D I P L O M A R B E I T

Modulare neuronale Systeme

ausgeführt am Institut für Automation,
Abteilung Mustererkennung und Bildverarbeitung
der Technischen Universität Wien

unter Anleitung von o. Univ. Prof. Dr. W. Kropatsch
und Univ. Ass. Dr. H. Bischof

durch

Elmar Thurner
Peter Jordan-Straße 157/1/5 A-1180 Wien
Matrikelnummer 87 26 857

Wien, am 12. Dezember 1994

Abstract

This thesis gives an introduction and an overview to a new approach in the field of neural networks: Modularity. It will be shown, that a single general purpose network will not fit optimally to any given problem. Contrary, modular networks *learn faster*, have *better generalization abilities* and *higher robustness* and *are easier to extend.*

To investigate the relationship between architecture and function the basic structures, found in modular neural architectures, like *parallelism* (integrative and competitive), *cascades* and *supervisor actor* structures are explored. Furthermore algorithms are presented to estimate the quality of results of networks, to combine the results of networks, to automatically decompose tasks and to combine different architectures.

The theoretical analysed advantages of modular neural networks are demonstrated by experiments in the field of optical character recognition, OCR. For the recognition of printed characters in 20 different fonts by using modularity the training time is reduced to less than a quarter and the miss classification rate by one third in comparison to the single network solution. For the classification of hand written digits using modular neural preprocessing, an improvement of the miss classification rate of 7% is achieved compared to the non modular network.

Dank

Eine Diplomarbeit ist ein wesentlicher Schritt jedes Studenten in Richtung selbstständiges, wissenschaftliches Arbeiten. Sie kann nicht völlig auf sich allein gestellt als Einzelarbeit im gläsernen Turm entstehen.

Für das Gelingen der Diplomarbeit sind Hinweise, Anleitungen und Inspirationen von erfahrenen Personen zur Arbeitsweise und Methodik genauso wie zur technischen Problemstellung von größter Bedeutung. In diesem Zusammenhang möchte ich mich ganz besonders bei Hrn. Prof. Walter Kropatsch für seine stete Unterstützung durch Diskussionen und durch das Diplomanden-Dissertanten-Seminar sowie bei Herrn Dr. Horst Bischof für unzählige wertvolle Gespräche und Anregungen bedanken. Auch meinen Kollegen bei Siemens, Thomas Baumann, Thomas Brunner, Bernhard Knapp, Gottfried Punz und Kutay Yurtsever, die mich während der gesamten Arbeit unterstützt haben, gilt mein besonderer Dank.

Neben dem fachlichen Umfeld muß für die Erstellung der Diplomarbeit auch das soziale Umfeld entsprechen. Die familiäre und berufliche Situation muß es ermöglichen, sich viele Monate lang Zeit und Energie zu nehmen. Deshalb an dieser Stelle mein herzlicher Dank an meine liebe Freundin Claudia Schauer, die mir den Freiraum zur Erstellung der Diplomarbeit eingeräumt und die nötige moralische Unterstützung gegeben hat. Mein besonderer Dank gilt auch meinen Freunden Christian Kasper, Peter Nestler und Herbert Synek, die mir mit ihren Erfahrungen aus Studium und Beruf und mit vielen Anregungen zur Seite gestanden sind. Last not least möchte ich auch meinen Eltern ganz besonders danken für ihre Hilfe und Unterstützung während meines gesamten Studiums.

Zusammenfassung

Neuronale Netze wurden bereits in sehr vielen Anwendungen der Musterekennung erfolgreich eingesetzt. Häufig wurden dabei einzelne große Netze verwendet. Mit den Vorteilen, die diese Netze gegenüber herkömmlichen Algorithmen brachten, entstanden jedoch auch neue Probleme wie unhandhabbar große Trainingszeiten, schlechte Erweiterbarkeit und schlechte Wiederverwendbarkeit.

In dieser Diplomarbeit wird eine Einführung und ein Überblick über eine noch relativ junge Vorgangsweise gegeben: Die Lösung einer Gesamtaufgabe durch Kombination mehrerer Netze innerhalb einer modularen Architektur. Im Unterschied zur existierenden Literatur wird hier erstmals eine Gesamtsicht von Modularität bei neuronalen Netzen präsentiert und nicht nur einzelne Aspekte dieser Thematik.

Nach einer einleitenden Definition grundlegender Begriffe wie *modulares* und *hierarchisches Netz* wird gezeigt, daß modulare neuronale Netze im Vergleich zu großen Einzelnetzen *kürzere Trainingszeiten*, *günstigere Scalingeigenschaften*, *bessere Generalisierungsfähigkeit* und *größere Robustheit* aufweisen.

Basierend auf diesem Zusammenhang zwischen Architektur und Funktion werden die Grundstrukturen, aus denen modulare neuronale Architekturen aufgebaut sind, nämlich *selektive und integrative Parallelität, Kaskadierung, Supervisor Actor-Strukturen und Hierarchie* vorgestellt. Weiters werden Algorithmen zur Schätzung der Qualität der Ergebnisse von einzelnen Netzen, zur automatischen Selektion von Teilnetzen und zur Integration von Ergebnissen paralleler Teilnetze präsentiert sowie Möglichkeiten zur manuellen und automatischen Dekomposition von Aufgabe und System vorgestellt.

Die theoretisch analysierten Vorteile modularer neuronaler Netze werden praktisch anhand mehrerer Experimente zur Buchstabenerkennung (Optical Character Recognition) evaluiert. Am Beispiel der Erkennung gedruckter Buchstaben in 20 verschiedenen Fonts wird durch Modularisierung die Trainingszeit auf weniger als ein Viertel und die Fehlklassifikationsrate um ein Drittel reduziert. Am Beispiel der Klassifikation von handgeschriebenen Ziffern wird durch modulares neuronales Preprocessing die Fehlklassifikationsrate um 7% gesenkt.

Inhaltsverzeichnis

1 Einleitung

Neuronale Netze (NN) erlebten in den letzten Jahren eine stürmische Entwicklung. Nach der Einführung von nichtlinearen Units und den dazu passenden Lernalgorithmen in den achtziger Jahren (Backpropagation z.B. durch Werbos [Werb74] und Rumelhart et al. [Rume86]) wurde das Thema neuronale Netze zum zweiten Mal in der Geschichte zum Mittelpunkt zahlreicher Aktivitäten in Forschung und Entwicklung. Viele neue Konferenzen und Fachzeitschriften entstanden. Die Gründe für diese rasante Entwicklung neuronaler Systeme liegen wohl in erster Linie in den innovativen Eigenschaften von neuronalen Netzen im Vergleich zu traditionellen Algorithmen.

Anstatt eine gegebene Aufgabe durch einen für die Aufgabe speziell entwickelten und fixen Algorithmus zu lösen, löst ein neuronales Netz die Aufgabe durch einen Prozeß des Lernens anhand von Trainingsbeispielen. Der Vorteil dieser Vorgangsweise liegt auf der Hand: Der konkrete Algorithmus zur Lösung der Aufgabe muß gar nicht erst gefunden werden. Dies ist besonders bei Aufgabenstellungen von Bedeutung, bei denen eine explizite Lösung überhaupt nicht gefunden oder nur schwer explizit beschrieben werden kann.

Neuronale Netze sind universell einsetzbare Funktionsapproximatoren [Horn91]. Sie sind adaptiv und zeigen Eigenschaften von Selbstorganisation. Wissen ist in Netzen implizit und verteilt repräsentiert, im Unterschied zur expliziten und lokalen Wissensrepräsentation bei traditionellen Algorithmen [Rume86]. Aus diesem Grund sind Netze in der Lage, Fälle zu lösen, auf die sie nicht trainiert wurden (unseen Cases). Aufgrund der verteilten Repräsentation von Wissen sind Netze außerdem tolerant gegenüber Fehlern in den Daten und gegenüber Fehlern im Netz selbst.

Waren es zu Beginn des “Netze-Booms” meist einfache Aufgaben zu deren Lösung einzelne neuronale Netze verwendet wurden, so sind die heutigen Problemstellungen bereits deutlich komplexer. Bei komplexen Aufgaben zeigen sich neben den genannten Vorteilen von neuronalen Netzen auch gravierende und teilweise unüberwindbare Nachteile:

- Die Anforderungen an die Rechenzeit, besonders für das Training, wachsen in Größenordnungen, die selbst mit modernen Computern zu Trainingszeiten von Stunden und Tagen führen und so Online- oder gar Echtzeit- Einsätze verhindern.

- Aufgrund der allgemeinen Verwendbarkeit von neuronalen Netzen werden bei Aufgaben, für die explizite oder modellbasierte Lösungen verfügbar sind, mit Netzen meist weniger genaue Ergebnisse erzielt.
- Die neuronalen Lösungen sind trotz der Allgemeinheit der Lernalgorithmen schwer erweiterbar oder auf andere Aufgabenstellungen übertragbar.
- Der Prozeß der Lösungsfindung durch ein neuronales Netz ist schwer nachvollziehbar. Erklärungskomponenten, wie bei Expertensystemen üblich, können nicht realisiert werden.
- Noch immer ist die Generalisierungsfähigkeit von Netzen zu gering (z.B. aufgrund von Translations-, Rotations- und Scalierungsvarianz)

Parallel zu den immer größer werdenden Aufgaben hat sich eine theoretische Basis für das Gebiet der NN entwickelt [Hert91]. Neuronale Netze werden nun nicht mehr rein als selbstorganisierende lernende Systeme, sondern auch als adaptive nichtlineare statistische Approximationsverfahren betrachtet. Dadurch wurde klar, daß ein einzelnes, allgemein verwendbares neuronales Netz nicht in der Lage sein kann, beliebig komplexe Aufgaben optimal zu lösen [Jaco90]. Eine geeignete Datenvorverarbeitung, und vor allem eine günstige Zerlegung des einzelnen Netzes in kooperierende Teilnetze ist für eine bestmögliche Lösung einer komplexen Aufgabe erforderlich. Die Idee hinter der Zerlegung der Gesamtaufgabe in weniger komplexe Teilaufgaben ist die Integration von Wissen über die Aufgabe in das Netz, um so die meisten der genannten Probleme in den Griff zu bekommen.

1.1 Ziel der Diplomarbeit

Das Hauptziel dieser Arbeit ist es, eine umfassende Darstellung des Themas Modularität bei neuronalen Netzen zu geben. Im Unterschied zu vielen bereits vorliegenden Arbeiten die sich jeweils mit einem Teilaspekt von Modularität bei Netzen beschäftigen [Hryc92][Jord94][Rogo94], soll hier eine Gesamtsicht des Themas Modularität präsentiert werden. Die verschiedenen Aspekte und technischen Realisierungen von Modularität bei Netzen sollen zueinander in Beziehung gesetzt werden. Wesentlich dabei ist die Frage, wie man von der Aufgabenstellung einerseits und den verschiedensten existierenden Lösungsansätzen und Lösungsideen andererseits, zu einem realen, optimal funktionsfähigen System kommt.

1.2 Struktur der Diplomarbeit

Die Diplomarbeit besteht aus 12 Kapiteln. Jedes Kapitel enthält eine kurze Einleitung, sodaß es weitgehend auch für sich alleine gelesen werden kann. Trozdem bestehen unter den Kapiteln gewisse Abhängigkeiten. Die Struktur der Arbeit und die Reihenfolge, in der die Kapitel aufeinander aufbauen, ist in Form eines Abhängigkeitsgraphen in Bild 1.1 dargestellt.

In dem nun anschließenden zweiten Kapitel werden zunächst wichtige Begriffe definiert und das verwendete Vokabular eingeführt. Darauf aufbauend werden im dritten Kapitel die Vorteile modularer Architekturen gegenüber großen Einzelnetzen detail-

liert dargestellt. Im vierten Kapitel werden die Grundarchitekturen beschrieben, aus denen jede modulare Netzwerkarchitektur aufgebaut ist.

In den folgenden vier Kapiteln werden Details zu den Themen Qualitätsmessung bei Netzen, Integration (Fusion) von redundanten Ergebnissen, selbstorganisierende Selektion von Teilnetzen (selbstorganisierende Dekomposition) und Kombination von Integration und Selektion erarbeitet.

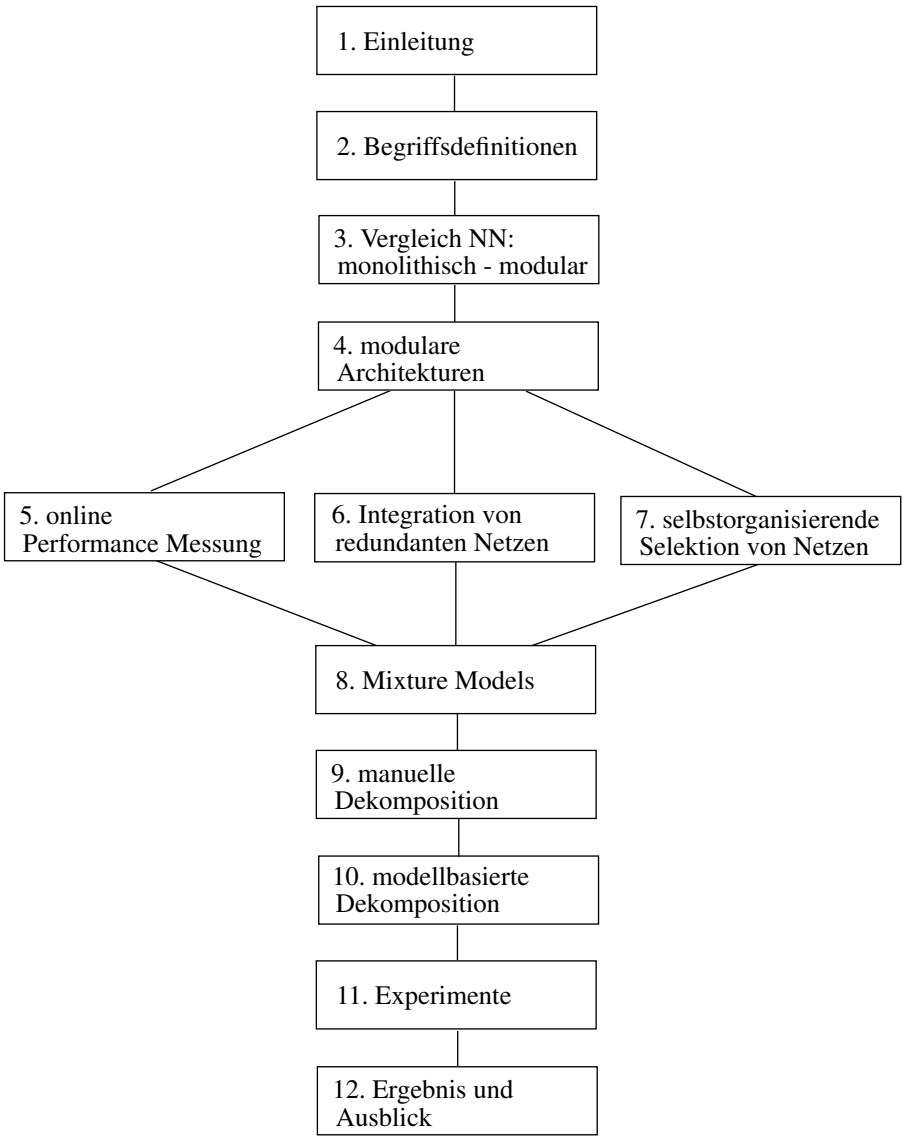


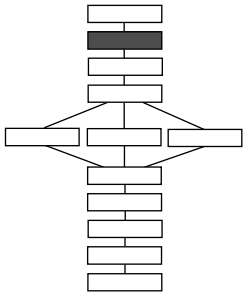
Bild 1.1 *Struktur der Diplomarbeit*

Kapitel 9 geht der Frage nach, wie eine komplexe Aufgabenstellung für ein neuronales System “manuell” in weniger komplexe Teilaufgaben zerlegt werden kann, um dadurch die Vorteile von modularen Architekturen aus Kapitel 3 auch real umsetzen zu können.

Kapitel 10 erläutert wie traditionelle Algorithmen und mathematische Modelle verwendet werden können, um daraus modulare neuronale Netzwerke zu konstruieren.

In Kapitel 11 werden einige Experimente präsentiert, die die Vorteile von modularen Systemen untermauern.

Im letzten Kapitel schließlich werden die Ergebnisse zusammengefaßt sowie Zukunftsperspektiven umrissen. Außerdem enthält dieses Kapitel eine Übersicht über alle behandelten Architekturen und Dekompositionsverfahren.



2 Begriffsbestimmungen

In diesem Kapitel werden grundlegende Begriffe definiert, auf denen alle weiteren Kapitel basieren. Gleichzeitig wird indirekt in die objektorientierte Sichtweise der Problematik eingeführt. Die verwendete Notation ist in Anhang A beschrieben.

2.1 Neuronales Netz

Ein neuronales Netz[1] (**NN**) ist ein Baustein, der Inputdaten auf Outputdaten abbildet. Diese Abbildung $f: I \subseteq \Re^n \rightarrow O \subseteq \Re^m$ wird im Unterschied zur herkömmlichen Programmierung anhand von Trainingsbeispielen erlernt.

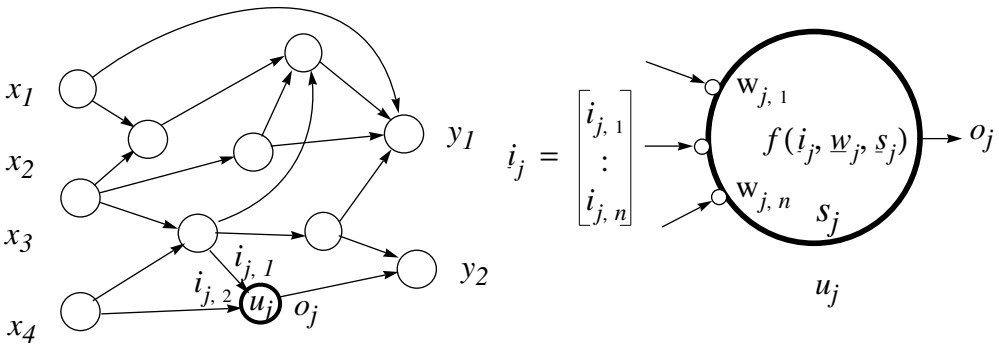


Bild 2.1 *Schematische Darstellung eines neuronalen Netzes und einer Unit des Netzes*

1. auch: künstliches neuronales Netz (-werk), neurales Netz [Köhl90], englisch: neural network, parallel distributed processing [Rume86],

Ein NN besteht aus einer Menge U von Units u, die in Form eines beliebigen, zyklischen[2] oder azyklischen[3], gerichteten Graphen t miteinander verbunden sind (Bild 2.1).

Eine **Unit** u_j berechnet aus einem Inputvektor $\underline{i}_j \in I_j \subseteq \Re^{n_j}$ einen skalaren Output $o_j \in O_j \subseteq \Re$. Zu diesem Zweck enthält die Unit einen **Gewichtsvektor** $\underline{w}_j \in W_j \subseteq \Re^{n_j}$ und einen **internen Zustand** $\underline{s}_j \in S_j \subseteq \Re^p$. Gewichtsvektor $\underline{w}_j$ und interner Zustand $\underline{s}_j$ sind die **lokalen Daten** der Unit.

Neben den lokalen Daten besitzt die Unit eine Propagate-Funktion und eine Lernfunktion (siehe auch Bild 2.2). Die **Propagate-Funktion**[4] $f_j: I_j \times W_j \times S_j \rightarrow S_j \times O_j$ berechnet aus dem Input $\underline{i}_j$, dem Gewichtsvektor $\underline{w}_j$ und dem internen Zustand $\underline{s}_j$ den Output o_j und einen neuen internen Zustand $\underline{s}_j$. Die Auswertung der Propagate-Funktion ist der sog. **Recall** bzw. **Test** der Unit. Die Komponenten des Gewichtsvektors $\underline{w}_j$, die Gewichte, sind die **freien Parameter** der Unit. Gemeinsam mit dem internen Zustand und der Propagate-Funktion bestimmen die Gewichte das Verhalten der Unit beim Recall.

Zur Adaptierung des Verhaltens der Unit u_j beim Recall besitzt die Unit die **Lernfunktion** $l_j: E_j \times W_j \times S_j \rightarrow W_j \times G_j$, die aus dem **Fehlerinput** $\underline{e}_j \in E_j \subseteq \Re^q$, dem Gewichtsvektor $\underline{w}_j$ und dem internen Zustand $\underline{s}_j$ einen neuen Gewichtsvektor $\underline{w}_j$ und den **Fehleroutput** $\underline{g}_j \in G_j \subseteq \Re^{n_j}$ berechnet. Neben den definierten Inputs $\underline{i}_j$ und $\underline{e}_j$ arbeiten die Funktionen einer Unit ausschließlich auf den lokalen Daten der Unit, d.h. jede Unit arbeitet ausschließlich lokal.

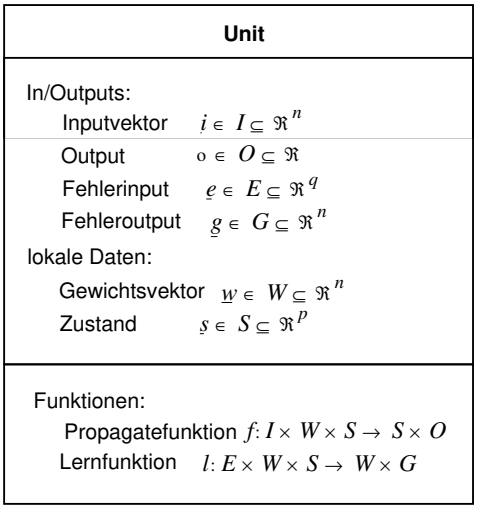


Bild 2.2 *Komponenten einer Unit*

2. bei rekurrenten Netzen [Ulbr92], bzw. vollverbundenen Netzen z.B. Hopfield Network [Hopf88].
3. bei Singllayer Feed Forward-Netzen, z.B. Topographic Feature Maps (Kohonen NN) [Koho89], und Multilayer Feed Forward-Netzen z.B. Back Propagation (**BP**) [Rume86].
4. Die Propagate-Funktion (auch **Update-Funktion**) wird manchmal unterteilt in **Aktivierungsfunktion** f und **Output-Funktion** g.

Der Graph $t = < U,C >$ definiert die Verbindungsstruktur (**Architektur**) des NN. Die Knoten u_j des Graphen sind die Units des Netzes. Die Verbindungen $c_i \in C \subseteq U \times U$ sind die Kanten des Graphen. Sie sind gerichtet. Zwei Units u_1 und u_2 können demnach in zwei Richtungen miteinander verbunden werden: $c_1 = <u_1,u_2>$ und $c_2 = <u_2,u_1>$. Dies wird für die oben angeführte Schreibweise eines propagierten Fehlers verwendet. Bidirektionale Verbindungen müssen durch die Verwendung von zwei getrennten Verbindungen modelliert werden.

Im Unterschied zur Definition in [Bisc93] ist zu beachten, daß der Fehler als Input der Unit angesehen wird und daß jede Unit auch einen Fehleroutput liefert. Außerdem sind die Gewichte den Units und nicht den Verbindungen zugeordnet. Dies ermöglicht ein rekursives Schema der Konstruktion von beliebig komplexen Anwendungen. Jede Anwendung kann aus rein lokal arbeitenden Processing Primitives zusammengesetzt werden. Die Aufgabe der Verbindungen ist es, den Datenaustausch zu ermöglichen und eventuell notwendige Formattransformationen durchzuführen. Der Informationsgehalt der Daten wird nur durch die Units, nicht aber durch Verbindungen verändert.

2.2 Neuronales System

Ein neuronales System ist ein Software-System (Programm), in dem zumindest ein NN enthalten ist, das wesentlich an der Gesamtfunktionalität des Systems teilhat. Wesentlich bedeutet, daß das System ohne dem NN nicht funktionieren würde. Insbesonders ist dies ein System, das nur aus Netzen aufgebaut ist, aber auch ein System, in dem eine große Zahl nicht neuronaler Komponenten enthalten ist.

Der Begriff “neuronales System” wird in dieser Arbeit häufig anstelle von NN verwendet, um eine allgemeinere Sichtweise anzuzeigen.

2.3 Modulares System

Ein System ist modular, wenn es aus identifizierbaren Einzelteilen besteht, wobei jeder Teil seinen eigenen Zweck und seine eigene Funktion erfüllt [Boer92].

Im technischen Sinne bedeutet dies: Ein System ist modular aufgebaut, wenn es aus Funktionsbausteinen (Modulen) zusammengesetzt ist, wobei jedes Modul einen abgeschlossenen Teil der Gesamtaufgabe bearbeitet.

Aus funktionaler Sicht ist ein Modul eine Menge von Funktionen, die auf den selben Daten arbeiten. Das wesentliche Kriterium ist die Lokalität der gemeinsamen Daten.

Aus objektorientierter Sicht ist ein Modul eine Menge von lokalen Daten, die durch eine oder mehrere Funktionen, die mit diesen Daten arbeiten, außerhalb des Moduls sichtbar und verwendbar werden.

In Verallgemeinerung der Definition aus Kapitel 2.1 “Neuronales Netz” folgt hier nun eine formale Definition eines modularen Systems. Ein vergleichbarer, aber mehr auf neuronale Netze zugeschnittener Ansatz ist in [Bott91] beschrieben. Ein anderer vergleichbarer Ansatz wird in [Roha92] vorausgesetzt. Die hier gegebene Moduldefini-

tion geht von einer Abbildung *f* pro Modul aus. In Anhang B ist eine erweiterte Moduldefinition gegeben, bei der das Modul *q* verschiedene Abbildungen enthält. Außerdem wird in Anhang B ein konkretes Beispiel für die Modellierung unter Verwendung dieser Definition gegeben.

Ein modulares System besteht aus einer Menge *M* von Modulen m_i, die untereinander verbunden sind.

Modul: Ein Modul (Bild 2.3) ist ein Objekt, das **n Inputgrößen** $\underline{x}_i \in I_i \subseteq \Re^{n_i}, \quad i = 1...n$ auf **m Outputgrößen** $\underline{y}_j \in O_j \subseteq \Re^{m_j}, \quad j = 1...m$ abbildet. Mit $I = I_1 \times I_2 \times ... \times I_n$, $O = O_1 \times O_2 \times ... \times O_m$ und den **lokalen Daten** $L = L_1 \times L_2 \times ... \times L_p \qquad \underline{l}_o \in L_o \subseteq \Re^{p_o}, \quad o = 1...p$ des Moduls, ist diese Abbildung eine vektorwertige Funktion $\underline{f}: I \times L \to O \times L$.

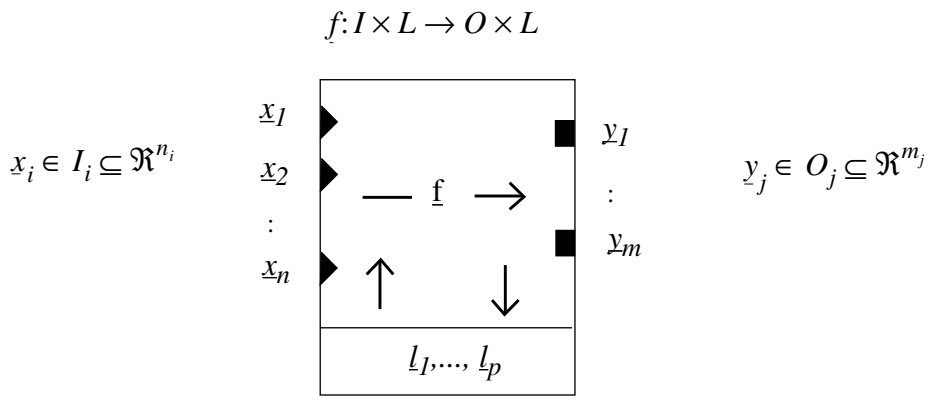


Bild 2.3 *Schematische Darstellung eines Moduls*

Architektur: Der Graph[5] $t = \langle M, g, C \rangle$ mit *M* der Menge der Module, *C* der Menge der Labels (Namen) der Kanten und *g* der Funktion $g : C \to \langle m_a, m_b \rangle$ mit $m_a, m_b \in M$, definiert die Architektur (Verbindungsstruktur) eines modularen Systems. Die Kanten des Graphen sind die Verbindungen zwischen den Modulen. Sie sind gerichtet. Zwei Module m_1 und m_2 können demnach in zwei Richtungen miteinander verbunden werden: $c_1 \to \langle m_1, m_2 \rangle, \quad c_2 \to \langle m_2, m_1 \rangle$. Im Unterschied zur Notation in 2.1 “Neuronales Netz” können hier mehrere Verbindungen in einer Richtung zwischen zwei Modulen existieren. Ein Modul kann auch mit sich selbst verbunden werden. Damit können rekurrente Strukturen modelliert werden.

Für die praktische Anwendung ist statt der oben verwendeten, in der Graphentheorie üblichen Schreibweise [Hara69] folgende semantisch gleichwertige Schreibweise günstiger:
Der Graph $t = \langle M,C \rangle$ mit $C \subseteq \langle m_a, \underline{y}_j \rangle \times \langle m_b, \underline{x}_i \rangle$ mit $m_a, m_b \in M, \ \underline{y}_j \in O_j, \ \underline{x}_i \in I_i$ besteht aus einer Menge *M* von Modulen und einer

5. Hier handelt es sich um eine verallgemeinerte Definition eines Graphens, da nicht nur binäre, sondern auch n-wertige Relationen zugelassen werden.

Menge C von Verbindungen zwischen den Modulen. Die Verbindungen verbinden einen beliebigen Ausgang $\underline{y_j}$ eines Moduls mit einem beliebigen Eingang $\underline{x_i}$ eines anderen Moduls.

2.4 Modulares Neuronales Netz

Intuitiv könnte man sagen, daß ein NN modular ist, wenn seine Knoten und Verbindungen einen Graph bilden, in dem es mehrere Teilgraphen gibt, in dessen Inneren die Verbindungsdichte wesentlich größer ist, als die Verbindungsdichte zwischen den Teilgraphen. Man unterscheidet Modularität im großen Maßstab (Large Scale Modularity) von Modularität im kleinen Maßstab (Small Scale Modularity). Bei **Large Scale-Modularität**[6] ist das Gesamtnetz aus unabhängigen Teilnetzen aufgebaut, wobei zwischen den Teilnetzen nur Verbindungen vom Outputlayer eines Teilnetzes zum Inputlayer eines anderen Teilnetzes existieren. Dies ist vor allem dann der Fall, wenn die Art der Teilnetze verschieden ist (supervised, unsupervised, usw.). Bei **Small Scale-Modularität** lernen alle Units des Netzes nach demselben Lernverfahren. Die modulare Architektur entsteht lediglich durch unterschiedliche Verbindungsdichten innerhalb des Netzes.

Ein vollverbundenes Multi Layer-Perceptron ist nicht modular, obwohl die einzelnen Layer als Module angesehen werden könnten, weil innerhalb des Netzes aufgrund der Vollverbindung keine Unterschiede in den Verbindungsdichten vorhanden sind.

Eine mathematisch exakte Definition von Modularität, die für jedes denkbare Netz eindeutig entscheidet, ob das Netz modular ist, kann nur schwer gefunden werden[7]. Auch läßt sich Large Scale-Modularität nicht immer eindeutig von Small Scale-Modularität unterscheiden. Vielfach sind die Übergänge fließend.

Bild 2.4 zeigt ein Beispiel für ein einfaches modulares NN[8]. Der Outputlayer von m_1 (rechts im Bild) entspricht der ersten bis dritten Unit des zweiten hidden Layers des Gesamtnetzes (links im Bild). Der Inputlayer von m_3 ist für die Moduldarstellung zusätzlich eingefügt. Wie bei allen Backpropagation Netzen ist es die Aufgabe des Inputlayers, die Daten auf die hidden Units zu verteilen. Er hat keinen Einfuß auf die Daten selbst (Transparents). So hat auch der für m_3 zusätzlich eingefügte Inputlayer keine funktionelle, sondern lediglich strukturelle Bedeutung. m_3 besitzt keinen hidden Layer.

---

6. Netze mit Large Scale-Modularität werden auch large scale neural networks genannt [Iwat90], [Mori90].

7. Z.B. ist bei Architekturen mit Large Scale-Modularität ein Modul (Teilnetz) intern meist vollvernetzt. Die Vollvernetzung ist jedoch keine notwendige Voraussetzung für die Modularität der Architektur.

8. Die Architektur des Beispiels für das Backpropagation-Netz (Unitdarstellung) ist aus [Boer92] entnommen. Boers definiert Modul als Gruppe untereinander nicht verbundener Units, wobei jede einzelne Unit dieser Gruppe mit den selben Units anderer Gruppen verbunden ist. Zu beachten ist, daß sich aufgrund der wesentlich allgemeineren Definition eines Moduls hier das Gesamtnetz nur aus drei statt aus sieben Modulen zusammensetzt.

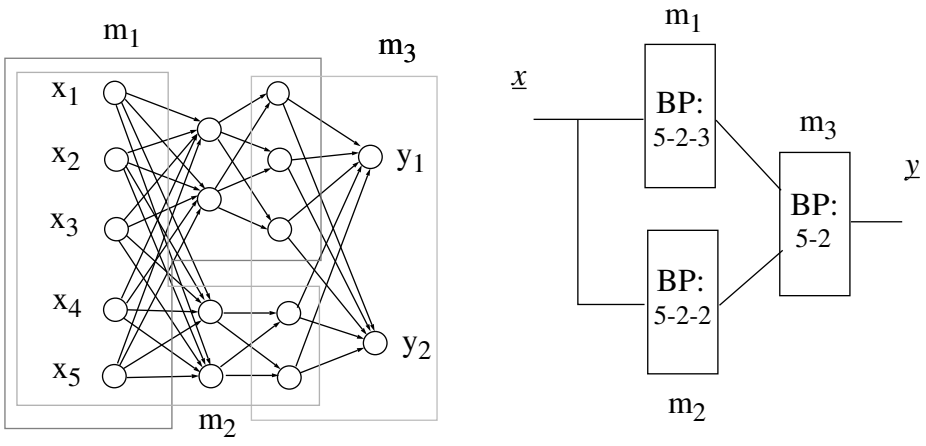


Bild 2.4 *Ein modulares Backpropagation-Netz. Links Darstellung der Units (Small Scale-Modularität), rechts Moduldarstellung (Large Scale-Modularität: 3 unabhängige Teilnetze, Schreibweise 5-2-3: fünf Inputunits, zwei hidden Units, drei Outputunits)*

2.5 Hierarchisches System

Ein hierarchisches System ist ein modulares System, wobei es mindestens ein Modul enthält, das selbst wieder aus Modulen aufgebaut ist (Bild 2.5). Ein Modul, das aus Untermodulen besteht, heißt **strukturiertes Modul** oder *Teilsystem*. Module, die nicht weiter strukturiert sind, heißen **unstrukturierte Module**. Das gesamte System ist ein strukturiertes Modul, das **Root Modul** des Systems.

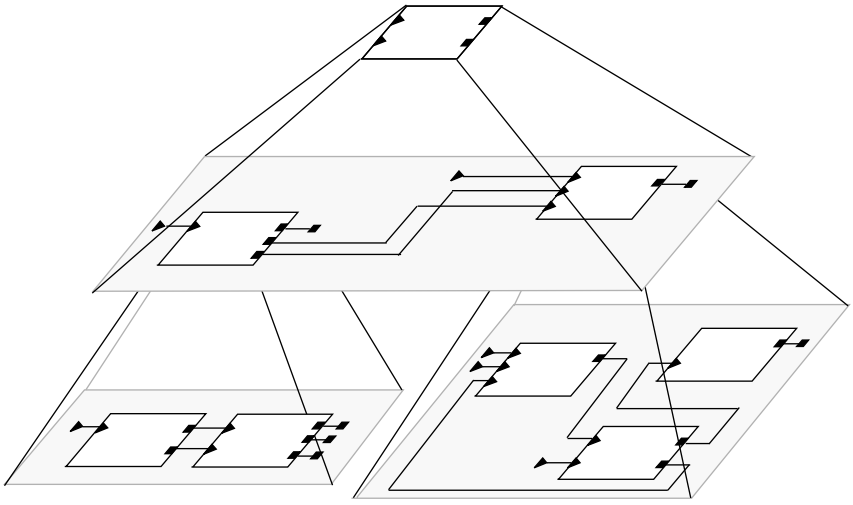


Bild 2.5 *Schematische Darstellung eines hierarchischen Systems*

2.6 Hierarchisches Neuronales Netz

Ein modulares NN wird als hierarchisch bezeichnet, wenn der Graph t, der aus den Units und Verbindungen des Netzes gebildet wird, eine hierarchische Struktur aufweist. D.h. der Graph läßt sich in Schichten einteilen, wobei jede Unit einer Schicht nur mit lokal benachbarten Units der nächstfolgenden Schicht verbunden ist. Wie in [Bisc92] bemerkt, ist ein hierarchischer Graph auch ein azyklischer Graph. Das einfachste hierarchische NN ist ein baumförmig aufgebautes Backpropagation-Netz (Bild 2.6).

Weitere Beispiele für hierarchische NN sind

- das Neokognitron [Fuku84],
- Hierarchien von Expert Networks [Jord92],
- biologisch motivierte NN [Barr90],
- pyramidale NN [Bisc93] und
- neural Trees [Tai93].

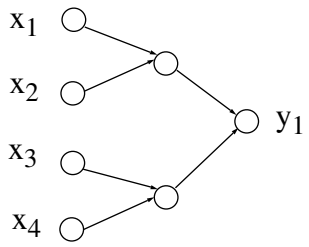
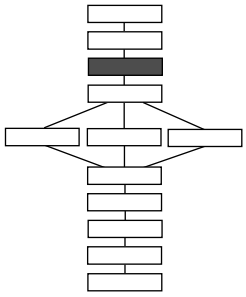


Bild 2.6 *Architektur eines einfachen hierarchischen NN*

2.7 Hybrides System

Ein modulares System wird als hybrid bezeichnet, wenn es aus Modulen unterschiedlicher Technologie besteht (z.B.: regelbasiertes Expertensystem gekoppelt mit NN) und die Bedeutung dieser Module im Gesamtsystem gleichwertig ist. Ein NN mit einem datenvorverarbeitenden Modul zur Histogrammequalisation ist z.B. kein hybrides System. Das System erfüllt seine Aufgabe auch ohne der Vorverarbeitung, allerdings mit schlechterer Performance.



3 Gegenüberstellung monolithischer und modularer neuronaler Netze

Die intuitive Begründung von Modularität bei neuronalen Systemen ist, daß komplexe Problemlösungsfähigkeiten (wie z.B. die meisten Probleme der Bildverarbeitung) nicht durch einzelne (auch noch so große) Netze erlernt werden können. Integration von Vorwissen über die Aufgabe, die Daten und die Eigenschaften der Lernalgorithmen ist notwendig, um die gestellte Aufgabe optimal, d.h. mit bestmöglicher Performance zu lösen (zur Messung der Performance siehe Anhang C). Neben der passenden Datenaufbereitung ist die Wahl einer geeigneten Netzarchitektur, d.h. Modularisierung, eine wesentliche Möglichkeit, Vorwissen zu integrieren.

In diesem Kapitel folgt nun eine detaillierte Betrachtung der durch Modularisierung erzielbaren Vorteile. Dabei wird, zur besseren Darstellung der Vorteile, eine optimale Dekomposition der Gesamtaufgabe in Teilaufgaben vorausgesetzt. Diese optimale Dekomposition zeichnet sich durch folgende Merkmale aus.:

- Die Komplexität der Teilprobleme[1] ist kleiner ist als die Komplexität des Gesamtproblems.
- Im Falle einer automatischen Dekomposition ist die Komplexität der Dekomposition kleiner als die Komplexität der Gesamtaufgabe.
- Die Dekomposition erfolgt in einer Art und Weise, daß Teilaufgaben direkt Teilnetzen zugeordnet werden können.
- Die Teilaufgaben können Teilnetzen zugeordnet werden, die zum Gesamtsystem hinzufügbar sind, ohne dabei das Gesamtsystem neu trainieren zu müssen.

1. Die Komplexität einer Aufgabe sei hier die Komplexität der besten Lösung der Aufgabe. Die Komplexität einer Lösung wird meist durch die für die Lösung erforderliche Rechenzeit ausgedrückt. Je nach Anwendung sind aber auch andere Komplexitätsmaßzahlen, wie z.B. der Speicherbedarf der besten Lösung, möglich.

- Beim Training und beim Recall (Test) der resultierenden Architektur wird an den richtigen Stellen zwischen den einzelnen Teilnetzen unterschieden.
- Die Teilnetze sind ungefähr gleich groß.

3.1 Trainingszeit

Die benötigte Zeit für Training und Recall ist für den konkreten Einsatz von NN eine über den Erfolg entscheidende Größe. Auch in der Forschung lassen sich manche Ideen und Ergebnisse aufgrund von zu großen Lernzeiten nicht validieren.

Bild wird eingeklebt + eintragen Vergleich mit Gehirn (ca. 1011 Neuronen je 103 -104 Verbindungen)

Bild 3.1 *Das Simulationsdilemma der neuronalen Netze*

Bild 3.1 zeigt das Ergebnis einer Studie [DARP89] in der für verschiedene Anwendungsgebiete für NN die benötigten Rechenleistungen und Speicherkapazitäten abgeschätzt wurden. Zugrundegelegt wurden hauptsächlich einfache Netzwerkarchitekturen mit großen Einzelnetzen. Selbst für heutige Möglichkeiten sind die erforderlichen Rechenleistungen (10^9bis 10^{12} **C**onnection **U**pdates per **S**econd, CUPs), die beispielsweise für Echtzeitanwendungen in der Bildverarbeitung benötigt werden, enorm. Diese Rechenleistung ist jedoch im Vergleich zu biologischen neuronalen Netzen immer noch relativ gering. Das menschliche Gehirn besteht aus ca. 10^{11} Neuronen und ca. 10^{14}-10^{15} Verbindungen (Synapsen) [Pinz94]. Bei einer Verarbeitungsgeschwindigkeit von ca. 100 Hz ergibt dies eine maximale Rechenleistung von 10^{16} CUPs.

Es existieren mehrere Ansätze diese Problematik in der technischen Realisierung in den Griff zu bekommen:

- Verwendung von schnelleren Lernverfahren am Einzelnetz, z.B. Quickprop [Fahl88], Netzwerke höherer Ordnung [Tana89], One Pass Learning, aufgabenspezifische algorithmische Optimierungen

- Spezialhardware (z.B. Hardware-Verdrahtung der bei NN wichtigen mathematischen Operatoren[2], spezielle Neurochips)
- Massive Parallelisierung

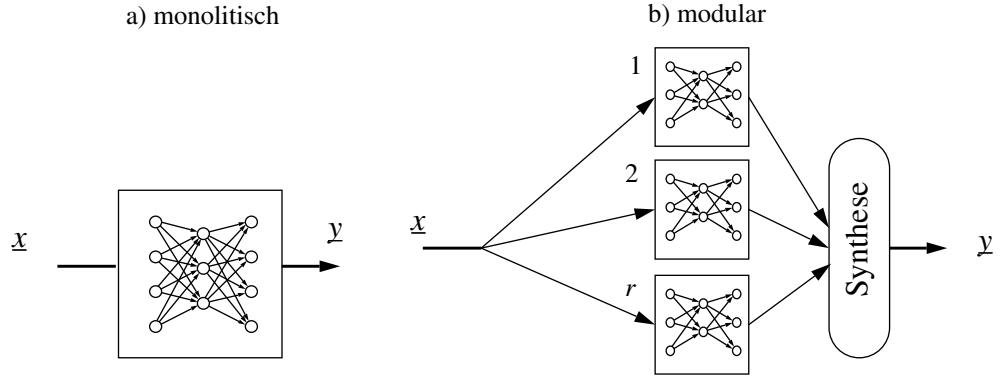
All diese Ansätze weisen in realistischen Anwendungen wesentliche Nachteile auf: Schnellere Lernverfahren sind meist nur für ganz konkrete Aufgabenstellungen verwendbar, da sie von Voraussetzungen ausgehen, die im generellen Anwendungsfall zum Entarten des Rechenaufwandes führen können. Spezialhardware ist aufgrund der zu geringen Stückzahl nicht wirtschaftlich und massive Parallelisierung (eine Unit ist ein Prozessor) ist in der technischen Realisierung, aufgrund der um Ordnungen größeren Verbindungszahl als Unitzahl, nicht sinnvoll.

Abhilfe schafft Modularisierung. Das Ziel dabei ist, bei neuronalen Systemen durch Zerlegung der Aufgabe signifikant Trainingszeit einzusparen. Dies läßt sich an zwei Beispielen demonstrieren: 1. an Netzen bei denen die Trainingszeit stark (polinomiell oder stärker) mit der Problemgröße steigt und 2. für Aufgaben die aus sehr verschiedenartigen Teilaufgaben bestehen:

Zu 1.: Polynomielles Scaling-Verhalten des Einzelnetzes

Das Scaling beschreibt den Zusammenhang zwischen Problem- bzw. Netzgröße und Zeitanforderungen für Training und Recall eines bestimmten Lernalgorithmus. Falls der Rechenaufwand des Lernverfahren für das Einzelnetz polynomiell ($O(n^x)$ mit n Anzahl Daten oder Anzahl Gewichte, $x>1$ d.h. mindestens stärker als linear) mit der Größe der Aufgabe wächst, ist der Rechenzeitbedarf für eine modulare Architektur signifikant günstiger. Für drei Layer-Netzwerke mit sigmoiden Schwellwertunits und Error Backpropagation[3] ist die Trainingszeit, entsprechend einer heuristischen Schätzung [Hint89], proportional zu $O(n^3)$, n = Anzahl Gewichte. In Bild 3.2 ist die Trainingszeit und das Scaling-Verhalten eines monolithischen NN mit Error Backpropagation dem einer einfachen modularen Architektur gegenübergestellt (zum Scaling-Verhalten siehe auch Abschnitt 3.2). In der modularen Architektur wird das Gesamtergebnis aus r Teilergebnissen zusammengesetzt. Für das Training des modularen Netzes ergibt sich im Idealfall der Geschwindigkeitsgewinn durch den Verbesserungsfaktor von mindestens r^2.

