Forschungszentrum Karlsruhe
Technik und Umwelt

Wissenschaftliche Berichte
FZKA 6258

Geometrische Interpretation neuronaler Netze

A. Bernatzki

Hauptabteilung Prozeßdatenverarbeitung und Elektronik

März 1999

Forschungszentrum Karlsruhe
Technik und Umwelt

Wissenschaftliche Berichte

FZKA 6258

Geometrische Interpretation neuronaler Netze

Andreas Bernatzki

Hauptabteilung Prozeßdatenverarbeitung und Elektronik

von der Fakultät für Informatik der Universität Karlsruhe
genehmigte Dissertation

Forschungszentrum Karlsruhe GmbH, Karlsruhe

1999

Als Manuskript gedruckt

Forschungszentrum Karlsruhe GmbH
Postfach 3640, 76021 Karlsruhe

Mitglied der Hermann von Helmholtz-Gemeinschaft
Deutscher Forschungszentren (HGF)

ISSN 0947-8620

Zusammenfassung

Neuronale Netze werden immer häufiger in kommerziellen Anwendungen eingesetzt. Ihr Vorteil liegt darin, daß sie nicht programmiert werden müssen, sondern ein Problem anhand von Beispielen zu lösen lernen. Das selbständige Lernen bringt jedoch Gefahren mit sich. Es ist sehr schwer, ein trainiertes Netz qualitativ zu beurteilen und zu verstehen, wie es das Problem gelöst hat. In vielen Einsatzgebieten wird aber eine gewisse Sicherheit verlangt (z. B. TÜV-Siegel). Die mangelnde Interaktion zwischen dem Menschen und dem Netz erlaubt es nicht, vorhandenes nichtformales Wissen in die Netzstruktur einzubringen. Ein rudimentäres Wissen über jedes Problem ist bei jedem Menschen vorhanden. Die menschliche Wissensakquisition erfolgt durch Beobachtung oder logisches Denken (man hat meistens ein gewisses Gefühl für das Problem). Jeder Mensch kann autodidaktisch fast jede beliebige Handfertigkeit erlernen, ohne ein Experte auf diesem Gebiet zu sein. Je nach persönlichen Vorkenntnissen fällt ihm das Lernen unterschiedlich schwer. Jeder kann aber das, was er gelernt hat, auf eine nichtformale Weise weitergeben, indem er Hinweise formuliert, falsche Alternativen aufzählt usw.

Die geometrische Interpretation ist ein Verfahren, welches dem Entwickler auf intuitive Weise helfen soll, ein trainiertes, vorwärtsgerichtetes neuronales Netz qualitativ zu beurteilen und vorhandenes, meistens unvollständiges, problembezogenes Wissen in die Netzstruktur einzubringen. Die Weitergabe von nichtformalisiertem Wissen ist einer der Grundgedanken, welcher hinter der geometrischen Interpretation steckt. Ein neuronales Netz soll nicht nur anhand von Beispielen angelernt, sondern auch durch die Nutzung des nichtformalen Wissens (in Form von Hinweisen seitens des Entwicklers) weiterentwickelt werden. Es soll möglich sein, dem Netz mitzuteilen, wie ein Problem besser gelöst werden kann, wie Fehler behoben werden können, wie es sich in Ausnahmesituationen verhalten soll oder wie Ausreißer zu behandeln sind. Diese Hilfe braucht nicht für das gesamte Problem, sondern nur für einen gewissen Teilbereich nützlich zu sein. Ein nichtformaler Hinweis kann einem Schüler helfen, einen bestimmten Fehler nicht mehr zu machen. Eine Hilfe für eine Behebung anderer Unzulänglichkeiten ist dieser Hinweis aber nicht. Mit ähnlicher Situation hat man auch bei neuronalen Netzen zu tun. Manchmal arbeitet das Netz im Prinzip korrekt, nur auf bestimmte Eingaben (Ausreißer) reagiert es fehlerhaft. Oft wird verlangt, daß das Netz bestimmte Eingaben sorgfältiger (genauer) als andere bearbeitet, weil diese aus sicherheitsrelevanten Gründen wichtig sind. Für eine zufriedenstellende Funktionsweise müssen diese nichtformalen und nichtvollständigen Informationen in das Netz eingebracht werden.

Die geometrische Interpretation visualisiert das Innenleben eines vorwärtsgerichteten neuronalen Netzes, stellt die (hochdimensionalen) Daten und das Netz intuitiv dar, ermöglicht Einblicke aus verschiedenen Blickwinkeln und eine beliebige Manipulation der Netzstruktur. Sie bietet eine bequeme und interaktive Mensch-Maschinen-Schnittstelle. Bei der Entwicklung wurde besondere Betonung auf die Interpretation klassifizierender Netze gelegt, da diese am häufigsten eingesetzt werden. Die Brauchbarkeit dieser Methode wurde an Netzen gezeigt, die bei der automatischen Auswertung von Ultraschallbildern eingesetzt werden.

Geometric Interpretation of Neural Networks

Abstract

Neural networks are used more and more often in commercial applications. The advantages are apparent in the capability of learning to solve problems by means of examples. However the capability of independent learning brings hazards upon itself. It is very difficult to judge the quality of a trained network and to understand how it has solved the problem. However, a certain degree of safety is demanded in many areas of its application. The weak communication between the user and the network does not allow the transfer of user knowledge into the network structure. Everybody has a rudimentary knowledge of almost any problem. The human acquisition of knowledge occurs through observation or logical reasoning (in most cases, one has a certain sense for the problem). Everyone can achieve almost any manual skill without being an expert in this area. Depending on prior personal knowledge, learning requires different amounts of effort. But by formulating hints or enumerating false alternatives, anyone can pass on in a non formal way what he has learned.

The geometrical interpretation is a technique which is intended to help the developer in an intuitive way to judge the quality of a trained feedforward neural network to introduce existing problems and related knowledge into the network structure. The transfer of non formal knowledge is one of the fundamental ideas that are behind the geometrical interpretation. The neural network should be evolved not only by means of examples, but also by the use of non formal knowledge (in the form of hints on the part of the developer). It should be possible to inform the network about how a problem can be solved in a more intelligent way, how mistakes can be repaired, how it should behave in exception situations, or how it should handle outliers. This aid does not need to be useful for the entire problem but only for a certain proportion. A non formal hint can help a student not to make a specific mistake again. This hint is however no help for the adjustment of other problems. In the case of neural networks one has to deal with a similar situation. Sometimes the network works in principle correctly, but it reacts faulty on certain inputs (outliers). It is often demanded that the network processes specific inputs more accurately than others, since they are important for security reasons. To obtain a satisfactory mode of operation, this non formal and non complete information has to be introduced into the network.

The geometric interpretation visualizes the inner life of a feedforward neural network, intuitively represents the (high dimensional) data and the network. This allows views from different angles and an arbitrary manipulation of the network structure. It offers a comfortable and interactive man-machine interface. During development, special emphasis was put on the interpretation of classifying networks since these are used most frequently. The usability of this method was demonstrated with networks which are utilized for automatical analysis of ultrasonic scans that stem from a pipeline inspection.

Einleitung

Der Hintergrund für diese Arbeit ist eine Anwendung, die Risse in der Rohrwand einer Ölleitung erkennt. Diese Anwendung ist sehr sicherheitsrelevant, weil das Übersehen eines Risses hohe Kosten verursachen kann. Risse sind die Hauptursache für Rohrbrüche. Das bei einem Rohrbruch herausströmende Öl gelangt in das Grundwasser und kann schwerste Umweltschäden verursachen. Deswegen werden Ölleitungen von staatlichen Behörden, wie z. B. dem TÜV, regelmäßig auf Defekte geprüft. Die herkömmlichen Kontrollmethoden beruhen auf Wasserdruckproben und sind deshalb kostenintensiv, da die Ölleitungen für einige Zeit stillgelegt und mit Wasser gefüllt werden müssen. Es fallen dabei große Mengen an Öl-Wasser-Gemisch an, die als Sondermüll entsorgt oder aufbereitet werden müssen. Neuere Verfahren basieren auf zerstörungsfreier Materialprüfung, z. B. mit Ultraschall. Die Entwicklung der geometrischen Interpretation neuronaler Netze (GINN) wurde durch eine Anwendung motiviert, welche eine Rohrwandprüfung mit Ultraschall vornimmt. Diese Anwendung, der sog. *Rißprüfmolch*, dient in dieser Arbeit als ein Beispiel, welches typische sicherheitsrelevante Problemstellungen illustriert. In die Klasse "sicherheitsrelevant" fallen viele wichtige Anwendungen, deren Lösung mit einem neuronalen Netz zwar prinzipiell möglich ist, deren praktischer Einsatz aber aufgrund der ungenügenden Qualitätskontrolle der Lösung scheitert.

Ein Entwickler einer sicherheitsrelevanten Anwendung ist fast immer mit folgender Situation konfrontiert: sowohl das Problemwissen als auch die bis dato gesammelten Meßdaten reichen separat gesehen nicht aus, um eine Lösung zu finden. Nur das Zusammenfügen der beiden Informationsarten kann unter Umständen, d. h. nur dann, wenn die beiden Informationen komplementär sind, zu einer Lösung führen. Die Schwierigkeit dabei ist eine gemeinsame Repräsentation zu finden, welche das Problemwissen und die Information aus den Meßdaten einheitlich und gleichzeitig darstellt. Das Problemwissen liegt entweder in formaler oder nichtformaler (beschreibender) Form vor. Die Information aus den Meßdaten wird durch Merkmalsfilter gewonnen und anschließend modelliert, z. B. mit Hilfe von statistischen oder Neuronale-Netz-Modellen. Die Information liegt also als statistische Größe oder als Netzparametersatz vor. Es gibt bisher keine brauchbaren Methoden, welche die beiden Informationsarten verbinden.

Der Schwerpunkt dieser Arbeit liegt in der Entwicklung einer Methode, welche das Problemwissen mit einem Neuronale-Netze-Modell verbindet und somit den Einsatz von neuronalen Netzen in sicherheitsrelevanten Anwendungen ermöglicht. Die dabei entwickelte Methode interpretiert (geometrisch) ein neuronales Netz, welches mit den verfügbaren Meßdaten trainiert wurde (Wissensextraktion aus den Meßdaten) und erlaubt eine Korrektur der vom Netz gefundenen Lösung durch Ausnutzung des verfügbaren

formalen bzw. nichtformalen Wissens. Diese Korrektur erfolgt interaktiv, indem die geometrische Interpretation des Netzes anhand des Problemwissens (z. B. Formeln, Hinweise, Beschreibungen von Fehlern etc.) manipuliert wird.

