch Prozesse auf höherem Abstraktionsniveau, wie zum Beispiel die Sprache oder die Fähigkeit der logischen Argumentation und Schlußfolgerung, erkennt man schnell, daß diese Vorgänge auf der Verarbeitung diskreter Symbole beruhen. Charakteristisch für Symbole ist, daß ihre syntaktische Repräsentation im allgemeinen in keinem Zusammenhang mit ihrer Bedeutung steht. Also ist es hier nicht möglich, eine enge semantische Nachbarschaft zweier Symbole auf deren syntaktische Ähnlichkeit zurrückzuführen und auf dieser Ebene eventuell eine Metrik für diskrete Symbole zu definieren.

Es stellt sich daher die interessante Frage, wie das Gehirn symbolische Objekte "sinnvoll" repräsentiert.

7.4.2 Lösungsvorschlag

Es wird basierend auf Kohonens Modell selbstorganisierender Karten ein Ansatz vorgestellt, der zeigt, wie man eine semantische Karte von Symbolen erhält, bei der logisch verwandte Symbole in räumlich benachbarte Gebiete abgebildet werden. Das Hauptproblem liegt dabei, wie bereits erwähnt, im Finden einer geeigneten Metrik, um mit ihr der Symbolmenge sozusagen eine "topographische Maske" überzuziehen.

Eine mögliche Lösung besteht darin, daß man während der Lernphase, d.h. während der Entwicklung der Karte, die Symbole dem neuronalen Netzwerk in einem "geeigneten" Kontext präsentiert. Das passende Umfeld eines Symbols sind nun gerade die Attribute, die es symbolisiert. Am einfachsten modelliert man dies, indem man einen Eingabevektor x zweiteilt. Die eine Komponete x_S steht für das syntaktische Symbol selbst und die zweite Komponente x_A kennzeichnet die dem Symbol zugeordnete Menge von Attributen. Man kann dann x als die Summe zweier zueinander orthogonaler Vektoren schreiben :

$$x = \begin{pmatrix} x_S \\ x_A \end{pmatrix} = \begin{pmatrix} x_S \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ x_A \end{pmatrix} \tag{7.13}$$

Die Kernidee bei selbstorganisierenden semantischen Karten beruht nun darauf, die während der Lernphase verwendete Metrik so zu wählen, daß der Beitrag von x_A den von x_S dominiert. Dies ist durch entsprechende Faktoren leicht zu erreichen. Ergebnis einer auf dieser Methodik beruhenden Anwendung des Algorithmus von Kohonen wird eine Karte sein, die enge Beziehungen auf der Attributen-Ebene widerspiegelt. Da die Symbolkomponete x_S stets mitgeführt wird, werden die Symbole entsprechend ihrer Attribute, d.h. entsprechend ihrer Semantik in der Karte abgelegt. In

Attribute	dove	hen	duck	goose	owl	hawk	eagle	fox
ist klein	1	1	1	1	1	1	0	0
ist mittel	0	0	0	0	0	0	1	1
ist groß	0	0	0	0	0	0	0	0
hat 2 Beine	1	1	1	1	1	1	1	0
hat 4 Beine	0	0	0	0	0	0	0	1
hat Haare	0	0	0	0	0	0	0	1
hat Hufe	0	0	0	0	0	0	0	0
hat Mähne	0	0	0	0	0	0	0	0
hat Federn	1	1	1	1	1	1	1	0
jagt	0	0	0	0	1	1	1	1
läuft	0	0	0	0	0	0	0	1
fliegt	1	0	0	1	1	1	1	0
schwimmt	0	0	1	1	0	0	0	0

Attribute	dog	wolf	cat	tiger	lion	horse	zebra	cow
ist klein	0	0	1	0	0	0	0	0
ist mittel	1	1	0	0	0	0	0	0
ist groß	0	0	0	1	1	1	1	1
hat 2 Beine	0	0	0	0	0	0	0	0
hat 4 Beine	1	1	1	1	1	1	1	1
hat Haare	1	1	1	1	1	1	1	1
hat Hufe	0	0	0	0	0	1	1	1
hat Mähne	0	1	0	0	1	1	1	0
hat Federn	0	0	0	0	0	0	0	0
jagt	0	1	1	1	1	0	0	0
läuft	1	1	0	1	1	1	1	0
fliegt	0	0	0	0	0	0	0	0
schwimmt	0	0	0	0	0	0	0	0

Abbildung 7.7: Tiernamen und ihre Attribute

der entstandenen Karte werden demnach logische oder semantische Ähnlichkeiten zweier Symbole durch deren enge räumliche Nachbarschaft zum Ausdruck gebracht. Damit ist das Problem der Repräsentation diskreter Symbole gelöst.

7.4.3 Beispiel einer selbstorganisierenden semantischen Karte

Ausgangspunkt ist Abbildung 7, wobei jede der 16 Spalten ein Tier anhand von 13 Eigenschaften beschreibt. Eine "1" bedeutet, daß das Tier diese Eigenschaft besitzt, eine "0", daß es sie nicht besitzt. Eine solche Spalte stellt im folgenden den Attributvektor x_A des entsprechenden Tieres dar. Die Ähnlichkeit zweier Tiere bzgl. dieser Attributenmenge kann man dann leicht mit Hilfe des Skalarproduktes feststellen; denn je mehr gemeinsame Attribute zwei Tiere haben, um so größer ist das Skalarprodukt ihrer beiden Merkmalsvektoren. Das Tier selbst ist nicht Teil von x_A, sondern erzeugt den Symbolvektor x_S. Um zu verhindern, daß in der Lernphase diese Komponente x_S Informationen über Ähnlichkeiten zwischen den Tieren mitführt, realisiert man x_S als einen $d = 16$-dimensionalen Vektor, in dem für das k-te Tier die k-te Stelle einen festen Wert $a = 0.2$ erhält; alle anderen Stellen werden auf Null gesetzt. Für den Tiger erhält man dann zum Beispiel den 29-dimensionalen Vektor $x = (x_S, x_A)^T$ mit $x_S = (0,0,0,0,0,0,0,0,0,0,0,0,0.2,0,0,0,0)$ und $x_A = (0,0,1,0,1,1,0,0,0,1,1,0,0)$.

Als neuronales Netzwerk wurde eine zweidimensionale 10×10 Matrix verwendet. Die 29 Synapsenstärken aller 100 Neuronen wurden im Initialisierungsschritt zufällig belegt, so daß keine a-priori-Informationen gegeben waren. Diesem Netzwerk wurden dann in zufälliger Reihenfolge jeweils einer der 16 29-dimensionalen Vektoren eingegeben. Nach 2000 Lernschritten gemäß dem Algorithmus von Kohonen hatten sich die einzelnen Zellen auf eine bestimmte Merkmalskombination spezialisiert.

Abbildung 7.8 ergab sich, wenn man das trainierte Netzwerk daraufhin testete, welche Zelle die stärkste Erregung zeigte, wenn man ihm als Eingabe nur das Tiersymbol $x = (x_S, 0)^T$ gab. Abbildung 7.9 zeigt, auf welches Tiersymbol jede einzelne Zelle am stärksten reagierte. In beiden

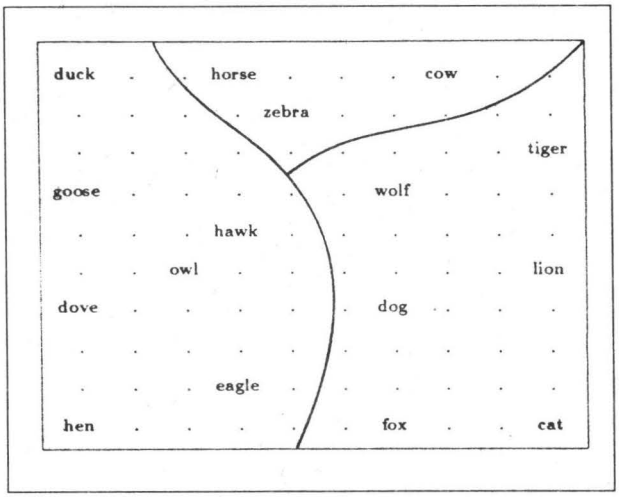


Abbildung 7.8: Simulationsergebnis

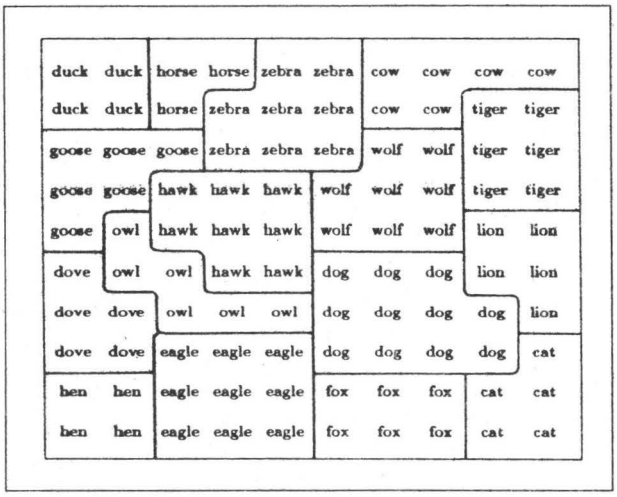


Abbildung 7.9: Simulationsergebnis

Abbildungen erkennt man, daß sich nicht nur benachbarte Zellen auf dasselbe Tier spezialisiert haben, sondern die räumliche Anordnung spiegelt wesentliche übergeordnete biologische Familienbeziehungen unter den Tieren wider; so findet man zum Beispiel in der linken Hälfte der Matrix die Gruppe der Vögel, in der rechten die der "Jäger". Das Netzwerk hat also selbständig eine hierarchische Ordnung aufgebaut, nach der die Karte zunächst in generelle Tierklassen unterteilt wurde und innerhalb einer solchen Klasse haben sich dann benachbarte Zellen auf ein konkretes Tier spezialisiert.

Auch wenn dies ein ziemlich idealisiertes Beispiel ist, so zeigt es doch, daß selbstorganiesierende Karten bei Verwendung von Kohonens Algorithmus in der Lage sein können, diskrete symbolische Daten so abzubilden, daß wesentliche Gemeinsamkeiten durch räumliche Nachbarschaft widergegeben werden.

7.5 Mathematische Betrachtung des Kohonen-Modells

Die in den letzten Paragraphen beschriebenen Beispiele haben gezeigt, daß das Modell von Kohonen in sehr unterschiedlichen Bereichen der Informationsverarbeitung beeindruckende Ergebnisse liefert. In diesem Paragraphen soll angedeutet werden, daß die in den Beispielen festgestellten Eigenschaften des Modells auch auf einer allgemeineren, mathematischen Ebene interpretiert werden können. Exemplarisch behandelt wird die bereits beim Problem des Handlungsreisenden festgestellte Fähigkeit des Algorithmus von Kohonen, Punkte eines höherdimensionalen Raumes in möglichst exakter Weise in einen Raum kleinerer Dimension abzubilden.

7.5.1 Problematik: "Verborgene Parameter"

Bei einem Experiment wird ein einzelnes Meßergebnis meistens durch eine Liste von Parametern $v = (v_1, ...v_L)$ charakterisiert. Dabei sind diese Parameter in der Regel nicht alle unabhängig voneinander, sondern die Einstellung eines Parameters beeinflußt den Wert eines anderen. Diese "sichtbaren" Parameter sind also oft beschreibbar durch eine kleinere Anzahl sogenannter "verborgener Parameter" $r_1, ..., r_D$ mit $D < L$; d.h. jedes v_i kann als Funktion der verborgenen Parameter geschrieben werden:

$$v_i = f_i(r_1, ..., r_D), i = 1, ..., L$$

Diese r_i repräsentieren zum einen realistischer die im Experiment wirklich vorhandenen Freiheitsgrade und sind zum anderen wegen $D < L$ meist einfacher zu handhaben. Deshalb ist es wünschenswert, diese "verborgenen" Parameter zu ermitteln. Ihre Bestimmung wird jedoch dadurch erheblich erschwert, daß sie nicht eindeutig festgelegt sind - vergleichbar mit der Nichteindeutigkeit der Basen des $\mathbb{R}^n$.

Wie so oft versucht man daher, nicht die exakten r_i zu bestimmen, sondern es genügen für praktische Anwendungen Näherungswerte. Die folgenden beiden Paragraphen beinhalten Verfahren, die zur näherungsweisen Bestimmung der "verborgenen" Parameter verwendet werden, sowie den Zusammenhang dieser Verfahren mit Kohonens Algorithmus.

7.5.2 Faktoranalyse, ein linearer Lösungsansatz

Beim ersten Verfahren, der **Faktor- oder Varianzanalyse**, wird die einschränkende Annahme gemacht, daß zwischen den sichtbaren Parametern $v = (v_1, ..., v_L)$ und den verborgenen $r = (r_1, ..., r_D)$ im wesentlichen ein linearer Zusammenhang besteht. Auf geometrischer Ebene entspricht dies der Einführung einer D-dimensionalen Hyperebene Λ , die im L-dimensionalen Raum Γ der Daten liegt und die so gewählt werden sollte, daß sich jeder Datenpunkt aus Γ möglichst gut durch einen Punkt der Hyperebene Λ approximieren läßt.

Wird die Hyperebene Λ durch die $D+1$ Vektoren $w^0, ..., w^D \in \mathbb{R}^n$ aufgespannt, so kann man jeden Datenpunkt $v \in \Gamma$ schreiben als

$$v = w^0 + \sum_{i=1}^{D} w^i r_i(v) + d_w(v) \tag{7.14}$$

Dabei sind die $r_j(v)$ die zu v gehörenden, neuen "verborgenen" Parameter bzgl. der Hyperebene Λ .

Da der oben angenommene lineare Zusammenhang im allgemeinen nicht gegeben ist, liegen die Datenpunkte v nicht alle in der Hyperebene, sondern besitzen einen bestimmten senkrechten Abstand zu Λ. Dies wird durch den Korrektursummanden $d_w(v)$ berücksichtigt. Dieser hängt natürlich entscheidend von der Wahl der Hyperebene Λ ab, was der Index w anzeigt.

Die optimale Hyperebene, d.h. die nach der Varianzmethode optimalen verborgenen Parameter, erhält man, wenn die Gesamtheit all dieser Fehler $d_w(v)$ möglichst klein ist. Dazu addiert man alle mittleren quadratischen Restfehler $\| d_w(v) \|^2 = \| v - w^0 - \sum_{i=1}^{D} w^i r_i(v) \|^2$, $v \in \Gamma$ auf. Berücksichtigt wird zusätzlich, daß nicht alle Datenpunkte v gleichwahrscheinlich sind; sondern gemäß

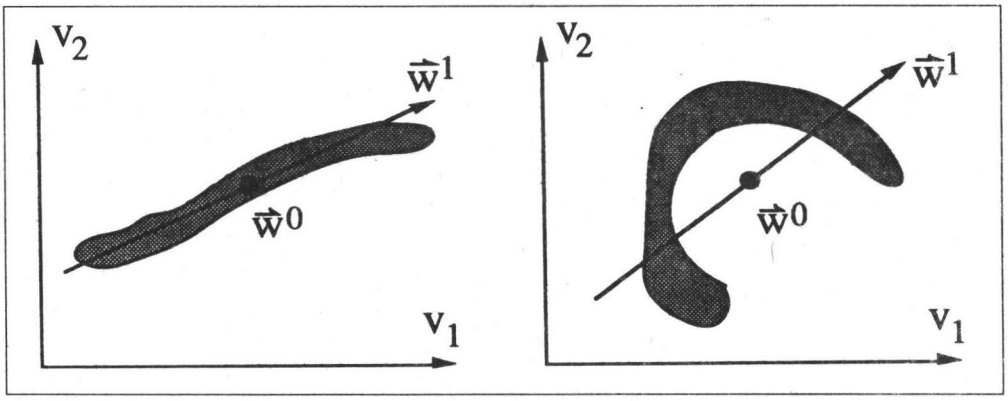


Abbildung 7.10: Beschreibung einer zweidimensionalen Datenverteilung (schraffiertes Gebiet) im $v_1 - v_2$-Raum durch eine eindimensionale "Ebene"; linearer Fall und nichtlinearer Fall

einer Wahrscheinlichkeitsdichte $P(v)$ auftreten. Mit dieser Wahrscheinlichkeit werden die Restfehler $\| d_w(v) \|^2$ gewichtet. Man erhält schließlich folgende Extremalbedingung an die Vektoren w^i:

$$\int \| v - w^0 - \sum_{i=1}^{D} w^i_{r_i}(v) \|^2 \, P(v) d^L v \| = Minimum! \tag{7.15}$$

Diese Forderung erlaubt eine Bestimmung der w_i, auf die hier nicht eingegangen werden kann.

Wichtiger ist die Bemerkung, daß dieses Verfahren dann gute Ergebnisse liefert, wenn die angenommene lineare Beziehung zwischen den v_i und den r_j gegeben ist (siehe Abb. 7.10a). Ansonsten liefert dieses Verfahren keine adäquate Beschreibung der Datenpunkte (siehe Abb. 7.10b).

7.5.3 Hauptkurven und Kohonen Algorithmus

Die in 5.2 vorausgesetzte Linearität wird jetzt fallengelassen und der allgemeine nichtlineare Zusammenhang zwischen Datenpunkten und verborgenen Parametern betrachte. Die Überlegungen beschränken sich auf den eindimesionalen Fall. In Abbildung 7.10b ekennt man, daß eine durch den Raum der Daten gelegte Kurve $f(s)$, die Anordnung der Datenpunkte besser annähert als die durch die Varianzanalyse erhaltenen Gerade. Es gibt auch innerhalb der Klasse von Kurven im Datenraum Γ eine Kurve f, die die Daten entsprechend der Wahrscheinlichkeitsdichte $P(v)$ am "besten" abbildet. Optimal heißt auch wieder, daß die Summe aller mittleren quadratischen Abstände der Datenpunkte v zur Kurve gewichtet mit $P(v)$ minimal wird. Ist also $d_f(v)$ der senkrechte Abstand von v zur Kurve f, so wird aus der Extremaleigenschaft (15) jetzt :

$$D_f = \int d^2 f(v) P(v) d^L v = Minimum! \tag{7.16}$$

Eine Kurve f, die diese Eigenschaft hat, nennt man **Hauptkurve**. Diese Überlegungen lassen sich mit dem entsprechenden mathematischen Aufwand auf höherdimensionale Räume ausdehnen, man erhält dann sogenannte **"Hauptflächen"**.

Hier ist interessanter, welche Beziehung zwischen Hauptkurven bzw. Hauptflächen und dem Algorithmus von Kohonen besteht. Der Algorithmus von Kohonen läßt sich, wie zum Beispiel auch das Problem des Handlungsreisenden zeigt, als ein Verfahren interpretieren, mit dem man im Bereich der neueronalen Netzwerke solche Hauptkurven bestimmen kann. Er stellt damit auch eine Verallgemeinerung und Verbesserung der Faktor- bzw Varianzanalyse dar.

7.6 Schlußbemerkung

Dieses Kapitel basiert auf zwei Literaturquellen. Paragraph 4, der die semantischen Karten als Anwendung des Algorithmus von Kohonen beinhaltet, liegt im wesentlichen die Arbeit von Kohonen und Ritter aus dem Jahre 1989 zugrunde. In ihr geben die Autoren zunächst eine mehr qualitative Beschreibung des Modells von Kohonen, bevor sie dessen Leistungsfähigkeit durch zwei Beispiele aus dem Bereich der diskreten Symbolverarbeitung demonstrieren. Ist man mehr an semantischen Karten interessiert, so ist diese Arbeit zu empfehlen. Um jedoch einen allgemeineren und tiefergehenden Überblick über das Modell von Kohonen zu gewinnen, empfiehlt es sich, das Buch von Ritter, Martinez und Schulten zu lesen. Dieses diente auch im wesentlichen als Grundlage für die anderen Teile des Kapitels, also für die Vorstellung des Modells in Paragraph 2, die restlichen Beispiele und die mathematische Interpretation in Paragraph 5. Dieses Buch ist sehr zu empfehlen, da es dem Leser ermöglicht, sich auf verschiedenen Ebenen mit der Thematik zu befassen. Derjenige, der sich nur einen schnellen Überblick verschaffen will, wird genauso zufrieden sein, wie der, der an einem tieferen Verständnis und an mathematischen Hintergründen interessiert ist. Das Buch ist so aufgebaut, daß man die Teile, die einen nicht so sehr interessiern, überspringen kann, ohne im folgenden Verständnisprobleme zu haben. Dazu tragen im wesentlichen auch die vielen interessanten Anwendungsbeispiele bei und die Vergleiche mit biologischen neuronalen Netzen.

7.7 Literatur

[1] Helge Ritter, Thomas Martinez, Klaus Schulten. *Neuronale Netze, Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke*, Bonn; München; Reading, Mass (u.a.): Addison-Wesley, 1991.

[2] H. Ritter, T. Kohonen. Self-Organizing Sematic Maps, *Biological Cybernetics 61, 241-254*, Springer-Verlag, 1989

Teil II

Anwendungen

Kapitel 8

Neuronale Netze in der Robotik

Michael Dieterich

Übersicht

Nachdem im vorangegangenen Kapitel die Grundlagen von Kohonens Modell erklärt wurden, geht es jetzt um dessen Anwendung in der Robotik. Zunächst werden einige Erweiterungen des Modells beschrieben, die es für Steuerungsaufgaben in der Robotik einsetzbar machen. Die Funktionsweise des erweiterten Kohonen-Algorithmus wird dann an zwei Beispielen demonstriert. Dabei geht es um das „Stabbalance-Problem" und ein Steuerungsbeispiel aus der Okulomotorik.

Der zweite Teil dieses Kapitels beschäftigt sich dann mit Problemstellungen in der Robotersteuerung. Hierzu werden Simulationen zur visuomotorischen Koordination eines Roboterarms und zur Manipulatorsteuerung mittels hierarchischer Netze vorgestellt. Das Kapitel schließt mit einigen Anmerkungen zur Problematik ballistischer Bewegungen eines Roboters.

8.1 Einleitung

In der Robotik werden Neuronale Netze für die verschiedensten Aufgaben verwendet. Die größte Bedeutung kommt hierbei den Kohonen-Netzen zu, die einerseits der Verarbeitung externer Signale dienen, andererseits die Robotersteuerung mithilfe dieser Signale übernehmen. Es kann sich dabei um eine einfache Positionierung des Manipulators in einem vorgegebenen Arbeitsbereich handeln, um die Steuerung des Manipulators entsprechend einer Greifstrategie, aber auch um die Koordination der Bewegungen eines mehrbeinigen insektenähnlichen Roboters mittels hierarchischer Netze.

Andere Netztypen, wie zum Beispiel „Perzeptronen-Netze", werden zur Sprach- und Mustererkennung eingesetzt. Sie verarbeiten also externe Signale und geben diese als Eingangssignale für Steuerungsaufgaben an die oben beschriebenen Netze weiter.

Die zentrale Bedeutung der Kohonen-Netze wird durch die wichtige Rolle topologieerhaltender Karten im Gehirn motiviert. Mit Hilfe eines stochastischen Lernalgorithmus', auf der Basis sensorischer Signale, bilden sich Kohonen-Netze als solche Karten aus und repräsentieren die Nachbarschaftsverhältnisse ihrer Eingabesignale.

8.2 Selbstorganisierende Karten

8.2.1 Anpassung von Kohonens Modell an Steuerungsaufgaben in der Robotik – Motorische Karten

Die Bewegungssteuerung der Muskulatur erfogt über motorische Karten im Gehirn. Dabei hängt die Auslösung vom Erregungsort in der entsprechenden Karte ab. Um dies auf Kohonens Modell

zu übertragen, wird jedem Neuron der Neuronenschicht zusätzlich zu dem bereits vorher eingeführten Gewichtsvektor $w_s^{(in)}$ ein Ausgabevektor $w_s^{(out)}$ zugeordnet. Diese Lösung ist einfacher und effizienter zu implementieren als eine zusätzliche Neuronenschicht allein für die Ausgabe der Bewegungssignale, wie sie im Gehirn zu finden ist.

Durch diese Anpassung wird in Kohonens Algorithmus mit jedem Lernschritt neben der Anpassung des Eingabevektors v auch die eines passenden Ausgabewertes u erforderlich. Der Ausgabewert kann als Skalar, lineare Abbildung aber auch als Vektor realisiert werden. Die Adaption der Ausgabewerte $w_s^{(out)}$ erfolgt analog zur Vorgehensweise auf der Eingabeseite. Alle Neuronen in der Umgebung des durch den Eingabevektor selektierten Neurons s lernen nämlich ähnliche Ausgabewerte wie Neuron s selbst. Dies ist ein Resultat der nachbarschafts- und damit stetigkeitserhaltenden räumlichen Anordnung, die auch im Algorithmus entsprechend realisiert ist:

1. Registriere die nächste Steurungsaktion (v, u).
2. Bestimme den Gitterplatz $s := \phi_w(v)$, der dem aktuellen Systemzustand v in der Karte zugeordnet ist.
3. Führe einen Lernschritt
$$\Delta w_r^{(in)} = w_r^{(in)} + \epsilon h_{rs}(v - w_r^{(in)})$$
für die Synapsenstärken $w_r^{(in)}$ aus.
4. Führe einen Lernschritt
$$\Delta w_r^{(out)} = w_r^{(out)} + \epsilon' h'_{rs}(u - w_r^{(out)})$$
für die Belegung mit Ausgabewerten $w_r^{(out)}$ aus und gehe nach 1.

Dieser Algorithmus verwendet „Lernen durch Unterweisung", wobei die Adaption der Ausgabewerte durch Schritt 4 erfolgt. Die Nachbarschaftkooperation wird durch die Funktion h'_{rs} realisiert und die Topologieerhaltung schließlich durch die symmetrische Behandlung von Ein- und Ausgabewerten.

Die Kartenabbildung entspricht nun dem Erlernen einer Steuerungsaufgabe. Im Hinblick auf die Steuerung von Bewegungen stellt Vektor v den gegenwärtigen Zustand des zu steuernden Systems dar, während Vektor u die Steuerreaktion in Form einer Auslenkung, einer Kraft oder eines Drehmoments angibt.

8.2.2 „Stabbalance-Problem" als technisches Beispiel

Versuchsbeschreibung

In diesem Beispiel ist es die Aufgabe des neuronalen Netzes, einen Stab im Schwerefeld der Erde durch geeignete Bewegungen des Fußpunktes vertikal zu balancieren. Dabei liegt das Interesse an der Untersuchung der Lernregeln, die ein möglichst langes Ausbleiben des Stabkippens, als einzigem Bewertungskriterium, bewirken und so jeder Stabneigung eine ganz bestimmte entgegenwirkende Kraft zuordnen. Durch einen Computer wird die Bewegung eines masselosen Stabes simuliert. Die Bewegungen des mit Punktmassen an Kopf- und Fußende versehenen Stabes beschränken sich dabei auf die vertikale Ebene, wobei das Fußende des Stabes nur entlang der horizontalen x-Achse durch eine geeignete Kraft f geführt werden kann.

Simulation mit Unterweisung

In diesem Lernverfahren übernimmt ein Lehrer die Kontrolle des Bewegungsablaufs bis das neuronale Netz diesen selbst ausführen kann. Das Netz wird für diese Simulation aus einem 25x25-Neuronengitter gebildet, wobei jedem Neuron ein zweidimensionaler Vektor $w_r^{(in)}$ und ein Ausgabewert $w_r^{(out)}$ zugeordnet ist. Der Eingabevektor beschreibt aufeinanderfolgende Stabneigungen

gegen die Vertikale, während der Ausgabewert eine diesen Neigungen entgegenwirkende Kraft angibt.

Die Stabsteuerung erfolgt nun durch jeweils neue Kräfte am Fußpunkt des Stabes zu äquidistanten Zeitpunkten $t_n = n\Delta t$. Diese Kräfte sind zwischen t_n und t_{n+1} konstant und werden zu Anfang je nach Stabneigung vom Lehrer vorgegeben. Der Einfluß des Lehrers nimmt jedoch mit fortschreitender Lerndauer zugunsten des neuronalen Netzes ab. In einer Folge von Versuchen wird nun der Stab aus einer zufälligen Anfangsneigung $\Theta \in [-30°, 30°]$ freigegeben und durch Anwendung einer Kraft f_n bis zum Umkippen oder bis zum Ablauf von 100 Zeitschritten balanciert.

Durch die Darstellung von Θ als Netzfläche im $w_1 - w_2 - w^{(out)} - Raum$ läßt sich das Lernverhalten des neuronalen Netzes gut verfolgen. Zu Beginn wird jedem Neuron r ein zufälliges $w_1 - w_2 - Paar$ aus $[-30°, 30°] \times [-30°, 30°]$, sowie eine Kraft $w_r^{(out)} = 0$ zugeordnet. Nach 100 Lernschritten haben bereits die meisten Neuronen ihre Eingabewerte zu Winkelpaaren in der Nähe der Diagonalen $w_1 = w_2$ konzentriert und entgegnwirkende Kräfte grob gelernt. Gegen Ende der Simulation, nach 1000 Zeitschritten, hat sich die Konzentration um die $w_1 - w_2 - Diagonale$ weiter verfeinert. Dies ist auch leicht einzusehen, da sich aufeinanderfolgende Stabneigungen kaum unterscheiden. Besonders um $(0°, 0°)$ verdichten sich die Zuordnungen der Winkelpaare, weil diese Stabneigungen mit zunehmender Lernzeit am häufigsten auftreten. Bereits nach 200 Zeitschritten kann das Netz eine Stabneigung von 20° gegen die Vertikale ausbalancieren, und nach 1000 Schritten wird sogar eine Neigung von 30° bewältigt.

Simulation ohne Unterweisung

Anstatt eines Lehrers gibt nun eine Bewertungsfunktion an, wie gut der erreichte Zustand die Aufgabenstellung erfüllt. Das Lernziel besteht in der Auswahl einer Kraft, die einen möglichst großen Zuwachs der Bewertungsfunktion R in einem Zeitschritt erzielt. Nur wenn der Bewertungszuwachs ΔR durch f_n den bisherigen durchschnittlichen Zuwachs übertrifft, wird f_n als Ausgabewert für s gespeichert. Das heißt, daß das neuronale Netz nur aus Aktionen lernt, die seine bisherige Leistung übertreffen, sodaß eine laufende Verbesserung der Steuerung eintritt.

Für die Simulation wird der Anfangszustand identisch zur Stabbalance mit Unterweisung gewählt. In 3000 Lernschritten ordnen sich die Werte wieder entlang der $w_1 - w_2 - Diagonalen$ und auch besonders um $(0°, 0°)$ an. Auch die gelernten Kräfte verhalten sich ähnlich wie im Falle mit Unterweisung. Sie steigen jedoch für Auslenkungen von der Vertikalen stärker an und führen so zu einer rascheren Ausregelung von Abweichungen. Zusätzlich können sogar Anfangsneigungen von 40° gegen die Vertikale ausbalanciert werden.

8.2.3 Okulomotorik als biologisches Beispiel

Biologischer Hintergrund

Erregt ein Objekt im Blickfeld des Auges die Aufmerksamkeit eines Betrachters, so wird dieses Objekt durch sakkadische Augenbewegungen ins Zentrum der Netzhaut gebracht. Dieser Bereich enhält sehr viel mehr lichtempfindliche Zellen als die übrige Netzhaut und wird „Fovea“ genannt. Er ist dadurch in der Lage einen betrachteten Gegenstand sehr viel höher aufzulösen. Sakkaden werden im „Superior Colliculus“ durch eine mehrlagige Neuronenschicht ausgelöst. Die oberen Lagen, die sogenannten sensorischen Karten, erhalten ihre Eingaben von der Retina und dem visuellen Kortex. Die Verbindungen zwischen Retina und visuellem Kortex sind dabei topographisch geordnet, das heißt, daß eine stetige Zuordnung zwischen den Lichtrezeptoren der Retina und der Lage erregter Neuronen im Superior Colliculus besteht. Darunterliegende Schichten bilden eine motorische Karte, deren Neuronen bestimmten sakkadischen Augenbewegugen zugeordnet sind und von entsprechenden Neuronen der sensorischen Karte erregt werden. Wird nun durch einen Lichtreiz auf der Retina ein Erregungszentrum auf der visuellen Karte lokalisiert, so wird dies auf unmittelbar darunterliegende Neuronen der motorischen Karte übertragen und der Lichtreiz in die Fovea bewegt.

Das okulomotorische System ist so eingerichtet, daß nachträgliche Adaptionen der sensorischen und motorischen Karten jederzeit möglich sind. Dadurch können Änderungen des Zusammenhangs zwischen visuellem Kortex und notwendiger Sakkade neu gelernt werden. Bestätigt wird dies durch folgendes Experiment: Versuchspersonen wurde eine Kontaktlinsen-Brillen-Kombination verabreicht, deren Sehschärfekorrekturen sich gerade aufhoben. Diese Kombination wirkte jedoch als „Galileisches Fernrohr", so daß die Sakkaden ihr Ziel knapp verfehlten und das Auge zu Korrekturbewegungen veranlaßten. Nach 9-14 Minuten hatte sich jedoch das okulomotorische System auf diese Situation eingestellt und die Sakkaden trafen wieder direkt ins Ziel.

Im verwendeten Modell wird jedem Neuron r ein rezeptives Feld mit Zentrum am Ort w_r der Retina, sowie ein Vektor $w_r^{(out)}$ zugeordnet. Anstatt die benötigten Informationen also auf Neuronen verschiedener Karten zu verteilen, wie es im Gehirn realisiert ist, werden sensorische und motorische Karten zu einem Gitter zusammengefaßt. Auch die Erregung einer ganzen Neuronengruppe durch einen visuellen Reiz wird auf die Auswahl lediglich eines im Zentrum lokalisierten Neurons beschränkt.

Das Lernverfahren

Um den Aufwand des Lernverfahrens zu reduzieren, beschränkt man sich im Modell auf die Ausführung nur einer Korrektursakkade. Sie geht jedoch nur dann in nachfolgende Sakkaden ein, wenn sie eine Verbesserung der Situation bewirkt. Die Auswahl neuer Objekte auf der Retina erfogt zufällig, entsprechend einer Wahrscheinlichkeitsdichte, die der Rezeptordichte auf der Netzhaut entspricht. Diese nimmt in Richtung Zentrum stark zu, wobei der Foveabereich selbst ausgeklammert bleibt, da hier keine Sakkaden mehr nötig sind. Wird nun ein Objekt an der Stelle v auf der Retina ausgewählt, so wird im Gitter das Neuron aktiviert, für das $w_s - v$ am kleinsten ist. Zusätzlich erfolgt aus Konvergenzgründen eine Adaption aller Neuronen r in der Umgebung von s. Neu ist auch hier die Erweiterung von Kohonens Modell um Ausgabewerte. Es handelt sich um zweidimensionale Vektoren, die eine Verschiebung eines Objektpunktes auf der Retina verursachen und nach Abschluß des Lernverfahrens alle in der Fovea enden müssen. Der hier benutzte Algorithmus verwendet „Lernen ohne Unterweisung". Dabei werden korrekte Steueraktionen durch einen Suchprozeß im Raum aller möglichen Werte bestimmt. Im verwendeten Modell geschieht dies durch Korrektursakkaden, wobei eine Bewertungsfunktion die Güte der Steuerung durch das Kriterium „Fovea nähergekommen" beurteilt.

Simulation und Vergleich zu Messungen an Versuchspersonen

Zur Simulation wird ein ringförmig geschlossenes 25x30-Neuronengitter A verwendet. Die Darstellung des Lernfortschritts erfolgt durch Belegung von A mit rezeptiven Feldern, also der Zuordnung von Eingabewerten zu bestimmten Retinaorten und den resultierenden Sakkaden. Da das Lernen der Sakkaden ein möglichst stabiles Netz bezüglich der Eingabewerte benötigt, ist der Lernalgorithmus so organisiert, daß das Netz schon zu einem sehr frühen Zeitpunkt im Raum der Eingabevektoren eingefroren wird. Denn durch eine nachträgliche Verschiebung des rezeptiven Feldes würden bereits gelernte Sakkaden wieder ungültig, und der Lernprozeß müßte von neuem beginnen.

Die Darstellung des Netzes erfolgt in einer gedachten Projektion auf die Retina, wobei Neuronennachbarschaften durch Linien angegeben werden. Sakkaden sind durch Pfeile vom Gitterpunkt ihres Ursprungs zum Ort der Verschiebung des Objekts wiedergegeben. Zu Anfang des Lernprozesses sind alle Neuronen mit zufälligen Synapsenstärken w_r und Ausgabevektoren $w_r^{(out)}$ versehen. Bereits nach 20000 Lernschritten ergibt sich eine regelmäßige Zuordnung zwischen Netzhaut und Gitterpunkten; zusätzlich tendieren alle Sakkaden in Richtung Fovea. Gegen Ende der Lernphase hat sich das Netz entsprechend der Eingangsreizverteilung $P(v)$ ausgebildet, und die meisten Neuronen konzentrieren ihre Synapsenstärken auf einen Bereich um die Fovea, während das Gitter nach außen hin dünn besetzt ist. Alle Augenbewegungen führen nun ins Zentrum der Netzhaut. Interessant ist, daß schon nach 10% aller Lernschritte sämtliche Sakkaden in Richtung Fovea zeigen, so daß keine Nachbarschaftskooperation mehr zwischen den Neuronen benötigt wird. Es stellt sich

also die Frage ob überhaupt eine Kooperation sinnvoll ist. Simulationen zeigen jedoch, daß sich ohne übergreifende Adaption eine schlechtere Gesamtkonvergenz einstellt und einzelne Sakkaden am Rand der Retina eine falsche Ausrichtung erhalten, weil nach einer zufälligen Anfangsbelegung keine geeignete Korrektursakkade gefunden wird.

Im Unterschied zu experimentellen Beobachtungen an Versuchspersonen lernt das Neuronengitter in unserem Modell die entsprechenden Sakkaden „zu gut", denn Augenbewegungen beim Menschen führen ein Objekt in den seltensten Fällen direkt in die Fovea, sondern sind zu kurz ausgelegt. Sie werden deshalb als „Undershoot" bezeichnet. Man nimmt an, daß dies Vorteile bei Vorausplanungen, also eventuell notwendigen Korrekturbewegungen hat, die beim Verfolgen bewegter Objekte auftreten. Jedoch reicht das Ergebnis der Simulation zum einfachen Fixieren unbeweglicher Ziele vollkommen aus.

8.3 Anwendungen in der Robotik

8.3.1 Probleme der Robotersteuerung

An Bewegungen des Menschen sind eine Vielzahl von Muskeln beteiligt. So stehen im Arm allein 52 Muskelpaare zur Verfügung. Ihre Steuerung erfolgt über Rückmeldungen der verschiedensten Sensoren in Haut und Muskeln, sowie über das visuelle System.

Die Komplexität der Armsteuerung soll durch Aufnehmen und Eindrehen einer Schraube mithilfe eines Roboters demonstriert werden. Um die Schraube greifen zu können, muß der Roboter zunächst ihre Lage erkennen und eine geeignete Griffposition auswählen. Dies ist abhängig von der Form des Werkstücks, seiner Lage und anderen Objekten, die das Aufnehmen möglicherweise behindern könnten. Zusätzlich spielt der Verwendungszweck eine Rolle, denn ein einfacher Transport der Schraube verlangt weniger Präzision als das Eindrehen. Dann muß eine geeignete Bewegungstrajektorie zum Ziel festgelegt werden. Es sind Berechnungen der Gelenkwinkel durchzuführen und, bei schnellen Bewegungen, entstehende Drehmomente zu beachten, um Grenzdaten einzuhalten. Am Ziel muß die Schraube schließlich exakt positioniert werden, um das Einschrauben zu ermöglichen. Ähnlich wie beim Menschen ist eine Steuerung erforderlich, die sich nach einem ungefähren Aufsetzen der Schraube von den Gegenkräften des Gewindes leiten läßt.

Man verwendet heute in der Regel zwei Ansätze zur Lösung solcher Probleme. Der erste beschäftigt sich im Rahmen der KI mit Programmen, die versuchen, möglichst viele potentielle Situationen im voraus abzufangen. Das Problem hierbei liegt im Auffinden von Heuristiken, die für viele oder alle in der Praxis vorkommenden Situationen funktionieren. Der zweite Ansatz verwendet neuronale Algorithmen und versucht Formen der Bewegungssteuerung biologischer Organismen zu kopieren. Im Mittelpunkt steht hier die „sensomotorische Koordination", also eine Verknüpfung von sensorischer Wahrnehmung und motorischer Steuerung, die sich erst in Lernprozessen herausbildet. Im Folgenden soll dies durch die Anwendung neuronaler Netze in der Robotersteuerung demonstriert werden. Die Eingabedaten zweier Kameras dienen dabei als sensorische Signale der Bewegungskoordination. In der ersten Simulation wird ausschließlich die Kinematik, also die Stellung des Arms in Abhängigkeit von den Gelenkwinkeln berücksichtigt, während in der zweiten auch Trägheitseffekte beachtet werden, die bei schnellen Bewegungen auftreten.

8.3.2 Visuomotorische Koordination eines Roboterarms

Zunächst geht es also um die Positionierung des Manipulators an einem beliebigen Ort im Arbeitsbereich. Diese Aufgabe kann als Voraussetzung für das Greifen von Objekten, das Setzen von Schweißpunkten oder das Umgehen von Hindernissen angesehen werden. Der Aufbau des Roboters ist in Abb. 8.1 dargestellt. Er besteht aus einem Arm mit 3 Freiheitsgraden, der Drehungen um die eigene Achse und Bewegungen in der Vertikalen über zwei Gelenke zuläßt. Informationen über die Position ausgewählter Ziele erhält er über zwei Kameras, die ihm ein räumliches Bild vermitteln. Der Arbeitsbereich vor dem Roboterarm wird durch ein dreidimensionales neuronales Netz nachgebildet. Am Algorithmus selbst sind kaum Änderungen nötig. Die Gittervektoren sind nun

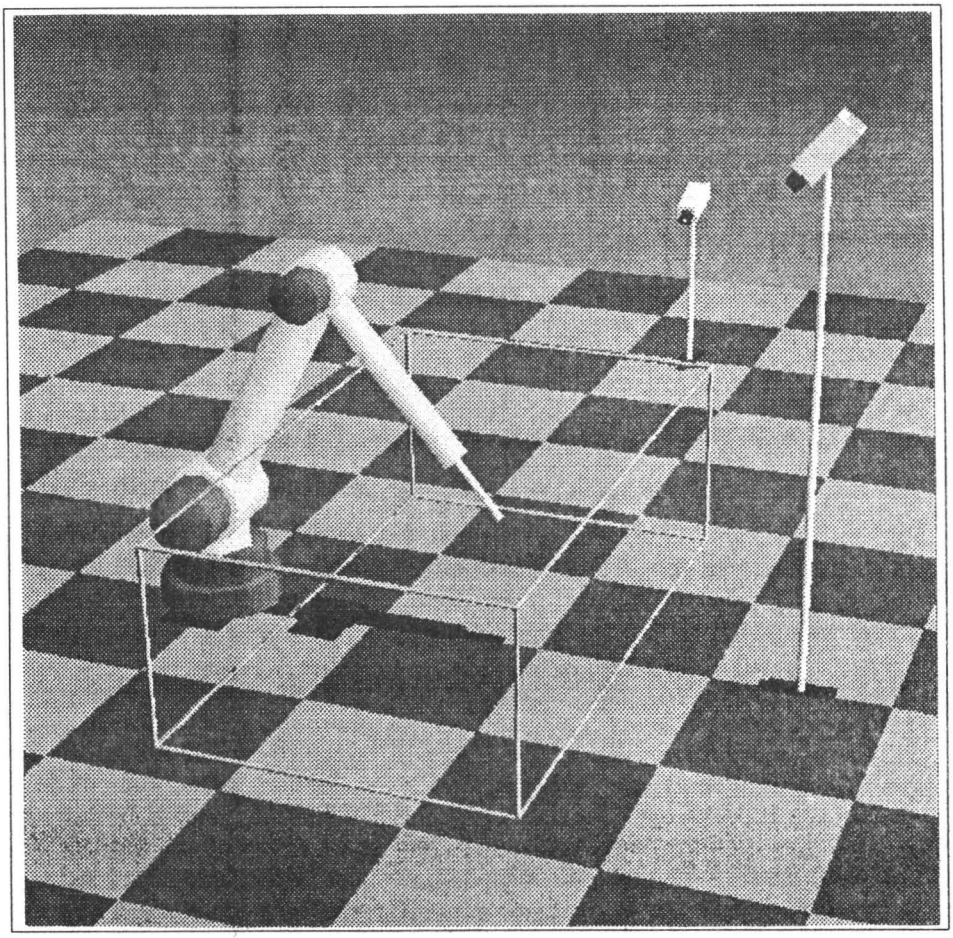


Abbildung 8.1: Simulationsmodell des Robotersystems

dreidimensional und der Abstand zum jeweiligen Erregungszentrum muß in 3 Ebenen betrachtet werden.