2. Dies wurde z.B. beim zur Zeit schnellsten Neurocomputer “Synapse” [Rama92] gemacht.
3. Backpropagation ist immer noch eines der meist verwendeten Netzwerkparadigmen [Akim93]



Anzahl zu trainierender Gewichte	Trainingszeit des monolithischen BP-NN mit n Gewichten	Trainingszeit des modularen Backpropagation-Netzes mit r Teilnetzen mit je $m = n/r$ Gewichten
n	$O(n^3) = O(m^3 r^3)$	$r\ O(m^3)$, ideal $m=n/r \Rightarrow$ **Verbesserungsfaktor** $\geq$ $\boldsymbol{r^2}$
$2n$	$O((2n)^3) = O(8m^3 r^3)$	$2r\ O(m^3)$ **Verbesserungsfaktor** $\geq$ $\boldsymbol{4r^2}$

Bild 3.2 *Das Scaling-Verhalten von monolithischen und modularen Backpropagation-Netzen.*

Zu 2.: Gesamtaufgabe zerfällt in unterschiedliche Teilaufgaben

Besteht die gesamte von einem NN zu lösende Aufgabe aus sehr unterschiedlichen Teilaufgaben, so reichen die Fähigkeiten eines einzelnen großen Netzes nicht aus, um gleichzeitig alle unterschiedlichen Teilaufgaben optimal schnell zu lösen. Wesentlich günstiger ist es, jede Teilaufgabe durch ein entsprechend der Charakteristika der Teilaufgabe ideal geeignetes Teilnetz zu bearbeiten. Von diesen speziell geeigneten Teilnetzen darf dann erwartet werden, daß sie wesentlich schneller lernen, als das “general purpose” Gesamtnetz.

Dies wird am Beispiel Funktions-Dekomposition illustriert: Die einfachste Funktion, die aus mehreren verschiedenen Bereichen besteht, ist eine stückweise lineare Funktion (siehe Bild 3.3). Diese Funktion kann durch ein nichtlineares NN mit einem hidden Layer beliebig genau approximiert werden [Horn91]. Sie kann aber auch durch zwei einzelne lineare Units, die exponentiell schnell lernen (Newton Verfahren), exakt repräsentiert werden.

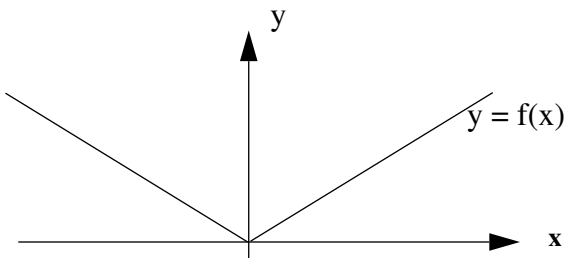


Bild 3.3 *Aufgrund der Charakteristika der zu lernenden Funktion kann das Gesamtnetz in Teilnetze zerlegt werden.*

3.2 Scaling

Das bereits oben angesprochene Scaling ist besonders für Lernalgorithmen ein Problem, für die nicht bekannt ist, wie sehr die Trainingszeit mit der Netzgröße steigt. Muß die Aufgabenstellung erweitert werden (z.B. kommen zusätzliche Daten bei einem Klassifizierungsproblem hinzu = incremental Learning), so muß im monolithischen Fall nicht nur das gesamte, nun vergrößerte Netz, neu trainiert werden (siehe auch Abschnitt “Software Engineering Aspekte” ab Seite 21), sondern es ist auch unklar wie lange dieses Retraining dauert. In der modularen Architektur werden für die neuen Teilaufgaben p zusätzliche Teilnetze hinzugefügt (siehe Bild 3.2). Damit bekommt man das Scaling-Verhalten des Gesamtnetzes in den Griff: Bei Verdopplung der Problemgröße wird die Anzahl der Teilnetze verdoppelt. Der Gesamtaufwand für die Trainingszeit steigt um den Faktor zwei, unabhängig davon welcher Lernalgorithmus für das Teilnetz verwendet wird.

In Bild 3.2 ist eine Schätzung des Scaling-Verhaltens für ein monolithisches und für ein modulares Backpropagation-Netz dargestellt: Beim monolithischen Backpropagation-Netz steigt im Gegensatz zum modularen Netz, der Gesamtaufwand für das Training bei Verdopplung der Problemgröße um den Faktor acht, beim modularen Netz hingegen nur um den Faktor zwei.

3.3 Crosstalk

In Abschnitt 3.1 “Trainingszeit” wurde für Backpropagation der Aufwand für das Training mit $O(n^3)$ angegeben. Der Grund für dieses schlechte Verhalten und für nicht optimale Lernergebnisse bei Backpropagation ist sogenanntes Crosstalk.

Mit Crosstalk [Jaco90] und Referenzen darin, bzw. Interference [Boer92], werden Effekte bei NN bezeichnet, die durch sich negativ beeinflussende Trainingsdaten ausgelöst werden. Jacobs unterscheidet räumliches (spatial) und zeitliches (temporal) Crosstalk:

3.3.1 Spatial Crosstalk

Spatial Crosstalk kann bei Backpropagation-Netzen auftreten, wenn zwei oder mehrere Units des Layers n mit einer Unit u_z des hidden Layers n-1 verbunden sind. In diesem Fall können sich die von den Units des Layers n zurückpropagierten Fehler für die Komponenten eines Trainingsvektors gegenseitig (teilweise) kompensieren, sodaß die Unit u_z des Layers n-1 nicht oder nur langsam lernt (Bild 3.4 a).

Bei einer modularen Architektur, bei der die Ergebnisse der einzelnen Module in einem Layer integriert werden (Bild 3.4 b), kann räumliches Crosstalk nicht auftreten.

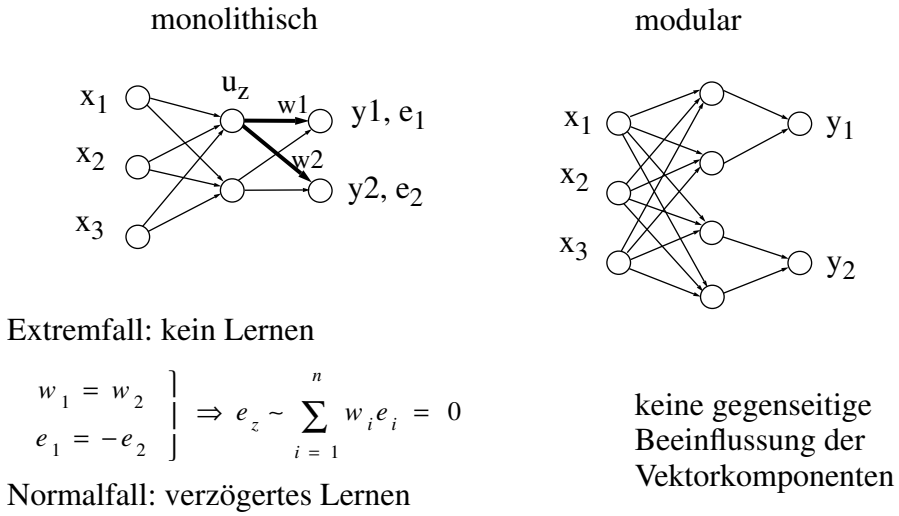


Bild 3.4 *Spatial Crosstalk kann nur bei monolithischen neuronalen Netzen auftreten*

3.3.2 Temporal Crosstalk

Temporal Crosstalk ist die gegenseitige negative Beeinflussung von verschiedenen Trainingsbeispielen. Temporal Crosstalk tritt besonders stark auf, wenn ein Netz verschiedene Funktionen gleichzeitig beherrschen soll (Bild 3.5 a). Wird in diesem Fall zuerst nur die eine Funktion trainiert und anschließend die andere, dann kommt es während des Trainings der zweiten Funktion zum teilweisen Vergessen der ersten Funktion[4]. Ein Ausweg besteht darin, beide Funktionen quasi gleichzeitig zu trainieren (random Pattern Selection). In diesem Fall ist zumindest die erforderliche Lernzeit wesentlich größer als die Summe der Lernzeiten zum Lernen jeweils nur einer Funktion. Häufig ist es jedoch nicht möglich, zwei oder mehrere Funktionen durch ein Netz gleich genau zu erlernen, wie es beim Erlernen jeder Funktion in einem eigenen Netz möglich wäre. Der Grund dafür ist, daß ein homogenes Netz erst zwischen verschiedenen hidden Units “umschalten” lernen muß, bis es überhaupt die zwei Funktionen ler-

4. Dies gilt für die meisten bekannten Lernalgorithmen. Spezialformen von Lernalgorithmen berücksichtigen durch Hinzufügen von neuen Units den Prozeß des sog. incremental Learnings.

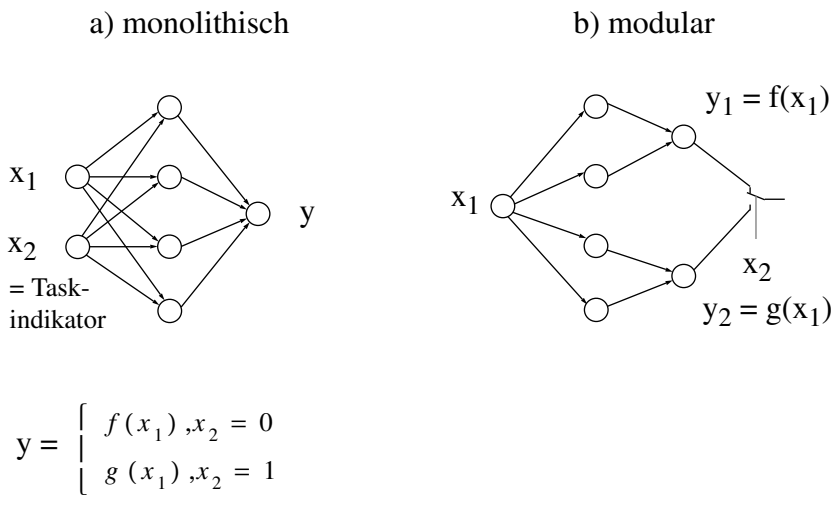


Bild 3.5 *Monolithisches und modulares Netz zur Approximation zweier Funktionen. Das monolithische NN muß zuerst zwischen den beiden Funktionen unterscheiden lernen.*

nen kann. Viele Anwender und Forscher haben sich mit diesem Problem beschäftigt, ohne eine wirklich brauchbare Lösung zu finden (z.B.: [Gutk90]). Die meisten Versuche haben zum Ziel die Trainingsreihenfolge zu optimieren, um einen möglichst großen Teil der Trainingsbeispiele erlernen zu können.

Wird statt einem monolithischen Netz ein modulares Netz verwendet, so kann temporal Crosstalk völlig ausgeschaltet werden (Bild 3.5 b). Das modulare Netz besteht aus zwei Teilnetzen, die jeweils genau eine der beiden Funktionen erlernen. Am Ausgang wird je nach gewünschter Funktion umgeschaltet.

Ein weniger extremer aber trotzdem wichtiger Fall von temporal Crosstalk ist die gegenseitige Beeinflussung von zu lernenden Input-Output-Relationen aus unterschiedlichen Bereichen einer Funktion. Dieser Fall ist interessant, wenn die Trainingsreihenfolge nicht beliebig wählbar ist. Dies ist bei Anwendungen von NN der Fall, bei denen online das Verhalten des Netzes adaptiert werden soll. Hier ist es erforderlich mit der z.B. vom Fertigungsprozeß vorgegebenen Reihenfolge der Trainingsdaten zurechtzukommen. Auch hier wird ein Netz benötigt, das nicht vergißt, was es früher gelernt hat.

3.4 Generalisierungsfähigkeit

Bei den meisten realen Aufgaben wird vom NN eine möglichst gute Fähigkeit verlangt von den im Training erlernten Beispielen auf die sogenannten “unseen cases” zu schließen. Drei Gründe sprechen dafür, daß das Generalisierungsverhalten von modularen NN besser ist, als das von monolithischen NN:

1. Lokalität: Bei der Lösung einer Aufgabe durch ein großes Netz mit sigmoiden Units entspricht das Generalisieren einer globalen Interpolation. Das bedeutet, daß aus allen mehr oder weniger gut erlernten Trainingsbeispielen (= Stützstellen) auf den Wert der

durch die Trainingsbeispiele repräsentierten Funktion an einer Stelle ungleich Stützstelle geschlossen wird. Ganz im Gegensatz dazu die Generalisierung bei modularen Netzen: Die bei modularen Netzen realisierte Zerlegung der Gesamtaufgabe in Teilaufgaben entspricht der Unterteilung einer zu lernenden Funktion in mehrere, eventuell überlappende Bereiche, wobei jeder Bereich von einem eigenen NN erlernt wird. Generalisierung erfolgt nun in diesem Fall lokal durch das Netz, das den lokalen Bereich beherrscht, für den es trainiert wurde. Dies entspricht einer stückweisen Interpolation (vergleiche Splines). Falls sich die Bereiche der Inputdaten, auf die die Einzelnetze trainiert wurden, überlappen, entspricht das Überlappen intuitiv den Bedingungen an die Glattheit in den Stoßstellen bei der stückweisen Interpolation. Aufgrund der Mechanismen, die beim Crosstalk angeführt wurden, kommt es bei lokaler Generalisierung nicht zur Beeinflussung durch unterschiedliche Inputbereiche.

Außerdem sind die Fähigkeiten zur Extrapolation von einigen neuronalen Netzen nicht zufriedenstellend (z.B. von Backpropagation [Schl94]). Dies verlangt, daß solche Netze ausschließlich innerhalb jener lokalen Bereiche des Inputraumes verwendet werden, auf die sie trainiert wurden.

2. Individuelle Netzwerkeigenschaften: Verschiedene Lernalgorithmen erlernen ein und dieselbe (Teil-) Funktion nicht nur unterschiedlich schnell, sondern auch unterschiedlich genau. Dies führt zu unterschiedlich guten Interpolationsverhalten. Zur Verdeutlichung sei eine Funktion angeführt, die in einem Teil linear, in einem anderen Teil nichtlinear ist (Bild 3.6). Natürlich kann auch diese Funktion durch ein einziges NN

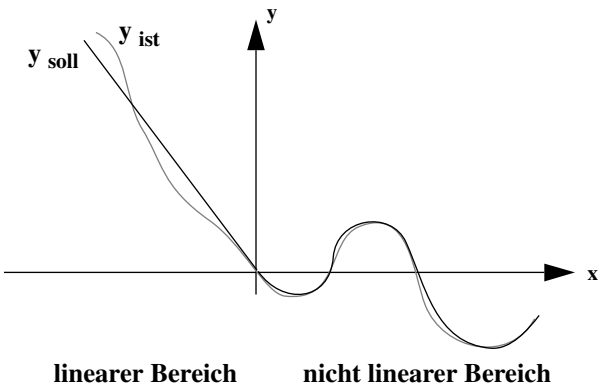


Bild 3.6 *Gegenseitige Beeinflussung unterschiedlicher Teile einer zu lernenden Funktion*

approximiert werden. Wenn den Teilbereichen jedoch je ein lineares und ein nichtlineares Netz zugeordnet werden, so ist eine bessere Generalisierungsfähigkeit des Netzes zu erwarten.

3. Integration von Vorwissen: Die Integration von Vorwissen über Aufgabe und Daten in ein NN ist wohl der wichtigste Grund für bessere Systemperformance. Vorwissen ist Wissen über die Input- und Outputgrößen des Systems. Darunter fällt Wissen über funktionale Abhängigkeiten und Korrelationen bzw. Hauptkomponenten, über die Verteilungen der Daten, über den Informationsgehalt von transformierten Größen und über Einschränkungen bezüglich Skalierungen. Aber auch Wissen über “die natürliche

Struktur einer Aufgabe", über noch nicht erfaßte Einflußgrößen, über die Bedeutung von Daten (Expertenwissen), sowie über vorhandene mathematische, statistische, logische oder physikalische Modelle stellen Vorwissen dar. Alle diese Informationen können (wie in Abschnitt 9.2.4 ab Seite 72 dargestellt werden wird) zur Modularisierung von neuronalen Systemen verwendet werden.

3.5 Robustheit und Fehlertoleranz

Fehlertoleranz bedeutet, auf Unvollkommenheiten im System (z.B. lokale Minima, ungünstige Parametrisierung), möglichst nicht mit Fehlern im Ergebnis zu reagieren. Robustheit bedeutet möglichst insensitiv gegenüber Störungen im Input (z.B. Rauschen, Meßfehler) zu reagieren. Robustheit hat bei NN mit Generalisierungsfähigkeit zu tun, wobei hier die Blickrichtung etwas anders ist. Grundidee der Verbesserung der Robustheit und der Fehlertoleranz ist, daß sich ein einzelnes Netz irren kann, nicht jedoch eine ganze Gruppe von Netzen, die parallel dieselbe Aufgabe bearbeiten. In der Entwicklung herkömmlicher Systeme entspricht diese Grundidee der Redundanz. Für Netze heißt dies im einfachsten Fall, daß nicht die Gesamtaufgabe in Teilaufgaben zerlegt wird, sondern, daß *r* gleiche aber unterschiedlich initialisierte NN parallel auf das Problem angesetzt werden (siehe Bild 3.7). Man erhält eine Menge von *r* Output-

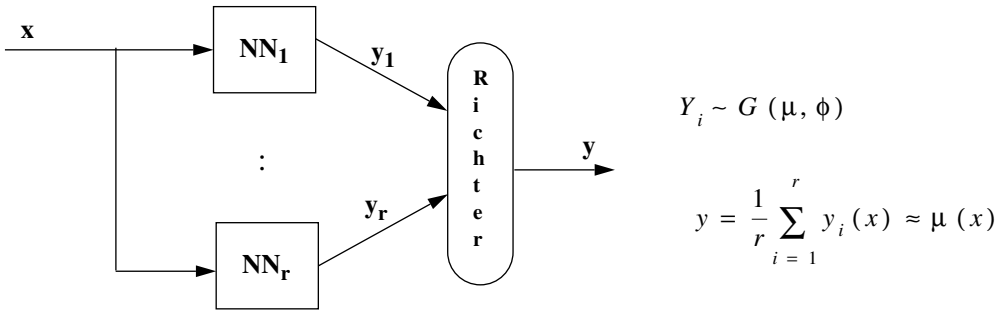


Bild 3.7 *Robustheit durch Redundanz*

werten. Sie stellen Ausprägungen einer stochastischen Größe Y_i dar, deren Erwartungswert μ dem richtigen Ergebnis näher kommt, als ein Ergebnis eines Einzelnetzes.

Eine weitere Verbesserung der Robustheit bzw. der Generalisierungsfähigkeit durch Redundanz gegenüber einem Einzelnetz ergibt sich, wenn Netze mit unterschiedlichen Lernalgorithmen gleichzeitig auf ein und dasselbe Problem angesetzt werden. Dies entspricht einer Diversifikation. In diesem Fall sollte statt dem arithmetischen Mittel, ein intelligenterer Richter verwendet werden, der z.B. die Gewichtung der Einzelergebnisse in Abhängigkeit von Störgrößen erlernt.

Die auf diese Weise erzielbaren Vorteile wurden bereits mehrfach demonstriert ([Linc90],[Batt94]). In Kapitel 6 werden verschiedene Formen von Redundanz und zugehörigen Richtern (Fusion bzw. Voting) ausführlich dargestellt.

3.6 Software Engineering Aspekte

Über den Erfolg bzw. Mißerfolg von NN entscheidet auch ein Punkt, der in der Forschung kaum berücksichtigt wird: Die Handhabbarkeit von NN in der Umsetzung in industrielle Projekte. Einerseits müssen dafür die traditionellen Software-Entwicklungs-Methodiken um die Belange der neuen Technologie der Informationsverarbeitung erweitert werden, andererseits gilt es, Bewährtes aus den konventionellen Methoden zu übernehmen. Traditioneller Systementwurf zielt auf eine modulare Definition des Systems ab [Somm92]. D.h. das Gesamtsystem wird in möglichst unabhängige Einzelteile zerlegt. Außerdem wird der Entwicklungsprozeß zeitlich gegliedert, z.B. in Analyse, Entwurf, Modularisierung, Moduldesign, Implementierung, Test, Integration, Systemtest, Pflege und Wartung. Je stärker die Sichtweise objektorientiert ist, desto stärker werden die Entwicklungsphasen zu einem evolutionären Prozeß des schrittweisen Verfeinerns des zu entwickelnden Systems [Rumb91].

In den traditionellen Software-Entwicklungs-Methodiken existieren eine Unzahl von Gründen für die Modularisierung des Systems. Teilweise sind diese Gründe auch auf neuronale Systeme anwendbar:

- Verifikation der Performance: Einzelnetze sind separat testbar und debugbar.
- Zergliederung in einfache Teilsysteme führt zu besser durchschaubarer Struktur des Systems und damit zu besserer Beherrschbarkeit und Interpretierbarkeit (Keep it simple and stupid, KISS).
- Wiederverwendbarkeit von Modulen: traditionelles Software Engineering zielt darauf ab, einmal entworfene Module möglichst in anderen Teilen wiederzuverwenden. Auf NN übertragen heißt dies einerseits Wiederverwendung von bereits trainierten bzw. vorstrukturierten Teilnetzen in anderen, ähnlichen Anwendungen, z.B. Featurededektoren, Transformations- und Analyseobjekte. Andererseits heißt Wiederverwendung auch, daß einmal programmierte neuronale Systembausteine auch für neue Paradigmen verwendbar sind.
- Erweiterbarkeit: Nachträgliches Erweitern der Aufgabenstellung (z.B. Hinzufügen von neuen Schriftarten in die zu erkennende Buchstabenmenge, oder nachträgliches Aufnehmen von Einflußgrößen auf die zu prognostizierende Zeitreihe, etc.) heißt bei Verwendung eines einzelnen großen monolithischen Netzes:

  Resize des Netzes mit allen Problemen des Scalings und der Parametereinstellung, sowie Retraining des Netzes vom Initialzustand aus. Deshalb ist es das Ziel, modulare neuronale Architekturen zu finden, bei denen die Erweiterung der Aufgabenstellung zum Hinzufügen eines zusätzlichen Teilnetzes führt. In solchen Architekturen ist im Idealfall nur das zusätzliche Teilnetz zu trainieren.
- Organisatorische Gründe: Meist ist die Systementwicklung ein inkrementeller Prozeß. Es wird nicht sofort ein System entworfen, das alle gewünschten Fähigkeiten aufweist. Vielmehr wird zunächst ein Teil der Aufgaben gelöst und in einer späteren Version kommen dann zusätzliche Leistungsmerkmale d.h. zusätzliche Module, hinzu.

3.7 Parametrisierung

Üblicherweise ist die Wahl der Netzparameter (Anzahl Layers, Units pro Layer, Lern- und Momentumsraten-Startwert, -Endwert und -Verlauf) eine nichttriviale Aufgabe. Um bei realen Anwendungen die erforderliche Performance zu erreichen, ist jedoch eine Optimierung der Parameter unerläßlich. Meist wird sie empirisch d.h. durch eine Reihe von Simulationsläufen gelöst. Schon die Wahl der optimalen Zahl an Hidden Units bei Backpropagation ist oft mit vielen Simulationsläufen verbunden. Ist bei einer “all in one”- Lösung die Netzgröße gefunden, so muß der optimale Lern- und Momentumsratenverlauf ermittelt werden. Dieser hängt unter anderem von der Netzgröße und von den Trainingsdaten ab.

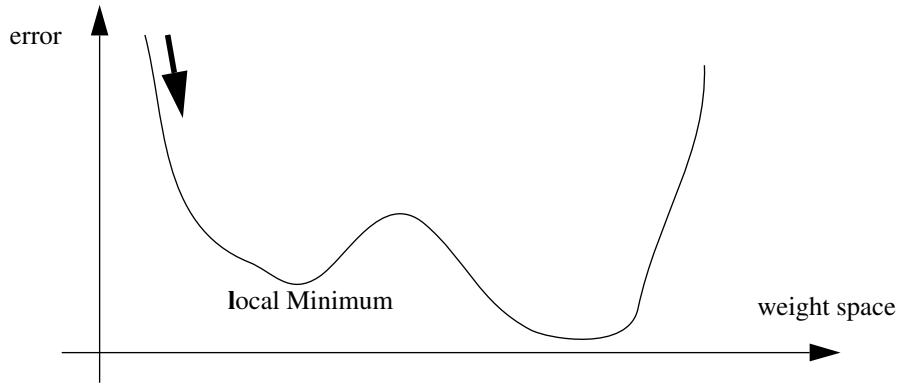


Bild 3.8 *Hyperfläche aufgespannt von den Gewichten eines neuronalen Netzes*

Bild 3.8 zeigt vereinfacht den Fehler eines neuronalen Netzes, der von den Gewichten des Netzes abhängt. Bei Gradient Descent Optimierungsverfahren muß die Lernschrittweite einen optimalen Verlauf und die Gewichte einen günstigen Startwert haben, damit das globale Minimum gefunden wird.

In modularen Architekturen können die Teilnetze für eine große Zahl von Anwendungen immer ungefähr gleich groß sein. Die Parametrisierung hängt dann nur noch von den Trainingsdaten ab, sodaß der Suchraum für die Parameter eingeschränkt werden kann. Dies ist nicht nur für manuelle Parameteroptimierung sondern auch bei algorithmischer Optimierung (z.B. genetischer Parameteroptimierung) sinnvoll.

3.8 Verteilbarkeit

Neuronale Netze sind implizit parallel. In realen Applikationen erfolgt jedoch meist eine Simulation des parallelen Lernprozesses durch einen sequentiellen Algorithmus. Der Grund liegt darin, daß der Kommunikationsaufwand innerhalb eines Netzes bei verteilter Lösung (eine Unit ist ein Processor) wesentlich größer ist als der Rechenaufwand. Bei vollverbundenen Netzen existieren für n lernende Units $O(n^2)$ Verbindungen. Nachdem bei modularen Ansätzen die Vernetzung zwischen den Modulen wesentlich geringer ist als innerhalb eines Moduls, kann die modulare Architektur direkt als Basis für die Rechnerarchitektur dienen. So ist es z.B. vorstellbar, unbenutzte Rechenzeit von untereinander vernetzten Workstations, durch Auslagerung von

in sich weitgehend abgeschlossenen Modulen zu nutzen. Spezialhardware für Parallelisierung entfällt.

3.9 Biologische Analogien

Obwohl es nicht das Ziel bei der Verwendung von NN ist, biologisch plausible Architekturen zu benutzen oder gar Reverse Engineering des Gehirns durchzuführen, kann ein Vergleich mit der Biologie hilfreiche Inspirationen und Ideen liefern.

Das Gehirn hat eine komplexe anatomische, histologische und funktionelle Struktur [Brai91]. Die Informationsverarbeitung und entsprechend die Vernetzung ist modular und hierarchisch aufgebaut. Das Gehirn ist keine unstrukturierte vollverbundene Menge von Neuronen, sondern besteht aus Arealen und Schichten (horizontale und vertikale Struktur) mit unterschiedlichen Aufgaben. Während die Neuronen eines Areals untereinander hochgradig vernetzt sind, ist die Verbindungsdichte zwischen Neuronen unterschiedlicher Areale wesentlich geringer. Die Modularität des Gehirn kann anhand einer Auswahl an Beispielen verdeutlicht werden:

Modularität im größten Maßstab: Das Zentralnervensystem des Menschen besteht in seiner groben Struktur aus Großhirn, Kleinhirn, Hirnstamm und Rückenmark [Ried93], wobei jeder Teil seine eigene Funktion erfüllt (funktionelle Modularisierung). Innerhalb der Hauptbestandteile existiert jeweils wieder eine feinere Modularisierung. Im Großhirn z.B. existiert im motorischen Cortex je Muskelgruppe ein eigenes Areal an Pyramidenzellen zur Ansteuerung der Muskelfasern. Dabei ist die Größe des Areals abhängig von der geforderten motorischen Auflösung (Bild 3.9).

wird eingeklebt

Bild 3.9 *Der motorische Homunculus*

Modularisierung im kleinen Maßstab: Biologische Neuronen benötigen Platz. Vollverbindung würde nicht nur eine enorme Zahl an Verbindungen mit sich bringen, sondern auch extrem große Verbindungslängen. So wird im Gehirn z.B. laterale Inhibition

zur Verstärkung von erkannten Features (Form, Ecken, Orientierung von Kanten, Größe, Bewegung, Ort von visuell wahrgenommenen Objekten u.s.w.) verwendet sowie mehrere unterschiedliche Repräsentationen von Features in unterschiedlichen Arealen [Jaco90]. Zur Reduktion von Verbindungslängen existieren eigene Retinotopic Maps je Gruppe von Features. Dadurch können mehrere Einzelfeatures trotz kurzer Verbindungslängen gleichzeitig verstärkt werden. Die unterschiedlichen Repräsentationen pro Feature ermöglichen die Koexistenz von räumlich nicht benachbarten Features trotz kurzer Verbindungen zu erkennen.

Modularisierung im kleinsten Maßstab: Auch in kleinstem Maßstab ist Modularisierung erkennbar: Synapsen sind teilweise auf den Dendritten in Gruppen angeordnet [Cher90].

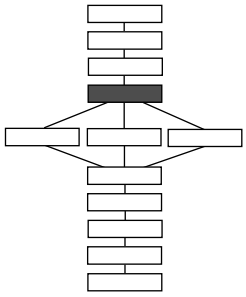
Hierarchische Struktur im Gehirn: Im primären visuellen Cortex existieren On-Off Zentren-Neuronen die auf helle bzw. dunkle Stellen in der Netzhaut reagieren. Mehrere On-Off Zentren Neuronen werden durch “Simple Cells” zusammengefaßt, sodaß helle oder dunkle Linien an festen Positionen erkannt werden können. Die “Complex Cells” wiederum verschalten jeweils mehrere auf dieselbe Orientierung reagierende “Simple Cells” um Linienstücke in einer gewissen Orientierung unabhängig von deren Position zu erkennen [Hube62].

3.10 Zusammenfassung und Ausblick

Im Bisherigen wurden eine Reihe von erwarteten Vorteilen von modularen neuronalen Systemen genannt. Es sei darauf hingewiesen, daß bei einer konkreten Anwendung nicht immer alle erwarteten Vorteile gleichzeitig zum Tragen kommen. Außerdem lassen sich die erwarteten Vorteile real nur dann erzielen, wenn es gelingt den Prozeß der Modularisierung methodisch und theoretisch in den Griff zu bekommen. D.h. es müssen

- modulare, vorgegebene und selbstentwickelnde, Netzarchitekturen und entsprechende Lernalgorithmen gefunden werden,
- die Schnittstellen zwischen den Modulen analysiert und definiert werden,
- konventionelle Software Engineering Methoden zur Modularisierung auf neuroinformatische Belange angepaßt werden
- und darin speziell zu den möglichen Architekturen Methoden zur Zerlegung der Gesamtaufgabe in Teilaufgaben entwickelt werden,
- Mechanismen erarbeitet werden, wie traditionelle Systeme in neuronale Systeme transformiert, bzw. um neuronale Komponenten erweitert werden können,
- sowie die Theorie anhand konkreter Beispiele evaluiert werden.

Der erste der genannten Punkte wird nun im anschließenden Kapitel behandelt.



4 Modulare Architekturen neuronaler Netze

Einige modulare Architekturen wurden bereits im Kapitel 3 implizit angesprochen. Hier erfolgt nun eine Systematisierung modularer neuronaler Systeme aus architekturorientierter Sicht. Es werden die Grundarchitekturen beschrieben, aus denen sich durch Kombination dieser Grundarchitekturen, beliebig komplexe Architekturen zusammensetzen lassen. Die Grundarchitekturen sind Ergebnis einer Analyse von existierenden Anwendungen neuronaler Netze. Außerdem werden Ideen aus der traditionellen Schaltungstechnik auf neuronale Netze übertragen und Mechanismen für das dynamische Wachsen von monolithischen Systemen für modulare Architekturen erweitert.

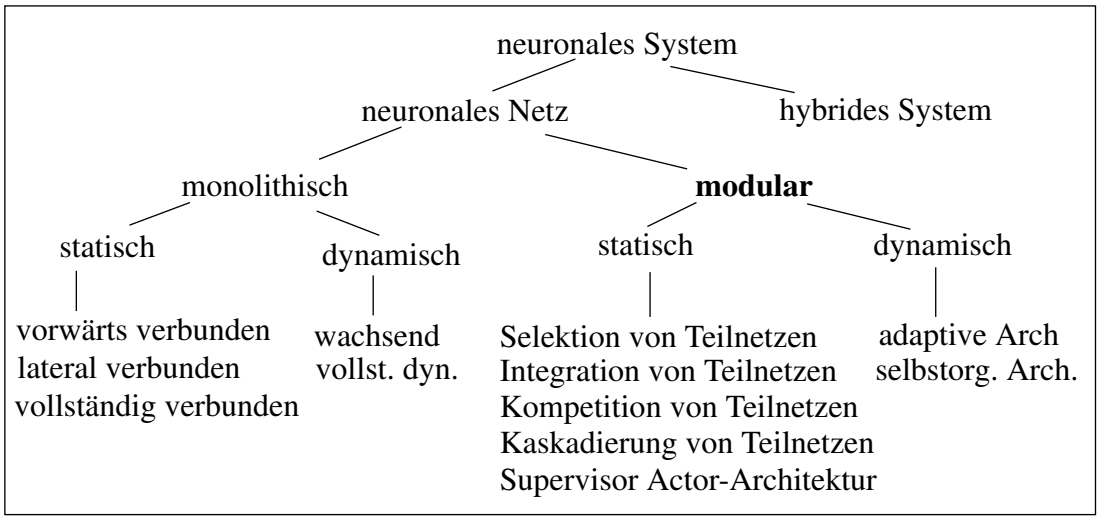


Bild 4.1 *Systematisierung neuronaler Systeme anhand ihrer Architektur*

In Bild 4.1 ist ein Überblick über diese Systematisierung gegeben. Außerdem ist dargestellt, wie sich modulare neuronale Architekturen in die Gesamtheit aller Architekturen einfügen:
Neuronale Netze lassen sich in rein neuronale Systeme und in hybride Systeme einteilen (siehe Kapitel 2 “Begriffsbestimmungen”). In dieser Arbeit werden nur rein neuronale Systeme, die neuronalen Netze betrachtet. Hybride Systeme werden nur kurz, der Vollständigkeit halber, angeschnitten.

Neuronale Netze können entsprechend ihrer Architektur in monolithische neuronale Netze (intern nicht strukturiert) und in modulare Netze eingeteilt werden. Sowohl die monolithischen Netze, wie auch die modularen Netze lassen sich anhand der Veränderbarkeit ihrer Architektur in Netze mit statischer (fixer) Architektur und Netze mit dynamischer (veränderlicher) Architektur unterteilen. Bekannte Beispiele für monolithische Netze mit statischer Architektur sind

- Multi Layer Backpropagation-Netze [Rume86],
- Topology Preserving-Maps (Kohohnen-Netz) [Koho89],
- Hopfield-Netz [Hopf88] und
- Elman Jordan-Netze [Ulbr92].

Monolithische Netze mit dynamischer Architektur sind selbständig wachsende Netze wie

- Growing Radial Basis Function Networks [Ghos90],
- Node Splitting Feed Forward Networks [Wynn92],
- Growing Cell Structures [Frit91] und
- Sequential Input Space Partitioning [Shad93].

In diesem Kapitel werden die prinzipiell möglichen Architekturen modularer neuronaler Netze erläutert. Auch sie lassen sich in statische und dynamische Architekturen unterteilen. Bei dynamischen Architekturen ist, im Unterschied zu den statischen, die Anzahl der Teilnetze und/ oder die Anzahl der Units und Verbindungen variabel.

4.1 Systeme mit statischer Architektur

Statische (fixe) Architektur heißt, daß die Anzahl einzelner Teilnetze (Module) und deren Verbindungen durch den Entwickler vorgegeben werden. Um auf die möglichen Architekturen zu kommen, können die Prinzipien, die innerhalb von Netzen wirksam werden, auf Module übertragen werden. Verwendbar sind die Mechanismen:

- Kompetition,
- Integration,
- Kaskadierung und
- (kollektive) Erregung, sowie
- Hierarchiebildung.

Zusätzlich können auch noch Architekturen von traditionellen Systemen bei neuronalen Systemen verwendet werden:

- Serienschaltung,
- Selektion (1 aus r, q < r aus r),
- Synthese (Parallelschaltung) und
- Master Slave-Architekturen (Supervisor Actor).

4.1.1 Selektion eines Teilnetzes

In dieser Architektur existieren r Teilnetze. Für jeden am Eingang des Systems ankommenden Inputvektor $\underline{x}$ wird genau ein Teilnetz ausgewählt. Dieses Teilnetz liefert das Ergebnis $\underline{y}$ für das gesamte System. Alle Teilnetze haben die gleichen Inputgrößen.

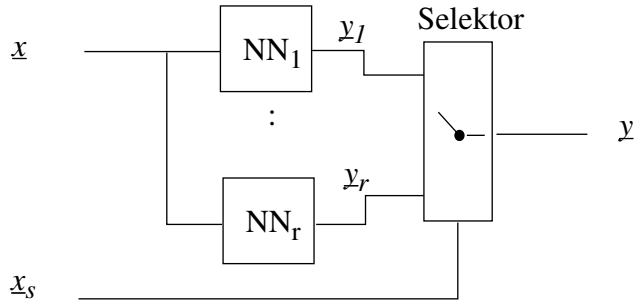


Bild 4.2 *Selektion eines Teilnetzes*

Die Aufgabe, für jeden am Eingang des Systems ankommenden Inputvektor das entsprechende Teilnetz auszuwählen, löst ein Selektor. Die Aufgabe des Selektors entspricht einer Klassifikation des jeweiligen Inputvektors in eine von r Klassen. Dadurch wird der gesamte Eingangsraum $I \subseteq \Re^m$ in r disjunkte Teilräume $J_k \subset I \subseteq \Re^m$ partitioniert, wobei jeder Teilraum genau einem Teilnetz zugeordnet wird. Die Idee dahinter ist, daß für jeden Teilraum ein Teilnetz verwendet wird, das für die jeweilige Teilaufgabe besonders gut geeignet ist, d.h. daß eine lokal optimale Behandlung des Teilraumes erfolgt. Von besonderer Bedeutung ist dies, falls in jedem Teilraum ein unterschiedliches (adaptives) Pre- oder Postprocessing erforderlich ist.

Die Selektionskriterien $\underline{x}_s$ sind dabei für die Funktion dieser Architektur von großer Bedeutung. Die Selektion kann manuell anhand von Vorwissen eingestellt werden, oder automatisch durch einen Lernalgorithmus anhand der Selektionskriterien erlernt werden (siehe dazu Kapitel 7).

Beim Training wie beim Recall wird bei dieser Architektur zunächst jener Teilraum bestimmt, dem der Datenvektor angehört und dann das dem Teilraum zugeordnete Teilnetz selektiert. Nur dieses Teilnetz wird trainiert. Beim Recall liefert dieses Teilnetz das Ergebnis.

Diese Architektur wird eingesetzt wenn in einem großen Datenraum viele von der Art identische Teilaufgaben zu erfüllen sind. Die Architektur wird nicht eingesetzt, wenn

die Dimensionalität des Inputraumes im Verhältnis zur Anzahl an zur Verfügung stehenden Trainingsdaten sehr hoch ist. In solch einem Fall stehen nicht genügend Trainingsbeispiele für das Training der Teilnetze zur Verfügung.

Der Vorteil dieser Architektur liegt besonders in der leichten Erweiterbarkeit und der geringen Größe der Teilnetze, aus der niedrige Trainingszeiten und gute Skalierbarkeit resultieren (siehe Abschnitt 11.2).

4.1.2 Integration von parallelen Teilnetzen

In dieser Architektur existieren r Teilnetze die parallel (gleichzeitig) betrieben werden (Bild 4.3). Die r Teilergebnisse der Netze werden in einer integrierenden Stufe zu einem Gesamtergebnis zusammengefügt. Jedes einzelne Netz trägt zum Gesamtergebnis bei. Aus diesem Grund müssen auch immer alle Teilnetze ausgewertet werden. In der Trainingsphase wird jedes Teilnetz auf seine Teilaufgabe trainiert. Dies kann sowohl parallel (gleichzeitig) wie auch sequentiell erfogen.

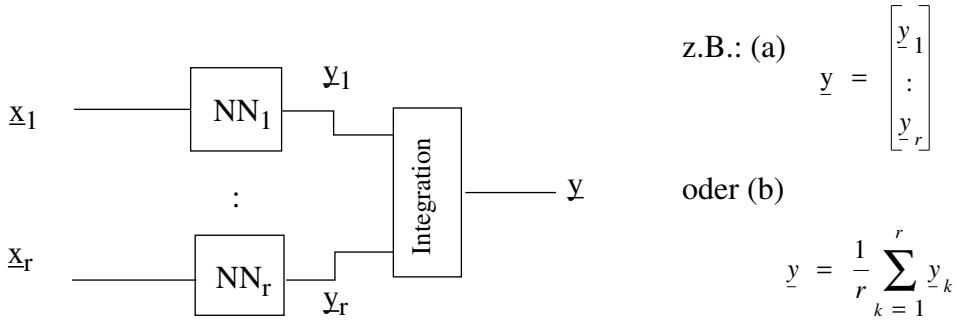


Bild 4.3 *Integration von Teilergebnissen zu einem Gesamtergebnis*

Bei integrativen Architekturen kann zwischen Aufgabenteilung und Redundanz unterschieden werden:

a) Aufgabenteilung

Bei Aufgabenteilung behandelt jedes Teilnetz seine eigene, unabhängige Aufgabe (vergleiche [Mare90], Kapitel “Hybrid and Complex Networks”). Die Integration der Teilergebnisse zum Gesamtergebnis besteht in diesem Fall aus dem Zusammensetzen (Merge) der Teilergebnisse (Bild 4.3 a). Die Teilnetze liefern unterschiedliche Ausgangsgrößen. Sie arbeiten dabei auf gleichen oder auf unterschiedlichen Inputgrößen. Aufgabenteilung führt zur Partitionierung des Ausgangsraumes $O \subseteq \Re^m$ in niedrigerdimensionale Teilräume $O_k \subset \Re^{m_k}$, $0 < m_k < m$, $\sum m_k = m$. Falls die Teilnetze auf unterschiedlichen Eingangsgrößen arbeiten, erlernt jedes Teilnetz einen Teil einer Abbildung der Inputdaten auf die Outputdaten. In diesem Fall wird auch der Eingangsraum $I \subseteq \Re^n$ partitioniert in niedrigerdimensionale Teilräume $J_k \subset \Re^{n_k}$, $0 < n_k < n$, $\sum n_k = n$.

Ähnlich dem Selektor von 4.1.1 kann auch hier bei einer “horizontalen” Aufteilung des Eingangsraumes eine “splittende Stufe” automatisch den Eingangsraum partitionieren.

Der Vorteil der Aufteilung von parallel zu erfüllenden Aufgaben auf unterschiedliche Teilnetze liegt in der kleineren Teilnetzgröße (siehe Kapitel 3.1). Voraussetzung für eine solche Aufteilung von parallelen Teilaufgaben auf parallele Teilnetze ist die Unabhängigkeit der Teilaufgaben. Diese Unabhängikeit zeigt im Fall von global optimierenden Netzen, (z.B. Backpropagation) einen zweiten Vorteil: das Netz muß nicht erst die Unabhängigkeit erkennen und erlernen. Dies führt zu besserer Generalisierungsfähigkeit.

b) Redundanz, Diversität

Bei dieser Art von integrativer Parllelität erfüllen alle r Teilnetze die gleiche Aufgabe (Bild 4.3 b). Zweck dieser Redundanz ist die Verringerung der Fehlerrate und die Erhöhung der Robustheit. Voraussetzungen zur Verringerung der Fehlerrate und Erhöhung der Robustheit durch Redundanz sowie verschiedene Variantien zur Integration von redundanten Ergebnissen werden in Kapitel 6 diskutiert.

4.1.3 Serienschaltung von Teilnetzen

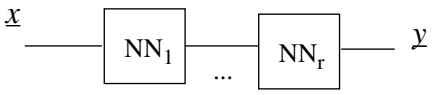


Bild 4.4 *Serienschaltung von Teilnetzen*

Diese Art der Modularisierung von Netzen zeichnet sich dadurch aus, daß mehrere Teilnetze in Serie hintereinander geschaltet sind (Bild 4.4). Die Ausgangsdaten von Netz k sind die Eingangsdaten von Netz k + 1. Trainiert wird meist jedes Netz für sich auf eine eigene Aufgabe. In diesem Fall werden die Teilnetze erst nach erfolgtem Training aller Teilnetze zusammengeschaltet. Es ist jedoch auch möglich die Teilnetze aufeinander aufbauend zu trainieren, d.h. Netz NN_{k+1} wird unter Verwendung der Netze NN_1 bis NN_k trainiert.

Der spezielle Vorteil dieser Archtitektur ist, daß die Einzelnetze nach ihrer Zusammenschaltung noch gemeinsam trainiert werden können, wobei dann der Fehler von Netz k zu Netz k - 1 zurückpropagiert wird. Das gesamte System ist klar strukturiert, jedes Modul läßt sich separat testen, wodurch eventuelle Performancemängel leichter behoben werden können.

Voraussetzung für die Verwendung der Serienschaltung ist, daß die gesamte Aufgabe in aufeinader aufbauende Teilaufgaben zerlegt werden kann. Die einfachste Serienschaltung ergibt sich durch die aufeinanderfolgenden Stufen: Vorverarbeitung, Featureextraktion, Processing (die eigentliche Aufgabe) und Nachverarbeitung. Sie ist im Prinzip bei jedem neuronalen System anzutreffen. Beispiele für eine modulare

neuro-nale Serienschaltung sind eine neuronale Featureextraktion gefolgt vom eigentlichen NN (siehe Abschnitt 11.4), bzw. eine neuronale Quantisierungsstufe vor dem eigentlichen NN [Hryc92].

4.1.4 Kaskadierung von Teilnetzen

Diese Grundschaltung (siehe Bild 4.5) ist der reinen Serienschaltung ähnlich. Bei der jeweils nächstfolgenden Stufe werden jedoch zusätzliche Inputdaten berücksichtigt. Das Training einer solchen Kaskade von NN muß zeitlich gestaffelt ablaufen, d.h. daß das Training der Stufe k der Kaskade ein trainiertes Netz der Stufe k -1 voraussetzt. Jedes Teilnetz wird dabei auf die Solloutputdaten $\underline{y}_{soll}$ trainiert. Zur Erziehlung optimaler Ergebnisse können wie bei der Serienschaltung die Teilnetze 1 bis k nach dem Training des Netzes k+1 nocheinmal gemeinsam trainiert werden.