Als Beispiel für die Anwendung dieser Methoden wurde ein Teilsystem des Rißprüfmolches entwickelt. Das System erkennt Muster, welche von Schadensstellen (Rissen) stammen. Das Identifizierungsmodul besteht aus einer Vorverarbeitungsstufe, die auf einer Wavelettransformation basiert, sowie aus einem Klassifikator, der mit neuronalen Netzen arbeitet. Diese Netze wurden mit Hilfe des Neuronale-Netze-Debugger (NNDB) entwickelt, welcher sowohl die GINN als auch die interaktive geometrische Manipulation (IGM) implementiert. Die GINN erlaubt das Verstehen und die Kontrolle einer neuronalen Lösung. Die IGM in Verbindung mit GINN bietet eine neuartige Methode, Wissen in ein neuronales Netz zu transferieren. Herkömmliche Systeme übermitteln das Wissen nur durch Lernbeispiele. Sie setzen voraus, daß entweder entsprechende Daten vorliegen, oder sie können künstlich erzeugt bzw. sinnvoll ausgewählt werden. Sie nutzen aber eine der größten Informationsquellen nicht aus: den Menschen. Sein Wissen ist zwar mächtig, kann aber häufig nichtformalisiert werden. GINN und IGM erlauben das Übermitteln von nichtformalen Hinweisen an ein neuronales Netz und das Aufzeigen von Fehlern in einer neuronalen Lösung. Somit ist diese Methode ähnlich derjenigen, welche verwendet wird, um z. B. Handfertigkeiten an andere Menschen weiterzugeben.

Während der Entwicklung stellte sich heraus, daß eine adäquate Darstellung des neuronalen Netzes zusammen mit den Meßdaten für eine korrekte Arbeitsweise der GINN entscheidend ist. Das Problem ist die Darstellung der hohen Eingabedimensionalität des neuronalen Netzes auf einem zweidimensionalen Bildschirm. Die vorgeschlagene Lösung ist eine geschickt gewählte Projektion. Es werden nicht die Originaldaten projiziert, sondern eine mit der Local Principal Component Analysis (LPCA) erzeugte Transformierte dieser Daten. Die LPCA wurde unter Berücksichtigung der Anforderungen, welche die GINN und die IGM stellen, entwickelt. Sie ist eine stückweise lineare Annäherung an eine nichtlineare Principal Component Analysis (NPCA), welche normalerweise eine gute Darstellung hochdimensionaler Räume bietet. Der Vorteil der LPCA gegenüber der NPCA liegt in der Interpretierbarkeit der Darstellung und der schnellen Berechnung, was in einem interaktiven System von sehr großer Bedeutung ist.

Als Folgerung aus der Anwendung der GINN entstand ein Verfahren zur Topologieoptimierung. Aus der Betrachtung der geometrischen Interpretation eines Netzes ist leicht abzulesen, welche Neuronen im Definitionsbereich keinen Beitrag zur Lösung leisten. Sie können aus dem Netz ohne weiteres entfernt werden. Eine Reihe derartiger Optimierungen werden als geometrische Optimierung neuronaler Netze (GONN) bezeichnet.

In den folgenden Kapiteln werden die geometrische Interpretation neuronaler Netze (Kapitel 4), interaktive geometrische Manipulation (Kapitel 5), sowie die LPCA (Kapitel 4.3) und deren Funktionsweise und Einsatztauglichkeit anhand der Anwendung bei der automatischen Rißerkennung (Kapitel 7) beschrieben. Der NNDB, welcher diese Werkzeuge implementiert, wird in Kapitel 6 vorgestellt.

Kapitel 1

Kurze Einführung in Neuronale Netze

Für ein besseres Verständnis der nachfolgenden Kapitel wird hier eine kurze Einführung in die Thematik neuronaler Netze gegeben. Diese Einführung erhebt keinen Anspruch auf Vollständigkeit, sondern erklärt lediglich die Begriffe, welche in dieser Dissertation benutzt werden. Interessierte Leser finden eine Reihe hervorragender Einführungen in dieses Gebiet in der einschlägigen Literatur [Hecht89], [Zell94] und [Schöneburg90].

1.1 Netztypen

Es gibt eine Vielzahl von Typen neuronaler Netze, die in weitere Untertypen aufgeteilt werden können. Die Aufteilung richtet sich meistens nach gewissen Kriterien, welche aber in der Literatur nicht einheitlich gehandhabt werden. In dieser Arbeit wurde die Typenaufteilung im Hinblick auf die Anwendung in der GINN und der IGM vorgenommen. Im folgenden werden fünf Kriterien angegeben, nach denen neuronale Netze klassifiziert werden können.

Das erste Kriterium unterscheidet rückgekoppelte und rückkopplungsfreie Netze. Zu den rückgekoppelten zählt z. B. das Hopfield-Netz [Hopfield84]. Eine Rückkopplung sorgt dafür, daß sich das Netz in einen stabilen Zustand einschwingt. Bei den rückkopplungsfreien wird die Information von der Eingabe bis zur Ausgabe schrittweise bearbeitet. Lediglich zwischen Neuronen einer Schicht darf es Querverbindungen geben, um z. B. das Neuron mit der höchsten Ausgabe zu ermitteln (winner-takes-all). Zu den rückkopplungsfreien Netzen zählen die vorwärtsgerichteten Netze (englisch feedforward, insbesondere die Multilayer-Perceptrons (MLP) [Rosenblatt58]), in denen der Datenfluß nur in eine Richtung erfolgt, nämlich vorwärts von der Eingabe bis zur Ausgabe. Daraus ergibt sich eine einfache und klare Struktur. Kohonen-Netze zählen ebenfalls zu den rückkopplungsfreien. Sie besitzen aber einen 'winner-takes-all-Mechanismus'. Die GINN und IGM kann nur Netze ohne Rückkopplung bearbeiten.

Das zweite Kriterium unterscheidet zwischen Netzen, die eine Lösung selber finden können, und solchen, welche für das Lernen einen Lehrer benötigen, der ihm entsprechende klassifizierte Trainingsbeispiele zur Verfügung stellt. Zu der ersten Gruppe (mit dem sog. *nicht überwachten Lernen*) gehört beispielsweise das Kohonen-Netz, zu der zweiten (mit dem sog. *überwachten Lernen*) der Multilayer-Perceptron. Für die GINN und IGM spielt das Lernverfahren keine Rolle. Allerdings ist es für die Interpretation vorteilhaft, wenn die Lernbeispiele klassifiziert sind.

Das dritte Kriterium betrifft die Funktionsweise der Neuronen. Jedes Neuron kann eine beliebige *Aktivierungsfunktion* haben. In künstlichen neuronalen Netzen werden allerdings nur Funktionen aus bestimmten Funktionsklassen eingesetzt. Zu diesen zählt z. B. die Klasse der sigmoiden Funktionen oder die der Gaußglockenfunktionen. Die GINN und IGM können vom Prinzip her jede dieser Funktionsklassen bearbeiten. In der ersten Version wird allerdings eine Beschränkung auf die logistischen sigmoiden Funktionen gemacht.

Ein weiteres Kriterium betrifft die Anzahl der sog. *neuronalen Schichten*. Wie in Kapitel 1.4 gezeigt wird, können MLPs mit mindestens einer verdeckten Schicht alle "normalen" Aufgaben lösen. Deswegen sind Netze mit einer verdeckten Schicht besonders interessant, da sie eine einfache Topologie haben und trotzdem allgemein einsetzbar sind. Die GINN und IGM wurden primär für MLPs mit nur einer verdeckten Schicht entwickelt und erst dann auf mehrere Schichten ausgedehnt.

Das letzte Kriterium betrifft weniger das neuronale Netz selbst, sondern mehr die Anwendung, in welcher es eingesetzt wird. Die Ausgabe eines Netzes kann entweder als eine reellwertige Zahl interpretiert werden (allen möglichen Zahlen zwischen *0* und *1* wird eine Bedeutung zugeschrieben), oder es können nur bestimmte Ausgabewertebereiche von Bedeutung sein. Im ersten Fall spricht man von einem approximierenden Netz, im zweiten von einem klassifizierenden. Die GINN und IGM eignen sich in ihrer ersten Version nur für klassifizierende Netze. In dieser Arbeit wird eine noch strengere Einschränkung gemacht, nämlich es können klassifizierende neuronale Netze mit GINN und IGM untersucht werden, die mit jedem Ausgabeneuron eine Klassenaufteilung in nur zwei Klassen vornehmen.

Zusammenfassend kann man sagen, daß die geometrische Interpretation und die interaktive geometrische Manipulation auf vorwärtsgerichtete, klassifizierende neuronale Netze mit einer logistischen sigmoiden Aktivierungsfunktion anwendbar sind. Eine Untersuchung[1] des Instituts für Mikroelektronik Stuttgart (IMS) zeigt, daß diese Einschränkung berechtigt ist. Etwa *57%* aller neuronalen Anwendungen verwenden MLPs, weitere *14.9%* verwenden Kohonen-Netze. Einen Anteil von *4.5%* erreichen Radial-Basis-Function-Netze (RBF-Netze), die zu den vorwärtsgerichteten neuronalen Netzen zählen. Time-Division-Netze (TD-Netze sind ebenfalls vorwärtsgerichtet) sind mit *1.3%* vertreten. Zusammen ergibt sich eine deutliche Mehrheit für vorwärtsgerichtete Netze von *77.3%*.

TABELLE 1. "Marktanteile" verschiedener Netztopologien

Topologie	"Marktanteil"
Multilayer Perceptron	*56.6%*
Kohonen-Map	*14.9%*
Hopfield	*8.4%*
Recurrent	*8.4%*
RBF	*4.5%*
ART-MAP	*3.9%*
TD-Netze	*1.3%*
Cascade-Correlation	*1.3%*
Counterpropagation	*0.6%*

1. Untersucht wurden 154 Applikationen neuronaler Netze aus der Literatur.

Zwecks besserer Erklärung werden in dieser Arbeit nur vorwärtsgerichtete Multilayer Perceptrons behandelt. Deshalb werden als erstes die wichtigsten Merkmale und die Funktionsweise eines Multilayer Perceptrons beschrieben.

1.2 Multilayer Perceptron

Das Multilayer Perceptron (MLP) ist das einfachste neuronale Netz, das sehr häufig eingesetzt wird. Aufgrund seiner einfachen Topologie sind die Ausführungszeiten sehr kurz.

1.2.1 Topologie

Als Topologie (oder Architektur) eines neuronalen Netzes wird die Anordnung der Neuronen zusammen mit der Verbindungsstruktur bezeichnet. Multilayer Perceptrons bestehen aus durchnumerierten Schichten von geordneten Neuronen. Somit hat jedes Neuron einen festen Platz in der Struktur. Die Schichten sind funktional angeordnet. Die Eingabeneuronen sind in der ersten, sog. *Eingabeschicht*, und die Ausgabeneuronen in der letzten, sog. *Ausgabeschicht*, angeordnet. Die *verdeckten Neuronen* (falls vorhanden), sind dazwischen in einer oder mehreren verdeckten Schichten gruppiert. Neuronen dürfen nur auf eine bestimmte Weise miteinander verbunden sein. Die Signalverbindungen laufen ausschließlich von Neuronen einer Schicht mit niedrigerer Nummer zu Neuronen von Schichten mit höheren Nummern. Rückkopplungen sind nicht erlaubt. Alle Verbindungen weisen also in die Richtung der Ausgabeschicht. Sogenannte „Shortcuts", d. h. das Überspringen einer oder mehrerer verdeckter Schichten, sind ausdrücklich erlaubt. Über ein verzweigtes Netz gewichteter Verbindungen empfangen alle verdeckten und alle Ausgabeneuronen von anderen Neuronen im Netz Signale. Die Eingabeneuronen nehmen ausschließlich Signale von der Außenwelt entgegen.