In der Trainingsphase werden Zielpositionen im Arbeitsbereich zufällig ausgewählt. Sie werden durch die beiden Kameras erfaßt und als entsprechende Signale an das Neuronengitter weitergegeben. Dadurch wird das Neuron s aktiviert, das für die Raumposition des Ziels verantwortlich ist, und dieses gibt die benötigten Steuersignale an die Gelenkmotoren weiter. Die Informationen der beiden Kameras werden jedoch nicht zu einem dreidimensionalen Vektor verarbeitet, sondern dem Netz als zwei unabhängige zweidimensionale Verktoren bereitgestellt. Das Netz muß diese Daten nun geeignet interpretieren, um entsprechende Ausgaben für die Armsteuerung liefern zu können. Dem Netz werden auch keine Informationen über die Abmessungen der Armsegmente oder die Stellung der Kameras gegeben. Diese muß das Gitter während des Lernprozesses selbst herausfinden, um eine korrekte Abbildung der Kamerainformationen auf die Motorsignale zu gewährleisten. Dadurch ergibt sich anfangs eine fehlerhafte Positionierung, die jedoch durch die Kameras registriert wird und schließlich die Ausgabewerte verbessert. Danach wird ein neues Ziel vorgegeben und der nächste Lernschritt durchlaufen. Der Roboter ist also ein autonomes System, das ohne Lehrer auskommt.

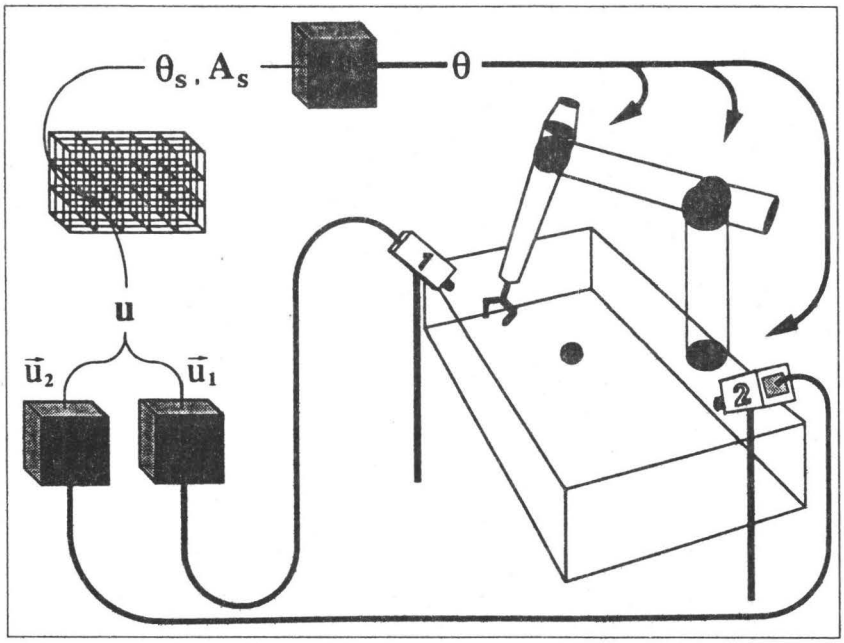


Abbildung 8.2: Schematische Darstellung des Positioniervorgangs

Positioniervorgang und Lernverfahren

Die Vorgabe eines Zielpunktes erzeugt auf jeder der beiden Kameras einen Bildpunkt auf der Bildebene. Diese werden jeweils durch zweidimensionale Vektoren $\vec{u_1}$, $\vec{u_2}$ dargestellt und zu einem vierdimensionalen Vektor $u = (\vec{u_1}, \vec{u_1})$ zusammengefaßt, der an das Kohonen-Gitter weitergeleitet wird. Zur korrekten Steuerung des Manipulators muß das Robotersystem die Eingabedaten in entsprechende Gelenkwinkel $\vec{\Theta} = \vec{\Theta}(u) = (\Theta_1, \Theta_2, \Theta_3)$ transformieren. Das heißt, Neuron s, das für den Raumbereich zuständig ist, in dem sich das Zielobjekt gerade befindet, wird aktiviert, falls $||w_s - u|| \leq ||w_r - u||$ für alle $r \neq s$, und gibt die passenden Gelenkwinkel $\vec{\Theta} = (\Theta_1, \Theta_2, \Theta_3)$ aus. Neben den Winkeln $\vec{\Theta_s}$ muß jedes Neuron auch die Matrix A_s als Ausgabegröße lernen. Sie dient durch $\vec{\Theta} = \vec{\Theta_s} + A_s(u - w_s)$ der Korrektur der Armwinkel, falls die durch $\vec{\Theta}$ erreichte Position von der gewünschten Zielposition abweicht. Der Positioniervorgang ist in Abb. 8.2 schematisch dargestellt.

Wie bereits erwähnt erhält das neuronale Netz keine Angaben über die Position der Kameras oder die Armabmessungen, sondern muß die entsprechenden Daten während des Lernprozesses selbst herausfinden. Nur so bleibt das System lernfähig und kann nachträgliche Veränderungen durch Verschleiß oder Veränderung der Kameraposition adaptieren. Während des Lernprozesses muß also der Unterraum $U = U_1 \otimes U_2$ herausgefunden werden, um die zur Repräsentation benötigten Stützstellen möglichst gut zu verteilen. Zu Beginn werden die Netzknoten im vierdimensionalen Eingangssignalraum U willkürlich verteilt. Da jedoch alle Eingangssignale aus dem dreidimensionalen Unterraum W stammen, ordnen sich die Elemente nur in diesem Teilbereich von U an und optimieren die Ausnutzung der vorhandenen Stützstellen. Die Dichte der Netzknoten wird also der Wahrscheinlichkeitsdichte der Eingangssignale angepaßt.

Der Lernprozeß für die Ausgabewerte basiert auf der Auswertung von Positionierfehlern. Wird ein Neuron durch die Zielvorgabe u ausgewählt, so erfolgt die Positionierung zunächst mithilfe von $\vec{\Theta_s}$ und A_s in eine Zwischenposition $\vec{\Theta_i} = \vec{\Theta_s} + A_s(u - w_s)$. Die sich dabei ergebenden Bildpunktkoordinaten v_i des Manipulators werden von den Kameras registriert und für eine Korrekturbewegung genutzt. Stimmt dann die Manipulatorposition schon recht gut mit der Objektposition überein, so ergibt sich die Korrekturbewegung zu $\Delta\vec{\Theta} = A_s(u - v_i)$. Theoretisch ist die

Anzahl der Korrekturschritte nicht begrenzt. Im verwendeten Modell beschränken wir uns jedoch auf eine Nachsteuerung. Die endgültige Manipulatorposition v_f wird ebenfalls von den Kameras registriert und geht zusammen mit v_i in den nächsten Lernschritt ein. Der Lernalgorithmus für das Kohonen-Netz und die Ausgabegrößen sieht dann folgendermaßen aus:

1. Gebe einen zufällig gewählten Zielpunkt im Arbeitsbereich vor.

2. Registriere mittels der Kameras das zugehörige Eingangssignal u.

3. Bestimme den Gitterplatz $s := \phi_w(u)$, der u im Gitter zugeordnet ist.

4. Führe einen Lernschritt für die Stützstellen w_r durch. Dieser lautet

$$w_r^{(neu)} = w_r^{(alt)} + \epsilon h_{rs}(u - w_r^{(alt)})$$

5. Bringe den Manipulator in eine Zwischenposition durch Einstellen der Gelenkwinkel gemäß

$$\vec{\Theta_i} = \vec{\Theta_s} + A_s(u - w_s)$$

   und registriere die Manipulatorkoordinaten v_i auf den Kamerabildebenen.

6. Führe eine Korrektur der sich durch 5. ergebenden Manipulatorposition mittels

$$\vec{\Theta_f} = \vec{\Theta_s} + A_s(u - v_i)$$

   durch und registriere wiederum mithilfe der Kameras seine Endposition v_f.

7. Bestimme die verbesserten Schätzwerte $\vec{\Theta^*}$ und A^* mittels

$$\vec{\Theta^*} = \vec{\Theta_s^{alt}} + \delta_1 A_s^{alt}(u - v_i)$$

$$A^* = A_s^{alt} + \delta_2 A_s^{alt}(u - v_f)(v_f - v_i)^T$$

8. Führe einen Lernschritt für die Ausgabegrößen von Neuron s sowie seiner Nachbarn r mittels

$$\vec{\Theta_r^{neu}} = \vec{\Theta_r^{alt}} + \epsilon' h'_{rs}(\vec{\Theta^*} - \vec{\Theta_r^{alt}})$$

$$A_r^{neu} = A_r^{alt} + \epsilon' h'_{rs}(A^* - A_r^{alt})$$

   durch und fahre mit Schritt 1 fort.

Für die Berechnung von $\vec{\Theta^*}$ und A^* in Schritt 7 wird ein Gradientenabstiegsverfahren auf einer quadratischen Fehlerfunktion als Lernalgorithmus benutzt. Dabei geben δ_1 und δ_2 die Lernschrittweiten an. Diese Größen werden dann zur Verbesserung der Ausgabewerte von Neuron s und seiner Nachbarn benutzt. Die Nachbarschaftskooperation in Schritt 8 ist auch hier sehr wichtig für die Konvergenz des Verfahrens, denn sie verhindert, daß die Ausgabewerte einiger Neuronen in sogenannten Nebenminima steckenbleiben, wie die folgende Situation zeigen wird.

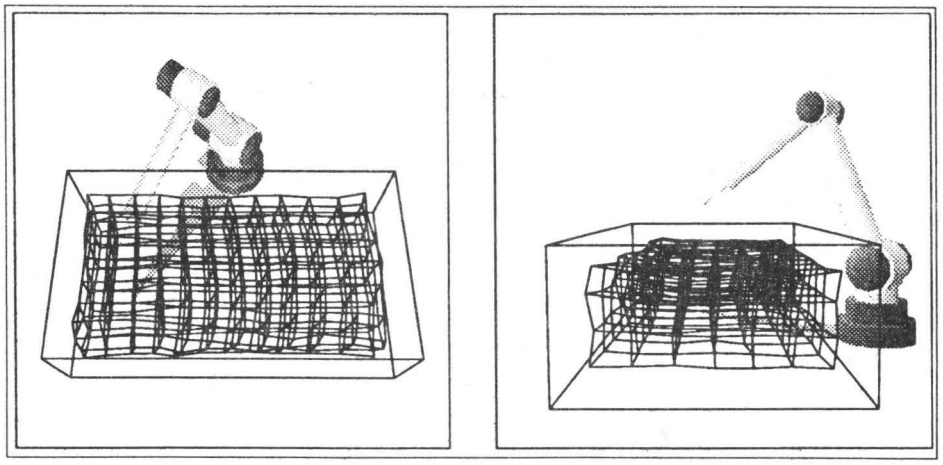


Abbildung 8.3: Kohonen-Netz nach 6000 Lernschritten

Simulation

Zur Simulation wird ein 7x12x4-Gitter mit 336 Neuronen verwendet. Die Darstellung des Kohonon-Netzes erfolgt durch eine Projektion der Netzknoten auf die Bildebenen der beiden Kameras. Ist also $w_r = (\vec{w_{r_1}}, \vec{w_{r_2}})$ der Ortsvektor von Neuron r, so wird r von Kamera 1 an Position w_{r_1} und von Kamera 2 an Position w_{r_2} dargestellt.

Zu Anfang des Lernprozesses wird jedem Neuron r eine Zufallsposition mit gleichverteilten Wahrscheinlichkeiten auf den Kamerabildebenen zugeordnet. Daraus ergibt sich eine Gleichverteilung der Gitterpunkte inner- und außerhalb des Arbeitsbereichs, so daß keine regelmäßige Struktur erkennbar ist. Bereits nach 2000 Lernschritten hat sich das Gitter im Arbeitsbereich ausgedehnt und repräsentiert nur noch den relevanten Unterraum. Die Situation nach 6000 Lernschritten ist in Abb. 8.3 dargestellt. Die Positionierfehler erreichen jetzt ihren Minimalwert, so daß alle Zielpositionen im Arbeitsbereich gleichmäßig durch das Gitter repräsentiert werden. Die notwendige Verzerrung des Netzes aus Sicht der jeweiligen Kamera ergibt sich dabei automatisch durch die Eingangsreizdichte.

Der Lernfortschritt der Ausgabegrößen wird ebenfalls auf den Bildebenen der beiden Kameras getrennt dargestellt. Dabei wird die Position des Manipulators im Arbeitsbereich durch Kreuze markiert und die Abweichungen von den Netzknoten durch Verbindungslinien angezeigt. Diese ergeben sich aufgrund des von $\vec{\Theta_r}$ verursachten Positionierfehlers bei Ansteuerung der Zielpunkte $u = (\vec{w_{r_1}}, \vec{w_{r_2}})$. Am Ende sollten jedoch alle Kreuze die Vektoren $\vec{w_{r_1}}$ und $\vec{w_{r_2}}$ der rezeptiven Felder der Netzknoten markieren. Die Adaption der Matrizen A_r wird durch kurze Testbewegungen entlang der drei Kanten des Arbeitsbereichs demonstriert, wobei die Ausgangsposition mit $\vec{\Theta_r}$ vorgegeben ist.

Zu Beginn des Adaptionsprozesses sind aufgrund zufällig eingestellter Gelenkwinkel $\vec{\Theta_r}$ alle Kreuzsymbole inner- und außerhalb des Arbeitsbereichs verteilt. Auch die Testbewegungen für die Matrizen A_r zeigen keine regelmäßigen Strukturen auf. Nach 2000 Lernschritten haben sich jedoch die Zwischenpositionen bei Einstellung der Gelenkwinkel auf den Arbeitsbereich konzentriert. Die Dreibeine zur Demonstration von A_r haben nun annähernd rechtwinklige Gestalt, auch wenn ihre Amplituden noch zu kurz sind. Nach 6000 Versuchen haben $\vec{\Theta_r}$ und A_r ihre Optimalwerte erreicht. Die Kreuze markieren nun die durch die Ortsvektoren w_{r_i} repräsentierten Netzknoten und die Dreibeine stimmen in Ausrichtung und Amplitude mit den gewünschten Werten überein.

Abb. 8.4 zeigt den Lernerfolg aus einer anderen Sicht. Hierzu wird die mittlere Abweichung des Manipulators vom Zielpunkt in Abhängigkeit von der Anzahl der Versuchsbewegungen in einem Koordinatensystem dargestellt. Es ist erkennbar, daß ohne Nachbarschaftskooperation kei-

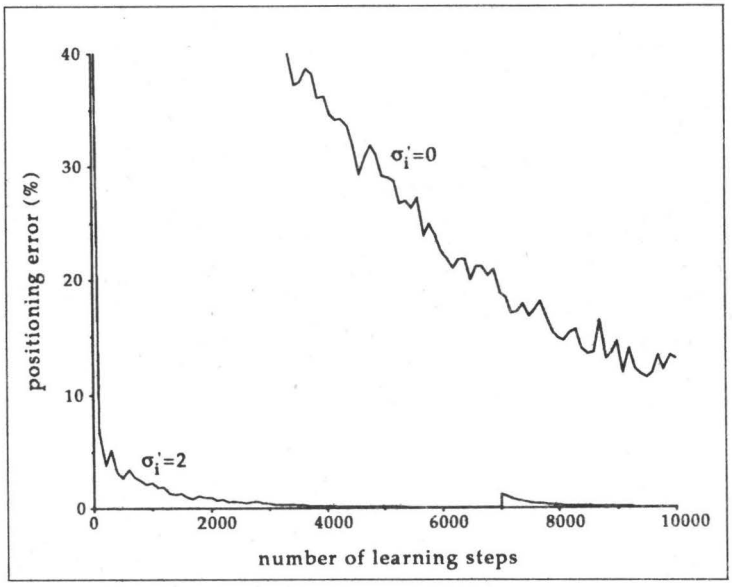


Abbildung 8.4: Der Positionierungsfehler in Abhängigkeit von der Anzahl der Versuchsbewegungen

ne Konvergenz eintritt und der Lernprozeß nur sehr langsam voranschreitet, so daß auch nach 10000 Versuchen noch Positionierungsfehler auftreten. Diese nehmen mit Nachbarschaftskooperation hingegen sehr schnell ab und erreichen bereits nach 100 Lernschritten einen Fehler von 0,0038 Längeneinheiten. Dies entspricht etwa 8% der Abmessungen des Roboterarms. Nach 6000 Lernschritten stellt sich ein Restfehler von 0,0004 Längeneinheiten ein, der etwa 0,08% der Roboterabmessungen entspricht. Dies bedeutet, daß Zielpositionen nun auf Zehntelmilimeter genau angefahren werden können.

Zum Schluß soll die Adaptionsfähigkeit des neuronalen Netzes gezeigt werden. Dazu wird das dritte Armsegment, an dem der Manipulator befestigt ist, um 0,05 Längeneinheiten, also 10%, verlängert. Danach erhöht sich zunächst der Positionierfehler, da sich das Netz zunächst auf die veränderte Situation einstellen muß, ist jedoch mit nur 1,2% um einen Faktor 10 geringer als erwartet. Der Grund hiefür liegt in der Korrekturbewegung. Durch sie wird der größte Teil der Abweichung von der Zwischenposition v_i zum Sollwert u bereits ausgeglichen. Von der Verlängerung sind hauptsächlich die Ausgabewerte $\vec{\Theta_r}$ betroffen. Sie müssen neu gelernt werden, während die A_r fast unverändert übernommen werden können und so zur Korrektur weiter geeignet sind. Nach weiteren Lernschritten hat das Netz schließlich die veränderten Abmessungen adaptiert und erreicht wieder die geringen Restfehlerwerte. Vorteilhaft wirkt sich diese Adaptionsfähigkeit bei Verwendung unterschiedlicher Manipulatoren oder Benutzung verschiedener Werkzeuge mit dem Greifer aus. Durch die hohe Flexibilität kann der Roboter für unterschiedliche Aufgabenstellungen ohne zusätzliche Trainingsphasen eingesetzt werden.

Erweiterung des Modells durch einen Roboterarm mit mehreren Freiheitsgraden und Ergebnisse einer Simulation

Die bisher betrachteten Roboter konnten einen Zielpunkt im dreidimensionalen Arbeitsbereich nur mit eindeutigen Gelenkwinkelkonfigurationen erreichen. Nun gibt es in der Natur aber oft Bewegungsapparate, die über redundante Freiheitsgrade verfügen. Betrachtet man den menschlichen Arm, so fällt auf, daß er 4 Freiheitsgrade, nämlich 3 in der Schulter und einen im Ellenbogen, besitzt. Dadurch können Objekte mit verschiedenen Armstellungen erreicht werden, das heißt, die Gelenkwinkelkonfigurationen sind nicht mehr eindeutig vorgegeben. Sinnvoll sind zusätzliche

Freiheitsgrade, falls Hindernisse den Bewegungsablauf beeinflussen oder nur bestimmte Griffpositionen zulassen. Solches wird jedoch in unserem Modell ausgeschlossen. Wie im vorangegangenen Simulationsbeispiel geht es hier ausschließlich um die Manipulatorsteuerung im „hindernisfreien Raum", jedoch mit zwei zusätzlichen Freiheitsgraden des Roboterarms. Es wird, bis auf kleine Anpassungen, der gleiche Lernalgorithmus wie im Falle des dreigelenkigen Roboters verwendet.

In der Praxis übliche Verfahren zur Lösung dieses Steuerungsproblems verwenden Gelenkwinkelstellungen, die zum Ziel führen, zusätzlich jedoch eine Kostenfunktion minimieren. Solches ist auch beim Menschen zu beobachten. Es werden Armstellungen bevorzugt, die die Gelenkwinkel mittlere Werte ihres gesamten Wertebereichs annehmen lassen. Dagegen wird völliges Strecken oder Anwinkeln der Gelenke vermieden.Überträgt man dies auf unseren Algorithmus, so muß die entsprechende Kostenfunktion ebenfalls extreme Gelenkwinkelstellungen vermeiden und mittlere Werte bevorzugen. Ein zweiter Ansatz unterstützt möglichst „faule Armbewegungen". Dadurch werden geringe Gelenkwinkeländerungen beim Anfahren einer Zielposition bevorzugt, da sie Verschleiß und Energieaufwand minimieren.

Die Forderungen in der Robotik konzentrieren sich nun aber auf möglichst glatte Bewegungen des Roboterarms, das heißt, benachbarten Zielpositionen soll eine ähnliche Armstellung zugeordnet sein. Dadurch erhält man beim Nachfahren einer Linie im Arbeitsbereich eine dynamische Bewegung des Roboterarms. Diese Eigenschaften sind aber gerade Kennzeichen des Kohonen-Algorithmus, denn eine topographische Karte zeichnet sich dadurch aus, daß benachbarten Positionen im Arbeitsbereich benachbarte Neuronen zugeordnet sind, die ähnliche Ausgabewerte liefern. Damit entfällt die Kostenfunktion bei Verwendung von Kohonen-Algorithmen.

Im simulierten Modell wird nun ein Arm mit 5 Gelenken, also 2 zusätzlichen Freiheitsgraden verwendet. Es sind Drehungen um die vertikale Achse möglich, während die übrigen Gelenke parallel zueinander und zur horizontalen Ebene stehen. Wie in der vorangegangenen Simulation wird wieder ein 7x12x4-Gitter verwendet; Simulationsparameter, Kamerastellungen und Arbeitsbereich bleiben gleich, lediglich $\vec{\Theta}_r$ und A_r müssen an die zwei zusätzlichen Gelenke angepaßt werden.

Zur Darstellung des Lernfortschritts sind die Positionierfehler gegen die Anzahl der Lernschritte in einem Koordinatensystem abgetragen. Man erkennt, daß der Fehler von Beginn an sehr schnell abnimmt. Bereits nach 200 Lernschritten ergibt sich ein Wert von 0,03 Längeneinheiten, was 6% der Roboterabmessungen entspricht. Nach 6000 Adaptionsschritten hat sich ein Restfehler von 0,0004 Längeneinheiten oder 8% des Roboterarms eingestellt. Erstaunlich ist, daß sich die gleichen Werte wie beim dreigelenkigen Roboter ergeben. Nach 7000 Lernschritten wird, wie in der vorangegangenen Simulation, das letzte Armsegment um 10% verlängert. Man will hiermit die Adaptionsfähigkeit des Systems testen. Auch in dieser Simulation stellt sich der Roboter rasch auf die veränderte Situation ein und erreicht wieder den Grenzwert von 0,0004 Längeneinheiten.

Um zu überprüfen, ob der Roboter auch wirklich in der Lage ist, glatte Bewegungen zu vollziehen, erhält er nach Abschluß der Lernphase die Aufgabe, mit dem Manipulator eine Diagonale im Arbeitsbereich nachzufahren. Wie Abb. 8.5 zeigt, bewegt sich der Arm dabei dynamisch zu den Punkten der Trajektorie. Unter den möglichen Armstellungen wird also jeweils diejenige ausgewählt, die nur eine geringe Änderung der Gelenkwinkel zur Folge hat. Deswegen ist keine Kostenfunktion notwendig, sondern allein das Prinzip der topologieerhaltenden Abbildung zwischen Eingangssignalraum, Neuronengitter und Ausgabegrößen reicht aus.

8.3.3 Manipulatorsteuerung mit hierarchischen Netzen

Änderung und Erweiterung des Robotermodells

Die Aufgabe, die im Folgenden behandelt werden soll, befaßt sich mit der Steuerung koordinierter Greifbewegungen für einfache Objekte. Dabei ist zu berücksichtigen, wo sich das Objekt im Arbeitsbereich befindet und ob es vom Greifer umfaßt werden kann. Zusätzlich spielen seine Gestalt, sein Schwerpunkt und seine Orientierung eine Rolle. Im Modell werden jedoch einige Annahmen gemacht, die die Komplexität der Aufgabe reduzieren. Wie Abb. 8.6 zeigt, wird als Manipulator ein Parallelzangengreifer verwendet. Dabei muß der Durchmesser des Zylinders kleiner sein als die Spannweite des Greifers. Der Schwerpunkt des Objekts wird idealerweise vernachlässigt, um es

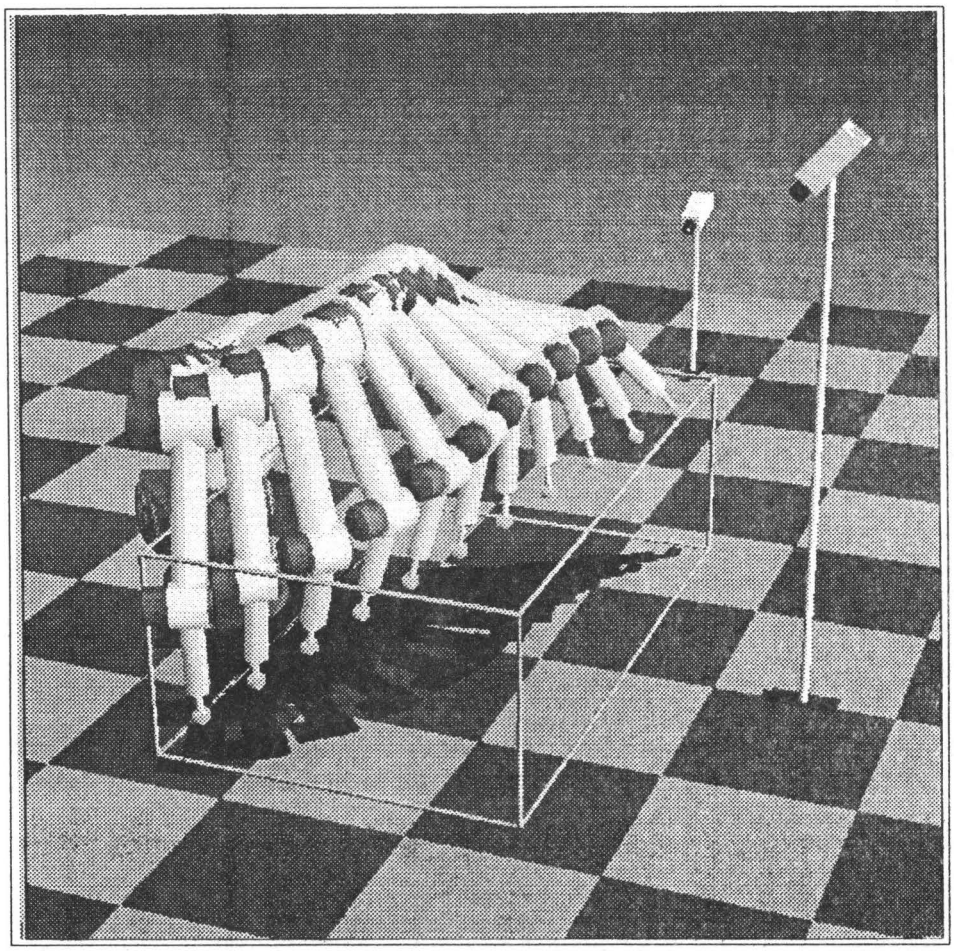


Abbildung 8.5: Stoboskopaufnahme einer Bewegung des Roboterarms.

immer in seinem Zentrum anfahren zu können.

In Abb. 8.7 ist das simulierte Robotermodell dargestellt. Die Position der Kameras und der Arbeitsbereich sind dabei identisch zu den bereits vorgestellten Simulationen. Der Roboter wird nun in seiner Haltung dem menschlichen Arm nachgebildet, um Objekte von „vorne" ansteuern zu können. Damit versucht man, den Bewegungsablauf des Menschen zu imitieren, der Gegenstände auch meist von „vorne" greift, falls keine Hindernisse den Weg versperren. Der Roboterarm besitzt nun wieder 3 Freiheitsgrade, die Drehungen um die vertikale Achse erlauben, während die beiden anderen Gelenke parallel zueinander und zur horizontalen Ebene stehen. Die Orientierung des Greifers kann mittels zweier Freiheitsgrade verändert werden, wobei die Achse des ersten Gelenks parallel zu den beiden horizontalen Armgelenken steht. Das zweite Gelenk kann den Manipulator um seine Symmetrieachse drehen. Abb. 8.6 zeigt den Greifer mit den Gelenkwinkeln β_1, β_2 und dem Normalenvektor $\vec{n}$, der seine Orientierung bestimmt. Zusätzlich ist der Punkt P als Zentrum des Greifers eingezeichnet. Bei Greifbewegungen muß P ins Zentrum des Zylinders geführt werden, während $\vec{n}$ parallel zu seiner Achse stehen muß.

Die Aufgabe des neuronalen Netzes besteht nun wieder darin, die Eingabedaten der Kameras zu entsprechenden Gelenkwinkelkonfigurationen zu verarbeiten. Deswegen müssen Informationen über die Position und die Orientierung des Zylinders im Arbeitsbereich gewonnen werden. Die Form, in der diese Informationen dem Netz zur Verfügung gestellt werden, ist in Abb. 8.8 erklärt:

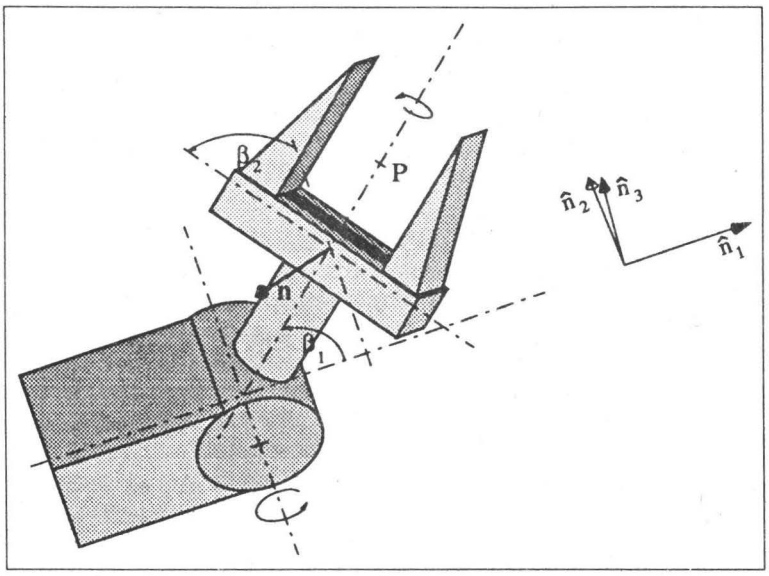


Abbildung 8.6: Eine Skizze des Greifers

Der Zylinder wird als Balken in die Bildebenen der beiden Kameras projeziert. Dabei liefert der Ort des Balkenzentrums die Raumposition des Zylinders, während seine Orientierung die Orientierung der Zylinderachse wiedergibt. Der Ort des Balkenzentrums wird durch zweidimensionale Koordinaten auf jeder der beiden Kamerabildebenen beschrieben. Durch Zusammenfassung dieser Koordinaten erhält man einen vierdimensionalen Vektor u, der die Position des Zylinders im Arbeitsbereich eindeutig festlegt. Eine Projektion des Balkens auf die x-y-Achsen der Kamerabildebenen bestimmt seine Orientierung. Hierzu muß jedoch durch Kennzeichnung eines Balkenendes eine Richtung zugeordnet werden. Dies geschieht in der Abbildung durch eine Pfeilspitze. Das auszuwählende Balkenende ist dabei beliebig, wichtig ist nur, daß in beiden Kameras die jeweils korrespondierenden Enden ausgezeichnet werden. In der Simulation hat man nun das Balkenende ausgewählt, das in der Bildebene von Kamera 2 „höher" liegt. Es ergeben sich zwei zweidimensionale Vektoren (x_{x_1}, x_{y_1}), (x_{x_2}, x_{y_2}), die zu einem vierdimensionalen Vektor $(x_{x_1}, x_{y_1}, x_{x_2}, x_{y_2})$ zusammengefaßt werden. Dieser enthält alle relevanten Informationen der Zylinderorientierung und liefert zusammen mit Vektor u sämtliche zur Manipulatorsteuerung nötigen Daten.

Steuerung mittels hierarchischer Kohonen-Netze

Die Verarbeitung der Eigabedaten erfolgt mittels eines hierarchischen Netzwerks, dessen dreidimensionales Übergitter aus zweidimensionalen Untergittern zusammengesetzt ist. Durch das Übergitter wird die Position des Zylinders im Arbeitsbereich repräsentiert, während ein zugeordnetes Untergitter die Orientierung des Zylinders an dieser Position wiedergibt. Ein Auswahlverfahren bestimmt nun dasjenige Unternetz s des Hauptgitters, das die Zylinderposition am besten repräsentiert. In diesem wird dann ein Neuron q ausgewählt, dessen Synapsenstärke der Orientierung des Zylinders am nächsten liegt. Das so bestimmte Netzelement gibt schließlich die Ausgabegrößen an die Gelenke des Roboterarms weiter. Die hierarchische Anordnung der Kohonen-Netze ermöglicht eine effiziente Suche nach den verantwortlichen Ausgabeneuronen, denn der Aufwand hat eine Komplexität von $O(N^3) + O(N^2) \approx O(N^3)$ im Gegensatz zu $O(N^5)$ bei einem fünfdimensionalen Kohonen-Netz. Auch der Speicherbedarf der Synapsenstärken ist gering, da anstatt eines achtdimensionalen Vektors für ein fünfdimensionales Kohonen-Netz zwei Vierdimensionale für das hierarchische Netzwerk verwendet werden können.

Nach dem Auswahlprozeß werden Adaptionschritte für die beiden Netzebenen durchgeführt.

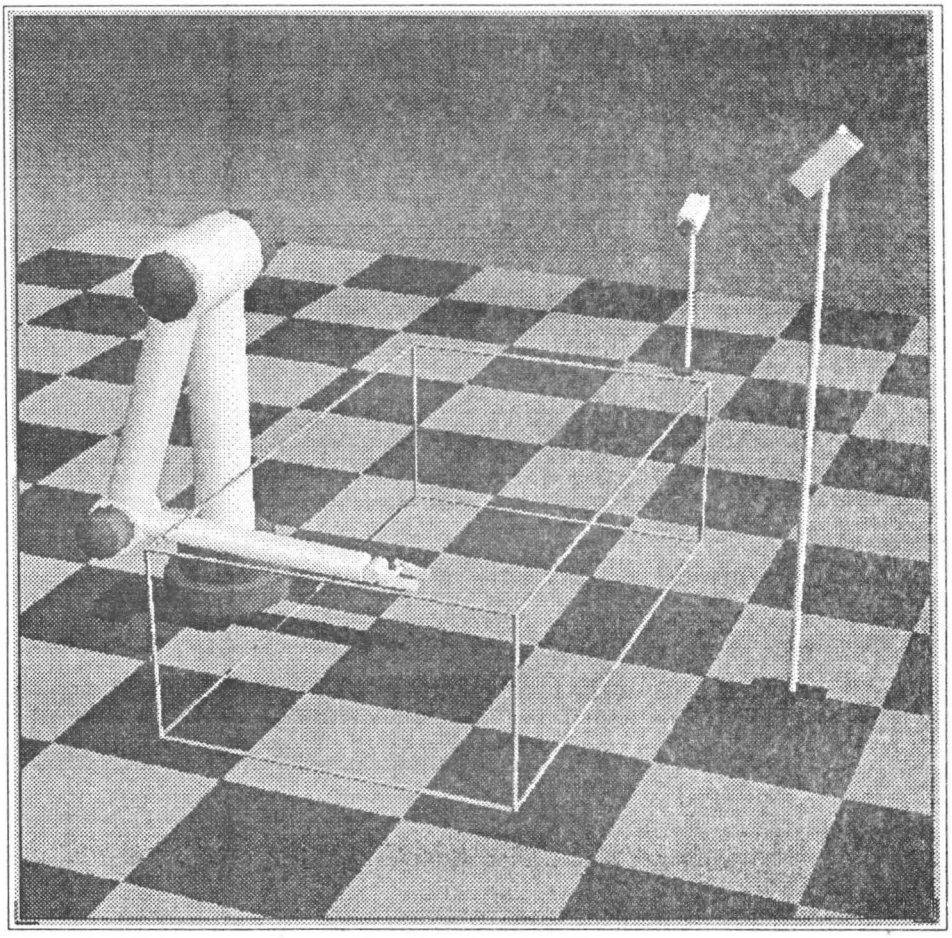


Abbildung 8.7: Modell des Roboters, dessen Geometrie dem menschlichen Arm nachgelidet ist.

Zunächst erfolgt ein Lernschritt für die Anpassung an die Raumposition des Zylinders im Arbeitsbereich gemäß $w_r^{neu} = w_r^{alt} + \epsilon h_{rs}(u - w_r^{alt})$, dann für die Ausrichtung des Neurons auf die Zylinderorientierung mittels $z_{sp}^{neu} = z_{sp}^{alt} + \delta g_{qp}(x - z_{sp}^{alt})$. Dabei beschreibt h_{rs} die Lernreichweite im Übergitter und g_{qp} im Untergitter. Nun ensteht aber durch die erste Gleichung eine topologieerhaltende Anordnung der Unternetze entsprechend der Eingabesignale u, so daß benachbarte Unternetze im Gitter ähnliche Zylinderorientierungen benachbarter Zylinderpositionen repräsentieren. Deshalb ist es sinnvoll, einen Adaptionsschritt für die Zylinderorientierung nicht nur auf ein Unternetz zu beschränken, sondern auf korrespondierende Neuronen in anderen Unternetzen auszudehnen. Dies geschieht mittels $z_{rp}^{neu} = z_{rp}^{alt} + \delta h_{rs} g_{qp}(x - z_{sp}^{alt})$.

Eine solche Datenrepräsentation bietet sich immer dann an, wenn sich die Eingangssignale zu Gruppen mit unterschiedlichen Eigenschaften des betrachteten Objekts einteilen lassen, die zudem noch verschiedene Prioritäten aufweisen. In unserem Modell sind dies die Position und die Orientierung des Zylinders im Arbeitsbereich. Hier kommt der Objektposition die größere Priorität zu, da der Manipulator zuerst am Objekt plaziert werden muß, bevor mit der Feinsteuerung des Greifvorgangs begonnen werden kann.

Erstaunlich ist, daß ähnliche hierarchische Anordnungen der Datenrepräsentation auch im visuellen Kortex höherentwickelter Tiere, wie zum Beispiel Hubel oder Wiesel, zu finden sind.

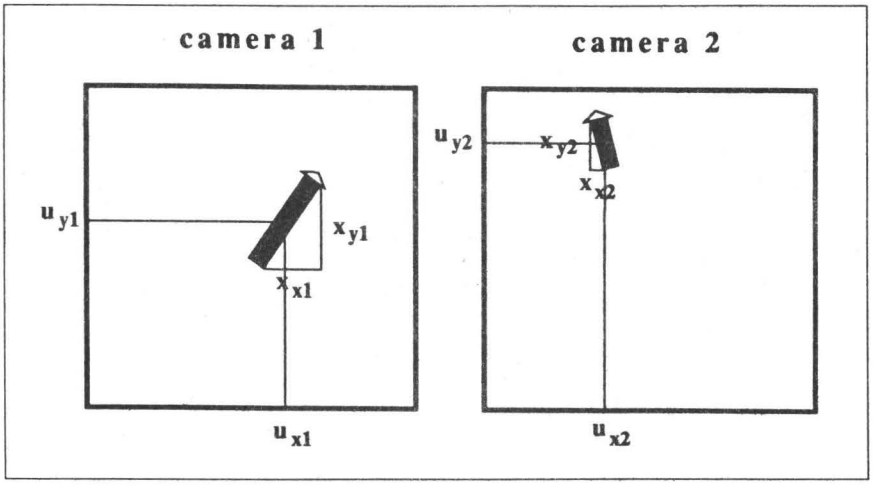


Abbildung 8.8: Projektion des Zylinders auf die Bildebenen der beiden Kameras.

Positioniervorgang und Simulation

Durch Raumposition und Orientierung des Zylinders ist die Gelenkwinkelkonfiguration des Roboterarms eindeutig festgelegt. Den 3 Freiheitsgraden der Raumposition und den beiden der Orientierung des Zylinders stehen die 5 Freiheitsgrade von Arm und Greifer des Roboters gegenüber. Dies hat zur Folge, daß unterschiedliche Orientierungen des Zylinders bei gleicher Raumposition stets verschiedene Winkelstellungen der Gelenke erfordern. Das gleiche gilt für konstante Orientierungen bei unterschiedlicher Raumposition.

Das durch die Vorgabe eines Objekts ausgewählte Neuron q im Unternetz s sei nun für die Angabe der 5 Gelenkwinkel verantwortlich. Zu diesem Zweck hat es einen fünfdimensionalen Vektor $\vec{\phi}_{sq}$ und zur Interpolation zwischen den Stützstellen, eine 5x8-Matrix A_{sq}, als Ausgabegrößen gespeichert. Die auszugebende Winkelkonfiguration errechnet sich dann wie folgt: $\vec{\phi} = \vec{\phi}_{sq} + A_{sq}(\tilde{u} - \tilde{w_{sq}})$ wobei $\tilde{u} = (u, x)$ und $\tilde{w_{sq}} = (w_s, z_{sq})$. Nach dieser ersten Positionierung erfolgt ein Korrekturschritt. Zuvor muß jedoch mit den Kameras die Zwischenposition- und Orientierung des Greifers bestimmt werden. Die Zwischenposition des Zentrums P wird durch den vierdimensionalen Vektor v_i wiedergegeben, die Zwischenorientierung durch den vierdimensionalen Vektor y_i. Beschreibt Vektor $\vec{x}$ die Orientierung des Zylinders, so ist zur Korrektur der Manipulatororientierung der Ausdruck $||x - y_i||$ zu minimieren.

Die Orientierung des Greifers wird durch einen virtuellen Normalenvektor beschrieben. In der Praxis könnte man diesen durch zwei gut sichtbare Markierungen direkt gegenüberliegend an der Ober- und Unterseite des Greifers realisieren. Die gedachte Verbindungslinie zwischen beiden Markierungen, projeziert auf beide Kamerabildebenen, könnte dann die Daten von $\vec{n}$ liefern.

Faßt man v_i und y_i zu $\tilde{v_i} = (v_i, y_i)$ zusammen, so bleibt die Differenz $\tilde{u} - \tilde{v_i}$, aus Zwischenstellung und Endposition des Manipulators, durch einen Korrekturschritt zu überbrücken. Die Anpassung der Gelenkwinkel beträgt dann für Neuron sq $\Delta\vec{\Theta} = A_{sq}(\tilde{u} - \tilde{v_i})$, wodurch sich als Endposition des Lernschritts $\tilde{v_f} = (v_f, y_f)$ ergibt. Dieser Korrekturschritt kann mehrmals hintereinander ausgeführt werden und erlaubt eine dem Menschen ähnliche Greifstrategie, die Kollisionen mit dem zu greifenden Objekt vermeidet. Im Modell wollen wir uns jedoch auf eine Korrekturbewegung beschränken.

Die beiden Phasen „Grobpositionierung und Korrekturbewegung“ sind also fast identisch zur Positionierung des Manipulators in den beiden vorangegangenen Simulationen, verändert hat sich aber, durch die hierarchische Netzanordnung, die Realisierung der Nachbarschaftskooperation zwischen den Neuronen. Wurde Neuron sq durch die Kameradaten als Erregungszentrum ausgewählt, so werden seine Ausgabewerte und die in der Nachbarschaft adaptiert. Das heißt, der Lernerfolg

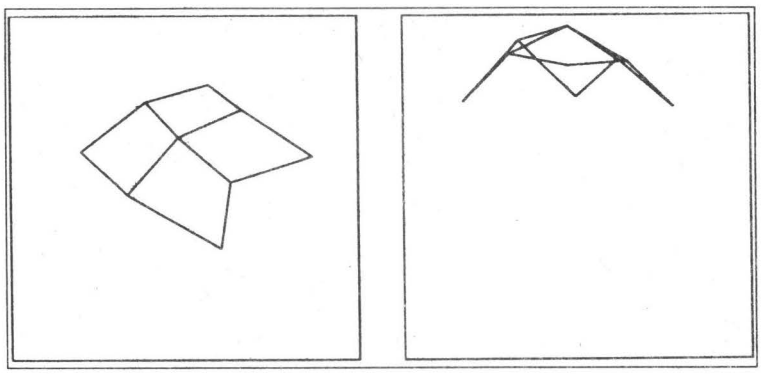


Abbildung 8.9: Das Unternetz nach 10000 Lernschritten

von Neuron sq wird auf die Nachbarneuronen p innerhalb des Unternutzes s übertragen und auf die Neuronen p benachbarter Unternetze r.