Anstatt die Ausgangsdaten von Netz k für das Netz k +1 als Eingangsdaten zu verwenden, kann auch der Fehler von Netz k (gemeinsam mit zusätzlichen Daten) als Eingangsdaten für Netz k+1 verwendet werden. In diesem Fall lernt das Netz k+1 den verbleibenden Fehler der Netze 1 bis k anhand zusätzlicher Einflußgrößen zu korrigieren. Die Auswirkung von Netz k+1 auf das Gesamtergebnis ist somit geringer als die von Netz k.

Cascade Correlation [Fahl90] und [Frie91] ist die erste Architektur, die Kaskaden von Teilnetzen verwendet. Im Unterschied zu der hier angeführten verallgemeinerten Sichtweise, werden bei Cascade Correlation, statt Teilnetzen nur einzelne Units hinzugefügt. Alle Units erhalten, neben den Outputs aller “davor” liegenden Units, denselben Input. Der Ausgang $\underline{y}$ ist jeweils die gewichtete Summe der Ausgänge aller Units.

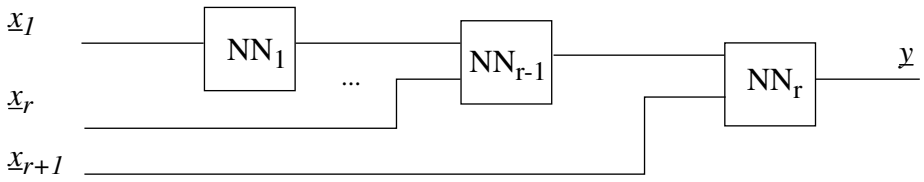


Bild 4.5 *Kaskanden von neuronalen Netzen*

Diese Architektur wird vor allem dann eingesetzt, wenn abgeschlossene Teilaufgaben von zusätzlichen Daten beeinflußt werden. Der besondere Vorteil dieser Architektur liegt in der inkrementellen Erweiterbarkeit zur schrittweisen Steigerung der Performance. Außerdem kann der Stärke bzw. Zuverlässigkeit von Features Rechnung getragen werden: weniger starke Features werden in höheren Stufen berücksichtigt.

4.1.5 Supervisor Actor-Architektur

Bei dieser Struktur fungiert ein NN als Steuereinheit (Supervisor) für ein anderes NN (Actor) (Bild 4.6). Im Prinzip kann jeder Parameter des Actor-Netzes durch das Supervisor-Netz gesteuert werden. Meist passiert dies in Abhängigkeit von der Performance des Actor-Netzes.

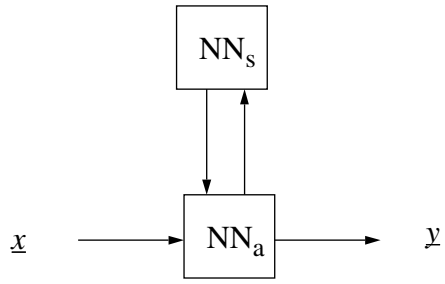


Bild 4.6 *Supervisor Actor-Architektur*

Der **Vorteil** der Supervisor Actor-Architektur ist der unterschiedliche Horizont der beiden Netze bezüglich Zeit und Abstraktion. Dadurch werden Meta-Architekturen möglich, in denen ein Parameter des Supervisors mehrere Parameter des Actors, bzw. die Parameter von mehreren Aktoren, steuert. Außerdem kann der Supervisor auch eine Performanceabschätzung des (Teil-)Systems liefern.

Eine Supervisor Actor-Architektur findet z.B. im Bereich der Systemidentifikation Anwendung [Mill90]. Dabei wird das Verhalten einer Maschine oder sonst einer Strecke durch ein zu dieser Strecke parallel geschaltetes NN erlernt. Eine Kopie dieses Netzes, wird in der inversen Betriebsart (Ausgänge werden auf Eingänge abgebildet) als inverses Modell zur Steuerung der Maschine verwendet. Das Supervisor-Netz stellt die Gewichte des Actor-Netzes ein.

Weiters sind einige Formen des Reinforcement Learnings Repräsentanten von Supervisor Actor-Architekturen. Ein Beispiel dafür sind adaptiv Critics [Bart83]. Bei dieser Architektur exisitert ein Actor-Netz, das die eigentliche Aufgabe lösen soll und ein Critic-Netz, das das Verhalten des Actor-Netzes bewertet. Das Actor-Netz lernt die gewünschte Abbildung des Eingangs- auf den Ausgangsraum durch stochastische Suche unter dem Einfluß eines Feedbacksignals. Das Critic-Netz ist das Supervisor-Netzwerk. Es erlernt mit Fortdauer des Trainingsprozesses anhand des Endergebnisses bzw. des Verlaufes der “Aktionen” des Actor-Netzwerks, die “Aktionen” des Actor-Netzwerks zu beurteilen. Auf diese Weise erhält das Actor-Netz mit der Zeit immer bessere Fehlersignale, anhand denen es lernt.

4.1.6 Hierarchische Architekturen

Hierarchische Architekturen entstehen, wenn Module selbst wieder modularisiert werden. Im Prinzip kann jedes einzelne Netz, das Teil einer der bisher genannten Architekturen ist, selbst wieder durch eine beliebige Architektur ersetzt werden. In realen Anwendungen findet man jedoch meist die rekursive Anwendung der gleichen Archi-

tektur [Jord92],[Mavr92],[Bisc93]. Im Falle der rekursiven Selektion (Bild 4.7) von Teilnetzen erfolgt eine rekursive Aufteilung des Eingangsraumes oder der Komponenten der Eingangsvektoren auf Einzelnetze.
Ein Beispiel für eine hierarchische Architektur bestehend aus unterschiedlichen Basisarchitekturen ist die Parallelschaltung (Redundanz) von n selektiven Architekturen.

Bemerkung: Eine hierarchische Datenvorverarbeitung, z.B. durch eine konventionelle Bildpyramide, gefolgt von einem neuronalen Netz, das auf Features aller Ebenen [Paar92] oder jeweils auf Features einer Ebene angesetzt wird [Bisc92], ist zwar ein modulares System, aber kein hierarchisches Netz. Eine neuronal realisierte Pyramide [Bisc93] stellt hingegen ein hierarchisches Netz dar.

4.2 Systeme mit dynamischer Architektur

Dynamische Architekturen sind Architekturen, bei denen sich modulare Netzstrukturen automatisch entwickeln. Entsprechend der Definition von Large Scale- und Small Scale-Modularität (siehe Abschnitt 2.4) sind dynamische Veränderungen der Architektur sowohl auf Teilnetzebene wie auch auf Unitebene möglich.

4.2.1 Systeme mit adaptiver Architektur

Bei Systemen mit adaptiver Architektur wird eine initiale Architektur während des Trainings verändert. Die initiale Architektur wird anhand von Vorwissen “manuell” erstellt. Der Prozeß der Optimierung der Architektur ist dann ein Hinzufügen bzw. Entfernen von Teilnetzen, Units und Verbindungen abhängig von vordefinierten Kriterien.

Man unterscheidet konstruktive Verfahren und destruktive Verfahren. Bei konstruktiven Verfahren werden solange Teilnetze, Units und Verbindungen hinzugefügt, bis es zu keiner weiteren Performanceverbesserung mehr kommt. Bei destruktiven Verfahren werden ausgehend von einer initialen Architektur (meist vollverbunden monolithischen Netzen) nicht benötigte Teile entfernt. Die destruktiven Verfahren haben das Ziel, die Generalisierungsfähigkeit der Netze zu verbessern und nicht das Ziel modulare Strukturen zu erzeugen. Die resultierende Architekturen entsprechen jedoch modularen Architekturen im Sinne der in Kapitel 2.4 gegebenen Definition von Modularität.

Ein Beispiel für eine konstruktive, dynamische Large Scale-Architektur ist eine Adaptive Hierarchie von 1 aus r selektiven Architekturen (Bild 4.7). Im Prinzip kann jedoch jede Basisachitektur aus Kapitel 4.1 “Systeme mit statischer Architektur” auf ähnliche Weise adaptiert werden. Die Architektur aus Bild 4.7 könnte folgendermaßen adaptiv entwickelt werden: Zunächst wird nur ein Selektor und r einzelne Teilnetze verwendet. Sollten im Laufe des Trainingsprozesses einzelne Teilnetze eine auf ihrem Datenraum konstant schlechtere Performance am Testset aufweisen als die durchschnittliche Performance der Teilnetze, so wird dieses einzelne Teilnetz durch zwei oder mehrere Teilnetze [Iwat92] oder ein Modul aus Selektor und n “Sub-”Teilnetzen ersetzt. Dadurch kommt es zu einer rekursiven Aufteilung des Inputraumes solange noch eine Perfor-

manceverbesserung erreicht werden kann. Von der Wurzel ausgehend wächst mit der Zeit, gesteuert von der Performance, ein in seinen Ästen unterschiedlich langer Baum.

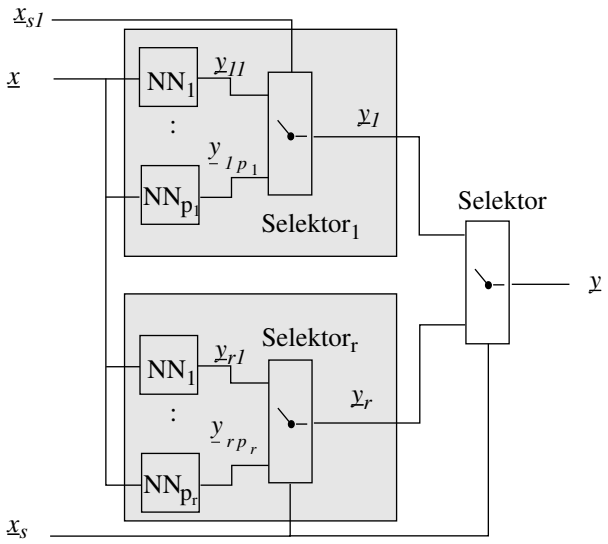


Bild 4.7 *Hierarchie von 1 aus r selektiven Architekturen*

Konkrete Beispiele für destruktive Verfahren zur Entwicklung modularer Small-Scale-Architetkuren sind

- Pruning (Input-, Unit-, Weightpruning) [Herg92],
- Optimal Brain Damage [Cun90] bzw.
- Penalties für die Begrenzung der Netzwerkkomplexität [Kend93].

Bei Pruning werden aus einem überdimensionierten Netz, zum oder vor dem Zeitpunkt der optimalen Performance (vergleiche Bild 5.2 “Typischer Verlauf des Fehlers während des Trainings eines Netzes”) jene Verbindungen und Units entfernt, die sehr wenig zum Gesamtergebnis beitragen.

4.2.2 Systeme mit selbstorganisierender Architektur

Bei Systemen mit selbstorganisierender Architektur werden Large- und Small Scale-Architekturen anhand von Bildungsgesetzen entwickelt. Die Bildungsgesetze beschreiben, wie aus einfachen Bausteinen durch Zusammensetzen modulare Systeme entstehen. Der Unterschied zu den adptiven Architekturen ist der, daß nun nicht konkrete vorgegebene Architekturen solange verändert werden, bis die gewünschte Performance erreicht ist, sondern die Architekturen vollständig automatisch konstruiert werden.

Das Gesamtsystem ist zweigeteilt in das eigentliche **Zielsystem**, das die gestellte Aufgabe lösen soll und in ein Metasystem, das die Architektur des Zielsystems entwickelt. Die Architektur des Zielsystems wird durch einen zyklischen Prozeß des Reinforcement Learnings optimiert. Dieser beginnt mit der Konstruktion neuer Architekturen, gefolgt vom Training der gefundenen Systeme. Die Performance der trainierten

Systeme wird dann evaluiert, um schließlich anhand der Performance dieser Systeme entweder den Prozeß der Optimierung zu beenden, oder durch die Erzeugung neuer Systeme, unter Verwendung der vorhandenen den Prozeß fortzusetzen (Bild 4.8).

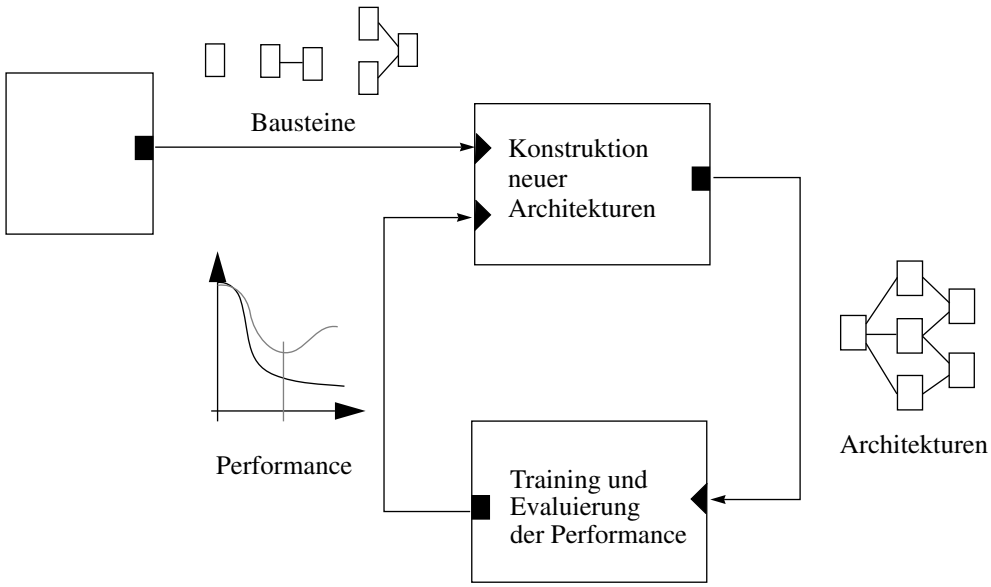


Bild 4.8 *Zyklischer Prozeß der Optimierung der Systemarchitektur*

Wesentliche Probleme dieser Vorgangsweise sind

- die Repräsentation der Bausteine des eigentlichen Systems,
- die Wahl der Performancekriterien für gefundene Architekturen, und
- die enormen Anforderungen an die benötigte Rechenzeit.

Die Wahl der Repräsentation von Netzwerkarchitekturen ist deshalb so entscheidend, weil sie den Raum aller möglichen Architekturen definiert. In einem zu kleinen Raum ist die optimale Architektur eventuell gar nicht enthalten, in einem zu großen Raum exisitiert zwar die optimale Version, sie kann jedoch nicht gefunden werden [Boer92], [Harp89]. Die Definition von konkreten Maßen für die Performance von Netzwerken (Architekturen) hat ebenfalls ganz entscheidenden Einfluß darauf, ob und wie schnell eine brauchbare Lösung durch den Prozeß des Reinforcement Learnings gefunden werden kann. Und schließlich wird auch bei günstiger Wahl von Repräsentation und Performancekriterium die tatsächlich benötigte Rechenzeit für “Real World Problems” ganz enorm sein. Es scheint derzeit überhaupt nur realistisch, an die Evolution von Architekturen zu denken, weil diese Evolution ein offline Prozeß sein kann (nur die gefundene Architektur muß Echtzeitanforderungen genügen) und weil aufgrund des entfallenden Kommunikationsaufwandes auf n Rechnern parallel n Architekturvarianten evaluiert werden können.

Das in Bild 4.8 dargestellte Modul zur Erzeugung von Architekturen enthält häufig einen genetischen Algorithmus [Holl75]. Dabei werden ganze **Populationen** von sogenannten Individuen erzeugt. Ein **Individuum** ist das Abbild genau einer möglichen Architektur für das Zielsystem. Es wird durch einen **Code** (binär usw.) repräsentiert. Die Population auf der der genetische Algorithmus arbeitet, ist also eine Menge von solchen Codes. Der genetische Algorithmus erzeugt unter Verwendung von **genetischen Operatoren** wie **Mutation**, **Crossover** und **Selektion** [Harp89] aus einer initialen Population, bzw. aus einer gerade evaluierten Population neue Individuen. Der Vorteil bei der Verwendung ganzer Populationen liegt in der schnelleren Konvergenz des Metasystems.

Für die genetisch gesteuerte Evolution modularer Systeme gibt es in der Literatur bereits mehrere Ansätze [Harp89], [Boer92], [Happ94] und Zitate darin. Boers konstruiert beispielsweise ein modulares Backpropagation-Netzwerk. Die Chromosomen beschreiben bei Boers nicht direkt die Verbindungsstruktur (d.h. einen Graph, siehe Kapitel 2.1 ab Seite 5) des zu evaluierenden Individuums (Netzes), sondern die oben angesprochenen Regeln zur Bildung eines neuronalen Netzes. Die Regeln sind Produktionsregeln eines L-Systems (einer speziellen Klasse von Fraktalen), das den Bildungsmechanismus für die Verbindungsstruktur des Netzes beschreibt. Der Vorteil im Umweg über das L-System liegt darin, daß die Chromosomen nicht mit zunehmender Netzwerkgröße exponentiell wachsen (wie das der Fall wäre, wenn jede Verbindung zwischen den Units codiert werden müßte) und in der biologischen Plausibilität.

Happel [Happ94] unterschiedet vier Ebenen der Adaption: Evolution, Ontogenese, Lernen und neuronale Aktivierung. Evolution als die höchste Ebene betrifft die Anpassung auf “globale” Bedingungen des “Environments”. Ontogenese ist die Entwicklung eines einzelnen Individuums während der Lebensdauer des Individuums. Lernen, die dritte Ebene der Adaption, sieht Happel als eine Feinabstimmung der phylo- und ontogenetisch entwickelten neuronalen Architekturen. Die neuronale Aktivierung schließlich ist die Reaktion eines konkreten Netzes auf einen einzelnen Stimulus. Happel konstruiert konkrete, modulare, neuronale Architkturen durch gentische Optimierung aus “Architektur Building Blocks”. Diese Building Blocks bestehen ausschließlich aus Instanzen eines speziell für modulare Strukturen geeigneten Netzes, dem Categorizing and Learning Module (CALM [Murr92] realisiert einen adaptiven Clustering Algorithmus durch eine Form des kompetitiven Lernens).

Aufgrund des bereits erwähnten Rechenzeitproblems wurden die genannten Ansätze jedoch nur auf sehr “kleinen” und unrealistichen Anwendungen erprobt.

4.3 Hybride Neurosysteme

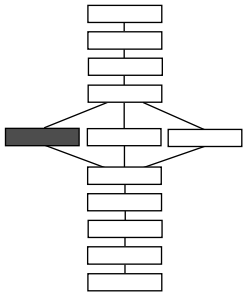
Wie in Abschnitt 2.7 “Hybrides System” definiert wurde, besteht ein hybrides System aus Komponenten unterschiedlicher Technologie. Gerade in jüngster Zeit ist zu beobachten, daß der neuronale Anteil an Gesamtsystemen etwas in den Hintergrund gedrängt wird. NN werden verstärkt mit Elementen anderer Technologien kombiniert bzw. als ein Funktionsbaustein in traditionelle Systeme integriert. Der Grund für die Kombination verschiedener Technologien liegt in der Vereinigung der Vorteile, die die jeweiligen Technologien mit sich bringen. Das Ziel ist einerseits die nichtneuronalen

Komponenten lern- und generalisierungsfähig, sowie robuster gegen Störungen der Eingangsdaten zu machen. Andererseits kann aus der Struktur von nichtneuronalen Komponenten Vorwissen für die Strukturierung von NN nutzbar gemacht werden und damit die Gesamtperformance gesteigert, sowie das Verhalten der Netze erklärbarer gemacht werden. (Siehe auch “Dekomposition unter Verwendung vorhandener Lösungen” ab Seite 72).

Beispiele für hybride Systeme sind hybride Modelle [Cael93], Konventionelle Mustererkennung kombiniert mit NN[Bish92], regelbasierte Systeme und NN[Holl93], [Gutk90], sowie Fuzzy Neural Systems [Taka90].

Ein Beispiel für ein hybrides System im Zusammenhang mit modularen Netzen im Bereich der Bildverarbeitung ist [Mora94]. Darin wird ein semantisches Netz verwendet, um Wissen über den Aufbau von physikalischen, in einem Bild zu erkennenden, Objekten explizit zu modellieren. Jeder Knoten entspricht einer Klasse von zu erkennenden Objekten (z.B. Würfel, Schraube). Die Kanten des Graphs modellieren Part of Relationen (z.B. Parallelogramm ist Teil eines Würfels) und Vorwissen über die Szene (Nachbarschaftsrelationen). Jedem Knoten (Klasse) des semantischen Netzes wird ein Backpropagation-Netz zugeordnet, das für Attribute wie Position und Größe, die Wahrscheinlichkeit abschätzen soll, ob im gegebenen Bild ein Objekt mit diesen Atributen vorhanden ist. Die Attribute werden über einen Steuermechanismus (ähnlich A*) anhand des expliziten Modells unter Verwendung der Wahrscheinlichkeitsaussagen der Netze optimiert. Der spezielle Vorteil dieser Art der Kombinatinon unterschiedlicher Techologien ist die unterschiedliche Modellierung (explizit fix und implizit adaptiv) auf unterschiedlichen Ebenen der Abstraktion.

In dieser Arbeit werden hybride Systeme nicht näher untersucht.



5 Online- Messung der Netzwerkperformance

In mehreren neuronalen Architekturen werden Kriterien und Verfahren benötigt, um während des Trainings- bzw. Recallprozesses (online) die Genauigkeit bzw. Zuverlässigkeit der Ergebnisse eines neuronalen Netzes beurteilen zu können. Die Genauigkeit eines Ergebnisses ist die Größe der Soll-Ist-Abweichung. Die Zuverlässigkeit des Ergebnisses entspricht dem Grad der Abwesenheit von Fehlern. Ein Fehler liegt vor, wenn die Genauigkeit des Ergebnisses kleiner als ein Schwellwert ist. Das Maß für die Größe der Soll-Ist-Abweichung hängt von der gegebenen Applikation ab. Ein einfaches Maß ist z.B. die euklidische (quadratische) Distanz. Im Unterschied zur euklidischen Distanz können die Ungenauigkeiten in den einzelnen Komponenten des Ergebnisses jedoch auch je nach Bedeutung und Folgen von Ungenauigkeiten unterschiedlich gewichtet werden. Darüberhinaus kann der gesamte Fehler logarithmiert bzw. potenziert werden oder, falls er bis zu einer gewissen Größe tolerierbar ist, abgeschrankt werden. Im weiteren wird als gemeinsamer Oberbegriff für Zuverlässigkeit und Genauigkeit die **Qualität des Ergebnisses** verwendet.

Die Abschätzung der Qualität von Ergebnissen eines Netzes wird in modularen Architekturen verwendet für die Auswahl eines geeigneten Teilnetzes, für die Bildung eines Ergebnisses mit höherer Qualität aus den Ergebnissen von r parallelen (redundanten) Teilnetzen und für die Entscheidung, ob das Ergebnis überhaupt brauchbar ist. Deshalb werden diese Verfahren in einem eigenen Kapitel vorgestellt.

Man kann grob zwei Ansätze zur Messung der Qualität von Ergebnissen unterscheiden: Erstens die Evaluierung der Outputvektoren der jeweiligen Netze und zweitens Verfahren, bei denen zusätzlich zu den Outputvektoren weitere Wertkomponenten zur Qualitätsmessung herangezogen werden.

5.1 Evaluierung der Outputvektoren

5.1.1 Messung der Ergebniseindeutigkeit

Für den Fall einer binären Codierung der Outputs, z.B. 1 aus n Codierung oder äquivalenten Codierungen wie Block Coding[1] (bei Klassifikationsaufgaben, jeder Klasse wird eine Unit zugeordnet) kann der analoge Ausgangswert der Outputunits des NN direkt ausgewertet werden, um ein Kriterium für die Zuverlässigkeit der Aussage des Netzes zu erhalten. Bei 1 aus n Codierung besteht der Outputvektor des Netzes im Idealfall aus n-1 mal 0 und 1 mal 1. Die zugrundeliegende Idee ist, daß ein Ergebnis umso zuverlässiger ist, je näher der Output des Netzes dem Idealfall kommt. Mit dem Ausgangswert y_j der j-ten Outputunit und m dem Index der maximal aktivierten Outputunit kann dies durch zwei Kenngrößen z_1 und z_2

$$z_1 = y_m = \max_j y_j \qquad z_2 = z_1 - \max_{j, j \neq m} y_j \tag{5.1}$$

dargestellt werden. Der Wert z_1 mißt die Differenz zwischen dem Output der maximal aktivierten Outputunit und dem idealen Wert 1, der Wert z_2 mißt die Differenz zwischen den Outputs der am stärksten und der am zweitstärksten aktivierten Unit. Für eine hohe Zuverlässigeit des Ergebnisses müssen beide Werte nahe 1 sein. Für binäre Ausgangsgrößen (Schwellwertunits) ist $z_2 \in \{0,1\}$, d.h. z_2 zeigt direkt an, ob das Ergebnis richtig ist (z_2=1) oder falsch (z_2=0).

Im Falle von Blockcoding kann auch der binäre Ausgangsvektor zur Schätzung der Zuverlässigkeit verwendet werden. Das Ergebnis gilt als zuverlässig, wenn am Ausgang ein zusammenhängender Block aus 1-en mit der korrekten Länge b erscheint (Bild 5.1).

Bild 5.1 *Zuverlässiger (a) und nicht zuverlässiger Outputvektor (b) bei Blockcoding*

5.1.2 Bewertung des mittleren Verhaltens eines Netzes, Validation

Die Bewertung der mittleren Qualität der Ergebnisse durch Validation ist ein weit verbreitetes Verfahren. Im Gegensatz zu den oben dargestellten Verfahren wird hier die Qualität der Ergebnisse während dem Training des Netzes gemessen. Dazu werden nach jeweils m Trainigsschritten n Validations- (Recall-) Schritte durchgeführt und die dabei auftretenden Fehler über die n Validationsschritte gemittelt. Dadurch kann der Verlauf der Netzwerkperformance während des Trainings beobachtet und ausgewertet werden. Für die Validationsschritte während des Trainings wird eine zum Trainings-

1. b 1-en ab einer gewissen Position (z.B. n = 8, b = 4 $\Rightarrow$ 00011110 = 3, 11000011 = 6).

und Testset disjunkte Datenmenge (Validationsset) verwendet, die im Idealfall genauso wie das Trainingsset den gesamten Inputraum abdeckt.

In Bild 5.2 ist ein typischer Verlauf eines (Validations-) Fehlers während des Trainingsprozesses eines einzelnen Netzes dargestellt. In diesem Beispiel steigt der Fehler am Validationsset ab einem gewissen Zeitpunkt wieder an, obwohl der Fehler am Trainingsset weiter abnimmt. Dies bedeutet, daß das Netz die Trainingsdaten zu genau lernt und damit auch Rauschen reproduzieren kann. Die Generalisierungsfähigkeit verschlechtert sich mit Fortdauer des Trainings (Overfitting). Diese Erkenntnis wird allgemein bei nichtkonvergenten Trainingsverfahren (Abbruch des Trainings zum Zeitpunkt der kleinsten Fehllerrate am Validationsset) eingesetzt [Finn93]. Bei modularen Systemen wird z.B. anhand des Validations Errors die Spezialisierung von Teilnetzen auf Teilaufgaben gesteuert (siehe Kapitel 7).

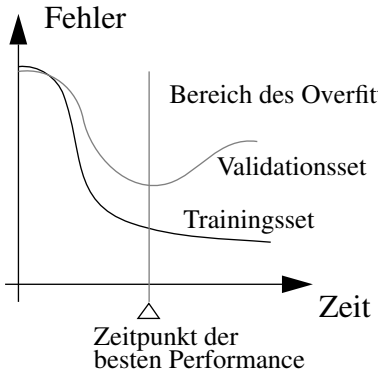


Bild 5.2 *Typischer Verlauf des Fehlers während des Trainings eines Netzes*

5.1.3 Vergleich von Ergebnissen verschiedener Netzwerke

Hier wird davon ausgegangen, daß mehrere verschiedene Netze parallel die gleiche Aufgabe lösen. Der Grad der Übereinstimmung der Einzelergebnisse ist ein Maß für die Qualität eines aus den Einzelergebnissen zusammengesetzten Gesamtergebnisses. Dieser Grad der Übereinstimmung der Einzelergebnisse wird durch eine Distanz der Einzelergebnisse gemessen. Die dafür verwendete Metrik hängt von der Aufgabenstellung ab. Voraussetzungen und Einschränkungen für die Anwendbarkeit dieses Kriteriums sowie Verfahren, aus n redundanten Ergebnissen ein robusteres zu bestimmen, sind in Kapitel 6 ab Seite 42 näher beschrieben.

5.2 Zusätzliche Wertkomponenten zur Qualitätsmessung

5.2.1 Input Reconstruction Reliabilty Estimation

Für den allgemeinen Fall eines analogen Netzausganges gibt es für supervised learning ein Verfahren zur Schätzung der Qualität des Ergebnisses, die Berechnung des Input Reconstruction Errors [Pome93]. Dazu wird während des Trainings des Netzes gleichzeitig zur eigentlichen Aufgabe die Rekonstruktion der Eingangsgröße (Autoassotiation, siehe Bild 5.3) trainiert. Zu diesem Zweck wird der Solloutput des NN und damit der Outputlayer um die Inputgrößen erweitert.

Die dahinter liegende Idee ist, daß der Grad der Korrelation des rekonstruierten (autoassoziierten) Inputs mit dem tatsächlichen Input ein Maß für die Zuverlässigkeit des anderen, des eigentlich interessanten Teils der Aussage des NN ist. Mit dem Mittelwert $\bar{I}$ der Komponenten des Inputvektors, dem Mittelwert $\bar{R}$ der Komponenten des rekonstruieren Inputvektors, dem Mittelwert $\overline{IR}$ der Produkte aller Komponenten von Inputvektor und rekonstruiertem Inputvektor und den Streuungen σ_I und σ_R der Komponenten der beiden Vektoren wird dies durch den üblichen Korrelationskoefizienten

$$\rho = \frac{\overline{IR} - \bar{I}\bar{R}}{\sigma_I \sigma_R} \tag{5.2}$$

gemessen. Der Korrelationskoeffizient ist ein Maß für die Zuverlässigkeit (Qualität). Der Reconstruction Error ist

$$z = 1 - |\rho| . \tag{5.3}$$

Versuche mit Backpropagation zeigten, daß der Fehler des Netzes tatsächlich stark mit dem Reconstruction Error korreliert ist [Pome93]. Der besondere Vorteil dieser Methode zur Qualitätsmessung ist, daß sie eine **absolute Performanceaussage** liefert. Es ist kein Vergleich mit anderen Ergebnissen erforderlich. Außerdem ist die Methode im Prinzip für Funktionsapproximation genauso verwendbar wie für Klassifiktionsaufgaben. Leider wurde jedoch die allgemeine Verwendbarkeit der Architektur noch nicht gezeigt, sodaß für jeden Einzelfall erst untersucht werden muß, ob diese Art der Zuverlässigkeitsschätzung wirklich funktioniert.

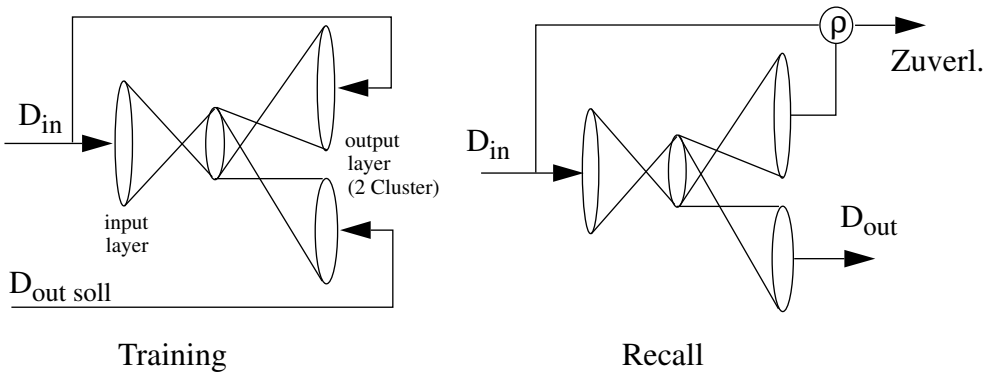


Bild 5.3 *Input-Rekonstruktion zur lokalen Schätzung der Zuverlässigkeit von Ergebnissen*

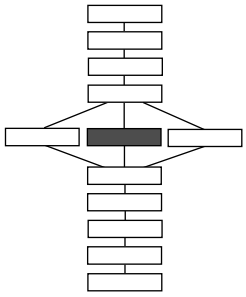
Eine Variation zur obigen Architektur besteht darin, die Gewichte des autoassoziativen Outputclusters nicht gemeinsam mit der eigentlichen Aufgabe zu trainieren, sondern erst im Anschluß daran, oder die Gewichte überhaupt zu berechenen. Die berechneten bzw. im zweiten Schritt trainierten Gewichte und die dadurch mögliche Rekonstruktion des Inputs ist ein Maß dafür, wie gut die Features im Netz codiert sind. Im Falle des zweistufigen Trainings müssen während des Trainings des autoassoziierenden Outputclusters alle übrigen Gewichte unverändert gehalten werden. Im Falle der Berechnung der Gewichte muß die Vereinfachung auf den linearen Fall getroffen werden (Inversion der Gewichtsmatrix des hidden Layers). Der Vorteil dieser Art der

Inputrekonstruktion liegt darin, daß das Training der eigentlichen Aufgabe nicht durch das gleichzeitige Training der Autoassoziation gestört wird. Jedoch liegen auch hier keine theoretischen Beweise vor, ob die Codierung der Inputs in den Output eine für die eigentliche Aufgabe günstige oder ungünstige Featurerepräsentation im hidden Layer erzwingt[2].

5.2.2 Bewertung der "Qualität des Modelles"

Zusätzlich zur direkten Abschätzung der Qualität der Ergebnisse von NN kann auch die Qualität des neuronalen Netzes, d.h. die Qualität des Modelles, berücksichtigt werden. In [Leon92], einer Arbeit, die sich mit "Model Recovery und Model Selection" im Bereich der Bildanalyse beschäftigt, werden Qualitätsaussagen über parametrische Modelle für geometrische Strukturen in den Bildern aus der Description Length jener Teile des Bildes, die durch das Modell repräsentiert werden, gebildet. Unter Verwendung dieser Qualitätsaussage werden aus n entwickelten Modellen diejenigen ausgewählt, die das gegebene Bild am effizientesten modellieren, d.h. zur größtmöglichen Ersparnis an Description Length führen. Übertragen auf NN bedeutet dies, daß als Qualitätsmerkmale die Anzahl freier Parameter, die Konvergenzgeschwindigkeit und dergleichen herangezogen werden kann, um aus mehreren Netzen auszuwählen.

2. Vergleiche rein autoassoziative Multilayer Perceptrons, die im mittleren hidden Layer den Raum der Principal Components aufspannen [Bour88], und Codierung der Outputs in den Input (output encoding) [Gosh90].



6 Integration von redundanten Einzelergebnissen

In Kapitel 4 wurde integrative Parallelität als eine der Basisarchitekturen vorgestellt. Bild 6.1 stellt diese Architektur nochmals dar. In diesem Kapitel werden Voraussetzungen und Lösungen zur Integration (Fusion) von redundanten Ergebnissen präsentiert.

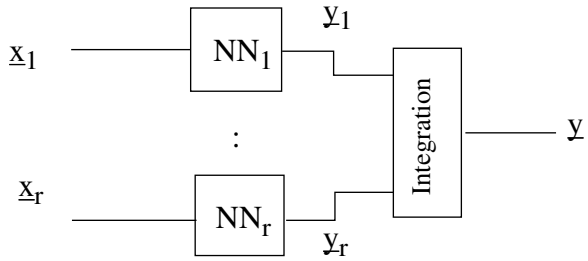


Bild 6.1 *Integration von Teilergebnissen zu einem Gesamtergebnis*

Bei Redundanz erfüllen r Teilnetze die gleiche Aufgabe. Alle Teilnetze werden einzeln auf die zu erfüllende Aufgabe trainiert. Beim Recall werden die redundanten Ergebnisse aller Teilnetze zu einem Gesamtergebnis integriert. Zweck der Redundanz ist die Verringerung der Fehlerrate und die Erhöhung der Robustheit[1] (siehe auch [Mint91]

1. Robustheit bedeutet geringe Störungsanfälligkeit auch in einer gestörten Umgebung. Eine gestörte Umgebung zeigt sich daran, daß Eingangsdaten und/ oder Sollausgangsdaten verrauscht sind (gleichverteilt, gaußverteilt, Salz- und Pfeffer Rauschen, additiv, multiplikativ). Mögliche Ursachen für Störungen sind: Meßungenauigkeiten, Übertragungsprobleme und Fehler in den Trainingsdaten (z.B. fehlerhafte Klassifizierungen). Zu bemerken ist, daß Redundanz systematische Fehler wie unvollständige Datenerfassung -- multimodale “Störungen” können z.B. auf fehlende Einflußfaktoren hindeuten -- und falsche Datenvorverarbeitung (z.B. Fehler durch Normalisierung) nicht korrigieren kann.

für robustes Model Fitting im Bereich der Bildverarbeitung). Ein intuitives Beispiel dazu ist in Bild 6.2 visualisiert.

Die der Verringerung der Fehlerrate durch Redundanz zugrundeliegende Hypothese ist, daß die Fehler der Einzelnetze voneinander unabhängig sind. D.h., daß verschiedene Netze verschiedene Resultate liefern, wenn ihre Resultate falsch sind (independent confusion) und dieselben Ergebnisse liefern, wenn ihre Ergebnisse richtig sind.

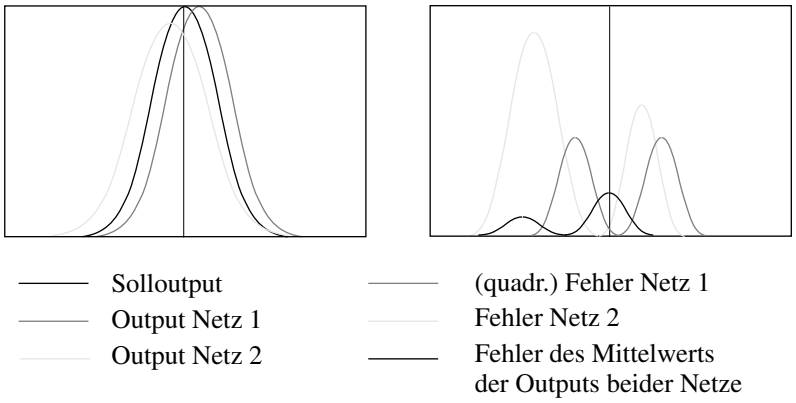


Bild 6.2 *Intuitives Beispiel zu Redundanz: Integration von 2 Teilnetzen*

Lösungen:

Unabhängigkeit der Fehler mehrerer Einzelnetze wird durch **Diversität** der parallelen neuronalen Netze erreicht, d.h. durch unterschiedliche Inputdaten (unterschiedliche Inputgrößen bzw. unterschiedliche Auflösungen der selben Inputgrößen), unterschiedliche NN- Paradigmen (z.B. lokal und global optimierende Netze, oder supervised und unsupervised lernende Netze), mindestens jedoch durch unterschiedliche Architektur- und Lernparameter. Eine weitere Steigerung der Diversität kann durch zusätzliche Verwendung von nichtneuronalen Modulen erreicht werden.

Darüber hinaus kann zunächst die Verteilung der falschen Ergebnisse für jedes Netz gemessen werden, um dann jene Netze (Module) mit möglichst unterschiedlichen Verteilungen der falschen Ergebnisse zu verwenden.

Außerdem haben zahlreiche Versuche [Batt94][Perr94] ergeben, daß der Performance- bzw. Zuverlässigkeitsgewinn durch Redundanz dann besonders groß ist, wenn im Mittel die parallelen Einzelnetze sehr ähnliche Einzelperformance aufweisen.

6.1 Integrationsprinzipien für Klassifikationsaufgaben

In diesem Abschnitt werden Prinzipien zur Integration von redundanten Ausgangsgrößen bei Klassifikationsaufgaben präsentiert. Die dargestellten Integrationsmechanismen sind jedoch auf alle Aufgaben mit binärer 1 aus n oder äquivalenter Codierung der Ausgangsgrößen übertragbar.

Die im folgenden vorgestellten Integrationsmechanismen liefern neben dem üblichen Klassifikationsergebnis auch eine Aussage über die Zuverlässigkeit des Ergebnisses bzw. eine Aussage darüber, ob das jeweils gegebene Inputmuster klassifizierbar ist oder als unklassifizierbar zurückgewiesen werden muß (reject). Allen Integrationsmechanismen gemeinsam ist der Konflikt zwischen Zuverlässigkeit und Rejection. Wird eine hohe Rate an Inputmustern zurückgewiesen (Einstimmigkeitsentscheid oder hohe Performancegrenzwerte), so sind die Ergebnisse für akzeptierte Inputmuster sehr zuverlässig. Und umgekehrt: Bei einer hohen Rate an akzeptierten Inputmustern ist auch die Rate der Fehlklassifikationen unter den akzeptierten Mustern groß.

Es sind eine Reihe von Integrationsmechanismen für Klassifikaitonsaufgaben bekannt. Einige davon wurden in [Cho93] u. [Batt94] untersucht:

6.1.1 Voting

- **Einstimmigkeitsentscheid** (r aus r Redundanz): Für jedes Netz wird die der maximal aktivierten Outputunit zugeordnete Klasse als Ergebnis genommen. Falls alle Einzelergebnisse übereinstimmen, ist ein Gesamtergebnis gefunden, ansonsten gilt das Eingangsmuster als nicht klassifizierbar.
- **Mehrheitsentscheid** (m aus n, m < n Redundanz): Mindestens m Netze müssen das gleiche Ergebnis liefern, damit ein Gesamtergebnis gefunden ist. Zusätzlich kann ein zweiter Schwellwert p mit m - k >= p und k die zweitgrößte Zahl an gleichen Ergebnissen eingeführt werden, der angibt, wieweit sich eine erste Mehrheit m von der zweiten Mehrheit k zahlenmäßig mindestens unterscheiden muß, damit das Ergebnis als zuverlässig angesehen wird. Dies funktioniert natürlich nur bei einer genügend großen Zahl an parallelen Netzen. Mit den Parametern m und k können Zuverlässigkeit und Rejection gesteuert werden.
- **Verwendung der Zuverlässigkeitsschätzungen**: Diese Form der Integration kann im Unterschied zu den beiden obigen Kriterien nur auf neuronale Netze, nicht aber auf beliebige Klassifikationsmodule angewandt werden. Dabei werden für jedes der parallelen Teilnetze eigene Zuverlässigkeitsschätzungen (siehe Kapitel 5) gebildet. Die einfachste Form der Integration unter Verwendung der Zuverlässigkeitsschätzungen ist es, als Gesamtergebnis das Teilergebnis mit der größten Einzel-Zuverlässigkeitsschätzung zu verwenden. Ein Ansatz, der die Redundanz jedoch besser ausnutzt, ist die Kombination der Zuverlässigkeitsschätzungen der Teilnetze mit dem Voting. Dabei werden analoge Schwellwerte (s, s_1,s_2) für die Zuverlässigkeitsschätzungen z, z_1, z_2 der Einzelnetze aus Kapitel 5 (Gl. 5.1 und Gl. 5.3) vorgegeben. Nur solche Teilergebnisse gelten als zuverlässig deren zugehörige Zuverlässigkeitsschätzwerte über dem Schwellwert liegen. Einstimmigkeitsentscheid bedeutet dann, daß alle Teilnetze ein zuverlässiges Einzelergebnis liefern und alle Teilergebnisse übereinstimmen (entsprechend für Mehrheitsentscheid).

  Eine weniger restriktive Möglichkeit, die Zuverlässigkeitswerte mit Einheitsentscheid bzw. Mehrheitsentscheid zu kombinieren, ist alle zurückweisenden Einzelnetze nicht zu berücksichtigen und die Entscheidung unter den Netzen zu treffen, die eine zuverlässige Aussage liefern. Die Idee dahinter ist, daß je nach verwende-

ten Features (und/oder Lernalgorithmen) eine Klassifikation möglich oder unmöglich ist. Im Durchschnitt senkt diese Vorgangsweise die Rejection Rate, hebt aber die Fehlerrate innerhalb der akzeptierten Muster.

6.1.2 Mittelwertbildung

- **Mittelwertbildung** der einzelnen Netzergebnisse: Dabei werden die reellwertigen Ausgangsvektoren der einzelnen Netze gemittelt.

$$\underline{y} = \frac{1}{r}\sum_{k=1}^{r}\underline{y}_k \tag{6.1}$$

  Das Inputmuster wird dann jener Klasse zugeordnet, deren zugehörige Komponente des Mittelwerts maximal ist.

  Im allgemeinen kann gesagt werden, daß Mittelwertbildung zu geringfügig besseren Resultaten führt als Integration durch Voting [Perr94]. Die Zuverlässigkeitsschätzungen z, z_1, z_2 aus Kapitel 5 können auch für den Mittelwert berechnet werden. Eine Entscheidung, ob das jeweilige Inputmuster akzeptiert oder zurückgewiesen werden soll, wird dann anhand der Zuverlässigkeitsschätzungen für den Mittelwert getroffen. Die Zuverlässigkeitsschätzungen für den Mittelwert sind ein geringfügig stärkeres Kriterium als die Zuverlässigkeitsschätzungen für die Einzelnetze (geringere Fehlerrate, aber größere Rejection Rate).

- **Verwendung von Performanceabschätzungen**: Der Mittelwert (Gl. 6.1) kann dadurch verbessert werden, daß die Qualitätsschätzungen aus Kapitel 5 zur Gewichtung der Einzelergebnisse verwendet werden. Mit z_k der Qualität des Ergebnisses des k-ten Teilnetzes und α einem Parameter zur Bestimmung der Stärke der Gewichtung ist dann

$$y = \frac{1}{r\bar{z}}\sum_{k=1}^{r} z_k^{\alpha}\underline{y}_k \qquad \text{mit} \qquad \bar{z} = \sum_{k=1}^{r} z_k^{\alpha} \tag{6.2}$$

  Dadurch tragen Teilnetze mit höherer Zuverlässigkeit stärker zum Gesamtergebnis bei. Qualitätsschätzungen für das Gesamtergebnis können dann auch aus den Qualitätsschätzungen der Teilergebnisse gebildet werden.