Beispiel 1.1. Abbildung 1.1 zeigt ein MLP mit einer verdeckten Schicht. Der Eingaberaum des Netzes ist dreidimensional, d. h. alle Eingabemuster besitzen drei Merkmale. Der Ausgaberaum ist eindimensional. In dieser Darstellung sind die Neuronen durch Kreise und die Signalverbindungen durch gerichtete Pfeile zwischen den Neuronen dargestellt. Die Eingabeneuronen sind durch kleinere Kreise dargestellt, da sie lediglich als Datenspeicher dienen. Sie führen keine Berechnungen durch. ■

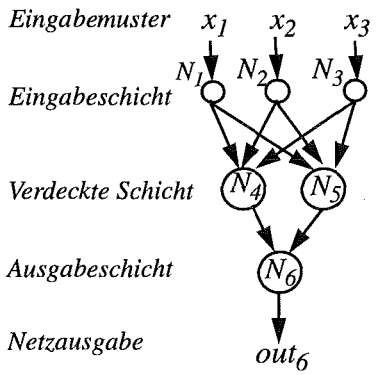


Abbildung 1.1: *Ein vorwärtsgerichtetes Netz mit einer verdeckten Schicht.*

1.2.2 Neuronentypen

Die in dieser Arbeit betrachteten Multilayer Perceptrons sind einfache neuronale Modelle, deren kleinste Einheit das Neuron ist. Um die Arbeitsweise eines MLPs zu verstehen, sollte man zuerst die Arbeitsweise eines Neurons begreifen. Ein einziges Neuron kann auch als ein kleinstmögliches Netz aufgefaßt werden.

Ein Neuron N_j ist die kleinste Verarbeitungseinheit eines MLPs. In einem Neuron treffen Informationen von anderen Neuronen (oder von der Außenwelt) zusammen. Sie werden zu einem Ausgabesignal verarbeitet, welches die Antwort des Neurons auf die Eingabe ist. Jedes Neuron reagiert unterschiedlich auf die Eingabe. Seine Ausgabe wird

als Antwort (sog. *Netzantwort*) des Neurons auf die Eingabe (sog. *Netzeingabe*) net_j bezeichnet.

Im Laufe der Zeit wurden verschiedene Neuronentypen entwickelt. Im folgenden werden die wichtigsten davon chronologisch beschrieben.

1.2.2.1 McCulloch und Pitts Neuron

Die ersten Modelle neuronaler Netze gehen bereits auf die vierziger Jahre zurück. Damals schlugen McCulloch und Pitts erstmals vor, ein Neuron als ein logisches Element mit zwei möglichen Zuständen zu beschreiben. Ein solches Element besitzt n Eingangsleitungen und eine Ausgangsleitung. Eine Eingangsleitung ist entweder aktiv (Pegel "*1*") oder ruhig (Pegel "*0*"). Die Eingabe ist also ein binäres Muster. Die Ausgabe des Neurons ist ebenfalls ein binäres Signal. Die jeweils an einem Eingang anliegende binäre Eingabe wird mit einem Wert $w_{ij} \in \{-1, 0, 1\}$ gewichtet. Die Indizierung $w_{Senke\ Quelle}$ gibt an, zwischen welchem Eingabeelement und welchem Neuron das Gewicht verläuft. Ist das Gewicht gleich *0*, dann sind die Eingabe *Quelle* und das Neuron *Senke* nicht verbunden. Zwecks besserer Lesbarkeit werden oft die Gewichte als ein Vektor geschrieben, z. B. alle Gewichte, welche zum Neuron j gehen, können als ein Gewichtsvektor $W_j = (w_{j1}, w_{j2}, ..., w_{jn})$ geschrieben werden. Die Berechnung des Ausgabesignals erfolgt in zwei Rechenschritten. Im ersten Schritt wird die Netzaktivierung net_j berechnet:

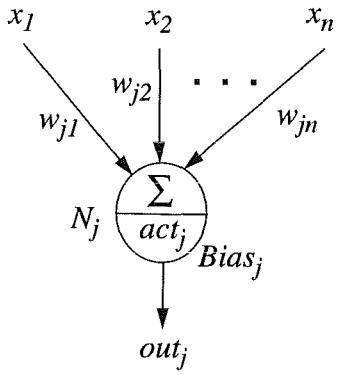


Abbildung 1.2: *Darstellung eines Neurons.*

$$net_j = \left(\sum_{i=1}^{n} w_{ji} x_i\right) - Bias_j. \tag{1.1}$$

Sie gibt an, wie die einzelnen Eingangssignale miteinander verknüpft sind. Dabei ist der $Bias_j$ eine Konstante und kann als ein Signal aufgefaßt werden, welches immer vorhanden ist. Im zweiten Schritt wird entschieden, wie die Antwort des Neurons aussieht, d. h. ob das Neuron feuert (eine *1* sendet) oder nicht (eine *0* sendet). Dazu wird das Ausgabesignal out_j gemäß der Aktivierungsfunktion act_j berechnet:

$$out_j = act_j(net_j). \tag{1.2}$$

Im Fall eines McCulloch und Pitts Neurons kann die Aktivierungsfunktion durch die folgende Regel beschrieben werden: Erreicht die Netzaktivierung eine vorgegebene Schwelle (den *Bias*), dann feuert das Neuron eine *1*. Ansonsten liegt eine *0* am Ausgang (s. Abbildung 1.3c. McCulloch und Pitts zeigten, daß mit Hilfe ihrer Neuronen alle logischen Funktionen realisierbar sind. Die Theorie von McCulloch und Pitts hat aber zwei wesentliche Schwächen: Zum einen läßt sich nicht erkennen, wie die Schaltpläne für die Neuronen zustande kommen, insbesondere wie dies durch Lernen geschehen kann. Zum anderen sind derartige Netze auf das fehlerfreie Funktionieren aller Komponenten angewiesen [Ritter90]. Ein verbessertes Modell eines Neurons, welches die beiden oben genannten Schwächen nicht hat, ist das Perceptron.

1.2.2.2 Das Perceptron

Im Jahre 1958 konstruierte F. Rosenblatt das Mark-I-Perceptron [Rosenblatt58], ein neuronales Netz für Mustererkennungsaufgaben. Im Unterschied zu McCulloch und

Pitts Neuronen verarbeitet ein Perceptron reelle Eingaben und hat reelle Gewichte. Die Ausgabe ist, wie beim McCulloch und Pitts Neuron, aufgrund der verwendeten stufenartigen Aktivierungsfunktion weiterhin binär. Die Berechnung der Ausgabe erfolgt nach denselben Formeln wie beim McCulloch und Pitts Neuron (s. Gleichung (1.1) und (1.2)).

Der größte Fortschritt des Perceptrons gegenüber dem McCulloch und Pitts Neuron liegt in seiner Lernfähigkeit. Das Perceptron lernt aus Beispielen die Gewichte w_{ij} so zu setzen, daß die gefundene Lösung bezüglich eines Fehlermaßes optimiert wird. Aufgrund der großen Anzahl von Parametern (Gewichten) findet ein Perceptron in der Regel nur eine suboptimale Lösung. Die Existenz von Gewichten w_{ij}, die eine Lösung darstellen, ist nicht immer gegeben. Das hat Minsky in seiner Arbeit "Perceptrons" [Minsky69] erstmals am Beispiel der Exklusiv-ODER-Funktion gezeigt. Er bewies, daß das Perceptron die Exklusiv-ODER-Funktion nicht realisieren kann. Der Grund dafür liegt in der Tatsache, daß ein Perceptron lediglich eine Aufteilung des Eingaberaumes mit einer Geraden vornehmen kann (lineare Separierung).

Perceptrons wurden in den Anfangszeiten der Neuroinformatik eingesetzt. Durch die stufenartige Aktivierungsfunktion entstehen aber große Einschränkungen, was die Leistung eines Netzes angeht. Ein allgemeines Neuron hat diese Einschränkungen nicht.

1.2.2.3 Das allgemeine Neuron

In modernen neuronalen Netzen kommen Neuronen zum Einsatz, welche eine reelle Eingabe, reelle Ausgabe und reelle Gewichte haben. Jedes Neuron darf eine beliebige Aktivierungsfunktion act_j haben. Manche Lernalgorithmen verlangen allerdings sigmoide Funktionen (sigmoid bedeutet S-förmig, s. Abbildung 1.3), andere, wie z. B. Backpropagation postulieren noch zusätzlich Differenzierbarkeit und Monotonie. Die am häufigsten verwendete sigmoide Aktivierungsfunktion ist die logistische Funktion. Ihre Formel lautet:

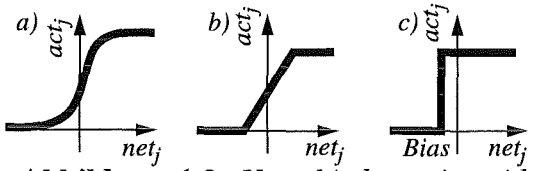


Abbildung 1.3: *Verschiedene sigmoide Aktivierungsfunktionen: a): Die logistische Funktion, b): Die stückweise lineare Funktion, c): Die Stufenfunktion.*

$$act_j = \frac{1}{1 + e^{net_j}}. \tag{1.3}$$

Sie wird in allen Neuronen verwendet, welche in dieser Arbeit betrachtet werden. Abbildung 1.3a zeigt ihren Funktionsverlauf.

1.2.3 Arbeitsweise

Ein vorwärtsgerichtetes Netz (ein MLP) hat den Vorteil, daß die Berechnung schichtweise, in wohldefinierten Schritten, erfolgt. Die Informationsverarbeitung wird anhand einer neuronalen Implementierung der binären Exklusiv-ODER-Funktion erläutert. Abbildung 1.4 zeigt das entsprechende Netz.

Die Merkmale x_1 und x_2 der Eingabemuster finden sich als Ausgangsaktivitäten der Eingabeneuronen N_1 und N_2 wieder. Nach einer Multiplikation mit den Gewichten w_{31} und w_{32} bzw. w_{41} und w_{42} dienen diese als Eingaben für die Neuronen N_3 bzw. N_4 der ersten verdeckten Schicht. In jedem dieser zwei Neuronen wird zuerst die Netzaktivierung net_3 bzw. net_4 nach der Formel (1.1) berechnet und anschliessend die Neuronenausgabe out_3 bzw. out_4 nach der Formel (1.2) bestimmt. Die Ausgaben der Neuronen N_3 und N_4 dienen, nach einer Gewichtung mit w_{53} und w_{54}, als Eingabe für das Ausgabeneuron N_5. Die Netzausgabe wird dann schließlich mit den Formeln (1.1) und (1.2), angewandt auf das Ausgabeneuron N_5, berechnet. In mehrschichtigen Netzen setzt sich dieser Vorgang für jede weitere Schicht fort.

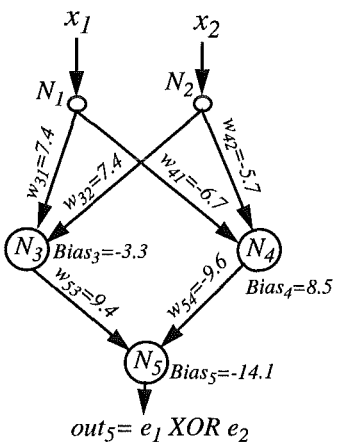


Abbildung 1.4: *Netz zur Realisierung der XOR-Funktion.*

1.2.4 Das Lernen

Das Wissen eines Netzes ist in den Netzparametern (Gewichten und Bias-Werten) sowie in der Netztopologie gespeichert. Die Netztopologie muß vor dem Beginn des Lernens festgelegt werden[1]. Ein Netz lernt ein Problem zu lösen, indem es anhand von Beispielen und durch Anwendung einer Lernregel die Parameter so ändert, daß das gegebene Problem möglichst gut (hinsichtlich eines Fehlermaßes) modelliert wird. Die Programmierung kann also in zwei Schritte aufgeteilt werden:

- Auswahl der Netztopologie und
- Anwendung einer Lernregel.