Nach Vorstellung des Lernalgorithmus kommen wir nun zu den Ergebnissen der Simulation. Zur Simulation wird ein 4x7x2-Hauptgitter, bestehend aus 3x3-Unternetzen benutzt. Kamerastellung und Arbeitsbereich sind identisch zu den Simulationen der vorangegangenen Kapitel und der Lernfortschritt wird durch eine Projektion der rezeptiven Felder der Unternetze auf die Bildebenen der beiden Kameras dargestellt. Die Entwicklung des Übergitters verläuft dabei ähnlich zu den bereits durchgeführten Simulationen, während die Adaption der Unternetze einer Erklärung bedarf. Sie wird im wesentlichen durch das Eingabesignal $x = (x_{x_1}, x_{y_1}, x_{x_2}, x_{y_2})$ bestimmt, das in jedem Neuron durch den Vektor z_{sp} repräsentiert ist. Die beiden ersten Komponenten geben dabei die Balkenorientierung aus Sicht von Kamera 1 an, die beiden Letzten aus Sicht von Kamera 2. Abb. 8.9 zeigt das Ergebnis nach 10000 Adaptionsschritten in den Bildebenen der beiden Kameras. Da ein Zylinder in jeder Lage in eine Kugel eingepaßt werden kann, versuchen sich alle Netze, als Teil einer Kugeloberfläche auszubilden, um die Orientierung des Zylinders zu beschreiben. Dabei ist zu beachten, daß es sich bei den Unternetzen um zweidimensionale Unterräume des vierdimensionalen Eingagssignalraums handelt. In der Bildebene von Kamera 2 erkennt man, daß das Netz nur die obere Kugelhälfte nachzubilden versucht. Dies hängt damit zusammen, daß wir immer das Balkenende ausgezeichnet haben, das in der Bildebene von Kamera 2 „höher" lag, sodaß y_2 nur positive Werte annehemen kann. Dagegen nimmt das von der linken Kamera dargestellte Unternetz eine beliebige Position auf der gedachten Kugeloberfläche an.

Stellt man den Lernerfolg des Netzes durch die Positionierungs- und Orientierungsfehler des Greifers in Abhängigkeit von der Anzahl der Lernschritte dar, so erhält man nach 1000 Lernschritten einen Positionierungsfehler von 0,004 Längeneinheiten, also 9% der Abmessungen des Roboterarms und einen Orientierungsfehler von $1,7°$. Der erste Fehler liegt zwar hier etwas höher als in den bereits durchgeführten Simulationen, die erzielte Genauigkeit reicht aber für die Aufgabenstellung genauso aus, wie die der Orientierung. Der Orientierungsfehler ist hier sogar kleiner als der der menschlichen Hand beim Greifen eines Gegenstandes.

Die Greifstrategie

Eine direkte Zielpositionierung des Greifers würde in der Regel zu Kollisionen mit dem Zylinder führen. Um dies zu verhindern wird der Roboterarm so konstruiert, daß Objekte „von vorne" angefahren werden können. Dabei macht man sich eine Bewegungstrajektorie zunutze, die der der menschlichen Hand sehr ähnlich ist. Das Greiferzentrum P wird nicht direkt ins Zylinderzentrum geführt, sondern vor dem Objekt plaziert. Dabei sollte der Manipulator bereits die passende Orientierung angenommen haben, um auf dem letzten Stück der Bewegungstrajektorie nur noch auf eine geeignete Positionierung achten zu müssen. Hierzu wird P in der Verlängerung der Symmetrieach-

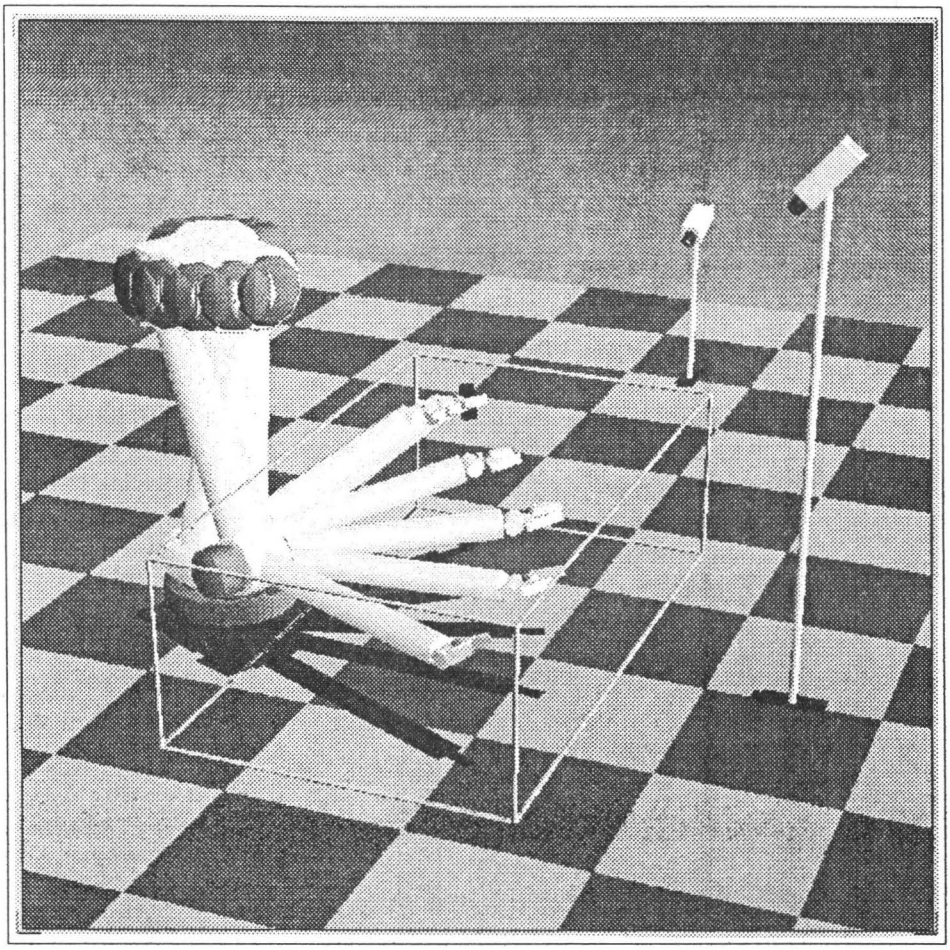


Abbildung 8.10: Stoboskopaufnahme einer Greifbewegung

se vor den Greifer aus Abb. 8.6 verlagert. Dadurch erreicht man mit dem ersten Bewegungsschritt den gewünschten Erfolg, so daß der Greifer nur noch durch Rückkopplung ins Zylinderzentrum geführt werden muß. In mehreren Korrekturschritten wird nun die Abweichung zwischen Greifer- und Zylinderzentrum minimiert, bis der Restfehler bezüglich Positionierung und Orientierung den geforderten Mindestwert unterschritten hat.

Sei v_i' die von den Kameras registrierte Position des Manipulators nach dem ersten Bewegungsschritt und y_i seine Orientierung. Mit $\tilde{v}_i' = (v_i', y_i)$ gilt dann für den Korrekturschritt: $\Delta\vec{\phi} = \gamma A_{sq}(\tilde{u} - \tilde{v}_i')$. Der bereits vorgestellte Algorithmus kann für diese Aufgabe mit kleinen Änderungen verwendet werden. Damit ist der Roboter in der Lage beliebig im Arbeitsbereich angeordnete Zylinder kollisionsfrei zu greifen und zu transportieren, wie in Abb. 8.10 zu sehen ist.

8.3.4 Lernen ballistischer Bewegungen

Bei schnellen Bewegungen beeinflussen sich die Gelenkbewegungen aufgrund der Massen der Armsegmente gegenseitig. Es treten Trägheits- Zentrifugal- und Kreiselkräfte auf, so daß die Motoren ihre Drehmomente nicht mehr allein aus Ist- und Sollgelenkstellung herleiten können, sondern zusätzlich die Bewegungssituation der übrigen Gelenke mitberücksichtigen müssen. Neben der Kinematik muß also auch die Dynamik des Arms, in Form von „Newtonschen Bewegungsglei-

chungen", beachtet werden. Solche Bewegungsgleichungen werden aber rasch sehr komplex und benötigen zusätlich Informationen von Trägheitssensoren der einzelnen Armsegmente. Deshalb versucht man auch hier adaptive Algorithmen einzusetzen, insbesondere den erweiterten Kohonen-Algorithmus.

Die Aufgabe besteht nun darin, den Roboterarm durch Drehmomentimpulse in seinen Gelenken auf eine bestimmte Geschwindigkeit zu beschleunigen, wobei er sich danach kräftefrei bewegen soll. Diese Steuerung soll durch das Netz gelernt werden. Im Gegensatz zu den vorangegangenen Simulationen wird die Armkonfiguration nun explizit durch den Vektor $\vec{\Theta} = (\Theta_1, \Theta_2, \Theta_3)$ angegeben, während die Bewegungssteuerung durch die Drehmomente $d = (d_1, d_2, d_3)$ erfolgt. Dazu wird wieder ein Gitter verwendet, das an jedem Neuron einen Vektor w_r und eine Matrix A_r speichert. w_r gibt dabei die Armkonfiguration direkt durch die Gelenkwinkel an.

Während der Lernphase spezialisiert sich dann jedes Neuron r auf eine Umgebung einer ganz bestimmten Gelenkwinkelkonfiguration des Arms, wobei A_r die für den jeweiligen Bereich gültige Transformation zwischen gewünschter Manipulatorgeschwindigkeit und erforderlichem Drehmoment lernen soll. In einer Reihe von Versuchsbewegungen wird jeweils eine zufällige Manipulatorposition durch $\vec{\Theta}$ vorgegeben von der aus der Manipulator mit einer zufälligen Geschwindigkeit u bewegt werden soll. Es wird nun das Neuron s als Erregungszentrum ausgewählt, für das $||w_s - \vec{\Theta}|| = min_r||w_r - \vec{\Theta}||$ gilt. A_s liefert dann zusammen mit der Vorgabegeschwindigkeit u die erforderlichen Drehmomentamplituden.

Zur Simulation werden die Roboterbewegungen in einem Rechner nachgebildet. Als Arbeitsfläche steht dabei ein Koordinatensystem mit Ursprung am Fußpunkt des Roboters zu Verfügung, das durch ein 15x24-Rechteckgitter repräsentiert wird. Zu Anfang wird jedem Neuron r eine zufällige Manipulatorposition zugeordnet und w_r dementsprechend mit Gelenkwinkeln belegt. Zur Darstellung des Lernfortschritts bezüglich der Eingabewerte w_r werden die einzelnen Gitterplätze den Manipulatorpositionen zugeordnet, die sich aufgrund der vorgegebenen Armkonfiguration $\vec{\Theta}$ ergeben. Die Entwicklung von A_r wird hingegen durch Testbewegungen entlang der Koordinatenachsen demonstriert. Ausgangspunkte dieser Testbewegungen waren jeweils die den Gitterplätzen zugeordneten Manipulatorpositionen.

Zu Beginn der Lernphase ist aufgrund der zufälligen Anfangswerte noch keine Zuordnung zwischen Manipulatorpositionen und Gitterplätzen erkennbar. Auch die Testbewegungen des Manpulators zeigen noch keine Ähnlichkeiten mit den Sollbewegungen. Doch bereits nach 500 Versuchsbewegungen haben sich die Grundstrukturen herausgebildet. Abb. 8.11 zeigt die Situation nach 10000 Lernschritten. Es hat sich eine regelmäßige Zuordnung zwischen Gitterplätzen und Manipulatorpositionen entwickelt und die Testbewegungen verlaufen wie gewünscht.

Wie in den vorangegangenen Simulationen spielt auch hier die Nachbarschaftskooperation zwischen den Neuronen eine wichtige Rolle für die Konvergenz der Ausgabegrößen. Eine Simulation ohne übergreifende Adaption für die Ausgabewerte zeigt nämlich erhebliche Abweichungen von den Sollbewegungen auf. Je geringer die Anfangsreichweiten für die Wechselwirkung der Neuronen untereinader sind, umso langsamer nimmt der mittlere Fehler zwischen Soll- und Istgeschwindigkeit des Manipulators ab und umso größer ist er.

Das hier vorgestellte Verfahren ist auch für andere Anwendungen geeignet. Beispielsweise könnte das Zusammenspiel zwischen Gelenkdrehmomenten und der vom Manipulator ausgeübten Kraft gelernt werden, wobei der Manipulator durch den Kontakt mit der Fläche in seiner Bewegung geführt wird. Damit sind die Anwendungen neuronaler Netze in der Robotik zur Genüge diskutiert.

8.4 Zusammenfassung

Zu Anfang dieses Kapitels wurde anhand des Stabbalance-Problems und der Steuerung von Augensakkaden deutlich, wie topologieerhaltende Karten für Steuerungsaufgaben eingesetzt werden können. Interessant dabei war, daß durch die topologieerhaltende Eigenschaft der Karte Kooperationen zwischen benachbarten Neuronen möglich waren, die wesentlich zur Konvergenz des Verfahrens beitrugen.

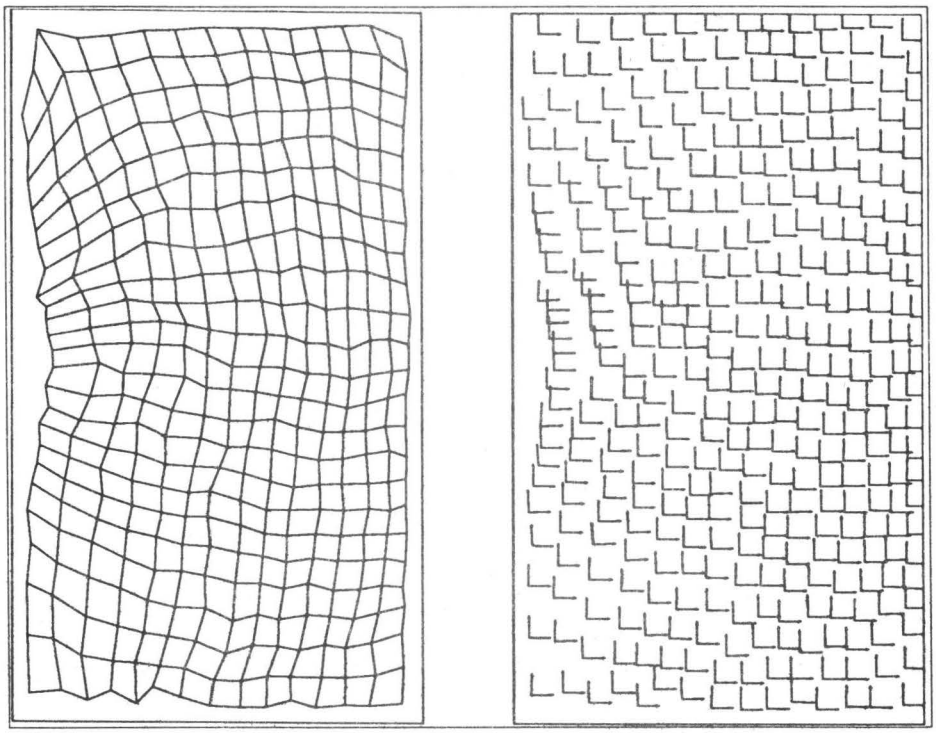


Abbildung 8.11: Endresultate nach 10000 Versuchsbewegungen.

Der zweite Teil beschäftigte sich vorwiegend mit der kinematischen Steuerung von Roboterarmen mittels neuronaler Netze. Dabei war es nicht nur möglich, eine beliebige Position im Arbeitsbereich anzusteuern, sondern auch das darauf aufbauende „Greifen von Objekten" wurde realisiert. Erstaunlicherweise konnte trotz zunehmender Komplexität in den Aufgabenstellungen durch eine geeignete Strukturierung der neuronalen Netze der Rechen- und Speicheraufwand verringert werden. Gegen Ende wurden dann noch Simulationsergebnisse einer dynamischen Steuerung von Robotern vorgestellt. Diese Art der Steuerung ist zum Beispiel dann wichtig, wenn der Einfluß der Schwerkraft auf Armbewegungen kompensiert werden soll, um so einen für die Änderung der Manipulatorgeschwindigkeit verantwortlichen Faktor zu eliminieren.

8.5 Literatur

[1] Helge Ritter, Thomas Martinetz, Klaus Schulten. *Neuronale Netze: Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke.* Addison-Wesley (Deutschland) GmbH, 1992.

[2] Paolo Ferrara, Alois Ferscha, Günter Haring. *A Collision Avoiding Six Legged Walking Machine based on Kohonen Feature Maps.* 1992, 10th European Conference on Artificial Intelligence Edited by B. Neumann, Published by John Wiley & Sons Ltd.

æ

Kapitel 9

Bildverarbeitung und Mustererkennung

Ulrike Becker

Übersicht

In diesem Artikel werden zwei Ansätze zur Mustererkennung und Bildverarbeitung in Neuronalen Netzen behandelt. Das von Kunihiko Fukushima entwickelte Neocognitron erkennt handgeschriebene Ziffern wieder; das von Seibert und Waxman beschriebene Modell ist in der Lage, Gegenstände im dreidimensionalen Raum aufgrund von Bildern wiederzuerkennen.

9.1 Einleitung

Zur Bildverarbeitung und Mustererkennung in Neuronalen Netzen gibt es eine Vielzahl unterschiedlicher Modelle, die auf den jeweiligen Anwendungsbereich zugeschnitten sind. Da es absolut unmöglich ist, auf sie alle auch nur im Ansatz einzugehen, sollen im folgenden zwei verschiedene Modelle betrachtet werden, die völlig verschiedene Ziele und Einsatzgebiete haben. Das erste Modell wurde entwickelt, um handgeschriebene Ziffern zu erkennen; das zweite Modell wurde mit dem Ziel entwickelt, Objekte zu lernen und wiederzuerkennen, die ihm im ‚Alltagsleben', d. h. im dreidimensionalen Raum, dargestellt wurden.

9.2 Das Neocognitron-Modell

Ein spezielles Neuronales Netzwerk-Modell zur Erkennung von optischen Signalen ist von Kunihiko Fukushima und Mitarbeitern entwickelt worden. Fukushima arbeitet in den NHK Broadcasting Laboratories/Japan. In seinem Forschungsbereich befaßt er sich mit dem Informationsfluß im Gehirn und da besonders mit optischer und akustischer Zeichenerkennung. Das Neocognitron-Modell (nach dem Cognitron bereits das zweite von Fukushima entwickelte Modell) entstand aufgrund von vielen Beobachtungen und Tierversuchen und soll (später einmal) das Auge sowie den bildverarbeitenden Teil des Gehirns simulieren.

Die Anwendung des Neocognitron besteht in der Erkennung von handgeschriebenen arabischen Ziffern.

Die Eingabeschicht U_S des Modells besteht aus einer Matrix von Photozellen. Jede der übrigen Stufen besteht aus zwei (Teil-) Schichten, der U_S-Schicht, die nur aus S-Zellen, und der U_C-Schicht, die nur aus C-Zellen aufgebaut ist.

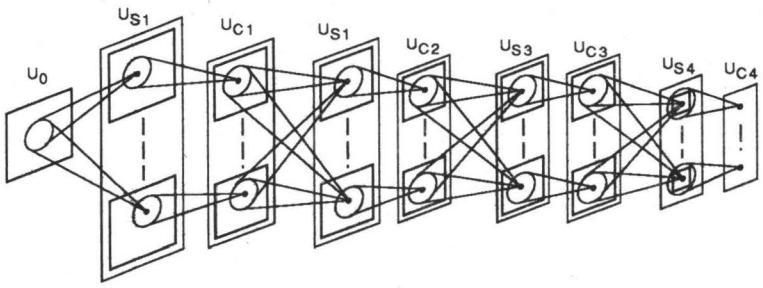


Abbildung 9.1: Schematischer Aufbau des Neocognitrons

Eine S-Zelle hat eine Reaktionscharakteristik, die einer einfachen (simple!) Zelle in der Klassifikation von Hubel/Wiesel entspricht; eine C-Zelle ähnelt eher einer komplexeren Zelle nach dieser Klassifikation.

Eine Zelle in einer oberen (kleineren) Schicht des Neocognitron-Modells hat einerseits die Fähigkeit, ganz gezielt auf ein eher komplizierteres Merkmal des stimulierenden Musters zu reagieren; gleichzeitig hat sie aber auch ein weiteres Aufnahmefeld (d. h. sie reagiert auf sehr viel breiteres Spektrum von auslösenden Merkmalen des Eingabemusters), und ist weniger empfindlich gegenüber Verschiebungen des Musters.

Jede S-Zelle hat modifizierbare Eingabesynapsen, die im Laufe des Lernprozesses verstärkt werden, und erwirbt so die Fähigkeit, ein ganz bestimmtes Merkmal des Eingabemusters ganz gezielt herauszufiltern. Mit der Zeit wird eine S-Zelle dann nur noch auf dieses bestimmte Merkmal des Eingabemusters reagieren.

Jede C-Zelle hat Verbindungen zu einer Gruppe von S-Zellen, die Rezeptorenfelder mit ähnlicher Reaktionscharakteristik an nahezu den gleichen Stellen der Eingabeschicht besitzen. Das hat zur Folge, daß alle präsynaptischen S-Zellen beinahe die gleichen Reaktionsmerkmale extrahieren, allerdings aus geringfügig verschiedenen ‚Blickwinkeln' der Eingabeschicht. Somit wird eine C-Zelle aktiviert, sobald wenigstens eine ihrer präsynaptischen S-Zellen aktiv ist. Wenn also ein Muster, das eine deutliche Reaktion bei einer C-Zelle bewirkt hat, etwas in der Position verschoben wird, wird die C-Zelle genauso reagieren wie zuvor, da eine andere ihrer präsynaptischen S-Zellen aktiv wird. Eine C-Zelle reagiert also auf gleiche Merkmale des gegebenen Reizes wie ihre präsynaptischen S-Zellen, ist aber weniger empfindlich gegenüber Verschiebungen.

In jeder Schicht werden S- und C-Zellen danach in Gruppen, den ‚Zellebenen', zusammengefaßt, die auf ganz bestimmte Merkmale reagieren.

Abb. 1 läßt die synaptischen Verbindungen zwischen den Schichten erkennen. Die dick gezeichneten Vierecke stellen eine S- bzw. eine C-Zellebene dar, die senkrechten, dünn gezeichneten Vierecke eine S- bzw. eine C-Schicht (eine Schicht enthält also mehrere Ebenen ihres Typs). Der Einfachheit und Übersichtlichkeit halber ist in dieser Abbildung nur eine Zelle pro Zellebene gezeichnet.

Jede Zelle erhält nun die Signale von all denjenigen Zellen, die innerhalb ihrer Ellipse auf der vorangehenden Schicht liegen. Innerhalb einer Zellebene haben alle Zellen ähnlich placierte Eingabesynapsen, nur die Positionen der vorgeschalteten Zellen sind parallel verschoben. Da die Zellen des Netzwerkes kaskadiert verbunden sind, wird der Aufnahmebereich größer, je tiefer die Schicht liegt (d. h. je größer ihr Index ist). Die Zellendichte einer Ebene nimmt ab, je mehr die Größe des Aufnahmebereichs zunimmt (die Anzahl der Zellen der einzelnen Schichten sind der Abbildung zu entnehmen).

In der letzten Schicht ist das Aufnahmefeld jeder C-Zelle genau so groß, daß die Bandbreite der Eingabe damit erfaßt werden kann (hier also zehn Zellen für die zehn möglichen Ziffern). Außerdem besteht hier jede Zellebene nur noch aus einer einzigen Zelle.

Abb. 2 zeigt die Verbindungen zwischen den einzelnen Zellen. S- und C-Zellen sind erregende

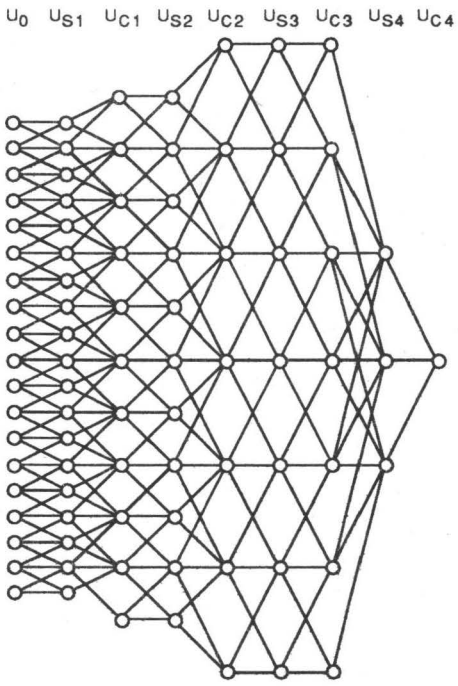


Abbildung 9.2: Darstellungen der Verbindungen zwischen Zellen verschiedener Schichten. In jeder Schicht ist der Einfachheit halber nur eine Zellebene dargestellt

Zellen; außerdem enthalten S-Zellen auch noch hemmende Zellen.

Abb. 3 zeigt – etwas vereinfacht – die Signalverarbeitung und -weitergabe. Der Eingabewert errechnet sich u. a. aus der Summe der gewichteten Eingabewerte der vorhergehenden Zellen, sowie dem gewichteten hemmenden Eingabewert. Ist der Wert > 0, so wird er an die nachfolgende(n) Zelle(n) weitergegeben, sonst wird der Wert 0 weitergegeben.

Die Synapsenverbindungen des Neocognitron-Modells werden mittels überwachtem Lernen (supervised learning) reinforced, d. h. das Netzwerk benötigt einen ‚Lehrer'. Während der Lernphase präsentiert der Lehrer der Eingabeschicht einen Satz von Trainings-Mustern und Vorgaben, wie welche Zelle des Netzwerkes auf jedes Muster zu reagieren hat.

Der Reinforcement-Prozeß in einer S-Zelle läuft von oben nach unten ab, d. h. die Schichten mit den kleinsten Indizes zuerst. Der Reinforcement-Prozeß der Eingabesynapsen der l-ten Schicht wird erst vorgenommen, nachdem die $(l-1)$-te Schicht komplett behandelt wurde. Die einzelnen Zellebenen der S-Schicht werden einzeln behandelt. Dazu präsentiert der Lehrer der Eingabeschicht ein Trainingsmuster und sucht eine Zelle der gesamten Ebene als Repräsentant aus. Beim Reinforcement-Prozeß werden diejenigen Synapsen verstärkt, die positive Eingabewerte haben (damit wird diese Zelle in Zukunft eine spezielle Reaktion gerade auf dieses Eingabemuster entwickeln). Die anderen Zellen dieser Ebene werden genauso modifiziert wie ihr Repräsentant.

Da die Verbindungsstrukturen zwischen zwei Schichten gleichartig sind, soll die Reaktion der C-Zelle anhand der obersten Schicht in Abb. 4 demonstriert werden (jede Zelle einer beliebigen Schicht hat ein ähnliches Reaktionsmuster, wenn man die Ausgaben ihrer präsynaptischen Zellen als Eingabe betrachtet).

Die Zelle u – genauer die Reaktion dieser Zelle – setzt sich aus den (gewichteten) Ausgabewerten $p(\nu)$ ihrer präsynaptischen Zellen zusammen; in diesem Fall sind dies Zellen der Eingabeschicht, sowie der Ausgabe b der hemmenden Zelle v. Je mehr die $p(\nu)$ von dem Trainingsmuster P abweichen, desto höher wird der Wert b und desto kleiner werden die gewichteten Eingaben $a(\nu)$ und damit auch der Ausgabewert. Die Zelle u bewertet also die Ähnlichkeit zwischen Eingabe-

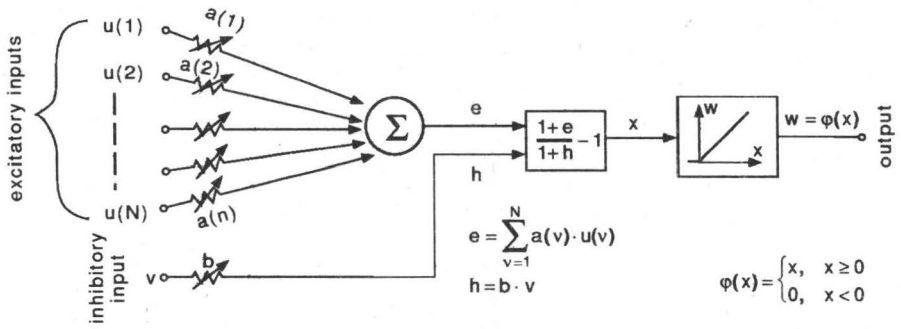


Abbildung 9.3: Ein-/Ausgabeverhalten einer S-Zelle

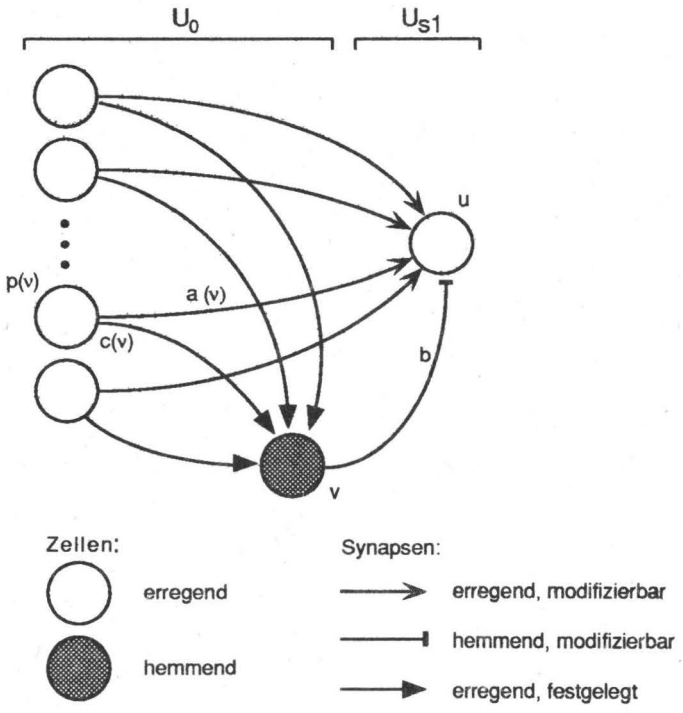


Abbildung 9.4: Synaptische Verbindungen einer S-Zelle

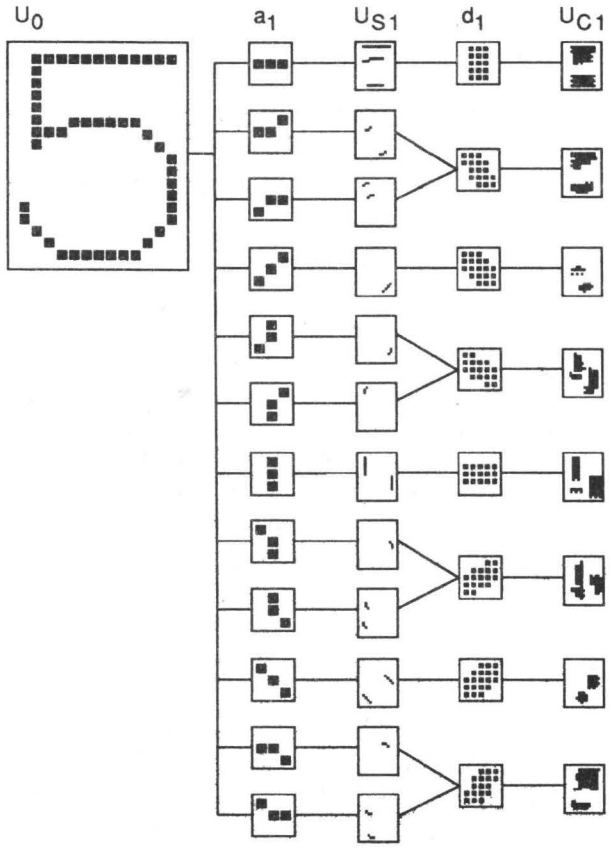


Abbildung 9.5: Reaktion der Zellen in Ebenen U_{S0}, U_{S1}, U_{C1}

und Trainingsmuster und reagiert nur auf Muster, die dem Trainingsmuster gegenüber eine gewisse Ähnlichkeit aufweisen [der exakte Ausgabewert einer Zelle ist das Ergebnis einer ziemlich aufwendigen und komplizierten Rechnung und kann durch Ändern eines bestimmten Faktors noch so verändert werden, daß die Zelle ganz gezielt auf bestimmte Merkmale anspricht. Diese Spezialisierung verändert jedoch auch die Toleranz gegenüber Deformationen, so daß ein Kompromiß zwischen beiden Werten gefunden werden sollte].

Im folgenden soll die Funktionsweise des Neocognitron schichtweise am Beispiel erklärt werden.

Schicht U_{S1} hat zwölf Felder, von denen jedes die gleiche Anzahl Zellen wie die Eingabeschicht U_0, nämlich 19×19, hat. Die Muster, die zum Trainieren dieser zwölf Zellebenen verwendet wurden, sind in Abb. 5 zu sehen.

Die Zellen der ersten Ebene sollen z. B. ein horizontales, vertikales oder diagonales Linienstück (a_1, a_4, a_8) aufspüren. Die a_i stellen also genau das Aufnahmefeld von U_{S1} dar. Da die Ausdehnung der erregenden Eingabesynapsen nur 3×3 beträgt, sollten in einigen Fällen zwei verschiedene Zellebenen auf ein bestimmtes Eingabemuster reagieren (wie z. B. in Abb. 6: Ebenen a_1 und a_3 oder a_5 und a_6 reagieren auf die gleichen Eingabemuster, allerdings in verschiedenen Orientierungen). Dermaßen ‚verwandte' Zellebenen werden in der Eingabe der nachfolgenden Schicht U_{C1} zusammengefaßt. Daher benötigt Schicht U_{C1} nur acht Ebenen.

Die nachfolgende C-Zelle hat 5×5 Eingabezellen, von denen zunächst einige – der äußere ‚Ring' – überflüssig erscheinen. Das Teilmuster wird jedoch in der Richtung, die rechtwinklig zu seiner Hauptorientierung liegt, verlängert.

Da jede C-Zelle erregende Signale von mehreren S-Zellen erhält und ähnlich wie ihre Nach-

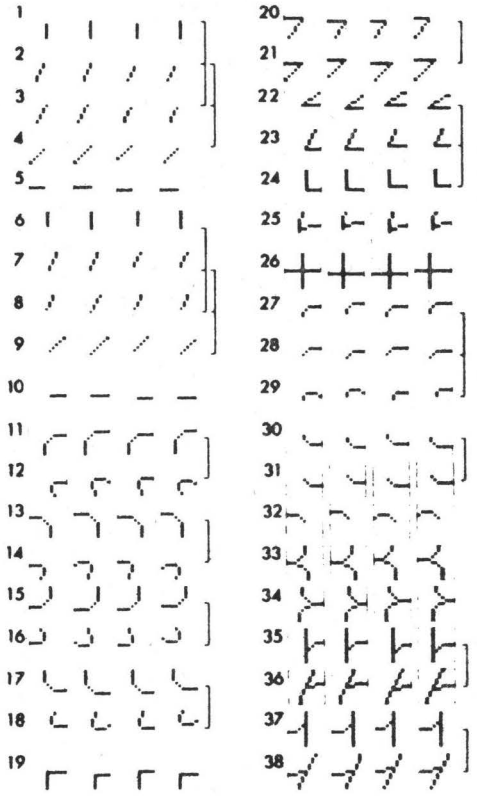


Abbildung 9.6: Die Trainingsmuster, die verwendet wurden, um die 38 Ebenen von Schicht U_{S2} zu trainieren

barzellen reagiert, kann man die Anzahl Zellen in jeder Lage im Vergleich zur vorherigen deutlich verringern. Beim Übergang von U_{S1} zu U_{C1} ist die Anzahl der Zellen von 19×19 auf 11×11 reduziert worden.

Schicht U_2 hat 38 Zellebenen mit je 11×11 Zellen. Bei U_{C2} genügen 22 Ebenen, da in der Vorgängerschicht U_{S2} mehrere Ausgänge zusammengelegt werden konnten. Jede Zellebene besitzt nur noch 7×7 Zellen. Die S-Zellen in U_{S2} haben veränderbare erregende Synapsen mit einer Ausdehnung von 3×3. Da die vorangehende Schicht acht Ebenen besitzt, hat jede Zelle somit $3 \times 3 \times 8$ Eingänge. Während des Lernprozesses werden jedoch nur wenige von ihnen modifiziert; die meisten behalten ihren Initialwert 0. Die Eingabesynapsen jeder C-Zelle in U_{C2} haben eine Ausdehnung von 5×5.

Abb. 6 zeigt die Muster, die verwendet wurden, um die 38 Zellen dieser Schicht zu trainieren. Es handelt sich hier um 38 Gruppen à vier Muster, wobei das Merkmal, das eine Reaktion hervorrufen soll, jeweils um ein Pixel nach oben, unten oder zur Seite verschoben wurde. Jede S-Zelle soll die Fähigkeit erhalten, ein spezielles Merkmal des Stimulus von vier verschiedenen Positionen der Schicht U_0 aus zu extrahieren.

Das Reduzieren von Zellen kann bewirken, daß die Reaktion in Schicht U_{C1} eine etwas andere Reaktion hervorruft, wenn das Muster um eine Position verschoben wurde. Um dem entgegenzuwirken, wird jede Zellebene von Schicht U_{S2} mit vier verschiedenen Eingabemustern trainiert.

Das Neocognitron soll darauf trainiert werden, handgeschriebene arabische Ziffern wiederzuerkennen. Bei handgeschriebenen Ziffern sind die eingegebenen Muster meistens sehr verzerrt, allerdings sind diese Verzerrungen nicht vollkommen willkürlich, sondern gehen i. a. in eine bestimmte Richtung. Diese Verzerrungen werden in mehreren Ebenen dieser Schicht ermittelt und in der Eingabeschicht zu Schicht U_{C2} zusammengelegt.

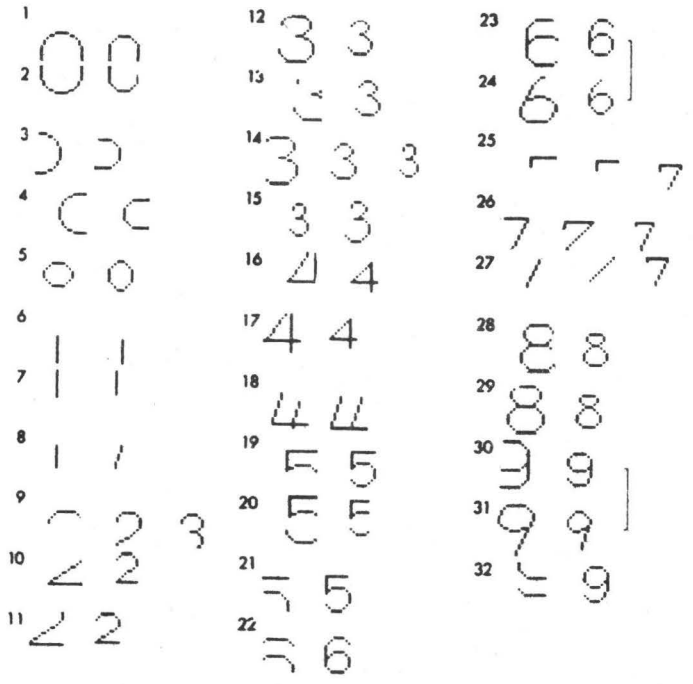


Abbildung 9.7: Trainingsmuster, die verwendet wurden, um die 32 Zellebenen in Schicht U_{S3} zu trainieren und ihre Assoziation an der Eingabe zu Schicht U_{C3}

Schicht U_{S3} hat 32 Zellebenen und U_{C3} hat 30 Zellebenen. Beide haben 7×7 Zellen pro Ebene. Jede S-Zelle hat $3 \times 3 \times 22$ modifizierbare erregende Eingabesynapsen, und die Eingabesynapsen jeder C-Zelle haben eine räumliche Verteilung von 3×3.

Abb. 7 zeigt die Trainings-Muster, mit denen die 32 Zellebenen in Schicht U_{S3} trainiert wurden, und zeigt, wie die Ausgaben dieser Zellebene in der Eingabephase zu Schicht U_{C3} zusammengefügt wurden. Die meisten dieser Trainings-Muster bestehen aus einigen Komponenten der Standard-Ziffern.

Wie die Abbildung zeigt, werden nur zwei oder drei verschiedene Muster benutzt, um jede Zellebene in Schicht U_{C3} zu trainieren. Sie sind deformiert oder unterscheiden sich voneinander in der Größe. Schicht U_{S4} hat 16 Zellebenen, von denen jede 3×3 Zellen hat. Jede dieser S-Zellen hat $5 \times 5 \times 30$ modifizierbare Eingabesynapsen. Die 16 Zellebenen von Schicht U_{S4} werden trainiert mit den 16 Mustersätzen aus Abb. 9 und werden zu zehn Zellebenen in Schicht Uc4 zusammengeführt. Jede Zellebene in Schicht U_{C4} hat nur eine Zelle, mit Verteilung der Eingabesynapsen von 3×3.

Die zehn Zellen der Schicht U_{C4} entsprechen den zehn arabischen Ziffern. Zwischen diesen Zellen existiert ein Mechanismus, der bewirkt, daß sich die Zellen gegenseitig hemmen können, auf den entsprechenden Abbildungen ist dieser Mechanismus jedoch nicht eingezeichnet.

Für einige Ziffern existieren mehrere ‚Schreibweisen'. Für jede von ihnen werden zwei S-Ebenen unabhängig voneinander mit zwei typischen Mustern von verschiedenen Schreibweisen trainiert.

Das Neocognitron, das wie beschrieben trainiert wurde, wird mit verschiedenen Eingabemustern getestet. Abb. 9 zeigt die Reaktion der Zellen auf jeweils eins der Trainingsmuster. Nur eine Zelle in Schicht U_{C2} hat einen Ausgabewert. Das bedeutet, das das Neocognitron dieses Eingabemuster richtig erkennt. elbst wenn das Eingabemuster leicht von dem Trainingsmuster abweicht, wie in Abb. 10, wird es vom Neocognitron noch korrekt erkannt. In Abb. 11 reagieren zwei Ausgabezellen, d. h. ein großer Ausgabewert kommt von Zelle 5, ein kleinerer von Zelle 6. Das bedeutet, daß das Neocognitron feststellt, daß das Eingabemuster eine ‚5' darstellt, aber auch die Ähnlichkeit

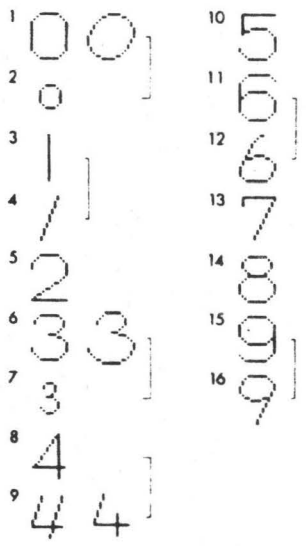


Abbildung 9.8: Trainingsmuster, die verwendet wurden, um die 16 Zellebenen in Schicht U_{S4} zu trainieren und ihre Assoziation an der Eingabe zu Schicht U_{C4}

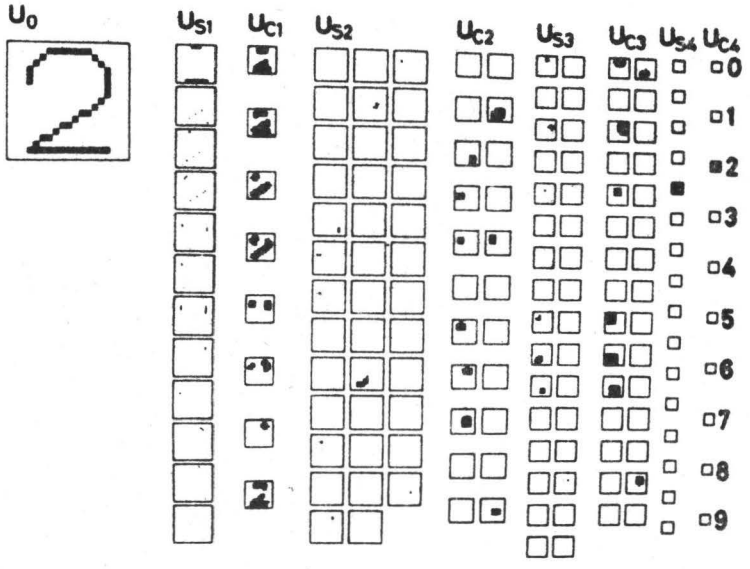


Abbildung 9.9: Reaktion der Zellen des Netzwerks auf das Trainingsmuster ‚2'

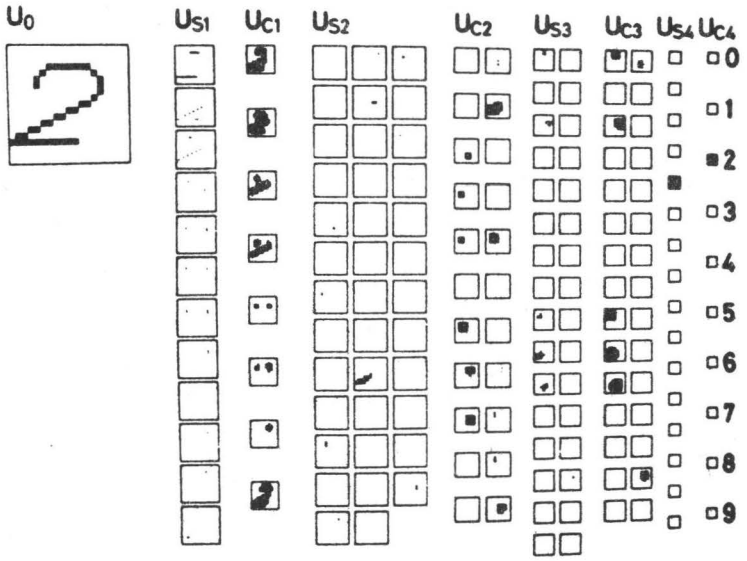


Abbildung 9.10: Reaktion der Zellen des Netzwerks auf eine deformierte Version des Trainingsmusters ‚2'

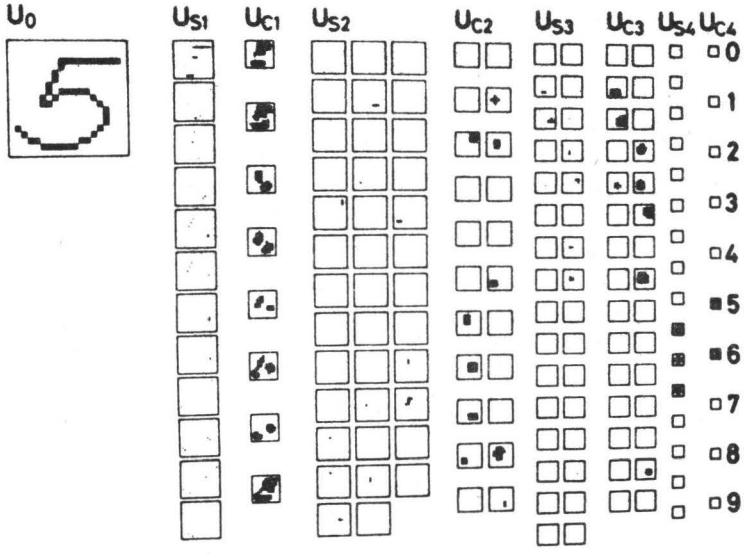


Abbildung 9.11: Reaktionen des Netzwerkes auf eine leicht deformierte ‚5'

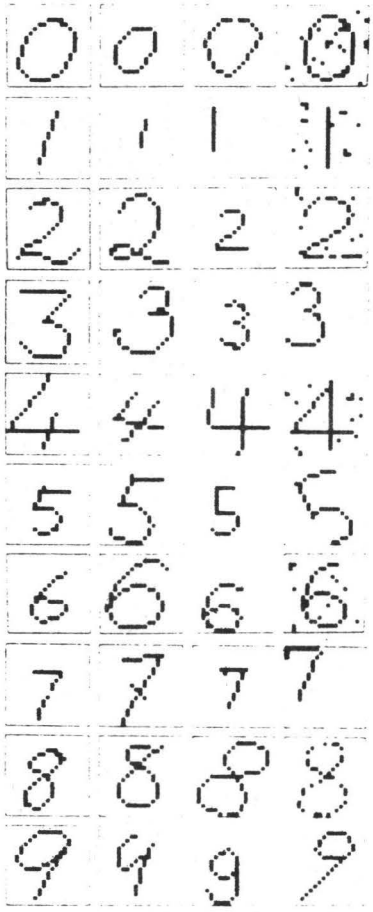


Abbildung 9.12: Beispiele von leicht deformierten Eingabemustern, die das Neocognitron noch korrekt erkennt

mit der Ziffer ‚6' erkennt.