6.1.3 Probabilistische Verfahren

Basis der probabilistischen Verfahren ist die Bayes'sche Entscheidungstheorie [Duda73] sowie die Erkenntnis, daß die Outputwerte von Multi Layer Perceptrons bei Klassifikaitonsaufgaben die a-posteriori Warscheinlichkeiten der Klassen approximieren [Wan90]. Grundlage der Bayes'schen Entscheidungstheorie ist die Bayes'sche Regel:

Mit der a-priori Wahrscheinlichkeit $P(w_c)$ der c-ten Klasse, der Wahrscheinlichkeits-Dichte $p(\underline{x})$ aller Inputmuster $\underline{x}$ und der Wahrscheinlichkeits-Dichte $p(\underline{x}|w_c)$ all jener

Inputmuster $\underline{x}$ die der Klassse c angehören[2] ergibt sich die a-posteriori Wahrscheinlichkeit $P(w_c|\underline{x})$ der c-ten Klasse für ein konkretes gegebenes Inputmuster $\underline{x}$ zu

$$P(w_c|\underline{x}) = \frac{p(\underline{x}|w_c)\,P(w_c)}{p(\underline{x})} \tag{6.3}$$

mit

$$p(\underline{x}) = \sum_{i=1}^{q} p(\underline{x}|w_i)\,P(w_i) \tag{6.4}$$

Für die Integration speziell von Ergebnissen von mehreren neronalen Netzen wird eine Verallgemeinerung der Bayes'schen Entscheidungstheorie verwendet: die Dempster-Shafer-Theorie.

Die **Dempster-Shafer-Theorie** ist eine Methode zum Umgang mit Glaubwürdigkeitsaussagen (measures of evidence). Im Unterschied zur Bayes'schen Entscheidungstheorie können mit der Dempster-Shafer-Theorie auch unvollständiges Wissen, Unsicherheit und Gegenhypothesen gehandhabt werden.

In [Rogo94] wird eine Methode zur Berechnung von Glaubwürdigkeitsaussagen für die Ergebnisse $\underline{y}_k$ von r unterschiedlichen NN präsentiert die auf der Dempster-Shafer-Theorie basiert. Mit $\underline{E}_{k,c}$ dem Mittelwert der Ergebnisvektoren $\underline{y}_k$ des k- ten Teilnetzes für alle Inputmuster $\underline{x}$ im Testset, die der Klasse c aus insgesamt q Klassen angehören und

$$d_{k,c} = \frac{\left(1 + \left\|\underline{E}_{k,c} - \underline{y}_k\right\|^2\right)^{-1}}{\sum_{1 \le i \le q} \left(1 + \left\|\underline{E}_{k,i} - \underline{y}_k\right\|^2\right)^{-1}}, \tag{6.5}$$

einem Maß für die "Nähe" des Ergebnisses $\underline{y}_k$ zum Mittelwert $\underline{E}_{k,c}$, ergibt sich die Glaubwürdigkeit des k- ten Teilnetzes, daß der Input $\underline{x}$ der Klasse c angehört zu

$$e_c(\underline{y}_k) = \frac{d_{k,c}\prod_{i \neq c}(1 - d_{k,i})}{1 - d_{k,c}\left(1 - \prod_{i \neq c}(1 - d_{k,i})\right)}. \tag{6.6}$$

Mit einer Normalisierungskonstanten a ist damit die gesamte Glaubwürdigkeit über alle Teilnetze, daß das Inputmuster $\underline{x}$ der Klasse c angehört

2. Mit w dem "State of Nature", der die Ausprägungnen $w_1 \ldots w_q$ haben kann, wird $p(x|w_c)$ allgemein als "state conditional propability density" bezeichnet [Duda73].

$$e_c(\underline{x}) = a \prod_{k=1}^{r} e_c(\underline{y}_k) \tag{6.7}$$

Ein Inputmuster $\underline{x}$ wird dann jener Klasse zugeordnet, für die die maximale Glaubwürdigkeit

$$e = max_{1 \le c \le q} e_c(\underline{x}) \tag{6.8}$$

erreicht wird.Die Integration der Aussagen von n Klassifikationsnetzwerken mittels Dempster-Shafer-Theorie ist mathematisch aufwendiger als die Voting- und Mittelwert-Verfahren. Die prinzipielle Verwendbarkeit der Methode wurde in [Rogo94] demonstriert, jedoch kein Performancevergleich mit anderen Integrationsverfahren durchgeführt.

6.1.4 Neuronale Fusion

Bei dieser Form der Integration werden die Einzelergebnisse der Teilnetze durch ein supervised lernendes neuronales Netz (Integrations- bzw. Fusionsnetz) zu einem Gesamtergebnis integriert. Der Input des Fusionsnetzes besteht aus allen Outputvektoren der Teilnetze. Zusätzlich kann das Fusionsnetz Zuverlässigkeitsabschätzungen der Teilnetze als Input erhalten.

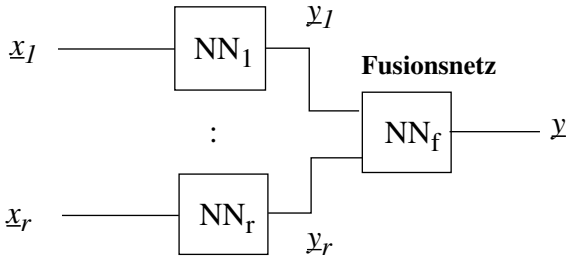


Bild 6.3 *Fusion der Einzelergebnisse durch neuronales Postprocessing*

Wie bei den anderen Integrationsprinzipien werden auch bei der neuronalen Fusion die Teilnetze einzeln auf die zu lösende Aufgabe trainiert. Das Fusionsnetz wird im Anschluß an die Teilnetze trainiert. Es erhält denselben Solloutput wie die Teilnetze.

Diese Form der Integration wird experimentell in Abschnitt 11.3 untersucht.

6.2 Integrationsprinzipien für kontinuierliche Ausgangssignale

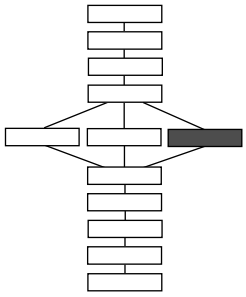
Kontinuierliche (reellwertige) Ausgangsgrößen treten bei der Approximation kontinuierlicher multidimensionaler Funktionen auf. Hier wird im Unterschied zum diskreten Fall der Klassifikation der gesamte Outputvektor des Netzes als Ergebnis verwendet.

- **Mittelwertbildung**: Mittelwertbildung ist die einfachste Form der Integration von r, dieselbe Funktion approximierenden Teilnetzen. In diesem Fall hat die Instanz, die alle Einzelergebnisse integriert, das arithmetische Mittel zu bilden. Liefert ein

Einzelnetz ein falsches Ergebnis, so wirkt es sich nur mit dem Gewicht *1/r* auf das Gesamtergebnis aus. Lincoln [Linc90] hat diesen Fall für die Verwendung eines Clusters von 5 Backpropagation-Netzen zur Buchstabenerkennung untersucht. Er hat empirisch festgestellt, daß durch Cluster aus Backpropagation-Netzen für nicht-triviale[3] Lernprobleme Performanceverbesserungen erreicht werden können. Das robustere Gesamtergebnis kann sogar unter gewissen Voraussetzungen zum weiteren Training der Einzelnetze verwendet werden.

- **Einstimmigkeitsentscheid**: Einstimmigkeit heißt hier, daß alle Distanzen zwischen dem Mittelwert der Ergebnisvektoren aller Teilnetze und dem Ergebnisvektor jeweils eines Teilnetzes kleiner sind als ein Schwellwert. Wie die Distanz zu berechnen ist, hängt von der Aufgabenstellung ab. Häufige Maße sind euklidische Distanz (quadratische Differenz), Manhatten Distanz bzw. die maximale Differenz einzelner korrespondierender Vektorkomponenten. Das Gesamtergebnis ist dann jeweils der Mittelwert aller Einzelergebnisse.
- **Mehrheitsentscheid**: Mehrheitsentscheid unter Verwendung der Schätzungen der Zuverlässigkeit ist ebenso sinngemäß wieder verwendbar. Der Unterschied ist jedoch wieder, daß das Gesamtergebnis nicht das Ergebnis einer Abstimmung ist, sondern deren Mittelwert. Es kann auch eine Gewichtung der Anteile am Gesamtergebnis entsprechend den Zuverlässigkeitsschätzungen erfolgen.
- **Neuronale Fusion**: Neuronale Fusion (siehe oben) verwendet die reellwertigen Outputvektoren der Teilnetze und liefert reellwertige Outputvektoren. Wie die Mittelwertbildung kann deshalb die neuronale Fusion unverändert auch für Funktionsapproximation verwendet werden.

3. Nichttrivial heißt, daß die geforderte Abbildung anhand der gegebenen Trainingsbeispiele durch ein einzelnes Backpropagation-Netz nicht hinreichend genau erlernt werden kann.



7 Selbstorganisierende Selektion von Teilnetzen

Bei selektiver Parallelität (siehe Kapitel 4.1 über neuronale Grundarchitekturen) bestimmt ein Selektor (-Netz) für jeden ankommenden Datensatz, welches Teilnetz ausgewertet wird (Bild 7.1). Dies entspricht einer Unterteilung des Eingangsraumes in disjunkte Teilräume.

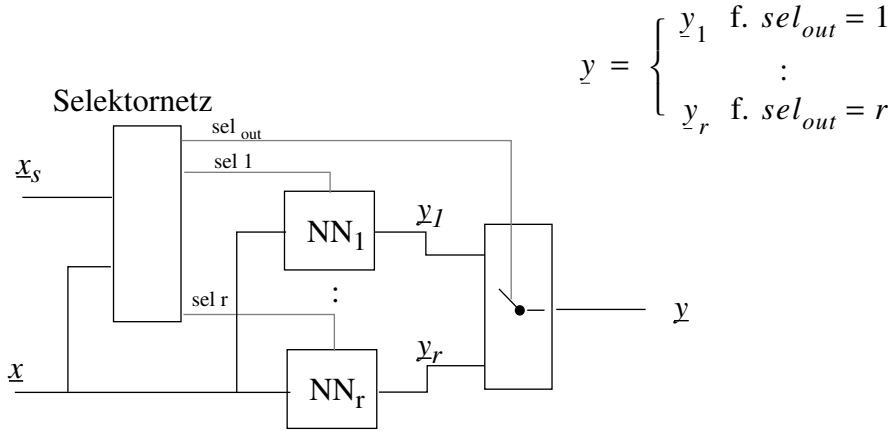


Bild 7.1 *Selektion eines Teilnetzes*

In diesem Kapitel werden Algorithmen zur selbstorganisierenden Selektion von Teilnetzen präsentiert. Im Unterschied zur fixen, z.B. anhand von Vorwissen eingestellten Selektion von Teilnetzen entwickelt sich bei selbstorganisierender Selektion die Zuordnung von Eingangsvektoren zu Teilnetzen und damit die Aufteilung des Eingangsraumes auf die Teilnetze automatisch. Die selbstorgansisierende Selektion von Teilnetzen entspricht damit einer **automatischen Dekomposition** der gesamten Aufgabe in Teilaufgaben.

7.1 Selektion durch Clustering

Die einfachste Form der automatischen Selektion (Dekomposition) ist Selektion durch Clustering der Eingangsdaten. Der Selektor ist ein beliebiger Clustering-Algorithmus. Jedem Cluster der Eingangsdaten (Teilraum) wird genau ein Teilnetz zugeordnet. Die Aufteilung des Eingangsraumes und damit die Dekomposition erfolgt unsupervised in Abhängigkeit von der Verteilung der Eingangsdaten und dem im Clustering-Netz verwendeten Ähnlichkeitsmaß.

Im Falle eines adaptiven Clusterings kann der Selektor ein NN (Clusterning-Netz) sein, z.B.

- ein Kohonen-Netz [Koho89],
- ein Soft Competitive Learning-Netz [Nowl90],
- eine growing- [Frit91] bzw. dynamic Cell-Structure [Brus93] bzw.
- ein Neural Gas-Netz [Mart92].

Jeder Unit des Clustering-Netzes wird genau ein Teilnetz zugeordnet. Beim Recall wird für jeden Inputvektor zunächst das Clustering-Netz ausgewertet. Dabei werden die Aktivierungen aller Units des Clustering-Netzes anhand der durch die Metrik des Clustering- Netzes definierten Ähnlichkeiten zwischen Inputvektor und Gewichtsvektoren der jeweiligen Units berechnet. Jenes Teilnetz, das der am stärksten aktivierten Unit des Selektors zugeordnet ist, wird zur Berechnung des eigentlichen Ergebnisses verwendet. Der Inputbereich, innerhalb dem eine Unit des Clusterning- Netzes maximal aktiviert wird, ist das rezeptive Feld des der Unit zugeordneten Teilnetzes.

Das Training dieser Architektur aus Clustering-Netz und Teilnetzen erfolgt zweistufig. Vor dem Training der Teilnetze wird das Clustering-Netz trainiert. Das Trainig der Teilnetze erfolgt anschließend unter Verwendung des bereits trainierten Clustering-Netzes.

Ein konkretes Beispiel für diese Architektur ist in [Iwat90] zu finden. Als Clustering-Netz für die Erkennung von chinesischen Schriftzeichen wurde ein Learning Vector Quantization-Netz, als Teilnetze Backpropagation-Netze verwendet. Eine Variante dieses Verfahrens ergibt sich, wenn beim Clustering-Netz, sobald ein Teilnetz nicht mehr in der Lage ist, die Trainingsbeispiele ausreichend genau zu erlernen, dynamisch zusätzliche Units und damit zusätzliche Teilnetze hinzugefügt werden [Iwat92]. Dadurch kann inkrementelles Lernen realisiert werden.

7.2 Selektion durch fehlergesteuertes Clustering

Wie bei der Selektion (Dekomposition) durch Clustering ist auch hier der Selektor ein Clustering-Netzwerk, wobei jeder Unit des Clustering-Netzes genau ein Teilnetz zugeordnet wird. Im Unterschied zu oben ist jedoch der Ordnungsprozeß des Clustering-Netzes nicht mehr ausschließlich von der Verteilung der Eingangsdaten abhängig, sondern erfolgt unter Berücksichtigung der Fehler der Teilnetze. Dadurch kommt es zu einer von der “Schwierigkeit der Teilaufgabe” gesteuerten Dekomposition des Eingangsraumes.

Dies wird erreicht durch synchrones Training von Clustering-Netz und Teilnetzen. Ein Trainingsschritt der gesamten Architektur bedeutet zunächst einen Recallschritt am Clustering-Netz, dann einen Recallschritt am selektierten Teilnetz, anschließend ein Trainingsschritt am Clustering-Netz unter Verwendung des Fehlers (Performancewert) des Teilnetzes und schließlich ein Trainingsschritt des Teilnetzes. Das Clustering wird derart beeinflußt, daß Teilnetze, die das gegebene Beispiel besonders gut lösen, stärker selektiert werden.

Ein konkretes Beispiel zu fehlergesteuerter Dekomposition ist [Fox91]. Dabei wird ein Kohonen-Netz als Selektor verwendet. Der Einfluß der Teilnetze beim Lernen wird dadurch modelliert, daß die Lernrate des Kohonen-Netzes mit dem Fehler des selektierten Teilnetzes gewichtet wird. Bei einem großen Fehler des Teilnetzes kommt es zu einer Verkleinerung des jeweiligen rezeptiven Feldes.

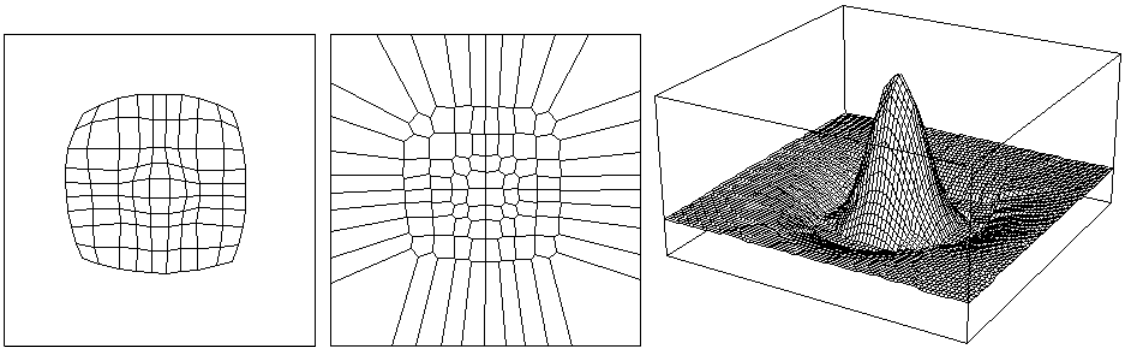


Bild 7.2 *Selektion (Dekomposition) durch fehlerabhängiges Clustering am Beispiel Approximation der Mexican Hat-Funktion (aus [Fox91])*
a) Verteilung der Teilnetze, b) Rezeptive Felder, c) Plot der Approximation

Im Bild 7.2 wird das Verhalten dieser Netzarchitektur am Beispiel der Approximation der Mexican Hat-Funktion illustriert. Die Teilnetze sind Standard-Backpropagation-Netze. Deutlich zu erkennen sind die unterschiedlichen Größen der Inputbereiche (rezeptiven Felder) der einzelnen Netze.

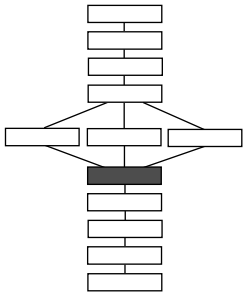
7.3 Selektion durch Kompetition der Teilnetze

Bei Selektion (Dekomposition) durch Kompetition entwickelt sich die Aufteilung des Eingangsraumes während des Trainingsprozesses ausschließlich anhand der online Performanceabschätzungen für die Teilnetze.

Der Selektor ist in diesem Fall ein supervised lernendes neuronales Netz. Es erhält beliebige Eingangsdaten (z.B. dieselben wie die Teilnetze und/oder spezielle Selektionskriterien), die es auf eine Aussage über das am besten geeignete Teilnetz abbilden soll. Die Aufgabe des Selektornetzes ist somit eine Klassifikationsaufgabe. Jedes supervised zu trainierende Netz, das zur Klassifikation geeignet ist, kann als Selektornetz verwendet werden. Das Training des Selektornetzes erfolgt simultan mit dem Training der Teilnetze. Im einfachsten Fall werden für jedes Trainingsbeispiel

zunächst alle Teilnetze ausgewertet. Aus dem Teilnetz mit dem geringsten Fehler ergibt sich der Soll-Ausgangsvektor für den Trainingsschritt des Selektornetzes (1 aus r Codierung, die Komponente des Teilnetzes mit dem geringsten Fehler hat einen Wert von 1, alle anderen 0). Nur das Teilnetz mit dem kleinsten Fehler wird trainiert.

Im konkreten Einsatz (Recall) dieser Architektur bestimmt zunächst das Selektornetz jenes Teilnetz, das dann das eigentliche Ergebnis liefert.



8 Mixture Models

In den letzten beiden Kapiteln wurden die Mechanismen Selektion und Integration von Teilnetzen behandelt. In diesem Kapitel werden diese beiden Mechanismen kombiniert, mit dem Ziel die Vorteile beider Mechanismen zu vereinen.

8.1 Soft und hard Split

Bisher wurde bei der Selektion von Teilnetzen davon ausgegangen, daß für jeden Inputvektor jeweils genau ein Teilnetz selektiert wird, das alleine das Ergebnis für das gesamte System liefert. D.h. die den Teilnetzen zugeordneten Teilbereiche des Eingangsraumes überlappen sich nicht. Man spricht von einer scharfen Unterteilung des Eingangsraumes (**hard Split**). Dieser hard Split ist sinnvoll, wenn er anhand von Vorwissen fix eingestellt wird (siehe Kapitel 9 zu manueller Dekomposition). Er hat jedoch in vielen Fällen den Nachteil, daß die Genauigkeit des Gesamtergebnisses in den Übergangsbereichen klein ist und Diskontinuitäten beim Übergang von einem Netz zum anderern auftreten (siehe Bild 7.2).

Abhilfe schafft die Kombination von Selektion und Integrtion in den Mixture Models (Bild 8.1). Dabei wird von den insgesamt r Teilnetzen nicht genau ein Teilnetz selektiert, sondern $1 \leq q \leq r$. Die Ergebnisse aller q selektierten Teilnetze werden zum Gesamtergebnis integriert. Man spricht von einer weichen Unterteilung des Inputraumes (**soft Split**). Die Inputbereiche der Teilnetze können sich dabei beliebig weit überlappen. Die reine integrative Architektur (alle Teilnetze werden ausgewertet, siehe Kapitel 6) und auch die reine selektive Architektur (genau ein Teilnetz wird ausgewertet, siehe Kapitel 7) sind beides extreme Ausprägungen von Mixture Models.

Die Mixture Models haben ihre Entsprechung bei den monolithischen neuronalen Netzen in den Radial Basis Function-Netzwerken [Rich91]. Bei den Radial Basis Function-Netzwerken wird die gesamte Abbildung durch die gewichtete Summe von radialen Teilabbildungen (meist multidimensionalen Gaußfunktionen) gebildet.

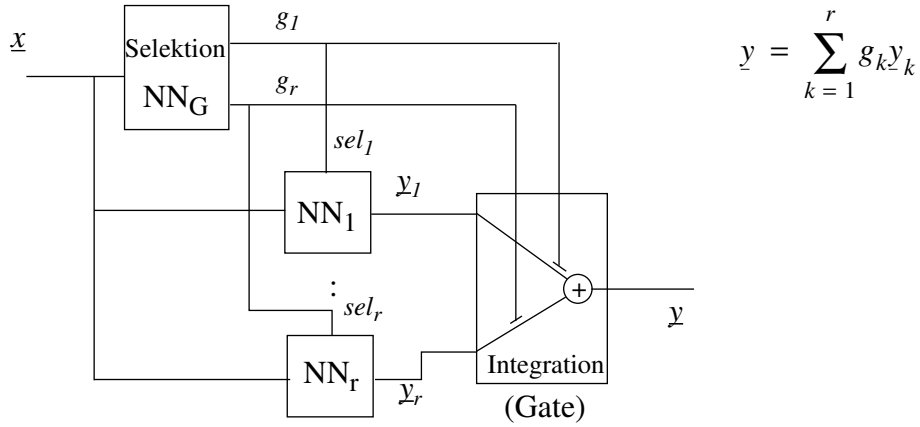


Bild 8.1 *Kombination von Selektion und Integration von Teilnetzen bei Mixture Models*

Beim Mixture Model ist diese Idee auf Teilnetze übertragen. Das Gesamtergebnis wird aus den Einzelergebnissen gemischt. Ein zusätzlicher Vorteil dieser Mischung ist, daß sich Teilergebnisse verschiedenartiger Teilnetze, z.B. linearer und nichtlinearer, mischen lassen. Dies bringt die im Kapitel Redundanz besprochenen Vorteile höherer Zuverlässigkeit, Genauigkeit und Robustheit. Da bei den Mixture Models meist nur q << r Teilnetze selektiert werden, bleibt auch der Vorteil des geringeren Rechenzeitbedarfs und der besseren Skalierbarkeit der selektiven Architektur weitgehend erhalten. Außerdem sind im Vergleich zur (rein) selektiven Architektur bei gleicher Ergebnisgenauigkeit insgesamt weniger Teilnetze erforderlich (vergleiche Coarse Coding [Hint86]).

8.2 Gating

Im Unterschied zur Selektion ist die Aufgabe des Selektornetzes bei soft Split keine reine Klassifikationsaufgabe (wähle das beste Netz). Vielmehr muß die optimale Kombination aus Teilergebnissen bestimmt werden. Für diese optimale Kombination muß nicht nur erlernt werden, welche der Teilergebnisse für den jeweiligen Inputvektor zum Gesamtergebnis beitragen, sondern auch, mit welchem Gewicht die einzelnen Teilergebnisse berücksichtigt werden. Das Gewichten der Ergebnisse der Teilnetze wird als **Gating** bezeichnet [Jaco90]. Man spricht deshalb auch vom **Gating-Netz** anstelle vom Selektornetz. Das Erlernen der optimalen Kombination aus Teilergebnissen entspricht wieder einer automatischen Dekomposition des Eingangsraumes.

8.2.1 Gating durch Clustering

Bei Gating durch Clustering bzw. fehlergesteuertes Clustering ist das Gating-Netz ein beliebiges Clustering-Netz. Jeder Unit des Clustering-Netzes wird genau ein Teilnetz zugeordnet. Im Unterschied zur Selektion durch Clustering werden all jene Teilnetze selektiert, die den q (fixe Zahl) am stärksten aktivierten Units des Gating-Netzes entsprechen, bzw. alle Teilnetze, deren zugehörige Units über einem gewissen Schwellwert liegen (siehe Visualisierung in Bild 8.2). Das rezeptive Feld eines Teilnetzes ist

nicht mehr jener Inputbereich der zur maximalen Aktivierung der entsprechenden Unit führt, sondern ein größerer, nämlich jener Inputbereich in dem eine Unit des Clustering-Netzes so stark aktiviert wird, daß das der Unit zugeordnete Teilnetz selektiert wird.

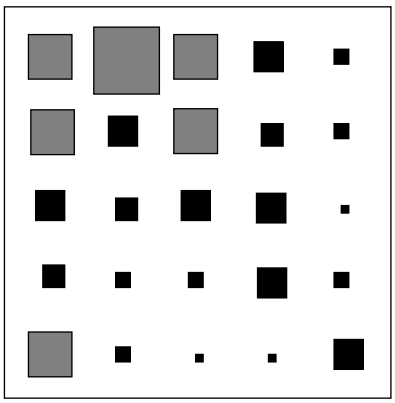


Bild 8.2 *Activity Map eines zweidimensionalen Clustering-Netzes mit 5 x 5 Units. Die Größe der Quadrate symbolisiert den Grad der Aktivierung der Units. Die Ergebnisse der den Units mit den grau eingefärbten Aktivierungen zugeordneten Teilnetze werden zum Gesamtergebnis integriert.*

Für die Integration der q zum Gesamtergebnis beitragenden Teilergebnisse kann prinzipiell jedes in Kapitel 6 beschriebene Verfahren verwendet werden. Meist werden jedoch gewichtete Mittelwerte verwendet [Jaco90],[Perr94]. Mit dem Aktivierungswert o_k der k-ten Unit des Gating-Netzes, dem Output $\underline{y}_k$ des k-ten Teilnetzes und

$$g_k = f(o_k) \quad \text{mit} \sum_{k=1}^{r} g_k = 1 \tag{8.1}$$

und $g_k = 0$ für alle nicht zu selektierenden Teilnetze ist das Gesamtergebnis

$$\underline{y} = \frac{1}{r} \sum_{k=1}^{r} \underline{y}_k g_k \tag{8.2}$$

Dadurch tragen stärker selektierte Teilnetze stärker zum Gesamtergebnis bei. Beim Training der Teilnetze wird die Lernrate des jeweiligen Teilnetzes mit dem Faktor g_k gewichtet, sodaß jene Teilnetze auch stärker lernen, die stärker selektiert sind. Zusätzlich kann ein Parameter z_k eingeführt werden der den Grad der Gewichtung beeinflußt. Er kann entweder fest, oder durch eine Performanceabschätzung (siehe Kapitel 5) für jedes Teilnetz individuell eingestellt werden. Die Geleichung (8.2) ändert sich in diesem Fall zu

$$\underline{y} = \frac{1}{r} \sum_{k=1}^{r} z_k \underline{y}_k g_k \qquad \text{mit} \qquad \sum_{k=1}^{r} z_k = 1 \qquad \text{und} \qquad \sum_{k=1}^{r} g_k = 1 . \tag{8.3}$$

Performanceschätzungen für das Gesamtnetz können sowohl aus Perfomanceabschätzungen für das Selektor-Netz (“wie eindeutig ist die Zuordnung”), als auch für das Gesamtergebnis gebildet werden.

8.2.2 Gating durch Kompetition der Teilnetze

Bei Gating (Dekompositon) durch Kompetition der Teilnetze lernen die Teilnetze und das Gating- (Selektor-) Netz synchron. Die Spezialisierung der r Teilnetze auf r Teilbereiche des Eingangsraumes entwickelt sich mit der Zeit während des Trainingsprozesses anhand der online Performanceabschätzungen für die Teilnetze. Zu Beginn des Trainings müssen deshalb alle Teilnetze parallel betrieben werden. Jedes Netz lernt alle Trainingsbeispiele. Beim Recall werden alle Teilergebnisse zum Gesamtergebnis integriert. Mit Fortdauer des Trainings der Teilnetze zeigen die einzelnen Teilnetze unterschiedliche Fehlerraten. Diese Fehlerraten werden verwendet, um das Gating-Netz zu trainieren und damit die Spezialisierung voranzutreiben. Das Gating-Netz lernt anhand der Fehler der Teilnetze jeden Inputvektor nur jenen Teilnetzen zuzuordnen, die diesen Inputvektor besonders gut erlernen können. Das Ergebnis dieses Prozesses ist, daß sich jedes Teilnetz auf einen Bereich des Inputraumes spezialisiert, wobei sich diese Bereiche jedoch überlappen können.

Eine Architektur, die Selektion und Integration kombiniert und durch Kompetition zu einer Dekompositon der Aufgabe führt, wurde erstmals von Jacobs et. al. [Jaco90] vorgestellt (Bild 8.3). In [Jaco90] wird auch demonstriert, daß diese Architektur dazu tendiert jedem Teilnetz, die für dieses Teilnetz besonders geeignete Teilaufgabe zuzuordnen. Die Teilnetze werden deshalb auch Expertnetze genannt.

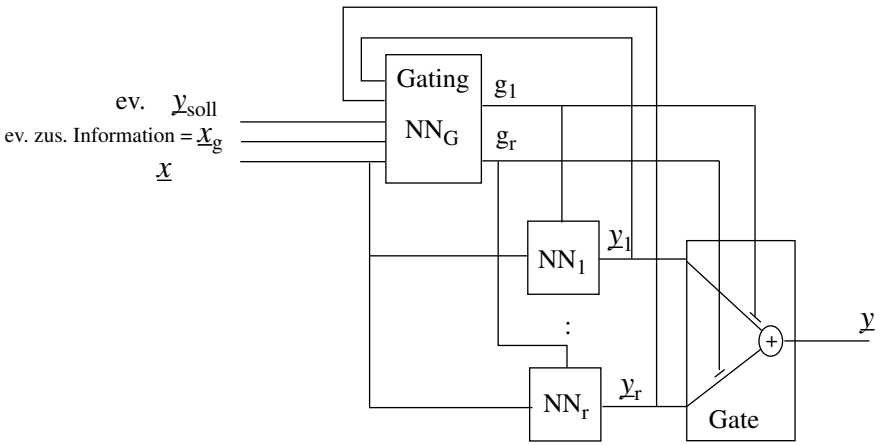


Bild 8.3 *Gating durch Wettstreit unter den Teilnetzen*

Bis heute wurden mehrere Lernalgorithmen zur selbstorganisierenden Entwicklung von Modularität durch Kompetition unter parallelen Teilnetzen vorgestellt. Darunter sind Gradient Descent-Verfahren [Jaco90][Jaco91][Nowl91] sowie ein Expectation Maximization-Algorithmus [Jord94]. Aufgrund der Wichtigkeit dieses Ansatzes wird einer der Gradient Descent-Algorithmen [Jaco91] für das Gating erläutert:

Das Gating-Netz ist ein Backpropagation-Netz ohne hidden Layer mit *n* Inputunits und *r* Outputunits, die Teil- (Expert-) netze sind beliebige neuronale Netze, im konkreten Fall lineare und nichtlineare Backpropagation-Netze Die Expertnetze realisieren jeweils einen Teil der gesamten Abbildung der Eingangsdaten $\underline{x} \in R^n$ auf die Ausgangsdaten $\underline{y} \in R^m$. Der Output $\underline{y} \in R^m$ der gesamten Architektur ergibt sich aus den Outputs $\underline{y}_i \in R^m$ der r Teilnetze und dem Output $\underline{g} \in R^r$ des Gating-Netzes durch gewichtete Summierung:

$$\underline{y} = \sum_{k=1}^{r} g_k \underline{y}_k \tag{8.4}$$

Für die Berechnung der Aktivierungen der Outputunits des Gating-Netzes wird die Softmax-Aktivierungsfunktion [Brid89] verwendet: Mit dem Gewicht v_{ik} für den i-ten Input der k-ten Outputunit des Gating-Netzes und s_k der gewichteten Summe aller Inputs x_i der k-ten Outputunit ist der Output g_k der k-ten Outputunit

$$g_k = \frac{e^{s_k}}{\sum_{j=1}^{r} e^{s_j}} \quad \text{mit} \quad s_k = \sum_{i=1}^{n} v_{ik} x_i \ . \tag{8.5}$$

Durch die Verwendung der Softmax-Aktivierungsfunktion wird erreicht, daß die Outputs g_k des Gating-Netzes positiv sind und in Summe eins ergeben. Es erübrigt sich eine explizite Normierung der g_k.

Zum besseren Verständnis der Bedeutung der Softmax-Aktivierungsfunktion für das Gating-Netz kann ein System von 2 Expertnetzen betrachtet werden. In diesem Fall hat das Gating-Netz zwei Outputs g_1 und g_2. Unter Verwendung der Aktivierungsfunktion aus Gl. 8.5 errechnet sich

$$g_1 = \frac{1}{1 + e^{(\underline{v}_2 - \underline{v}_1)^T \underline{x}}} \quad \text{und} \quad g_2 = 1 - g_1 \ . \tag{8.6}$$

Das Gating-Netz realisiert die “weiche” Partitionierung des Eingangsraumes. Die zwei Expertnetze stellen nach dem Training zwei sich überlappende (lokale) Hyperfächen im Inputraum dar. Für einen Input $\underline{x}$ ergibt sich der gesamte Output $\underline{y}$ aus $\underline{y} = g_1 \underline{y}_1 + g_2 \underline{y}_2$. Die Distanz zwischen den beiden Gewichtsvektoren $\underline{v}_1$ und $\underline{v}_2$ bestimmt, wie stark sich die Bereiche der beiden Expertnetze überlappen. Im Extremfall mit $\underline{v}_2 - \underline{v}_1 = \underline{0}$ ist der Gesamtoutput der Architektur für alle $\underline{x}$ aus dem gesamten Inputraum der Mittelwert aus beiden Teilergebnissen. Je größer die Differenz zwischen den beiden Gewichtsvektoren ist, desto “schärfer” ist die Partitionierung des Inputraumes (siehe Bild 8.4 für eine eindimensionale Eingangsgröße x ohne Berücksichtigung des Bias).

Das Training des Gating-Netzes und der Expertnetze erfolgt simultan, d.h. für jedes Trainingspattern wird zunächst ein Recallschritt an den Expertnetzen ausgeführt,

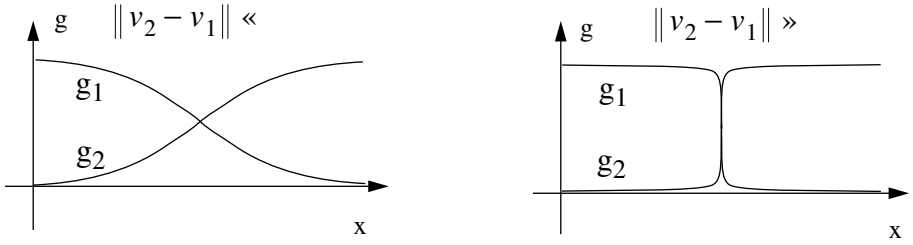


Bild 8.4 *Weiche und scharfe Partitionierung des Eingangsraumes abhänging von den Gewichten des Gating Netzes*

anschließend anhand der erzielten Ergebnisse das Gating-Netz trainiert und dann unter Verwendung der neuen Outputs des Gating-Netzes die Expertnetze trainiert.

Beim Training des Gating-Netzes wird mit dem Soll-Output $\underline{y}_{soll}$ und einem Skalierungsparameter des *k*-ten Teilnetzes σ_k^2 folgende "Qualitätsfunktion" durch Gradientenanstieg maximiert:

$$ln\ L = ln \sum_{k=1}^{r} g_k e^{-\frac{1}{2\sigma_k^2}\|y_{soll} - y_k\|^2} \tag{8.7}$$

Die konkrete Lernregel errechnet für die k-te Outputunit des Gating-Netzes die Gewichtsänderung aus der partiellen Ableitung der Qualitätsfunktion nach den Gewichten v_{ik} unter Verwendung der Kettenregel:

$$\frac{\partial e}{\partial v_{ik}} = \frac{\partial e}{\partial g_k}\frac{\partial g_k}{\partial s_k}\frac{\partial s_k}{\partial v_{ik}} \tag{8.8}$$

Mit der partiellen Ableitung

$$\frac{\partial}{\partial s_k} ln\ L = g_k \frac{e^{-\frac{1}{2\sigma_k^2}\|\underline{y}_{soll} - \underline{y}_k\|^2}}{\sum_{k=1}^{r} g_k e^{-\frac{1}{2\sigma_k^2}\|\underline{y}_{soll} - \underline{y}_k\|^2}} - g_k = g\,(k|\underline{x}, \underline{y}_{soll}) - g_k \tag{8.9}$$

und der Lernrate α des Gating-Netzes ergibt sich die Gewichtsänderung bei einem Trainingsschritt des Gating-Netzes zu

$$\Delta v_{ik} = \alpha\,(g\,(k|\underline{x}, \underline{y}_{soll}) - g_k)\,x_i. \tag{8.10}$$

$g\,(k|\underline{x}, \underline{y}_{soll})$ kann probabilistisch als a-posteriori Wahrscheinlichkeit, daß der Outputvektor $\underline{y}_k$ des k-ten Expertnetzes bei einem konkreten (gegebenen) Inputvektor $\underline{x}$ dem (gegebenen) Sollouputvektor $\underline{y}_{soll}$ entspricht, interpretiert werden.

Beim Training des k-ten Expertnetzes werden dessen Gewichte proportional zur partiellen Differentiation der Log-Likelihood Funktion L von Gl. 8.7 nach $\underline{y}_k$

$$\frac{\partial}{\partial \underline{y}_k} ln\ L = \frac{g\,(k|\underline{x}, \underline{y}_{soll})}{\sigma_k^2}\,(\underline{y}_{soll} - \underline{y}_k) \tag{8.11}$$

adaptiert. Dadurch wird die “Wahrscheinlichkeit”, daß das k-te Teilnetz für den Inputvektor $\underline{x}$ das richtige Ergebnis liefert beim Training des Teilnetzes berücksichtigt. Im Falle von Backpropagation als Lernverfahren für die Expertnetze ist Gl. 8.11 der (gewichtete) Fehler des Netzes. Im Falle von unsupervised lernenden Expertnetzen erfolgt eine Adaptierung proporitonal zum Gradienten aus Gl. 8.11[1].

Abschließend sei noch darauf hingewiesen, daß der Input für das Gating-Netz x_i gleich dem Input der Expertnetze sein kann, daß aber zusätzlich zum Input auch der zugehörige Output, sowie jede andere Datenquelle ($\underline{x}_g$ in Bild 8.3) als Input für das Gatingnetz verwendet werden kann. Inbesonders sind Informationen, die eine Art Taskindikator darstellen, als zusätzliche Information für das Gatingnetz geeignet.

8.3 Kombinationsmöglichkeiten

Im Prinzip können alle Arten von Selektion mit allen Prinzipien zur Integration kombiniert werden. Für jede Art der Integration können zusätzlich noch Performance-Schätzwerte der Teilnetze (siehe Kapitel 5) verwendet werden, um eine Verbesserung der Integration zu erreichen. Dabei sind einige Kombinationsmöglichkeiten jedoch nicht sinnvoll. So werden z.B. bei der probabilistischen Integration (siehe Abschnitt 6.1.3) Glaubwürdigkeitsaussagen für die Ergebnisse der Teilnetze aus den reellwertigen Outputvektoren der Teilnetze berechnet. Dies erübrigt eine Qualitätsabschätzung nach Gl. 5.1 (siehe Abschnitt 5.1.1).

Tabelle 8.1 gibt einen Überblick über die sinnvollen und nicht sinnvollen Kombinationsmöglichkeiten. Aufgrund der kleineren Fehler bei Mittelwertbildung im Vergleich zu Voting [Perr94] wird letzteres nur eingesetzt werden, wenn Module integriert werden sollen, die keinen reellwertigen Output liefern.

Zusätzlich zu den in Tabelle 8.1 dargestellten Kombinationen können jeweils für das Selektornetz und jedes Teilnetz ein eigenes adaptives Preprocessing, eine eigene Featureextraktion (z.B. durch Principal Component Analysis) und/oder eine Quantisierung der Features verwendet werden. Die “Maximalarchitektur” ist in Bild 8.3 dargestellt. Üblicherweise wird jedoch nur eines der vorverarbeitenden Netze je Teilnetz verwendet. Es ergibt sich eine Spezialisierung des Preprocessings auf den jeweiligen lokalen Eingangsbereich der Teilnetze.

1. Statt der Gewichtung des Fehlers kann auch die Lernrate mit $\frac{g\,(k|\underline{x}, \underline{y}_{soll})}{\sigma_k^2}$ gewichtet werden.

Selektion \ Integration		Voting					Mittelwert, Neuronale Fusion					Propabilistische Integration				
			Performance Kriterium					Performance Kriterium					Performance Kriterium			
			z1, z2	z	valid.	quali.		z1, z2	z	valid.	quali.		z1, z2	$z^{2)}$	valid.	quali.[2)]
Clustering	-	ok	ok	ok	-	ok	ok	ok	ok	-	ok	ok	-	ok	-	ok
	ed	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	-	ok	ok	ok
supervised	ed	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	-	ok	ok	ok
	init	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	-	ok	ok	ok
	cin	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	-	ok	ok	ok

Tabelle 8.1 *Kombinationen von Selektion und Integration*

	in diesem Kapitel für Mittelwertbildung behandelt
ed	Selektion gesteuert duch den Validation Error
init	Initalisierung des Gating-Netzes duch Clustering
cin	Aktivierungen der Units eines Clustering-Netzes als zusätzlicher Input für das Gating-Netz
z1, z1	Ergebniseindeutigkeit als Performancewerte zur Verbesserung der Integration
z	Input Reconstruction Reliability Estimation zur Verbesserung der Integration
quali.	Qualität des Modells zur Verbesserung der Integration
valid.	Validation während des Trainings zur Steuerung der Selektion
2)	Die Glaubwürdigkeit des Ergebnisses des Einzelnetzes wird vor der Berechnung der Gesamtglaubwürdigkeit gewichtet (siehe Kapitel 5)

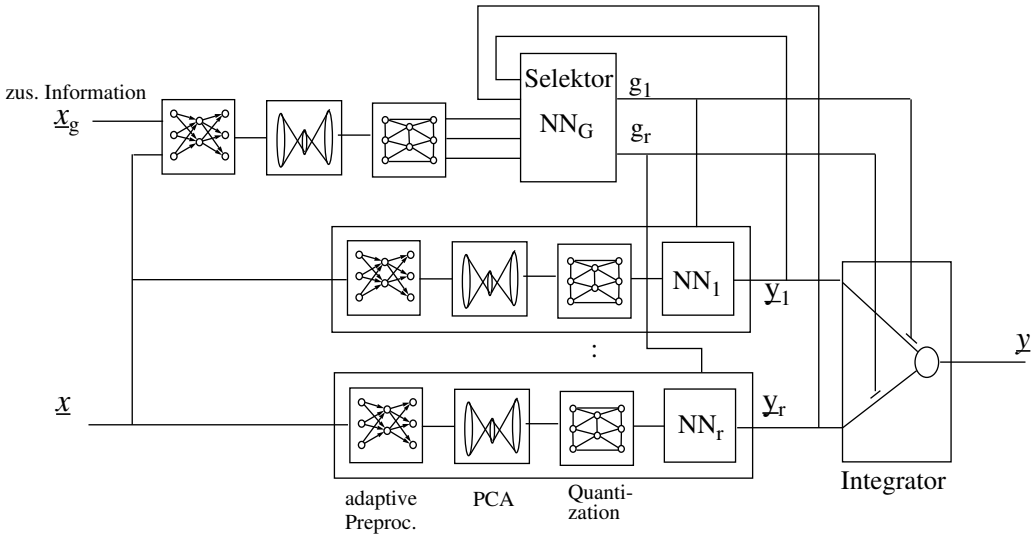
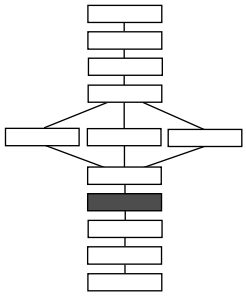


Bild 8.5 *Maximalausbau eines nicht hierarchischen Mixture Models*



9 Heuristiken zur manuellen Dekomposition

In den bisherigen Kapiteln wurden die Grundarchitekturen vorgestellt, aus denen sich alle modularen Systeme aufbauen lassen. Die Frage, die sich nun stellt, ist, für welche Aufgabe welche Art von Modularität geeignet ist. Anders formuliert heißt dies, wie kann ein komplexes Problem in Teilprobleme zerlegt werden, sodaß die Teilprobleme "einfacher" zu lösen sind, dabei aber zugleich das gesamte System schneller trainiert werden kann, bessere Performance aufweist und besser verständlich und zuverlässiger ist.

Eine allgemeine Lösung des Problems der optimalen Dekomposition bei neuronalen Systemen liegt noch in weiter Ferne. Es ist deshalb ein so schwieriges Problem, weil bei neuronalen Netzen die Aufgaben vielfach keine offensichtliche "natürliche Struktur" aufweisen (z.B. die Klassifikaiton von Buchstaben). Gerade aber aufgrund der Fähigkeit, komplexe, nicht explizit modellierbare und deshalb nicht strukturierbare Abbildungen zu approximieren, werden neuronale Netze eingesetzt. Außerdem können nie alle Vorteile von Modularität gleichzeitig zum Tragen kommen. Oft stehen verschiedene Anforderungen an das zu entwerfende System im Widerspruch zueinander. So ist es intuitiv klar, daß eine niedrige Fehlerrate und damit Redundanz im Konflikt zur Forderung nach niedrigen Rechenzeiten steht.