Es gibt eine Vielzahl von Lernregeln für vorwärtsgerichtete neuronale Netze. Die meisten davon sind eine Spezialisierung bzw. Erweiterung der sog. *Hebbschen Lernregel.* Eine gute Beschreibung viele Lernalgorithmen findet man z. B. in [Zell94].

Grob gesehen kann man den Lernprozeß folgendermaßen beschreiben: Zum Beginn werden die Netzparameter initialisiert, z. B. mit Zufallswerten. Während des Lernprozesses werden dem Netz Lernbeispiele präsentiert. Beim überwachten Lernen sind es Paare von Ein- und Ausgabemustern. Das Eingabemuster gelangt an die Eingabe und das Ausgabemuster wird mit der Netzausgabe verglichen. Der dabei entstandene Fehler liefert dem Lernalgorithmus eine Information über die Richtung der Netzparameteränderung. Das Lernen ist nach dem Unterschreiten einer vorgegebenen Fehlerschranke oder nach einer bestimmten Anzahl von Lernschritten beendet. Danach wird die Qualität der gefundenen Lösung getestet. Ist die Qualität nicht ausreichend, so wird der Lernvorgang fortgesetzt oder ein neuer gestartet.

1.3 Qualität einer neuronalen Lösung

Die Begriffe der Korrektheit bzw. des Fehlers können nicht so, wie sie allgemein bekannt sind, auf eine neuronale Lösung übertragen werden. Somit erlangt der Begriff der Qualität eine ganz andere Bedeutung. Das ist eng damit verbunden, daß sogar der Mensch nie so genau weiß, wann ein Problem wirklich gut gelöst wurde. In der Regel

1. Es gibt allerdings auch Lernverfahren, welche die Topologie selbst optimieren oder konstruieren.

wird nur eine zufriedenstellende Lösung erwartet. Der Ausdruck "zufriedenstellend" ist aber subjektiv und hängt von vielen Faktoren ab, wie z. B. Geld- bzw. Zeitaufwand, persönliches Empfinden usw.

Beispiel 1.2. Angenommen, ein Mensch soll eine Aufgabe lösen, welche Grauwerte in hell und dunkel aufteilt. Die Testperson bekommt 100 Quadrate der Größe *2 x 2* cm mit den zu klassifizierenden Grauwerten. Sie soll diese in helle und dunkle aufteilen. Ohne zusätzliche Hilfsmittel (wie z. B. einer Vergleichfarbskala), trifft jeder Proband diese Entscheidung anhand von Erfahrungswerten, welche er im Laufe seines Lebens gesammelt hat, den gerade herrschenden Umweltbedingungen (z. B. Beleuchtung) und Sinnestäuschungen (z. B. empfindet man denselben Grauwert auf einer weißen Unterlage anders als auf einer schwarzen). Die Erfahrungen bilden sein Ausgangsmodell der reellen Welt, die Umweltbedingungen können als additives Rauschen bezeichnet werden und die Sinnestäuschungen als Schwächen der Sensoren. Als Ergebnis entsteht im Gehirn der Testperson ein Modell der Aufgabenlösung. Diese Modell ist aber sehr subjektiv. Eine andere Testperson löst diese Aufgabe etwas anders, jedoch annähernd gleich. Es ist objektiv nicht entscheidbar, wer diese Aufgabe besser gelöst hat. Wesentlich ist nur, daß beide Personen es annähernd gleich gemacht haben. Beide Lösungen sind zwar nicht identisch, können aber mit dem Prädikat "gut" bewertet werden, da sie den Erwartungen von einer guten Lösung entsprechen. Die kleinen Unterschiede im Grenzbereich sind für die Qualität belanglos.
■

Bei der Bewertung eines neuronalen Netzes geht es auch nur darum, seine Leistung mit der Erwartung zu vergleichen. Man vergleicht also immer zwei Modelle der Lösung, und zwar das Modell im Gehirn des Anwenders mit dem des Netzes. Bei einer Klassifikation erwartet man beispielsweise, daß ein Netz annähernd dieselbe Klassentrennung vornimmt, wie es der Anwender tun würde. Die Erfüllung einer Erwartung steht also im Mittelpunkt der Qualitätsbewertung. Bevor ein neuronales Netz in einer Anwendung eingesetzt werden kann, wird es ausgiebig getestet. Während das Lernen mit einem Teil der verfügbaren Daten geschieht, wird das Testen mit den übrigen Daten durchgeführt. Das Ziel der Tests ist es, eine Bewertung zu formulieren, welche eine Auskunft über die Brauchbarkeit des vom Netz gefundenen Modells für die Lösung der Problemstellung gibt. Was dabei beurteilt wird, ist nicht die Korrektheit des Netzes (oder besser gesagt des Modells), sondern die Übereinstimmung des Netzverhaltens mit dem erwarteten Verhalten. Insbesondere will man wissen, wie das Netz auf Eingaben reagiert, die dem Netz in der Lernphase nicht präsentiert wurden. Wenn das Netz so reagiert, wie es erwartet wird, dann wird es positiv beurteilt, und man schreibt ihm eine gute Generalisierungsfähigkeit zu. Reagiert es nicht so, wie erwartet, dann geht man davon aus, daß das Netz ein Modell gebildet hat, welches sich mit den Vorstellungen des Anwenders nicht deckt. Von einer guten Lösung wird eine gute Generalisierung der Trainingsdaten erwartet. Das Prädikat "gut" ist dabei nicht objektiv, sondern den Vorstellungen des Anwenders angepaßt. Zwei konträre Größen beschreiben dieses Verhalten, nämlich das Überlernen und die Generalisierung.

1.3.1 Überlernen und Generalisierung

Ein neuronales Netz lernt aus Beispielen, ein Problem zu lösen. Diese Beispiele sind ein Sammelsurium von Meßdaten unterschiedlicher Qualität. Sie können mit Meßfehlern behaftet sein (Rauschen), völlig falsche Information enthalten (Ausreißer), den Definitionsbereich nicht gleichmäßig abdecken u. Ä. Die Aufgabe eines Lernalgorithmus ist es, das neuronale Netz so zu parametrisieren, daß die Ausreißer und das Rauschen die Lösung nicht beeinflussen. Dies ist eine komplexe Aufgabe, weil ein neuronales Netz viele Parameter besitzt (Gewichte und Bias-Werte). Das Lernen beginnt immer mit einer zufälligen Belegung der Parameter. Der Lernalgorithmus versucht die Parameter so

zu optimieren, daß die Differenz zwischen der Ist- und Sollausgabe des Netzes, gemittelt auf alle Lernmuster, möglichst klein ist. Im Extremfall kann das neuronale Netz alle Trainingsbeispiele gut, die Testdaten aber völlig falsch annähern (klassifizieren). In solchem Fall spricht man vom Überlernen ("das Netz lernt auswendig"). Eine gute Generalisierung ist das Gegenteil vom Überlernen. Bei einer guten Generalisierung reagiert das Netz auf neue Eingaben so, wie das von ihm erwartet wird. Schon wenige Lernbeispiele genügen, um das Problem ausreichend zu modellieren. Das Erreichen einer guten Generalisierung ist durch viele Faktoren bedingt, wie z. B. gut verteilte und aussagekräftige Daten, die Anfangsbelegung der Netzparameter, dem Lernalgorithmus und nicht zuletzt der Lerndauer. Ein zu langes Lernen führt meistens zum Überlernen. Da man auf diese Faktoren keinen Einfluß hat, ist das Finden einer guten Lösung nur durch Erfahrung und Ausprobieren möglich.

Als gängiges Mittel zur Qualitätsbeurteilung einer neuronalen Lösung und der Qualität von Lerndaten werden statistische Methoden eingesetzt. Einige davon werden im nächsten Kapitel näher erläutert.

Neben der Generalisierung und dem Überlernen gibt es weitere Eigenschaften, welche die Qualität eines neuronalen Netzes spezifizieren. In der Literatur findet man oft folgende Attribute (diese hier stammen aus [Schöneburg90]):

- **Plastizität.** Sie verleiht den Neuronen eines Netzes die Fähigkeit, verschiedene Aufgaben zu übernehmen. Plastizität kann dann von Vorteil sein, wenn Kontextänderungen (z. B. wegen Alterung der Eingangssensoren) durch Änderung der Gewichte aufgefangen werden müssen.
- **Dynamische Stabilität.** Sie ist eine Fähigkeit, welche dem Netz erlaubt, seine Funktionsfähigkeit zu erhalten und immer in einem stabilen Zustand zu bleiben, egal wie abartig die Eingabe ist.

Für die meisten Anwendungen neuronaler Netze, die auf Netzsimulatoren ablaufen, ist die dynamische Stabilität von Bedeutung. Plastizität verlangt, daß während des Betriebs das Netz weiterlernt. Das ist aber meistens nur schwer realisierbar.

1.3.2 Statistische Bewertungsmethoden

Statistische Methoden liefern verschiedene Größen und helfen eine Beurteilung einer neuronalen Lösung oder der Trainingsdaten zu formulieren. Häufig wird der mittlere quadratische Fehler (Mean Square Error, *MSE*) als grobes Maß für die Güte der gefunden neuronalen Lösung genommen (z. B. benutzt es der Backpropagationalgorithmus während des Lernens). Folgende Tabelle gibt eine Auswahl häufig benutzter Größen an.

TABELLE 2. Beispiele statistischer Bewertungsgrößen

Größe	Formel	Bedeutung
MSE	$MSE = \frac{SSE}{n-p}$	Mittlerer quadratischer Fehler. Häufig verwendetes Maß für die Güte einer Lösung. Wird z. B. vom Backpropagation Lernalgorithmus in der Lernphase benutzt.
RMSE	$MSE = \sqrt{\frac{SSE}{n-p}}$	Wurzel aus *MSE*. Hat dieselbe Bedeutung wie *MSE*.
Pearsons R^2	$R^2 = 1 - \frac{SSE}{TSS}$	Proportion der Varianz. Ein Kriterium für die Angemessenheit der Beispiele für die Modellierung der Population.
R^2_{adj}	$R^2_{adj} = 1 - \frac{n-1}{n-p}(1-R^2)$	Proportion der Varianz normiert auf die Freiheitsgrade.

Dabei ist n die Anzahl der Testmuster und p die Anzahl der optimierten Parameter (Gewichte und Bias-Werte). SSE (Sum of Squared Errors) ist die Summe der Fehlerquadrate und TSS (Total Sum of Squares) die Gesamtsumme der Fehlerquadrate, korrigiert um den Mittelwert der abhängigen Variablen [SNNS99].

Einen wesentlichen Nachteil haben die statistischen Methoden. Die aufgeführten Grössen sind nur dann verwendbar, wenn genügend aussagekräftige Daten zur Verfügung stehen. Es gibt zwar Größen, welche die Qualität der Daten beurteilen helfen (z. B. R^2), es bleiben aber immer Mittelwerte, welche in sicherheitsrelevanten Anwendungen kaum brauchbare Aussagen liefern.