Abb. 12 zeigt einige Eingabemuster, die das Neocognitron noch korrekt erkennt, obwohl einige von ihnen schon ziemlich verformt sind.

Abb. 13 zeigt einige Eingabemuster, die so deformiert sind, daß das Neocognitron nicht mehr in der Lage ist, sie korrekt zu erkennen.

Das Neocognitron kann ebenfalls darauf trainiert werden, daß es handgeschriebene Buchstaben korrekt erkennt; dazu müssen allerdings die Anzahlen der Zellebenen entsprechend geändert werden, abhängig davon, wie groß der Zeichensatz ist, den das Neocognitron wiedererkennen soll.

Obwohl jede S-Zelle eine große Anzahl von modifizierbaren Eingabesynapsen hat, werden nicht alle von ihnen durch den Lernprozeß modifiziert. Die meisten von ihnen behalten ihren Initialzustand bei, in dem ihr Ausgabewert ‚0' beträgt.

9.3 Lernen und Wiedererkennen von 3D-Objekten aufgrund von Bildern

Der folgende von Seibert and Waxman beschreibt ein System, das in der Lage ist, dreidimensionale Gegenstände, die es anhand von Abbildungen gelernt hat, wiederzuerkennen. Auch dieser

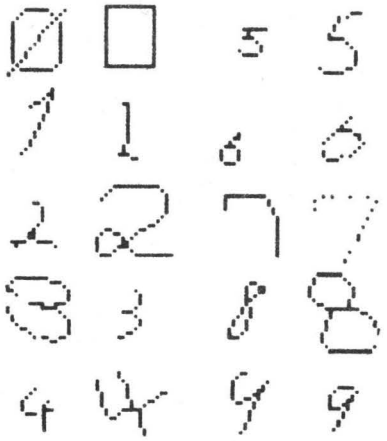


Abbildung 9.13: Eingabemuster, die so deformiert sind, daß sie vom Neocognitron nicht mehr erkannt werden

Ansatz für maschinelles Sehen ist gewissen Gehirnfunktionen nachgebildet. In Tierversuchen wurden Affen verschiedene Bilder (zwei- und dreidimensional) ihrer Artgenossen gezeigt. Dabei wurde festgestellt, daß – je nach Art der Aufnahme – Vorder-, Seiten- oder Rückansicht – verschiedene Gruppen von Zellen aktiviert wurden. Zusätzlich gab es jedoch noch Zellen, die aktiv wurden, wenn ein Übergang der Sichtweise stattfand. Dieses Prinzip wurde auch in dem jetzt beschriebenen System zum Erkennen und Lernen von dreidimensionalen Bildern angewendet.

Abb. 13 zeigt ein System für maschinelles Sehen, basierend auf physiologischen und anatomischen Erkenntnissen. Das System lernt die 2D-Zwischendarstellungen, d. h. die Ansichten zwischen z. B. Vorder- und Seitenansicht (diese sind z. T. sehr verschieden von Größe, Position, Orientierung und Perspektive des tatsächlichen Gegenstandes), und benutzt diese 2D-Information, um bereits gelernte 3D-Objekte zu bearbeiten. Das System hat eine modulare Struktur. Die Eingabe für das System besteht i. a. aus einer Sequenz von Kamera-Bildern, die das Objekt von mehreren Seiten zeigen.

Die theoretische Funktionsweise des System wird hier nur sehr kurz und oberflächlich behandelt; da in fast jedem Modul die Ausgabe wieder mit eingeht, wird das System vor allem durch eine große Zahl von Differentialgleichungen beschrieben; diese jedoch alle mit aufzuführen würde den Rahmen des Vortrags/Artikels bei weitem sprengen. Dafür wird jedoch an einem Beispiel die Arbeitsweise des Systems demonstriert.

Die niedrigsten Module des Systems arbeiten in erster Linie mit feed-forward-Verfahren, um Merkmale zu extrahieren, um 2D-Formen zu lernen, die unabhängig von Größe, Position, Orientierung, leichten Deformationen (wie z. B. perspektivische Verkürzungen) des Sehfeldes sind. Diese 2D-Repräsentationen werden im folgenden als Aspekte bezeichnet.

Die ersten Module in Abb. 13 trennen das eigentliche Objekt von dem Hintergrund (dieses geschieht u. a. aufgrund der unterschiedlichen Helligkeit von Bild und Hintergrund), sie verstärken den Kontrast und extrahieren charakteristische Merkmale des Objektes mit Hilfe des DEB (Diffusion Enhancement Bilayer). Charakteristika sind in diesem Fall Merkmale, die verläßlich extrahiert sind und in Gruppen zusammengefaßt werden können, und ergeben einen Eingabevektor für das nächste Modul, die 2D-Shape-Codierung. Das DEB besteht hauptsächlich aus zwei Schichten, die mittels feedforward und feedback-Verbindungen gekoppelt sind. Einfachere Merkmale, die aus dem Bild extrahiert werden, dienen als Eingabe der unteren Ebene, diese Eingabe wird in eine Verteilungsfunktion der Zeit über ein zweidimensionales Feld umgerechnet. Die zweite Ebene des DEB

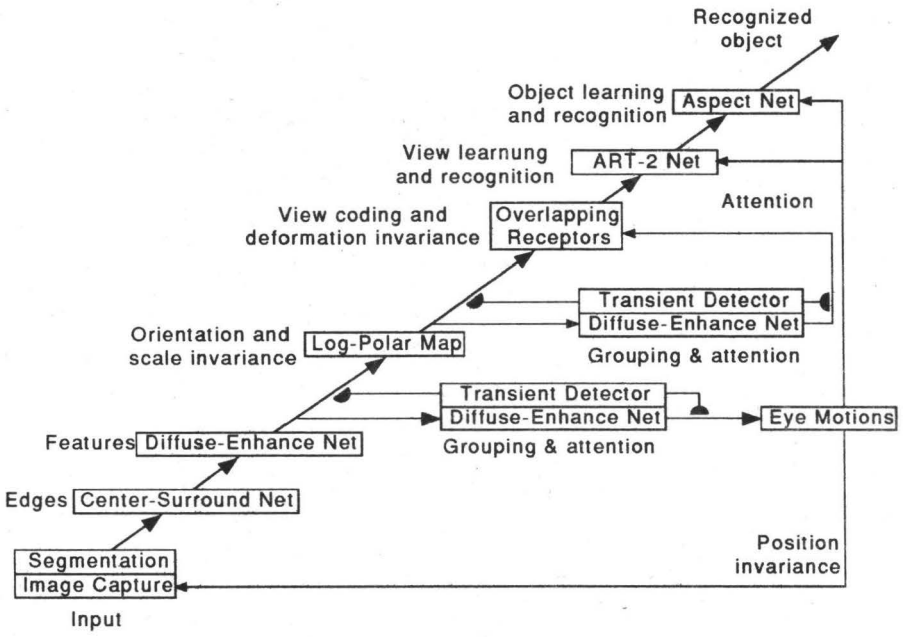


Abbildung 9.14: System-Architektur zum Lernen und Erkennen von 3D-Objekten

findet die lokalen Maxima dieser Funktion und füttert sie wieder in die untere Ebene ein. Auch spätere Module benutzen das DEB mehrfach, um eine Zwischen-Darstellung der 2D-Repräsentation des Objekts zu erzeugen, die unabhängig von Größe, Orientierung oder Lage des Objektes im ‚Sehfeld' sind.

Die Merkmale, die bisher verarbeitet wurden, werden codiert, damit sie durch einen Satz von sich überlappenden Aufnahmefeldern mit Gauss-gewichteter Eingabe klassifiziert werden können. Jeder dieser Eingabevektoren, die in dieser Schicht entstanden sind, dient als Eingabe für ein ART-Modul. Hier muß kurz die Arbeitsweise eines ART-Systems beschrieben werden: Bei den Modellen der ART (Adaptive Resonance Theory)-Familie wird ein neues Eingabemuster erst nach einer Ähnlichkeitsprüfung gelernt. Falls die Übereinstimmung mit einer bereits bestehenden Klasse groß genug ist, wird das neue Muster in diese Klasse mit aufgenommen, sonst wird eine neue Klasse erzeugt. Diese Vorgehensweise hat den Vorteil, daß das bereits Gelernte nicht durch neue Eingaben verfälscht wird.

In diesem Fall sind die einzelnen Klassen Aspekt-Klassen. Jede Aspekt-Klasse hält erregende Eingabewerte sowohl für statische Ansichten als auch für 3D-Übergänge bereit.

Um festzustellen, welches Objekt nun erkannt wurde, wird nach dem ‚Winner-takes-all'-Prinzip der am meisten aktive Knoten ausgewählt.

Das Vorgehen in diesem Netz soll nun am Beispiel erklärt werden.

Abb. 14 zeigt die Ansichten, die von den Objekten – bestehend aus Modellflugzeugen – gesammelt wurden. Die Bildfolgen können in verschiedenen Anordnungen zusammengefügt werden und ergeben so viele verschiedene Trainings- und Testsequenzen, um das Flugzeug zu lernen und wiederzuerkennen. In diesem Fall gibt es 22 Schnittpunkte, an denen Bildfolgen zusammengefügt werden können. In der Trainingsphase wurde eine Folge konstruiert, die jedes einzelne Element erfaßt.

Wenn die 2D-unabhängigen Merkmale durch die unteren Module erfaßt worden sind, werden sie verwendet, um den Sehraum in verschiedene Klassen aufzuteilen. Ähnliche Ansichten von eng beieinanderliegenden Blickpunkten werden in einem gelernten Aspekt zusammengefaßt, der einer Art ‚Mittelwert'-Blickpunkt des entsprechenden Bereichs entspricht. Wenn nun ein neues Merkmal

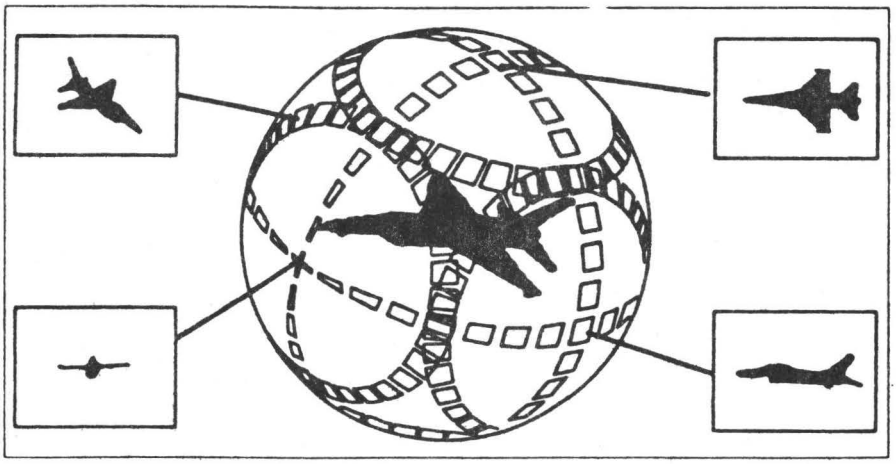


Abbildung 9.15: Aufnahme der verschiedenen Ansichten

– oder Merkmals-Vektor – (erzeugt durch das überlappende Rezeptoren-Feld) mit einem bereits existierenden Merkmal (erzeugt durch ART) übereinstimmt, wird es Teil dieser Klasse und damit Teil des Aspektes, den diese Klasse darstellt. Wenn eine Ansicht in eine bereits existierende Aspektklasse eingeht, hilft sie, die Repräsentation zu verbessern. Auf diese Weise wird die möglicherweise unendlich große Zahl von Darstellungen für ein Objekt auf eine endliche und damit handliche Zahl von Aspekten reduziert.

Abb. 15 zeigt die Bildsequenz eines Flugzeugs, das sich entlang des ‚Nullmeridians' von Abb. 14 bewegt. Die Bildfolge beginnt mit der Vorderansicht. Viele Bilder sind nicht eindeutig, d. h. eine Silhouetten-Ansicht von oben sieht der von unten sehr ähnlich, ebenso fällt es schwer, die Vorder- von der Rückansicht zu unterscheiden.

Abb. 16 zeigt die Graphen-Darstellung, die im ART-Modul durch die Sicht-Kategorien gelernt wurde. Die Sichten von oben und unten fallen in eine einzige Klasse, da ihre Silhouetten identisch sind. Vorder- und Rückansicht bleiben hier getrennt, da es – bedingt durch die leichte Verschiebung des Objektes bzgl. der ‚Erdachsen' – keine Rückansicht gibt, die mit der Vorderansicht übereinstimmt.

In der gröbsten Klassifizierung wurden die 26 Ansichten in zehn Aspekten zusammengefaßt. Andere Test-Flugzeuge waren – v. a. in der Vorderansicht – diesem Testflugzeug sehr ähnlich, so daß sie in vielen Aspekten übereinstimmen.

Dadurch erhalten wir mehrdeutige Bilder, die ohne weitere Information nicht genau erkannt werden. Weitere Sichten sind oft ebenfalls zweideutig, aber eine Folge von Bildern, die eindeutig auf dieses Objekt hinweist, wird durch dieses System schnell entdeckt und löst so dieses Problem.

Viele der hier angedeuteten Ideen sind implementiert worden, vor allem in autonomen mobilen Robotern. Eine vereinfachte Version des hier beschriebenen Objekterkennungs-Prozesses wurde verwendet, um die optischen Stimuli für hemmende und erregende Reize bei der Conditionierung des Roboters MAVIN zu verarbeiten. Im Initialzustand wandert der Roboter ziellos umher. Mit Hilfe seines Sehapparates erkennt und lernt er sie, so daß er sie später wiedererkennen kann. Durch die beschriebene Technik kann der Roboter Objekte im Raum wiedererkennen, unabhängig von Position und Lage im Raum.

9.4 Zusammenfassung

Auch in Zukunft wird es ein breites Spektrum interessanter und verschiedenartiger Einsatzgebiete für künstliche Neuronale Netzwerke geben. Sicherlich ist es faszinierend, daß eine Maschine in der

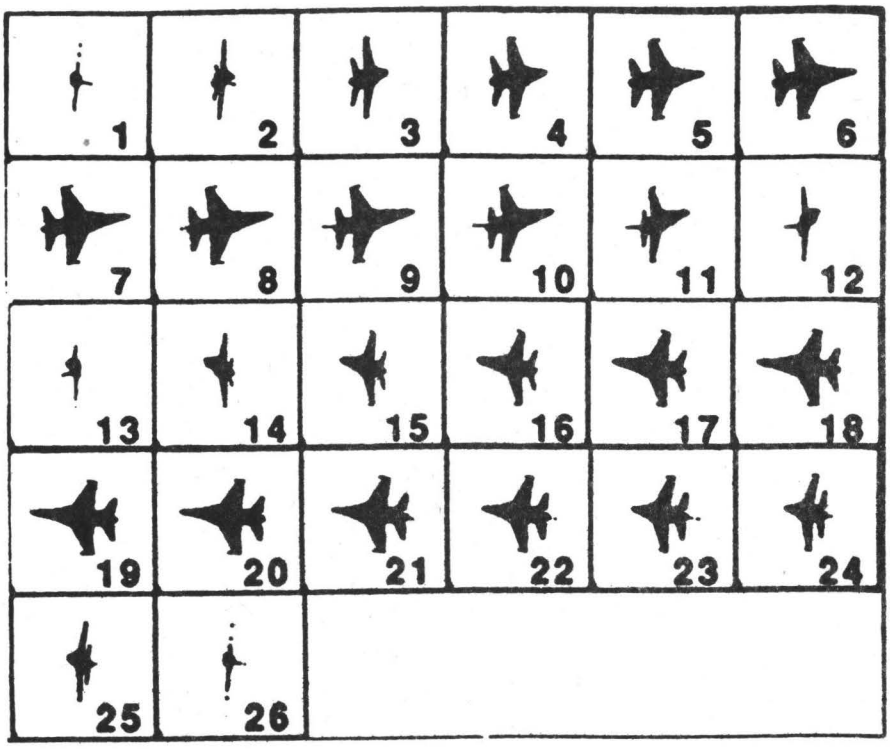


Abbildung 9.16: Die gesammelten Bildsequenzen

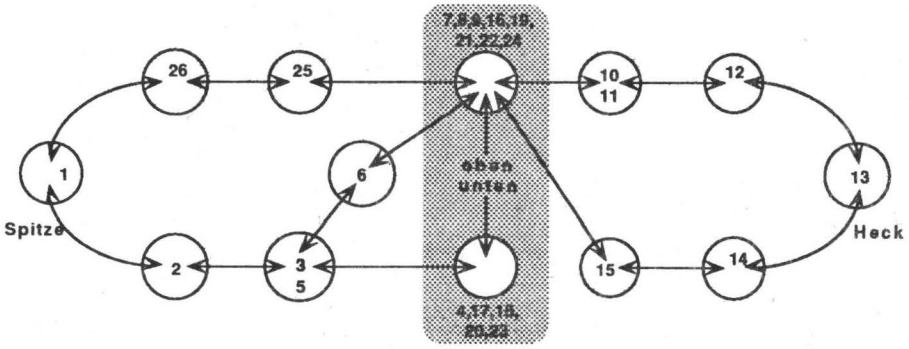


Abbildung 9.17: Die von dem ART-Module gelernten Aspekt-Kategorien

Lage ist, so komplexe Vorgänge wie Bildererkennung zu erlernen und damit ein Teil menschlicher Gehirnfunktionen zu übernehmen. Wenn man jedoch bedenkt, daß das Neocognitron zum Erkennen und Erlernen von nur zehn Ziffern über 14000 Zellen und einen enormen Trainingsaufwand benötigt, kommt die Leistung eines Gehirns (und nicht nur im Bereich Mustererkennung und Bildverarbeitung) deutlich zu Bewußtsein. Das Gehirn dürfte so komplex in Aufbau und Funktion sein (das ganze dabei auf sehr engem Raum), daß es sicherlich nie gelingen dürfte, auch nur Teile davon in der Funktionsweise vollständig zu verstehen oder nachzubilden.

9.5 Literatur

[1] V. Vermuri. *Neural Networks, Theoretical Concepts.* IEEE Computer Society Press, New York, 1988.

[2] Harry Wechsler. *Neural Networks for Perception.* Academic Press, San Diego, 1991.

Kapitel 10

Sprachverarbeitung mit neuronalen Netzen

Christoph Krome

Übersicht

Für die Sprachverarbeitung existieren drei Hauptbereiche:

- Codierung von Sprache
  Die Aufgabe der Codierung ist es, das Sprachsignal in eine Form zu bringen, die einfach weiterverarbeitet und abgespeichert werden kann.
- Synthese von Sprache (Sprachausgabe)
  Die Synthese behandelt die Umwandlung von (geschriebenen) Text in Laute (Phoneme).
- Spracherkennung
  Bei der Spracherkennung wird ein codiertes Sprachsignal als Eingabe für ein neuronales Netz benutzt, um Muster (-sequenzen) zu erkennen.

10.1 Einleitung

Neuronale Netzte eignen sich auf allen Anwendungsgebieten, wo man die Funktionen nicht kennt, die berechnet werden sollen. Hierfür ist Sprachverarbeitung eine gute Anwendung, da Sprache kontextabhängig ist, und es darum sehr schwierig ist, Sprache zu formalisieren.
Für die Sprachausgabe wurden zwar schon Expertensysteme geschrieben, die sehr gute Leistungen gebracht haben, allerdings war die Lösung sehr aufwendig. Durch neuronale Netze werden hier neue Ansätze möglich, die bei weit weniger Aufwand höhere Leistungen erbringen können.
Bei der Spracherkennung gibt es auch andere Ansätze, aber diese sind ebenfalls sehr aufwendig, und könnem micht mehr erweitert werden. Auch hier stellen neuronale Netze eine echte Alternative dar.

Auf die oben angegebenen Punkte soll noch weiter eingegangen werden.

Codierung von Sprache

Hier werden die Grundlagen behandelt, auf der die Weiterverarbeitung durch neuronale Netze aufbaut. Mögliche Komponenten die zur Codierung bzw. zur Signalverarbeitung verwendet werden, sind z.B.

- Filter
- A/D - Wandlung

- FFT (fast Fourier transformation)
- LPC (linear prediction coding)

Allerdings wird auf die Codierung von Sprache nicht ausführlicher eingegangen, da dieses Thema eher zur technischen Informatik bzw. Elektrotechnik zählt. Unter "Spracherkennung" wird aber ein Beispiel gegeben, was alles nötig ist, um eine gesprochene Eingabe aufzubereiten.

Synthese von Sprache

Für die Synthese von Sprache gibt es ein System, welches zum Standardbeispiel für Sprachausgabe, und auch für die Anwendung des *Backpropagation- Algorithmus* geworden ist. Dem NETtalk - Projekt ist darum auch ein eigener Abschnitt gewidmet.

Spracherkennung

Das Thema Spracherkennung ist noch aktueller Forschungspunkt. Bei gesprochener Sprache sind eine ganze Menge von Problemen zu bewältigen:

- es gibt große akustische Abweichungen von einem Wort bei unterschiedlichen Sprechern und verschiedenen Umgebungen
- es bestehen stark nichtlineare zeitliche Abweichungen des Eingabemusters
- Erkennung von Wortgrenzen in fließender, kontinuierlicher Sprache ist schwierig

Hierzu werden zwei Ansätze behandelt. Im ersten Ansatz wird die Erkennung von einzelnen (ganzen) Wörtern durch ein mehrschichtiges neuronales Netz behandelt. Der zweite Ansatz baut gewissermaßen auf dem ersten Ansatz auf, nur sollen diesmal nicht ganze Wörter auf einmal, sondern nur Teile eines Wortes erkannt werden. Diese Wortfragmente werden dann mit Hilfe eines "konventionellen" Algorithmus wieder zu Wörtern zusammengesetzt. Der Ansatz stellt dadurch kein reines neuronales Netz dar, sondern ein hybrides Modell, welches die Vorteile von neuronalen Netzen und konventionellen Programmen vereinigt.

10.2 Beschreibung des NETtalk - Projektes

10.2.1 Aufgabe des Netzwerkes

Das Netz hat die Aufgabe, einen Text, der in ASCII - Form vorliegt, in Lautschrift umzuwandeln, die dann über einen Sprachchip als verständliche Sprache ausgegeben werden kann.

Als Eingabe wird dem Netz der aktuelle Buchstabe, sowie je drei Buchstaben als Kontext nach vorne und hinten gegeben (d.h. es werden immer sieben Buchstaben gleichzeitig betrachtet). Die Kontextinformation ist wichtig, weil die Aussprache eines Buchstaben in verschiedenen Wörtern unterschiedlich ist. Als Ausgabe liefert das Netz 23 Lautmerkmale (Phoneme), und noch drei Artikulationsmerkmale.

10.2.2 Topologie des Netzes

Für jeden möglichen Buchstaben, und noch drei Sonderzeichen (Punkt, Komma, Wortgrenze) existiert ein Eingabeneuron (29 * 7 = 203 Eingabeneuronen). In einer versteckten Schicht existieren 80 Neuronen, und in der Ausgabeschicht existiert für jedes Laut- und Artikulationsmerkmal ein Neuron (26 Ausgabeneuronen). Die einzelnen Schichten sind voll miteinander vernetzt, so daß es insgesamt über 18000 Gewichte gibt.

10.2.3 Training von NETtalk

Für das Training wurde der Backpropagation - Algorithmus verwendet (s. Seminarvortrag 4: Feedforward Perceptrons und Backpropagation), allerdings mit einer leicht geänderten Regel für das Ändern der Gewichte.

$$w_{ij}(t)^{(n)} = \alpha * w_{ij}(t-1)^{(n)} - (1-\alpha) * \gamma * \delta_i^{(n+1)} * S(z_j)^{(n-1)}$$

Dabei stellt $w_{ij}(t)^{(n)}$ das Gewicht von Neuron j zu Neuron i der Schicht n dar, γ ist der Lernfaktor, α ist ein empirischer Wert ($\alpha = 0.9$), der eine Gewichtung vornimmt, und $S(z_j)^{(n-1)}$ ist der Ausgang von Neuron j.

Zum Training wurden zwei Datensorten benutzt:

- von Kindern gesprochene Sätze, die protokolliert wurden
- zufällige Wörter aus einem Wörterbuch

Das Netz bekam also auch (falsche) Aussprachen von Kindern als Eingabe, damit es robuster wird, und besser auf unbekannte Wörter reagieren kann. Dabei wurde als Lehrervorgabe die phonologische Schreibweise benutzt.

Während des Training des Netzes konnten drei Phasen in der Sprachenwicklung festgestellt werden:

- zuerst wurden Vokale und Konsonanten als einzelne Klassen unterschieden. Innerhalb der Klassen blieben die Phoneme noch gemischt, so daß sich die Laute wie Babbeln anhörten.
- es wurden Wortgrenzen erkannt, man konnte einzelne Wörter voneinander unterscheiden
- nach ca. 10 Durchgängen der gesamten Datenbasis entstand eine verständliche Sprache, die sich nach weiteren Durchgängen noch verbesserte.

Innerhalb von 16 CPU - Stunden konnte das Netz so trainiert werden, daß es eine Genauigkeit von 98% erreichte. (Zum Vergleich: Der Vorläufer dieses Systems war ein regelbasiertes Expertensystem (DECtalk), welches mit einem Aufwand von 20 Mann-Jahren eine Genauigkeit von 95% erreichte.)

10.3 Spracherkennung: Ansatz von P.C. Woodland

Im Folgenden wird der Ansatz zur Spracherkennung von P.C. Woodland betrachtet, der in dem Aufsatz "Spoken alphabet recognition using multilayer perceptrons" [1] beschrieben ist.

10.3.1 Aufgabe des Netzwerkes

Es sollen die gesprochenen Buchstaben des englischen Alphabets erkannt werden. Diese Aufgabe ist als bedeutendes Problem anerkannt, da man durch Buchstabieren auch unbekannte Wörter darstellen kann. (Jedes System zur Spracherkennung hat einen stark begrenzten Wortschatz, meistens können nur wenige Wörter, wie z.B. Ziffern erkannt werden.)

Im englischen Alphabet gibt es einige schwer differenzierbare Buchstabengruppen, die sich nur durch den Anfangskonsonanten, bzw. den Endkonsonanten unterscheiden:

- E- set: B, C, D, E, G, P, T, V
- A- set: A, J, K
- die beiden Buchstaben M, N

Die meisten Erkennungsfehler treten zwischen Mitgliedern innerhalb dieser Gruppen auf.

Ein Netzwerk zur Spracherkennung sollte zwei Eigenschaften besitzen:

- es sollte robust sein, d.h. Sprecher - unabhängig, und auch unter nicht-idealen Audio-Bedingungen arbeiten können (z.B. könnten Eingaben über Telefon gemacht werden).
- es sollte effizient sein, d.h. es sollte die Sprache in "Echtzeit" verarbeiten können.

10.3.2 Erstellung der Datenbasis zum Training des neuronalen Netzes

Die Datenbasis für die Eingabedaten wird aus drei Sprechproben von jedem Buchstaben des englischen Alphabets von 104 Sprechern gewonnen. Die Sprache muß zunächst vorverarbeitet werden, bevor sie dem Netzwerk als Eingabe dienen kann (d.h die Sprache muß *codiert* werden. Für die Umwandlung müssen eine Menge von Aktionen ausgeführt werden; auf die technischen Hintergründe der einzelnen Aktionen wird nicht weiter eingegangen.

Aktionen:

- die Sprache wird gefiltert und digitalisiert
- für jedes Wort wird der Endpunkt markiert
- jedes Wort wird in Frames von 25.6 ms aufgeteilt, die sich zur Hälfte mit den benachbarten Frames überlappen (ein Frame stellt hier ein kurzes Stück Text dar).
- für jedes Frame wird eine Fast Fourier Transformation ausgeführt
- jedes Wort wird durch eine lineare Interpolation der Koeffizienten der FFT auf 15 Frames begrenzt (ursprüngliche Wortlänge : 17 - 82 Frames)
- die ersten acht Koeffizienten einer diskreten Kosinus - Transformation werden benutzt, um jedes Frame zu beschreiben.

Die zum Schluß erhaltenen Koeffizienten werden als Eingabe für das neuronale Netz benutzt.

10.3.3 Aufbau des Netzwerkes

Der beschriebene Spracherkenner benutzt ein mehrschichtiges neuronales Netz. Als Eingabe erhält das Netzwerk die Beschreibung von dem aktuellen Frame, sowie sieben Frames nach vorne und nach hinten als Kontext. Insgesamt erhält das Netzwerk also 15 Framebeschreibungen mit je acht Koeffizienten als Eingabe. Insgesamt gibt es also 15 * 8 = 120 Eingabeneuronen.

Das Netzwerk bildet für jeden Buchstaben, der erkannt werden soll, eine Klasse aus. Für jede Klasse gibt es ein Ausgabeneuron (es gibt also 26 Ausgabeneuronen). Wenn am Eingang des Netzes ein Muster angelegt wird, sollte nur der dem Muster zugeordnete Ausgang einen hohen Wert haben.

Es gibt eine versteckte Schicht mit einer variablen Zahl von Neuronen (25 - 75). Die besten Ergebnisse wurden mit 75 versteckten Neuronen erzielt. Ascheinend kann sich das Netz umso besser an die Eingabedaten anpassen, je mehr Neuronen es in der versteckten Schicht gibt. Allerdings gab es bei der Erhöhung von 50 auf 75 Neuronen keine signifikante Verbesserung der Erkennungsrate mehr (Verbesserung von 87.6% auf 88.3%).

10.3.4 Training des Netzwerkes

Mit den Daten der Datenbasis sollten verschiedene Anforderungen getestet werden, darum wurde das Netzwerk je nach Anforderung mit unterschiedlichen Daten der Datenbasis trainiert.

- "multiple speaker data set": das Netzwerk soll sich möglichst gut an eine grössere Anzahl von Sprechern anpassen. Von jedem Wort existieren drei Sprechproben von jedem Sprecher. Für das Training wurden dann die ersten beiden Sprechproben jedes Wortes zum Trainieren, und die dritte zum Testen genommen.

- "speaker independent data set": das Netzwerk soll Sprache von beliebigen Personen erkennen. Für das Training wurde daher die eine Hälfte der Sprecher genommen, und die andere Hälfte zum Testen.

Das Netzwerk wurde mit einem Backpropagation - Algorithmus trainiert (s. Vortrag 4). Dabei wird eine Fehlerfunktion minimiert, die den Unterschied zwischen der gewünschten und der tatsächlichen Ausgabe für ein bestimmtes Eingabemuster darstellt. Das Netzwerk wird zuerst mit allen Eingabedaten gefüttert, bevor die Gewichte geändert werden (off-line training). Dabei werden die Fehler für jedes Gewicht von jedem Neuron für alle Eingabemuster aufaddiert. Für jedes Eingabemuster wurde der Zielwert des entsprechenden Ausgabevektors auf 0.9 gesetzt, der Zielwert der anderen Knoten auf 0.1. Dadurch ist es weniger wahrscheinlich, daß die Gewichte groß werden, und die Fähigkeit des Netzwerkes zu verallgemeinern, wird vergrößert. Dadurch werden dann ähnliche Eingabemuster leichter erkannt.

Die Standard - update - Regel ist:
$\Delta W(t) = \eta * E(t) + \alpha * \Delta W(t-1)$

Dabei bedeuten :
$\Delta W(t)$: Änderung der Gewichte
η : Lernfaktor
$E(t)$: Gesamtfehler für alle Eingabemuster
α : Momentum

Die Backpropagationregel konvergiert sehr langsam, deshalb ist die Wahl der Parameter η und α wichtig. In diesem Ansatz wird deshalb eine adaptive Parameter-update-Regel verwendet. Zur Berechnung der Parameter werden zwei Beziehungen verwendet:

$$\eta_t * |E(t)| + \alpha_t * |\Delta W(t-1)| = S \tag{10.1}$$

$$|sin\gamma| = \frac{\alpha_t * |\Delta W(t-1)|}{\eta_t * |E(t)|} \tag{10.2}$$

Dabei gibt γ den Winkel zwischen den Vektoren $E(t)$ und $\Delta W(t-1)$ an; S gibt die "Schrittgröße" an, die durch eine Formel berechnet wird, die die Anzahl der Gewichte mit einbezieht. S wird verkleinert, falls $|E(t) - E(t-1)|$ einen Grenzwert überschreitet.
Durch Lösen des Gleichungssystems (1.1), (1.2) werden die Parameter η und α bestimmt.

10.3.5 Leistungen des neuronalen Netzes im Vergleich mit anderen Methoden

Die meisten anderen Spracherkennungssysteme benutzen ein Ähnlichkeitsmaß zwischen den Eingabemustern und gespeicherten Referenzmustern. Das neuronale Netz wird mit einem anderen System verglichen: Dynamic Time Warping (DTW). Beim DTW - Mustervergleich werden Schablonen für ein Wort erzeugt, (gemittelt,) und parametrisiert. Das Ähnlichkeitsmaß hier ist der "Abstand" zwischen den Referenzmustern (Schablonen), und dem unbekannten Zeichen.

Im diesem DTW - System werden für die Eingaben für jedes Wort (ein Wort bedeutet hier einen gesprochenen Buchstaben) 12 Schablonen gebildet, dies ergibt insgesamt 312 gemittelte Schablonen. Zum Testen wurde das Eingabemuster mit jedem der Schablonen verglichen, und der Abstand berechnet. Der Abstand der drei am besten passenden jeder Klasse wurde gemittelt, und

die Klasse mit dem geringsten mittleren Abstand wurde als Klasse für das zu erkennende Wort angenommen.

Es folgt eine Tabelle der beiden Systeme (neuronales Netz, DTW) mit den Systemarchitekturen, die die besten Werte geliefert haben:

Vergleich	neuronales Netz	Dynamic Time Warping
multiple speaker	91%	85.8%
speaker independent	88.3%	86.0%

Aus diesen Resultaten geht hervor, daß sich das neuronale Netz besser zur Spracherkennung eignet, als ein vergleichbares DTW - System.

Zum Ermitteln dieser Ergebnisse benötigt das neuronale Netz nur ca. 0.1% der Rechenzeit des DTW - Systems; allerdings ist die Trainingsdauer um eine Größenordnung höher als die Zeit, die das DTW - System braucht, um die Schablonen zu ermitteln.

10.3.6 Anwendung des neuronalen Netzes für eine Telefonansage mithilfe eines Digitalen Signal Prozessors (DSP)

Von P.C. Woodland wurde eine Anwendung des neuonalen Netzes als automatische Telefonansage vorgeschlagen: der Benutzer buchstabiert den Namen, aus einer Datenbank wird dann die zugehörige Telefonnummer herausgesucht, die dann auf dem Bildschirm des Computers erscheint. Eine eingeschränkte Version dieser automatischen Telefonansage wurde von dem Autor realisiert. Dazu wurde ein normaler PC benutzt, der mit einer Karte mit einem digitalem Signalprozessor ausgestattet war.

Ein digitaler Signalprozessor ist optimiert für die Aufgaben, die bei Berechnungen in einem neuronalem Netz vorkommen (Multiplikation und Addition); zusätzlich kann ein DSP auch noch eine A/D - Wandlung vornehmen. Der DSP kann auch noch die Codierung des Signals durch Fourieranalyse effizient berechnen - so braucht man zur Simulation eines neuronalen Netzes sonst kaum noch externe Hardware.

Bei einem Netz mit 50 versteckten Neuronen werden insgesamt 120 * 50 (Gewichte zwischen Eingangs- und versteckter Schicht) + 50 (Bias-Gewichte der versteckten Schicht) + 50 * 26 (Gewichte zwischen versteckter Schicht und Ausgangsschicht) + 26 (Bias-Gewichte der Ausgangsschicht) = 7376 Gewichte benötigt. Um die Ausgabe des neuronalen Netzes zu berechnen, benötigt der DSP nur ca. 8000 Zyklen. Zusammen mit der Codierung des Eingabesignals wird zum Erkennen eines Wortes weniger als 1 ms benötigt (bei einer Zyklusdauer des DSP von 97.5 ns).

Die Realisierung eines neuronalen Netzes durch einen DSP finde ich besonders innovativ, insbesondere, da DSP's schon in Homecomputern (siehe Atari Falcon) eingesetzt werden. Durch den immer stärkeren Trend, den PC als Multimedia - Gerät einzusetzen, werden DSP's in größeren Mengen eingesetzt werden. Dadurch haben es neuronale Netz - Anwendungen leichter sich durchzusetzen, als mit spezieller Hardware.

10.4 Spracherkennung: Ansatz von S.G. Smyth

Im Folgenden wird der Ansatz zur Spracherkennung von S.G. Smyth betrachtet, der in dem Aufsatz "Segmental sub-word unit classification using a multilayer perceptron" beschrieben ist.

Bei dem zuletzt beschriebenen Ansatz von P.C. Woodland wird das Problem der variablen Länge eines Wortes nur schlecht gelöst. Es sollten immer ganze Wörter auf einmal erkannt werden (die unterschiedlich lang sein können), das Netzwerk hat aber nur eine konstante Zahl von Eingabeknoten. Obwohl die Erkennungsrate ziemlich gut sein kann, gibt es doch eine Reihe von Problemen:

- das Ende der Wörter muß markiert werden
- die unterschiedliche Länge der Wörter wird durch Normalisieren bzgl. der Zeit konstant gehalten

- der Ansatz kann nicht auf fließende Sprache erweitert werden, da nur einzelne Worte erkannt werden können.

Es ist ungünstig, die Zahl der Eingabeneuronen so weit zu vergrößern, daß selbst das größte Wort auf einmal erkannt werden kann.
In diesem Ansatz wird das Problem der variablen Länge eines Wortes dadurch gelöst, daß nicht das ganze Wort auf einmal, sondern nur Teile eines Wortes auf einmal erkannt werden. Diese Subwords werden dann zu dem Wort zusammengesetzt, für das die größte Wahrscheinlichkeit besteht.

Dadurch können dann auch die angesprochenen Probleme gelöst werden: sobald das letzte Subword eines Wortes erkannt wurde, kann das Wort ausgegeben werden; dadurch wird dann auch die Markierung des Endes eines Wortes überflüssig. Man braucht auch die Wörter nicht in ein "Eingabefenster" zu pressen die unterschiedliche Länge der Wörter könnte einfach durch eine unterschiedliche Zahl von Subwords modelliert werden (dies wird in dem Originalaufsatz leider nicht konsequent eingehalten). Dieser Ansatz kann dann auch auf fließende Sprache erweitert werden.

10.4.1 Aufgabe des Netzwerkes

In diesem Ansatz werden kurze Sprachfragmente (bis 90 ms) auf Subword-units abgebildet. Ein Subword ist eine Einheit, die kleiner ist als ein Wort, die aber nicht mit Phonemen gleichgesetzt werden kann. (Es wurde festgestellt, daß sich Phoneme nicht zur Charakterisierung von akustischen Ereignissen eignen. Subwords stellen somit ein allgemeineres Konzept dar.) Diese Subwords werden dann durch "dynamisches Programmieren" zu Wörtern zusammengefügt. Dem Netzwerk wird die gleiche Aufgabe gestellt wie im vorhergehenden Ansatz, es sollen die Buchstaben des englischen Alphabets erkannt werden. In diesem Ansatz wurde jedes Wort in drei Subwords aufgeteilt, was sich als optimal für die Erkennung des Alphabets bei einer gegebenen Größe der Datenbasis herausgestellt hat. (Allerdings schränkt die konstante Zahl der Subwords pro Wort auch die maximale Gesamtlänge der Wörter ein, die erkannt werden können!)

10.4.2 Aufbau des Netzwerkes

Das Netzwerk besteht aus einer Eingabeschicht, die als Eingabe ein aktuelles Frame, sowie mehrere Kontextframes bekommt. Es wurden Tests mit einer unterschiedlichen Anzahl Kontextframes gemacht, um die Abhängigkeit der Sprache vom Kontext zu testen. Dabei hat sich natürlich gezeigt, daß Sprache kontextabhängig ist; brauchbare Ergebnisse wurden erst mit vier Kontextframes erzielt, bessere Ergebnisse aber mit sechs Kontextframes. Weiterhin bestand das Netzwerk aus einer versteckten Schicht mit 70 Neuronen, und einer Ausgabeschicht, die 48 unterschiedliche Subwords klassifizieren kann.

Die Ausgangsneuronen liefern einen Wert zwischen null und eins, die als Pseudo - Wahrscheinlichkeit (Summe aller Wahrscheinlichkeiten muß nicht eins ergeben) für ein bestimmtes Subword interpretiert werden kann. Somit kann das Netzwerk als eine Approximation für ein Hidden Markov Modell (HMM) angesehen werden (ein HMM ist ein stochastisches Modell, welches für jedes Wortmodell die Wahrscheinlichkeit angibt, daß das Wortmodell von dem beobachteten Eingabemuster erzeugt wurde).

10.4.3 Training des neuronalen Netzes

Das Netz wird auch wieder mit dem Backpropagation - Algorithmus trainiert. Die Eingabedaten für das Netzwerk sind die gleichen wie aus dem vorhergehenden Ansatz, ein Eingabeframe besteht auch wieder aus acht Koeffizienten.

Die Eingabedaten waren am Anfang nicht in Subwords aufgeteilt, deshalb wurden sie einfach linear segmentiert (für jedes Subword wurde ein Drittel des Wortes angenommen). Das Netz bekommt Frames als Eingabe, die es dann als Subwords klassifizieren soll. Dabei wird dann als Lehrvorgabe dann das erste Subword für das erste Drittel des Wortes, das zweite Subword für das zweite Drittel, und das dritte Subword für das dritte Drittel des Wortes genommen.