Dekomposition wird hier deshalb als ein Prozeß verstanden, der vom gegebenen Problem, den zur Verfügung stehenden Daten und Hilfsmitteln sowie von den zu erzielenden Ergebnissen abhängt. Bei der nun folgenden Beleuchtung verschiedener Ansätze wird Dekomposition aus zwei Blickrichtungen betrachtet:

Zunächst werden Überlegungen zur **Dekomposition der Aufgebe** (aufgabenorientierte Sicht)

- anhand der Struktur von Aufgabe und Daten und

- anhand der Art der Aufgabe

präsentiert, sowie auf die

- Dekomposition von unstrukturierten Aufgaben

eingegangen. Im Anschluß daran wird die **Dekomposition des Systems** (lösungsorientierte Sicht)

- anhand der Anforderungen an das System,
- durch Aufteilung des Trainingssets,
- durch Evaluierung der Performance des Systems und
- anhand existierender Lösungen der Aufgabe

dargestellt[1].

Die Dekomposition ist ein rekursiver Prozess. Das bedeutet, daß identifizierte und definierte Teilaufgaben und Module selbst wieder unterteilt werden können.

9.1 Dekomposition der Aufgabe

Dekomposition der Aufgabe ist die Zerlegung der Aufgabe in möglichst unabhängige Teilaufgaben. Die Suche nach Teilaufgaben ist ein weitgehend intuitiver Prozeß. Häufig wird dabei Vorwissen über die Aufgabe verwendet. Überlegungen hinsichtlich der Art der Lösung der Teilaufgabe spielen bei der Dekomposition der Aufgaben nicht die zentrale Rolle. Im Unterschied zu traditionellen Systemen kommt bei neuronalen Systemen im Rahmen der Zerlegung der Aufgabe auch den Ein- und Ausgangsdaten des Systems und damit auch deren “Struktur” und Informationsgehalt große Bedeutung zu.

9.1.1 Dekomposition anhand der Struktur der Aufgabe

Die Struktur der Aufgabe wird definiert durch die Menge der Teilaufgaben aus denen die Aufgabe besteht und durch die Abhängigkeiten zwischenen den Teilaufgaben. Die Abhängigkeiten zwischen den Teilaufgaben werden durch die Reihenfolge der Lösung der Teilaufgaben festgelegt. Zwei Teilaufgaben sind voneinander abhängig, wenn deren Ergebnisse von der Reihenfolge der Lösung der Teilaufgaben abhängt. Die Teilaufgaben und die Reihenfolge der Teilaufgaben beschreiben somit die Struktur der Aufgabe.

Im Rahmen der Dekomposition wird die Struktur der Aufgabe auf die Architektur des Systems abgebildet. Umgekehrt können auch die möglichen Architekturen eines Systems (siehe Kapitel 4) als Basis zur Suche nach Struktur in der Aufgabe herangezogen werden.

1. Die Betrachtung von Dekomposition aus aufgabenorientierter Sicht und aus lösungsorientierter Sicht spiegelt auch die Vorgangsweise beim Systementwurf wider. Dabei wird mit einer Analyse der Aufgabe begonnen und daraus konkrete Architekturen entworfen [Somm92].

Folgende Strukturen sind möglich:

- Sequentielle Teilaufgaben

  Sequentielle Teilaufgaben sind aufeinander aufbauende Teilaufgaben. Jede Aufgabe an ein neuronalses System hat die bereits in 4.1.3 erwähnte serielle Grundstruktur: Datenerfassung, Vorverarbeitung, Featureextraktion, Lösung der eigentlichen Aufgabe, Nachverarbeitung und Datenausgabe (Visualisierung, Speicherung) (Bild 9.1).

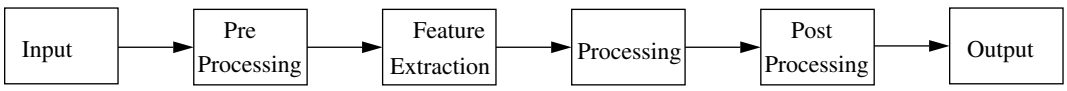


Bild 9.1 *Modulare Struktur jedes neuronalen Systems*

  In vielen Fällen läßt sich jedoch auch die eigentliche Aufgabe (Processing in Bild 9.1) in sequentielle Teilaufgaben zerlegen. Ein Beispiel dafür ist Objekterkennung bzw. Objektklassifizierung: Dabei wird zunächst die Position eines Objekts im Bild bestimmt und aufbauend darauf das Objekt klassifiziert (z.B. [Ahm91] [Pinz90]).

- Parallele Teilaufgaben

  Parallele Teilaufgaben sind unabhängige Teilaufgaben. Paralle Teilaufgaben können wahlweise oder gleichzeitig zu erfüllen sein . Im ersten Fall erfolgt eine Abbildung auf eine selektive Architektur (Abschnitt 4.1.1), im zweiten Falle eine Abbildung auf eine integrative Architektur (Abschnitt 4.1.2).

  Ein Beispiel für parallele Teilaufgaben, das besonders in der Bildverarbeitung Verwendung findet, ist die neuronale Featureextraktion [Vinc92],[Ahm91]. Dabei wird je ein Teilnetz pro zu erkennendem Feature (Kanten, Winkel, Texturen u.s.w.) verwendet.

- Hierarchische Teilaufgaben

  Hierarchien sind Systeme mit Über- und Unterordnungsbeziehungen zwischen den Elementen. Hierarchische Teilaufgaben entstehen, wenn Aufgaben Teil einer anderen Aufgabe sind. In einer Hierarchie von Aufgaben exisitiert genau eine Aufgabe (die hierarchisch höchste), die selbst nicht Teil einer anderen Aufgabe ist. Ist jede andere Teilaufgabe Teil von genau einer Aufgabe, so ist die Hierarchie **strikt**. Als verallgemeinerte Hierarchie von Aufgaben wird hier jede Struktur betrachtet, die sich aus “part of”- Beziehungen der (Teil-) Aufgaben ergibt, mit der Einschränkung, daß die Struktur auf einen zyklenfreien Graphen mit einem Wurzelknoten abbildbar ist.

  Hierarchische Teilaufgaben werden auf eine hierarchische Architektur (siehe Abschnitt 4.1.6) abgebildet. Strikt hierarchische Teilaufgaben sind eindeutig auf eine baumförmige Architektur abbildbar.

Ein Beispiel für eine hierarachische Aufgabe ist eine mehrstufige Klassifizierung. Dabei erfolgt zunächst eine grobe Unterteilung der Inputmuster in Hauptklassen und innerhalb jeder Hauptklasse eine Unterteilung in Unterklassen. In Bild 9.2 a ist eine merhstufige Klassifizierung für einen zweidimensionalen Inputraum veranschaulicht. Die dicken Linien symbolisieren die Grenzen zwischen den Hauptklassen. Die dünnen Linien (nur für eine Hauptklasse dargestellt) unterteilen die Hauptklassen in Unterklassen. Bei der jeweiligen Feinunterteilung können andere Features zum Tragen kommen als bei der Grobunterteilung.

In [Smit93] wird dies beispielsweise zur Erkennung von vier Alkoholgruppen und sechs Carbonylgruppen (Bild 9.2 b) aus den Infrarotspektren von chemischen Proben verwendet. Statt einem Netz mit 10 Outputs zur direkten Erkennung der vier Alkoholgruppen und der sechs Carbonyl-Gruppen werden drei Netzwerke, eines zur Unterscheidung von Alkohol von Carbonyl und je eines für die vier Alkoholgruppen und die sechs Carbonyl-Gruppen, verwendet. Das hierarchisch höhere Alkohol-Carbonyl-Netz entscheidet, ob das Alkohol- oder das Carbonyl-Teilnetz ausgewertet wird.

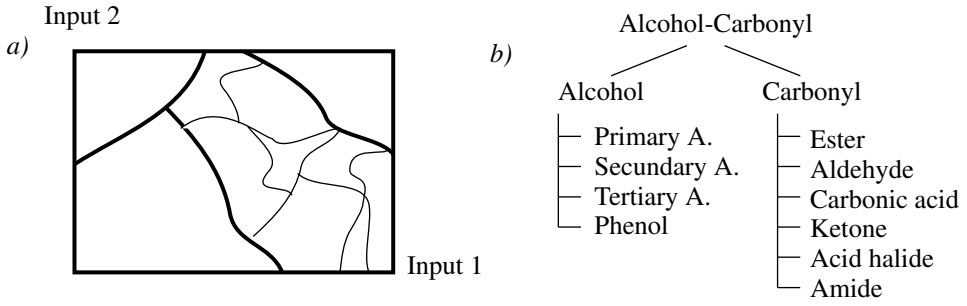


Bild 9.2 *Symbolische Darstellung einer zweistufigen Partitionierung eines zweidimensionalen Inputraumes (a) und der Klassen für eine zweistufige Klassifizierung aus [Smit93] (b)*

Ein anderes Beispiel für eine hierarchische Struktur in der Aufgabe ist eine hierarchische Robotersteuerung [Mart92]. Dabei ist es die hierarchisch höhere Aufgabe, mit dem Greifer eines Roboterarmes gesteuert durch zwei Kameras jede Position im dreidimensionalen Arbeitsraum des Roboters zu erreichen. In jeder Position des Armes erfolgt dann eine lokale Ausrichtung des Greifers je nach Lage des zu erfassenden Gegenstandes. Die Positionierung des Armes wird durch ein dreidimensionales Kohonen-Netz gelöst, die Ausrichtung des Greifers durch je ein zweidimensionales (lokales) Kohonen-Netz für jede Unit des hierarchisch höheren Netzes, d.h. für jede Position des Armes.

9.1.2 Dekomposition anhand der Struktur der Daten

Dabei erfolgt eine Gruppierung der Eingangsdaten und auch der Ausgangsdaten anhand räumlicher, zeitlicher und/oder konzeptueller (sinngemäßer) Zusammengehörigkeit und Signifikanz für die zu lösende Aufgabe. Je eine Gruppe von Daten wird dann entweder einem von *r* parallelen Teilnetzen zugeordnet oder, abhängig von der Signifikanz der Datengruppe, einer Stufe einer Kaskade von Teilnetzen.

Folgende Gruppierungen können unterschieden werden:

- Zeitliche Gruppierung

  Bei der zeitlichen Gruppierung werden unterschiedliche Datenvektoren, die jeweils alle Werte zu einem Zeitpunkt beinhalten, zu Gruppen zusammengefaßt und die Gruppen auf unterschiedliche Teilnetze aufgeteilt. Man kann zeitliche Fenster und zeitliche Spezialisierung unterscheiden (siehe Bild 9.3 a). Bei zeitlichen Fenstern werden die Daten aufgrund ihrer zeitlichen Nachbarschaft zusammengefaßt. Jedes Netz erhält dadurch einen eigenen zeitlichen Kontext. Die Fenster können dabei überlappend sein. Bei zeitlicher Spezialisierung werden regelmäßig n Vektoren einem Netz NN_1, die nächsten n Vektoren dem NN_2 usw. zugeordnet, bis nach r mal n Vektoren wieder n Vektoren dem NN_1 zugeordnet werden. D.h. die Daten werden nach Zeitpunkten (Bedeutung) gruppiert.

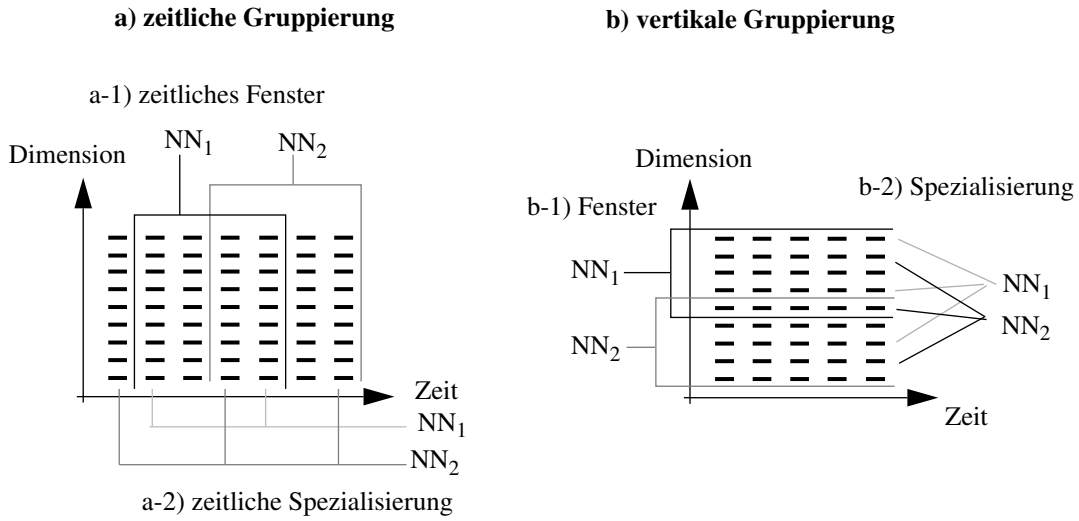


Bild 9.3 *Vertikale und horizontale Gruppierung von Datenvektoren*

  Die zeitliche Gruppierung entspricht einer “horizontalen” Struktur in den Daten. Sie ist meist bei der Analyse von Zeitreihen anzutreffen.

  Ein Beispiel für zeitliche Spezialisierung ist [Baum93]. Dabei wird bei der Prognose von elektrischen Lasten für jeden Wochentag ein eigenes neuronales Netz verwendet. Jedes dieser Einzelnetze prognostiziert aus dem Verlauf der Last des Tages t den Verlauf der Last des Tages t+1.

- “Vertikale” Gruppierung

  Bei der vertikalen Gruppierung werden verschiedene Komponenten eines Vektors aufgrund ihrer Bedeutung (Spezialisierung) oder aufgrund ihrer räumlichen Nachbarschaft (Fenster) zu Gruppen zusammengefaßt und verschiedenen Teilnetzen zugeordnet (siehe Bild 9.3 b). Die Bedeutung von Daten kann durch den physikalischen Prozeß, der die Daten erzeugt, gegeben sein (Daten von verschiedenen Teilen des physikalischen Systems bzw. verschiedene Meßgrößen) oder durch die Messung (verschiedene Meßreihen- und Meßkanäle, z.B. Multispektralbilder). Ein Beispiel ist die Fusion von Ultraschall- und Infrarotentfernungsmessung [Gosh90] in einem baumförmigen NN.

Eine Gruppierung von Komponenten jedes einzelnen Inputvektors kann aber auch durch die Art der Verarbeitung erforderlich werden. Ein Beispiel dafür ist die Verwendung lokaler Nachbarschaft in der Bildverarbeitung.

- hierarchische Gruppierung

  Bei der hierarchischen Gruppierung wird eine rekursive Struktur in den Daten gesucht und diese auf hierarchische Architekturen abgebildet. Diese rekursive Struktur in den Daten ergibt sich oft durch unterschiedliche Auflösungen derselben Information. Jedem Teilnetz werden die Daten einer Auflösung zugeordnet, sodaß die parallelen Teilnetze verschiedene “Horizonte” haben. Das Gesamtergebnis entsteht durch Überlagerung der Einzelergebnisse.

  Beispiele dazu sind die Prognose von Zeitreihen mit unterschiedlichen Horizonten (5h = kurzfristige Aktionen, 24 h, 96h, Trend) und Überlagerung der Prognosen bzw. die Objekterkennung in der Bildverarbeitung unter Verwendung von Bildpyramiden, wobei zwischen einer Erkennung der Grobform und der Feinform unterschieden wird. Ein konkretes Beispiel dafür ist [Vinc92]. Neben dem eigentlichen Grauwertbild wird ein durch Mittelwertbildung und Subsampling erzeugtes Bild mit einer um den Faktor 16 kleineren Auflösung verwendet, um Features aus dem Bild zu extrahieren. In der kleineren Auflösung werden durch je ein Multi Layer Perceptron pro Feature, sog. Candidate Features extrahiert. Das rezeptive Feld der NN wird dabei im Rahmen eines Scanvorganges über das Low Resolution-Bild geführt. Unter Verwendung von Zusatzinformation über die mögliche relative Lage von Features werden anschließend einige der Candidate Features ausgeschieden. Schließlich werden im hochauflösenden Bild die korrespondierenden Bereiche der verbleibenden Candidate Features durch eigene neuronale Featuredetektoren darauf hin untersucht, ob das jeweilige Feature auch tatsächlich vorhanden ist.

- Kombinationen

  Für das jeweils konkrete Problem können die Gruppierungen der Daten anhand der horizontalen und vertikalen Struktur miteinander kombiniert werden. Dadurch wird ein höherer Modularisierungsgrad erreicht und eine mögliche Abhängigkeit der Outputs von den Informationen aus verschiedenen Gruppen der Inputs berücksichtigt.

  In [Mavr92] wird beispielsweise für die Klassifikation der Ergebnisse einer Destillationsanlage die in Bild 9.4 schematisch dargestellte Gruppierung der Daten vorgeschlagen. Zur Beurteilung der Destillationsanlage werden in jeder Stufe der Anlage eine Reihe von physikalischen Größen gemessen. Die anfallenden Daten werden sowohl durch zeitliche (Zeitfenster) als auch durch konzeptuelle Gruppierung auf 186 Teilnetze aufgeteilt. Die Ergebnisse der Teilnetze werden hierarchisch zum Gesamtergebnis integriert.

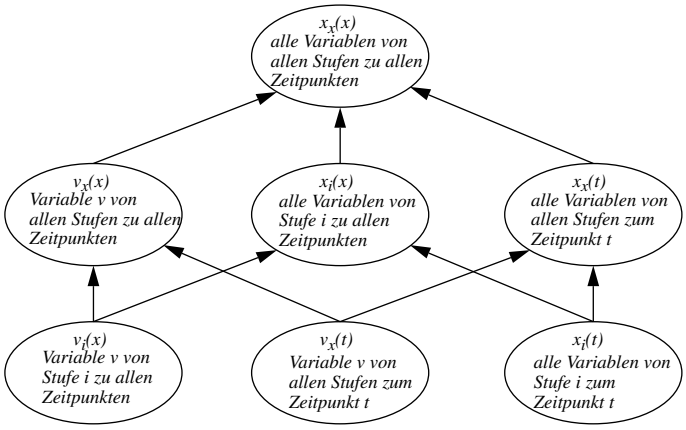


Bild 9.4 *Schematische Darstellung der Zuordnung von Datengruppen zu Teilnetzen aus [Mavr92]. Jede Ellipse symbolisiert ein oder mehrere Teilnetze.*

9.1.3 Dekomposition anhand der Art der Aufgabe

In einigen Fällen läßt sich die gesamte Aufgabe in Teilaufgaben zerlegen, wenn in der Gesamtaufgabe Teilaufgaben mit unterschiedlichen Eigenschaften gesucht werden. Soche Eigenschaften sind Eigenschaften der Abbildung der Inputdaten auf die Outputdaten oder Eigenschaften des Lernens (z.B. supervised-, unsupervised-, reinforcement-, local- und global learning)

Ein wichtiges Beispiel dazu ist die bereits genannte Zerlegung in einen linearen und einen nichtlinearen Teil. Bei der What and Where Vision Task [Jaco90] liegt solch ein Fall vor. Die Positionen (“Where Task”) von Objekten in einem Binärbild sind linear separierbar, nicht jedoch die “What Task”.
Falls diese Zerlegung in einen linearen und einen nichtlinearen Teil nicht apriori ersichtlich ist, ist eine automatische Dekomposition erforderlich (siehe “Mixture Models” ab Seite 53).

9.1.4 Dekomposition von unstrukturierten Aufgaben

Nach der Dekomposition der Aufgabe anhand deren Struktur und anhand der Struktur der Daten bleiben häufig große Teilaufgaben übrig, die sich nicht so offensichtlich weiter strukturieren lassen. Groß ist eine Teilaufgabe, wenn ein Netz zur Lösung dieser Teilaufgabe sehr viele Eingangs- / Ausgangsdaten (z.B. je über 100) und eine sehr große Zahl an Gewichten haben müßte, um die Aufgabe zu lernen. Ein solches Netz stellt entsprechend hohe Anforderungen an die Rechenzeit bzw. liefert keine ausreichende Fehlerrate.

Ein Beispiel dafür ist die Erkennung von chinesischen Schriftzeichen (z.B. [Iwat90]). In diesem Fall ist eine automatische Dekomposition der Aufgabe notwendig. Eine der in Abschnitt 8 (Mixture Models) dargestellten Architekturen kann verwendet werden, um die fast unbeschränkte Menge an zu erkennenden Zeichen automatisch in Teilmengen zu zerlegen (siehe auch Abschnitt 11.2).

Speziell für Klassifikationsaufgaben kann eine automatische Dekomposition in einen Klassifikationsbaum auch selbstorganisierend erfolgen. In [Guo92] werden z.B. mit Backpropagation trainierte binäre Klassifikationsbäume selbstorganisierend entwikkelt. In jedem Knoten des Baumes wird eine Entscheidung über die Zugehörigkeit des Inputmusters zu einer von zwei Unterklassen getroffen. Interessant dabei ist, daß in jedem dieser Knoten ein eigenes Backpropagation-Netz zur individuellen Feature-Extraktion eingesetzt wird.

9.2 Dekomposition des Systems

Dekomposition des Systems ist die Zerlegung des Systems in Teilsysteme (Module), d.h. die Modularisierung von Netzwerken und Algorithmen. Bei der Modularisierung des Systems steht die Performance des Systems, sowie Gesichtspunkte der Entwicklungsmethodik wie Überschaubarkeit, Verständlichkeit, Nachvollziehbarkeit, Testbarkeit und Wartbarkeit im Vordergrund. Die Dekomposition des Systems ist ein methodischer Prozeß der schrittweisen Verfeinerung der Architektur des Systems.

9.2.1 Dekomposition aufgrund der Anforderungen an das System

Die Anforderungen an das System können direkt verwendet werden, um die den Anforderungen am besten entsprechende Architektur auszuwählen. Speziell Aufgaben ohne erkennbare Struktur können so durch modulare Architekturen gelöst werden. Nachdem die Vorteile der einzelnen Architekturen bereits in Kapitel 4 erläutert wurden, wird hier nur noch eine Crossreferenz gegeben. Tabelle 9.1 gibt diesen Überblick über die für die verschiedenen Anforderungen passenden Architekturen.

Anforderung an das System	**Architektur**
minimale Fehlerrate	Integration von Teilnetzen (Diversität)
maximale Robustheit und Zuverlässigkeit	Integration von Teilnetzen (Redundanz). I.a. reichen 3 parallele Teilnetze aus [Batt94].
minimale Rechenzeit	Selektion von Teilnetzen
beste Erweiterbarkeit	Kaskadierung oder Selektion von Teilnetzen
Durchschaubarkeit, Aufteilbarkeit der Entwicklung, Wiederverwendbarkeit, Wartbarkeit	Serienschaltung, fix eingestellte Selektion von Teilnetzen (Aufgabenteilung)
hohe Konvergenzgeschwindigkeit	Selektion von Teilnetzen, Mixture Models mit linearen Teilnetzen
inkrementelles Lernen	dynamische Architekturen
geringer Speicherbedarf	modellbasierte Netze
Einfachheit der Parametrisierung	modellbasierte Netze, selbstorganisierende Selektion von Teilnetzen gleicher Größe

Tabelle 9.1 *Überblick über Performanceanforderungen und dafür geeignete Architekturen*

9.2.2 Dekomposition durch Aufteilung des Trainingssets

Die Idee hinter der Dekomposition durch Aufteilung des Trainingssets besteht darin, mehrere Teilnetze jeweils auf einen anderen Teil der gesamten Trainingsbeispiele zu

trainieren. So entstehen redundante Teilnetze, deren Ergebnisse z.B. durch Mittelwertbildung zum Gesamtergebnis integriert werden können.

Die einfachste Dekomposition ergibt sich aus der Aufteilung in Trainings- und Testset. Hier können die für das Training zur Verfügung stehenden Daten z.B. je zur Hälfte für Training und Test des einzelnen neuronalen Netzes verwendet werden. Auf diese Weise können zwei Netze trainiert werden, wobei das Trainingsset des einen Netzes dem Testset des anderen Netzes entspricht (siehe Bild 9.5). Durch die Verwendung von zwei (oder mehreren) Netzen erreicht man, daß insgesamt mit allen zur Verfügung stehenden Daten trainiert werden kann, aber trozdem das Training zur Vermeidung von Overfitting (siehe Bild 5.2) jeweils zum Zeitpunkt der besten Performance abgebrochen werden kann.

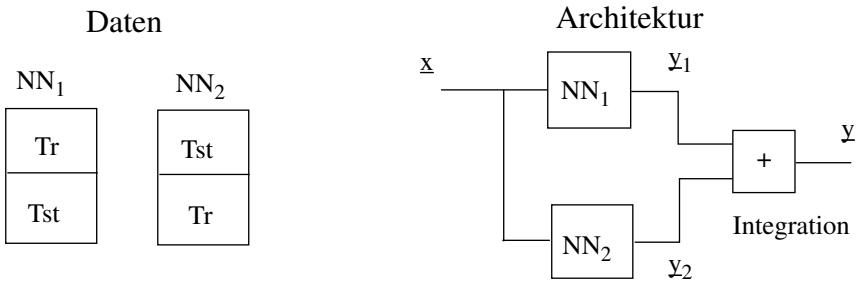


Bild 9.5 *Dekomposition durch unterschiedliche Aufteilung in Trainings- und Testset*

Eine aufwendigere Möglichkeit der Dekomposition durch Aufteilung des Trainingssets ergibt sich durch das Training eines zusätzlichen Netzes speziell auf jene Teile der Trainingsdaten, die von einem ersten Netz nicht zufriedenstellend gelöst werden können. Diese Methode wurde bekannt und speziell für Klassifikationsaufgaben untersucht unter dem Namen Boosting [Druc93]. Beim Boosting werden die Trainingsbeispiele auf insgesamt drei Netze aufgeteilt (Bild 9.6). Zu diesem Zweck wird zunächst ein erstes Netz mit einem Teil des Trainingssets trainiert. Es sei dabei zunächst angenommen, daß das Trainingsset beliebig groß ist, sodaß für jedes Teilnetz eine hinreichende Anzahl an Trainingsbeispielen zur Verfügung steht.

Das trainierte erste Teilnetz wird dann verwendet, um das Trainingsset für das zweite Netz zu erstellen. Dabei werden Trainigsbeispiele, die nicht Teil des Trainingsets des ersten Netzes sind, durch dieses erste Netz propagiert und je zur Hälfte Beispiele, die falsch und Beispiele die richtig klassifiziert werden, dem neuen Trainingsset hinzugefügt.

Mit dem so erzeugten Trainingsset wird dann das zweite Netz trainiert und anschließend beide trainierten Netze verwendet, um ein drittes Tainingsset zu generieren. Zu diesem Zweck werden wieder neue Traingsbeispiele durch beide Netze propagiert. Falls die Aussagen der beiden Netze unterschiedlich sind, wird das jeweilige Beispiel in das Trainingsset für das dritte Teilnetz aufgenommen.

Für den eigentlichen Recallbetrieb der gesamten Architektur werden dann alle drei Teilnetze parallel betrieben und die Einzelergebnisse nach einem der in Kapitel 6 genannten Möglichkeiten zu einem besseren Gesamtergebnis integriert.

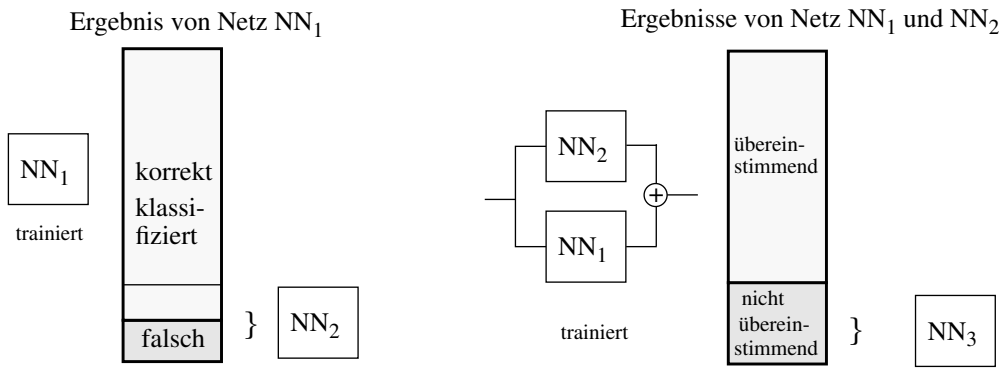


Bild 9.6 *Boosting: Auswahl von Trainingsbeispielen für Teilnetze anhand bereits trainierter Netze.*

Nun noch ein kurzer Blick auf die Größe des Trainingssets. Nachdem bereits das erste Netz eine möglichst geringe Fehlklassifikationsrate aufweisen soll und das zweite Trainingsset zu 50% aus Beispielen besteht, die das erste Netz nicht richtig lösen konnte, wird für die Erzeugung des zweiten (und dritten) Trainingssets insgesamt eine sehr große Zahl an Beispielen benötigt. (Jedes Trainingsset muß soviele Beispiele enthalten, bis durch zusätzliche Beispiele keine Performanceverbesserung mehr auftritt). Bei den meisten realen Anwendungen steht keine so große Zahl an Beispielen zur Verfügung. Deshalb müssen neue Trainingsbeispiele aus den vorhandenen Beispielen durch Transformation (geringfügige Translation, Rotation, Skalierung, u.s.w.) der Muster generiert werden. Auf diese Weise kann einerseits das vorhandene Trainingsset für Klassifikationsaufgaben nahezu beliebig vergrößert werden. Andererseits können durch geeignete Wahl der Transformationen auch genau jene Grenzbereiche besonders genau trainiert werden, die bei den tatsächlich vorhandenen Daten besonders schwierig zu klassifizieren sind.

9.2.3 Dekomposition durch Performanceanalyse

Hier wird die Performance eines bereits grob funktionsfähigen Systems gemessen und aus den Ergebnissen auf mögliche Verbesserungen geschlossen. Die Verbesserungen können durch Änderung der Architektur (z.B. durch Einführung von Redundanz bei zu großer Fehlerrate, siehe Tabelle 9.1) und Optimierung der Einzelnetze erreicht werden. Basis dafür sind Simulationsläufe und Auswertung der Performancekriterien (z.B. Fehlerrate, Konvergenzgeschwindigkeit, Robustheit, Speicherbedarf) wie auch Simulationsläufe mit künstlich veränderten Daten. Für einige Kriterien wird im folgenden dargestellt, wie solche Verbesserungen erfolgen können.

- Analyse des Restfehlers

  Durch Messung und Interpretation der Fehler des neuronalen Systems bzw. durch Ermittlung von Abhängigkeiten zwischen Fehlern und zusätzlichen Features kann in vielen Fällen auf Unzulänglichkeiten in der Architektur rückgeschlossen werden. Beispiele dafür sind:

  Bei Funktionsapproximation kann ein Histogramm des Fehlers über den zu approximierenden Bereich der Funktion gebildet werden. Zeigt sich, daß in gewissen Bereichen im Durchschnitt größere Fehler auftreten, so sind zusätzliche Features einzuführen. Gleiches gilt, wenn "regelmäßig wiederkehrende Muster" in den Fehlern von Funktionen auftreten. In diesen Fällen kann durch die Einführung einer Kaskadierung, der approximierte Funktionswert anhand zusätzlicher Features korrigiert werden.

  Zusätzliche Features lassen sich besonders leicht dann finden, wenn den Fehlermustern Bedeutungen zugeordnet werden können. Bei der bereits zitierten Lastprognose [Baum93] hat sich beispielsweise durch Fehleranalyse gezeigt, daß ein regelmäßiger "Peak" im Fehler des prognostizierten Lastverlaufs eines Tages zum Zeitpunkt des Sonnenaufgangs auftritt, d.h. der Fehler ist mit diesem Zeitpunkt korreliert. Das zusätzliche Feature für eine Korrekturstufe ist dann dieser Zeitpunkt. Wenn sich keine expliziten Abhängigkeiten der Fehler von nicht berücksichtigten Features feststellen lassen (z.B. bei nichtlinearen Abhängigkeiten), so kann eine Kaskadierung verwendet werden, um experimentell auf solche Abhängigkeiten zu prüfen.

  Bei Klassifikationsaufgaben kann für jede Klasse die Verteilung der Fehlklassifikationen (Confusion Matrix) erstellt werden. Zeigt sich, daß Beispiele einer Klasse überproportional häufig fäschlicherweise einer bestimmten anderen Klasse zugeordnet werden, so sind zusätzliche Features zur Separierung der zwei Klassen einzuführen. Diese sind nur jenen Teilnetzen zuzuorden, die bei der Unterscheidung der beiden Klassen beteiligt sind.

  Bei Selektion von Teilnetzen kann es vorkommen, daß einzelne Netze konstant schlechtere Ergebnisse liefern, als die Mehrheit der Teilnetze. In diesem Fall kann durch detailliertere Spezialisierung (Teilung des Teilnetzes) oder aber durch Integration von mehreren Teilnetzen die Performance verbessert werden.

- Sensitivitätsanalyse, Inputpruning

  Bei der Sensitivitätsanalyse wird die Kreuzkorrelation der Ausgangsdaten eines Netzes zu einzelnen Inputgrößen dieses Netzes bestimmt. Zu diesem Zweck werden einzelne Eingangsgrößen des trainierten Netzes variiert und die Veränderungen der Outputs beobachtet. Gibt es Inputs, die sich kaum auf Outputs auswirken, so ist dies ein Indiz dafür, daß dieser Input nicht relevant ist.
  Eine andere Möglichkeit ist die vom Pruning [Reed93],[Herg92] bekannte inkrementelle Messung der Relevanz von Inputs durch Schätzung der Sensitivität des Systemfehlers auf Veränderungen der Inputs.

  Bei statischen modularen Architekturen wird die Sensitivitätsanalyse für jedes Teilnetz durchgeführt. Dadurch werden einerseits die Teilnetze kleiner und andererseits die Aufteilung des Eingangsraumes auf die Teilnetze verstärkt. Ziel ist es, daß jedes Teilnetz nur noch jene Eingangsdaten erhält, die es für die Lösung der Teilaufgabe benötigt.

9.2.4 Dekomposition unter Verwendung vorhandener Lösungen

In vielen Fällen existieren bereits technische, nichtneuronale Lösungen bzw. biologische Vorbilder für die Lösung einer gestellten Aufgabe. Diese können als Vorwissen beim Design von modularen Architekturen dienen. Die Architektur von existierenden Lösungen kann als Ausgangspunkt für das Design der Architektur des neuronalen Systems verwendet werden. Das konkrete Verhalten einer exisiterenden Lösung kann (wie in Kapitel 10 ausgeführt werden wird) auf das initiale Verhalten (das ist das Verhalten des neuronalen Systems nach der Initialisierung und vor dem Training) des neruonalen Systems abgebildet werden. Dadurch kann die Trainingszeit um Ordnungen reduziert werden (z.B. durch modellbasiertes Lernen, siehe Abschnitt 10.3) und die Performance des Systems verbessert werden. Durch die Umsetzung von traditionellen Modulen im Neurosystem in neuronale Netze können zusätzlich auch Fehler über ursprünglich nicht adaptive Module propagiert werden, bzw. ursprüglich nicht lernfähige Module an die spezifischen Anforderungen der jeweiligen Anwendung angepaßt werden.

Die Vorgehensweise bei der Verwendung traditioneller technischer Lösungen für die Strukturierung und Initialisierung von neuronalen Systemen ist im Prinzip immer dieselbe (siehe Bild 9.7): Ausgehend vom nichtneuronalen System wird dieses auf ein oder mehrere neuronale Netze abgebildet. Die Architektur des traditionellen Systems wird dabei möglichst unverändert auf die Large Scale-Architektur des Neurosystems abgebildet. In der vorhandenen Lösung verwendete Relationen und Abhängigkeiten werden auf die Small Scale-Architektur des einzelnen NN abgebildet. Das gefundene initiale Neurosystem wird dann trainiert. Das trainierte Neurosystem kann dann, sofern möglich, in das traditionelle System zurückübersetzt werden. Dadurch können die Veränderungen der traditionellen Lösung durch die Adaption des NN interpretiert werden.

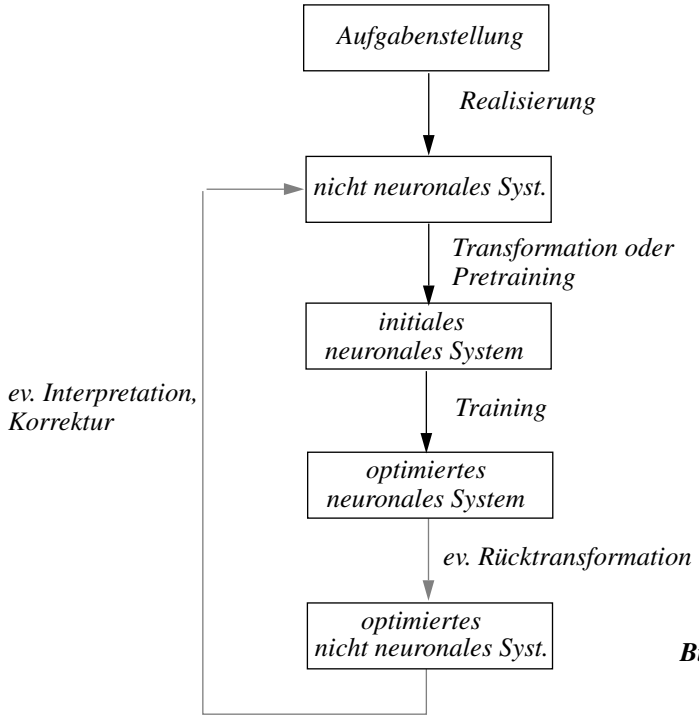
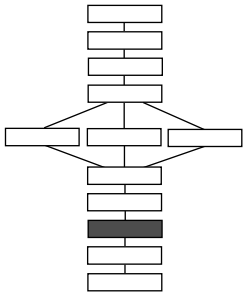


Bild 9.7 *Prozeß der Abbildung von traditionellen Systemen in neuronale Systeme*

Es sind eine Reihe von Möglichkeiten zur Strukturierung von neuronalen Netzen anhand vorhandener Lösungen bekannt. Darunter ist die Abbildung von regelbasiertem Wissen auf neuronale Netze [Louk94], sowie die Transformation von mathematischen und statistischen Modellen auf die Netzarchitektur (siehe Kapitel 10).



10 Modell- und algorithmusbasierte Strukturierung neuronaler Netze

In Kapitel 9 wurde auf die Bedeutung von Dekomposition unter Verwendung vorhandener, traditioneller Lösungen hingewiesen. In diesem Kapitel erfolgt nun eine Darstellung, wie Algorithmen und mathematische Modelle verwendet werden können, um neuronale Netze zu strukturieren.

Die Abbildung eines mathematischen Modells oder Algorithmus auf ein neuronales Netz erfolgt entweder durch Transformation des Modells in ein Spezialnetz, das dann bereits untrainiert die gleiche Funktionalität hat wie das Modell, oder durch Training eines neuronalen Netzes auf das Verhalten des Modells. Die Transformation des Modells in ein Netz führt zu einer modellbasierten Architektur des neuronalen Netzes. Ein Spezialfall der modellbasierten Architektur ist modellbasiertes Lernen. Dabei werden beim Training nicht die Gewichte des NN adaptiert sondern die Parameter des Modells.

Die Vorteile, die sich aus der Transformation von Algorithmen und Modellen in neuronale Netze ergeben, sind vielfältig:

- einheitliches Schema: Die Adaptierung kann auch über bislang nicht adaptive Module im neuronalen System erfolgen.
- Unvollständige oder teilweise fehlerhafte Algorithmen bzw. Modelle werden anhand der Trainingsbeispiele erweitert bzw. korrigiert.
- Durch die Integration der im Modell steckenden Information in das Netz werden wesentlich kürzere Trainingszeiten und kleinere Fehlerraten erreicht (siehe Kapitel 3)

10.1 Fallbeispiel

Bevor die Abbildung eines Algorithmus oder Modells auf ein neuronales Netz detaillierter behandelt wird, erfolgt hier zunächst die Präsentation eines Beispiels aus der Verfahrenstechnik, das den Prozeß der modellbasierten Dekomposition illustriert.

In einem Walzwerk werden verschiedene Bleche (unterschiedliche Dicken, Materialien, usw.) in beliebiger Reihenfolge produziert. Die glühenden Rohbleche werden dazu in mehreren Stufen gewalzt. Um die gewünschte Dicke zu erzielen, regelt ein konventioneller Regelkreis (PID) den Anpreßdruck der Walzen. Aufgrund von Totzeiten kommt es am Beginn eines Walzvorganges zu Einschwingvorgängen und damit zu erheblichen Ausschüssen. Die Ausschüsse werden reduziert durch eine Vorsteuerung der Walzkräfte (d.h. Voreinstellung der Walzkräfte vor dem Beginn der Produktion). Für diese Vorsteuerung wird der benötigte Walzdruck mittels einem Modell der Walzen und Bleche geschätzt. Für die richtige Schätzung des zu verwendenden Walzdrucks sind allein zur Erreichung der Profilgenauigkeit für die Modellierung der Walzen die gleichzeitige Berücksichtigung von einem Biegemodell, einem Temperaturmodell und einem Verschleißmodell notwendig (Bild 10.1). Alle drei Walzenmodelle werden aufgrund von mechanischen Parametern mathematisch formuliert. Es handelt sich dabei um einfache Modelle, in denen keine gegenseitige Beeinflussung (z.B. eine Walze mit höherer Temperatur biegt sich leichter als eine mit niedriger Temperatur) berücksichtigt sind. Die Ergebnisse der drei Walzenmodelle müssen in einer intergrierenden Stufe zum Gesamtmodell integriert werden. In Bild 10.1 ist die Architektur des Gesamtmodells dargestellt. In der integrierenden Stufe können noch "Dreckeffekte" berücksichtigt werden.

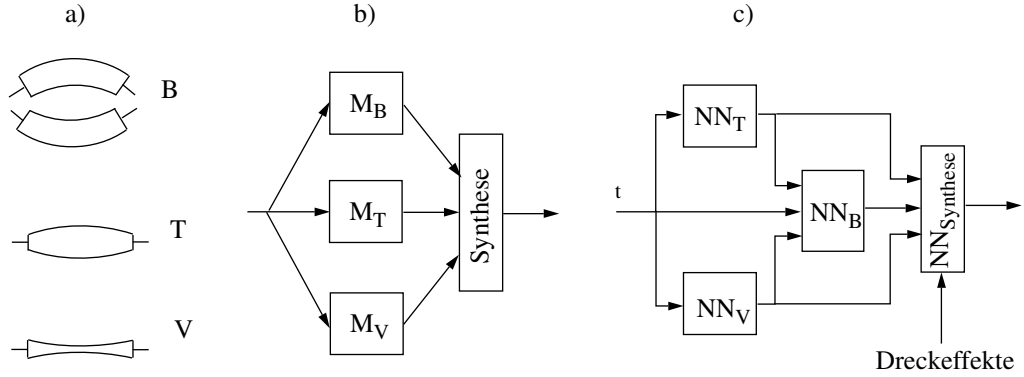


Bild 10.1 *Adaptive Modellierung der Walzen eines Walzwerkes zur Vorsteuerung der Walzkräfte*
a) Schematische Darstellung von Biege-, Temperatur- und Verschleißmodell
b) Darstellung der Architektur des Modells
c) Darstellung der Architektur des neuronalen Netzes

Sowohl die gegenseitigen Beinflussungen der Walzen wie auch die Dreckeffekte sind nur sehr aufwändig mathematisch zu formulieren. Deshalb wird eine neue Architektur entworfen (Bild 10.1 - c), in der diese Beeinflussungen berücksichtigt werden. Alle einzelnen Module werden durch NN realisiert. Die mathematischen Formulierungen der Modelle werden jeweils in ein eigenes NN transformiert. Während des Betriebes des Walzwerkes werden anstelle der starren mathematischen Modelle die neuronalen

Modelle verwendet. Durch die Adaptivität der neuronalen Modelle werden online, d.h. während des Betriebes, die nicht modellierten Störeffekte und die gegenseitigen Beeinflussungen der Teilmodelle erlernt. Außerdem bleibt das Gesamtmodell adaptiv, wodurch neue Materialeigenschaften und Ungenauigkeiten in den Modellen online ausgeglichen werden können.

Die Struktur der mathematischen Modelle und deren Verhalten stellen das Vorwissen für das neuronale System dar. Würde dieses Vorwissen nicht offline trainiert, so müßten online nicht nur der Einfluß von Störgrößen und der Modellabhängigkeiten gelernt werden, sondern das Modell als ganzes. Dies würde dazu führen, daß die Vorsteuerung erst nach einer großen Zahl an Serien wirksam wird und wahrscheinlich nie die Qualität der vorstrukturierten Lösung erreicht.

10.2 Modellbasierte Architektur

Eine modellbasierte Architektur entsteht durch die analytische Abbildung eines Modells auf die Architektur eines neuronalen Netzes. Dazu werden die im Modell verwendeten Operationen bzw. Algorithmen so umgeformt, daß das Modell nur noch aus Operationen besteht, die den Operationen von Units eines Backpropagation-Netzes beim Recall entsprechen. Anders formuliert bedeutet dies, daß bei der Transformation von Algorithmen spezifische neuronale Netze konstruiert werden, die aus speziellen Units (addierend, multiplizierend, logarithmierend usw.) mit “händisch” voreingestellten Gewichten bestehen. Die Verbindungsstruktur wird je nach Erfordernis im Modell individuell gestaltet. Das jeweils konstruierte Netz bekommt dadurch bereits vor dem Training die gleiche Funktionalität wie das mathematische Modell. Das Training verfeinert die Einstellung der Gewichte und vervollständigt bzw. korrigiert auf diese Weise die Funktionalität des Modells.

Für konkrete Beispiele veranschauliche man sich eine Unit eines Backpropagation-Netzes (Bild 10.2 und Abschnitt 2.1).