Viele Probleme ergeben sich also, wenn eine sicherheitsrelevante Anwendung beurteilt werden soll. In sicherheitsrelevanten Anwendungen steht das Verhalten des Systems in kritischen Situationen im Vordergrund. Kritische Situationen sind meistens durch kritische Bereiche im Definitionsraum dargestellt, welche in der Regel nur in einem kleinen Teil des Definitionsbereiches angesiedelt sind. Für diese Bereiche existieren meistens nur wenige Meßdaten, so daß diese in einer statistischen Analyse untergehen.

Zusammenfassend kann man sagen, daß eine statistische Analyse nur dann ein gutes Werkzeug der Qualitätssicherung ist, wenn es bei der Beurteilung nicht auf kleine, schlecht mit Daten belegte Bereiche des Definitionsbereiches ankommt. Somit eignet sich dieses Verfahren nur bedingt für die Kontrolle sicherheitsrelevanter Anwendungen. Ein Verfahren, welches ein neuronales Netz auch mit wenigen Daten trainieren kann ist das sog. *Crossvalidierungsverfahren.*

1.3.3 Crossvalidierung

Crossvalidierung wird dann eingesetzt, wenn die verfügbare Datenmenge so klein ist, daß es nicht möglich ist, eine Lern- und eine Testbeispielmenge zu erzeugen. Crossvalidierung verbessert im allgemeinen das Generalisierungsverhalten eines neuronalen Netzes.

Bei einer Crossvalidierung wird die Menge der verfügbaren Daten in k disjunkte Teilmengen zerlegt. Das neuronale Netz wird mit $n<k$ Teilmengen trainiert und mit den restlichen k-n Mengen getestet. Dieser Vorgang wird mit zufällig gewählten Kombinationen wiederholt, bis der Fehler unter eine gegebene Schranke fällt.

Crossvalidierung liefert auch bei kleineren Datenmengen gute Ergebnisse, hat aber dieselben Nachteile wie das Lernen mit einer einzigen Lernmenge, wenn die Daten unbalanciert verteilt sind. Somit ist ihr Einsatz in sicherheitsrelevanten Anwendungen nur bedingt sinnvoll.

1.4 Berechenbarkeit eines neuronalen Netzes

Die Berechenbarkeit eines neuronalen Netzes ist etwas anders definiert, als es aus der theoretischen Informatik bekannt ist. Ein neuronales Netz berechnet nicht, sondern modelliert ein Problem. Ein gegebenes Problem kann immer als eine Funktion f der Merkmale $x_1,\ldots, x_n$ dargestellt werden. Ein neuronales Netz (MLP) modelliert dieses Problem, indem es die Funktion f als eine Kombination der Aktivierungsfunktionen seiner Neuronen darstellt. Kolmogorov [Kolmogorov57] hat gezeigt, daß es für jedes "normale" Problem eine bestimmte Anzahl n verdeckter Neuronen gibt, so daß ein Netz mit einer verdeckten Schicht das Problem (bis auf einen Fehler von $\varepsilon>0$) modellieren kann. Als ein "normales" Problem bezeichnet er ein Problem, das sich mit Hilfe einer gleichmäßig stetigen Funktion ausdrücken läßt. Formal sieht es folgendermaßen aus:

$$\forall \text{ gleichmäßig stetige f } \exists \hat{f} \text{ so daß } |f - \hat{f}| < \varepsilon$$

wobei:

$$\hat{f} = act_{out}\left(\sum_{i=1}^{n} act_i\left(\sum_{j=1}^{m} x_j\right)\right)$$

und:

$\hat{f}$	die vom Netz gefundene Lösung,
act_i	die Aktivierungsfunktion des i-ten verdeckten Neurons und
act_{out}	die Aktivierungsfunktion des Ausgabeneurons.

Kolmogorov hat die Existenz einer neuronalen Lösung gezeigt, gab aber nicht an, wie man für ein konkretes Problem ein neuronales Netz konstruktiv erzeugen kann. Die Fähigkeit eines MLPs, mit einer verdeckten Schicht alle "normalen" Probleme lösen zu können, ist der Grund dafür, daß in dieser Arbeit vor allem solche Netze betrachtet werden.

1.5 Zusammenfassung

In diesem Kapitel wurde eine kurze Einführung in die Thematik neuronaler Netze gegeben. Der Schwerpunkt lag auf der Erklärung derjenigen Begriffe, welche in dieser Arbeit benutzt werden. Außerdem wurde ein kurzer Überblick über Methoden der Qualitätsbewertung einer neuronalen Lösung gegeben. Diese Methoden sind ausreichend, wenn sie auf Netze angewendet werden, die nicht in sicherheitsrelevanten Anwendungen arbeiten. Beim Einsatz eines neuronalen Netzes in einer sicherheitsrelevanten Anwendung, wie z. B. in der Ultraschall-Rißerkennung in Öl-Leitungen, werden an die Netzqualität strengere Anforderungen gestellt, insbesondere will man wissen, wie sich das Netz in bestimmten, sicherheitsrelevanten Situationen verhält. Die einzige akzeptable Methode, dies zu realisieren, ist eine Interpretation des Netzes. Im folgenden Kapitel wird eine Relevanzdiskussion geführt und Abwägungen über den Sinn und Zweck einer Interpretation gebracht.

Für das Verständnis der Dissertation ist es wichtig, sich klar zu machen, für welche Netztypen die GINN und IGM konzipiert wurden. Sie wurden für Multilayer Perceptrons mit einer verdeckten Schicht und Neuronen mit einer logistischen Aktivierungsfunktion entwickelt. In der weiteren Beschreibung werden immer nur klassifizierende Netze mit einem Ausgabeneuron und zwei Klassen betrachtet. Eine Erweiterung auf mehrere Ausgabeneuronen und mehrere Klassen wird angegeben.

Kapitel 2

Relevanz einer Interpretation neuronaler Netze

In der Einführung wurden die Grundlagen von neuronalen Netzen erläutert. Ein besonderes Gewicht wurde dabei auf die verschiedenen Methoden der Qualitätsbewertung gelegt. Insbesondere wurde versucht, klar zu machen, daß konventionelle und statistische Bewertungsmethoden für neuronale Netze nicht geeignet sind, vor allem nicht für die Bewertung solcher Netze, welche in sicherheitsrelevanten Anwendungen eingesetzt werden. In diesem Kapitel wird der Sinn und der Zweck einer Interpretation diskutiert. Dabei geht es um eine Motivation, die den hohen Aufwand rechtfertigt, welcher mit der Visualisierung, der Manipulation und der Optimierung eines neuronalen Netzes verbunden ist.

2.1 Motivation

Ein Anwender, welcher ein neuronales Netz einsetzen möchte, will die Sicherheit und die Gewißheit haben, daß das Netz auch das tut, was von ihm verlangt wird. Reine statistische Methoden reichen da nicht immer aus. Aufgrund seiner Komplexität ist ein neuronales Netz eine sog. *Black Box*, d. h. es kann nur sein Ein- und Ausgabeverhalten untersucht werden. Im Unterschied zu einem Programm, das nach gewissen Regeln entwickelt wurde (sicherheitsrelevante Eingaben werden genauer behandelt), lernt ein neuronales Netz selbständig. Weil das Netz zwischen wichtigen, weniger wichtigen und unwichtigen Eingaben nicht unterscheiden kann, werden diese gleich behandelt. Das bedeutet, daß es mit ziemlicher Sicherheit durchaus (sicherheitsrelevante) Eingaben gibt, auf die das Netz falsche Antworten liefert. In Fällen, in denen ein unerwartetes Verhalten hohe Kosten verursachen kann, sind statistische und die gängigen Black-Box-Test-Methoden nicht ausreichend. Viel mehr erwartet man Aussagen, die ein Fehlverhalten ausschließen oder zumindest aufzeigen. Solche Aussagen sind nur dann möglich, wenn die innere Struktur eines Netzes bekannt ist. Selbstverständlich könnte man dieses umgehen, indem die fraglichen Situationen (Eingaben) nachgestellt oder simuliert würden. Leider ist das bei sicherheitsrelevanten Anwendungen nicht immer möglich, z. B. wegen der zu hohen Kosten für die Erzeugung der Testdaten oder zu langen Laufzeiten der Simulationssoftware. Man ist gezwungen, neue Wege zu gehen, um eine Sicherheitsprüfung zu ermöglichen. Genau das war der Fall bei der Entwicklung der Mustererkennung für die Rißsuche in Ölleitungen. Es war unmöglich, eine bestehende

Ölleitung so lange zu betreiben, bis ein genügend großer Riß entstanden ist, weil dieser zum Bruch dieser Leitung führen könnte und man eine Umweltkatastrophe in Kauf nehmen müßte. Auf der anderen Seite waren die synthetisch erzeugten Risse nur bedingt verwendbar, da sie zu "künstlich" sind. Man ist also auf die vorhandenen Daten und das Problemwissen angewiesen. Die in dieser Arbeit entwickelte Methode, die IGNN, bietet einen neuen Ansatz zur Qualitätssicherung.

Die Qualität eines neuronalen Netzes hängt von allen Phasen der Netzentwicklung ab. Wenn man den Prozeß der Netzgenerierung betrachtet, dann lassen sich die sensiblen Phasen wie folgt definieren: (1) Das Dilemma bei der Auswahl der Topologie stellt den Neuroinformatiker immer vor die Wahl, entweder ein zu großes Netz zu wählen, welches dann möglicherweise überlernt, oder ein zu kleines zu wählen, welches zwar besser generalisiert, aber die Feinheiten der Lösung, aufgrund der zu niedrigen Anzahl von Neuronen, nicht modellieren kann. (2) Die Initialisierung der Netzparameter ist ebenfalls kritisch. Wegen der hohen Dimension des Lösungsraumes bestimmt der Startpunkt im wesentlichen das (lokale) Minimum, in welchem der Lernalgorithmus das Lernen beendet. (3) Die Schwierigkeit des Lernens besteht darin, daß man nicht präzise sagen kann, was ein Netz lernt, z. B. welche Merkmale es nutzt, auf welche Weise es die Klassen trennt, wann die beste Lösung bezüglich der Klassifikation gefunden wird[1]. Es gibt bisher für neuronale Netze keine Möglichkeit anzuzeigen, auf welche Art ein Netz eine Aufgabe löst.

Zusammenfassend kann man sagen, daß ein neuronales Netz eine Unbekannte für den Anwender ist, deren Verhalten nur sehr ungenügend verstanden werden kann. Aus diesem Grunde werden neuronale Netze nur mit Vorsicht in sicherheitsrelevanten Anwendungen eingesetzt.

Durch eine Interpretation können diese drei oben genannten Punkte gemildert werden. Was die Topologie angeht, so können überflüssige Neuronen entdeckt und entfernt werden. Ein Netz, welches zu wenig Neuronen hat, kann um neue ergänzt werden. Somit wird seine Topologie verbessert. Was die Initialisierung der Parameter angeht, so können diese vor dem Lernen mit sinnvollen Werten belegt werden (Vorstrukturierung). Das Lernen kann sichtbar gemacht werden, indem die Interpretation in irgendeiner Form als Animation angezeigt wird. Der Anwender kann das Lernen anhalten, wenn er sieht, daß eine befriedende Lösung gefunden wurde. Durch eine Analyse der Visualisierung können Qualitätsaussagen gewonnen werden, die einen Einsatz in sicherheitsgerechten Anwendungen ermöglichen. Ferner kann durch eine Manipulation der Interpretation das Problemwissen in die Netzstruktur eingebracht und somit die Qualität der Lösung verbessert werden.