10.4.4 Zusammensetzung der Subwords durch "Dynamisches Programmieren"

Der Abbildungsprozess des Netzwerkes ist nicht genau genug, als daß das Frame einfach durch das Ausgangsneuron mit dem höchsten Wert identifiziert werden könnte. Die Erkennungsrate für die Subwords liegt nur bei ca. 50%. An diesem Punkt setzt nun das "Dynamische Programmieren" auf (hier wird der "Viterbi alignment algorithm" verwendet). Die Zusammensetzung der Wörter aus den Subwords ist für jedes Wort des zu erkennenden Vokabulars definiert, darum weiß man, welche Symbolsequenzen legal sind. Für jeden legalen Pfad (Symbolsequenz) wird eine Gesamtwahrscheinlichkeit ausgerechnet, die sich aus den Einzelwahrscheinlichkeiten für jedes Subword zusammensetzt, die von dem Netzwerk geliefert werden. Jedes Wort beginnt mit dem ersten Frame, und endet mit dem dritten. Dabei darf sich jedes erkannte Frame (beliebig oft) wiederholen. Es ist unmöglich, die Wahrscheinlichkeiten von allen möglichen legalen Pfaden zu berechnen, deshalb wird nur der Pfad weiterverfolgt, der die höchste Wahrscheinlichkeit hat. Je nachdem, ob die Wahrscheinlichkeit für das aktuelle oder das nächste Subword im Pfad höher ist, wird die Gesamtwahrscheinlichkeit mit der von dem neuronalen Netz gelieferten Wahrscheinlichkeit für das aktuelle bzw. nächste Subword multipliziert. Wenn Pfade abgeschlossen werden, wird das Wort des Pfades mit der höchsten Gesamtwahrscheinlichkeit erkannt.

10.4.5 Resegmentierung der Trainingsdaten

. Die Trainingsdaten sind am Anfang linear segmentiert (je ein Drittel für jedes Subword), was natürlich nicht der tatsächlichen Segmentierung entspricht. Dadurch werden falsche Lehrvorgaben gemacht, und die Ermittlung des Fehlers ist am Anfang falsch.

Durch das DP fällt als Nebenprodukt der optimale Pfad für jedes Wort ab, woraus man die Segmentierung eines Wortes erhält, d.h. die Trennstellen zwischen den Subwords. Jedes richtig erkannte Wort kann dann neu segmentiert werden, so daß es dem erkannten optimalen Pfad entspricht. Dadurch kann die Trainingseffizenz stark gesteigert werden. (Bei der dritten Resegmentierung nimmt der Fehler allerdings wieder zu, wahrscheinlich wegen Ungenauigkeiten bei den Subwordübergängen).

Bei der Erkennungsrate schneidet dieser Ansatz etwas schlechter ab als der vorhergehende Ansatz, die Leistungen sind aber besser, als die von einem "reinen" HMM-Ansatz, und dem DTW-Ansatz.

10.4.6 Beispiel für die Anwendung von Dynamischem Programmieren

Hier wird ein Beispiel gegeben, welches die Zusammensetzung der Subwords zu Wörtern demonstrieren soll.

Annahme : die Sprache wird auf drei zu erkennende Worte beschränkt, die insgesamt fünf verschiedene Subwords haben. Jedes Subword wird durch eine der Ziffern 1 .. 5 gekennzeichnet, und die drei Worte, die erkannt werden sollen, setzen sich aus folgenden Sequenzen zusammen:
erstes Wort : 1 3 5
zweites Wort : 2 3 4
drittes Wort : 3 4 5

Das neuronale Netz erhält nun sieben Sprachframes als Eingabe, die das zu erkennende Wort darstellen. Als Ausgabe erzeugt das neuronale Netz Pseudo - Wahrscheinlichkeiten für jedes Subword, die durch folgende Tabelle gegeben sind:

Subword	Frame 1	Frame 2	Frame 3	Frame 4	Frame 5	Frame 6	Frame 7
1	0.5	0.2	0.2	0.1	0.1	0.1	0.1
2	0.7	0.6	0.4	0.2	0.1	0.1	0.1
3	0.3	0.7	0.8	0.7	0.5	0.3	0.1
4	0.2	0.2	0.2	0.4	0.5	0.6	0.6
5	0.2	0.2	0.1	0.1	0.2	0.2	0.2

Die Pfade für die Worte setzen sich dann aus folgenden Subwords zusammen:
Wort 1: 1 3 3 3 3 3 5
Wort 2: 2 3 3 3 4 4 4
Wort 3: 3 3 3 3 4 4 4 ...

Die (Pseudo-) Gesamtwahrscheinlichkeit für die einzelnen Worte ergibt sich aus dem Produkt der Einzelwahrscheinlichkeiten entlang dem Pfad:
Wort 1 erhält einen Wert von 0.006, und Wort 2 erhält den Wert 0.035, Wort 3 ist noch nicht abgeschlossen. Es wird also Wort 2 erkannt, da es die größte Wahrscheinlichkeit besitzt. Falls das Wort richtig erkannt wurde, und die Daten neu segmentiert werden sollen, erhält es die neue Segmentierung 2 3 3 3 4 4 4. Diese Segmentierung wird dann als Lehrvorgabe benutzt, wenn das Netz erneut mit dem Wort trainiert wird.

10.4.7 Erweiterung auf fließende Sprache

Für jedes Wort muß ein Pfad von Subwords gefunden werden. Bei mehreren Wörtern, die aufeinander folgen, werden einfach die Pfade verbunden. Allerdings muß für jedes Wort, das zu einem Zeitpunkt erkannt werden können soll, ein eigener Pfad angelegt werden. Bei vielen Worten steigt die Rechenzeit für die Pfadverfolgung stark an. Hier wäre es sinnvoll, ein Grammatik zu definieren, die beinhaltet, welche Wörter aufeinander folgen dürfen. Das Konzept läßt hier noch Möglichkeiten zur Erweiterung offen.

10.5 Zusammenfassung

Der erste Ansatz zur Spracherkennung von P.C. Woodland repräsentiert gewissermaßen eine Standardlösung dieses Problems mit neuronalen Netzen. Die Sprache muß zuerst mit Methoden der Sprachcodierung vorverarbeitet werden, bevor sie in das Netz eingespeist werden kann. Allerdings sind diese Methoden erprobte Technik, und durch die Verwendung von DSP's können sie effektiv berechnet werden (z.B. Fourier Transformation).
Der zweite Ansatz zur Spracherkennung geht besonders auf das Problem der variablen Wortlänge ein. So kann es schon bei nur einem Wort große Unterschiede in der Wortlänge bei verschiedenen Sprechern geben. Darum wird das Wort linear in Subwords unterteilt, die dann als Ganzes erkannt werden können. In dieser linearen Unterteilung liegt meiner Ansicht nach der Schwachpunkt in diesem Ansatz. Hier bieten sich noch Erweiterungsmöglichkeiten, das Konzept der Subwords kann noch verbessert werden. So gibt es Aufsätze nur zu dem Thema der Segmentierung von Sprache (siehe z.b. [5], "Neuronale Netze zur Erkennung von Silbenkernen").

Viele andere Ansätze unterscheiden sich hauptsächlich durch die Architektur des neuronalen Netzes. (Anscheinend kann man noch deutlich höhere Erkennungsraten erreichen, indem man zwei versteckte Schichten anstatt einer nimmt, allerdings nimmt dabei die Zeit, die zum Trainieren benötigt wird, stark zu.)

Dann wurde das NETtalk-Projekt als System zur Spracherkennung vorgestellt. Hier ist die Systemarchitektur klar, und es wurde eine Genauigkeit von 98% erreicht. Neue Erkenntnisse sind hier nicht mehr zu erwarten; möglicherweise kann es noch ein paar Verbesserungen geben.

Ein noch relativ neues Forschungsgebiet ist "natürliche Sprache". Die natürliche Sprachverarbeitung beschäftigt sich hauptsächlich mit dem *Verstehen* von Sprache, es wird versucht, Gram-

matiken für natürliche Sprachen zu erstellen. Hier gibt es zwar noch keine durchschlagenden Ergebnisse, aber neuronale Netze haben gegenüber anderen Systemen einige Vorteile:

- robuster gegen Fehler in der Eingabe
- Muster können vervollständigt werden
- implizite Regeln der Sprache können erkannt werden
- neuronale Netze werden mit Unregelmäßigkeiten der Sprache fertig (s. NETtalk)
- es könnte möglich werden, daß ein neuronales Netz eine Sprache automatisch lernen kann, z.B. von einer Datenbank (siehe z.B. NETtalk, hier wurde zum Training ein Wörterbuch benutzt).

Noch eine Anmerkung zum Schluß: die meiste angegebene Literatur zu den beschriebenen Themen ist aus dem Jahr 1991, oder jünger, also gerade mal zwei Jahre alt. Das Thema neuronale Netze ist daher noch lange nicht ausgereizt.

10.6 Literatur

[1] "Neuronal Networks for Vision, Speech and Natural Language"
BT telecommunication series 1
Linggard, Robert (Hrsg.)
London, 1991

Aufsätze:
P.C. Woodland, "Spoken Alphabet Recognition using Multilayer Perceptrons" (S. 134 .. 147)
S.G. Smyth, "Segmental Sub - Word Unit Classification using a Multilayer Perceptron" (S. 177 .. 192)
R. Linggard, "Neuronal Networks for Speech Recognition : an Introduction" (S. 129 .. 134)

[2] Skript zur Vorlesung "Künstliche Intelligenz 1" 1991
Prof. Barth
Kapitel 9 : Neuronale Netze

[3] "Neuronale Netze"
Rüdiger Brause
Teubner Verlag, 1991
Beschreibung von NETtalk (S. 112)

[4] "Artifical Neuronal Networks"
A. Prieto (Hrsg.)
Springer Verlag, 1991

Aufsätze:
Canas, Ortega, Fernandez, Prieto, Pelayo, "An Approach to Isolated Work Recognition using Multilayer Perceptrons" (S. 340)

Michael Franzini, Alex Waibel, Kai - Fu Lee, "Continous Speech Recognition with the connectionist Viterbi Training Procedure : a Summary of Recent Work" (S. 370)

[5] "Mustererkennung 1992"
S. Fuchs, R. Hoffmann (Hrsg.)
Springer Verlag

Aufsatz:
W. Reichl, "Neuronale Netze zur Detektion von Silbenkernen" (S. 261 .. 268)

Kapitel 11

Neuronale sematische Netze

Susanne Meyfarth

Übersicht

1. Einleitung 2. Was sind und was können neuronale semantische Netze? 3. Aufbau eines statischen neuronalen Netzes 4. Das Lernen eines semantischen, neuronalen Netzes 5. Zusammenfassung 6. Literatur

11.1 Einleitung

Semantische Netze dienen der Wissensrepräsentation. Sie zeichnen sich dadurch aus, daß sie sehr flexibel sind. Diese Flexibilität kostet jedoch auch etwas: eine Anfrage dauert, besonders bei großen Netzen lange. Die Darstellung semantischer Netze durch neuronale Netze ist nun eine Möglichkeit der effektiven Nutzung. Im Folgenden wird zunächst beschrieben, wie semantische Netze prinzipiell funktionieren, dann wird ein Ansatz vorgestellt, aus einem semantischen Netz ein statisches, neuronales Netz zu erzeugen. Schließlich wird gezeigt, wie ein schon vorhandenes neuronales Netz lernen kann.

11.2 Was sind und was können semantische Netze?

11.2.1 Darstellung semantischer Netze

Semantische Netze werden durch Knoten und Kanten als Graph dargestellt. Dabei besteht eine eindeutige Beziehung zwischen den Objekten des zu beschreibenden Bereichs (z.B. Hund) und den Knoten im Graphen (Objekte, Attributwerte,...) Die Kanten drücken die Beziehung der Objekte untereinander aus. Die Beschriftung der Kanten zeigt an, um was für eine Beziehung es sich handelt (Attribute). So ist auch Vererbung möglich.

Beispiel: Abb. 11.1

- Hier sind „Frucht“, „Apfel“ und „Traube“ concepts (= Objekte)
- „Rot“, „Süß“, „Grün“ und „Rund“ stellen Attributwerte dar
- „hat-Form“, „hat-Farbe“ usw. sind Attribute
- “IS-A” beschreibt eine Vater - Sohn - Beziehung; das bedeutet, daß „Apfel“ und „Trauben“ von „Frucht“ erben, d.h. sie haben auch den Wert „Rund“ f'ur das Attribut „hat-Form“.

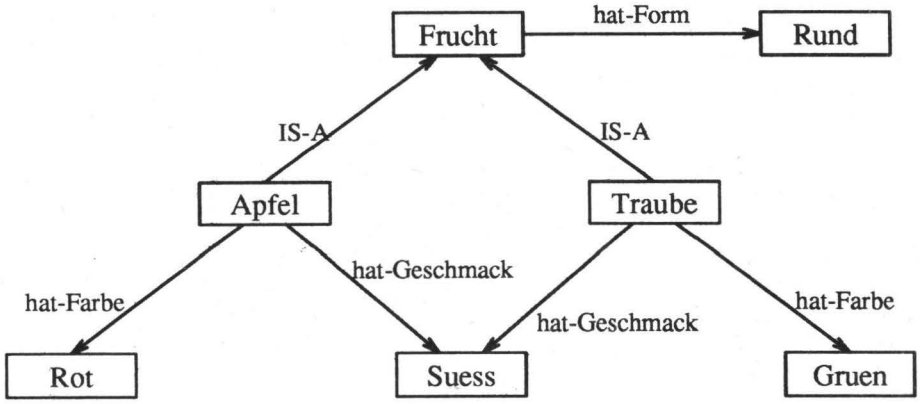


Abbildung 11.1: Ein semantisches Netz

11.2.2 Interpreter und Anfragen

Um semantische Netze nutzen zu können, wurden sie früher als Datenstruktur dargestellt und über einen Interpreter Anfragen gestellt. In dem obigen Beispiel könnte eine Anfrage z.B. so aussehen:

- Was ist die Farbe von „Apfel"? Der Interpreter würde die Verbindungen, die von Apfel ausgehen, untersuchen und feststellen, daß eine Verbindung „hat-Farbe" heißt und mit dieser die richtige Antwort „Rot" finden.
- Was ist die Form von „Apfel"? Auch hier würde der Interpreter alle Verbindungen untersuchen, die von „Apfel" ausgehen, keine passende finden und daher in der Vererbungshierarchie eine Stufe nach oben gehen (zu „Frucht") und dort suchen. Dort würde er mit „hat-Form" die Antwort „Rund" finden.

Diese beiden Anfragen stellen inheritance-Anfragen dar.

11.3 Aufbau eines statischen, neuronalen Netzes

11.3.1 Prinzipielle Darstellung der semantischen Netze durch neuronale Netze

Notwendigkeit von Parallelismus

Semantische Netze werden meist für zwei grundlegende Operationen benötigt, inheritance- und recognition-Anfragen (Erklärung später). Diese werden häufig zum Sprachverstehen gebraucht und sollten daher schnell durchgeführt werden können, um Echtzeit-Anwendung zu ermöglichen.Bei einem großen Graphen, der sehr leicht entstehen kann, ist daher Parallelismus notwendig.

Parallelismus mit neuronalen Netzen

- Jedem Knoten im semantischen Netz wird ein sehr einfacher Prozessor (Neuron) zugewiesen.
- Jede Kante wird durch eine feste Verbindung zwischen den Prozessoren dargestellt, mit festen Gewichten aus [0;1]. Das mit einer Kante im semantischen Netz assoziierte Attribut (Beschriftung) geht dabei zunächst verloren (nicht der Attribut*wert*).
- Die Knoten kommunizieren miteinander wie in neuronalen Netzen üblich, d.h. der Output eines Knotens gelangt über die Verbindungen zu allen Nachbarknoten und liegt dort, korrigiert um die jeweiligen Gewichte, als Input an.

Einschub: Probleme des Parallelismus

Hardware: Die hardwaremäßige Realisierung dieses Ansatzes wirft zunächst Probleme auf (viele Prozessoren nötig). Bei diesem Ansatz wird jedoch davon ausgegangen, daß

1. sich die Technologie ständig weiterentwickelt und damit das Problem in Zukunft gelöst sein wird.
2. man dieses massiv parallele Netz auf existierende parallele Maschinen (z.B. Connection Machine) abbilden kann.

Kontrolle des Nachrichtenflusses: bei anderen parallelen Ansätzen als einem mit neuronalen Netzen wirft dies einige Probleme auf (dort Lösung durch zentrale Kontrolle), hier jedoch ist das Problem durch die Einfachheit der Nachrichten zwischen den Knoten (nur skalare Werte) und beschränkte Weiterleitung (s. später) gelöst.

Effektivität: wird in diesem Ansatz gewährleistet durch

- viele parallel rechnende Elemente (Neuronen)
- keine zentrale Kontrolle und damit keinen Flaschenhals
- feste Verbindungen $\Longrightarrow$ geringe Kommunikationskosten
- einfache Nachrichten $\Longrightarrow$ geringe Kommunikationskosten
- einfache Berechnungsfunktionen innerhalb der Neuronen

Binderknoten

Mit dem bisher Festgehaltenen können wir noch nicht die Attribute selbst darstellen (die im semantischen Netz durch die Beschriftung der Kanten realisiert waren), sondern nur die zugehörigen Werte und das entsprechende Objekt. Damit läßt sich das Vererbungsproblem (inheritance) jedoch noch nicht lösen. Dies soll gleich an einem Beispiel erläutert werden.

Zunächst aber die Definitionen: (dabei entspricht "concept" einem Objekt)

- Inheritance:
  - Geg.: concept C, Attribut
  - Ges.: Wert des Attributs für C, wobei zunächst bei C gesucht wird, dann bei höheren Knoten im Netz, den superconcepts von C.
- Recognition:
  - Geg.: Eine Beschreibung durch eine Menge von Attribut/Werte-Paaren
  - Ges.: concept, das am besten auf die Beschreibung paßt.

Beispiel:

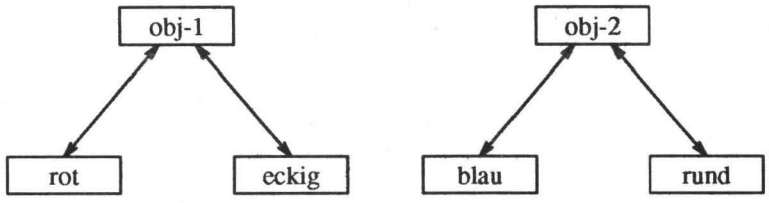


Abbildung 11.2: Demonstration der Notwendigkeit von Binderknoten

Folgende Anfragen wären möglich:

- Recognition:

  „Was ist rot?“ Hier würde also der Knoten „rot“ aktiviert, dieser wiederum würde den richtigen Knoten, „Obj-1“ aktivieren.

- Inheritance:

  „Welche Farbe hat Obj-1?“ Hier müßte der Knoten Obj-1 aktiviert werden, dieser würde aber zu einer Aktivierung von „rot“ und „eckig“ führen, da man das Attribut „Farbe“ nicht darstellen kann. Die Inheritance-Anfrage funktioniert so also nicht korrekt.

In beiden Fällen würde das zweite Subnetz („Obj-2“ usw.) nicht beteiligt.

Lösung:

- Einführung eines extra Knotens pro Attribut
- Einführung eines Binderknotens △, der
  - jeweils die zusammengehörenden Knoten von Objekt, Attribut und Attributwert miteinander verbindet
  - bei zwei Inputs aktiv wird und die Aktivität an den übrigen Knoten weitergibt

(Dies ist eine vereinfachte Darstellung des später vorgestellten, konkreten neuronalen Netzes.) Mit diesem neuen Konzept sieht das Beispiel so aus:

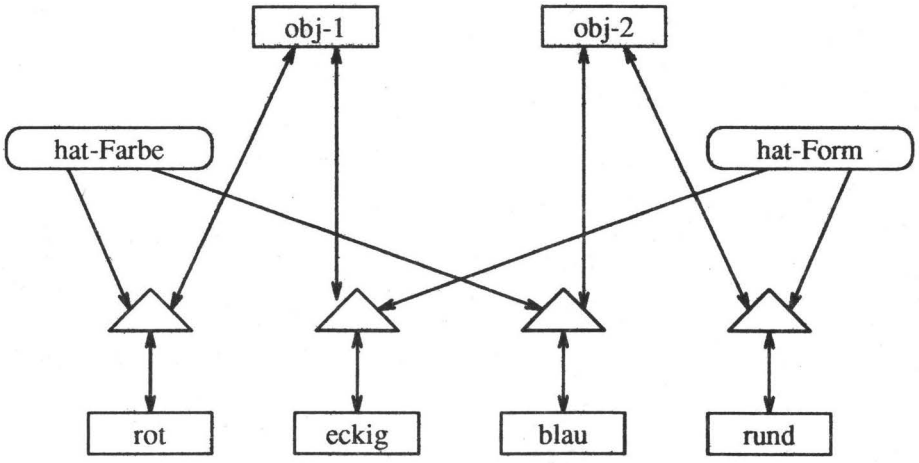


Abbildung 11.3: Neuronales Netz mit Binderknoten

11.3.2 Ein konkretes neuronales Netz

Um das Wissen, das man über eine bestimmte Teilwelt hat, in einem neuronalen Netz darzustellen, wird es durch eine Repräsentationssprache beschrieben. Diese Beschreibung wird einem Netzwerkcompiler als Eingabe gegeben, der daraus das Netzwerk erzeugt. Im folgenden Abschnitt soll beschrieben werden, wofür die verschiedenen Knotentypen im generierten Netz dienen und wie sie untereinander verbunden sind, allerdings zunächst ohne Beachtung der Gewichte an den Verbindungen. Im darauffolgenden Abschnitt soll betrachtet werden, wie sich die Gewichte ergeben.

Knoten und Verbindungen

1. concept-Knoten (ξ- Knoten)

   Mit diesem Knoten werden Objekte und Attributwerte dargestellt (z.B. Apfel, Traube, rot, grün, rund). Entsprechend der Vererbungshierarchie werden die concept-Knoten untereinander verbunden (Achtung: auch bei Attributwerten ist Vererbung möglich, z.B. bunt – rot). Der Einfachheit halber wird dies meist durch eine ungerichtete Verbindung beider Knoten dargestellt, dies ist jedoch nicht ganz korrekt. Die exakte Lösung findet sich bei der Beschreibung der relay-Knoten. Jeder concept-Knoten hat 6 Eingangsstellen, QUERY,RELAY,CP, HCP,PV und INV, jede für einen bestimmten Zweck:

   - QUERY: Über diese Eingangsstelle wird der Knoten von außen aktiviert, wenn eine Anfrage diesen Knoten direkt betrifft.
   - RELAY: Hier kommt der Input der oben angeführten Verbindung zwischen je 2 concept-Knoten an.
   - CP/HCP: diese Eingangsstellen werden benutzt, wenn der concept-Knoten einen Attributwert darstellt. Der Input kommt vom Binderknoten δ_{inh} (s.später).
   - PV/INV: diese Eingangsstellen werden benutzt, wenn der concept-Knoten ein Objekt darstellt. Der Input kommt vom Binderknoten δ_{rec} (s.später).

   Ein concept-Knoten ist dann aktiv, wenn mindestens ein Input anliegt.

2. property-Knoten (ϕ-Knoten)

   Diese Knoten stellen Attribute dar (z.B. hat-Farbe). Sie haben nur eine Eingangsstelle, QUERY, die dann einen externen Input bekommt,wenn eine Anfrage dieses Attribut direkt betrifft.

   Der Knoten ist also aktiv, wenn an QUERY ein Input anliegt. Die Aktivität wird an die betreffenden Binder-Knoten weitergegeben.

3. Binder-Knoten ($\delta_{inh}, \delta_{rec}$)

   Sie dienen dazu, je ein Attribut, Attributwert und Objekt zu verbinden, also 2 concept- und ein property-Knoten Dabei gibt es je nach Funktion zwei verschiedene Typen: Sollen Inheritance-Anfragen gestellt werden, so werden die δ_{inh}-Knoten benötigt, bei Recognize-Anfragen die δ_{rec}-Knoten. δ_{inh}- und δ_{rec}-Knoten treten immer paarweise auf.

   Beide Typen haben zwei Eingangsstellen: ENABLE (E) und EC. Der Einfachheit halber sollen hier nur die Inputs an ENABLE untersucht werden.

   Die δ_{inh}-Knoten bekommen dort Input von einem Objekt(concept-Knoten C), einem Attribut (property-Knoten P) und dem Inheritance-Knoten (s. unter 4.), sie geben ihre Aktivität weiter an einen Attributwert (concept-Knoten V_i), dort an Eingangsstelle CP oder HCP. Schreibweise [C,P $\rightarrow V_i$], also z.B. [Apfel, hat-Farbe $\rightarrow$ rot].

   Die δ_{rec}-Knoten bekommen Input von einem Attribut (property-Knoten P), einem Attributwert (concept-Knoten V_i) und dem Recognize-Knoten (s. auch unter 4.), sie geben ihre Aktivität weiter an ein Objekt(concept-Knoten C), diese kommt beim Objekt an der Stelle PV oder INV an.

   Binderknoten sind nur dann aktiv, wenn an EC alle drei Inputs anliegen.

4. enable-Knoten

   Auch hier gibt es zwei Arten: Recognize- und Inheritance-Knoten.

   Sie haben je eine Eingangsstelle: QUERY. Damit wird von außen der entsprechende Knoten aktiviert, wenn eine Recognize- oder Inheritance-Anfrage gestellt wird.

   Die Recognize-Knoten geben ihre Aktivität an alle δ_{rec}-Knoten weiter, Inheritance-Knoten analog. Somit werden also einmal Recognize-Anfragen ermöglicht, im anderen Fall Inheritance-Anfragen.

5. relay-Knoten

   Wie oben schon angesprochen, dienen sie der genauen Darstellung der Vererbungshierarchie von concept-Knoten. Damit soll eine kontrollierte Weitergabe der Aktivität ermöglicht werden. Jeder concept-Knoten C hat zwei relay-Knoten, C-↑ und C-↓. Beide Typen haben die folgenden Eingangsstellen: ENABLE, OWNER, RELAY und UPSTREAM. Die C-↑-Knoten geben die Aktivität in der concept-Hierarchie nach oben weiter, die C-↓-Knoten nach unten. Auf eine genaue Beschreibung soll hier jedoch verzichtet werden, da sie für das Verständnis nicht unbedingt nötig ist und auf Graphiken diese Art der Darstellung der concept-Hierarchie meist sowieso nicht verwendet wird.

Nun soll ein Beispielnetz gezeigt werden (Abb. 11.4).

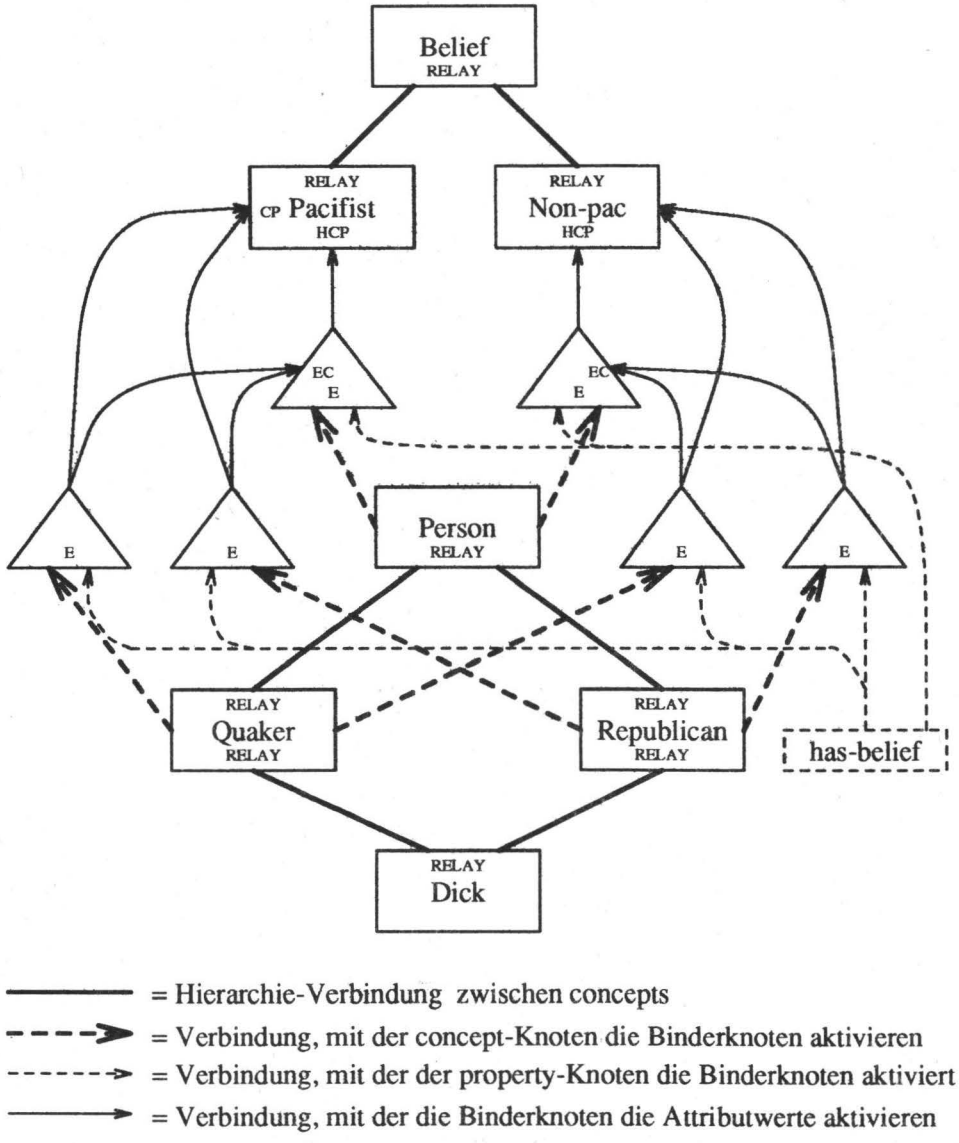


Abbildung 11.4: Ein Beispielnetz

Dieses Netz ist jedoch nicht vollständig: Hier werden nur die Knoten gezeigt, die für eine Inheritance-Anfrage nötig sind, es sind also nur δ_{inh}-Knoten gezeigt. Bei einem realen Netz existie-

ren aber auch noch die zugehörigen δ_{rec}-Knoten. Weiterhin wurden keine relay-Knoten verwendet, sondern stattdessen wurden die concept-Knoten jeweils durch ungerichtete Linien verbunden. Man muß sich die kontrollierte Weitergabe von Aktivität innerhalb der concept-Hierarchie also dazu denken. Außerdem wurden keine enable-Knoten eingezeichnet. Die hier benötigten Inheritance-Knoten haben in Wirklichkeit aber Input bei allen δ_{inh}-Knoten. Darüberhinaus wurden nicht alle Eingangsstellen eingezeichnet, sie sind aber vorhanden.

Die Gewichte

Bisher haben wir uns nur angeschaut, welche Knoten miteinander verbunden sind. Nun müssen wir aber auch noch die Gewichte für diese Verbindungen festlegen. Hier müssen auch die Aspekte Ausnahmen, multiple Vererbung und konfliktäre Information beachtet werden.

Beispiel: Ein Agent glaubt, daß die meisten Quaker Pazifisten sind und die meisten Republikaner Nicht-Pazifisten. Dick ist Quaker und Republikaner. Was gilt aufgrund obiger Annahmen? Ist Dick Pazifist oder Nicht-Pazifist?

Die Gewichte müssen nun so festgelegt werden, daß schließlich der Knoten die meiste Aktivität hat, der in der Realität am wahrscheinlichsten zutrifft. (Um Mißverständnissen vorzubeugen: die Gewichte werden einmal zu Beginn durch den Netzwerkcompiler festgelegt, danach ändern sie sich nicht mehr, d.h. das Netz lernt nicht (zumindest nicht mit diesem Ansatz). Die Frage ist nun: was sind die optimalen Gewichte, die der Netzwerkcompiler berechnen sollte?)

Eine Lösungsmöglichkeit ist, daß man unbekannte Wahrscheinlichkeiten („mit welcher Wahrscheinlichkeit ist Dick Pazifist?“) durch bekannte Wahrscheinlichkeiten abschätzt („mit welcher Wahrscheinlichkeit ist ein Republikaner die Person Dick?“, „mit welcher Wahrscheinlichkeit ist ein Republikaner Pazifist?“, usw.). Diese Wahrscheinlichkeiten sind durch die Anzahl der zu den concepts gehörenden Elemente abhängig, also z.B. die Anzahl der Republikaner in der betrachteten Teilwelt. Die Höhe der Gewichte wird nun von den Wahrscheinlichkeiten bestimmt.

D.h. also, daß man alle Informationen über ein concept, die man hat, verfügbar macht, indem man die Gewichte an den Verbindungen zu entsprechenden anderen Knoten mittels relativer Häufigkeiten (bedingte Wahrscheinlichkeit von Bayes) festlegt. Dies ist das Prinzip der *maximalen Entropie.*

Im Beispiel von Abb.12.4 würde bei der oben angegebenen Anfrage also so vorgegangen werden:

Zu Anfang werden die Knoten “Dick”, “has-belief” und der (nicht abgebildete) Inheritance-Knoten aktiviert. “Dick” aktiviert nun seinerseits “Quaker” und “Republikaner” und diese wiederum geben Aktivität ab an alle hier gezeigten Binder-Knoten. Alle vier Binderknoten bekommen außerdem Input von “has-belief” und vom nicht abgebildeten Inheritance-Knoten. An allen vier gezeigten Binderknoten liegen an ENABLE (E) also 3 Inputs an, soviel wie benötigt wird, damit die Binderknoten selbst aktiv werden können.

Wie hoch das Potential der aktivierten Knoten ist,hängt von der Anzahl und der Höhe aller eintreffenden Inputs ab. Die Höhe der Inputs wiederum hängt von der Höhe des Outputs des absendenden Knotens ab und von der Höhe des Gewichts an der eintreffenden Verbindung.

Die Aktivität wird nun durch das Netz immer weiter nach oben gegeben, bis sie an den entscheidenden Knoten eingetroffen ist, also bei “Pacifist” und “Non-Pac”. Dort wird die Information von den verschiedenen, nach oben führenden Wegen kombiniert (im Beispiel also zum einen der Weg über “Quaker” und zum anderen der Weg über “Republikaner”). Dann wird geschaut, wer die meiste Aktivität hat, also “Pacifist” oder “Non-Pac”, und aufgrund dessen entschieden.

Insgesamt haben (aufgrund der Gewichtsverteilung nach Wahrscheinlichkeiten) die Knoten die größte Aktivität, die auch in der Wirklichkeit am wahrscheinlichsten sind.

11.4 Das Lernen eines semantischen, neuronalen Netzes

Dieses ist ein anderer Ansatz als der eben behandelte. Er behandelt wissenintensives Recruitment-Lernen. Wissenintenives Lernen heißt, daß schon vorhandenes Wissen verwendet wird. Dieses

Wissen kann z.B. ein schon existierendes konnektionistisches, neuronales Netz sein, wie es mit dem ersten Ansatz (Abschnitt 12.3) gewonnen werden kann.

11.4.1 Vorgehensweise und Ziel dieses Ansatzes

- Es wird auf eine eingebaute Wissensbasis, dargestellt als konnektionistisches, semantisches Netz, aufgebaut.
- Das Lernen besteht darin, daß in die vorhandene Wissensstruktur Unternetzwerke aufgenommen werden (s. Abb. 11.5). Damit gewinnt man neue Informationen.
- Das Ziel des Ansatzes ist es, komplexe Herleitungen zu verkürzen.

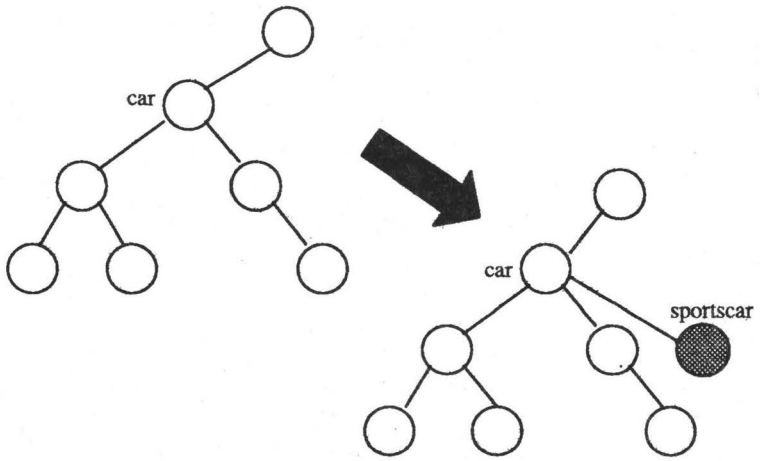


Abbildung 11.5: Einfache Spezialisierung

Das Lernen erfolgt anhand des in Abbildung 12.5 zu sehenden Konzeptes der *einfachen Spezialisierung.*. Das bedeutet, daß an ein existierendes concept eine neue Klasse angehängt wird. Die neue Klasse erbt, wie beim ersten Ansatz auch, die Eigenschaften der Oberklasse, also im Beispiel: wenn für car gilt #wheels(car,4), dann gilt auch #wheels(sportscar,4). Darüberhinaus hat die neue Klasse aber zusätzliche Eigenschaften, die sie von der Oberklasse unterscheiden , z.B. speed(sportscar, fast).

11.4.2 Architektur des Netzes

Alle Einheiten werden auf vier Bereiche aufgeteilt:

- concept space
- attribute space
- instance space
- free space

Wie man sieht, ist in diesem Ansatz der Begriff "concept" ein anderer als im ersten Ansatz: hier wird darunter auf jeden Fall ein Objekt verstanden, für Attribute und Attributwerte gibt es jetzt den attribut space.

Die Einheiten im concept space, attribute space und instance space sind Einheiten, die schon gelernt haben (d.h. sie verkörpern bestimmte Informationen). Sie werden auch als *committed units* bezeichnet.

Im Gegensatz dazu haben die Einheiten aus dem free space noch nichts gelernt. Sie sind sozusagen der Vorrat an „leeren" Einheiten und werden als *free units* bezeichnet.

Alle spaces sind **physikalisch** vollständig mit den anderen spaces verbunden, d.h. jede unit aus dem einen space ist mit allen units aus den anderen spaces verbunden. **Aber:** Die Stärke der Assoziation zweier units ist abhängig vom Gewicht der Verbindung zwischen ihnen. Ein Gewicht von 0 bewirkt, daß keine Assoziation zwischen den Knoten da ist, d.h. es wirkt in dem Moment, als ob gar keine Verbindung vorhanden ist. Der Sinn davon ist, daß man auf diese Weise darstellen kann, daß zunächst keine Assoziation zwischen den Knoten da ist, daß sich das aber ändern kann, wenn sich die Gewichte ändern.Das ist auch die Idee beim *Recruitment-Lernen.*

Recruitment-Lernen

(to recruit = erneuern, ergänzen)

= ein neues concept erzeugen und mit vorhandenem Wissen verbinden

d.h. Umformen eines free units in ein committed unit, indem die Verbindungen zwischen dem free unit und einigen committed units gestärkt werden.

Die Spaces

1. concept space

   - Jedes concept entspricht einem triple unit subnetwork wie in Abb. 11.6:

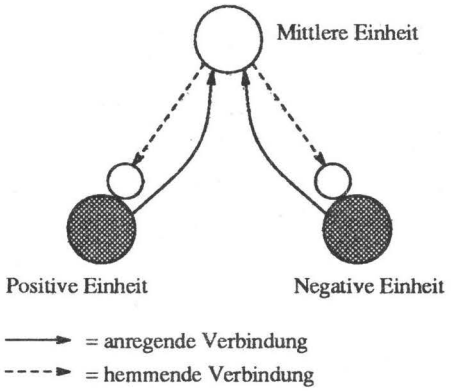


Abbildung 11.6: Realisierung der concepts

   - Eine Hierarchie unter den concepts wird dargestellt wie in Abb. 11.7:

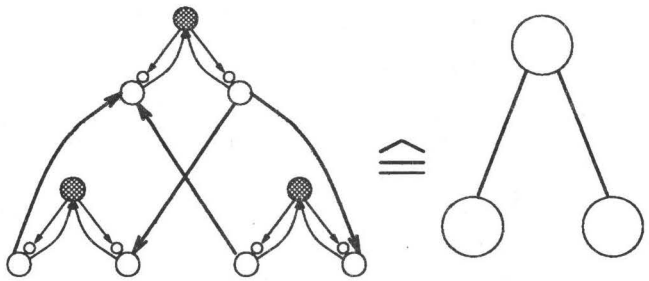


Abbildung 11.7: Realisierung der concept-Hierarchie

- Die Funktion von positiver und negativer Einheit ist, analog den relay-Knoten des ersten Ansatzes, die kontrollierte Weitergabe von Aktivität nach oben bzw. nach unten.
- Vererbung findet statt wie bisher.

2. attribute space

   - Alle Werte eines Attributs sind in einem subnetwork enthalten, das Attribut selbst ist nur implizit darin enthalten.
   - Jedes Attribut wird durch ein Netz analog dem aus Abb. 11.8 repräsentiert.

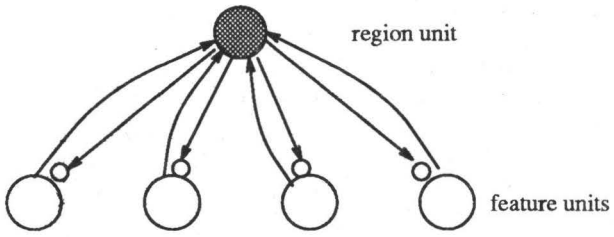


Abbildung 11.8: Darstellung eines Attributs

   z.B. könnten die feature units die Eigenschaften „groß", „mittel" und „klein" sein, das Attribut „Größe" würde dann implizit drin stecken.

   - Das Netz ist als ein "Winner-Takes-More"-Netz (WTM) organisiert, d.h. die region unit hat die Aufgabe, die durchschnittliche Aktivität von allen feature units zu „ermitteln" und alle die feature units zu hemmen, die unter dem Durchschnitt liegen und alle die units aktiviert zu lassen, die über dem Durchschnitt liegen.
   - Wenn ein concept eine bestimmte Eigenschaft hat, wird dies dadurch angezeigt, daß die Verbindungen zwischen der positiven und negativen Einheit des concepts und dem jeweiligen feature unit des WTM ein bestimmtes Gewicht hat.

3. instance space

   - Hier handelt es sich um einzelne units, die mit concept units verbunden sind. Es sind Instanzen der entsprechenden concept units.
   - Sie erben die Eigenschaften der zugehörigen concepts, können aber noch zusätzliche Eigenschaften haben. Dann haben die Verbindungen zu den entsprechenden feature units größere, positive Gewichte.
   - Die Präsentation eines Trainingsbeispiels entspricht dem Aktivieren einer instance unit. Als Folge davon werden das zugehörige concept und evtl. zusätzliche feature units aktiviert.

4. free space

   - Auch hier sind es triple unit subnetworks, wie beim concept space. Es sind im Moment noch „leere" Einheiten, sie sollen später zu concepts werden (deshalb die gleiche Organisation als triple unit subnetwork).
   - Es gibt bidirektionale Verbindungen zwischen
     - jedem feature unit $\leftrightarrow$ jedem free unit
     - jedem pos./neg. concept unit $\leftrightarrow$ jedem pos./neg. free unit
   - Diese Verbindungen haben kleine Zufallsgewichte

11.4.3 Das Lernen

Wie oben schon erwähnt, ist es das Ziel des Recruitment-Lernens, die Verbindungen eines free units zu concept und feature units zu stärken und so aus einem free unit ein committed unit zu machen.Dazu wird Folgendes benötigt:

- Eine domain theory, d.h. das in einem semantischen, neuronalen Netz enthaltene Wissen
- Ein Ziel-concept (davon die positive unit, die u.a. für die Aktivitätsweitergabe nach oben zuständig ist)
- Ein Trainingsbeispiel (= Instanz vom Ziel-concept)

Die gesamte Lernphase wird im Folgenden in Teilphasen erklärt, diese Teilphasen laufen in Wirklichkeit aber parallel ab!

A Recognition-Phase

1. Trainingsbeispiel aktivieren
   Dadurch werden auch das zugehörige concept unit und evtl. zusätzliche feature units aktiviert.
2. Ziel-concept unit aktivieren
   Dadurch werden auch die zum Ziel-concept gehörenden feature units und super-concepts aktiviert (super-concept: concept, das in der Hierarchie darüber steht).
3. Das Netz solange laufen lassen, bis es konvergiert
   = * Alle WTM-Netze der Attribute haben klare Sieger
   * Das Netzwerk ist stabil
   ⇒ * Auch weniger spezifische features (indirekte features) sind aktiv, da sie durch super-concepts aktiviert werden (siehe auch im folgenden Bsp.). Deren Aktivität wird jedoch durch einen Verfallsprozeß gemindert, und zwar stärker als bei den „näheren"(direkten) features.
   Hierbei werden unter „direkten" features solche verstanden, die sich direkt vom Trainigsbeispiel und dem Ziel-concept ableiten lassen, unter „indirekten" features alle anderen (z.B. vom super-concept).
   * Deshalb sind insgesamt die feature units vom Trainingsbeispiel und vom Ziel-concept stärker aktiviert.