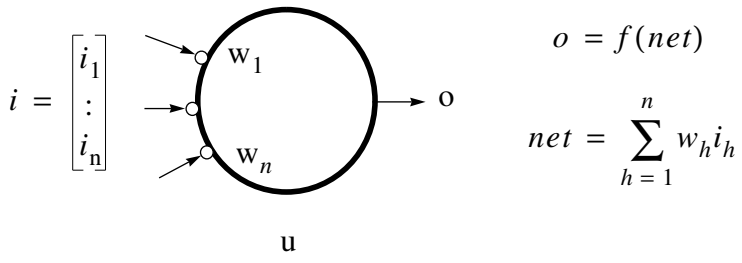


Bild 10.2 *Unit eines Standard Backpropagation-Netzes*

Jede Unit des Netzes bildet beim Recall aus ihren Eingangswerten i_h zunächst einen Nettoinput *net* und errechnet daraus unter Verwendung einer Propagate-Funktion *f* den Outputwert *o*. Die im Standard Backpropagation-Netz verwendete Unit berechnet ihren Nettoinput *net* durch die gewichtete Summierung aller ihrer Inputs (Inneres Produkt des Gewichtsvektors und des Inputvektors) und verwendet für die Propagate-Funktion *f* eine sigmoide Funktion, z.B.:

$$o = f(net) = \frac{1}{1 + e^{-net}}. \tag{10.1}$$

Die Transformation einer Multiplikation einer m x n Matrix mit einer n x p Parametermatrix wird z.B. durch m x p Standardunits realisiert. Jede Unit hat in diesem Fall n Gewichte (die n Parameter jeweils einer Spalte der Parametermatrix), die beim Training adaptiert werden.

Für andere Operationen sind Erweiterungen der Standardunit notwendig: So kann die Propagatefunktion *f* durch jede andere Funktion ersetzt werden. Diese Funktion muß, falls mit Gradientenabstieg gelernt werden soll, stetig differenzierbar sein. Für die Lernregel

$$\Delta w_h \sim -\frac{\partial e}{\partial w_h} = \frac{\partial}{\partial w_h} net \; f'(net) \frac{de}{do} \tag{10.2}$$

mit dem quadratischen Fehler *e* der Unit und

$$\frac{\partial}{\partial w_h} net = i_h \tag{10.3}$$

und

$$\frac{de}{do} = 2\,(o_{soll} - o) \qquad \text{für eine Outputunit} \tag{10.4}$$

für den Standardfall muß dann f' ersetzt werden.

Statt der gewichteten Summe zur Berechnung des Nettoinputs kann das gewichtete Produkt verwendet werden. Verallgemeinert erhält man Sigma-Pi Units [Rume86], deren Nettoinput sich errechnet nach

$$net = \sum_i w_i x_i + \sum_{i,j} w_{ij} x_i x_j + \ldots = \sum_i w_{ij} \prod_{k=j}^{i} x_k \tag{10.5}$$

Schwieriger wird die Abbildung von Iterationen und Verzweigungen auf ein Backpropagation-Netz. Dabei muß ein spezielles Teilnetz aus mehreren Spezialunits gebildet werden.

Es exisitieren eine Reihe von Beispielen für die Transformation von Modellen und Algorithmen in ein NN. Ein neuronales Netz, das mit den standard Backpropagation Units bzw. Schwellwertunits auskommt, ergibt sich z.B. bei Transformation aller linearen Filter der Bildverarbeitung (Hoch-, Tief- und Bandpaß, Kantendedektoren u.s.w.) und morphologischen Operationen auf ein NN (z.B. [Roha92]). Dabei wird für jedes Pixel des Ergebnisbildes eine Unit mit n mal n Inputs (das jeweilige lokale Fenster) verwendet und die Gewichte entsprechend des Filterkernes voreingestellt. Der Vorteil solch eines Filters liegt darin, daß sein Filterkern abhängig von den Ergebnissen der

nachgeschalteten Stufen im Gesamtsystem adaptiert werden kann und daß sich dadurch eine positionsabhängige Veränderung ergeben kann.

Ein weiteres Beispiel für Vorstrukturierung von standard Backpropagation-Netzen ist [Zalz91]. Dabei wird die numerische Lösung der Differentialgleichungen eines einfachen kinematischen Modells eines Roboters durch Matrixmultiplikationen dargestellt und auf diese Weise in das Backpropagation NN transformiert.

Ein Beispiel für die Verwendung von Spezialunits ist die Bestimmung der Position der letzten Komponente eines Vektors, deren Wert größer als ein Schwellwert ist. Dies kann z.B. durch das in Bild 10.3 dargestellte Teilnetz erreicht werden. Durch die Schwellwertschicht werden zunächst all jene Komponenten, deren Werte über dem gewünschten Schwellwert liegen, auf eins abgebildet. Die Outpunit ist eine logarithmierende Unit, deren Gewichte auf Potenzen von *e* voreingestellt sind.

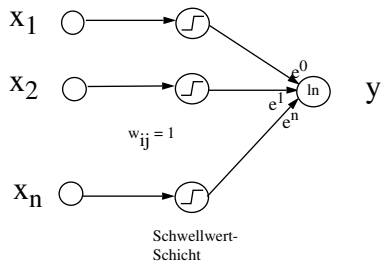


Bild 10.3 *Neuronales Netz zur Erkennung von Objektgrenzen*

Weitere Beispiele sind die Konstruktion von Neural Curve Pyramids in [Bisc93] sowie spezielle Preprozessing-Operatoren wie Aspect Ratio und Pixel Counts, z.B. bei der Erkennung von handgeschriebenen Zahlen (siehe Kapitel 11).

10.3 Modellbasiertes Lernen

Bei modellbasiertem Lernen wird wie bei den modellbasierten Architekturen ein mathematisches Modell auf ein händisch konstruiertes Spezialnetz abgebildet. Die (Vor-) Einstellung der Gewichte erfolgt hier jedoch durch eine parametrisch definierte Funktion $w_{ij} = h(a_{ij},d_{ij})$ mit a_{ij} einem individuellen Parameter für jede Verbindung im Netz und d_{ij} einer räumlichen Distanz zwischen i-ter und j-ter Unit des Netzes. D.h. die Gewichte des Netzes werden parametrisch modelliert. Die räumliche Distanz zwischen zwei Units ergibt sich aus der Definition einer Nachbarschaft. Z.B. werden die Units für die Verarbeitung von Bildern als in Matrixform auf einem zweidimensionalen Gitter angeordnet betrachtet (4- Nachbarschaft).

Beim Training des Netzes werden nicht die Gewichte sondern die Parameter der Funktion *h* optimiert. Mit dem Fehler *e* des Netzes ergibt sich bei Backpropagation die Änderung des Parameters a_{ij} proportional zu

$$\frac{\partial e}{\partial a_{ij}} = \frac{\partial e}{\partial w_{ij}} \frac{\partial w_{ij}}{\partial a_{ij}} = \frac{\partial e}{\partial h} \frac{\partial h}{\partial a_{ij}} \tag{10.6}$$

Dies bringt neben den Vorteilen der modellbasierten Architektur den Vorteil, daß die erlernten Werte der Parameter sofort interpretiert werden können. Für viele auf Netze abgebildete Modelle ist es zusätzlich möglich, nicht einen eigenen Parameter a_{ij} für jedes Gewicht, sondern nur für jede Unit des Layers ($w_{ij} = h(a_j, d_{ij})$) zu verwenden, oder überhaupt nur einen Parameter a für einen ganzen Layer ($w_{ij} = h(a,d_{ij})$). In diesem Fall ergibt sich durch die Reduzierung der Zahl freier Parameter eine wesentliche Steigerung der Trainingsgeschwindigkeit.

Ein einfaches Beispiel zu modellbasiertem Lernen ist die lineare, adaptive Filterung [Cael93]. Der jeweilige Filterkern wird parametrisch beschrieben. Für den Fall eines Tiefpasses ist damit die Funktion h z.B. eine Gaußfunktion

$$w_{ij} = a_1 e^{-a_2 d_{ij}^2} \tag{10.7}$$

mit der euklidischen Distanz d zwischen i-ter und j-ter Unit. Falls die Grenzfrequenz des Filters über den gesamten Inputvektor bzw. die gesamten Inputmatrix bei der Filterung von Bildern gleich sein kann (keine Positionsabhängigkeit der Filterparameter), sind nur zwei Parameter zu optimieren.

Ein etwas anspruchsvolleres Beispiel zu modellbasiertem Lernen ist die Klassifikation von transienten Signalen anhand der parametrisch modellierten (Gaussian Mixture-Modell bzw. Pole Mixture-Modell) und in neuronale Netze transformierten Signalspektren [Perl94]. Die Klassifikation der Signale erfolgt durch Schätzung der Modellparameter und Klassifikation der geschätzten Parameter. Der Vergleich der modellbasierten Netze mit einem Maximum Likelihood Classifier brachte für dieses Beispiel nicht nur wesentlich bessere Klassifikationsraten, sondern auch extrem kurze Lernzeiten.

Prinzipiell kann gesagt werden: Je mehr Vorwissen in ein transformiertes Modell gesteckt werden kann, desto besser ist die Performance bei gleichzeitiger Reduzierung der Zahl freier Parameter.

10.4 Modellbasierte Initialisierung, Pretraining

Umfangreiche Modelle und Algorithmen beinhalten bereits in traditioneller Realisierung sehr komplexe mathematische Formulierungen. In diesem Fall ist eine direkte Transformation des Modells in ein neuronales Netz nicht möglich oder würde zu einem sehr großen und komplexen Netz führen. Außerdem existiert oft kein explizites Modell sondern nur eine reale Instanz. Anstatt der Transformation kann ein neuronales Netz parallel zum Modell direkt auf das Verhalten des Modells bzw. auf das Verhalten der realen Instanz trainiert werden (Bild 10.4). Ein Beispiel dafür ist der bereits bei der

Supervisor Actor- Architektur erwähnte Roboter, dessen kinematisches Modell erlernt wird, sodaß es auch in der inversen Betriebsart verwendet werden kann [Mill90].

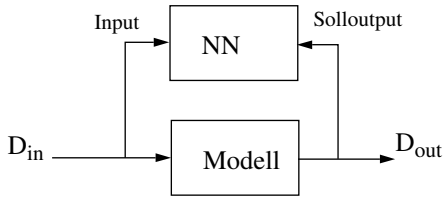
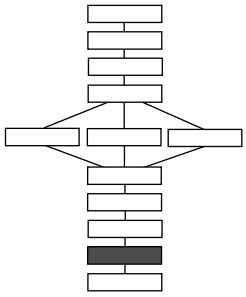


Bild 10.4 *Pretraining von Netzen anhand von Modellen oder physikalischen Prozessen*



11 Experimente

In diesem Kapitel wird das Design von modularen neuronalen Netzen anhand mehrerer konkreter Beispiele demonstriert. Das Ziel ist die praktische Evaluierung der durch Modularisierung gegenüber monolithischen Netzen erzielbaren Vorteile. Exemplarisch wird gezeigt, wie

- durch Gating durch Clustering (selbstorganisierende Selektion kombiniert mit Integration, siehe Kapitel 8 "Mixture Models") die Trainingszeit reduziert werden kann,
- wie durch Kombination von zwei redundanten Netzen der Fehler verringert und die Generalisierungsfähigkeit verbessert werden kann und
- wie durch neuronale Featureextraktion anstelle fix programmierter Featureextraktion ebenfalls der Fehler verringert werden kann.

Für alle durchgeführten Experimente (sofern nicht eplizit anders erläutert) gilt, daß die Parameter der verwendeten Netze **empirisch** ermittelt wurden[1](siehe auch Abschnitt 11.5.1). Bei allen Experimenten wurden Backpropagation-Netze als Vergleichs- bzw. als Teilnetze verwendet. Der Grund dafür liegt darin, daß Backpropagation das am weitesten verbreitete Lernverfahren ist [Akim93].

Im folgenden werden zunächst die zwei bei den Experimenten bearbeiteten Aufgaben vorgestellt. Im Anschluß daran erfolgt die Beschreibung der durchgeführten Experimente. Im letzten Teil des Kapitels wird auf die praktische Durchführung der Experimente eingegangen.

1. Für die Festlegung der Parameter von neuronalen Netzen exisieren üblicherweise keine expliziten Lösungen.

11.1 Aufgabenstellungen

Für eine aussagekräftige Evaluierung der durch Modularität erzielbaren Vorteile sind realistische Aufgabenstellungen (“real world problems”) mit zugehörenden Testdaten erforderlich. Die Aufgaben müssen hinreichend schwierig sein, sodaß sie nicht durch ein einzelnes neuronales Netz optimal gelöst werden können. Konkret wurden zwei unterschiedliche Aufgaben zu Optical Character Recognition (OCR) gewählt. Diese Aufgaben erfüllen die gestellten Forderungen [Hryc92].

Für beide Aufgaben gilt, daß es nicht das Ziel ist, ein OCR-Programm zu entwickeln, sondern die Eigenschaften der verschiedenen neuronalen Architekturen zu evaluieren.

11.1.1 Aufgabe 1: Erkennung maschinengeschriebener Buchstaben

Für die Experimente zum Gating durch Clustering und zur Kombination von zwei redundanten Netzen (Abschnitte 11.2-11.4) wurde die UCI Letter Recognition Database[2] verwendet. Sie enthält 20 000 maschinengeschriebene Buchstaben in 20 verschiedenen Fonts, die zusätzlich noch gestört wurden (Bild 11.1). Die Buchstaben werden in der UCI Letter Recognition Database durch 16 Features, die aus 45 x 45 Pixel großen Binärbildern errechnet wurden, repräsentiert. Die Binärbilder der Buchstaben stehen in der Datenbasis nicht zur Verfügung.

wird eingeklebt

Bild 11.1 *Buchstaben der UCI Letter Recognition Database*

2. Diese Datenbasis ist Teil der UCI Machine Learning Database, ftp: ics.uci.edu

Features:

- Feature 1 - 4: Position und Größe des kleinsten den Buchstaben vollständig enthaltenden Rechtecks (Ferret Box)
- Feature 5: Anzahl gesetzter Pixel (die Fläche des Buchstaben)
- Feature 6 - 12: Mittelwerte der Positionen aller gesetzten Pixel (x,y) von: x, y, x^2, y^2, x * y, x^2 * y, x * y^2
- Feature 13: Mittlere Anzahl vertikaler Kanten pro Zeile
- Feature 14: Summe der y- Positionen aller vertikalen Kanten
- Feature 15: Mittlere Anzahl horizontaler Kanten pro Spalte
- Feature 16: Summe der x- Positionen aller horizontalen Kanten

Die Aufgabe des neuronalen Systems besteht darin, die durch jeweils 16 Features repräsentierten Buchstaben zu erkennen (klassifizieren). Dabei sind keine Inputmuster als nicht klassifizierbar zurückzuweisen.

Als Performancekriterium dient die Fehlerrate (= Fehlklassifikationsrate)

$$f = \frac{\textit{Anzahl falsch klassifizierter Inputmuster}}{\textit{Gesamtzahl Inputmuster}} \tag{11.1}$$

bzw. die Erkennungsrate $e = 1 - f$. Für das Experiment zum Gating durch Clustering ist die Trainingszeit in Stunden ein zusätzliches Performancekriterium. Bei den übrigen Experimenten wird die Trainingszeit als irrelevant betrachtet.

Die Aufgabe der Erkennung der Buchstaben der UCI Letter Recognition Database wurde bereits von Frey [Frey91] mittels eines adaptiven Classifiersystems gelöst. Die beste von Frey erzielte Performance ist 80% Erkennungsrate am Testset, wobei mit 16000 Buchstaben gelernt wurde und 4000 Buchstaben als Testset verwendet wurden.

11.1.2 Aufgabe 2: Erkennung handgeschriebener Ziffern

Für das Experiment zur neuronalen Featureextraktion (Abschnitt 11.4) müssen die (Roh-) Daten zur Verfügung stehen, aus denen Features für das Training der neuronalen Netze extrahiert werden. Es wurde die “United States Postal Service Office of Advanced Technology Handwritten ZIP Code Database (1987)” (im weiteren kurz ZIP Code Database genannt) verwendet (siehe z.B. [Cun89]). Diese Datenbasis enthält händisch vorsegmentierte Grauwertbilder von handgeschriebenen Ziffern 0 - 9 (Bild 11.2). Je nach Ziffer existieren zwischen 676 und 1422 verschiedene Datensätze.

Die Aufgabe besteht darin, die handgeschriebenen Ziffern zu erkennen (klassifizieren)[3], ohne dabei Inputmuster zurückzuweisen. Als Performancekriterium dient die Fehlerrate *f* (Gl. 11.1).

3. Die Aufgabe der Klassifikation der vorsegmentierten Ziffern ist Teil einer automatischen Postleitzahlenerkennung.

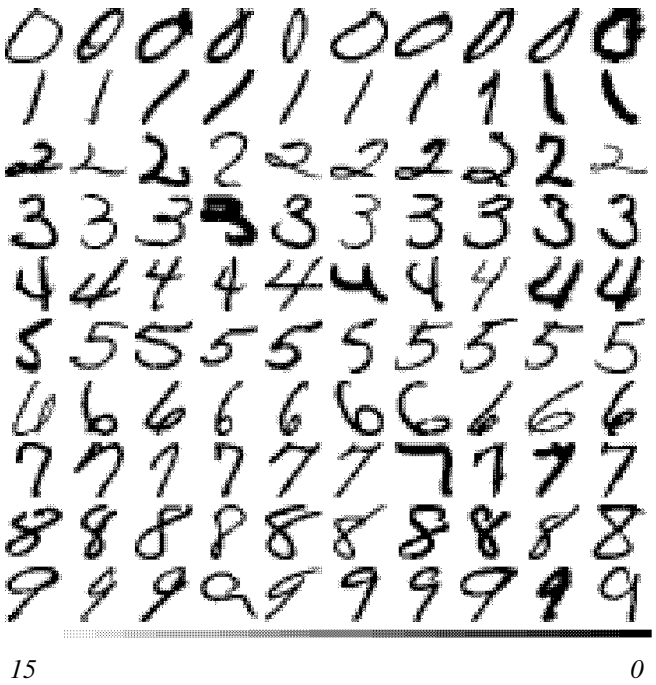


Bild 11.2 *Handgeschriebene Ziffern der ZIP Code Database*

11.2 Gating durch Clustering

Bei diesem Experiment wurde die Aufgabe der Erkennung maschinengeschriebener Buchstaben (Aufgabe 1) gelöst. Dazu wurde eine modulare Architektur verwendet, die Selektion mit Integration verbindet (siehe Kapitel 8). Die Performance der resultierenden Architektur wurde mit der Performance eines einzelnen monolithischen Netzes verglichen.

11.2.1 Monolithische Lösung

Als monolithisches Einzelnetz wurde ein 3-Layer Backpropagation Netz mit 50 hidden Units verwendet. Die Komponenten der Featurevektoren aus der UCI Letter Recognition Database wurden auf den Bereich -1 bis 1 skaliert. Für das Training dieses Netzes wurde Conjugate Gradient-Lernen mit Line-Search [Kins92] benutzt. Dadurch erübrigt sich eine manuelle Einstellung der Lern- und Momentumsraten. Um die Ergebnisse mit denen von Frey vergleichen zu können, wurde wie in [Frey91] mit 16000 Buchstaben trainiert und mit 4000 getestet.

Die mit diesem Netz erzielten Fehlerraten sowie die benötigte Trainingszeit sind in Tabelle 11.1 auf Seite 88 dargestellt.

11.2.2 Modulare Lösung

a) Architektur

Für die Dekomposition der Aufgabe wurde das Vorwissen, daß die Menge der Inputmuster in eine Reihe von apriori nicht vorgegebene Cluster zerlegt werden kann, wobei innerhalb jedes Clusters mehr als eine, aber nicht alle Klassen vorkommen [Iwat90], verwendet. Eine solche Zerlegung ist in Bild 11.3 für einen aus zwei Features bestehenden Eingangsraum veranschaulicht.

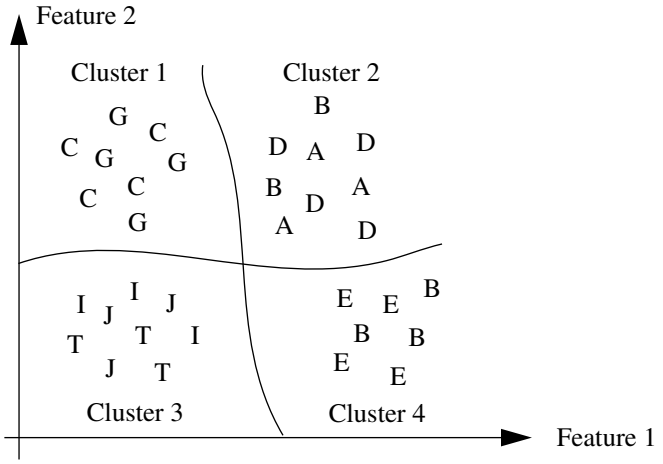


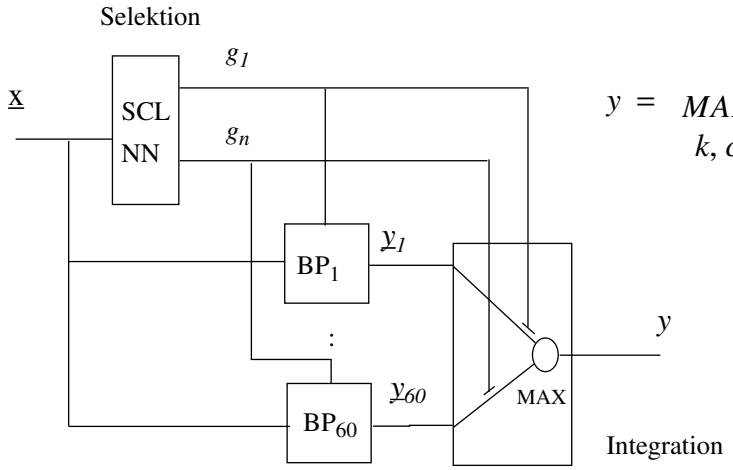
Bild 11.3 *Zerlegung des Eingangsraumes durch Clustering der Inputmuster*

Die Gesamtaufgabe kann daher in zwei Stufen zerlegt werden: 1. Finden von Clustern und 2. Klassifikation innerhalb eines Clusters. Diese Zerlegung der Aufgabe entspricht der Selektion durch Clustering (siehe Kapitel 7 und Bild 11.4).

Im Experiment wurde als Clustering-Netz (Selektor) ein Soft Competitive Learning-Netz (SCL-Netz) [Nowl90] mit 60 Output Units verwendet[4]. Jeder Outputunit des Clustering-Netzwerkes ist ein kleines Backpropagation-Netzwerk (Teilnetz) zugeordnet. Die Aufgabe des jeweiligen Teilnetzes ist es, die Buchstaben innerhalb des Clusters zu klassifizieren. Die Outputs der Teilnetze (der jeweilige Buchstabe) wurden durch 1 aus n Codierung [Rume86] repräsentiert.

4. Das Training der klassifizierenden Teilnetze könnte durch Initialisierung der Gewichte eines noch nicht trainierten Teilnetzes mit den Gewichten eines bereits trainierten Teilnetzes, dessen zugehöriger Cluster im Eingangsraum benachbart ist, beschleunigt werden. Für das Clustering wäre deshalb ein Kohonen-Netz [Koho89] bzw. ein Growing Cell Structur-Netz [Frit91] aufgrund des bei diesen Netzen vorhanden Topologieerhalts vorteilhaft. (Topologieerhalt bedeutet, daß im Inputraum benachbarte, aber in unterschiedliche Cluster fallende Inputvektoren zur maximalen Aktivierung von im Netz benachbarten Units führen.) Nachdem jedoch die Simulation mit dem XERION Netzwerk Simulator durchgeführt werden sollte und in der vorliegenden Version des Simulators das Kononen-Netz nicht verfügbar und sonst kein Netz mit Topologieerhalt vorhanden war, wurde ein Soft Competitive Learning-Netz für das Clustering verwendet.

Die Zuordnung der Inputmuster zu einem Cluster durch das Clustering-Netz ist nicht immer eindeutig. (Ein Inputmuster kann z.B. genau auf der Grenze zwischen zwei Clustern liegen.) Aus diesem Grund wurden beim Test der Architektur (Recall) alle Teilnetze ausgewertet, deren korrespondierende Cluster-Units über einem Schwellwert $g = g_{max}/2$ mit g_{max} der maximalen Aktivierung aller Units des Clustering Netzes aktiviert sind (Integration). Es ergibt sich eine gesamte modulare Architektur (Bild 11.4), die Selektion mit Integration verbindet. Dies entspricht dem Gating aus Abschnit 8.2.



$$y = \underset{k,\,c}{MAX}\ (y_{k,\,c} g_k^{\alpha}) \qquad (12.2)$$

Bild 11.4 *Modulare Architektur zur Klassifikation: Kombination von Selektion und Integration*

Mit dem Output y_k der am stärksten aktivierten Unit des k-ten Teilnetzes und dem Output g_k der k-ten Unit des Clustering-Netzes wird das Inputmuster beim Recall (Test) jenem Buchstaben zugeordnet, für den $y_k g_k^{\alpha}$ maximal ist[5] (siehe Gl. 12.2 in Bild 11.4). Der Exponent α hat die Aufgabe, den Einfluß des Clustering-Netzes zu gewichten. In den Versuchen wurde $\alpha = 0.2$ gesetzt.

b) Training

Wie beim monolithischen Vergleichsnetz wurde auch das modulare Netz mit 16 000 Buchstaben trainiert und mit 4000 getestet. Das Training der Architektur aus Clustering-Netz und Teilnetzen muß zweistufig erfolgen (siehe Abschnitt 7.1). Zuerst wurde das Clustering-Netz trainiert. Im Anschluß daran wurden die Trainingsbeispiele auf die 60 Teilnetze aufgeteilt.

Dabei entsteht das Problem der Zuordnung von Klassen zu Teilnetzen. Nachdem die Teilnetzgröße möglichst gering gehalten werden soll, ist es nicht sinnvoll, daß alle Teilnetze alle 26 Buchstaben unterscheiden können, sondern nur die im jeweiligen Cluster vorkommenden Buchstaben.

5. Diese Form der Integration ergab sich historisch (siehe auch Ergebnisse).

Die einfachste Aufteilung anhand obiger Überlegung wäre, für alle 16 000 Muster des Trainingssets das bereits trainierte Clustering-Netz auszuwerten und dabei jedes Muster genau jenem Teilnetz zuzuordnen, dessen zugehörige Unit des Clustering-Netzes am stärksten aktiviert ist. Diese Aufteilung aller Muster des Trainingssets berücksichtigt jedoch die durch Redundanz erzielbaren Vorteile (siehe Kapitel 3) nicht.

Aus diesem Grund erfolgte die Zuordnung von Klassen zu Teilnetzen durch einen Optimierungsprozeß mit dem Ziel, jedes Inputmuster mehr als einem, jedoch im Mittel nur 2-3 Teilnetzen zuzuordnen. Der Optimierungsprozeß (er ist detailliert in [Bart93] beschrieben) wurde anhand eines globalen, die Zuordnung aller Muster des Trainingssets bewertenden Qualitätsmaßes q gesteuert. Mit der Qualität q_s der Zuordnung des s-ten Inputmusters ergibt sich das globale Qualitätsmaß q durch Summierung aller Qualitäten q_s unter der zusätzlichen Bedingung, daß insgesamt möglichst wenige Outputunits existieren sollen. Mit einer Konstanten γ und der Summe $\sum_k m_k$ der Anzahl an Outputunits aller r Teilnetze ist die globale Qualität

$$q = \sum_s q_s - \gamma \sum_k m_k. \tag{11.3}$$

Existiert für ein Inputmuster kein zugehöriges Teilnetz, so ist keine Klassifizierung möglich, die Qualität q_s der Zuordnung des s-ten Inputmusters ist gleich Null. Sobald jedoch mindestens ein Teilnetz existiert, das auf die Klasse des Inputmusters trainiert wird und dessen Outputunit des Clustering-Netzes stark aktiviert ist, steigt die Qualität q_s sofort stark an. Wird das Inputmuster in mehreren Teilnetzen behandelt, deren Outputunits des Clustering-Netzes stark aktiviert sind, so steigt die Qualität q_s nur noch geringfügig weiter an. Mit der Summe p der Aktivierungen jener Units des Clustering-Netzes, in deren zugehörigen Teilnetzen der Buchstabe tatsächlich behandelt wird, wird die Qualität q_s der Zuordnung eines Inputmusters definiert als

$$q_s = \frac{4\sqrt[4]{p} - p}{3}. \tag{11.4}$$

Für $p \in [0,1]$ [6] hat diese Funktion hat einen Wertebereich von 0 bis 1 und erfüllt die oben dargestellten Forderungen (Bild 11.5). (Anstelle der Funktion von Gl. 11.4 wäre auch jede andere Funktion, die die dargestellten Eigenschaften aufweist verwendbar.)

Die algorithmische Optimierung der Zuordnung der Klassen (10 Durchläufe durch das Trainingsset) ergab, daß ein Teilnetz im Durchschnitt nur 3.5 Buchstaben voneinander unterscheiden können mußte.

6. Die Aktivierungen der Outputunits des Competitive Learning-Netzes sind durch die Aktivierungsfunktionen der Units so normiert, daß deren Summe eins ergibt.

Mit dieser Zuordnung von Klassen zu Teilnetzen wurden die Teilnetze trainiert. Als Lernalgorithmus für die Teilnetze wurde wie beim monolithischen Einzelnetz Conjugate Gradient-Lernen mit Line-Search [Kins92] verwendet.

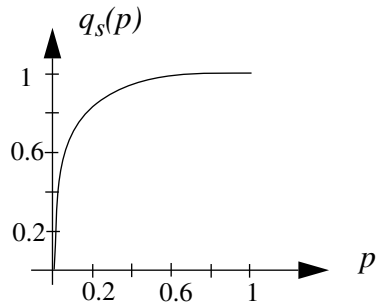


Bild 11.5 *Graph der Qualitätsfunktion $q_s(p)$*

11.2.3 Ergebnisse

In Tabelle 11.1 sind die erzielten Fehlerraten dargestellt. Bei der Fehlerrate am Testset von 4000 Buchstaben unterscheidet sich die modulare Lösung von der monolithischen nur um 0.2%. Beide neuronale Lösungen sind mit 7.9% bzw. 8.1% Fehlerrate deutlich besser als die Classifier-Systeme von Frey et. al. [Frey91], die am selben Testset eine Fehlerrate von 20% aufwiesen.

Im Unterschied zur Fehlerrate bestehen bei der Trainingszeit jedoch erhebliche Unterschiede zwischen dem modularen und dem monolithischen Netz: Das modulare Netz braucht insgesamt nur 3h Trainingszeit, während die monolithische Architektur 13h Trainingszeit benötigt. Alle Versuche wurden auf einer DEC-5000 mit demselben Netzwerksimulator durchgeführt.

	Trainingset	**Testset**	**Trainingszeit**
monolithisches Netz	3.9%	7.9%	13h
modulares Netz	3.1%	8.1%	3h

Tabelle 11.1 *Vergleich der Fehlerraten und Trainingszeiten von monolithischer und modularer Lösung zu Gating durch Clustering*

Der Unterschied in der Trainingszeit zeigt den wesentlichen Vorteil der Selektion von Teilnetzen. Die Fehlerrate der modularen Architektur könnte durch Verwendung eines Integrationsverfahrens, das nicht nur eines der 60 Teilnetze auswählt, sondern die reellwertigen Ergebnisse der Teilnetze mischt (z:B. gewichtete Mittelung), verbessert werden (siehe Abschnitt 11.3).

11.3 Kombination von zwei redundanten Teilnetzen

Mit diesen Experimenten sollen die theoretischen Überlegungen zur Verbesserung der Generalisierungfähigkeit durch Selektion (siehe Abschnitt 3.4) bzw. Integration (siehe Kapitel 6) anhand der Erkennung maschinengeschriebener Buchstaben (Aufgabe 1) experimentell untermauert werden. Dazu werden zunächst zwei verschiedene neuronale Netze unabhängig voneinander auf die Aufgabe trainiert. Im Anschluß daran werden die Ergebnisse der beiden Netze zu einem Ergebnis kombiniert.

11.3.1 Monolithische Lösungen

Für die Reduktion der Fehlerrate durch Integration von redundanten Teilnetzen ist eine weitgehende Unabhängigkeit der Fehler der Teilnetze und damit eine Diversität der Teilnetze erforderlich (siehe Kapitel 6). Dies wurde durch die Wahl eines lokal optimierenden und eines global optimierenden Lernverfahrens gewährleistet. In Tabelle 11.2 sind die beiden verwendeten Netze beschrieben.

	Lernverfahren	**Parameter**
Netz 1	Standard Backpropagation (BP) [Rume86] Gradientenabstieg, global optimierend	16 Input-, 50 Hidden-, 26 Outputunits (1 aus n Codierung), Lernrate = 0.01, Momentumsrate = 0.7, Logistic-Aktivierungsfunktion
Netz 2	Learning Vector Quantization 2 (LVQ2) [Koho90] lokal optimierend	900 Units, euklidische Distanz, Lernrate verlaufend nach 1/t von einem Startwert von 0.3 bis zu einem Endwert von 0.

Tabelle 11.2 *Lernverfahren und Parameter der für die Integration verwendeten Teilnetze*

Das Learning Vector Quantization 2-Lernverfahren für Netz 2 ergab sich empirisch: Zunächst wurde versucht, als lokal optimierendes Teilnetz ein Kohonen-Netz zu verwenden. Mit einem Kohonen-Netz wurde jedoch eine Fehlerrate von nur 16% im Vergleich zu 9.03% mit Backpropagation erreicht. LVQ1 [Koho90] brachte eine Verbesserung auf 11.35%, aber erst LVQ2 erreichte eine mit Backpropagation vergleichbare Fehlerreate von 9.13% (siehe Tabelle 11.3 und 11.4 für detailliertere Fehlerraten von Netz 1 und Netz 2).

Training der Teilnetze

Wie bei der selektiven Architektur wurde mit 16 000 Datensätzen trainiert und mit 4000 Datensätzen getestet.

Für das Netz 1 (BP) wurden die Features aus der Letter Recognition Database auf den Bereich -1 bis 1 skaliert. Die Gewichte von Netz 1 wurden mit zufälligen, im Intervall [-0.1, +0.1] gleichverteilten Werten eingestellt.

Das Netz 2 (LVQ2) wurde direkt auf die Features der Letter Recognition Database trainiert. Die Gewichte von Netz 2 wurden mit den Gewichten eines zweidimensionalen Kohonen-Netzes mit 30 x 30 Units, das zunächst auf die Trainingsdaten trainiert wurde, voreingestellt. Das Training des Kohonen-Netzes erfolgte mit euklidischer Nachbarschaft. Die Nachbarschaftsreichweite wurde abnehmend nach 1/t von 15 bis 0 eingestellt.

In Tabelle 11.3 sind die erzielten Fehlerraten dargestellt.

	Trainingsset	**Testset**
Netz 1 (BP)	6.8%	9.03%
Netz 2 (LVQ2)	4.8%	9.13%

Tabelle 11.3 *Fehlerraten von Netz 1 und Netz 2 bei der Erkennung gedruckter Buchstaben*

Zur Überprüfung der Unabhängigkeit der Fehler der beiden Netze wurde die Verteilung der Fehler auf die zwei Netze untersucht. Für das Auftreten von Fehlern existieren fünf Möglichkeiten: Im Fall der Übereinstimmung der Ergebnisse von Netz 1 und Netz 2 kann das Ergebnis falsch (I) oder richtig (II) sein. Im Fall, daß sich die Ergebnisse von Netz 1 und Netz 2 unterschieden (disagree), kann jeweils ein Ergebnis richtig und das andere falsch sein (III u. IV) oder es sind beide Ergebnisse falsch (V). Mit dieser Unterteilung in fünf Fälle ist in Tabelle 11.4 dargestellt, wie sich die Fehler auf die beiden Netze verteilen.

		Trainingsset		**Testset**	
		Anzahl	**Anteil**	**Anzahl**	**Anteil**
I	Übereinstimmung, Ergebnis korrekt	14 395	89.97%	3 429	85.73%
II	Übereinstimmung, Ergebnis falsch	134	0.84%	63	1.56%
III	verschied. Ergebnisse, Netz1 falsch	753	4.71%	206	5.15%
IV	verschied. Ergebnisse, Netz2 falsch	519	3.24%	210	5.25%
V	verschied. Ergebnisse, beide Netze falsch	199	1.24%	92	2.30%

Tabelle 11.4 *Aufteilung der Fehler auf Netz1 (BP) und Netz2 (LVQ2)*

Aufgrund der verschiedenen Lernalgorithmen der beiden Netze ist der Anteil jener Inputmuster, bei denen sowohl Netz 1 wie auch Netz 2 ein falsches Ergebnis liefern (II und V in Tabelle 11.4), insgesamt nur 3.9% gegenüber 9.03% der Fehlerrate von Netz 1, dem besseren der beiden Teilnetze. Unter der Vorraussetzung, daß es durch die Kombination der beiden Netzwerke gelingt, immer das richtige Ergebnis auszuwählen, ließe sich damit die Fehlerrate des Gesamtsystems auf 3.9% reduzieren.

11.3.2 Kombination von Netz 1 und Netz 2

In diesem Abschnitt werden drei Experimente zur Kombination der beiden Teilnetze Netz 1 und Netz 2 präsentiert, die das Ziel haben, die Fehlerrate zu reduzieren: Fehlergesteuerte Selektion, Mittelwertbildung und neuronale Fusion.

a) Fehlergesteuerte Selektion

Die erste getestete Art der Kombination der Ergebnisse von Netz 1 und Netz 2 ist die Selektion eines der Ergebnisse von Netz 1 und Netz 2 (Bild 11.6). Dazu wurde ein zusätzliches neuronales Netz (Selektions-Netz) verwendet, das entscheidet, welches der Teilnetze das richtige Ergebnis liefert. Dies entspricht der fehlergesteuerten Selektion aus Kapitel 7 mit dem Unterschied, daß das Selektions-Netz hier im Anschluß an die Teilnetze trainiert wurde und nicht gemeinsam mit den Teilnetzen. Außerdem wurden die Teilnetze unabhängig voneinander auf die Lösung der gesamten Aufgabe trainiert und nicht nur jeweils auf einen Teil der Aufgabe.

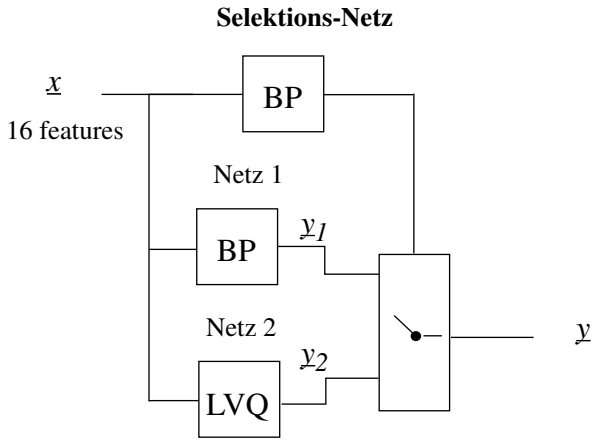


Bild 11.6 *Auswahl eines Ergebnisses durch ein Selektionsnetz*

Da bei 90.81% der Trainingsbeispiele die Ergebnisse von Netz 1 und Netz 2 übereinstimmen (I und II in Tabelle 11.4) und nur bei 9.19% der Trainingsbeispiele eine Entscheidung notwendig ist, wurde das Selektions-Netz nur auf jene Trainingsbeispiele aus dem Trainingsset trainiert, bei denen die zwei Teilnetze verschiedene Ergebnisse liefern (Disagree-Trainingsset, Bild 11.7). Dies sind die 1471 Trainingsbeispiele der Zeilen III - V in Tabelle 11.4.

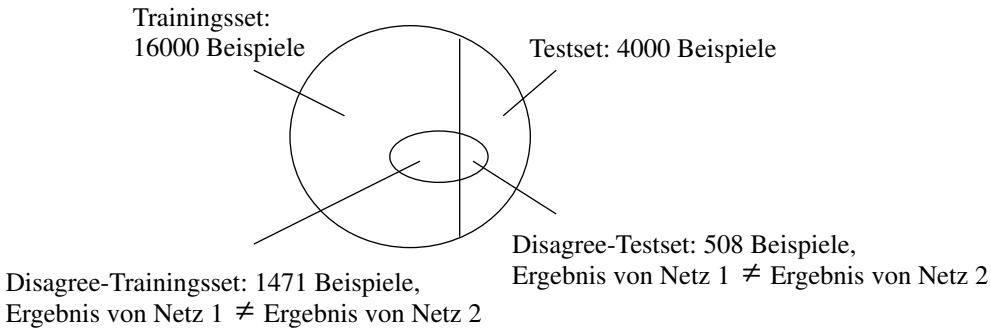


Bild 11.7 *Trainingsset, Testset, Disagree-Tainingsset und Disagree-Testset*

Konkret wurde ein Backpropagation-Netz mit 16 Input- 10 Hidden- und 3 Outputunits als Selektions-Netz verwendet (Bild 11.8).

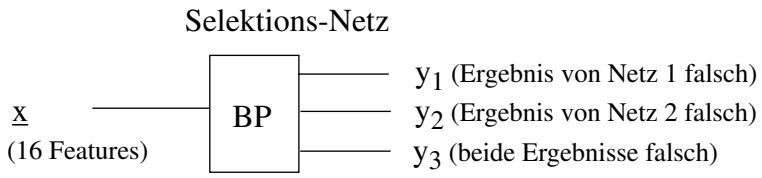


Bild 11.8 *Selektions-Netz zur Auswahl eines der Ergebnisse von Netz 1 und Netz 2*

Die Fehlerrate dieses Selektions-Netzes am 505 Beispiele umfassenden Disagree-Testset (siehe Bild 11.7 und Zeilen III - V in Tabelle 11.4) ist 49.4%. Die gesamte modulare Architektur erreichte damit eine Fehlerrate[7] von 8.2% am 4000 Beispiele umfassenden Testset gegenüber der Fehlerrate des einzelnen Backpropagation-Netzes von 9.03% (siehe Zusammenfassung der Ergebnisse zur Integration in Tabelle 11.5).

Diese Verbesserung der Fehlerrate ist enttäuschend. Eine Hypothese zur Erklärung der geringen Verbesserung ist, daß die Entscheidungsgrenze zwischen den Klassen “Ergebnis von Netz 1 falsch” und “Ergebnis von Netz 2 falsch” sehr komplex ist, sodaß sie durch die 1471 Trainingsbeispiele des Disagree-Trainingssets nicht erlernt werden kann. (siehe Veranschaulichung einer komplexen Grenze zwischen zwei Bereichen eines zweidimensionalen Eingangsraumes in Bild 11.9).

7. Im Falle der Aussage des Selektions-Netzes, daß sowohl das Ergebnis von Netz 1 wie auch das Ergebnis von Netz 2 falsch ist, wurde das Ergebnis von Netz 1 als das Gesamtergebnis verwendet.

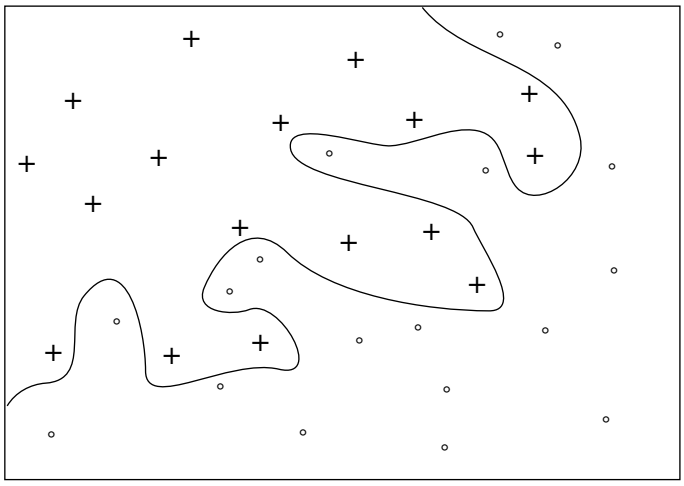


Bild 11.9 *Symbolische Darstellung einer komplexen Grenze zwischen zwei Klassen*

Zur Verbesserung der Fehlerrate (und Erhärtung der Erklärungshypothese) wurde das Trainingsset für das Selektions-Netz folgendermaßen erweitert: Aus den 16 000 Vektoren in der Datenbasis für das Training von Netz 1 und Netz 2 wurden weitere Vektoren durch Störung jeweils einer Komponente des Vektors generiert. Dadurch lassen sich weitere Trainingsbeispiele erzeugen, bei denen Netz 1 und Netz 2 verschiedene Ergebnisse liefern. Die Größe des Disagree-Trainingsset für das Selektions-Netzwerk wurde auf diese Weise von 1471 auf 4413 Beispiele verdreifacht (erweitertes Disagree-Trainingsset). Durch das Training des Selektions-Netzes auf das erweiterte Disagree-Trainingsset konnte die Gesamtfehlerrate der modularen Architektur auf 7.83% am 4000 Beispiele umfassenden Testset und auf 3.76% am 16000 Beispiele umfassenden Trainingsset von Netz 1 und Netz 2 reduziert werden.

b) Mittelwertbildung

Die Mittelwertbildung $\underline{y} = (\underline{y}1 + \underline{y}2)/2$ (siehe Kapitel 6 und Bild 11.10) ist eine mathematisch wenig aufwendige Art der Integration von redundanten Ergebnissen.