Ein großes Problem bei der Entwicklung einer Interpretation ist eine adäquate Wissensrepräsentation, insbesondere die Art, auf welche ein Netz und die Lern- bzw. Trainingsmuster repräsentiert werden. Es gibt dafür mehrere Möglichkeiten.

Die parametrische Information aus einem neuronalen Netz liegt in einer sog. *subsymbolischen Form* vor. Sie ist für den Menschen schlecht bzw. gar nicht verständlich. Deswegen muß sie in eine verständliche Form umgewandelt werden. Was ist aber eine verständliche Form? Mit Sicherheit hat jeder dazu eine etwas andere Meinung, denn im wesentlichen hängt die Antwort auf diese Frage von bisherigen Erfahrungen des An-

1. Ein kleiner Approximationsfehler, wie z. B. der MSE, garantiert nicht, daß auch die vom Netz gefundene Klassifikation gut ist. Es kann durchaus vorkommen, daß bei einem größerem MSE die Klassifikationsqualität besser ist als die bei einem kleineren.

wenders, dem erlangten Wissen, persönlichen Vorlieben etc. ab. In dieser Arbeit wurden deshalb vorhandene Repräsentationsformen für neuronale Netze untersucht und dann diejenige ausgesucht, welche am allgemeinsten und am einfachsten zu verstehen ist. Dabei stellt sich die Frage, was eine adäquate Wissensrepräsentation ist?

2.2 Adäquate Wissensrepräsentation

Das Wissen zu repräsentieren wird um so schwieriger, je weniger abstrakt es ist, z. B. lassen sich Reihen von Zahlen, wie die Parameter eines neuronalen Netzes, schwer interpretieren. Stellt man diese Zahlen graphisch dar, so können sie einfacher verstanden werden. Dabei geht ein Teil der Information verloren, bzw. wird er bewußt weggelassen. Bei einer Interpretation stehen weniger die genauen Zahlenwerte im Mittelpunkt, vielmehr sind es die Proportionen, denen Bedeutung beigemessen wird, z. B. entspricht das Verhältnis der Eingangsgewichte eines Neurons der Winkellage der Trennlinie. Es kommt dabei nicht so sehr auf die genaue Winkelposition in Grad an, sondern mehr auf den groben Verlauf, der für die Interpretation von Bedeutung ist. Von einer Interpretation erwartet man aber noch mehr. Man will, daß sie die Lösung eines konkreten Problems visualisiert. Insbesondere soll dabei die verwendete Terminologie problembezogen sein, d. h. die repräsentierten Objekte sollen Größen aus der Problemstellung darstellen. Eine derartige Darstellung, ohne zusätzliche Transformation, ist nur selten möglich. Deswegen ist die Wissensrepräsentation entweder problembezogen, oder sie ist so allgemein, daß noch eine zusätzliche Zwischenstufe eingebaut werden muß, welche die Metainterpretation in eine problembezogene umwandelt. Wenn man aber gewisse Einschränkungen zuläßt, daß z. B. nur eine bestimmte Klasse von Problemen mit einer Interpretationsmethode behandelt werden kann, dann erhält man eine für die gegebene Klasse allgemeine Interpretationsmethode.

Es gibt zwei Möglichkeiten, Wissen zu beschreiben. Die erste sind die sog. *formalen* oder *formalisierten Methoden*. Sie drücken das Wissen mit Hilfe von mathematischen Formeln oder durch Regeln aus. Wenn man die Repräsentationsarten betrachtet, dann ergeben sich (grob gesehen) drei formalisierte Methoden: (1) Beschreibung durch Regeln, (2) Beschreibung mit mathematischen Formeln und (3) unscharfe Methoden. Eine Beschreibung durch Regeln ist nur dann möglich, wenn entsprechende Regeln auch formuliert werden können. Sie sind meistens der Form "wenn x, dann y". Dabei ist die Bedingung x "scharf" formuliert, d. h. sie ist ein mathematischer oder logischer Ausdruck. Eine Beschreibung durch mathematische Formeln findet ihren Einsatz vor allem in der Physik, wo die Bedeutung einzelner Terme aus der Theorie gut bekannt ist. Unscharfen Methoden, welche auf Fuzzy-Mengen [Zadeh65] oder sog. *Rough-Sets* [Ban93] basieren, lassen unscharfe Formulierungen in ihrer Repräsentation zu. Fuzzy-Regeln z. B. erlauben, im Gegensatz zu normalen "scharfen" Regeln, Beschreibungen mit Attributen, wie "groß", "klein", "etwas", "mittel", "wenig" usw. Die Repräsentation ähnelt einer mündlichen Beschreibung. Mit Hilfe einer sog. *Defuzzifizierung* werden aus diesen Beschreibungen Steuerungssignale für Regler, Roboter usw. erzeugt, die dann diesen Fuzzy-Regeln folgen.

Die zweite Möglichkeit der Wissensrepräsentation liegt in einer nichtformalen Beschreibung. Diese Beschreibung läßt sich in der Regel nicht in eine der oben erwähnten formalisierten Repräsentation vollständig umwandeln. Nichtformale Beschreibungen basieren in der Regel auf "selbstverständlichen" Begriffen. Sie sind fast umgangssprachlich und von anderen Menschen leicht verständlich. Sie lassen sich aber nicht mathematisch präzise formalisieren.

In dieser Aufstellung nehmen neuronale Netze eine Zwischenstellung ein. Deren Beschreibung läßt sich zwar mit Hilfe einer Formel niederschreiben (das ist die Funktion,

welche sich durch das Hintereinanderreihen der Neuronenfunktionen, entsprechend der Netztopologie, ergibt), ihre Interpretation bezogen auf ein konkretes Problem ist aber fast unmöglich. Ansätze, welche versuchen, einzelnen Parametern eine Bedeutung zuzuschreiben, waren bisher für mittlere und größere Netze erfolglos. Aufgrund der Ähnlichkeit einer sigmoiden Funktion mit einer sog. *Fuzzyzugehörigkeitsfunktion* gibt es auch Versuche, neuronale Netze zum einen mit Fuzzy-Regeln zu interpretieren [Eppler93] und zum anderen Fuzzy-Wissen in sie einzubringen [Eppler93]. Das Problem bei der Interpretation eines neuronalen Netzes verschiebt sich somit nur auf die Interpretation von Fuzzy-Regeln. Leider hat sich bisher gezeigt, daß nur vorstrukturierte Netze, die durch das anschließende Lernen nur wenig verändert wurden, einigermaßen gut interpretiert werden können.

Andere Methoden, welche die subsymbolische Darstellung eines neuronalen Netzes in symbolische Repräsentation umzuwandeln versuchen, scheitern meistens an der hohen Komplexität und Dimensionalität des Netzes bzw. des Eingaberaumes. Bisher konnten nur einfache Netze mit diesen Methoden analysiert werden.

Eine rein symbolische Wissensrepräsentation ist sehr oft schwer zu verstehen, deshalb verwenden viele Interpretationsmethoden eine graphische Visualisierung. Dargestellt werden verschiedene Netzparameter, wie die numerischen Werte der Gewichte oder die Aufteilung des Eingaberaums durch die Klassentrennlinien (eine Beschreibung einiger ist in Kapitel 3 zu finden). Der Vorteil einer graphischen Methode liegt in der einfachen Interpretation (vorausgesetzt, die Darstellung ist übersichtlich), so daß auch ein Anwender, welcher mit der Theorie neuronaler Netze wenig zu tun hat, diese Repräsentation interpretieren und entsprechende Schlüsse ziehen kann.

Das intuitive Verständnis der graphischen Visualisierungsmethoden ist ausschlaggebend dafür, daß in dieser Arbeit die Wahl auf eine graphische, oder besser gesagt geometrische, Interpretation neuronaler Netze (GINN) fiel. Im Unterschied zu einer rein graphischen Methode, erlaubt die geometrische auch eine Manipulation des Netzes durch eine graphische Manipulation der Darstellung. Dadurch kann ein Anwender sein Wissen in das Netz transferieren. Dieser Transfer wird als interaktive geometrische Manipulation (IGM) bezeichnet. Die Anwendung einer derartigen Interpretation geschieht nach einem bestimmten Paradigma.

2.3 Paradigma einer Interpretation

Das Paradigma, welches hinter der Idee einer geometrischen Interpretation und interaktiven geometrischen Manipulation steht, ist: Sehen-Verstehen-Manipulieren (SVM). Der Entwicklung von SVM lag eine Beobachtung zugrunde, welche folgendes besagt: wenn man eine vom Netz gefundene Lösung zusammen mit den Trainingsdaten graphisch darstellt, dann findet man immer eine Möglichkeit die Funktionsweise eines Netzes zu verbessern, indem man die Klassentrennlinien manipuliert. Diese Beobachtung hängt mit der besseren Generalisierungsfähigkeit des Menschen zusammen. Auch dann, wenn kein Vorwissen vorliegt, erkennt ein Mensch schneller und besser als ein neuronales Netz gewisse Regularitäten oder Anhäufungen von Mustern, die dann eventuell mit einer Trennlinie voneinander getrennten werden könnten. Das verbessert sich noch weiter, wenn Problemwissen vorhanden ist, z. B. wenn es bekannt ist, daß die Trennlinie eine Gerade, Parabel oder Hyperbel ist.

Abbildung 2.1 zeigt das Paradigma von SVM. Das neuronale Netz wird mit Hilfe der Lerndaten trainiert. Während das Netz lernt, beobachtet der Anwender die Animation der Lösung. Wenn ein Punkt erreicht wird, in dem das Netz eine zufriedenstellende Lösung liefert, oder wenn es anfängt zu überlernen, dann stoppt der Anwender den Lern-

vorgang. Jetzt beginnt die Analysephase, in welcher die vom Netz gefundene Lösung, betrachtet wird. In die Darstellung werden jetzt auch die Testdaten eingeblendet. Nachdem die Lösung (teilweise) verstanden wurde, erfolgt eine Manipulation des Netzes durch die Manipulation seiner graphischen Darstellung. Der Anwender kann jetzt die vom Netz gemachten Fehler korrigieren und sein Wissen in die Netzstruktur einbringen.