B Constructive Generalization

4. Free triple unit für ein neues concept holen
   d.h.
   * Durch die Aktivierung der feature units in 1.-3. werden auch free units aktiviert, und zwar umso stärker, je größer das zufällige Gewicht der Verbindung zu den aktivierten feature units ist.
   * Das free unit mit der größten Aktivierung wird „ausgewählt".
5. Netz laufen lassen, damit sich die Gewichte ändern
   Bei diesem Ansatz wird eine veränderte Hebb-Regel verwendet, die bewirkt, daß sich die Gewichte nur langsam ändern. Zusammen mit dem Verfallsprozeß bedeutet das Folgendes: Die Verbindung zwischen neuem concept und
   * „direkten" features ist stark
   * „indirekten" features ist schwach
   * Ziel-concept ist stark
   * super-concepts des Ziel-concepts ist schwach

   So erreicht man also, was man zu Beginn wollte: es gibt keine direkten Verbindungen zu allgemeinen Eigenschaften des neuen concepts, sondern nur zu speziellen Eigenschaften (siehe auch im folgenden Bsp.)

11.4.4 Ein Beispiel

Hier soll anhand eines Beispiels erklärt werden, wie die Lernphase abläuft. Vergleiche hierzu Abb. 11.9.

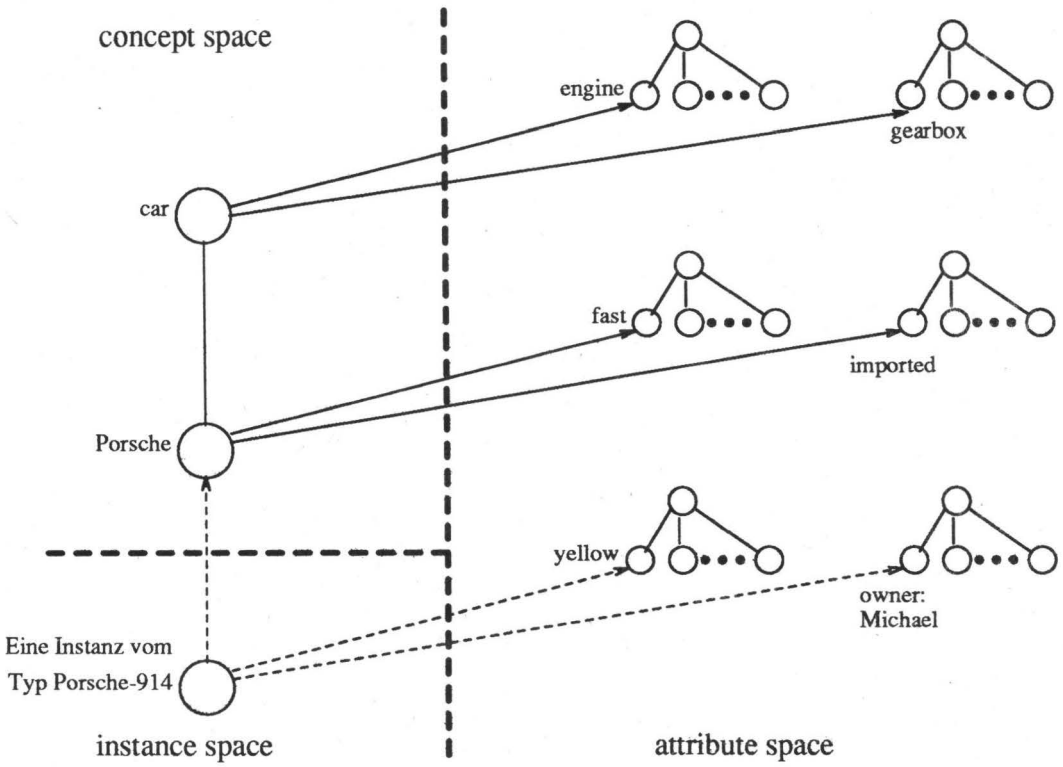


Abbildung 11.9: Das Beispielnetz vor dem Lernen

Es wird ein Netz verwendet, daß die concept-Knoten Porsche und car hat, wobei car das super-concept von Porsche ist. car hat die Attribute engine und gearbox, d.h. car ist mit den entsprechenden feature units verbunden. Porsche hat darüberhinaus noch die Attribute fast und imported. Porsche ist das Ziel-concept. Als Trainingsbeispiel wird jetzt eine Instanz dem Netz präsentiert, ein Porsche-914. Diese Instanz hat die zusätzlichen Attribute yellow und owner: Michael. Das Netz sieht vor der Lernphase also aus wie in Abbildung 12.9.

Wir haben also jetzt alles, was wir zum Lernen brauchen: Ein Ziel-concept, eine domain theory und ein Trainingsbeispiel.

Zu Beginn werden das Ziel-concept Porsche und das Trainingsbeispiel aktiviert. Die Aktivierung des Trainingsbeispiels führt dazu, daß dessen feature units, yellow und owner: Michael, aktiviert werden, außerdem bekommt das Ziel-concept Porsche stärkere Aktivierung. Diese stärkere Aktivierung von Porsche aktiviert ihrerseits auch die features fast und imported stärker. Außerdem wird das super-concept car aktiviert, dieses gibt die Aktivität an seine features engine und gear-box weiter. Durch den schon erwähnten Verfallsprozess wird die Aktivität bei jeder Weitergabe geringer, so daß car eine geringere Aktivität als die direkt „angesteuerten" Knoten Porsche und die Instanz aufweist. Die feature units von car, engine und gear-box, haben so auch eine deutlich geringere Aktivität als die anderen features. Das war eine Beschreibung, was in der Recognition-Phase passiert, die Ergebnisse sind in Abb. 11.10 zu sehen.

In der parallel ablaufenden Phase "Constructive Generalization" wird das free unit bestimmt, welches in Zukunft das neue, spezialisierte concept darstellen soll: Wie schon erwähnt, haben alle free units Verbindungen mit zufälligen, kleinen Gewichten zu den feature units. Die aktivierten

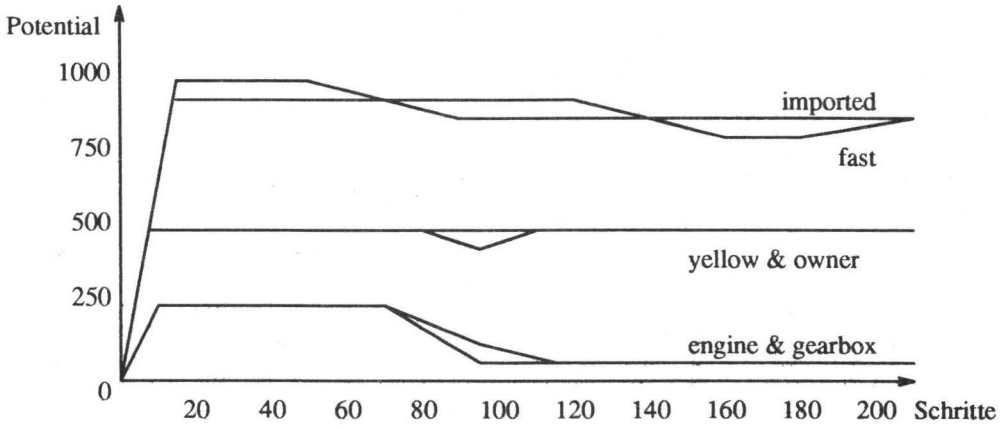


Abbildung 11.10: Zeitliche Anpassung vom Potential im Beispiel

feature units aktivieren jetzt ihrerseits über diese Verbindungen units aus dem free space. Es erhalten jetzt diese free units die meiste Aktivität, die die größten, zufälligen Verbindungen zu den „richtigen" feature units haben. Das free unit, das insgesamt die meiste Aktivität hat, wird ausgeähnlt und könnte in Zukunft als Porsche-914 interpretiert werden.

Durch die abgewandelte Hebb-Regel, die bewirkt, daß sich die Gewichte nur langsam ändern, ist es möglich, daß die Aktivität, die die feature units zu Beginn haben, sich insgesamt schwächer auf die Gewichtsbildung zwischen free und feature units auswirkt. D.h. daß es für die Ausbildung der Gewichte entscheidender ist, wie hoch die Aktivität der feature units über den ganzen Zeitraum gesehen ist. Dies führt schließlich dazu, daß die Gewichte zwischen dem neuen concept unit und den feature units fast und imported stark ist, zu yellow und owner:Michael mittel und zu engine und gear-box schwach ist.(s.Abb. 11.11)

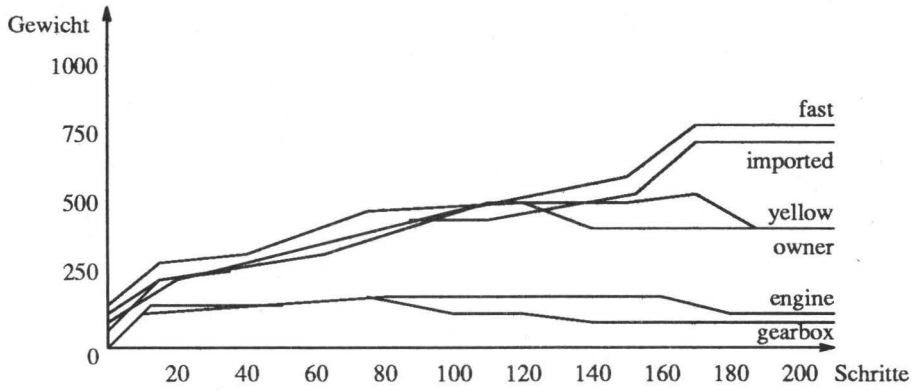


Abbildung 11.11: Zeitliche Anpassung vom Gewicht im Beispiel

Als Ergebnis des Lernvorgangs ist festzuhalten, daß das neuronale Netz jetzt spezielleres Wissen hat und mit diesem einfacher solche Eingaben erkennen kann, die dem Trainingsbeispiel ähnlich sind. Der Sinn ist bei dem gewählten Beispiel wohl z.T. weniger zu erkennen, da "owner: Michael" z.B. nicht sehr häufig die Eigenschaft eines Autos sein dürfte. Sinnvoller ist da die Eigenschaft "yellow". Nach dem Lernvorgang kann das Netz nun bei Aktivierung des feature units "yellow" leichter erkennen, daß es sich dabei um einen Porsche-914 handelt (das Beispiel ist natürlich nur sinnvoll, wenn man davon ausgehen kann, daß die meisten Porsche-914 gelb sind). Erkennen heißt

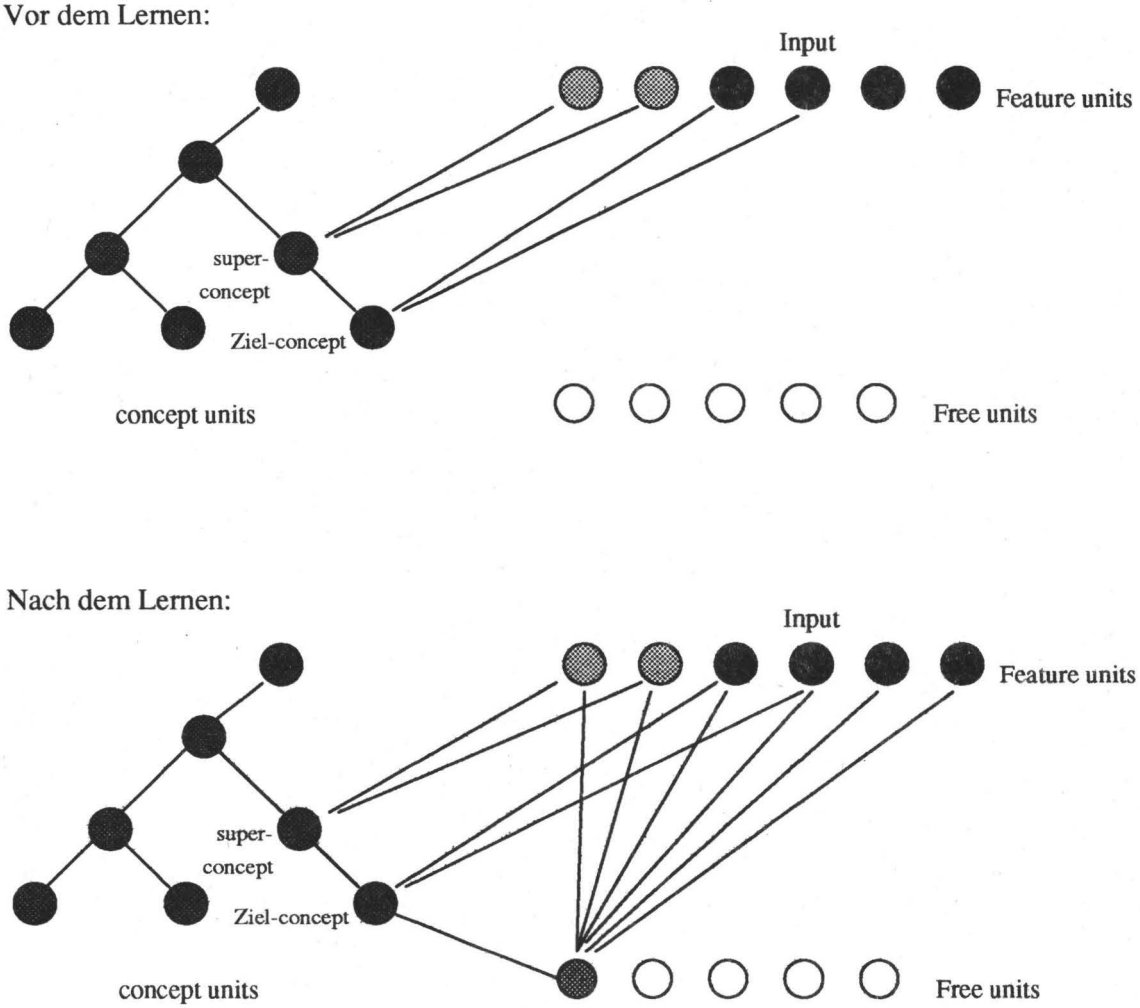


Abbildung 11.12: Veränderung des Netzes

hier, daß man ein möglichst gut zutreffendes concept findet (also hier das concept "Porsche-914"), das dann aktiv wird.

In Abbildung 12.12 ist zu erkennen, daß nach dem Lernvorgang feature units, die vorher irrelevant waren (ganz rechts), jetzt zu den speziellen Eigenschaften des neuen concepts gehören. Dabei ist unter Input zu verstehen, daß diese feature units durch das Ziel-concept und das Trainingsbeispiel aktiviert wurden und so als Input für den eigentlichen Lernvorgang zur Verfügung stehen.

11.5 Zusammenfassung

Semantische Netze sind eine sehr flexible Möglichkeit, komplexes Wissen darzustellen. Dabei ist der Einsatz innerhalb der KI gut möglich. Bisherige Realisierungen hatten jedoch den Nachteil, daß sie z.B. sehr zeitaufwendig sind. Mit neuronalen Netzen kann man dieses Manko aber beseitigen, da der parallele Ansatz sehr zeitsparend ist. Zudem hat es den Vorteil, daß man auch „schwammiges" Wissen, wie es in der Realität meist gegeben ist, noch einigermaßen einfach darstellen kann.

Der erste vorgestellte Ansatz (Abschnitt 12.3) erlaubt es jedoch nur, einmal bereitgestelltes

Wissen als neuronales Netz darzustellen. Dies wirft meiner Meinung nach aber einige Probleme auf:

- der Aspekt der hardware-mäßigen Realisierung ist unzureichend geklärt; insbesondere der enorme Aufwand der Verdrahtung bzw. der Zuweisung von Gewichten an jede Verbindung durch den Netzwerkcompiler erscheint mir zuwenig berücksichtigt.
- die Eingabe aller Daten aus einer Teilwelt ist sehr aufwendig; auch durch Lernen wird das Problem nur unzureichend gelöst

Der zweite Ansatz (Abschnitt 12.4) beschäftigt sich damit, in ein semantisches, neuronales Netz spezielleres Wissen zu bringen. Dafür wird ein Traingsbeispiel benötigt, von dem die Eigenschaften bei dem spezielleren concept übernommen werden. Dies stellt meiner Meinung nach aber den Sinn des Verfahrens in Frage: Statt daß man eine Instanz mit bestimmten Eigenschaften eingibt, könnte man genausogut direkt irgendein free unit auswählen und dessen Verbindung zu den gewünschten Eigenschaften stärken.

11.6 Literatur

[1] Lokendra Shastri. *Semantic Networks: An Evidential Formalization and its Connectionist Realization.* Pitman Publishing, London, 1988.

[2] Joachim Diederich. *Knowledge-Intensive Recruitment Learning.* International Computer Science Institute, Berkeley, 1988.

Kapitel 12

Neuronale Netze in Hybriden Systemen

Markus Schmitz

Übersicht

Die Eigenschaften symbolischer und subsymbolischer Wissensverarbeitung ergeben sich direkt aus der Art der Wissensrepresentation. Dabei erweisen sich die Vor- und Nachteile von neuronalen Netzen und regelbasierten Systemen als komplementär. Deshalb ist es nur natürlich zu versuchen, durch Kombination beider Arten von Wissensverarbeitung in hybride Systeme die Vorteile zu vereinen und die Nachteile zu vermeiden. Da verschiedene Ansätze der Kombination denkbar sind, soll an Hand von drei verschiedenen beispielhaften hybriden Systemen deren Eigenschaften und Grenzen untersucht werden.

12.1 Einleitung

In der Praxis erweisen sich die Vor- und Nachteile von neuronalen Netzen und regelbasierten Systemen als komplementär. Dies soll durch folgende Tabelle veranschaulicht werden.

	regelbasierte Systeme	neuronale Netze
Wissen	formal, explizit in Regeln und Fakten	numerisch in den Gewichten
Wissensaquisation	durch Analyse (nicht trivial, da Erfahrungswissen schwer formalisierbar)	durch Vorgabe von (repräsentativen) Beispielen
Lernverhalten	gering bis nicht vorhanden	gut
Robustheit	schlecht (geringe Fehler in der Eingabe oder in einer Regel oder das Fehlen einer Regel katastrophal)	gut (einzelne Fehler einer Komponente oder beim Lernen haben kaum Auswirkungen)
Strukturverhalten	stark	schwach
Assoziativverhalten	schwach	stark
Selbsterklärung	gut (durch Verfolgen der Inferenz)	schlecht (große Teile des Netzes beeinflussen das Ergebnis)
Parallelverarbeitung	schwierig	leicht (bietet sich auf natürliche Weise an)
Anwendungsgebiete	gut erforscht (geeignet für Konfigurierung, Design, Planung, Interpretierung)	kaum erforscht (fehlende Theorie welches Netz für welches Problem geeignet)

Durch die Wahl der Wissensrepräsentation ergeben sich die anderen Eigenschaften (fast) zwangsläufig.

Die Art der Wissensaquisation erweist sich deshalb für beide Ansätze als Grenze. Während es sich bei einigen Problemstellungen als unmöglich erweist Erfahrungswissen, z.B. das Erkennen eines fehlerhaften Motors am Geräusch, zu formalisieren, kann das Finden repräsentativer Beispiele zum Trainieren des neuronalen Netzes zu einer ebenso großen Hürde werden.

Beim Lernverhalten und der Robustheit sind regelbasierte Systeme klar im Nachteil. Das schlechte Lernverhalten wirkt sich dabei in zwei Punkten aus. Einerseits kann bei Änderungen der Umgebung keine Anpassung stattfinden und andererseits findet trotz wachsender Erfahrung keine Steigerung der Leistung statt, da einmal gelöste Probleme immer wieder neu gelöst werden müßen.

Unter Strukturverhalten versteht man die Fähigkeit Strukturen festzuhalten und wiederzugeben. Hier ist das regelbasierte System natürlich im Vorteil, da diese Strukturen direkt als Regeln festgehalten werden können. Assoziativverhalten dagegen ist die Fähigkeit ähnliche Eingaben auf ähnliche Ausgaben abzubilden. Für solche Klassifizierungen werden neuronale Netze momentan am meisten eingesetzt.

Offensichtlich sind neuronale Netze auf den Gebieten stark, wo regelbasierte Systeme Schwächen zeigen und umgekehrt. Es bietet sich also an beide Modelle der Wissensverarbeitung zu integrieren und so möglichst viele Stärken zu erben und Schwächen zu eliminieren.

Eine solche Kooperation kann erfolgreich sein, wie Untersuchungen an menschlichen Experten [Eli 81, Mon 89] zeigen. Danach verlassen sich Experten zum großen Teil auf Erfahrungswissen, häufig auf eine Mischung aus (formalisierten) Sachbuch- und Erfahrungswissen, nur in Ausnahmen vollends auf gelernte Regeln. Anleihen aus der Psychologie [Day] machen diese Analogie noch deutlicher. Hier wird zwischen automatisierten und kontrolliertem Verhalten unterschieden. Automatisiertes Verhalten ist charakterisiert durch eine hohen Grad der Paralellität, Verbesserung der Leistung durch Training und einer geringen Kontrolle des Problemlösers. Kontrolliertes Verhalten zeichnet sich durch Seriellität, geringe Verbesserung mit Übung und Kontrolle vom Problemlöser aus. Automatisiertes Verhalten zeigt also Parallelen zu neuronalen Systemen und kontrolliertes Verhalten zu regelbasierten Systemen. Laut Erfahrung arbeiten diese beiden Verhaltensmuster aber eng zusammen. Viele kontrollierte Verhalten werden mit steigender Praxis automatisiert. Dadurch werden anfangs komplexe und langsame Arbeiten (scheinbar) leicht und schnell. Als Beispiel hierfür mag der Schaltvorgang im Auto dienen. Der Vorgang "Gas lösen, Kupplung treten,

Schalten, Kupplung lösen" wird anfangs konzentriert nacheinander und schwerfällig ausgeführt und mit der Zeit wie von selbst erledigt, ohne bewußt (also kontrolliert) darüber nachzudenken. Das heißt kontrolliertes geht zu automatisiertem Verhalten über. Aber auch der Übergang vom automatisierten zum kontrollierten Verhalten ist denkbar. So werden zum Beispiel bei der Beweisfindung Hypothesen intuitiv (also unkontrolliert) gefunden und müßen dann mit Hilfe von Formeln und Regeln erschlossen werden.

12.2 Bekannte Ansätze hybrider Systeme [Goo]

1. Ausgehend von der Tatsache, daß beim Menschen alle kognitiven Vorgänge subsymbolisch durch Neuronen realisiert sind, wird versucht, die Konzepte der Symbolverarbeitung in neuronalen Modellen zu implementieren. Schon bei einfachen Konzepten wie Variablenbindung und Inferenz stößt man dabei auf Schwierigkeiten.

2. Die vielen Erfahrungen auf dem Gebiet der neuronalen Netze in den Vergangenen Jahren führten zur Idee, charakteristische Eigenschaften dieser Modelle in Expertensysteme mit aufzunehmen. Dabei werden diese Ideen aber mit der klassischen Methode der symbolischen Informationsverarbeitung realisiert und keine neuronalen Netze explizit verwendet.

3. In einem dritten Ansatz werden Modelle untersucht, in denen sowohl Expertensysteme als auch neuronale Netze als Komponenten eines komplexeren Systems zusammenarbeiten. Diese Komponenten können sowohl sequentiell als auch parallel angeordnet werden. Einerseits können die von den neuronalen Netzen gelieferten Resultate als Eingaben für den regelbasierten Teil verwendet werden (und vice versa). Man spricht in diesem Fall von hierarchischer Ordnung. Andererseits besteht die Möglichkeit, daß Expertensysteme und neuronale Netze auf dem gleichen Hierachieniveau zusammenarbeiten und sich dabei gegenseitig unterstützen und kontrollieren.

Im Folgenden sollen für den Punkt i und für die beiden Ordnungsmöglichkeiten in Punkt iii Beispielsysteme im Detail beschrieben und deren (hoffentlich verbesserten) Eigenschaften untersucht werden. Punkt ii wird nicht weiter behandelt, da keine neuronalen Netze verwendet werden.

12.3 Symbolic Connectionist Net: SC-Net

12.3.1 Einführung von Symbolverarbeitung in ein connectionistisches Modell

Die Intention für diesen hybriden Ansatz lag darin, regelbasierte Expertensysteme mit einer ähnlichen Lernfähigkeit auszustatten, wie NN sie besitzen. D.h. es sollte von Beispielen gelernt werden. Zusätzlich sollte das System Konzepte beherrschen können, die mit Ungenauigkeit behaftet sind, und Variablen und relationale Operatoren besitzen. Hierdurch sollen Standard-Regel-Basen in diese Repräsentation kodiert werden können.

Als Modell liegt ein (neuronales) Netz, SC-Net genannt, zu Grunde. Es besteht aus drei Arten von Zellen (input,output und hidden Cells). Jede Zelle ist mit einem Vorwert (Bias) behaftet und die Zellen sind untereinander gewichtet verbunden. Jede Zelle kann einen Aktivierungswert (activation value) zwischen 0 und 1 annehmen. Dies steht im Einklang zur Fuzzy Set Theorie, die Anwendung findet, um Ungenauigkeiten behandeln zu können. Entsprechend können Zellen im Netz als fOR (Maximum), fAND (Minimum), oder fNOT (strenge Negation) fungieren. Welche Operation eine Zelle realisiert, wird durch das Vorzeichen ihres Bias indiziert:

$$\begin{aligned} \text{Bias} < 0 &\quad\leftrightarrow\quad \text{fuzzy AND} \\ \text{Bias} > 0 &\quad\leftrightarrow\quad \text{fuzzy OR} \\ \text{Bias} = 0 &\quad\leftrightarrow\quad \text{fuzzy NOT} \end{aligned}$$

Faßt man dies zusammen, so berechnet sich ein Aktivierungswert CA_i einer Zelle C_i durch seinen Bias CB_i und die Gewichte $CW_{i,j}$ mit verbundenen Zellen C_j wie in Abb. 12.1.

$$
\begin{aligned}
CA_i &= \text{Aktivierungswert der Zelle} C_i, CA_i \in [0,1] \\
CW_{i,j} &= \text{Gewicht der Verbindung der Zellen } C_i \text{ und } C_j, CW_{i,j} \in \mathrm{R} \\
CB_i &= \text{Vorwert (bias) der Zelle} C_i, CB_i \in [-1,+1]
\end{aligned}
$$

$$
CA_i = \begin{cases} \min\limits_{j=0,\ldots,i-1,i+1,\ldots,n} (CA_j \cdot CW_{i,j}) \cdot |CB_i|, & CB_i < 0 \\ \max\limits_{j=0,\ldots,i-1,i+1,\ldots,n} (CA_j \cdot CW_{i,j}) \cdot |CB_i|, & CB_i > 0 \\ 1 - (CA_i \cdot CW_{i,j}), & CB_i = 0 \text{ und } CW_{i,j} \neq 0 \end{cases}
$$

Abbildung 12.1: Berechnung der Aktivierungswerte

Neben den Zellen, die als Fuzzy-Opeartoren wirken, existiert noch eine weitere Art von Zellen, genannt 'ltc' (linear threshold cell). Zellen, die beim Aufbau des Netzes als 'ltc' deklariert werden, addieren schlicht gewichtet die Aktivierungswerte der verbundenen Zellen. D.h. es gilt dann :

$$CA_i = \sum_{j=0,\ldots,n} CA_j \cdot CW_{i,j}$$

Diese speziellen Zellen werden dann später zur Realisierung der relationalen Operatoren benutzt.

Ragt der berechnete Aktivierungswert aus dem erlaubten Bereich [0,1] heraus, so wird er entsprechend abgeschnitten. Eine Aktivierung von 0 deutet dann falsch, 0.5 unbekannt und 1 wahr an.

12.3.2 Übersetzung von Regeln in ein SC-Netz

Dies mag am besten durch ein Beispiel verdeutlicht werden. Es sei folgende Regel zu übersetzen:

if and(or(s1,s2),s3) then d1 (0.8)

Nur wenn die Vorbedingung mit 1.0 berechnet wird, sei das Ergebnis gleich 0.8, in allen anderen Fällen graduell geringer. Die Übersetzung in ein SC-Net ist in Abb. 12.2 dargestellt.

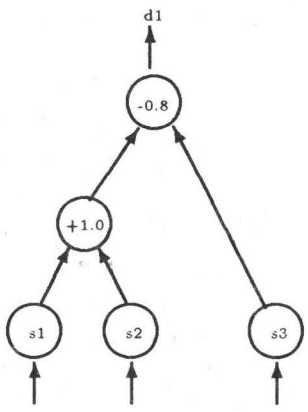


Abbildung 12.2: SC-Netz der übersetzten Regel

Indem einzelne Regeln zu Teilnetzen übersetzt und diese dann zusammengesetzt werden, lassen sich vollständige Netze konstruieren. (Haken: dies ist das einzige Mittel zur Inferenz)

12.3.3 Lernen und Modifizieren

Da nun ein connectionistisches Modell zu Grunde liegt, besteht auch die Möglichkeit die SC-Netze aus Beispielen lernen zu lassen. Dabei kann sowohl erst eine Regelbasis transformiert und später inkrementell hinzugelernt werden, als auch von Anfang an nur das Netz trainiert werden. Der vom Author propagierte Lernalgorithmus "Recruitment of Cells Learning Algorithm" gleicht bei bekannten Konzepten Gewichte an und fügt bei unbekannten Konzepten neue Zellen hinzu. Das Netz wächst also mit der Anzahl gelernter verschiedener Konzepte. Näheres siehe [1].

12.3.4 Implementierung von Variablen

Variablen erweisen sich für größere XPS als sehr nützlich. Als ebenso schwierig erweist es sich jedoch, solche in ein connectionistisches Modell auch zu implementieren. Beim SC-Netz ist dies jedoch möglich. Es werden dabei zwei Arten von Variablen unterschieden, von denen eine, die Fuzzy Variablen, ausführlich erläutern werden soll.

Fuzzy Variablen ermöglichen es den numerischen Bereich einer Variablen in sein unscharfes Äquivalent zu unterteilen.

Beispiel: Alter einer Person $\in [0, 100]$
Die folgenden Attribute seien mit dieser Variablen 'Alter' verbunden:

Kind: 0-12 (0-16)
Teenager: 13-19 (5-25)
Jung: 0-30 (0,34)

Wird dem 'Alter' also ein Wert von 15 zugewiesen, so soll das Attribut 'Teenager' ganz wahr und das Attribut 'Kind' noch ein wenig wahr sein. Die Nummern in Klammern geben also den Bereich an, indem noch ein gewisser Wahrheitsgrad anerkannt wird, der aber zu den Rändern hin abnimmt und außerhalb der Ränder 0 ist. Die Repräsentation im SC-Net erfolgt durch ein Teilnetz (Siehe Abb. 12.3). Das Alter im Beispiel wurde in den Bereich [0,1] kompremiert, d.h. ein Alter von 15 entspricht 0.15.

Allgemein liegt folgendes Format vor:

`attribut wert:untere,obere Grenze (unterer, oberer Eckwert)`

Dann gilt für die Gewichte in Abb. 12.3:

$$
\begin{aligned}
a &= \frac{1}{oberer Eckwert} = \frac{1}{0.25} = 4 \\
b &= 1 \\
c &= \frac{oberer Eckwert}{oberer Eckwert - obere Grenze} = \frac{0.25}{0.25 - 0.19} = 4.167 \\
d &= \frac{1}{1 - unterer Eckwert} = \frac{1}{1 - 0.05} = 1.053 \\
e &= \frac{1 - unterer Eckwert}{untere Grenze - unterer Eckwert} = \frac{1 - 0.05}{0.13 - 0.05} = 11.875
\end{aligned}
$$

12.3.5 Fuzzy und relationale Operatoren

Relationale Operatoren (Ergebnisse $\in \{0, 1\}$) und Fuzzy Operatoren (Ergebnisse $\in [0, 1]$) sind in XPS ebenso wichtig wie Variablen. Fuzzy Operatoren sind z.B. dann nützlich, wenn man Bedingungen wie 'um 12 Uhr' ausdrücken will. Dies geschieht dann sehr elegant mit fEQUAL(time,12).

Im SC-Netz lassen sich beide Typen von Operatoren realisieren. Als Beispiel diene der Größergleich Operator in Abb. 12.4. Weißt man den Zellen x und y die Werte 0.8 und 0.6 zu, so wird (von unten gesehen) zuerst der größere der beiden Werte mit Hilfe der Max-Zelle (und den Gewichten +1) bestimmt. Dies ist natürlich 0.8. Die mit 'ltc' markierte Zelle wird nun (wegen dem Gewicht -1 der Verbindung zur x-Zelle) x von diesem Wert subtrahieren. Damit erhalten wir entweder eine Wert gleich 0, falls x grössergleich y ist, oder einen Wert grösser als 0. Dieser Wert wird dann mit unendlich (auf einem Computer mit dem größten Integer) multipliziert und negiert. War das Ergebnis der ltc-Zelle gleich 0, wie in unserem Fall, so gibt der Inverter also eine 1 aus. War das Ergebnis ungleich 0, d.h. x kleiner als y, so wird die Multiplikation einen Wert grösser 1 und der Inverter also eine 0 ausgeben.

Durch die Multiplikation mit Unendlich liefert das Netz also diskrete Werte 0 oder 1. Ersetzt man diesen Faktor durch einen endlichen Wert, so liefert das Netz Werte aus [0,1], d.h. ein unscharfes Ergebnis. Dabei bestimmt die Höhe des Faktors, wie schnell der realisierte Fuzzy-Operator fGREATEREQUAL degradiert.

12.3.6 Fazit für SC-Netze

- Die Mittel zur Inferenz fehlen
- Die vorgestellten Teilnetze für Variablen und Operatoren sind alles andere als trivial
- Die Selbsterklärungsfähigkeit der regelbasierten XPS geht deshalb verloren.

12.4 *echte* hybride Systeme

Werden sowohl konventionelle auf symbolischer Verarbeitung basierende Expertensysteme und (subsymbolische) neuronale Netze als Komponenten in einem System integriert, so ergeben sich zwei Möglichkeiten.

1. Die Ausgaben der einen Komponente werden als Eingabe der anderen Komponente verwendet. Es liegt also sequentielle bzw. hierachische Ordnung vor.
2. Die Komponenten kontrollieren und unterstützen sich gegenseitig. Es liegt also eine parallele Ordnung vor.

Für beide Möglichkeiten wird jeweils ein System vorgestellt, das in beiden Fällen an der ETH Zürich entwickelt, implementiert und auf seine Eigenschaften ausgetestet wurden.

12.4.1 HSHSP - Hybrides System für High School Physics

HSHSP wurde nach einer Idee von K.Lamberts an der ETH Zürich von Mathias Gutknecht und Rolf Pfeifer weiterentwickelt. Ziel von HSHSP ist es bestimmte Bereiche von Expertenverhalten zu modellieren. Als Anwendungsbereich wurde High School Physik gewählt. Eine Beispielanfrage könnte dann so lauten:

```
gegeben:   Startgeschwindigkeit,
           Beschleunigung und
           Dauer einer linearen Bewegung

gesucht:   die Endgeschwindigkeit
```

Bei solchen Problemstellungen zeigen sich in der Praxis (High School) zwei grundsätzlich verschiedene Vorgehensweisen. Anfänger neigen dazu Formeln zu suchen, die die Zielgrösse (Endgeschwindigkeit) enthalten und versuchen nacheinander die unbekannten Größen zu ersetzen. Experten/Fortgeschrittene dagegen erkennen intuitiv welche zusätzlich unbekannten Zwischengrössen

für die Lösung sinnvoll sind. Das Anfänger Verhalten wird in HSHSP deshalb durch ein regelbasiertes System modelliert und das Expertenverhalten durch ein neuronales Netz, welches dem regelbasierten Teil Zwischenziele vorschlägt. Die Komponenten arbeiten also eindeutig hierarchisch miteinander.

Die generelle Architektur eines solchen Systems ist in Abb. 12.5 dargestellt. Systeme kooperieren über ein (standartisiertes) Interface. Die Abbildungsdefinitionen (Map Defs) beschreiben die Umformung der symbolischen Variablenrepräsentation des XPS und der Vektorrepräsentation des NN. Abbildungsfunktionen (Map Funcs) interpretieren und führen diese Definitionen aus. Die Funktion "learn" trainiert und die Funktion "suggest" fragt das Netz ab.

Bei HSHSP wurde diese Struktur im Detail so umgesetzt:

Die Regeln im XPS repräsentieren die physikalischen Gleichungen, z.B:

Gleichung: $V = v_o + a \cdot t$
Regel:

```
if    (or   (die Endgeschwindigkeit ist gesucht)
            (die Endgeschwindigkeit ist unbekannt)
      (die Anfangsgeschwindigkeit ist bekannt)
      (die Beschleunigung ist bekannt)
      (die Dauer ist bekannt)
then  (die Endgeschwindigkeit ist bekannt)
      (lisp (print 'Anwendung von v = v0 + a·t'))
```

Die print Anweisung wird für die Rückverfolgung der Inferenz verwendet.

Das NN besteht aus einer "Standard-3-Schichten-feed-forward" Architektur mit Backpropagation-Lernalgorithmus. Bei der Abbildung der Repräsentation im XPS auf das NN werden den Symbolen des XPS jeweils direkt Knoten des NN zugeordnet. Die linke Seite des Input-Vektors deutet dabei die gegebenen Größen an. Bei einer 1 ist die Variable bekannt, bei einer 0 unbekannt. Die rechte Seite gibt die gesuchten Variablen (normalerweise eine einzelne) an. Siehe Abb. 12.6.

Entsprechend werden im Output-Vektor die Größen angezeigt, die als Zwischengrößen zun Ziel führen. Zusätzlich zeigt ein Knoten im Output-Vektor (der Solvability Index) an, ob eine Lösung überhaupt möglich oder das gestellte Problem unterspezifiziert ist. Dieser Index ist 1, falls mindestens eine Zwischengrösse angestrebt oder die Zielgröße direkt aus den gegebenen berechnet werden kann. Hierdurch werden uneffektive Rückwärtsanalysen des XPS vermieden, was allerdings nur bei gut trainierten Netz funktioniert.

Training der NN-Komponente

Statt vor Anwendung von HSHSP (d.h dem Stellen von Anfragen) das Netz mit Beispielaufgaben auf einen Schlag zu trainieren, geschieht dies Schritt für Schritt während des Arbeitens mit HSHSP, d.h. das XPS enthält alle Regeln und Inferenzmethoden, um eine Anfrage allein zu lösen und während dieses Lösungsprozesses wird ein Stack mit den (erfolgreichen) Zwischengrößen aufgebaut. Dieser Stack repräsentiert dann einen (von wahrscheinlich mehreren möglichen) erfolgreichen Lösungswegen. Beim Trainieren des NN wird dieser Stack nun wieder stückweise abgebaut, d.h. das oberste Stackelement wird als anzustrebende Zwischengröße angesehen und auf den Target-Vektor mit Solvability Index 1 abgebildet. Die Anfangsgrößen zusammen mit den darunterliegenden Zwischengrößen werden als die bekannten Größen auf den Input-Vektor abgebildet.

Beispiel:

Anfrage:	gegeben sei m,F,t,v_0
	gesucht ist v
das XPS findet:	v = $v_0 + v_a$
	$v_a = a \cdot t$

a = F/m
aufgebauter Stack: (v , v_a , a)

vom NN gelernte Vektoren:

Input-Vektor								Target-Vektor							
v	v_0	v_a	t	m	F	a		v	v_0	v_a	t	m	F	a	Index
0	1	1	1	1	1	1		1	0	0	0	0	0	0	1
0	1	0	1	1	1	1		1	0	0	0	0	0	0	1
0	1	0	1	1	1	0		1	0	0	0	0	0	0	1

															1
															1
															1

Fazit für HSHSP

Durch den verwendeten Ansatz ergeben sich einige Vorteile

- Der Arbeitsschwerpunkt des Systems wechselt im Laufe der Zeit vom XPS zum NN, d.h. die Performance des Gesamtsystems steigert sich mit wachsender Erfahrung.
- Selbst wenn das NN später die Hauptarbeit leistet, so bleibt das XPS als Erklärungskomponente erhalten.
- Es müssen keine repräsentativen Beispiele zum Trainieren gefunden werden, sondern das gut strukturierte Wissen kann in diesen Strukturen auch in HSHSP eingebracht werden.

Leider ist dieser Ansatz wegen der strengen hierarchischen Anordnung nur für wenige Probleme anwendbar. In der Kooperation etwas weiter geht das folgende System NESSY und ist so auch flexibler und für mehr Probleme einsetzbar.

12.4.2 NESSY - Neural Network and Symbolic Reasoning System

Kombination durch Kooperation

NESSY stützt sich auf ebenbürtiger Zusammenarbeit von NN und XPS und folgt diesen 6 Grundgedanken:

1. Nessy soll Klassifikationsprobleme lösen. Zu konstruieren ist also eine Funktion $f : X_1..X_n \rightarrow Y$ mit $X_i, Y = [0,1]$ oder $\{0,1\}$
2. Diese Aufgabenstellung wird nun in verschiedene, z.T. sich überlappende Teilaufgaben zerlegt.
3. Diese Teilaufgaben werden je nach Eignung der regelbasierten, der neuronalen oder beiden Komponenten zugeordnet
4. Bei überlappenden oder gleichen Teilaufgaben sollen sich die Komponenten gegenseitig kontrollieren.
5. Bei allen anderen Aufgabenstellungen sollen sie sich unterstützen.
6. Beide Komponenten können von den Antworten der anderen Komponente bzw. des Benutzers lernen, d.h. das XPS ändert seine Regelbasis und das NN seine Gewichte.

Die Verteilung der Aufgaben (Punkt 3) richtet sich nach menschlichem Vorbild, d.h. alle vom Experten sehr schnell ausgeführten Abklärungen, die auf assoziative erfahrungsbasierte Vorgehensweisen beruhen, werden dem NN zugeordnet. Dem XPS dagegen werden die Teilaufgaben zugeordnet, die auch beim menschlichen Denken durch bewußtes gedankliches Vorgehen gelöst werden.

Aufgaben der regelbasierten Komponente

- die Klassifikation selbt
- die Aktualisierung der Regelbasis.

Aufgaben der NN-Komponente

Die NN-Komponente besteht aus mehreren NN verschiedenen Typs, die je für eine der folgenden Aufgaben zuständig sind:

- Das Klassifikations-Netz löst die gleichen Aufgaben wie das XPS und lernt so von den Ein-/Ausgabedaten desselben *und* von externen Erfahrungswissen. Es kann sich also vom XPS in seinem Verhalten unterscheiden.
- Das Relevanz-Test-Netz hebt *die* Teile der Eingabedaten hervor, die für die Lösung relevant sind und schränkt so den Suchraum für das Expertensystem ein.
- Das Eingabe-Vervollständigungs-Netz vervollständigt aus Erfahrungswerten unvollständige Daten
- Das Ausnahme-Netz prüft, ob ein üblicher Fall vorliegt und teilt dies dem XPS oder dem Benutzer mit.
- Das Neue-Regeln-Netz registriert, ob Ausnahmen gehäuft vorkommen und sich so neue Regeln herleiten lassen.
- Das Mehrdeutigkeitsnetz soll lernen die richtige Wahl zu treffen, falls das XPS mehrere Lösungen anbietet.
- Das Regelauswahlnetz soll lernen zu wählen, wenn während der Inferenz mehrere Regeln zur Anwendung zur Verfügung stehen.

Kooperation der Komponenten

- gegenseitige Kontrolle: Das XPS und das Klassifikations-Netz haben beide die Aufgabe der Klassifikation. Stimmen die Resultate nicht überein, d.h. die regelbasierte Lösung stimmt nicht mit der bisher gesammelten Erfahrung überein, so werden diese Resultate dem Benutzer zur Begutachtung vorgelegt.
- gegenseitige Unterstützung: Die meisten Netze sind im wesentlichen zur Unterstützung des XPS da. Beispiele:
  - findet das XPS keine Lösung, da die Eingabedaten unvollständig sind, so dient das Vervollständigungs-Netz zur Unterstützung
  - meldet das Ausnahme-Netz Verdacht auf einen Spezialfall, der natürlich nicht in den Regeln enthalten ist, so wird dies dem Benutzer mitgeteilt.

Abb.12.7 stellt diesen Zusammenhang grafisch dar.