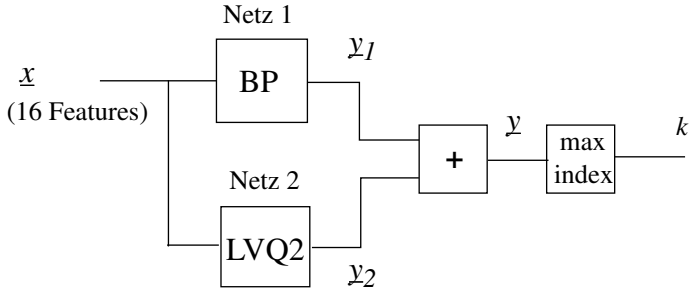


Bild 11.10 *Integration durch Mittelwertbildung*

Um diese Art der Integration anwenden zu können, müssen die beiden Teilnetze in Typ, Dimension und Wertebereich gleiche Outputwerte liefern. Das trainierte Netz 1 (BP) liefert einen 26-dimensionalen Outputvektor $\underline{y}_1$, dessen Komponenten aus dem Intervall [0,1] sind. Nicht jedoch das Netz 2 (LVQ2). Dieses liefert standardmäßig nur die Klassse $k \in \{0, 1, 2, ... ,26\}$, die dem Inputmuster zugeordnet wird und den Referenzvektor $\underline{y}_b \in \Re^{16}$ der "Best Matching Unit". Aufgrund dieser Inkompatibilität der Outputs der beiden Teilnetze wurde das Netz 2 (LVQ2) um einen für Klassifikationsaufgaben unüblichen Teil erweitert:

Wie bei vielen unsupervised lernenden Clustering-Netzen besitzt auch bei Netz 2 jede Outputunit u_j des Netzes einen Referenzvektor $\underline{x}_j$ der gleichen Dimension wie der Inputvektor $\underline{x}$. Für jeden zu klassifizierenden Inputvektor $\underline{x}$ wird nun nicht nur die "Best Matching-Unit" und deren Aktivierung o nach

$$o = \underset{j}{MIN} \left\| \underline{x} - \underline{x}_j \right\| \tag{11.5}$$

berechnet, sondern ein Vektor $\underline{y}$ mit einer Komponente y_c für jede der 26 Klassen. y_c ist die minimale Distanz zwischen Inputvektor $\underline{x}$ und den Referenzvektoren all jener Units des Netzes, die der Klasse c zugeordnet sind:

$$y_c = \underset{j_c}{MIN} \left\| \underline{x} - \underline{x}_{j_c} \right\| \qquad j_c \text{ ... units der Klasse c} \tag{11.6}$$

Wird ein Inputmuster der Klasse c zugeordnet, so ist damit der Wert der c-ten Komponente von $\underline{y}$ der kleinste aller Komponenten von $\underline{y}$. Bei Netz 1 (BP) hingegen entspricht aufgrund der 1 aus n Codierung der Inputvektor jener Klasse, deren zugehörige Outputkomponente maximal wird. Um einen Outputvektor $\underline{y}_2$ von Netz 2 zu erzeugen, der mit dem Outputvektor $\underline{y}_1$ von Netz 1 gemittelt werden kann, wurde jede Komponente y_c von $\underline{y}$ mit dem Wert *max* der größten Komponente von $\underline{y}$ transformiert nach

$$y_c' = \left(1 - \frac{1}{max} y_c\right)^4 . \tag{11.7}$$

Durch die Skalierung mit dem Wert *max* wird erreicht, daß die Werte aller Komponenten y_c' des Outputvektors $\underline{y}_2$ im Intervall [0,1] liegen. Die Potenzierung mit vier ergab sich empirisch. Sie führt zur Verstärkung der Differenz zwischen der größten und der zweitgrößten Komponente des Vektors $\underline{y}$.

Mit dieser Erweiterung von Netz 2 konnte der Mittelwert der Ergebnisse der beiden Netze gebildet werden. Die Fehlerraten, die durch Mittelwertbildung erreicht wurden, sind 7.4% am Testset und 5.07% am Trainingsset (siehe Tabelle 11.5). Dies liegt unter denen der fehlergesteuerten Selektion.

c) Neuronale Fusion

Neuronale Fusion ist die Integration von redundanten Ergebnissen zu einem Gesamtergebnis durch ein supervised lernendes neuronales Netz (Fusionsnetz, Bild 11.11 und Abschnitt 6.1.4). Das Fusionsnetz erhält die Outputvektoren der Teilnetze als Input. Es wird direkt auf den Solloutpupt des Gesamtsystems trainiert.

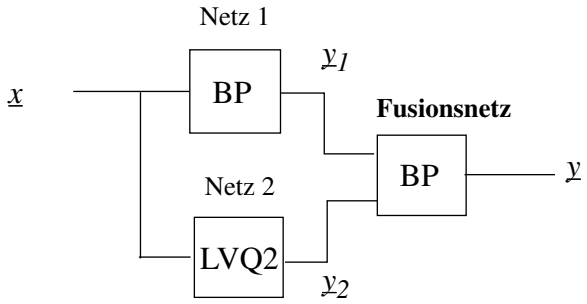


Bild 11.11 *Neuronale Fusion der Ergebnisse von Netz 1 und Netz 2*

Im Experiment wurde für das Fusionsnetz ein Standard Backpropagation-Netz mit 50 hidden Units (dieselbe Anzahl wie für Netz 1) gewählt. Zur Erzielung minimaler Fehlerraten müssen gleichbedeutende Inputs eines Netzes gleich codiert werden [Hint86]. Daher wurde auch bei diesem Experiment das erweiterte LVQ2-Netz als Netz 2 (siehe vorhergehenden Abschnitt “Mittelwertbildung”) verwendet. Dieses erweiterte LVQ2-Netz liefert einen 26-dimensionalen, reellwertigen Outputvektor. Damit ergibt sich für das Fusionsnetz eine Architektur mit 52 Input-, 50 Hidden- und 26 Outputunits. In Bild 11.12 sind typische Outputvektoren von Netz 1 und Netz 2 sowie der zugehörige fusionierte Outputvektor für einen Fall, in dem die Teilnetze verschiedene Ergebnisse liefern, dargestellt.

Für das Training des Fusionsnetzes wurde das gesamte 16000 Zeichen umfassende Trainingsset verwendet. Bereits nach einem Durchlauf durch das gesamten Trainingsset wurde die Fehlerrate des besseren der beiden Teilnetze von 9% erreicht. Die Fehlerraten nach eingetretener Konvergenz des Fusionsnetzes sind 5.8% am Testset und 2.9% am Trainingsset (siehe Tabelle 11.5).

In einem zweiten Versuch zur neuronalen Fusion wurde das Fusionsnetz auf das bei den Experimenten zum Gating durch Clustering (Abschnitt 11.2) erzeugte erweiterte Disagree-Trainingsset trainiert. Dieses Trainingsset enthält nur Inputmuster, für die Netz 1 und Netz 2 unterschiedliche Ergebnisse liefern. Es ergibt sich eine Fehlerrate von 5.2% am Testset und 2.0% am Tainingsset.

Output Netz 1

Output Netz 2

Output Fusionsnetz

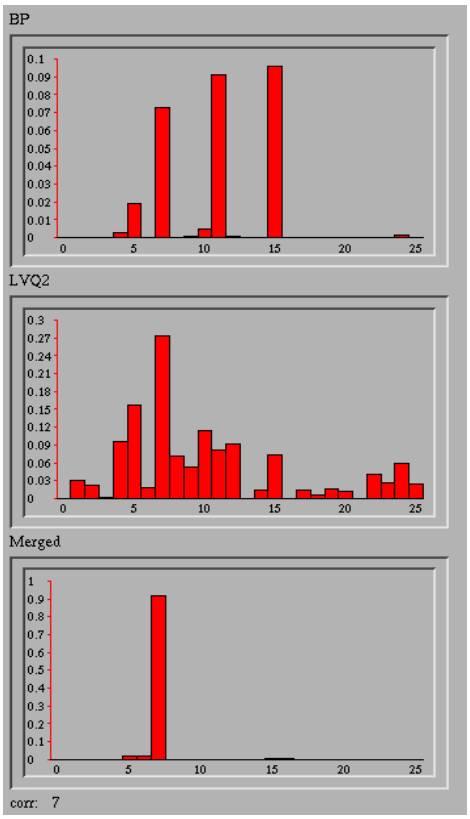


Bild 11.12 *Beispiel für die Outputvektoren der beiden Teilnetze und für den daraus fusionierten Outputvektor*

11.3.3 Ergebnisse

In Tabelle 11.5 sind alle Ergebnisse der Experimente zu den drei getesteten Möglichkeiten der Kombination von redundanten Teilnetzen zusammengefaßt. Zum Vergleich enthält diese Tabelle auch die Fehlerraten von Netz 1 und Netz 2.

modulare Architektur	**Trainingsset des Selektions-/ Integrationsnetzes**	**Fehlerrate Trainingsset**	**Fehlerrate Testset**
a) fehlergesteuerte Selektion	Disagreeset	3.9%	8.2%
a) fehlergesteuerte Selektion	erweitertes Disagreeset	3.76%	7.83%
b) Mittelwertbildung	-	5.07%	7.4%
c) Neuronale Fusion	orig. Trainingsset	2.9%	5.8%
c) Neuronale Fusion	erweitertes Disagreeset	2.0%	5.2%
Teilnetz			
Netz 1 (BP)	orig. Trainingsset	6.8%	9.03%
Netz 2 (LVQ2)	orig. Trainingsset	4.8%	9.13%

Tabelle 11.5 *Übersicht über die durch Kombination der Ergebnisse von Netz 1 und Netz 2 erreichten Fehlerraten sowie (zum Vergleich) die Fehlerraten von Netz 1 und Netz 2*

Insgesamt konnte demonstriert werden, daß die Fehlerrate durch Kombination zweier unabhängig trainierter Netzwerke um mehr als ein Drittel gesenkt werden kann. Die neue Form der Integration durch Fusion mittels eines neuronalen Netzes ist dabei allen anderen getesteten Verfahren überlegen. Dieses Ergebnis hat großes Gewicht, weil laut übereinstimmender Meinung verschiedener Autoren [Perr94][Batt94] die Mittelwertbildung bereits eine der besten Integrationsmethoden ist.

11.4 Neuronales Preprocessing

In diesem Experiment soll demonstriert werden, inwieweit eine Abbildung einer starren algorithmischen Featureextraktion auf ein neuronales Netz (siehe Kapitel 10) zu einer Reduktion der Fehlerrate des Gesamtsystems führt.

In Bild 11.13 ist die Architektur des modularen Systems dargestellt. Sie entspricht der Serienschaltung. In Bild 11.13 sind im Unterschied zu Kapitel 4 auch die für das Training notwendigen Verbindungen der Netze dargestellt (strichliert). Da die Datenbasis für die Erkennung der gedruckten Buchstaben (Aufgabe 1) die Pixelmaps der Buchstaben nicht enthält, wurde für das Experiment zu neuronalem Preprocessing die ZIP Code Database (Aufgabe 2) verwendet.

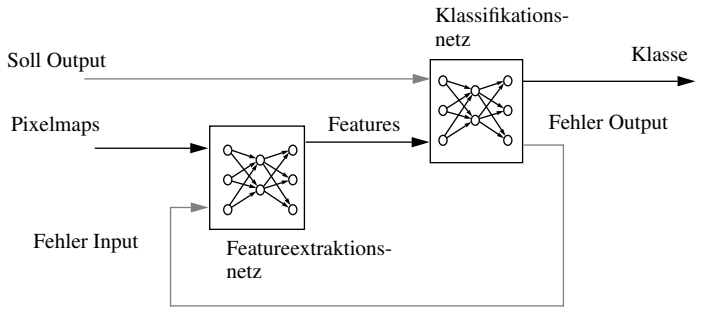


Bild 11.13 *Neuronale Featureextraktion für neuronale Klassifikation*

Die ZIP Code Database des US Postal Service enthält Grauwertbilder unterschiedlicher Größe von handgeschriebenen Ziffern (siehe Abschnitt 11.1.2). Diese Grauwertbilder wurden auf 16 x 16 Pixel Grauwertbilder mit 16 Graustufen skaliert und dann binarisiert. Als Features wurden dieselben wie bei den bisherigen Versuchen gewählt (siehe Abschnitt 11.1.1). Zunächst wurde für jedes der 16 Features ein Teilnetz gebildet, das die Aufgabe hat, das jeweilige Feature zu errechnen. Dazu wurde für jedes Feature ein Teilnetz mit modellbasierter Architektur (siehe Abschnitt 10.2) mit “händisch” eingestellten Gewichten erstellt. Alle Teilnetze zusammen ergeben das Featureextraktionsnetz.

Die Teilnetze zur Featureextraktion sind in [Neme94] detailliert beschrieben. In Bild 11.14 ist ein Beispiel für ein Teilnetz dargestellt. Es berechnet die mittlere Anzahl vertikaler Kanten pro Zeile des Pixelmaps (Feature 13). Als Kante wurde hier ein 0 - 1 Übergang in einer Zeile definiert. Die Threshold-Units im ersten hidden Layer des Netzes ermitteln die 0 - 1 Übergänge, die folgende sigmoide Unit summiert deren

Anzahl. Das Gewicht der Outputunit führt zur Mittelwertbildung. Die doppelt umrandeten Units erhalten nur einen Input. Sie bearbeiten den Rand des Pixelmaps.

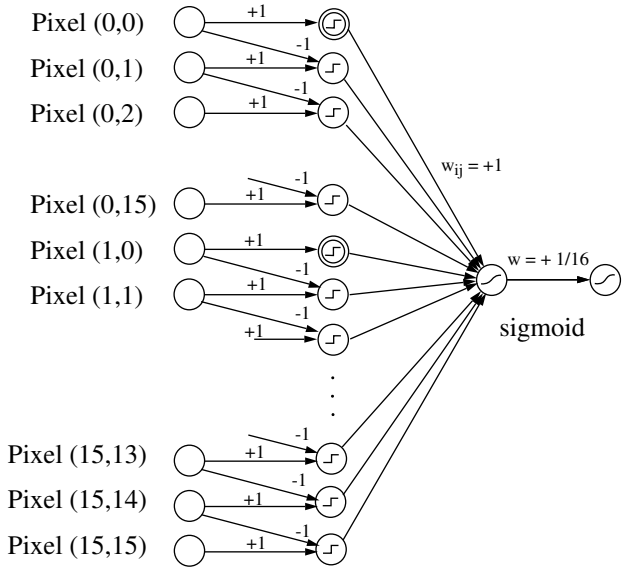


Bild 11.14 *Neuronales Netz zur Berechnung des Mittelwertes der Anzahl vertikaler Kanten pro Zeile des Pixelmaps (Feature 13)*

Ein weiteres Beispiel für ein Featureextraktions-Teilnetz ist in Bild 10.3 auf Seite 78 dargestellt. Mit zwei solchen Teilnetzen können das am weitesten rechts und das am weitesten link liegende gesetzte Pixel bestimmt werden (die Grenzen). Mit einer sigmoiden Unit kann aus den beiden Grenzen die Breite der Ziffern bestimmt werden (Feature 3, angenähert).

Mit dem “händisch” konstruierten Featureextraktionsnetz wurde dann ein Backpropagation-Netz (Conjugate Gradient-Lernverfahern mit Line Search) zur Klassifikation der Features trainiert. Dabei wurden nur die Gewichte des Klassifikationsnetzes, nicht aber die Gewichte des Featureextraktionsnetzes adaptiert. Wie bei den bisherigen Experimenten wurden 80% der Ziffern als Trainingsset und 20% als Testset verwendet. Für die Codierung der Klassen wurde wieder 1 aus n Codierung verwendet. Die vom Extraktionsnetz gelieferten Werte wurden durch einen zwischen Featurextraktionsnetz und Klassifikationsnetz zusätzlich eingefügten Layer auf den Bereich -1 bis 1 skaliert (in Bild 11.13 nicht dargestellt). Das Training des Klassifikationsnetzes wurde zum Zeitpunkt der besten Performance am Testset abgebrochen. Außerdem wurde die Architektur des Klassifikationsnetzes optimiert. Die besten Ergebnisse wurden mit einem hidden Layer mit 16 Units erzielt. Die erreichten Fehlerraten sind in Tabelle 11.6 aufgelistet.

Anschließend an das Training des Klassifikationsnetzes wurde das Featureextraktionsnetz gemeinsam mit dem Klassifikationsnetz trainiert. Dadurch wurde die Feinabstimmung von Featureextraktions- und Klassifikationsnetz bewirkt.

Beim Training des Featureextraktionsnetzes trat ein Problem auf: die Werte von Gewichten, die sich innerhalb einer Unit um mehr als 1 unterscheiden (z.B. der Gewichte aller Outputunits der Featureextraktions-Teilnetze, aller Units des 1. hidden Layers für die Features 1 bis 4 und aller Units des Skalierungslayers), gingen beim Lernen völlig verloren. Dadurch ging auch die Fähigkeit der Featureextraktion verloren. Um dies zu verhindern, mußten diese Gewichte beim Lernen "eingefroren" werden. Dies schränkt jedoch die Anpassungsfähigkeit der neuronalen Featureextraktion ein. Mögliche Lösungen dieses Problems wären:

- die Lernrate abhängig von der Größe der Gewichte zu machen,
- die Gewichte über eine Funktion zu berechnen und statt der Gewichte die Parameter der Funktion zu optimieren (siehe Abschnitt zu modellbasiertem Lernen ab Seite 78), oder
- für Netze mit solch unteschiedlich großen Gewichten statt händischem Design der Netze ein Pretraining anhand der algorithmischen Featureextraktion (siehe Abschnitt über Pretraining ab Seite 79) durchzuführen.

In Tabelle 11.6 sind die Fehlerraten der modularen Architektur vor und nach der Adaptierung des Featureextraktionsnetzes dargestellt. Durch das gemeinsame Training von Featureextraktionsnetz und Klassifikationsnetz ergab sich eine Verringerung der Fehlerrate am Testset von 0.75% absolut. Die vorhergehende Optimierung des Klassifikationsnetzes anhand der neuronalen, nicht lernenden Featureextraktion stellt sicher, daß diese Verbesserung ausschließlich durch die Adaptivität der Featureextraktion erzielt wurde und nicht durch ein falsch (z.B. zu klein) dimensioniertes Klassifikaionsnetz.

	Training	**Test**
vor Adaptierung der Featureextraktion	8.66 %	9.37 %
nach Adaptierung der Featureextraktion	3.91 %	8.62 %

Tabelle 11.6 *Fehlerraten bei der Ziffernerkennung vor und nach Adaptierung der Featureextraktion*

Dieses Experiment hat gezeigt, daß durch neuronales Preprocessing anstelle eines fixen Preprocessings eine Verbesserung der Fehlerrate möglich ist. Die erzielten absoluten Fehlerraten liegen jedoch über denen von Cun [Cun89]. Cun erreichte mit einem Fünf-Layer-Netz mit 8020 freien Parametern im Vergleich zu den 416 freien Parametern des Klassifikationsnetzes im Experiment hier eine Fehlerrate von 5.4% am Testset und 0.5% am Trainingsset. Für die praktische Anwendung des neuronalen Preprocessings sind deshalb nicht nur die oben geschilderten Probleme mit Gewichten unterschiedlicher Größe beim Featureextraktionsnetz zu lösen, sondern auch die Auswahl der Features zu verbessern sowie eine weitere Optimierung des Klassifikationsnetzes (z.B. andere Lernverfahren) durchzuführen.

11.5 Praktische Durchführung der Experimente

Die Experimente zum Gating durch Clustering und zu neuronalem Preprocessing wurden im Rahmen von Praktikas [Bart93][Neme94] durchgeführt. Dabei wurde der XERION Netzwerksimulator auf einer DEC 5000 Workstation verwendet. Der XERION Netzwerksimulator ist ein unit- und clusterorientierter Simulator. Der besondere Vorteil von XERION ist die Möglichkeit zur Konstruktion eigener Netzwerkparadigmen. Dies wurde für die Konstruktion des Featureextraktionsnetzes für das Experiment von Abschnitt 11.4 verwendet.

Die Experimente zur Integration von zwei redundanten Teilnetzen wurden unter Verwendung von ECANSE (Environment for Computer Aided Neural Software Engineering) der Firma Siemens auf einer SUN Sparc10 Workstation durchgeführt. ECANSE ist ein modulorientierter Simulator mit graphischer Benutzeroberfläche. Er erleichtert vor allem die Kombination von verschiedenen neuronalen Netzen sowie die Konstruktion hybrider Systeme. In Bild 11.15 ist ein Beispiel für die Oberfläche von ECANSE (die Simulaiton zur fehlergesteuerten Selektion von Abschnitt 11.3.2 a) dargestellt.

11.5.1 Rechenzeit und Parametrisierung

Das wesentliche praktische Problem bei allen Experimenten waren die großen Trainingszeiten von bis zu 72h für einen einzigen Versuch (200 Trainingsepochen entsprachen 4 Mio. Trainingsbeispielen mit jeweils z.B. 2100 “Connection Updates” für ein einziges 16-50-26 Backpropagation-Netz).

Eine vollständige Parameteroptimierung konnte aus diesem Grund und aufgrund der kombinatorischen Explosion der Anzahl an erforderlichen Simulationsläufen mit der Anzahl an Parametern nicht durchgeführt werden. Diese Vorgangsweise findet jedoch ihre Rechtfertigung in der Aufgabenstellung zu den Experimenten, nach der die Vorteile der modularen Architekturen im Vergleich zu monolithischen Netzen evaluiert werden sollten. Dies wird bereits durch suboptimale Parametereinstellungen bei den modularen Architekturen möglich.

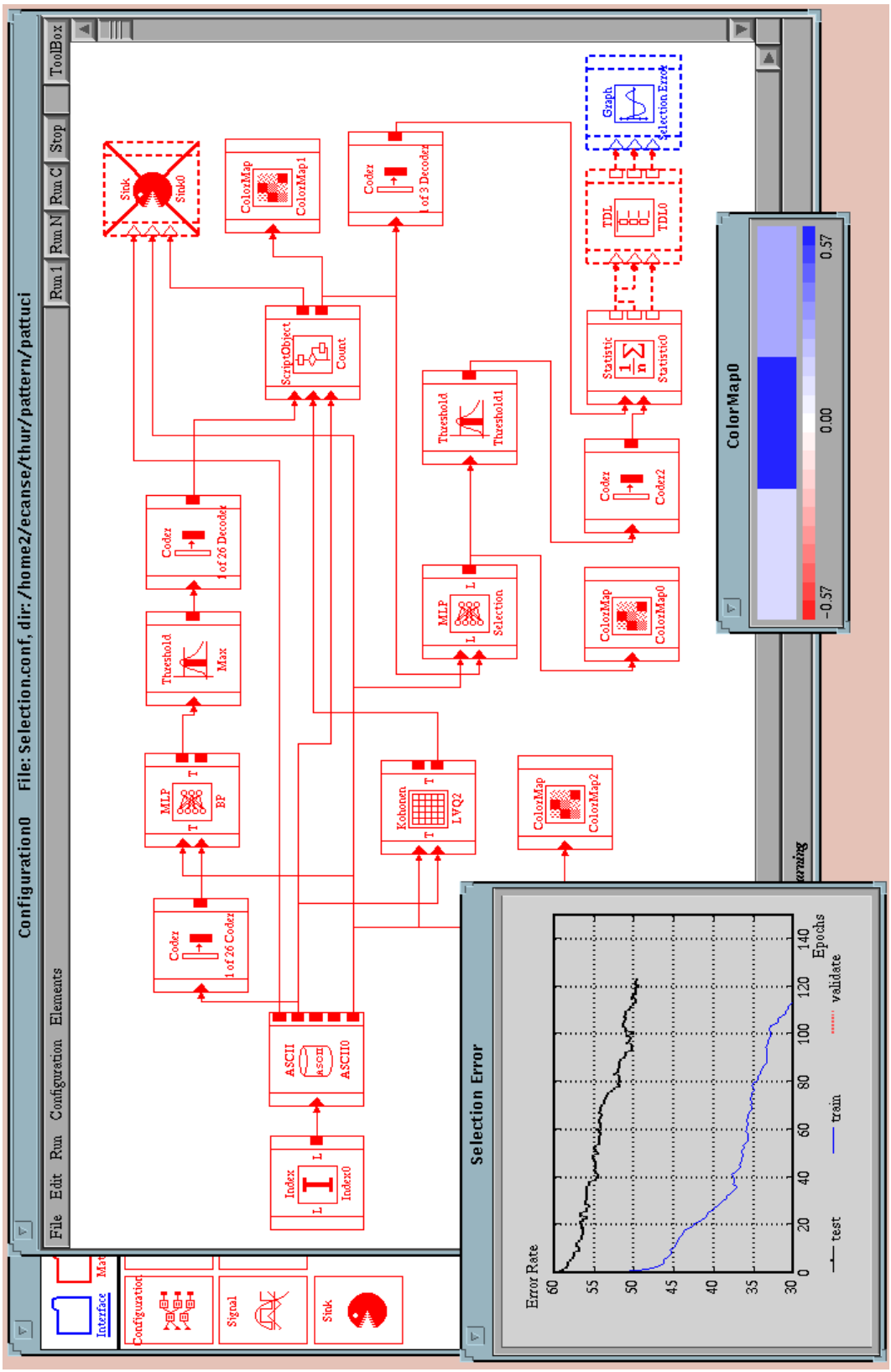
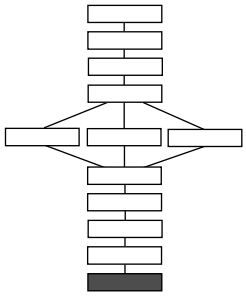


Bild 11.15 *Beispiel für die Simulation mit ECANSE: Kombination redundanter Ergebnisse durch Selektion*



12 Ergebnis und Ausblick

Ein Hauptziel dieser Arbeit war eine umfassende Darstellung von Modularität bei neuronalen Netzen. Um dies zu erreichen, wurden die neuronalen Large Scale-Architekturen strukturiert und überblicksartig analysiert. Diese Gesamtsicht wurde um eine detallierte Darstellung der möglichen Realisierungen für einige statische modulare Architekturen (Selektion, Integration und Mixture Models) ergänzt. Bild 12.1 zeigt nochmals die Strukturierung der modularen Architekturen. Außerdem sind in diesem Bild die detailliert beschriebenen statischen Architekturen aufgelistet.

Die Vorteile der detailliert dargestellten Architekturen wurden durch Experimente demonstriert. Es wurde demonstriert,

- wie durch Verwendung von mehreren kleinen Netzen anstelle eines großen Netzes die für das Training des Gesamtsystems benötigte Zeit (bei gleicher Fehlerrate) auf weniger als ein Viertel reduziert werden kann und
- wie durch Fusion der Ergebnisse von zwei verschiedenen parallel arbeitenden neuronalen Netzen die Fehlerrate des Gesamtsystems fast halbiert werden kann.

Neben dieser stark architekturorientierten Sichtweise wurde in dieser Arbeit auch der anwendungs- und entwicklungsorientierten Betrachtung großer Raum gegeben. Das wesentliche Problem beim Design von modularen Systemen, die Dekomposition der Aufgabe in Teilaufgaben, wurde detailliert analysiert. In Bild 12.2 sind die verschiedenen Möglichkeiten der Dekomposition zusammengefaßt.

Der Schwerpunkt der Analysen lag bei der manuellen Dekomposition und bei Dekomposition durch selbstorganisierende Selektion von Teilnetzen. Experimentell wurde demonstriert,

- wie fix programmiertes Preprocessing auf eine modulare neuronale Architektur abgebildet werden kann, sodaß dadurch das Preprocessing durch Lernen optimiert und damit die Fehlerrate des Gesamtsystems reduziert werden kann.

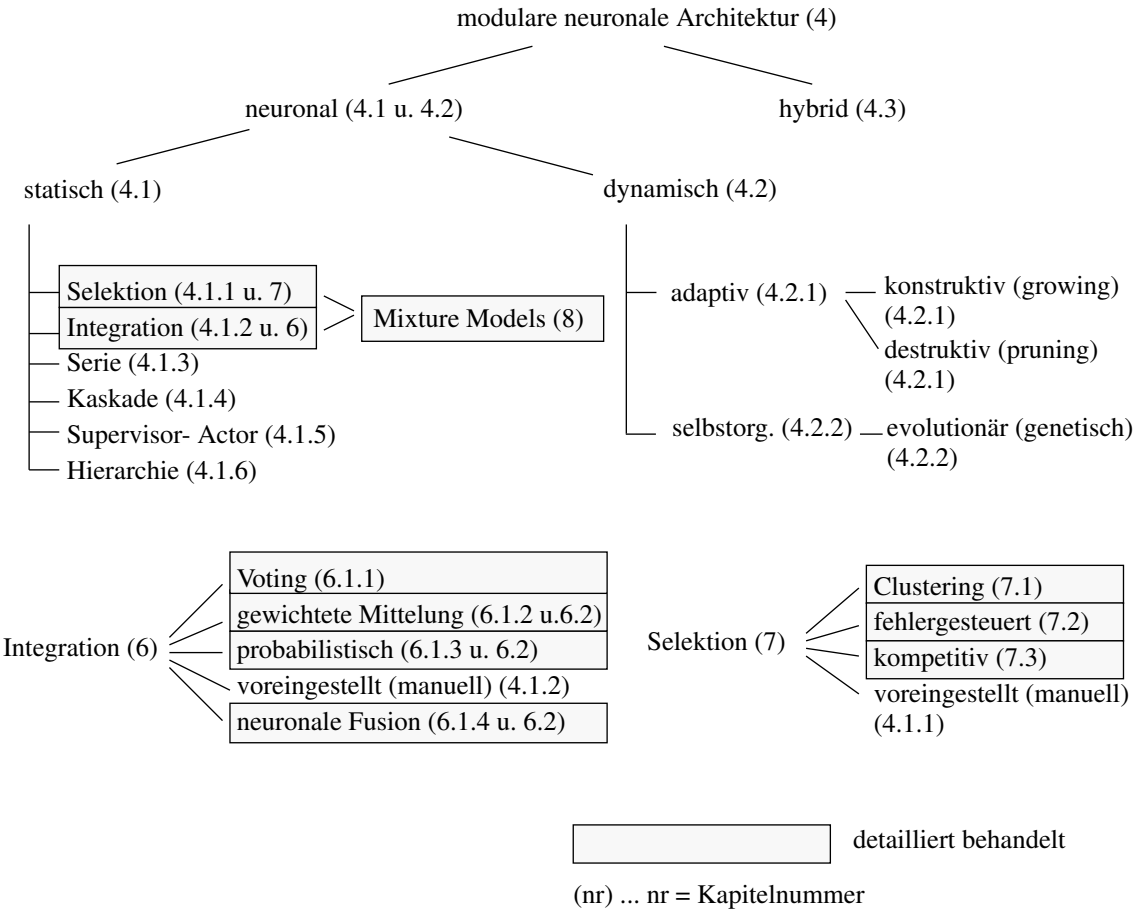


Bild 12.1 *Übersicht über die modularen neuronalen Architekturen*

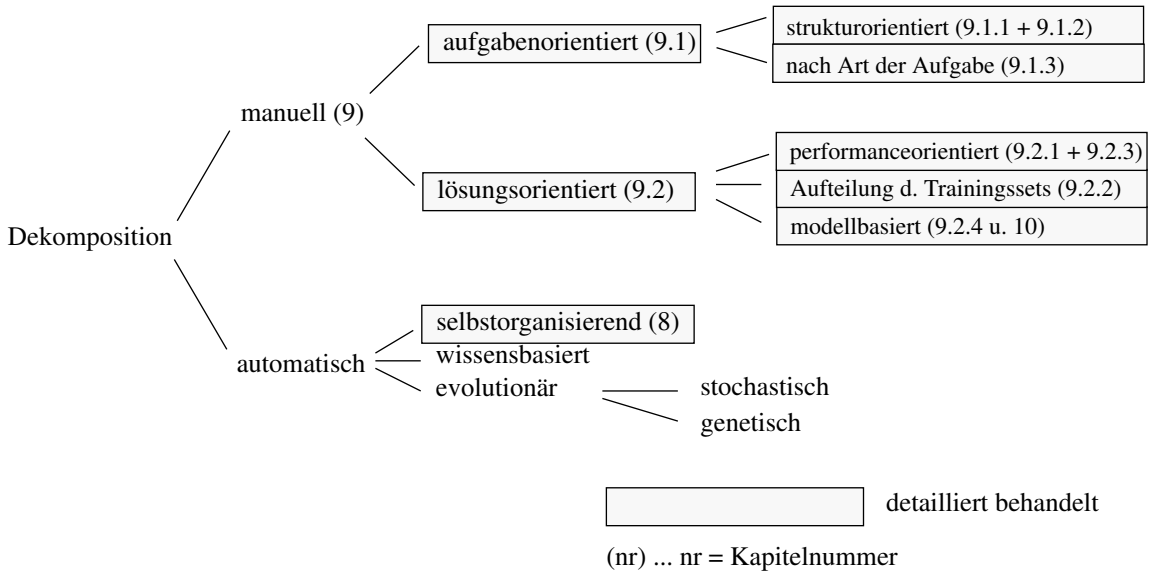


Bild 12.2 *Übersicht über die Dekompositionsmethoden*

12.1 Ausblick

Bei der durchgeführten Analyse der Architekturen lag der Schwerpunkt bei neuronalen Architekturen. Bei der Betrachtung von modell- und algorithmusbasierten neuronalen Netzen in Kapitel 10 hat sich jedoch gezeigt, daß die Abbildung von traditionellen Lösungen auf neuronale Netze die Vorteile des traditionellen Ansatzes mit den Vorteilen des neuronalen Ansatzes verbindet. Darüberhinaus gewinnen hybride Systeme, also Systeme, die aus Modulen verschiedener Technologie bestehen, ganz allgemein an Bedeutung. Aus diesem Grund sollte neben den neuronalen Architekturen auch die Verbindung von neuronalen Netzen mit Modulen anderer Technologien in hybriden Architekturen systematisch analysiert werden.

Ein Teil dieses großen Gebietes ist die automatische Strukturierung (Dekomposition) von neuronalen Systemen unter Verwendung von nichtneuronalen Technologien. Dazu könnten die in Kapitel 4 dargestellten Grundarchitekturen und deren Kombinationen formal beschrieben werden. Aus den Grundarchitekturen entstünden dadurch Konstruktionsprinzipien für komplexe Architekturen. Diese Konstruktionsprinzipien könnten dann für eine automatische Entwicklung oder Optimierung von modularen Architekturen verwendet werden. Besonders die evolutionäre Optimierung der Architektur des modularen neuronalen Systems scheint diesbezüglich ein vielversprechender Ansatz zu sein [Happ94].

12.2 Fazit

Die Nachteile von neuronalen Systemen, wie hohe Trainingszeiten, schlechtes Scalingverhalten und ungenügende Performance können durch Modularisierung entschärft werden. Vielfach ist es beim Design eines neuronalen Systems nicht so entscheidend, den optimalen Lernalgorithmus bzw. das optimale Einzelnetz zu verwenden. Vielmehr gilt für modulare neuronale Systeme die Vorstellung “das Ganze ist mehr als die Summe seiner Teile”. Insbesonders für hochkomplexe Aufgaben gilt, daß eine optimale Modularisierung des Systems notwendig ist, um die Aufgabe optimal zu lösen. Gerade aber die optimale Dekomposition der Aufgabe ist eines der schwierigsten Probleme. Hier sollte für die zukünftige Arbeit ein Schwerpunkt gesetzt werden. Besonders die automatische oder teilautomatische Strukturierung bzw. Optimierung von Systemen in hybriden Architekturen ist dabei ein lohnendes Gebiet.

A Notation

Skalare Größen werden mit Kleinbuchstaben bezeichnet (x, x), (Spalten-) Vektoren mit unterstrichenen Kleinbuchstaben ($\underline{x}$). Großbuchstaben stehen für Mengen (M, U), unterstrichene Großbuchstaben für Matrizen ($\underline{W}$,$\underline{V}$). N ist die Menge der natürlichen Zahlen, $\Re$ die Menge der reellen Zahlen. $\Re^n$ ist die Menge der n-dimensionalen reellen Zahlen und $\Re^{n_i}$ mit i einem Index ist die Menge der n_i-dimensionalen reellen Zahlen.

Die verwendeten Funktionen f_k und f seien immer vektorwertige Funktionen. Statt $w(t+1) = f(w(t))$ mit w aus der Menge W und t der Zeit wird der Einfachheit halber $f: W \rightarrow W$ bzw. $w = f(w)$ geschrieben, statt $f_1: W \rightarrow W$ und $f_2: W \rightarrow O$ wird $f: W \rightarrow W \times O$ geschrieben.

Bei $\underline{y} = n\underline{x}$ ist $\underline{y}$ der Vektor der sich aus elementweiser Multiplikation des Vektors $\underline{x}$ mit dem Skalar n ergibt. $y = \underline{x_1}\underline{x_2} = \langle \underline{x_1}, \underline{x_2} \rangle$ ist das innere Produkt der Vektoren $\underline{x_1}$ und $\underline{x_2}$, auf eine explizite Darstellung der Transponierung von Vektor $\underline{x_2}$ wird verzichtet.

A 1 Variablennamen

In der nun folgenden Tabelle A.1 werden die meisten der verwendeten Variablennamen aufgelistet. Variablen, die nur einmal verwendet wurden und Ausnahmen sind im jeweiligen Kontext beschrieben.

Variablenname	Bedeutung
$\underline{x} \in \Re^n$	Inputvektor eines Moduls bzw. neuronalen Netzes
x_i	i-te Komponente des Inputvektors
i	Index der Komponenten des Inputvektors
n	Dimensionalität des Inputvektors
$\underline{y} \in \Re^m$	Outputvektor eines Moduls bzw. neuronalen Netzes
y_j	j-te Komponente des Outputvektors
j	Index der Komponenten des Outputvektors bzw. Index der Units eines neuronalen Netzes
m	Dimensionalität des Outputvektors

Tabelle A.1 *Variablennamen*

$\underline{l} \in \Re^p$	Vektor der lokalen Daten eines Moduls
l_o	o-te Komponente des Vektors der lokalen Daten
o	Index der Komponenten des Vektors der lokalen Daten bzw. Output einer Unit
p	Dimensionalität des Vektors der lokalen Daten
$\underline{i}$	Inputvektor einer Unit eines neuronalen Netzes
$\underline{y}^{(h)} \in \Re^{m(h)}$	Outputvektor eines hidden Layers eines neuronalen Netzes
$y_h^{(h)}, o_h$	h-te Komponente des Outputvektors eines hidden Layers
h	Index der Komponenten des Outputs eines hidden Layers
z	Dimensionalität des Outputs eines hidden Layers bzw. Qualitäts- (Performance-) Maßzahl für ein NN
$\underline{g} \in \Re^r$	Outputvektor des Gating Netzes eines Mixture Modells
g_k	k-te Komponente des Outputvektors des Gating Netzes
r	Anzahl (paralleler) Teilnetze in einer modularen Architektur
k	Index der Teilnetze in einer modularen Architektur
q	Kardinalität einer Untermenge von q < r Teilnetzen der r Teilnetze einer modularen Architektur bzw. Anzahl an Klassen
I_k	Inputraum des k-ten Moduls
O_k	Outputraum des k-ten Moduls
M	Menge der Module eines modularen Systems
m_a, m_b	zwei kokrete Module
C	Menge der (Namen der) Verbindungen zwischen Modulen
t	Architektur eines modularen Systems
c	Verbindung zwischen zwei Modulen
w_{ij}	Gewicht des i-ten Inputs der Unit j
v_{ik}	Gewicht des i-ten Inputs der Unit k eines hidden Layers oder Gating-Netzes
net_j	Nettoinput der Unit j (meist die gewichtete Summe aller Inputs)
e_j	Fehler der Unit j
u_j	Unit j
f	Propagate-Funktion einer Unit eines neuronalen Netzes
$G(\mu, \phi)$	Normalverteilung mit Mittelwert μ und Streuung ϕ
α	Lernrate eines neuronalen Netzes
$O(n^2)$	Ordnung von n^2 (für die Abschätzung des Rechenaufwandes)

Tabelle A.1 *Variablennamen*

A 2 Abkürzungen

Abkürzung	Bedeutung
NN	Neuronales Netz
BP	Backpropagation
LVQ	Learning Vector Quantization
SCL	Soft Competitive Learning
CUPs	Connection Updates per Second
PCA	Principal Component Analysis

Tabelle B.1 *Abkürzungn*

B Erweiterte Definition eines Moduls

Die Definition eines Moduls von Abschnitt 2.3 basiert auf einer objektorientierten Sicht. Sie ist hinreichend für die theoretische Betrachtung neuronaler Systeme. Für die praktische Realisierung fehlen jedoch einige Teile. Nach einer kurzen Einführung in die objektorientierte Modellierung erfolgt eine erweiterte Definition eines Moduls. Außerdem wird ein statisches Modell eines allgemeinen logischen Moduls vorgestellt, das auch praktische Aspekte berücksichtigt. Im letzten Teil dieses Anhangs wird dann als konkretes Beispiel ein Backpropagation-Netz modelliert.

Die verwendete Terminologie dieses Anhangs ist, sofern nicht explizit anders defniniert, aus [Rumb91] entnommen.

B 1 Objektorientierte Modellierung

Ein objektorientiertes Modell beschreibt ein konkretes modulares System. Es besteht zumindest aus 3 Teilen: einem Datenflußmodell, einem Steuerungsmodell und einem statischen Modell [Rumb91] [Shla92].

B 1.1 Datenflußmodell

Das Datenflußmodell der Aufgabe ist ein Graph, dessen Knoten Module (Objekte) und dessen Kanten Verbindungen (Datenflüsse) zwischen den Modulen sind. Dies entspricht der Definition eines modularen Systems von Abschnitt 2.3. Die im Datenflußmodell enthaltenen Module repräsentieren Unteraufgaben bzw. Teilmodelle des gesamten Systems. Über die Verbindungen tauschen die Module untereinander Daten aus. Alle bisherigen Architekturskizzen sind Darstellungen von Datenflußmodellen.

Das Datenflußmodell eines Moduls (d.h. einer Teilaufgabe) ist ein einziger Knoten des Graphens (siehe z.B. Bild B.1). Ist das Modul strukturiert, d.h. besteht es aus Untermodulen, so ist auch das Datenflußmodell des Moduls ein Graph mit Knoten und Verbindungen (siehe Bild 2.6 auf Seite 11).

Das Datenflußmodell definiert nicht nur, welches Modul mit welchem anderen Modul Daten austauscht, sondern legt auch die Ablaufreihenfolge der Module fest: Module, die im Datenfluß weiter vorne liegen, werden zuerst abgearbeitet. Bei Architekturen,

bei denen der Datenfluß keine eindeutige Ablaufreihenfolge definiert[1], wird die Reihenfolge als zufällig angenommen.

B 1.2 Steuerungsmodell

Das Steuerungsmodell (dynamisches Modell) legt Besonderheiten im Ablauf des Systems fest und gibt Aufschluß über temporäre Zustände bzw. Verhaltensweisen der Module. Es besteht aus modulindividuellen State Event-Modellen und aus dem modulübergreifenden Steuerflußmodell.

Ein **State Event-Modell** ist ein Graph, dessen Knoten Zustände und dessen Kanten Zustandsübergänge sind. Die Zustandsübergänge werden durch die Ausführung von Funktionen des Moduls oder durch Ereignisse (Events) bewirkt. Das State Event-Modell wird durch ein **State Event-Diagramm** visualisiert. **States** definieren Zustände von Modulen. Für neuronale Netze sind dies beispielsweise die Zustände *uninitialized*, *initialized*, *propagating* (testing), *enabled* und *disabled* (siehe Bild B.5). **Events** sind Stimuli an das jeweilige Modul. Stimuli können von anderen Objekten kommen (Nachrichten) oder vom Benutzer (Eingaben).

Das **Steuerflußmodell** ist ein Graph ähnlich dem Datenflußmodell, dessen Knoten Module und dessen Kanten Verbindungen (Kontrollflüsse) zwischen den Modulen sind. Es definiert Quelle und Ziel von Steuersignalen. Mit Steuersignalen werden Events anderen Modulen mitgeteilt und damit der Zustand von anderen Modulen verändert (z.B. von *enable* auf *disable*). Dadurch wird insbesonders die Ablaufreihenfolge der Module für Fälle, in denen aus dem Datenflußdiagramm diese Reihenfolge nicht eindeutig definiert ist, festgelegt.

Das Steuerflußmodell wird durch ein **Steuerflußdiagramm** visualisiert. Für einfache Situationen wird das Steuerflußdiagramm mit dem Datenflußdiagramm kombiniert. Ein Beispiel dazu ist Bild 7.1 zur Selektion von Teilnetzen.

Für ein Neurosystem muß das Steuerflußdiagramm zumindest aus einem Teil für das Training des Systems und einem Teil für den Recall (Normalbetrieb) des Neurosystems bestehen.

B 1.3 Statisches Modell

Das statische Modell des Systems ist ein Graph. Seine Knoten sind Klassen von Objekten und seine Kanten sind Relationen zwischen den Klassen. Das statische Objektmodell beschreibt die statische Struktur des Systems. Eine Klasse beschreibt eine ganze Gruppe von Objekten mit ähnlichen Eigenschaften. Die Relationen sind Assoziationen, Part of Beziehungen und Vererbungen. Neben den Relationen ist eine Klasse definiert durch ihre Daten (Attribute) und ihre Funktionen (Methoden). In Bild B.3 ist ein Beispiel für ein statisches Modell dargestellt.

1. z.B. bei der Integration von parallelen Modulen

B 2 Definition eines Moduls

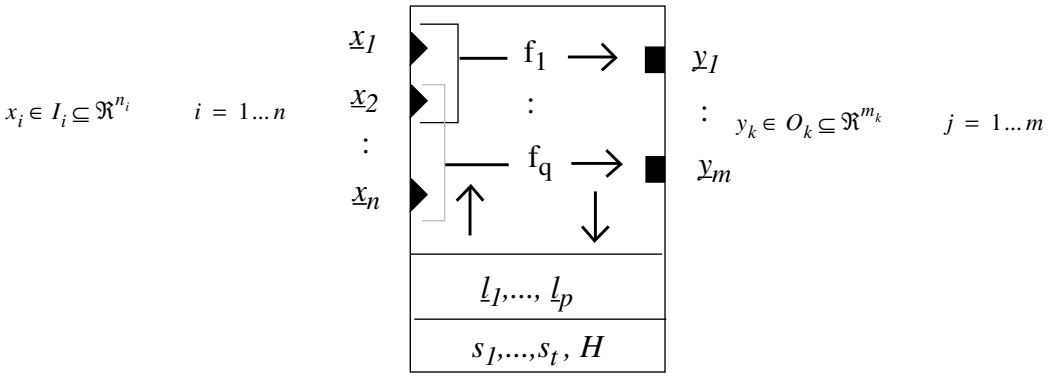
B 2.1 Datenflußmodell

Modul: Ein Modul ist laut Definition von Abschnitt 2.3 ein Objekt, das **n Inputgrößen** $\underline{x}_i \in I_i \subseteq \Re^{n_i} \quad i = 1...n$ auf **m Outputgrößen** $\underline{y}_j \in O_j \subseteq \Re^{m_j} \quad j = 1...m$ abbildet. Mit $I = I_1 \times I_2 \times ... \times I_n$, $O = O_1 \times O_2 \times ... \times O_m$ und den **lokalen Daten** $L = L_1 \times L_2 \times ... \times L_p \quad \underline{l}_o \in L_o \subseteq \Re^{p_o} \quad o = 1...p$ des Moduls, ist diese Abbildung eine vektorwertige Funktion $f: I \times L \rightarrow O \times L$.

Diese Abbildung $f: I \times L \rightarrow O \times L$ der Inputgrößen auf die Outputgrößen kann (modulintern) aus **q** einzelnen **Abbildungen** $f_k: I_k \times L_k \rightarrow O_k \times L_k \quad k = 1...q$ mit $I_k \subseteq I, O_k \subseteq O, L_k \subseteq L$ zusammengesetzt sein.