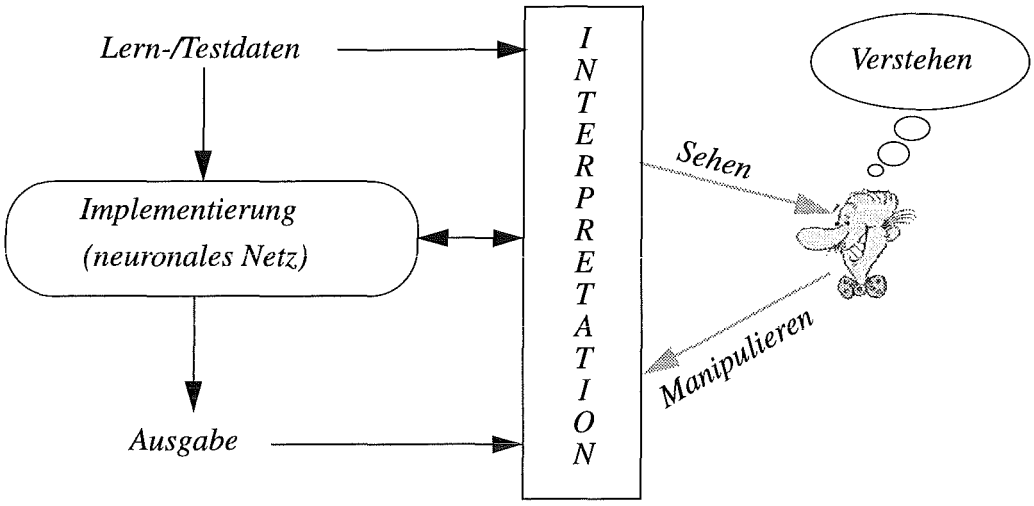


Abbildung 2.1: *Sehen-Verstehen-Manipulieren – das Paradigma für das Übermitteln von Wissen an ein neuronales Netz mittels einer Interpretation.*

Da die Darstellung mit Hilfe von Datenpunkten und Trennlinien zwischen den Klassen realisiert wird, kann nahezu jeder Benutzer, d. h. auch einer, der mit der Theorie der neuronalen Netze nicht vertraut ist, mit SVM arbeiten.

Eine gewisse Schwierigkeit bereitet die Übermittlung des Wissens. Für Aufgaben, in welchen neuronale Netze eingesetzt werden, gibt es normallerweise keine formalen Beschreibungen (sonst würde man konventionelle Lösungsmethoden vorziehen). Was vorhanden ist, sind Meßdaten und ein oft unvollständiges Wissen, welches meistens nichtformaler Natur ist. Dieses unvollständige Wissen kann aber hilfreich sein, um eine vorhandene, nicht ausreichende neuronale Lösung zu verbessern.

Das Übermitteln von Wissen an ein neuronales Netz ist nicht trivial, weil dies durch eine Änderung der Netzparameter geschehen muß. Da die Bedeutung einzelner Parameter für die Lösung aber (meistens) nicht bekannt ist, können die Gewichte oder Bias-Werte, welche die gewünschte Änderung der Lösung hervorrufen, nicht ohne weiteres ermittelt werden. Die Manipulation kann also nur durch eine Veränderung einer höheren Darstellungsform erfolgen, die sowohl vom Anwender als auch vom Netz verstanden wird. Im Falle der geometrischen Interpretation geschieht diese Manipulation durch eine Änderung der graphischen Darstellung. Um dies zu demonstrieren betrachten wir eine Klassifikationsaufgabe:

Beispiel 2.1. Neun Meßpunkte (s. Abbildung 2.2) sollen mit Hilfe eines Netzes klassifiziert werden. Dabei gibt es unendlich viele Möglichkeiten, eine Trennlinie zwischen den Meßpunkten zweier Klassen durchzuziehen, aber nur einige davon sind akzeptabel. Die Abbildung 2.2a und die Abbildung 2.2b zeigen zwei mögliche Aufteilungen des Merkmalsraumes. Die Linien teilen die Daten in zwei Klassen auf. Die Trainingsdaten sind in beiden Beispielen korrekt klassifiziert worden (volle Kreise), die Testdaten (leere Kreise) in Abbildung 2.2b aber nicht. Es gibt einige Fehlklassifikationen.Wenn bekannt ist, daß die Klassentrennlinie einen bestimmten Verlauf haben soll, dann erkennt

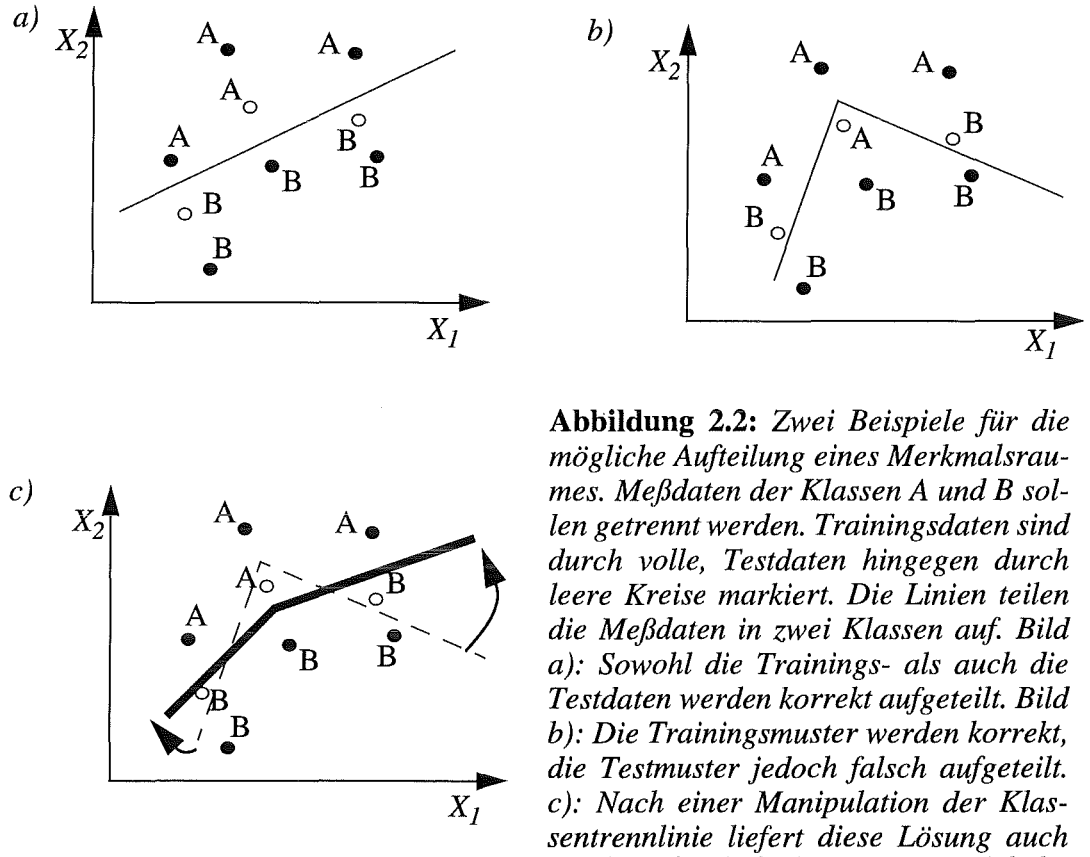


Abbildung 2.2: *Zwei Beispiele für die mögliche Aufteilung eines Merkmalsraumes. Meßdaten der Klassen A und B sollen getrennt werden. Trainingsdaten sind durch volle, Testdaten hingegen durch leere Kreise markiert. Die Linien teilen die Meßdaten in zwei Klassen auf. Bild a): Sowohl die Trainings- als auch die Testdaten werden korrekt aufgeteilt. Bild b): Die Trainingsmuster werden korrekt, die Testmuster jedoch falsch aufgeteilt. c): Nach einer Manipulation der Klassentrennlinie liefert diese Lösung auch eine korrekte Aufteilung. Die gestrichelte Linie ist die Lösung aus b). Durch eine Rotation der Darstellungselemente (entsprechend den Pfeilen) wird die Manipulation vollbracht.*

man sofort, daß die zweite Lösung (Abbildung 2.2b) nicht ausreichend gut ist. Ob aber die erste (Abbildung 2.2a) akzeptabel ist, müßte noch ausführlich geprüft werden. Es ist aber leicht, sich vorzustellen, daß durch eine Manipulation der zweiten Lösung eine korrekte Aufteilung des Merkmalsraumes erreicht werden kann. Eine Möglichkeit zeigt die Abbildung 2.2c. Die gefundene Lösung ist zwar anders als in Abbildung 2.2a, sie genügt aber den Anforderungen (steigende Trennlinie). Durch weiteres Manipulieren kann diese Lösung, in die aus Abbildung 2.2a überführt werden.
■

Das Beispiel demonstriert: wenn ungefähr bekannt ist, wie eine korrekte Aufteilung auszusehen hat (linear, s-Form, v-Form, exponentiell usw.), dann kann sofort eine falsche Lösung erkannt und so korrigiert werden, daß sie den Vorstellungen des Anwenders entspricht. Der umgekehrte Weg, dem Netz eine korrekte Lösung zu zeigen, ist schwieriger und aus einem nichtformalen Wissen nicht immer sofort klar. Dennoch ist es in vielen Fällen möglich, durch zusätzliches Überlegen eine schrittweise Annäherung zu erreichen, indem falsche Alternativen ausgeschlossen werden. Beispiel 2.1 zeigt, daß die verwendete geometrische Darstellungsform sowohl für den Anwender als auch für das neuronale Netz verständlich ist. Der Mensch versteht diese Darstellung und kann dem neuronalen Netz die richtige Lösung zeigen, indem er den Verlauf der Trennlinie verändert. Die Änderung der Trennlinie wird in eine Änderung der Netzparameter übersetzt, die vom Netz verstanden wird.

Ein anderer Aspekt bei der Entwicklung neuronaler Netze ist die Qualitätsbeurteilung. Beispiel 2.1 zeigt, wie eine schlechte Lösung von einer guten unterschieden werden kann. Dabei bezieht sich die Aussage "gut" bzw. "schlecht" nur auf den groben Verlauf der Klassentrennlinie, und zwar immer im Vergleich zu unserer Vorstellung von einer guten Lösung. Bei der Qualitätskontrolle geht es manchmal auch um die Beurteilung der Feinheiten, insbesondere in sicherheitsrelevanten Anwendungen, weil das Verhalten in konkreten Regionen des Definitionsbereiches von Bedeutung ist. Deshalb muß der Verlauf der Klassentrennlinie an manchen Stellen etwas genauer unter die Lupe genommen werden. Die geometrische Interpretation erlaubt das Hereinzoomen in solche Bereiche. Es können auch Beschränkungen definiert werden, die den Verlauf der Klassentrennlinie festlegen. Sie gelten dann beispielsweise in den sicherheitsrelevanten Bereichen. Eine Manipulation der groben Darstellung verändert diese Fixierung nicht.

2.4 Zusammenfassung

In diesem Kapitel wurde das Ziel einer Interpretation erläutert. Es geht immer darum, die Qualität einer neuronalen Lösung zu beurteilen und sie zu verbessern. Ein Entwickler sollte also zu einer Interpretation greifen:

- wenn Fehler auftreten. Das Ziel einer Interpretation ist somit die Fehlersuche. Dabei sollte hier noch einmal betont werden, daß im Hinblick auf neuronale Netze die Definition eines Fehlers immer als eine Abweichung des Verhaltens eines Netzes vom erwarteten Verhalten verstanden wird.
- wenn ein Netz in einer sicherheitsrelevanten Anwendung eingesetzt wird. Dann wird bereits vor der Inbetriebnahme postuliert, daß das neuronale Netz gewisse Sicherheitsanforderungen erfüllt. Eine Interpretation kann die verlangten Garantien liefern und die Kontrollbehörden davon überzeugen, daß das System korrekt arbeitet.
- wenn das vom Netz erzeugte Modell von Interesse ist. Manchmal will man aus rein wissenschaftlich motivierter Neugier wissen, auf welche Weise das Netz eine Aufgabe gelöst hat. Dann kann man mit der Interpretation herumexperimentieren, um neues Wissen zu gewinnen.