Lernen der Komponenten

Beide Komponenten sollen lernfähig sein, d.h. ihre Wissensbasis revidieren können. Die Netze lernen dabei aus den Ein-/Ausgaben des XPS und direkt aus den Erfahrungswerten nach folgenden Muster:

1. dem System werden Eingabedaten geliefert
2. das XPS versucht eine Lösung zu finden

3. der Benutzer begutachtet die Situation
4. der Lernvorgang beginnt
   - Das Klassifikations-Netz lernt die Ein-/Ausgabedaten
   - das Relevanz-Netz lernt welche Eingabedaten für die Auswertung relevant waren
   - das Vervollständigungsnetz lernt vollständige Eingabetupel
   - das Ausnahme-Netz lernt den Fall, falls der Benutzer ihn als Ausnahme deklariert

Fazit für NESSY

NESSY geht in der Kooperation wesentlich weiter als HSHSP. Dies bedeutet:

- Wie bei HSHSP dienen neuronale Netze dem XPS zur Unterstützung
- Dadurch kann formalisiert vorliegendes Wissen direkt eingebracht werden.
- Neben den "dienenden" NN, existieren NN auf gleicher Hierachiestufe, die mit dem XPS konkurrieren.
- Dadurch ergibt sich nicht nur eine quantitative Verbesserung der Performance des Systems, sondern auch eine qualitative, da NESSY schlecht formalisierbares Wissen lernen und auf Änderungen in der Umgebung reagieren kann. Dies ist besonders bei technischen Systemen entscheidend.

Leider wurde das Projekt NESSY an der ETH Zürich vorzeitig abgebrochen, so daß keine Erfahrungen der Leistungen von NESSY bei einer konkreten Problemanwendung vorliegen.

12.5 Fazit für Hybride Systeme

Das Konzept der SC-Netze hebt einige der Beschränkungen "normaler" neuronaler Netze auf. Dabei ist vor allem- die einfache Kodierung von Regeln, aber auch die Möglichkeit Variablen und relationale Operatoren zu verwenden interessant. Allerdings verhindert das Fehlen von jeglichen Mitteln zur Inferenz den Einsatz von SC-Netzen für komplexere Probleme. Zudem geht das Verständnis der Netze bei Verwendung von Variablen zu schnell verloren.

Weitaus erfolgversprechender scheint da der Ansatz von HSHSP und NESSY. Bei beiden Systemen kooperieren neuronale Netze mit einem regelbasiertem Teilsystem. So kann bei beiden Systemen schon formalisiert vorliegendes Wissen auch in dieser formalen Form in das XPS eingebracht werden. Bei HSHSP ermöglicht dies dem neuronalen Netz den Zugriff auf solches Wissen, ohne vom Anwender gezielt mit Beispielen trainiert zu werden. Hierdurch geht die Arbeitsleistung mit der Zeit auf die neuronale Komponente über und die Performance des Systems steigert sich mit wachsender Erfahrung. Bei NESSY ist dies ebenfalls der Fall. Die neuronalen Netze lernen jedoch nicht nur vom Verhalten des XPS, sondern auch vom Benutzer. Hierdurch läßt sich schwer formalisierbares Wissen, wie z.B. Ausnahmen und Umgebungsänderungen, ebenfalls in das System integrieren. NESSY ist hier also HSHSP überlegen. Das endgültige Leistungsvermögen von NESSY ist allerdings erst in konkreten Problemanwendungen, die bis heute fehlen, überprüfbar.

12.6 Literatur

[1] Steve G. Romanuik, Lawrence O. Hall, 'Injecting Symbol Processing Into a Connectionist Model' aus Branko Soucek, *Neural and Intelligent System Integration*

[2] Henk Goorhuis u.a., *Neuronale Netze und regelbasierte Systeme: Ein hybrider Ansatz*, Departement Informatik, ETH Zürich

[3]] Matias Gutknecht, Rolf Pfeifer, *Approach to Integrating Expert Systems with Connectionist Networks*, AI Lab, Institut für Informatik, ETH Zürich

[4] David S. Day, *Towards Integrating Automatic and Controlled Problem Solving*, Experimental Knowledge Systems Laboratory, University of Massachusetts

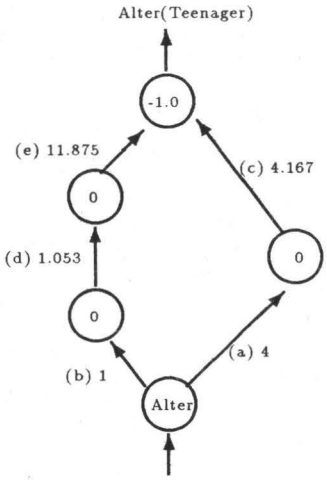


Abbildung 12.3: SC-Teilnetz für die Fuzzy Variable 'Alter'

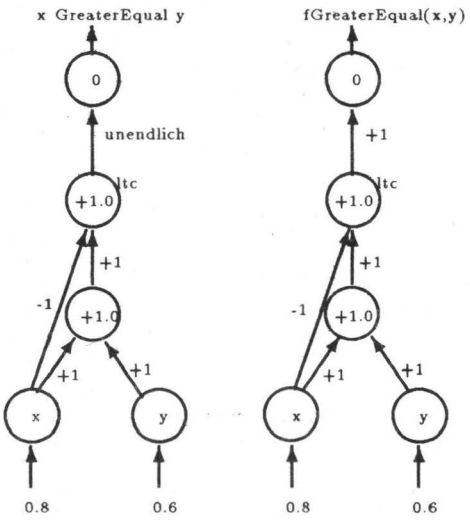


Abbildung 12.4: SC-Netze für GrößerGleich - Operatoren

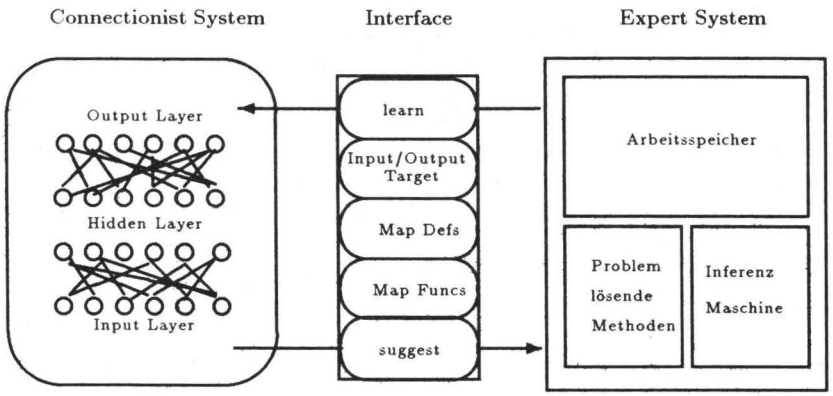


Abbildung 12.5: Architektur eines hybriden Systems

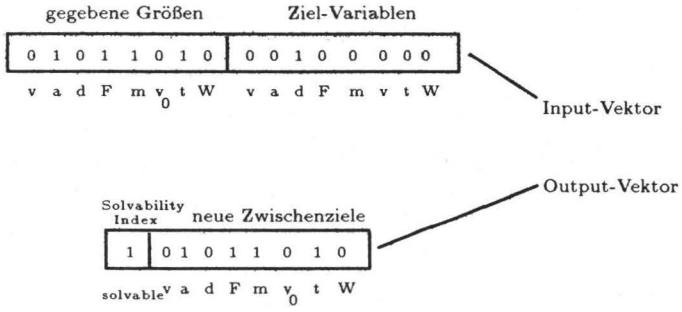
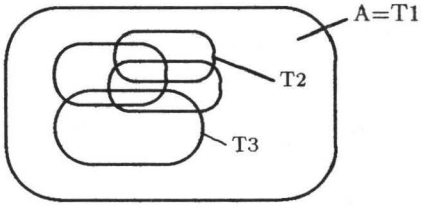


Abbildung 12.6: Beispiele für Input- und Output-Vektoren



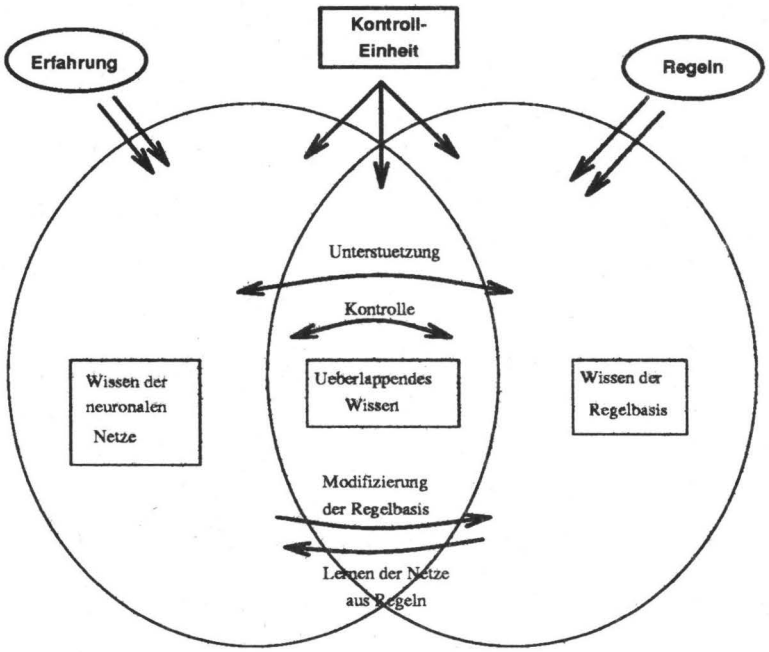


Abbildung 12.7: Kooperation der beiden Komponenten

Kapitel 13

Wissensbasierte neuronale Netze

Kenan Çarki

Übersicht

Die zwei Arten des Maschinenlernens, regelbasiertes und empirisches Lernen, haben beide einige Schwächen, welche es erschweren, sich für eine als Universal-Lernmethode zu entscheiden. Es hat sich gezeigt, daß die Schwächen zueinander komplemetär sind. Wenn nun eine Lernmethode diese beiden kombiniert, und die Schwächen der einen Methode durch die Stärken der anderen kompensiert, ist sie überlegen.

Das hybride Lernsystem, welches hier betrachtet wird, hat als Grundlage neuronale Netze, die aufgrund von gegebenen Regeln erstellt werden. Es heißt KBANN (Knowledge-Based Artificial Neural Networks). Diese Lernmethode wird aufgrund der Eigenschaft, daß sie Regeln aus speziell trainierten Netzwerken extrahieren kann, als Regelverbesserungssystem verwendet.

13.1 Einleitung

Künstliche neuronale Netze stellen eine besondere Methode des empirischen Lernens dar. Es wurde schon bewiesen, daß sie anderen empirischen Lernverfahren ebenbürtig, wenn nicht sogar überlegen, sind [Atlas89, Fisher89, Shavlik91, Weiss89]. Aber man hat es bei der Benutzung von neuronalen Netzen auch mit einigen für sie typischen Problemen zu tun. Das eine sind die langen Trainingszeiten, die im Vergleich zu einigen symbolischen Lernalgorithmen wesentlich größer sind. Desweiteren können die Initialisierungsparameter des Netzwerkes das Lernverhalten doch sehr beeinflussen. Weiterhin stellt die Wahl der Netzwerktopologie ebenfalls ein Problem dar.[1,2]

Aber das Hauptinteresse gilt hier der Tatsache, daß nachdem ein neuronales Netzwerk trainiert wurde, eine „black box" vorliegt. Zwar bekommt man zu den Eingaben, mit einer gewissen Fehlertoleranz, die gewünschte Ausgabe, aber keine Begründung für das Ausgabeverhalten. Hier nun ist es wünschenswert, die Extraktion von passendem, verständlichem Symbolwissen zu erhalten.

Diese Abhandlung wird ein Lernsystem stellvertretend als Beipiel vorstellen, welches auch in der Lage ist, aus einem neuronalen Netz Regeln zu ziehen. Einschränkend ist jedoch zu erwähnen, daß es zuerst aus gegebenen Regeln ein wissensbasiertes neuronales Netz erstellt und nach einer Verbesserung dessen, nur aus diesem Regeln extrahieren kann. Aber nichtsdestotrotz bringt es Licht in die sonst so große Undurchsichtigkeit des Netzwerkwissens und ist daher einer Betrachtung wert.

13.2 KBANN-Knowledge-Based Artificial Neural Networks

KBANN [1] ist ein System, welches sowohl aus Regeln, als auch aus Daten lernt durch Kombinierung der Aspekte von regelbasierten und empirischen Lernsystemen. Dieses hybride Lernsystem

verfeinert symbolisches Wissen, d.h. gegebene Regeln werden verbessert, und als Werkzeug dazu dienen neuronale Netze.

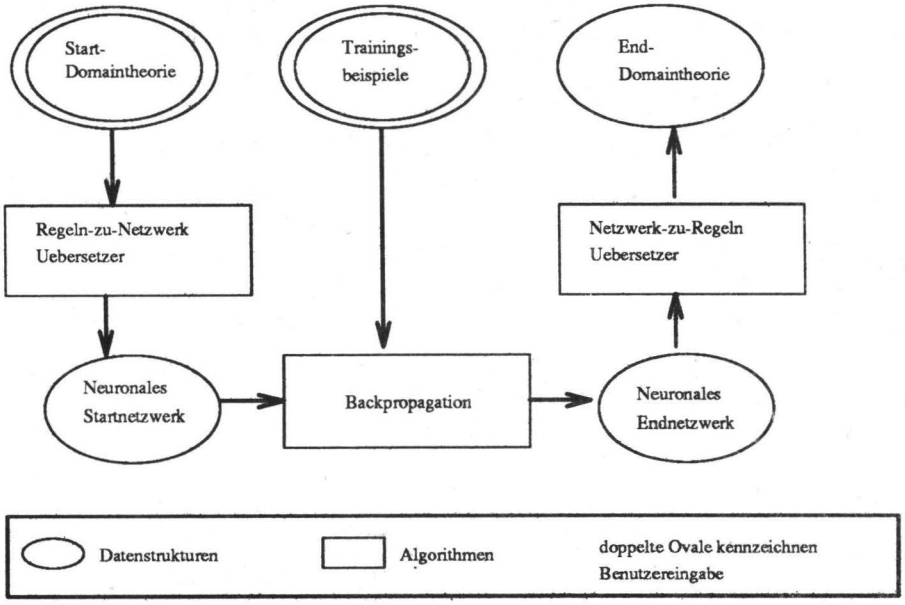


Abbildung 13.1: Informationsfluß durch KBANN

KBANN hat einen prinzipielle Aufteilung in 3 Algorithmen, was anhand der Abb. 13.1 deutlich wird. Der erste Teil setzt symbolische Regeln in ein neuronales Netzwerk um. Das somit gewonnene Netzwerk verhält sich dann wie der ursprüngliche Satz von Regeln.

Der zweite der drei Algorithmen verfeinert das Netzwerk durch Backpropagation.

Der letzte extrahiert schließlich aus dem trainierten Netzwerk symbolische Regeln. Somit ist nun das Wissen des Netzwerkes für den Menschen verständlich. Weiterhin kann man diese Regeln wieder in KBANN eingeben und damit weiterhin verbessern.

(siehe Abb. 13.2 [1])

Nun jedoch nach dieser kleinen Übersicht zu einer intensiveren Betrachtung von KBANN.

13.2.1 Einfügung von Wissen in ein neuronales Netz

In diesem Abschnitt wird nun der erste Algorithmus von KBANN betrachtet, welcher eine Menge von Regeln in ein wissensbasiertes neuronales Netz übersetzt. Es gibt einige Vorteile, die auf der Hand liegen, wenn man ein Netzwerk aufbaut, indem man gegebene Regeln benutzt. Zum einem sind nun wichtige Eigenschaften bekannt, die benötigt werden, um richtige Entscheidungen zu treffen, desweiteren können auch hergeleitete Eigenschaften, die ebenfalls zur Entscheidungsfindung wichtig sind, dem Netzwerk eingeimpft werden. Schließlich können sich die übersetzten Regeln auch auf relativ kleine Bereiche des Eigenschaftsraums beziehen. Dies beseitigt die Probleme von empirischen Lernsystemen, die sie beim Lernen von kleinen disjunkten Mengen haben. Die zu übersetzenden Regeln sind als Hornklauseln gegeben.

Es gibt jedoch drei Einschränkungen der Regelmenge:

Die Regeln dürfen keine Variablen enthalten, da der benutzte neuronale Lernalgorihmus diese nicht behandeln kann. Desweiteren sollten die Regeln azyklisch sein, weil dies die Übersetzung und das darauffolgende Training des enstandenen Netzwerks vereinfacht. Es ist zu erwähnen, daß dies keine prinzipielle Einschränkung darstellt, einfach aufgrund der Tatsache, weil Algorithmen existieren, die auch Netzwerke mit Zyklen problemlos trainieren. Zusätzlich sollten die Regelmengen hierarchisch gegliedert sein.

Tabelle 13.1: Übereinstimmungen zwischen Wissensbasen und neuronalen Netzwerken.

Wissensbasis	Neuronales Netzwerk
Schlußfolgerungen	Ausgangs-Units
Unterstützende Fakten	Eingangs-Units
Zwischenfolgerungen	Versteckte-Units
Abhängigkeiten	Gewichtete Verbindungen.

Nun zum eigentlichen Algorithmus, dem einfacherweise genannten „Regel – zu – Netzwerk – Algorithmus". Er besteht aus folgenden 7 Schritten, die nach dieser ersten Übersicht dann noch genauer erläutert werden.

Der Regel-zu-Netzwerk-Algoritmus von KBANN [1, 2]

1. Umschreiben der Regeln, daß Disjunktionen nur in Regeln sind, die nur ein Literal auf der rechten Seite haben.
2. Die Regelstruktur in ein neuronales Netzwerk umwandeln.
3. Die Units in dem KBANN- Netz entsprechend ihrer „Stufe" markieren.
4. Versteckte Units in das Netzwerk in einer benutzerdefinierten Stufe einfügen. (optional)
5. Units für bekannte Eigenschaften einfügen, die nicht in den Regeln benutzt wurden.
6. Verbindungen, die durch die Übersetzung nicht spezifiert wurden, zwischen alle Units in topologisch-unterschiedlichen Stufen befindlichen Units einfügen.
7. Das Netzwerk durch Addition von Zufallszahlen, nahe Null, zu allen Verbindungsgewichten und Biasen „beunruhigen".

Nun zu einer genaueren Betrachtung:

Schritt 1: Umschreiben der Regeln

In diesem Schritt wird der Regelsatz in ein Format gebracht, welches die hierarchische Struktur verdeutlicht und eine direkte Übersetzung in ein neuronales Netzwerk ermöglicht.

Jede Regel mit derselben Folgerung wird untersucht, um festzustellen, ob irgendeine mehr als eine Voraussetzung besitzt. Dies ist nötig, da die einzige Form von Disjunktionen, die von KBANN zugelassen wird, mehrere Regeln mit derselben Folgerung sind.

Falls nun mehr als eine Regel zu einer Folgerung gefunden wird, dann wird jede Regel mit mehr als einer Voraussetzung in zwei Regeln umgeschrieben. Eine der Regeln hat die ursprüngliche Folgerung und einen neuen Term als Voraussetzung, die andere Regel jedoch den neuen Term als Folgerung und die bisherigen Voraussetzungen. Zum besseren Verständnis dient Abb. 13.3 [1] , welche die Umwandlung von zwei Regeln zeigt.

Schritt 2: Umwandlung in ein neuronales Netzwerk

Mit diesen Übereinstimmungen aus Tab. 13.1 erstellen wir ein KBANN Netz. Die Gewichte an allen durch die Regeln hergestellten Verbindungen wird auf ω bzw. $-\omega$ gesetzt, je nachdem ob die Voraussetzung positiv oder negiert ist (aufgrund empirischer Untersuchungen [1] wird $\omega = 4$ benutzt). Die Biase(Vorwerte) werden auf $\frac{\omega}{2}$ bei Disjunktionen und sonst auf $\frac{(2P-1)\cdot\omega}{2}$, wobei P die Anzahl unnegierter Terme ist. Wenn auf diese Art und Weise die Gewichte gesetzt werden, dann sind Units in einem solchen Netz nur dann aktiv, wenn die entsprechende Regelfolgerung erreicht wird. Somit spiegelt das KBANN Netz das Verhalten des Regelsatzes wieder, mit dessen Hilfe es erstellt wurde. Am Ende dieses Schrittes hat das

Netzwerk Informationen übernommen, die wichtige Eingaben und abgeleitete Eigenschaften betreffen. Dennoch gibt es keine Garantie, daß die Regeln sich auf alle wichtigen Eigenschaften beziehen oder eine bedeutsame Sammlung von abgeleiteten Eigenschaften haben. Daher vergrößern die nächsten vier Schritte das Netzwerk mit Verbindungen, Eingabe-Units und eventuell versteckten Units.

Schritt 3: Die Markierung

Die Units werden hier gemäß ihrer „Stufen" mit Zahlen versehen. Die Stufe einer Unit ist als die Länge des kürzesten Pfades mit einer Eingabe-Unit definiert.

Schritt 4: Hinzufügen von versteckten Units (siehe hierzu auch Anmerkungen zu KBANN)

Durch diese Erweiterung ist nun das Netzwerk in der Lage, abgeleitete Eigenschaften zu lernen, die nicht im ursprünglichen Regelsystem spezifiziert wurden. Es ist also fähig, das Vokabular der anfänglich gegebenen Regeln zu erweitern.

Schritt 5: Hinzufügen von Eingabe-Units

Dies ist erforderlich, da ein Regelsatz, der nicht vollständig korrekt ist, wohlmöglich nicht jede zum korrekten Lernen des Konzepts nötigen Eingabeeigenschaften identifizieren kann.

Schritt 6: Hinzufügen von Verbindungen

Hier werden Verbindungen mit dem Gewicht 0 hinzugefügt, unter Benutzung der Markierungen aus Schritt 4. Sie verbinden jede Unit aus Stufe $n-1$ mit jeder aus der Stufe n. Die Anwendung dieser Methode zusammen mit der Markierung aus Schritt4 hat sich als geringfügig besser erwiesen als andere Methoden.

Schritt 7: Verwirrung des Netzwerkes

Dies wird durch Addition kleiner Zufallszahlen zu allen Gewichten getan. Diese Verwirrungen sind zu gering, um die Trainingseigenschaften negativ zu beeinflussen, aber sie genügen, Problemen aus dem Weg zu gehen, die man mit dem Training von symmetrischen Netzwerken hat.

(siehe Abb. 13.4 [1])

Die Übersetzung von Regeln in ein KBANN-Netzwerk:

Um die Betrachtungen in diesem Abschnitt zu vereinfachen, werden nur binärwertige Eigenschaften betrachtet. Falls diese nicht in dieser Form vorliegen, werden sie umgewandelt. Betrachtet man als Beispiel mal die Eigenschaft Farbe, die als Umwandlung die drei Werte rot, grün und blau hat.

Der „Regel-zu-Netzwerk-Algorithmus" setzt die Gewichte der Verbindungen und die Biase der Units, so daß diese nur dann aktiviert werden, wenn eine entsprechende Deduktion aus der Wissensbasis existiert. Andererseits, wenn die Deduktion nicht mit der Wissensbasis möglich ist, dann sollte die entsprechende Unit inaktiv sein, also eine Aktivierung nahe Null haben.

Konjunktive Regeln werden in ein neuronales Netzwerk übersetzt, indem alle Gewichte der Verbindungen, bei positiven Voraussetzungen auf ω, bei negativen auf $-\omega$ gesetzt werden und das Bias der Unit, welches der Regelfolgerung entspricht, auf $\frac{(2P-1)\cdot\omega}{2}$. Das P bezeichnet die Anzahl der positiven Voraussetzungen einer Regel. Wie oben schon erwähnt, benutzt KBANN für ω den Wert 4, da dieser sich nach einer Reihe von empirischen Tests als geeignet erwiesen hat.

Als Erläuterung dient nun Abb. 13.5 [1, 2].

In dem ersten Teil sieht man, daß die Eingänge mit dem Bias zusammen nur Null überschreiten, wenn alle der positiven Voraussetzungen und keine der negativen Voraussetzungen wahr sind. Es sei erwähnt, daß Units nur bei einer Überschreitung von Null aktiv sind. Also ist die Unit nur dann merklich aktiv, wenn alle positiv gewichteten Verbindungen Signale nahe an 1 und die negativen nahe an 0 haben.

Um disjunktive Regeln nun umzusetzen, schaut man den zweiten Teil an und hier ist zu beachten, daß die Eingangssignale den Bias übertreffen, sobald eine der Regelvoraussetzungen erfüllt ist.

13.2.2 Die Verbesserung der KBANN-Netze

Die Netzwerke von KBANN werden mit Backpropagation trainiert, einer Standardlernmethode. Leider erzeugen aber dann die KBANN-Netze Probleme, da die Antworten schon zu Beginn zuversichtlich sind, d.h. die Ausgabeunits haben entweder Aktivierungszustände nahe an 0 oder 1, abgesehen davon ob sie korrekt sind. Diese Eigenart ist das Problem für Backpropagation, denn wenn Antworten zuversichtlich sind, werden kaum Änderungen am Netzwerk getan.

Es gibt Experten[Rumelhart86], die dies für eine erwünschenswerte Eigenschaft bei standardmäßigen neuronalen Netzen halten, da diese somit rauschresistent wären. Aber davon abgesehen, ist es ein großes Problem, ein Netzwerk umzukehren, welches total falsch antwortet. Man bezeichnet dies als das „flat spot" Problem.

Es gibt diverse Lösungen, die nur kleine Änderungen erfordern, bei KBANN wird der Vorschlag von Hinton[Hinton89] angewandt, welcher die Ersetzung der „standard error function" (Error) durch die „cross entropy function" (C) bedeutet.

$$Error = \frac{1}{2}\sum_{i=1}^{n}(d_i - a_i)^2 \tag{13.1}$$

$$C = -\sum_{i=1}^{n}\Big((1-d_i)\cdot log_2 a_i) + (d_i \cdot log_2(1-a_i)\Big) \tag{13.2}$$

wobei:
a_i die Aktivierungsfunktion von Ausgabe-Unit i
d_i die erwünschte Aktivierung von Unit i
n die Anzahl von Ausgabe-Units ist.

13.2.3 Die Extraktion von Regeln aus einem KBANN-Netz

Nachdem die Netze nun trainiert wurden, stellen sie hochrangige Klassifizierer dar, es folgen später noch einige Erläuterungen dazu. Aber wie gut sie auch sein mögen, man bekommmt leider keine Begründungen für die Antworten. Die Resultate des Lernens sind weder für uns ersichtlich, noch kann das gewonnene Wissen auf verwandte Probleme angewandt werden. Wenn man nun symbolische Regeln extrahiert, spricht man genau dieses Problem an. Das gelernte Wissen ist nämlich nun verständlich und begründet die Antworten. Weiterhin können die veränderten Regeln als Teil einer Wissensbasis zur Lösung verwandter Probleme benutzt werden.

Nun werden zwei Verfahren betrachtet, welche zur Extraktion der Regeln dienen. Zunächst aber noch zu Eigenschaften, welche beide Lösungen besitzen, und zu zwei Annahmen, die über die neuronalen Netze gemacht werden.

Untermauerung der Regelextraktion

Die erste Annahme ist, daß das Training nicht wesentlich die Bedeutung der Units ändert. Dadurch sind die Methoden fähig Marken an jede Regelfolgerung und -voraussetzung zuzuweisen.

Genauere Untersuchungen [1,4] von trainierten Netzwerken zeigen, daß die Bedeutungen sehr stabil bleiben, außer bei den unsichersten der Anfangsregeln.

Die zweite Annahme ist, daß beinahe alle Units entweder total aktiv oder inaktiv sind, dh. Aktivierung nahe 1 oder 0 haben. Durch diese Festlegung kann jede Nicht-Eingabe Unit als eine „step function" oder eine Boolesche Regel aufgefasst werden. Wieder hat eine genauere Untersuchung [1,4] die Annahme als gültig erwiesen.

Die Methoden, die hier benutzt werden, arbeiten so, daß sie versuchen Kombinationen zu finden, bei denen die Eingabewerte einer Unit eine Aktivierung nahe 1 haben. Die Regelextraktionsalgorithmen suchen also nach Kombinationen in denen die Summe der Eingänge das Bias überschreitet.

Durch die Tatsache, daß keine negativen Aktivierungen existieren, wird die Arbeit erleichtert. Diese Eigenschaft erlaubt es, die Gewichte als einen perfekten Anzeiger für die Art der Anwendung der Regelvoraussetzung zu verwenden, d.h. negativ gewichtete Verbindungen können nur negative Voraussetzungen steigern, analoges gilt bei positiven.

Die Subsetmethode

Sie hat ihren Namen daher, daß sie versucht, Teilmengen der Units zu finden, die mit einer folgenden Unit verbunden sind. Diese Art der Regelextraktion wurde unabhängig voneinander von verschiedenen Forschern (Saito und Nakano, Fu, Masurka) entwickelt. Es ist also nicht weiter verwunderlich, daß sie recht häufig in der Literatur anzutreffen ist.

Der Subset-Algorithmus [1]

1. Das Bias jeder versteckten und Ausgabe-Unit auf θ setzen.
2. Bis zu β_p Gruppen von positiv gewichteten Verbindungen zu jeder Unit finden, so daß

   $0 < \theta + \sum(\textit{Verbindungsgewichte einer Gruppe})$

   Die gefundene Menge von Gruppen wird mit G_p bezeichnet.
3. Für jedes $P \in G_p$

   bis zu β_n Gruppen von negativ gewichteten Verbindungen zu jeder Unit finden, so daß $0 > \theta + \sum(\textit{Verbindungsgewichte von } P) + \sum(\textit{Verbindungsgewichte einer Gruppe})$

   Die gefundene Menge von Gruppen wird mit G_n bezeichnet.

   Für jedes $N \in G_n$

   eine Regel folgender Form bilden: *if* $\forall$ P *and not* $\forall$ N *then* (*name of* U)
4. Duplikate von Regeln entfernen.

Sie basiert stark auf der Annahme, daß die Units entweder aktiv oder inaktiv sind, und so werden nur noch Verbindungsgewichte betrachtet. Daher ignorieren Schritt 2 und 3 die Aktivierung der Units an den sendenden Enden der Verbindungen.

(siehe Abb. 13.6 [1])

Das Hauptproblem der Subsetmethode stellt der Aufwand zum Finden aller Teilmengen dar. Man sollte hiermit nur einfache Netzwerke und Domains benutzen. Um dies zu umgehen, kann man eine obere Schranke für die Anzahl der Regelvoraussetzungen in extrahierten Regeln festlegen, aber die unterste mögliche Schranke für einige Probleme liegt bei 10^5.

Wenn man also den Subsetalgorithmus benutzt, hat man auch aufgrund dieser Beschränkung, mit einer Diskrepanz zwischen dem Netzwerkverhalten und dem Regelverhalten zu rechnen.[1,4]

Leider schafft es diese Methode auch nicht, wichtige Strukturen aus dem trainierten Netzwerk zu ziehen. Im obigen Beispiel haben die Verbindungen zu B, C, D dasselbe Gewicht und zu E das negative Pendent. Unter dieser Beobachtung kann man die Regeln umschreiben, so daß man dann eine genauere Begründung für A erhält.

IF (3 of {B,C,D,not(E)}) then A

Diese Idee führt zu der zweiten Methode.

Die NofM Methode

Hier sucht man konkret nach Regelvoraussetzungen der Form:

IF (N of the following M antecedents are true) then ..

Der NofM-Algorithmus

1. Gruppen aus versteckten und Ausgabe-Units bilden, die ähnliche Gewichte haben.
2. Das Verbindungsgewicht aller Gruppenmitglieder auf den Durchschnitt setzen.

3. Jede Gruppe. die nicht wesentlich die Unit beeinflußt, eliminieren, unabhängig davon ob die Unit aktiv oder inaktiv ist.

4. Alle Verbindungsgewichte konstant halten und die Biase aller versteckten und Ausgabe-Units durch Backpropagation optimieren.

5. Für jede versteckte und jede Ausgabe-Unit einzelne Regeln bilden. Die Regeln bestehen aus einer Schwelle, gegeben durch das Bias, und gewichteten Regelvoraussetzungen, angegeben durch die verbliebenen Verbindungen.

6. Regeln durch Elimination von Gewichten und Schwellen vereinfachen.

Die Idee, die dem Algorithmus zugrunde liegt, ist, daß einzelne Regelvoraussetzungen, also Verbindungen im Netzwerk, alleine keine allzu große Bedeutung haben. Es ist eher so, daß Gruppen von Verbindungen Äquivalenzklassen bilden. Dieser Grundgedanke erlaubt dem Verfahren, nur noch Gruppen zu betrachten und einzelne Verbindungen außer acht zu lassen.
Im Folgenden werden ein wenig die einzelnen Schritte betrachtet.

Schritt 1: Clusterbildung

Nach dem Training der Netzwerke liegen normalerweise keine Äquivalenzklassen vor, sondern diese müssen erst gebildet werden. Dazu faßt man zunächst einmal Verbindungen , die einen Abstand von 0,25 haben, zu Gruppen zusammen. Hier wird der Join-Algorithmus[Hartigan75] verwendet, der mit n Clustern beginnt und dann diese solange zusammenfaßt, bis der Abstand dazwischen 0,25 beträgt.

Schritt 2: Mittelwertbildung

Nun kann man in jeder gebildeten Gruppe den Durchschnitt der Verbindungsgewichte ermitteln und ihn jeder Verbindung der Gruppe zuweisen. Jetzt liegen die anfangs erwähnten Äquivalenzklassen vor.

Schritt 3: Eliminierung

Hier wird versucht, solche Gruppen ausfindig zu machen, die keinen Einfluß auf die Regelfolgerung haben. Meist haben sie geringe Verbindungsgewichte und sind nicht sehr groß.

Für diesen Schritt stehen zwei Lösungen zur Auswahl, die eine ist algorithmisch, die andere heuristisch. Zur ersten ist zu sagen, daß nun diese Cluster wie folgt gesucht werden, indem nämlich die ingesamt mögliche Aktivierung berechnet wird, die jedes Cluster aussenden kann. Dann werden diejenigen eliminiert, welche keine Änderungen bewirken, egal ob die Netzeingabe das Bias überschreitet oder nicht.

(sieh Abb. 13.7 [1])

Der heuristische Ansatz geht folgendermaßen: Man testet das Netzwerk mit einem Beispiel und setzt dann nacheinander jede Clustereingabe auf 0. Wenn dies einen Einfluß auf die Aktivierung der Unit hat, dann wird sie als benötigt markiert, ansonsten nicht. Wenn dies mit allen Clustern getan wurde, werden alle unmarkierten Cluster eliminiert.

Schritt 4: Optimierung

Dies ist erforderlich, da die vorhergehenden Schritte das Aktivierungsverhalten der Units ändern können, die nach dem Training erzielten Fehlerquoten könnten sich verschlechtert haben.

Zur Lösung dieses Problems kann man die Gewichte einfrieren und dann die Biase trainieren. Um das regelähnliche Verhalten des Netzwerkes zu zeigen, muß auch noch die Aktivierungsfunktion ein wenig geändert werden, damit die Units näher an das Verhalten einer „step function“ kommen.

Schritt 5: Extrahierung

Jetzt werden einfache Regeln aus dem Netzwerk gebildet, die dessen Verhalten wiederspiegeln. Die Bildung dieser ist ziemlich einfach. Man entnimmt die Biase sowie die Clustergewichte und formt daraus Regeln, die wahr sind, wenn die Summe der erfüllten gewichteten Regelvoraussetzungen das Bias überschreitet.

Schritt 6: Vereinfachung

Letztendlich werden die erhaltenen Regeln noch vereinfacht, indem versucht wird, die Gewichte und die Biase aus ihnen zu entfernen. Dazu muß man jeweils Gruppen von Regelvoraussetzungen bilden, die das Bias überschreiten.

Beispiel:

Es sei folgende Regel extrahiert worden:

A :- 10.0 < 5.1*NumberTrue{B, C, D, E} + 3.5*NumberTrue{X, Y, Z}.

Die Vereinfachung dieser Regel erzeugt folgende 3 Regeln:

A :- 2 of {B, C, D, E}

A :- 1 of {B, C, D, E} and 2 of {X, Y, Z}

A :- X, Y, Z.

Falls die Elimination der Gewichte und Biase jedoch aus einer Regel mehr als fünf Regeln macht, dann wird keine Änderung vollzogen.

(siehe Abb. 13.8 [1])

Beide dieser Regelextraktionsalgorithmen haben Stärken und Schwächen, z.B. sind die einzelnen Regeln von Subset einfacher zu verstehen, aber er extrahiert meist eine viel größere Anzahl. Andererseits ist die Gesamtheit der Regeln, also die Regelmenge, von NofM einfacher zu verstehen. Aber das Entscheidenste ist der Vergleich der Komplexität, der zeigt, daß Subset exponentiellen und NofM quadratischen Aufwand hat. Desweiteren haben eine Reihe von Tests gezeigt, daß die Regeln von NofM weitaus besser das Netzwerkverhalten wiederspiegeln als die von Subset. Aufgrund dieser Vorteile ist es sicherlich sinnvoll, sich für NofM zu entscheiden.

13.3 Anmerkungen zu KBANN

KBANN wurde mit Erfolg auch experimentell erprobt. Die Testwissensbasen stammten aus dem Bereich der Molekularbiologie, genauergesagt aus dem „Human Genome Project". Das Hauptziel dieses Forschungprojekts liegt darin, die menschliche DNA zu entschlüsseln. Das hier vorgestellte hybride Lernsystem wurde dabei in verschiedensten Aufgabenbereichen („promoter recognition", „splice-junction determination") zur Verbesserung vorhandener Regelsätze benutzt.

Mit Hilfe dieser Tests hat man auch weitere tiefgehende Erkenntnisse über KBANN erhalten. Die Aufzählung all dieser gehört jedoch nicht an diese Stelle, bei weiterem Interesse sei auf [1] verwiesen.

Zu dem vorhandenen Grundgerüst von KBANN gibt es auch Erweiterungen, von denen hier nun zwei kurz erläutert werden sollen.

Die erste ist DAID (Desired Antecedent IDentification). DAID ist ein symbolischer Preprozessor, der die Trainingsbeispiele dazu benutzt Eingabeeigenschaften zu erkennen, welche helfen sollen, Fehler im gegebenen Domainwissen zu erkennen. Denn es hat sich bei oben erwähnten Experimenten gezeigt, daß KBANN leichter unnütze Regelvoraussetzungen ablernt, als zuvor für unwichtig gehaltene zu lernen. Somit hat man eine Methode, die eventuell nützliche Eigenschaften in unser Anfangswissen einfügt, um das Lernverhalten der KBANN zu verbessern.

Die zweite Erweiterung ist das Hinzufügen von versteckten Units, welches die Möglichkeit gibt, das Grundvokabular der gegebenen Domaintheorie zu vergrößern. Dies hat die Wirkung, daß sich das Netzwerkverhalten verbessern und die Verständlichkeit der extrahierten Regeln erhöhen kann.

Hier soll jedoch nicht weiter gegangen werden, Interessierte seien auf Towells Doktorarbeit[1] verwiesen.

Die erwähnten Testbasen wurden später auch dazu benutzt, um Vergleiche mit empirischen Lernmethoden (Standard Backpropagation, ID3, Nearest Neighbor, PEBLS, Perceptron, Cobweb) zu machen. Diese Kombination von subsymbolischem und symbolischem Lernen war effizienter und effektiver als diese Verfahren, welche nur eine der beiden Informationsquellen benutzen.[1]

KBANN wurde weiterhin auch mit anderen hybriden Systemen (EITHER, Labyrinth-k) verglichen und hat sich auch hier als überlegen erwiesen.[1]

Dieser Abschnitt sollte nur kurz aufzeigen, daß keineswegs die Erforschung von KBANN abgeschlossen ist.

13.4 Zusammenfassung

Diese Abhandlung beschreibt KBANN eine hybride Lernmethode, welche aus Regelwissen und Trainingsbeispielen lernt. Es liegt hier eine gelungene Kombination von subsymbolischem und symbolischem Lernen vor.

Außer daß KBANN eine hochrangige Klassifikationsmethode ist, gilt das Augenmerk der Tatsache, daß KBANN Regeln aus einem neuronalen Netzwerk extrahieren kann. Dadurch kann KBANN als Regelverbesserungssystem eingesetzt werden. Diese Extraktion schafft es also, das neu gelernte Wissen des neuronales Netzes in einer verständlichen Form, nämlich gerade diesen Regeln, offenzulegen.

Für zukünftige Forschungen bietet der Bereich der Kombinierung von Lernen aus Regeln und Beispielen noch einiges an Perspektiven.

13.5 Literatur

[1] Geoffrey G. Towell, *Symbolic Knowledge And Neural Networks: Insertion, Refinement And Extraction*, PhD Thesis, 1991.

[2] Geoffrey G. Towell, Jude W. Shavlik, „Knowledge-Based Artificial Neural Networks“, 1/92 *Artificial Intelligence* vorgelegt.

[3] Geoffrey G. Towell, Jude W. Shavlik, Mark W. Craven, „Constructive Induction In Knowledge-Based Neural Networks“, erschienen in *Machine Learning: Proceedings of the Eighth Internatioal Workshop*, Birnbaum, L. and Collins, G. (eds.), Morgan Kaufmann, San Mateo, Ca., 1991.

[4] Geoffrey G. Towell, Jude W. Shavlik, „Extracting Refined Rules From Knowledge-Based Neural Networks“, 9/91 *Machine Learning* vorgelegt, 11/92 angenommen.

[5] Geoffrey G. Towell, Jude W. Shavlik, „Using Symbolic Inductive Learning To Improve Knowledge-Based Neural Networks“, 1/92 der *Tenth National Conference On Artificial Intelligence* vorgelegt.

[6] Richard Maclin, Jude W. Shavlik, „Using Knowledge-Based Neural Networks To Improve Algorithms: Refining The Chou-Fasman Algorithm For Protein Folding“, erschienen in *The Proceedings of the Tenth National Conference On Artificial Intelligence* (AAAI-92).