Der Vorteil der Aufteilung der gesamten Abbildung in q Teilabbildungen liegt darin, daß damit auch nur Teile der Funktionalität des Moduls verwendet werden können.

$$f_k: I_k \times L_k \rightarrow O_k \times L_k \qquad k = 1...q$$



H z.B.

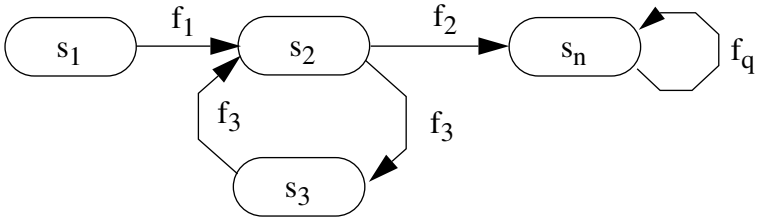


Bild B.1 *Datenflußmodell (erweitert um die lokalen Daten) und Steuerungsmodell eines Moduls*

In Bild B.1 ist solch ein Modul mit q verschiedenen Abbildungen schematisch dargestellt. Im Unterschied zu diesem Modul enthält das Modul von Abschnitt 2.3 (Bild 2.3) nur eine Funktion f. Sie beinhaltet alle einzelnen Funktionen f_1 bis f_q. In diesem Fall

ist die Funktion f eine Art Realisierung des Graphen H und der Zustände s_r. Sie legt die Ablaufreihenfolge der q Teilabbildungen fest.

B 2.2 Steuerungsmodell

Per Definition ist jede der q Abbildungen des Moduls ablauffähig sobald die von ihr benötigten Inputdaten vorhanden sind. Wenn mehrere Abbildungen den selben Output berechnen so entspricht der tatsächliche Output dem Wert der (zufällig) zuletzt berechneten Funktion. Ist eine Ablaufreihenfolge notwendig , so werden Zustände (States) $s_r \in N$ für das Modul definiert. Die Reihenfolge des Ablaufs wird dann durch einen Graph H festgelegt (siehe Bild B.1). Die Knoten des Graphen sind die Zustände s_r bzw. logische Verknüpfungen dieser Zustände. Die Kanten dieses Graphen sind die Funktionen f_k.

Jeder Funktion f_k sind auf diese Weise eine Anzahl an Zuständen zugeordnet in denen sich das Modul befinden muß, damit die Funktion ablaufen darf. Außerdem ist für jeden Ausgangszustand definiert, welcher Zielzustand des Moduls durch Abarbeitung der Funktion f_k erreicht wird.

Der Zustandsübergangsgraph H ist eindeutig, d. h. von einem Ausgangszustand führt eine Funktion nur in einen Zielzustand (Endlicher deterministischer Automat).

Jedes Modul besitzt per Definition mindestens die Zustände *enabled* und *disabled*, die darüber eintscheiden ob überhaupt eine Funktion des Moduls arbeiten darf. Zusätzlich ist jede Funktion individuell aktivierbar bzw. deaktivierbar (funktionsindividuelle Zustände *enabled* und *disabled*). Dadurch können Funktionen die keinen Input benötigen abgeschaltet werden (z.B. die Updatefunktion zur Berechnung der Gewichte von supervised lernenden neuronalen Netzen bildet nur lokale Daten auf lokale Daten ab)

B 2.3 Statisches Modell

Ein Modul besteht aus seinen lokalen Daten l_o, den Inputs $\underline{x}_i$, den Outputs $\underline{y}_j$ und seinen Funktionen f_k. Werden die Daten des Moduls als flach, d.h. intern nicht strukturiert angenommen, so ist das statsiche Modell des Moduls ein einziger Knoten (Bild B.2).

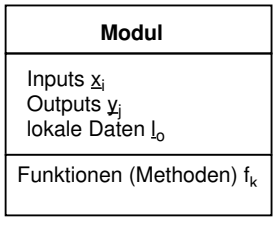


Bild B.2 *Statisches Modell eines Moduls*

B 3 Ein allgemeines, logisches Objektmodell

Die oben dargestellte Definition eines Moduls deckt die Modellierung der Funktion des Moduls für die gegebene Aufgabenstellung ab. Für die reale Anwendung muß das Modul jedoch zusätzliche Fähigkeiten besitzen. Es sind dies die Fähigkeit zum Dialog mit dem Benutzer zur Prametrierung der Module, die Möglichkeit zur Visualisierung von Daten und Abläufen zur Beobachtung des Verhaltens des Moduls und die Fähigkeit die eigenen Daten und Zustandsinformationen abzuspeichern und wieder zu laden. Aus diesem Grund wird ein allgemeines, logisches Modul definiert, das die genannten Fähigkeiten für jedes reale Modul zur Verfügung stellt und außerdem das Problem des Datenaustausches zwischen den Modulen löst. In Bild B.3 ist das statische Modell eines allgemeinen logischen Moduls dargestellt.

Die Basisklasse dieser Hierarchie ist das Element. Ein Element ist ein abstraktes Objekt das seinen Namen kennt und weiß woraus es besteht (aus anderen Elementen) und in welchen Elementen es enthalten ist (es kann zugleich in mehreren anderen Elementen enthalten sein). Jedes Element hat die Fähigkeit sich durch eine Outputbox am Bildschirm zu visualisieren bzw. durch eine Input/Outputbox seine eigenen Daten vom Benutzer einstellen zu lassen. Zur Errichtung eines Dialogs mit dem Benutzer kann das Element eine Dialogbox erzeugen in die es seine eigene Input/Outputbox und alle Input/Outputboxen der im Element enthaltenen Objekte (Elemente) einfügt.

Alle existierenden konkreten Klassen sind von der Basisklasse Element abgeleitet. Im Besonderen sind das die Variablen. Für jeden Typ (int, float, double, Pointer, Array, Referenz, Vektor, Matrix, Spline u.s.w.) gibt es eine eigene Klasse, die ihre eigene Art an Input/Outputbox besitzt. Eine Integer-Variable hat z.B. nur einen einzelnen ganzzahligen Wert. Die In/Outputbox der Integer-Variable ist daher ein einzelnes Eingabefeld für eine Zahl oder graphisch ein Schieberegler oder eine Farbbox. Jedes Element bestimmt selbst wie es aussieht und welche Wertebereiche es zuläßt.

Ein anderes Element ist das AnyObject. Wie alle von Element abgeleiteten Klassen erbt auch das AnyObject die Eigenschaft, daß es aus beliebig vielen anderen Elementen aufgebaut ist. Ein AnyObject kann jedoch einige Elemente aus denen es besteht speziell als Parameter auszeichnen. Dies bedeutet, daß ein AnyObject, wenn es einen Dialog mit dem Benutzer errichtet, nur die Parameter visualisert, nicht aber alle Elemente aus denen es besteht.
Eine konkrete Instanz eines Elements kann Teil mehrerer anderere Elemente sein. Genau ein anderes Element, ein AnyObject, ist jedoch der Besitzer des Elements und entscheidet darüber wann das Element erzeugt bzw. gelöscht werden soll.
Neben den Parametern besteht ein AnyObject auch noch aus seinen Methoden. Eine Methode kann beliebige andere Elemente als Parameter haben und liefert, wenn sie ausgeführt wird, ein Ergebnis. Eine Methode ist ein Element. Sie kann sich deshlab auch selbst visualisieren. Eine Methode ohne Parameter visulisiert sich z.B. duch einen Button.

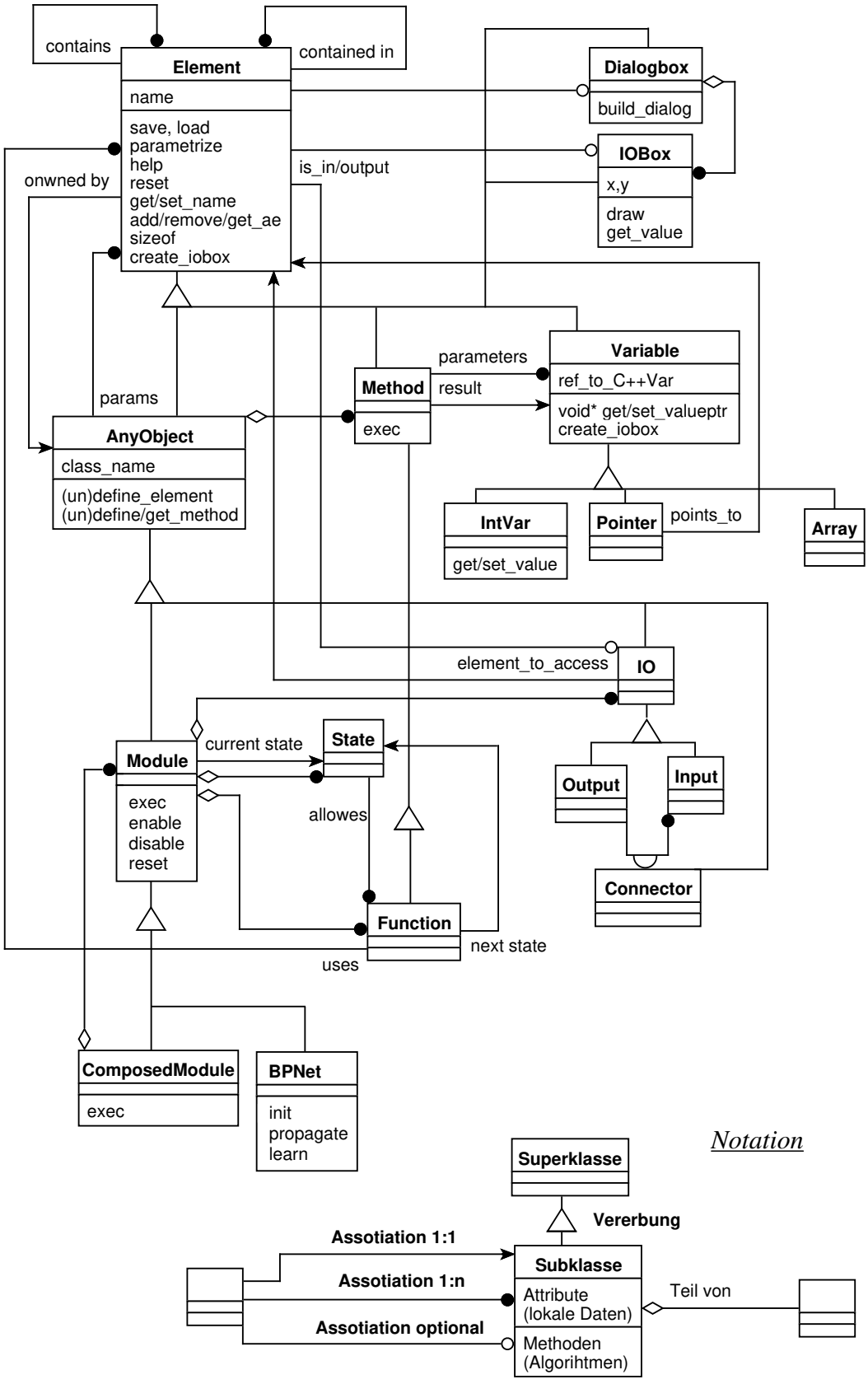


Bild B.3 *Ein statisches Objektmodell für ein allgemeines, logisches Modul*

Eine besondere Art eines AnyObjects ist das Modul. Das Modul besitzt mindestens eine Funktion, lokale Daten und States. Genau ein State ist besonders ausgezeichent. Er ist der aktuelle State des Moduls. Jede Funktion weiß welcher Zustand durch Ausführung der Funktion erreicht wird und welche lokalen Daten die Funktion verwendet. Die lokalen Daten wissen ob sie Input bzw. Output sind. Jeder Zustand enthält eine Liste all jener Funktionen die in diesem Zustand ablaufen dürfen. Inputs, Outputs, Funktionen und Zustände lösen somit gemeinsam (und lokal für jedes Modul) die Synchronisation (Ablaufreihenfolge) im System.

Für hierarchishe Modellierung gibt es das ComposedModule, das selbst ein Modul ist und aus beliebig vielen Modulen besteht.

Zusammengafaßt heißt das: Alle lokalen Daten und alle Funktionen von Modulen, alle Module und alle zusamengesetzten Module sind Elemente die sich apriori selbst visualisieren, mit dem Benutzer kommunizieren und ihre Daten speichern und laden können. Alle Module können zusätzlich untereinader kommunizieren und so einen Datenfluß realisieren der den Ablauf des gesamten Systems steuert. Bei der Modellierung von konkreten Modulen muß damit nur noch der für die Funktionalität der Anwendung relevante Teil (z.B. eines Backpropagation-Netzes, siehe Abschnitt B 4) berücksichtigt werden.

B 4 Definition eines Backpropagation-Moduls

Exemplarisch wird hier ein einfaches dreischichtiges Backpropagation-Netz (Bild B.4) modelliert. Es sei darauf hingewiesen, daß die Funktionen f_k gemeinsam mit den lokalen Daten l_o nicht notwendigerweise neuronale Netze repräsentieren müssen (siehe Definition von “modularem System” auf Seite 7). Daher können auch Filter, Operatoren der Bildverarbeitung und Transformationen u.s.w. mit dieser Notation beschrieben werden.

B 4.1 Datenflußmodell

Die **Inputgrößen** des Backpropagation-Netzes sind

die Eingangsdaten $\underline{x} \in \Re^n$ und

die Sollausgangsdaten für das Netz $\underline{y}_{soll} \in \Re^n$.

Die **Outputgrößen** sind

die Ausgangsdaten $\underline{y} \in \Re^m$ des Netzes und

der Fehlerausgang des Neztes $\underline{\delta}^{(i)} \in \Re^n$, der zur weiteren Rückwärtspropagierung dient[2].

2. Das hochgestellte i deutet an, daß es sich um den Fehler am Inputlayer handelt.

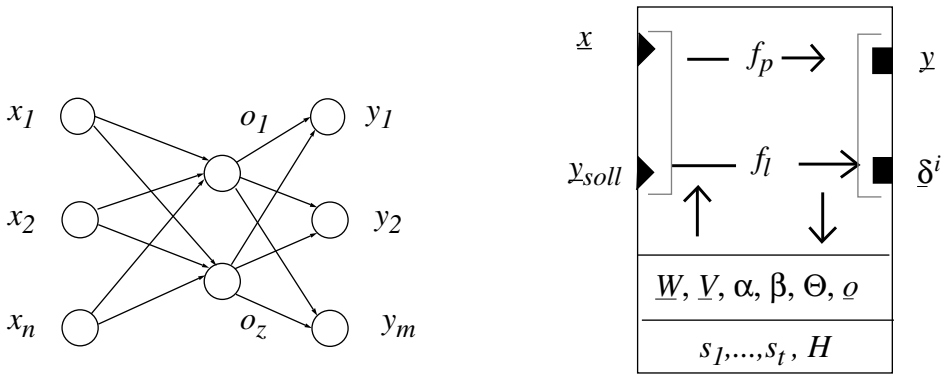


Bild B.4 *Struktur- und erweitertes Datenflußmodell eines dreischichtigen Backpropagation-Netzes*

Die **lokalen Daten** des Backpropagation-Netzes sind:

die Gewichte des hidden Layers $\underline{W}$ mit $w_{ih} \in \Re \quad i = 1...n, \; h = 1...z,$

die Gewichte des output Layers $\underline{V}$ mit $v_{hj} \in \Re \quad h = 1...z \quad j = 1...m,$

die Lernrate $\alpha \in \Re,$

die Momentumsrate $\beta \in \Re$

der Bias $\theta \in [-1,1]$

und die Aktivierungswerte der hidden Units $o_h \in \Re.$

Das Netzwerkmodul beinhaltet drei **Funktionen**:

eine Propagatefunktion $f_p: \underline{x} \rightarrow \underline{y}$,

eine Lernfunktion $f_l: \underline{o}_h \times \underline{y} \times \underline{W} \times \underline{V} \times \underline{y}_{soll} \rightarrow \underline{W} \times \underline{V} \times \underline{\delta}^{(i)}$

und eine Initialisierungsfunktion $f_{init}: \varnothing \rightarrow \underline{W} \times \underline{V}.$

Mit

$$\underline{x} = \begin{bmatrix} x_1 \\ : \\ x_n \end{bmatrix}, \; \underline{y} = \begin{bmatrix} y_1 \\ : \\ y_m \end{bmatrix}, \quad \underline{W} = \begin{bmatrix} w_{11} \; \ldots \; w_{1z} \\ : \qquad : \\ w_{n1} \; \ldots \; w_{nz} \end{bmatrix}, \quad \underline{V} = \begin{bmatrix} v_{11} \; \ldots \; v_{1m} \\ : \qquad : \\ v_{z1} \; \ldots \; v_{zm} \end{bmatrix} \text{ und } \underline{y}_{soll} = \begin{bmatrix} y_{soll\,1} \\ : \\ y_{soll\,m} \end{bmatrix}$$

$$x_i, y_j, w_{ih}, v_{hj}, y_{soll\,j} \in \Re$$

und der sigmoiden Aktivierungsfunktion $f(x) = \frac{1}{1+e^{-x}}$
sind diese Funktionen folgendermaßen definiert:

Propagatefunktion: Sie bildet die Eingangsdaten auf die Ausgangsdaten ab:

$$y_j = f(\sum_{h=1}^{z} v_{hj} o_h + \theta) \ , \ j = 1...m \ \text{ mit } \quad o_h = f(\sum_{i=1}^{n} w_{ih} x_i + \theta) \ , \quad h = 1...z$$

Lernfunktion: Sie repräsentiert ein Gradientenverfahren zur Adaption der Gewichte. Dazu bildet sie die Aktivierungswerte der Units, die Gewichte zum Zeitpunkt t und den Solloutput auf die Gewichte zum Zeitpunkt t+1 ab. Zu beachten ist, daß die Lernfunktion vorraussetzt, daß vor ihrer Abarbeitung alle Aktivierungswerte der Units durch die Propagatefunktion berechnet wurden.

Die Lernfunktion minimiert den quadratischen Fehler
$e = \frac{1}{2} \sum_{j=1}^{m} (y_{soll\,j} - y_j)^2$ mit y_j laut Propagatefunktion.

Mit dem Fehler der j-ten Outputunit $\delta_j^{(o)} = f'(net_j) \frac{\partial}{\partial y_j} e = y_j (1 - y_j) \ (y_{soll\,j} - y_j)$
ist

$v_{hj}(t+1) = v_{hj}(t) + \alpha \delta_j^{(o)} o_h + \beta \ (v_{hj}(t) - v_{hj}(t-1))$ für den Output Layer und mit

$\delta_h^{(h)} = o_h (1 - o_h) \sum_{j=1}^{m} \delta_j^{(o)} v_{hj}$ ist

$w_{ih}(t+1) = w_{ih}(t) + \alpha \delta_h^{(h)} x_i + \beta \ (w_{ih}(t) w_{ih}(t-1))$ für den Hidden Layer.

Es ergibt sich der Fehleroutput des Netzes $\underline{\delta}^{(i)} = \begin{bmatrix} \delta_1^{(i)} \\ \vdots \\ \delta_n^{(i)} \end{bmatrix}$ mit $\delta_i^{(i)} = \sum_{h=1}^{z} \delta_h^{(h)} w_{ih}$.

Eine Variante der Lernfunktion ergibt sich wenn die Berechnung des Fehlers e aus dem Modul herausgenommen wird. Statt dem Solloutput y_{soll} ist dann die Ableitung des Fehlers $-\frac{\partial}{\partial y_j} e = y_{soll\,j} - y_j$ der Input des Moduls. Dadurch kann nicht nur leichter

ein anderer als der quadratische Fehler optimiert werden, sondern vor allem kann der Fehleroutput $\underline{\delta}^{(i)}$ anderer Netze verwedet werden.

Initialisierungsfunktion: Die Funktion f_{init} erzeugt gleichverteilte Zufallszahlen im Intervall [-1,1] und initialisiert damit die Gewichte w_{ih} und v_{hj}.

B 4.2 Steuerflußmodell

Es sind die States: *enabled* und *disabled* (wie bei jedem Modul) und als Substates zu *enabled* die States *uninitialized*, *initialized*, und *propagating* definiert. Das hierarchische State Event-Diagramm für dieses Netz ist in Bild B.5 dargestellt.

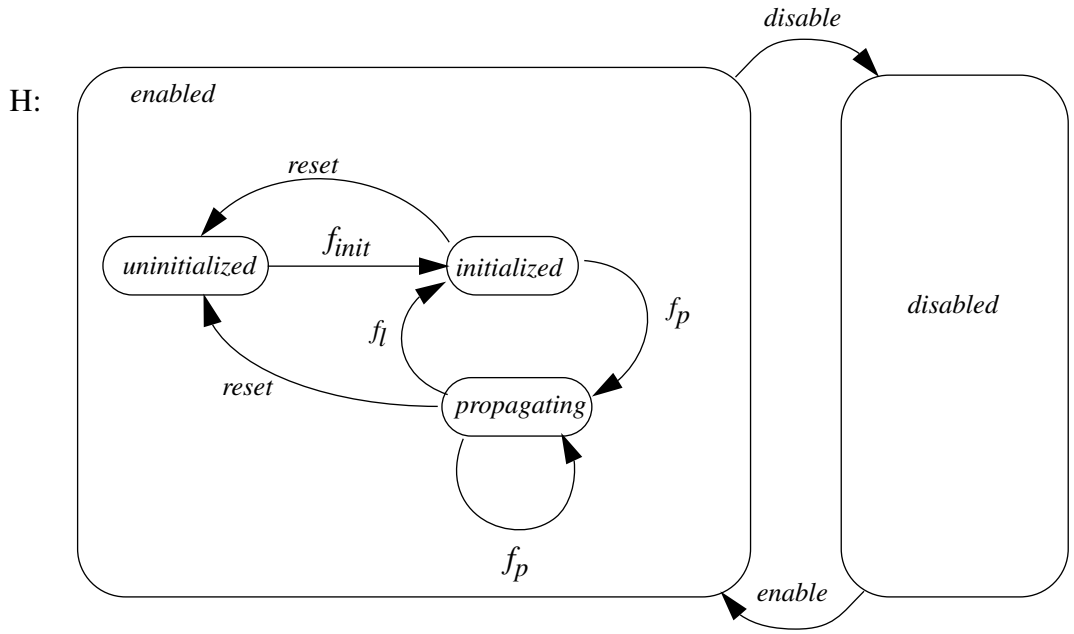


Bild B.5 *State Event Diagramm für einfache neuronale Netze*

Die Funktionen *enable*, *disable* und *reset* dienen nur zum Wechseln der Zustände. Sie sind (bei jedem Modul) zusätzlich zu den modulindividuellen Funktionen f_l, f_p und f_{init} vorgesehen und können durch Benutzereingabe (externes Event) oder von anderen Modulen (Kontrollfluß) aufgerufen werden.

C Performance-Maßzahlen für neuronale Systeme

a) Fehlerrate (maximale, mittlere über mehrere Trainingsversuche, je Klasse c) des trainierten Systems bei Klassifikationsaufgaben

$$e = \frac{\text{Anzahl falsch klassifizierter Inputmuster}}{\text{Gesamtzahl Inputmuster}}, \tag{C.1}$$

b) Rejectrate bei Klassifikationsaufgaben

$$r = \frac{\text{Anzahl als nicht klassifizierbar abgewiesener Inputmuster}}{\text{Gesamtzahl Inputmuster}}, \tag{C.2}$$

c) absoluter Fehler (maximaler, mittlerer) bei Approximationsaufgaben
($i = 1 .. n$, n Anzahl Testvektoren)

$$e_{max} = \max_i \left\| \underline{y}_i - \underline{y}_{i\,soll} \right\|, \tag{C.3}$$

$$e_{mean} = \frac{\sum_i \left\| \underline{y}_i - \underline{y}_{i\,soll} \right\|}{n}, \tag{C.4}$$

d) relativer Fehler (maximaler, mittlerer) bei Approximationsaufgaben

$$e_{max} = \max_i \frac{\left\| \underline{y}_i - \underline{y}_{i\,soll} \right\|}{\left\| \underline{y}_{i\,soll} \right\|}, \tag{C.5}$$

$$e_{mean} = \frac{\sum_i \frac{\left\| \underline{y}_i - \underline{y}_{i\,soll} \right\|}{\left\| \underline{y}_{i\,soll} \right\|}}{n}, \tag{C.6}$$

e) Konvergenzgeschwindigkeit v gemessen anhand der Anzahl benötigter Traingsschritte bis zum Erreichen von 90%, 10%, 0% der minimalen Fehlerrate am Testset und Verlauf des Fehlers $e = e(t)$ während des Trainingspozesses (Bild C.1).

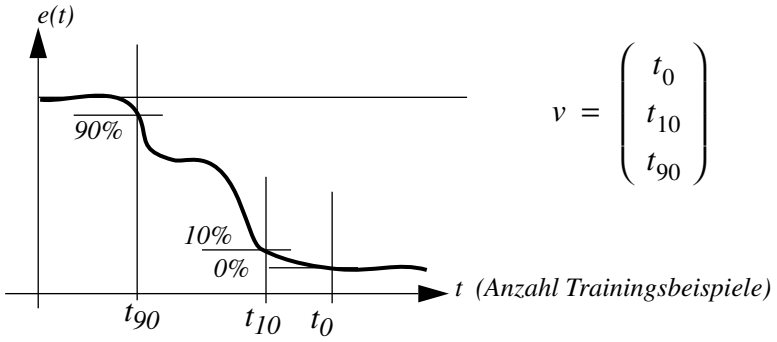


Bild C.1 *Messung der Konvergenzgeschwindigkeit anhand des Verlaufes des Fehlers eines neuronalen Netzes während des Trainings*

f) Robustheit gegenüber Rauschen der Eingangsdaten und/ oder Sollausgangsdaten (gleichverteilt, gaußverteilt, Salz- und Pfeffer Rauschen, additiv, multiplikativ) und Fehler in den Trainingsdaten (zB. fehlerhafte Klassifizierungen).

$$q_x = \frac{e(x)}{e(0)} \quad \text{mit x\% Rauschanteil} \tag{C.7}$$

g) Generalisierungsfähigkeit

$$g = \frac{e_{Training}}{e_{Test}} \tag{C.8}$$

h) Stabilität: Streuung bzw. Varianz des Fehlers bei n Trainingsversuchen

$$s = \frac{\sum_i (e_i^2 - E(e))}{n} \qquad \text{mit} \qquad E(e) = \frac{\sum_i e_i}{n}, \tag{C.9}$$

i) Speicherbedarf in kbyte,

j) Skalierbarkeit: Messung der Abhängigkeit der Konvergenzgeschwindigkeit oder des Speicherbedarfs von der Aufgabengröße. Messung durch Verdopplung der Anzahl an Units, Gewichten, Ein- oder Ausgangsdaten

$$c = \frac{t_{x,n}}{t_{x,2n}} \quad \text{n Anzahl Gewichte, x 0\%, 10\%, 90\% der Restfehlerrate} \tag{C.10}$$

k) **Inkrementelle Trainierbarkeit** (ja, nein)

l) **Komplexität** in Anzahl freier Parameter

m) **Parametrierbarkeit**: Anzahl Parameter des Lernalgorithmus, Sensitivität des erreichten minimalen Fehlers auf Parametervariationen

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Ahm91] S. Ahmad: **VIST: An Efficient Computational Model for Human Visual Attention**; report TR-91-049, International Computer Science Institute, Berkley, 1991.

[Akim93] Y. Akimoto: **Survey on Neural Network Applications to Power Systems in Japan**; Neural Network Applications to Power Systems, ANNPS93, p. 2 - 12, Yokohama, Japan, 1993.

[Barr90] J.M. Barrilleaux, **A biologically motivated algorithm for image interpretation based on multi- pass multi-resolution techniques**, International Joint Conference on Neural Networks, San Diego, California, vol. 2, p. 813 - 818, June 1990.

[Bart83] A.G. Barto, R.S. Sutton, C.W. Anderson: **Neuronlike adaptive elements that can solve difficult learning control problems**, IEEE Transactions on Systems, Man, and Cypernetics SMC-13 p.537-549, 1983.

[Bart93] S. Barth: **Modulare Netze: Buchstabenerkennung unter Verwendung des Ansatzes von Iwata**; Praktikumsbericht am Institut für Automation, Abteilung für Mustererkennung und Bildverarbeitung, TU Wien, 1993.

[Batt94] R. Battiti, A.M. Colla: **Democratcy in Neural Nets: Voting Schemes for Classification, Neural Networks**; vol. 7, no. 4, p. 691-707, 1994.

[Baum93] T. Baumann, A.J. Germond, **Application of the Kohonen Network to Short- Term Load Forecasting**, Second International Forum on Applications of Neural Networks to Power Systems, April 19-22. Yokohama, Japan, 1993.

[Bisc92] H. Bischof: **Neural Networks and Image Pyramids**; in H. Bischof, W. Kropatsch (ed.): Pattern Recognition, Schriftenreihe OCG-62, Oldenburg, p. 249-260, 1992.

[Bisc93] H. Bischof: **Pyramidal Neural Networks**; Ph. D. Thesis, Departement for Pattern Recognition and Image Processing, Technical Univesity Vienna, 1993.

[Boer92] J.W. Boers, H. Kupier: **Biological Metaphors and the design of modular artificial neural networks**; M. thesis, Departments of Computer Science and Experimental and Theoretical Psychology at Leiden University, Leiden, Aug. 1992.

[Bott91] L. Bottou, P. Gallinari: **A Framework for the Cooperation of Learning Algorithms**; Advances in Neural Information Processing Systems 3, p. 781-788, Morgan Kaufmann Publishers, Inc., San Maeto, California.

[Bour88] H. Bourlard, Y. Kamp: **Auto-Association by Multilayer Perceprons and Singular Value Decomposition**; Biol. Cybernetics 59, p. 291-294, 1988.

[Brai91] V. Braitenberg, A. Schütz: **Anatomy of the Cortex - Statistics and Geometry**; Springer, 1991.

[Brid89] J. Bridle: **Propabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition**; In Neuro-computing: Algorithms, Architectures, and Applications; F. Fogleman-Soulie & J. Herault (eds.), Springer Verlag, New York, 1989.

[Brus93] J.Bruske, J. Pauli, G. Sommer: **Dynamic Cell Structures**; internal report, Christian Albrecht University of Kiel - Computer Science Dep., Kiel, Germany, 1993.

[Cael93] T.M. Caelli, et.al.: **Model-based neural networks**; Neural Networks, vol.6, no.5, p. 613- 625. USA, 1993.

[Cher90] A. Chernjavsky, J. Moody: **Note on Development of Modularity in Simlpe Cortical Models**; Advances in Neural Information Processing Systems 2, p. 133-140, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1990

[Cho93] S.B. Cho, J.H. Kim: **Feedforward Neural Networ Architectures for Complex Classification Problems**; technical report, , Center for Artificial Intelligence Research and Computer Science Department, Korea Advanced Institute of Science and Technology 373-1, Korea 1993.

[Cun89] Y.L. Cun et. al.: **Handwritten Digit Recognition: Applications of Neural Network Chips and Automatic Learning**; IEEE Communications Magazine, vol. 27, no. 11, Nov. 1989.

[Cun90] Y.L. Cun, J.S. Denker, S.A. Solla: **Optimal Brain Damage**; Advances in Neural Information Processing Systems 2, p. 598-605, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1990.

[DARP89] DARPA **Neural Network Study**, pp. 34 (figure 2.14-15), AFCEA, 1989.

[Druc93] H. Drucker, R. Schapire, P. Simard: **Improving Performance on Neural Networks Using a Boosting Algorithm**; Advances in Neural Information Processing Systems 5,p. 42-49, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1993.

[Duda73] R.O. Duda, P. E. Hart: **Pattern Classification and Scene Analysis**; John Wiley & Sons; New York, 1978.

[Fahl88] S.E. Fahlman: **Faster-Learning Variations on Back-Propagation: An Empirical Study**; Connectionist Models Summer School 1988, Morgan Kaufmann Publishers. 1988.

[Fahl90] S.E. Fahlman, C. Lebiere: **The Cascade Correlation Learning Architecture**; report CMU-CS-90-100, Scool of Computer Science, Carnegie Mellon University, Pittsburgh, 1990.

[Finn93] W. Finnof, H. Hergert, H. G. Zimmermann: **Improving Model Selection by Nonconvergent Methods**; Neural Networks, vol. 6, p 771-783, 1993.

[Fox91] D. Fox, et. al., **Learning by Error-Driven Decomposition**; International Conference on Artificial Nerual Networks, 1991.

[Frey91] P.W. Frey, D.J. Slate: **Letter Recognition Using Holland-Style Adaptive Classifiers**; Machine Learning vol. 6 no. 2, p. 161-182, March 1991.

[Frie91] J.H. Friedman: **Adaptive Spline Networks**; Advances in Neural Information Processing Systems 3, p. 675-683, Morgan Kaufmann Publishers, Inc., San Maeto, California.

[Frit91] Bernd Fritzke: **Wachsende Zellstrukturen- Ein selbstorganisierendes neuronales Netzwerkmodell**; Arbeitsbericht des Instituts für Mathematische Maschinen und Datenverarbeitung (Informatik), Friedrich Alexander Universität Erlangen, ISSN 0344 - 3515, Erlangen, 1992.

[Fuku84] K. Fukushima, **Neocognitron: A Self-oranizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position**; Biological Cybernetics. 50, p. 377-384, 1984.

[Ghos90] J. Ghosh, R. L. Holmberg: **Multisensor Fusion using Neural Networks**; Second IEEE Symposium on Parallel and Distributed Processing, p. 812-815, Dallas, USA, Dec. 1990.

[Guo92] H.Guo, S.B. Gelfand: **Classification Trees with Neural Network Feature Extraktion**; IEEE Transactions on Neural Networks, vol. 3, no. 6, p. 923 - 933, November 1992.

[Gutk90] M. Gutknecht, R. Pfeifer: **An Approach to Integrate Expert Systems with Connectionist Networks**; AICOM vol. 3, no. 3, p. 116 - 127, Sept. 1990.

[Happ94] B. Happel, J. Murre: **Design and Evolution of Modular Neural Netwok Architectures**, Neural Networks, vol. 7. nos. 6/7, p. 985-1004, 1994.

[Hara69] F. Harary: **Graph Theory**, Addison Wesely, 1969.

[Harp89] S.A. Harp, T. Samad, A.Guha: **Towards the Genetic Sysnthesis of Neural Networks**; International Conference on Genetic Algorithms, p.360 - 370, 1989.

[Herg92] F. Hergert, W.Finnof, H.G. Zimmermann: **Evaluation of Pruning Techniques**; Report of ESPRIT Project 5293 - Galatea, München, August 1992.

[Hert91] J. Hertz, A. Krogh, R. G. Palmer: **Introduction to the theory of neural computaiton**; Addison- Wesley, 1991.

[Hint86] G.E. Hinton, J. L. McCelland, D. E. Rumelhart: **Parallel Distributed Processing;** Vol. I, MIT Press, 1986.

[Hint89] G.E. Hinton: **Connectionist Learning Procedures**: Artificial Intelligence no. 40, p. 185-234, 1989.

[Holl75] J. H. Holland: **Adaptation in natural and artificial systemes**; Ann Arbor, MI: University of Michigan Press, 1975.

[Holl93] J. Hollatz: **Rule-based knowledge in neural computing**; IEE Colloquium on 'Grammatical Inference: Theory, Applications and Alternatives' p. 19/1-8, London, 1993.

[Horn91] K. Hornik: **Approximation Capabilities of Multilayer Feedforward Networks**; Neural Networks, vol. 4, p. 251-257, 1991.

[Hopf88] J. Hopfield, D. Tank: **Computing with Neural Circuits: A Modell**; Science vol. 233, p. 625-632, 1988.

[Hryc92] T. Hrycej: **Modular Learning in Neural Networks, A Modularized Approach to Neural Network Classification**. John Wiley & Sons, Inc. New York 1992.

[Hube62] D.H. Hubel and T.N. Wiesel: **Receptive fields, binocular interaction and functional architecture in the cat's visual cortex**; Journal of physiology, no. 160, p. 106-154. 1962.

[Iwat90] A. Iwata et. al.: **A Large Sacale Neural Network "CombNET" and its Application to Cinese Character Recognition**, International Neural Network Conference, p. 83 - 86, Paris,1990.

[Iwat92] A. Iwata et. al.: **Hand-written Alpha-numeric Recognition by a Self-Growing Neural Network "CombNET-II"**, Proceedings of the International Joint Conference on Neural Networks, vol. IV, pp 228 - 234, 1992.

[Jaco90] R.A.Jacobs, M.I. Jordan, A.G. Barto: **Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks**; COINS technical report 90-27, Departement of Computer Science, University of Massachusetts, March 1990.

[Jaco91] R.A. Jacobs and M.I. Jordan: **A competitive modular connectionist architecture**; Advances in Neural Information Processing Systems 3, p. 767-773, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1991.

[Jord92] M.I. Jordan and R. A. Jacobs: **Hierarchies of adaptive experts**; Advances in Neural Information Processing Systems 4, p. 985-992, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1992.

[Jord94] M.I. Jordan, R. A. Jacobs: **Hierarchical Mixtures of Experts and the EM Algorithm**; Neural Computation, vol. 6, no. 2, p. 181-214, 1994.

[Kend93] G.D. Kendall, T.J. Hall: **Optimal network construction by minimum description length**; Neural Comutations, vol.5, no. 2, p.210-212, March 1993.

[Kins92] J.A. Kinsella: **Comparison and Evaluation of Variants of the Conjugate Gradient Method for Efficient Learning in Feedforward Neural Networks with Backward Error Propagation**; Computation in Neural Systems, vol. 3, no. 1, p. 27-34, Feb. 1992.

[Koho89] T. Kohonen: **Self- Organization and Associative Memory**; 3 rd. ed., Springer, Berlin Heidelberg New York, 1989.

[Koho90] T. Kohonen: **Improved Versions of Learning Vector Quantization**; International Joint Conference on Neural Networks, San Diego, California, vol. 1, p. 545 - 550, June 1990.

[Köhl90] M. Köhle: **Neurale Netze**; Springer Verlag, Wien, 1990.

[Leon92] A. Leonardis: **Image Analysis Using Parametric Models**; Ph. D. Thesis, Faculties of Electrical Engineering and Computer Science, University of Ljubljana, 1992.

[Linc90] W.P. Lincoln, J.Skrzypek: **Synergy of Clustering Multiple Back Propagation Networks**; Advances in Neural Information Processing Systems 2, p. 650-657, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1990.

[Louk94] C. Loukatzikos, J.E. Galletly: **Constructing neural networks from expert system rules**; Tutorial, Journal of Microcomputer Applications, vol. 17, no. 1, p. 35-53, 1994.

[Mare90] A.J. Maren, C.T. Harston, R.M. Pap : **Handbook of Neural Computing Applications**, p. 203-217, Academic Press, Inc., San Diego, California 1990.

[Mart92] T. Martinetz: **Selbstorganisiernende neuronale Netzwerkmodelle zur Bewegungssteuerung**; Verlag: Sankt Augustin: Infix (Dissertationen zur künstlichen Intelligenz; Band 14), 1992.

[Mavr92] M.L. Mavrovouniotis, S. Chang: **Hierarchical Neural Networks**; Computers chem. Engineering, vol. 16, no. 4, p.347-369, April 92.

[Mill90] I. Miller et. al. (ed.): **Neural Networks for Control**; National Science Foundation (U.S.) V. Series, MIT Press, 1990.

[Mint91] D.Mintz: **Robustness by Consensus**; CAR-TR-576, Computer Vision Laboratory, Center for Automation Research, University of Mayland, 1991.

[Mora94] R. Moratz, S. Posch, G. Sagerer: **Controlling Multiple Neural Nets with Semantic Networks**; Mustererkennung 94, p. 288 - 295, Vienna 1994.

[Mori90] Y. Mori and K. Joe: **A Large-Scale Neural Network which Recognizes Handwritten Kanji Characters**; Advances in Neural Information Processing Systems 2, p. 415-422, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1990.

[Murr92] J. Murre: **Learning and categorization in modular neural networks**; Hemel-Hempstead: Harvester Wheatseaf, Hillsdale, 1992.

[Neme94] T. Nemec: **Modulare Netze: Feature-Extraktions-Verfahren als Neurales Netz implementiert**; Praktikumsbericht am Institut für Automa-tion, Abteilung für Mustererkennung und Bildverarbeitung, TU Wien, 1993.

[Nowl90] S.J. Nowlan: **Maximum Likelihood Competitive Learning**; Advances in Neural Information Processing Systems 2, p. 574-582, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1990

[Nowl91] S.J. Nowlan and G.E. Hinton: **Evaluation of Adaptive Mixtures of Competing Experts**; Advances in Neural Information Processing Systems 3, p. 774-780, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1991.

[Paar92] G. Paar: **Hierarchical Texture Classification of Wooden Boards**; in H. Bischof, W. Kropatsch (ed.): Pattern Recognition, Schriftenreihe OCG-62, Oldenburg, p. 261-267, 1992.

[Perl94] L.I. Perlovsky: **A Model-Based Neural Network for Transient Signal Processing**; Neural Networks, vol. 7, no. 3, p. 565-572, 1994.

[Perr94] M.P. Perrorne: **Pulling It All Together: Methods for Combining Nerual Networks**; NIPS 5 postconference workshop proceedings, 1993.

[Pinz90] A.J. Pinz: **Wissensbasierte Informationsverarbeitung in der österreichischen Waldzustandsinventur**; Handbuch der Umwelttechnik'91, S. 228-232, Trend commerz, 1990.

[Pinz94] A.J. Pinz: **Bildverstehen**; Springer Verlag, Vienna 1994.

[Pome93] D.A. Pomerlau: **Input Reconstruction Reliability Estimation**; Advances in Neural Information Processing Systems 5, p. 740 - 747, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1993

[Rama92] U. Ramacher: **Synapse- a neurocomputer that synthesizes neural algorithms on a parallel systolic engine**; Parallel Distributed Computing, vol. 14. no. 3, p. 306-318, March 1992.

[Reed93] R. Reed: **Pruning Algorithms - A Survey**; IEEE Transactions on Neural Networks, vol. 4, no. 5, p. 740 - 747, 1993.

[Rich91] P. Richard et.al., ed. : **Advances in Neural Information Processing Systems 3**, **Section Local Basis Functions**, p. 675-757, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1991.

[Ried93] B. Rieder: **Einführung in die Sportmedizin**; Unterlagen zum Lehrwartkurs für Tae Kwon Do an der Bundesanstalt für Leibeserziehung, S. 99. Innsbruck,1993.

[Rogo94] G. Rogova: **Combining the Results of Several Neural Network Classifiers**; Neural Networks, vol. 7, no. 5, p. 777-781, 1994.

[Roha92] K. Rohani, M.T. Manry: **Nonlinear Neural Network Filters for Image Processing**; ICASSP-92, IEEE International Conference on Acustics, Speech and Signal Processing, vol. 2, p. 373 -376, San Francisco, 23- 26. 3. 1992.

[Rumb91] J. Rumbaugh, et. al.: **Object Oriented Modeling and Desing**; Prentice-Hall, Inc., New Jersey, 1991.

[Rume86] D.E. Rumelhart, J.L. McClelland (Eds.): **Parallel Distributed Processing: Explorations in the Microstructure of Cognition**; Vol.I, Bradford Book/MIT Press, Cambridge, 1986.

[Shad93] R.S. Shadafan, M. Niranjan: **A Dynamic Network Architecture by sequential Partitioning of the input space**; Cambridge University Engineering Department, Trumpington St., Cambridge, CB2 1PZ, England, May 1993.

[Shla92] S. Shlaer, S.J. Mellor: **Object Lifecycles- Modeling the World in States**; Yourdon Press, New Jersey 07632, 1992.

[Schl94] M. Schlang, M. Haft, K. Abraham-Fuchs: **A Comparison of RBF and MLP Networks for Reconstruction of Focal Events from Bioelectric/ Biomagnetic Field Patterns**; Mustererkennung 94, 16. DAGM Symposium und 18. Workshop der ÖAGM, p. 265 - 272, Vienna 1994.

[Smit93] J.R.M. Smits, P. Schoemakers, A. Stehmann, F. Sijstermans, G. Kateman: **Interpretation of infrared spectra with modular neural-network systems**; Chemometrics and Intelligent Laboratory Systems, vol. 18, no. 1, p.27 -39, Elsevier Science Publishers B.V., Amsterdam, Jan. 1993.

[Somm92] I. Sommerville: **Software Engineering**; Addison-Wesley, 1992.

[Tai93] Tai Li, Luyan Fang: **Hierarchical classification and vector quantization with neural trees**; Advances in Neural Information Processing Systems 4, p. 1141- 1147, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1992.

[Taka90] H. Takagi: **Fusion Technology of Fuzzy Theorie and Neural Networks - Survey and Future Directions** (Matsushita). 1st International Conference on Fuzzy Logic and Neural Networks, Iizuka 1990.

[Tana89] K. Tanaka, M. Yamamura, S. Kobayashi: **A connectionist learning with high-order functional networks and its internal representation**; IEEE International Workshop on Tools for Artificial Intelligence, Architectures, Languages and Algorithms, p. 542-547, Farifax, USA, Okt. 1989.

[Ulbr92] C. Ulbricht, et. al.: **Mechanisms for handling sequences with neural networks**; Report TR-92-29, Austrian Res. Inst. Artificial Intelligence, Vienna, Austria, 1992.

[Vinc92] J. M. Vincent, D. J. Myers, R. A. Hutchinson: **Image feature location in multi-resolution images using a hierarchy of multilayer perceptrons**; in R. Linggard et. al., ed., Neural Networks for Vision, Speech and Natural Language,Chapman & Hall, 1992.

[Wan90] E. A. Wan: **Neural network classification: A Bayesian interpretation**; IEEE Transactions on Neural Networks, vol. 1, no. 4, p. 303 - 304

[Werb74] P.J. Werbos: **Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences**; Ph. D. thesis, Harvard University, Cambridge, 1974.

[Wynn92] M. Wynne-Jones: **Node Splitting: A Constructive Algorithm for Feed-Forward Neural Networks**; Advances in Neural Information Processing Systems 4, p. 1072-1079, Morgan Kaufmann Publishers, Inc., San Maeto, California, 1992.

[Zalz91] A.M.S. Zalzala, A.S. Morris: **A neural network approach to adaptive robot control**; International Journal of Neural Networks, vol. 2, no. 1, p. 17- 34, March 1991.