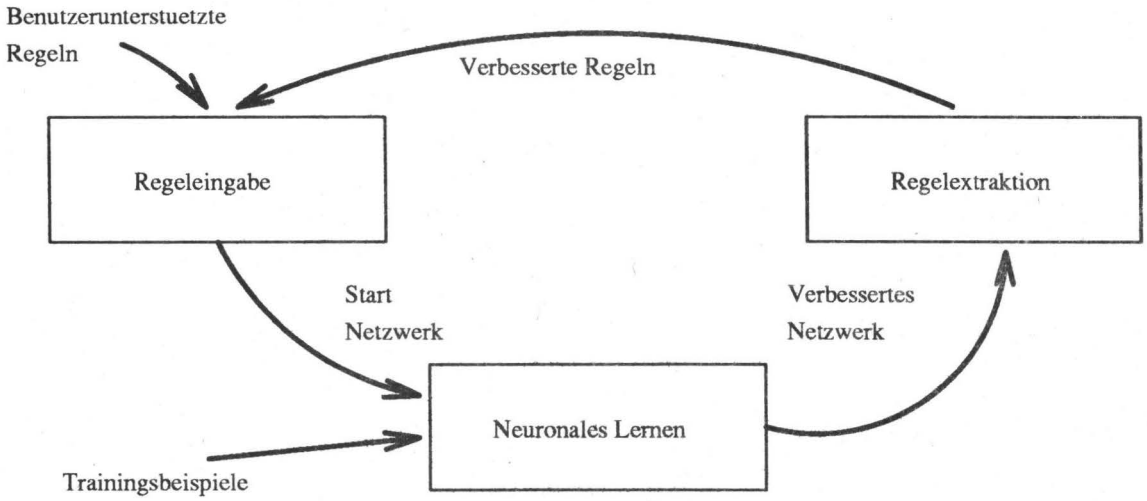


Abbildung 13.2: Informationsrückfuhr bei KBANN

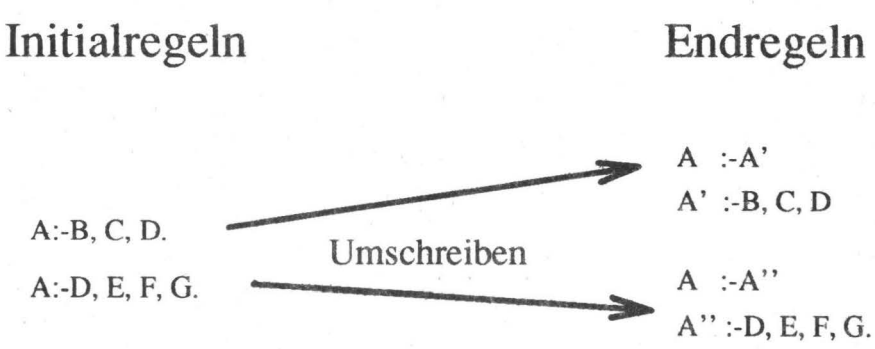


Abbildung 13.3: Umschreiben von Regeln

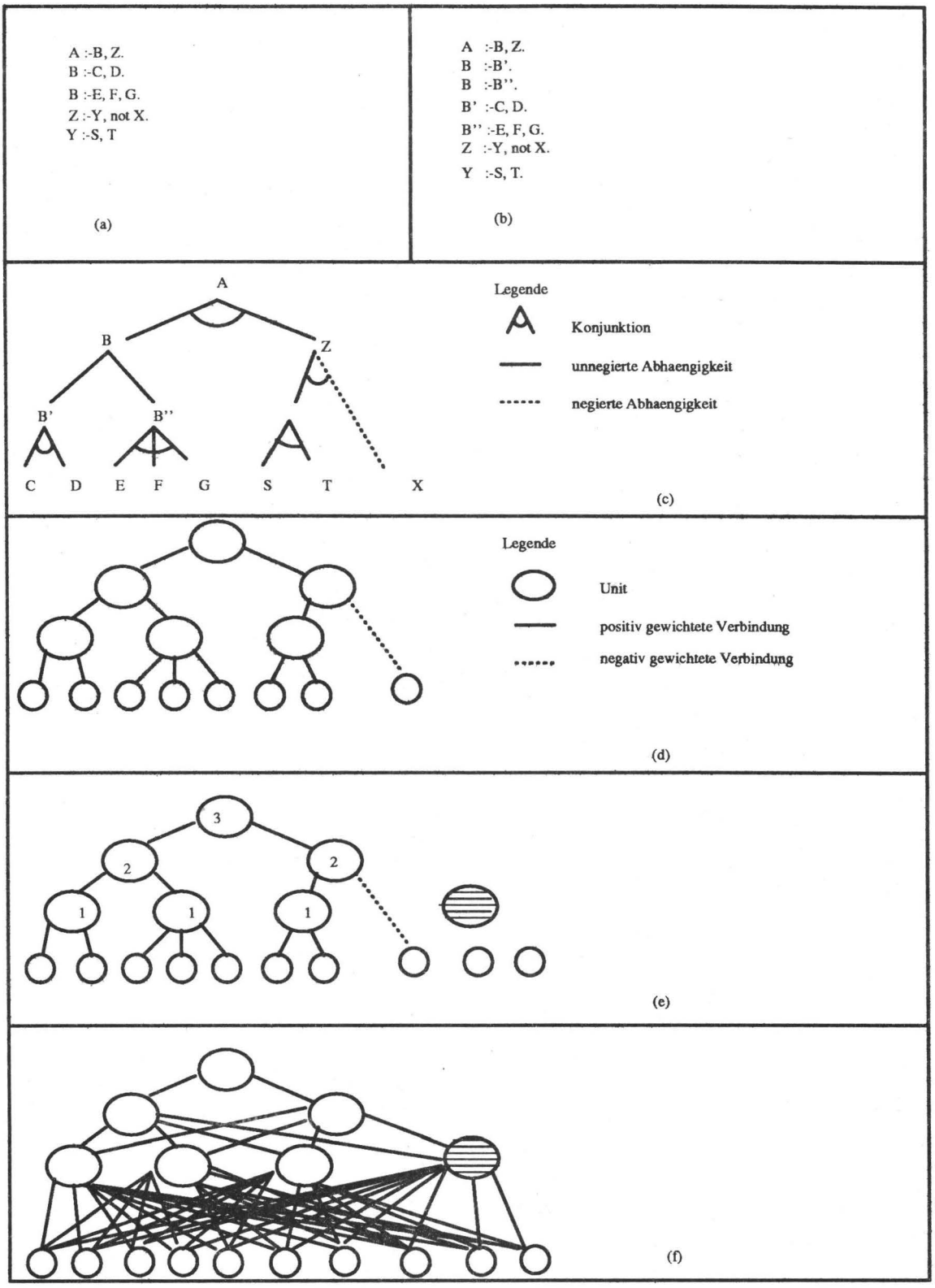
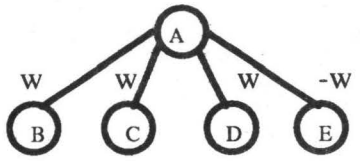


Abbildung 13.4: Beispiel zum Regel-zu-Netzwerk-Algorithmus

(a) Eine konjunktive Regel

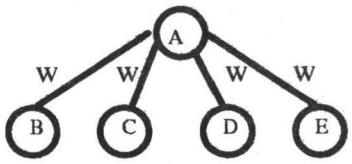
A :-B, C, D, not (E).



bias= (2*3-1)*w/2= 5/2*w

(b) Disjunktiver Regeln

A :-B. A :-C. A :-D. A :-E.



bias= w/2

Abbildung 13.5: Übersetzung von Regeln

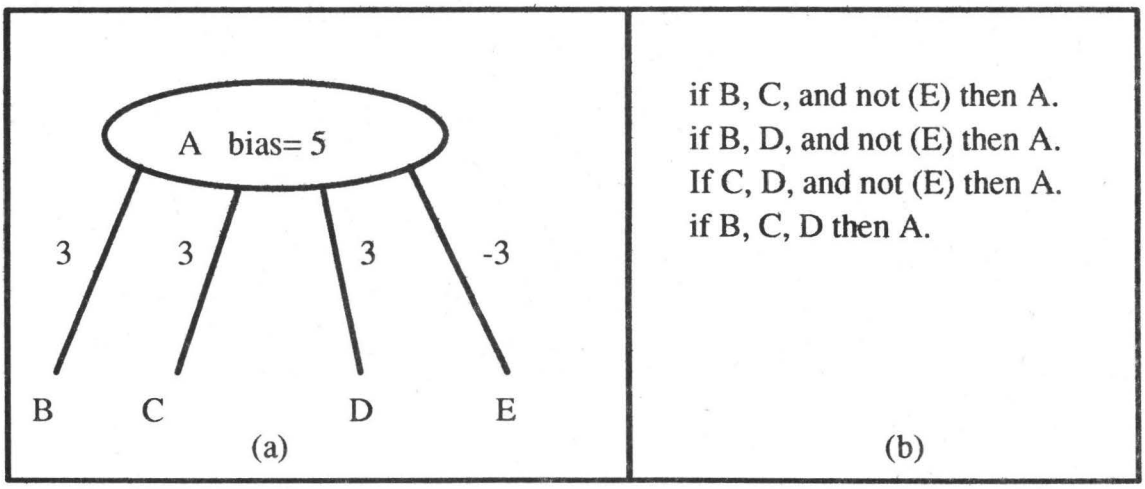
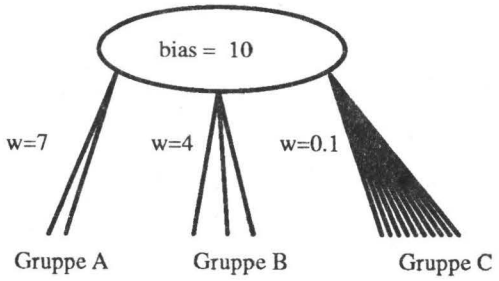


Abbildung 13.6: Beispiel zum Subset-Algorithmus



Einige Kombinationen von A und B	Gesamtgewicht einer Kombination
1B	4
1A	7
2B	8
1A+1B	11
3B	12
2A	14
1A+2B	15
...	...

Abbildung 13.7: Idee des NofM-Algorithmus

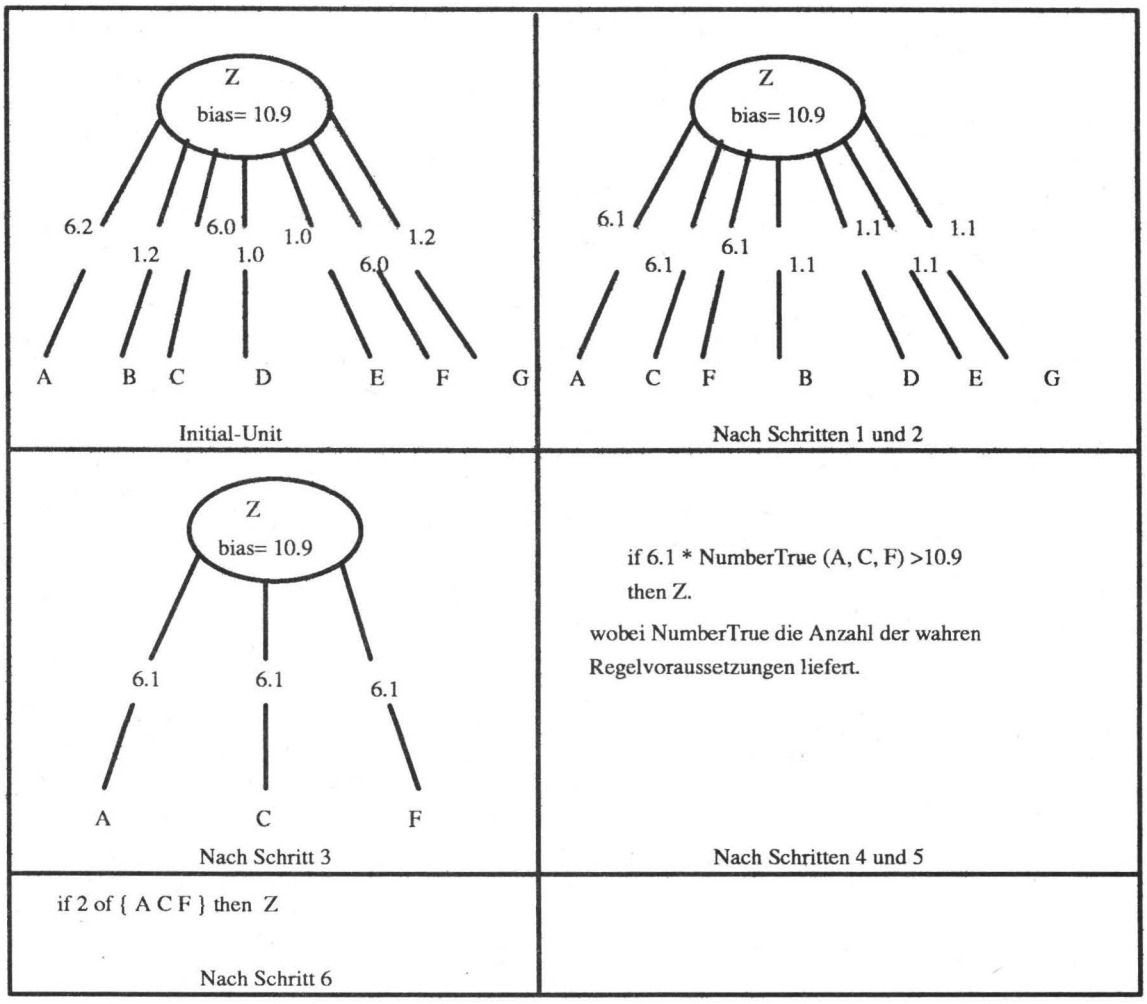


Abbildung 13.8: Beispiel zum NofM-Algorithmus

Kapitel 14

Modellierung biologischer Systeme

Oliver Schweikart

Übersicht

Dieser Beitrag stellt den Weg von biologischen Daten zu einem mathematischen Modell dar. Es werden zwei verschiedene Ansätze zur Modellierung von Dendriten vorgestellt („Kabeltheorie“ und „Compartmental Model“), um damit die Verfahren zu beschreiben, mit denen eine Simulation von neuronalen Verbänden möglich ist. Die Kabeltheorie wird sehr ausführlich hergeleitet, um die Vorgehensweise beim Erstellen von mathematischen Modellen zu verdeutlichen. Zu den theoretischen Grundlagen folgen praktische Beispiele von Simulatoren für biologische Systeme und deren Anwendung.

14.1 Einleitung

14.1.1 Motivation: Warum und wofür ?

Noch vor einigen Jahrzehnten bestand Forschung aus dem Zyklus: „Aufstellen einer Hypothese oder Vermutung “, „Experimente und Tests“, sowie „Analysen“. Dieses Vorgehen führte zu einfach zu lösenden Gleichungen. Doch seit einigen Jahren reicht diese Methode nicht mehr aus. Die großen technischen und wissenschaftlichen Probleme sind nicht mehr mit einfachen Gleichungssystemen zu lösen. Hierzu zählen unter anderem das 3D-Problem bei der Darstellung von Proteinen aufgrund ihrer Aminosäuresequenz, das Temperaturmodell der Erdatmosphäre in Abhängigkeit vom CO_2-Gehalt oder das menschliche Gehirn. Es ergeben sich nicht mehr zu überschauende Modelle und Gleichungen, die eine Simulation am Computer erfordern, um sie verstehen und aussagekräftige Versuchsreihen gestalten zu können. Der Zyklus wird also um die „Simulation“ erweitert („mathematische Theorien“, „Simulation der Theorien und Vorhersagen“, „Experimente und Tests“, „Analysen“).

In diesen Bereich der Forschung fallen Gebiete wie Meteorologie und Neuro-Wissenschaften, die vom massiven Computereinsatz abhängig sind. Im Vergleich zur KI sind die Neuro-Wissenschaften nicht an der optimalen, sondern an der natürlichen Problemlösung interessiert. Die Neuro-Wissenschaften orientieren sich mehr an der biologischen Realität und sind ohne Anspruch auf leistungsfähige Methoden. Gemeinsam aber ist beiden Teilgebieten der Versuch, komplexe, informationsverarbeitende Systeme zu verstehen und zu modellieren.

Trotz dem Versuch der Neuro-Wissenschaften, der biologischen Realität möglichst nahe zu kommen, müssen viele Sachverhalte stark abstrahiert werden, was jedoch nicht die Leistungsfähigkeit des Modells einschränken muß. Analog dazu sei die Modellierung von Gasen erwähnt, bei der

auch nicht auf einzelne Gasmoleküle eingegangen wird. Es gibt die „top-down-" und die „bottom-up-Strategie" zur Modellierung. In den Neuro-Wissenschaften wird meist eine Kombination aus beiden Verfahren benutzt, die stark auf experimentellen Daten aufbaut. Im Gegensatz zu diesem datengestützten Ansatz ist der mehr theoretische „top-down-Ansatz" eher im Bereich der KI angesiedelt.

14.1.2 Wege zu einem Modell

Hier wird gezeigt, wie man durch methodisches Vorgehen zu einem Modell von sehr komplexen Zusammenhängen kommt. Diese Techniken können natürlich nicht nur zur Modellierung biologischer Systeme benutzt werden. Zunächst wird der Verband von Neuronen abstrahiert, um dann kleine Teillösungen zu erhalten, welche anschließend wieder zusammengesetzt werden und das gesamte Modell ergeben.

Der Verband von Neuronen besteht aus vielen Neuronen (siehe Abbildung 14.1 eines Neurons), die miteinander verknüpft sind (Grundlagen siehe im entsprechenden Vortrag). Das gesuchte Modell muß also die Neuronen und die Verbindungen beschreiben. Hier wird am Beispiel der Dendriten gezeigt, wie man zu einer mathematischen Beschreibung kommt. Die Vorgehensweise bei den anderen Teilen eines Nervensystems ist analog. Diese Teilmodelle werden dann zusammengesetzt. Am Schluß wird auf real existierende Simulationsprogramme eingegangen, die das hier aufgebaute Modell benutzen.

14.2 Dendritenmodell

Es gibt zwei unterschiedliche Methoden, Dendriten zu modellieren: die Kabeltheorie und das Compartmental Model.

14.2.1 Kabeltheorie

Die Ansätze der Kabeltheorie stammen aus dem Jahre 1855 und sind von Prof. William Thomson (Lord Kelvin). Sie dienten zur Berechnung des ersten transatlantischen Telegraphie-Kabels. Die Parallelen zum Neuron werden deutlich, wenn man einen unverzweigten Abschnitt eines Dendriten betrachtet. Die Kabelgleichung

$$\lambda^2 \frac{\partial^2 U}{\partial x^2} - U - \tau \frac{\partial U}{\partial t} = 0 \tag{14.1}$$

wird schrittweise hergeleitet, bevor sie auf bestimmte Randbedingungen angewendet wird. $U = U_i - U_e - E_r$ ist die Membranspannung, x die Länge in Richtung der Achse des Dendriten und λ ist die Längenkonstante des Mantelleitwertes (Definition in 14.11 und 14.12). τ_m ist die Zeitkonstante der passiven Membran (Definition 14.10).

Bei statischem Gleichstrom wird die Ableitung der Spannung Null:

$$\frac{d^2 U}{dX^2} - U = 0 \tag{14.2}$$

Mit dieser vereinfachten Gleichung kann man sehr gut Eingangswiderstände und Spannungsabfälle entlang der Achse betrachten. Im Fall des statischen Wechselstroms (Sinusschwingungen) erhält man auch eine vereinfachte Gleichung

$$\frac{d^2 \hat{U}}{dX^2} - q^2 \hat{U} = 0 \tag{14.3}$$

mit komplexen Variablen, die bei Analysen der Übertragung und der Kapazität hilfreich ist.

Doch wie kommt man vom Dendrit zur Kabelgleichung ? Dendriten sind dünne Röhren ($0,5 \mu m bis zu 1 mm$) aus einer Nervenmembran, die man als Zylinder idealisieren kann. Sie entsprechen einem Mantelkondensator, da die Membran von Ionen, die sich im Intra- und im Extrazellulärraum befinden,

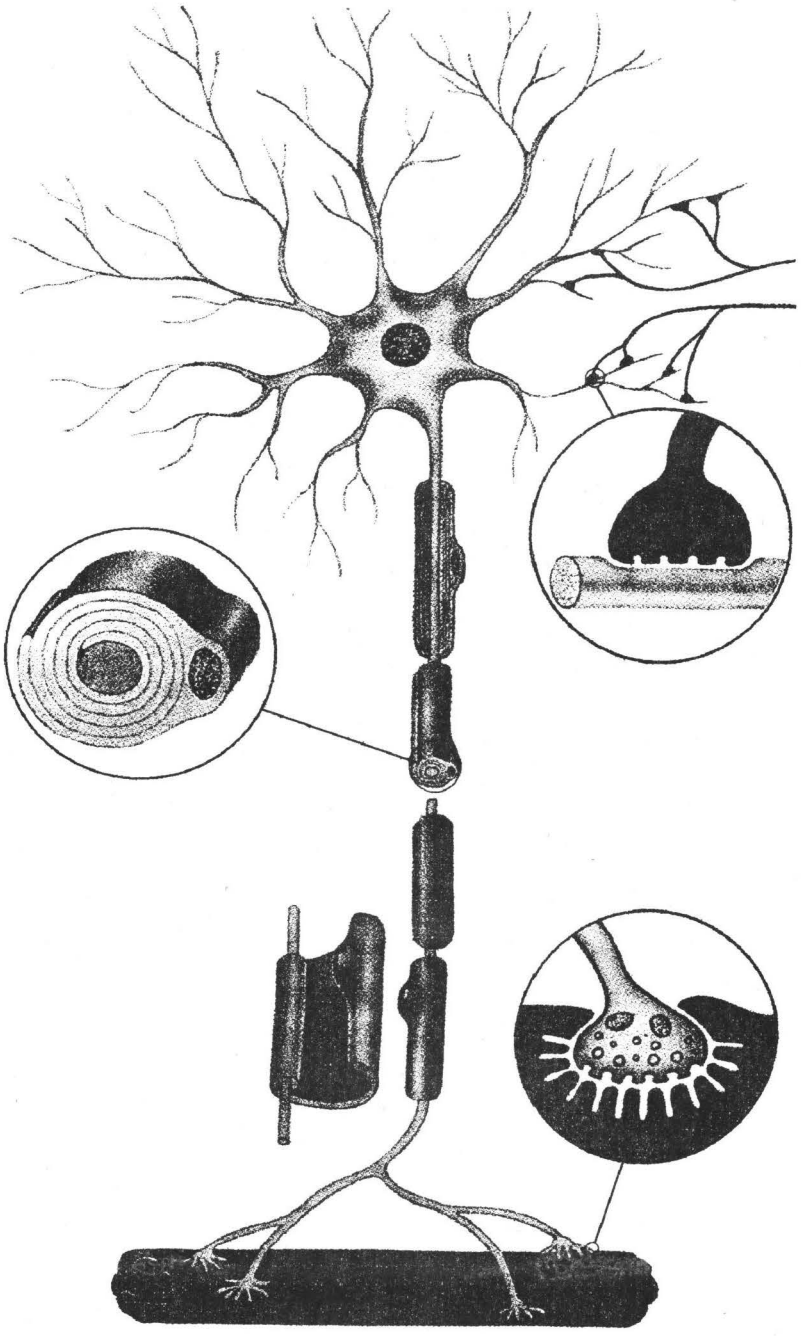


Abbildung 14.1: Neuron[5]

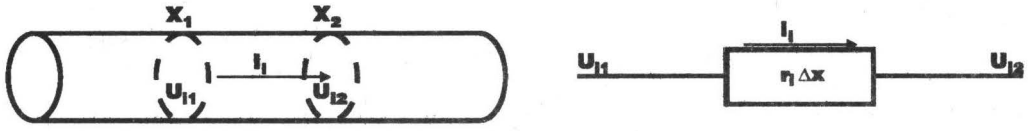


Abbildung 14.2: Intrazellulärer Strom und Widerstand [1]

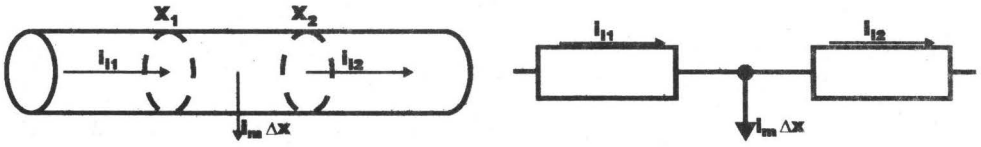


Abbildung 14.3: Stromdichte der Membran [1]

umgeben ist. Bei kurzen Abschnitten (bezüglich λ, aber ein Mehrfaches des Durchmessers) ist der Membranwiderstand wesentlich größer als der Widerstand in Richtung der Achse (sowohl innerhalb wie auch außerhalb). Daraus ergibt sich ein wesentlich größerer Strom parallel zur Membran als durch diese hindurch. Dieser Effekt ist bei markhaltigen Axonen[1] besonders ausgeprägt. Idealisiert man den Dendriten auf einen gleichförmigen, zylindrischen Mantelkondensator, so bedeutet dies gleichförmige Membraneigenschaften und gleichförmigen Innenwiderstand r_i pro Längeneinheit. Eine Grundvoraussetzung der eindimensionalen Kabeltheorie ist, daß die intrazelluläre Spannung U_i nur eine Funktion von zwei Variablen ist, der Zeit t und der Länge x. Außerdem gilt

$$\frac{\partial U_i}{\partial x} = -i_i r_i \tag{14.4}$$

wobei i_i der intrazelluläre Strom und r_i der intrazelluläre Widerstand ist. Diese Beziehung geht aus folgender Graphik (14.2) hervor, wenn $\Delta x \to 0$ geht.

Bei einem gleichförmigen Mantelkondensator ist r_i von t und x unabhängig. Die Ableitung nach x von 14.4 ergibt die zweite Ableitung von U_i bezüglich. x:

$$\frac{\partial^2 U_i}{\partial x^2} = -r_i \frac{\partial i_i}{\partial x} \tag{14.5}$$

Aus der Abbildung 14.3 ist die Stromdichte der Membran pro Einheitslänge des Zylinders ersichtlich, wenn kein Strom durch Elektroden zugeführt wird. Mathematisch ergibt das

$$i_m = -\frac{\partial i_i}{\partial x} \tag{14.6}$$

Aus den letzten beiden Gleichungen erhält man

$$\frac{1}{r_i}\frac{\partial^2 U_i}{\partial x^2} = i_m \tag{14.7}$$

[1] siehe Grundlagenvortrag

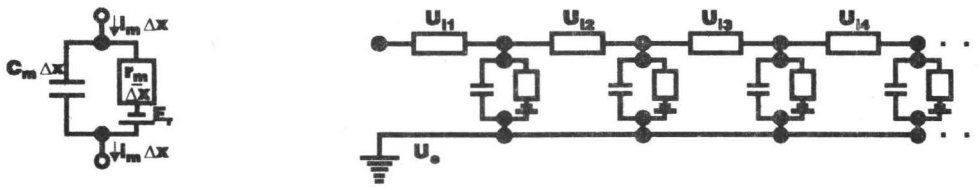


Abbildung 14.4: Einfaches Membranersatzschaltbild und Teil eines Dendriten (Zylinder) [1]

Bisher wurden weder das Membranersatzschaltbild noch die extrazellulären Spannungsverhältnisse berücksichtigt. Geht man von extrazellulärem Isopotential aus, ist U_e von t und x unabhängig, was die Herleitung vereinfacht. Im allgemeinen ist dies eine sehr gute Approximation. Außerdem sei das Membranruhepotential E_r ebenfalls von t und x unabhängig. Dadurch werden die Ableitungen bzgl. x oder t dieser beiden Variablen Null, womit $U = U_i$ gilt ($U = U_i - U_e - Er$). Ersetzt man U_i durch U in 14.7 und multipliziert beide Seiten mit r_m, so erhält man

$$\frac{r_m}{r_i} \frac{\partial^2 U}{\partial x^2} = i_m r_m \tag{14.8}$$

Der Zylinder einer passiven Nervenzelle hat eine Membrankapazität pro Längeneinheit $c_m = C_m \pi d$ (in F/cm) (siehe Abbildung 14.4). Diese Membrankapazität liegt parallel zum Membranleitwert pro Längeneinheit $g_m = G_m \pi d$ (in S/cm). Daraus ergibt sich der Membranwiderstand $r_m = \frac{1}{g_m} = \frac{R_m}{\pi d}$ (in Ωcm). Dieser liegt in Serie zu einer Spannungsquelle, welche das Membranruhepotential nachbildet. Daraus ergibt sich

$$i_m r_m = \tau_m \frac{\partial U}{\partial t} + U \tag{14.9}$$

wobei τ_m die Zeitkonstante der passiven Membran ist

$$\tau_m = r_m c_m = R_m C_m \tag{14.10}$$

Setzt man die Gleichungen 14.8 und 14.9 gleich und definiert

$$\lambda = \sqrt{\frac{r_m}{r_i}} = \sqrt{\frac{R_m}{R_i} \frac{d}{4}} \tag{14.11}$$

so erhält man die Kabelgleichung 14.1.

Jetzt ist der Weg von einem (stark abstrahierten) Dendrit-Abschnitt zum mathematischen Modell beschrieben. Von diesem einfachen Modell ausgehend kann man wieder Annahmen und Abstraktionen entfernen und damit das Gleichungssystem verfeinern und besser approximieren. Dies soll nun an einigen Beispielen gezeigt werden, bevor der Schritt von dem Dendriten-Abschnitt zu einem Baum von solchen Abschnitten durchgeführt wird.

Geht man statt von dem extrazellulären Isopotential von einem eindimensionalen extrazellulären Strom durch einen extrazellulären Widerstand pro Längeneinheit r_e (in Ω/cm) aus, so ändert man die letzte Definition wie folgt ab

$$\lambda = \sqrt{\frac{r_m}{r_i + r_e}} \tag{14.12}$$

Diese Erweiterung ist besonders bei eng benachbarten, parallelen und aktivierten Mantelkondensatoren (also Dendriten) nötig, da hier $r_e \gg r_i$ werden kann. Bei Bäumen von einzeln aktivierten Neuronen kann man dagegen $r_e = 0$ setzen.

Ist r_i von x abhängig (weil sich der Durchmesser ändert, sich der Dendrit gabelt oder wegen einer Inhomogenität im Zytoplasma), so ist seine Ableitung bzgl. x nicht Null und 14.5 muß erweitert werden zu

$$\frac{\partial U_i}{\partial x^2} = -r_i \frac{\partial i_i}{\partial x} - i_i \frac{\partial r_i}{\partial x} \tag{14.13}$$

Dies erschwert die Herleitung, da die eigentliche Kabelgleichung nur für homogene Kabel gilt. Haben dagegen zwei benachbarte Abschnitte unterschiedliche Eigenschaften, so kann man diese als eigenständige Abschnitte mit homogenen Eigenschaften wie oben modellieren. Um sie mathematisch zu verbinden, muß man Randbedingungen erfüllen, da beispielsweise keine Senke oder Quelle auftreten darf. Für eine Klasse von Dendriten, die diese Bedingungen erfüllen, kann man die einzelnen Abschnitte mathematisch verbinden und einen äquivalenten Ersatzzylinder für einen ganzen Baum berechnen. Hierzu muß man eine Lösung der Kabelgleichung finden, die ja eine partielle Differentialgleichung ist. Dies erreicht man durch geeignete Annahmen und Abstraktionen. Betrachtet man nur einen statisch aufgeprägten Strom oder eine statisch aufgeprägte Spannung, so kann man mathematische Lösungen der einfachen Differentialgleichung 14.2 bestimmen (Variablenseparation).

$$U(X) = A_1 e^X + A_2 e^{-X} \tag{14.14}$$

$$U(X) = B_1 \cosh(X) + B_2 \sinh(X) \tag{14.15}$$

$$U(X) = C_1 \cosh(L - X) + C_2 \sinh(L - X) \tag{14.16}$$

$$2A_1 = B_1 + B_2 = (C_1 - C_2)e^{-L} \tag{14.17}$$

$$2A_2 = B_1 - B_2 = (C_1 + C2)e^{+L} \tag{14.18}$$

Die drei oben genannten Gleichungen (14.14 - 14.16) sind äquivalent. Die Beziehung der freien Variablen untereinander ist darunter angegeben. Man sieht, daß diese Lösungen zwei freie Variablen besitzen, die nun durch Randbedingungen eindeutig bestimmt werden müssen. Je nach Randbedingung wählt man eine geeignete Gleichung (die es ermöglicht, die Unbekannten möglichst leicht zu bestimmen). Betrachtet man ein Kabel von semi-unendlicher Länge ($x = 0$ bis $x = \infty$). Eine aufgeprägte Spannung $U = U_0$ bei $x = 0$ ist eine Randbedingung. Man nimmt an, daß es sich um ein quellen- und senkenfreies Kabel handelt, d.h., durch Elektroden wird nichts hinzugefügt oder entnommen; sie dienen nur zur Aufzeichnung. Die zweite Randbedingung könnte $U = 0$ bei $x = \infty$ sein. Damit wäre $A_1 = 0$ und $A_2 = U_0$ eindeutig bestimmt. In der Realität existieren natürlich keine unendlichen Dendriten. Wenn jedoch das Ende weiter als $4 * \lambda$ vom Meßpunkt entfernt ist, dann ist der Fehler vernachlässigbar. Für endliche Kabel muß man sowohl die Länge als auch die Randbedingungen an beiden Enden definieren. Hier kommen vor allem offene, kurzgeschlossene und feste Spannungen an den Enden in Frage. Je nach Randbedingungen erhält man einen unterschiedlichen Spannungsabfall längs des Kabels. Berechnet man mit Hilfe der Kabelgleichung den Eingangswiderstand, so hat man das nötige Werkzeug, um für einen verzweigten Dendriten einen Ersatzzylinder zu berechnen. Man erhält für den dendritischen Eingangsleitwert

$$G_D = G_{md} A_D F_{dga} \tag{14.19}$$

wobei G_{md} für den Membranleitwert pro Einheitsfläche steht. A_D ist die Membranoberfläche, die aus den Dimensionen des Ersatzzylinders leicht zu bestimmen ist. F_{dga} ist ein Korrekturfaktor, der für bestimmte Typen von Bäumen bestimmt werden kann. F_{dga} ist im allgemeinen eine approximierte Zahl, da die explizite Darstellung mathematisch aufwendig ist (Grundlage bilden oft andere Modelle oder Versuchsreihen). Bis hierher wurde ein Dendrit (Baum) modelliert. Um ein komplettes Neuron zu beschreiben, muß man dessen Eingangsleitwert bestimmen, indem man die Membran des Soma hinzunimmt. Den Eingangsleitwert des Axons kann man meist vernachlässigen

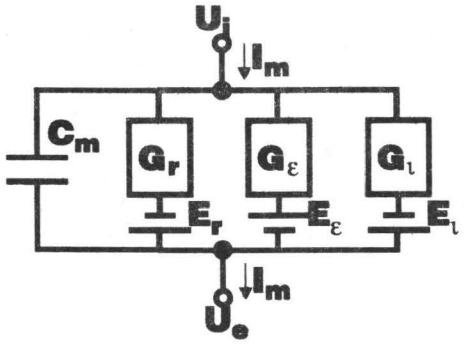


Abbildung 14.5: Membranersatzschaltbild, erweitert um die synaptische Erregung und Inhibition [1]

(außer bei einem besonders dicken Axon). Damit erhält man für den gesamten Eingangsleitwert eines Neurons

$$G_N = G_S + \sum_{j=1}^{n} G_{D_j} \tag{14.20}$$

wobei G_S der Eingangsleitwert des Soma und G_{D_j} die Eingangsleitwerte von n dendritischen Bäumen sind.

Um eine synaptische Erregung und eine synaptische Inhibition zu modellieren, muß das Ersatzschaltbild der Membran ergänzt werden. Dies erreicht man am einfachsten durch Hinzunahme zweier entsprechender Spannungsquellen. (siehe Abbildung 14.5) Nach Kirchhoff folgt für den Membranstrom

$$I_m = C_m \frac{dU_m}{dt} + G_r(U_m - E_r) + G_\varepsilon(U_m - E_\varepsilon) + G_\iota(U_m - E_\iota) \tag{14.21}$$

14.2.2 Compartmental Model

„Compartmental modeling für Neuronen" wurde in den 60er Jahren eingeführt. Man betrachtet dabei Compartments (Einheiten) und deren Verbindungen. Ein Compartment kann für einen dendritischen Baum, für eine Gruppe von Bäumen oder auch nur für einen Abschnitt eines Dendriten stehen. Dies ist von der Problemstellung abhängig. Der Bereich des Neurons, der durch ein einzelnes Compartment simuliert wird, wird als isopotential angesehen. Man geht (wie oben) von gleichförmigen, idealisierten Bedingungen aus, um das einzelne Compartment nicht allzu komplex zu gestalten. Um unterschiedliche Bedingungen (z.B.: unterschiedliche Durchmesser) zu simulieren, teilt man diesen Abschnitt in verschiedene, für sich jedoch homogene, Compartments auf. Es ergeben sich zwei Strategien: einfache Compartments, die durch aufwendige Verschaltung und große Stückzahlen das Geforderte modellieren oder sehr komplexe Compartments, die in geringer Stückzahl übersichtlich verschaltet sind. Hierbei entscheidet, ob man die Vorgänge an einem einzelnen Neuron detailliert betrachten, oder ob man das Verhalten großer neuronaler Verbände untersuchen will.

Ein Compartment besteht aus einem Ersatzschaltbild für die Membran und einem für das intrazelluläre Zytoplasma. Die einzelnen Compartments werden dann direkt miteinander verschaltet. Zur Modellierung der Membran kann man die gleichen Überlegungen wie bei der Kabeltheorie anstellen. (Siehe 14.5) (siehe Abbildung 14.6)

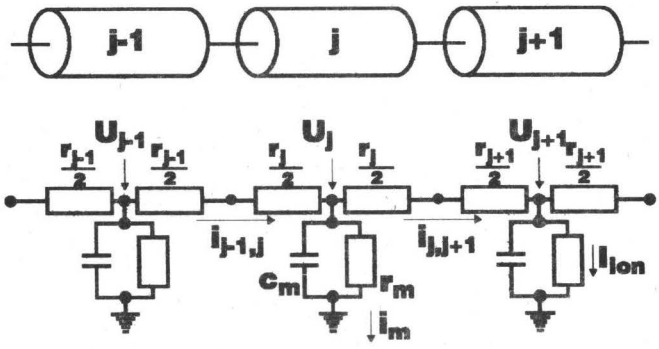


Abbildung 14.6: Eilzelne Compartments und ihr Ersatzschaltbild [1]

Es ergeben sich für diese einfachen Membranschaltbilder für jedes Compartment

$$\hat{i}_{m_j} = i_{j-1,1} - i_{j,j+1} \tag{14.22}$$

$$\hat{i}_{m_j} = \hat{c}_{m_j} \frac{dU_j}{dt} + I_{ion_j} \tag{14.23}$$

14.3 Simulatoren

Ein Vorteil von Ersatzschaltbildern zur Beschreibung von Neuronen liegt darin, daß bereits vorhandene Techniken (Modelle und Programme) zur Simulation elektrischer Schaltkreise voll genutzt werden können. Man braucht lediglich eine neuronenspezifische Oberfläche hinzuzufügen, um die Ein- und Ausgabe den Versuchsanordnungen und anatomischen Gegebenheiten anzupassen. In dem Projekt MOBIS der Universität Kaiserslautern wird das System zur Simulation kombiniert mit vielfältigen Hilfen zur Planung, Durchführung und Verwaltung der Versuche.

Die folgende Graphik (14.7) veranschaulicht den Weg von der anatomischen und physiologischen Untersuchung zum „Compartmental Model". Dieses Diagramm (14.8) stellt eine beispielhafte Simulation des Neurons wieder. Da zur Simulation auch Elektroniksimulatoren in Frage kommen, ist es relativ schwer, einen Überblick über Simulatoren für Neuronen zu schaffen. Einige mögliche Simulatoren, die auf sehr unterschiedlichen Systemen laufen, sind SAAM, ASTAP, SABER, SPICE, NODUS, DENDR, AXON-TREE, NET2, ADVICE, SCEPTRE und BioSim. Zu dem letztgenannten Programm gibt es auch eine PC-Version unter Windows$^{\heartsuit}$. Es basiert auf dem Compartmental Model und ermöglicht das Erstellen von einfachen Verbänden direkt auf der graphischen Oberfläche. Die Kanaleigenschaften der einzelnen Compartments und des Soma sowie die aufgeprägten Ströme können über verschiedene Fenster eingestellt oder als Standardwerte vorgegeben werden. Auch die Eigenschaften der Synapsen sind modifizierbar, wodurch verschiedene Typen (z.B. elektrische Typen, ionische Typen etc.) nachgebildet werden können. Das Programm bietet, zur graphischen Ausgabe der Simulation, für jedes Neuron ein unabhängiges Fenster, in dem Spannungsverläufe an unterschiedlichen Stellen beobachtet werden können. Außerdem gibt es ein größeres Fenster, in dem die Spannungsverläufe im Soma der Neuronen verglichen werden können.

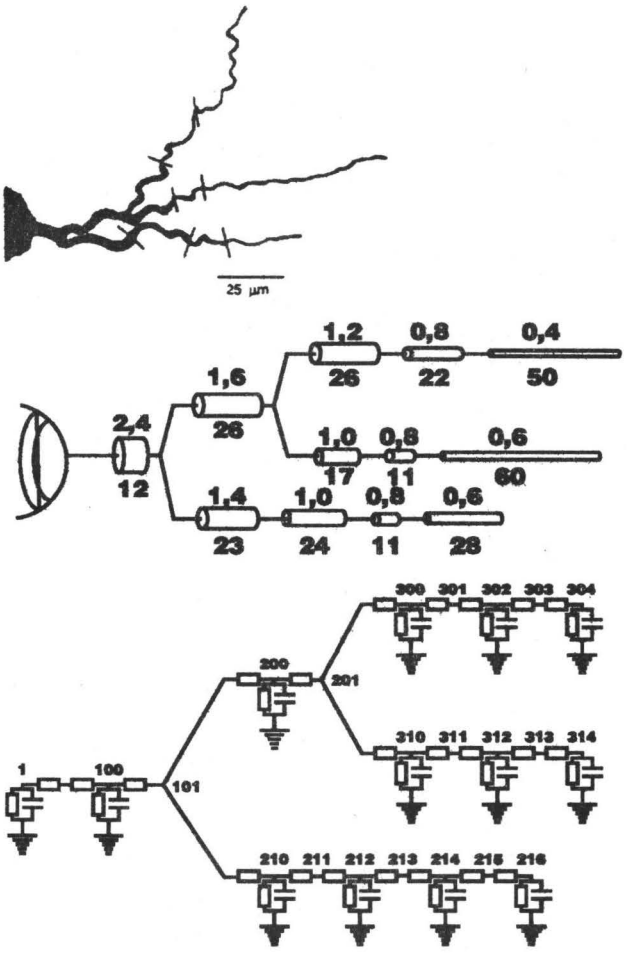


Abbildung 14.7: Weg vom Neuron zum Compartmental Model [1]

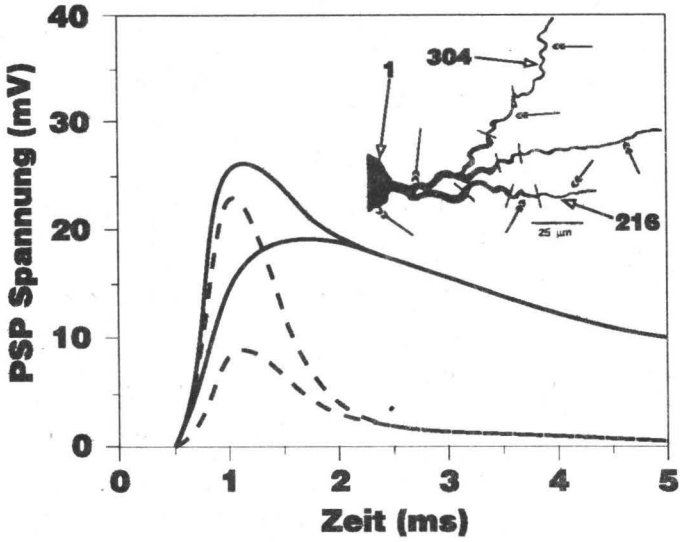


Abbildung 14.8: PostSynaptische Potentiale, simuliert am Compartmental Model des Neurons (letzte Abbildung) [1]

14.4 Zusammenfassung

Neuro-Wissenschaften erreichen Forschungsergebnisse nur durch die enge Verzahnung von biologischen Experimenten mit der Simulation biologischer Systeme. Bei Versuchen mit einem Simulator (BioSimPC) wurde schon bei einfachen Verbänden mit bis zu fünf Neuronen der hohe mathematische Aufwand deutlich. Eine Simulation erfordert die Berücksichtigung von einer großen Menge an Daten, wodurch große und leistungsfähige Computer nötig werden, um vertretbare Rechenzeiten zu erreichen.

14.5 Literatur

[1] *Methods in Neuronal Modeling*–From Synapses to Networks edited by Christof Koch and Idan Segev 1989 A Bradford Book The MIT Press

[2] *Gehirn und Nervensystem*–Verständliche Forschung Ein Spektrum der Wissenschaft Buch

[3] *Human Anatomy and Physiology*– Second Edition Solomon Schmidt Adragna 1990 Saunders College Publishing

[4] *Pocket Medical Dictionary*– Fourteenth Edition edited by Nancy Roper Churchill Livingstone

[5] *Linder Biologie*– 19.Auflage Knodel Bayrhuber J.B.Metzlersche Verlagsbuchhandlung

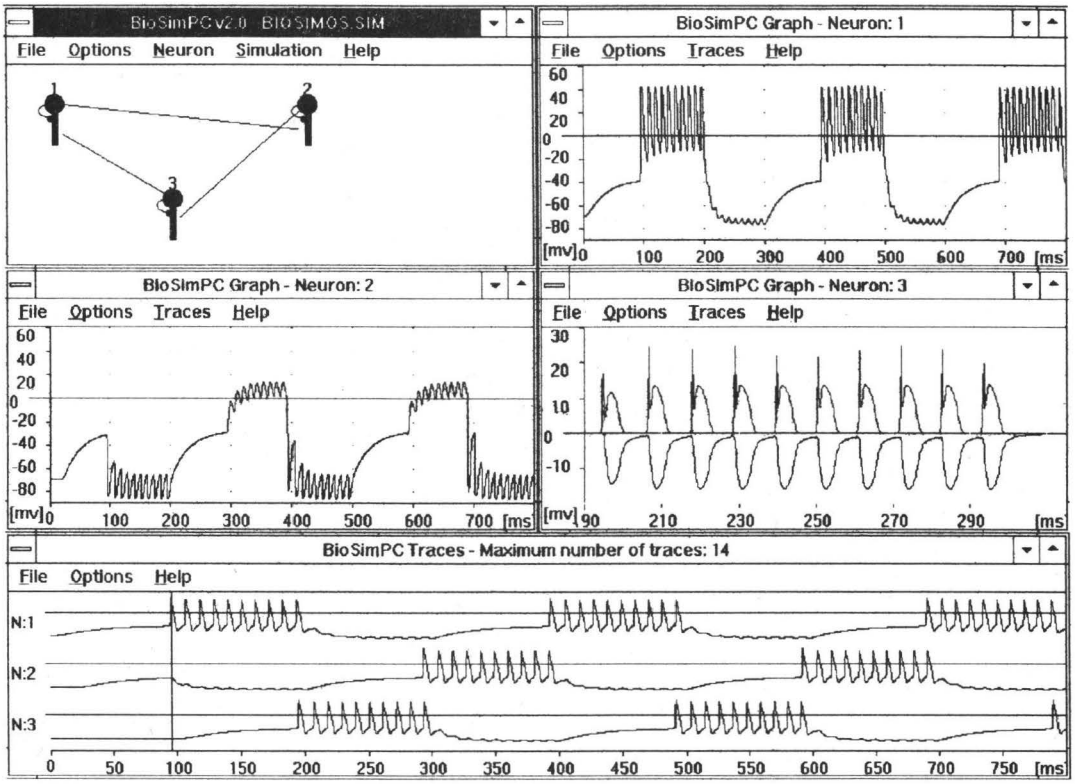


Abbildung 14.9: Simulation mit BioSimPC