Eine Interpretation soll nicht zuletzt auch der Akzeptanz von neuronalen Anwendungen dienen. Viele Industriebetriebe würden Systeme mit neuronalen Netzen einsetzen, wenn sie die Garantie einer zuverlässigen Funktionsweise hätten. Weil es bisher keine ausreichenden Methoden gibt, welche eine leichte Interpretation und Manipulation ermöglichen, wurde die geometrische Interpretation entwickelt. Alle bisherigen Methoden sind entweder nur auf einfache Beispiele anwendbar, oder sie sind dermaßen zeitintensiv, daß sie sich nicht bewährt haben. Die geometrische Interpretation wurde bei der Entwicklung einer praktischen Anwendung (dem Rißprüfmolch) als Mittel der Qualitätskontrolle erfolgreich eingesetzt und getestet.

Kapitel 3

Visualisierung neuronaler Netze

Eine graphische Visualisierung ist meistens die beste Methode, um komplexe Zusammenhänge übersichtlich darzustellen. Sie nutzt dabei die Fähigkeit des menschlichen Gehirns, aus zweidimensionalen Bildern dreidimensionale räumliche Verhältnisse zu schaffen. Eine Visualisierung eines neuronalen Netzes soll ein Werkzeug sein, welches das Verständnis der Prozesse unterstützt, die während des Lernens und während einer Berechnung in einem neuronalen Netz vor sich gehen. Sie soll eine qualitative Bewertung sowie weitere Aussagen liefern, wie z. B. über die Generalisierungsfähigkeit eines Netzes.

Es gibt viele verschiedene Methoden der Visualisierung von neuronalen Netzen. Die bekanntesten werden im folgenden kurz beschrieben. Zuerst aber wird eine Liste mit Anforderungen angegeben, die eine gute Visualisierung erfüllen sollte. Die vorgestellten Methoden werden nach diesen Kriterien bewertet.

3.1 Anforderungen an eine Visualisierung

Damit man eine Visualisierung bequem und effizient nutzen kann, sollte sie mehrere Anforderungen erfüllen. Folgende Liste faßt diese Anforderungen zusammen:

1. Darstellung des gesamten Netzes. Der Betrachter muß in der Lage sein, anhand der Visualisierung die Arbeitsweise des Netzes zu verstehen. Insbesondere soll er die vom Netz realisierte Funktion, im Hinblick auf die Anwendung, interpretieren können.
2. Visualisierung der Beiträge beliebiger Neuronen. Aus der Visualisierung soll möglich sein, den Beitrag jedes einzelnen Neurons im Netz zu entnehmen. Insbesondere sollen Neuronen, welche keinen Beitrag zur Lösung des Problems beitragen, leicht zu finden sein.
3. Visualisierung beliebiger Neuronen. Manchmal bringt die Sicht auf die Eingabedaten aus der Perspektive eines einzelnen Neurons aufschlußreiche Information und trägt zu einer besseren Interpretation bei.
4. Darstellung anwendungsrelevanter Merkmale. In der Visualisierung sollen nur Merkmale dargestellt werden, welchen in der Anwendung eine Bedeutung beigemessen wird. Irrelevante Informationen sollen, so weit es geht, unterdrückt werden. Sie können auf Wunsch abgerufen werden.

5. Manipulation des Netzes durch Manipulation der Darstellung. Die Darstellung soll so beschaffen sein, daß durch die Manipulation der Visualisierung eine Manipulation des Netzes möglich ist, d. h. aus der Darstellung müssen sich die Netzparameter berechnen lassen.
6. Miteinbeziehung der Trainings-, Test- und Arbeitsdaten in die Darstellung. Bei der Visualisierung ist nicht nur die Darstellung des Netzes, sondern auch der Daten, welche in das Netz eingegeben werden, von großer Bedeutung. Anhand der Lage von Datenpunkten innerhalb der Netzdarstellung lassen sich viele Aussagen über die Generalisierungsfähigkeit machen. In manchen Fällen können dadurch falsch klassifizierte Daten erkannt und neu bewertet werden.
7. Darstellung hochdimensionaler Raume. Es muß möglich sein, hochdimensionale Räume auf einem 2D-Bildschirm darzustellen. Dazu werden beispielsweise entsprechende Projektionen verwendet.

Im folgenden Abschnitt werden bereits existierende Visualisierungsmethoden besprochen und nach den obigen sieben Kriterien bewertet.

3.2 Existierende Methoden

Im Mittelpunkt dieser Arbeit steht die Interpretation vorwärtsgerichteter neuronaler Netze, die am Beispiel des Multilayer Perceptrons gezeigt wird. Deswegen werden in diesem Kapitel nur Visualisierungsmethoden für diesen Netztyp besprochen. Die Visualisierung von MLPs wurde in der Literatur nur wenig behandelt. Für andere Netztypen, wie z. B. Kohonen-, ART- oder Hopfield-Netze, gibt es eine große Palette von Methoden.

MLPs haben zwar eine einfache geschichtete Struktur. Eine sinnvolle Darstellung, und somit eine Interpretation von einzelnen oder Gruppen von Neuronen, ist aber nicht trivial. Aus diesem Grunde wurden bisher lediglich die Neuronen der ersten verdeckten Schicht einigermaßen gut visualisiert (Hyperplane-Diagramme, Response-Plot-Diagramme).

Eine weitere Hürde bei der Visualisierung ist die hohe Zahl der zu visualisierenden Parameter. Auf einem Bildschirm lassen sich nur zwei Dimensionen darstellen. Auf diese zwei Dimensionen müssen höherdimensionale Räume abgebildet werden, ohne daß die tragende Information verloren geht.

Nachfolgende Abschnitte präsentieren eine Auswahl von bereits existierenden Visualisierungsmethoden. Diese Auswahl zeigt die Trends, welche sich durchgesetzt haben. Die meisten Derivate unterscheiden sich nur unwesentlich, so daß auf sie nur am Rande eingegangen wird.

3.2.1 Hinton-Diagramme

Eine der ersten Methoden der Visualisierung von neuronalen Netzen ist das Hinton-Diagramm [Hinton86]. Es erlaubt eine kompakte Darstellung der Gewichte und des Bias eines Neurons in einem MLP. Abbildung 3.1 zeigt die Hinton-Diagramme für die Neuronen N_3, N_4 und N_5. Es zeigt sowohl das Vorzeichen als auch die Stärke aller zu einem Neuron eingehenden bzw. vom Neuron ausgehenden Gewichte sowie den Wert des Bias. Jedes Gewicht wird durch ein Quadrat dargestellt. Die Fläche des Quadrats entspricht der Stärke der Verbindung und seine Farbe dem Vorzeichen. Weiße Quadrate repräsentieren positive Gewichte, schwarze hingegen negative Gewichte. Das Diagramm ist so angelegt, daß jedem Neuron im Netz eine feste Position zugewiesen ist,

z. B. belegt das Ausgabeneuron den unteren Teil des Diagramms, die verdeckten Neuronen befinden sich in der Mitte und die Eingabeneuronen im oberen Teil. Die Lage eines Quadrats bestimmt das Neuron, mit dem das dargestellte Neuron auf der Eingabe- bzw. Ausgabeseite verbunden ist. Der Bias eines Neurons wird an derjenigen Position dargestellt, an der das Neuron eine Verbindung zu sich selber haben müßte. In dem Diagramm von Ausgabeneuron N_5 (Abbildung 3.1) stellen die zwei mittleren Quadrate die Gewichte dar, welche von den verdeckten Neuronen *3* und *4* kommen. Das untere Quadrat repräsentiert hingegen den Bias des Neurons N_5.

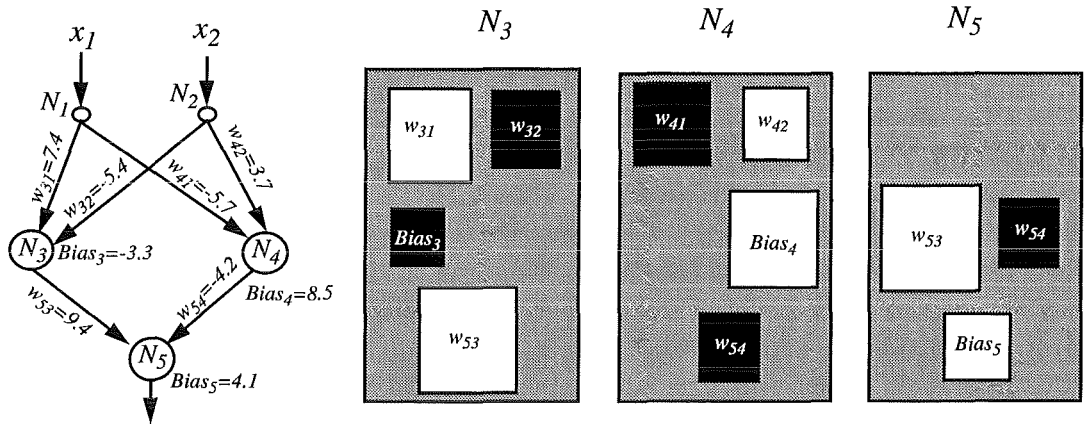


Abbildung 3.1: *Hinton-Diagramm.*

Das Hinton-Diagramm ist in vielen Fällen eine gute visuelle Darstellung der Neuronenparameter. Sowohl die Stärke und das Vorzeichen der eingehenden und ausgehenden Gewichte, der Bias und zum Teil auch der Einfluß eines Neurons auf die Aktivierung anderer Neuronen in der nächsten Schicht ist relativ einfach abzulesen. Für manche Probleme kann diese Darstellung eine verständliche Visualisierung bieten, z. B. benutzte [Pomerleau89] diese Diagramme, um Netze zu verstehen, welche ein Fahrzeug entlang einer Straße steuern. Die Diagramme für die verdeckte Schicht ermöglichten ihm teilweise eine einfache Beschreibung der kritischen Lernmuster. Tesauro und Sejnovski [Tesauro89] verwendeten ebenfalls die Hinton-Diagramme, um die interne Struktur eines Netzes zu beschreiben, welches Backgammon spielen lernte. Sie waren mit Hilfe der Hinton-Diagramme imstande, verschiedene angelernte Spielkombinationen zu erkennen.

Die Hauptnachteile dieser Visualisierungsmethode sind jedoch, daß die Topologie nur schwer und die Partitionierung des Eingaberaumes überhaupt nicht zu sehen sind. Der Bezug zu den Eingabedaten (Lern- und Testmustern) ist nicht gegeben. Der Betrachter braucht viel Übung und sehr gute Kenntnisse über neuronale Netze, um diese Darstellung zu verstehen.

3.2.2 Bond-Diagramme

Das Bond-Diagramm [Wejchert90] stellt ähnlich wie das Hinton-Diagramm die Neuronenparameter dar. Der größte Vorteil des Diagramms gegenüber dem Hinton-Diagramm ist die Einbeziehung der Topologie in die Repräsentation. In Abbildung 3.2 ist ein Bond-Diagramm dargestellt. Jedes Neuron ist als ein Kreis markiert, dessen Größe dem Bias-Wert entspricht. Die Gewichte werden als Bänder (englisch bond) zwischen zwei Neuronen dargestellt. Die Länge des Bandes entspricht der Gewichtsstärke; die Farbe indiziert das Vorzeichen.

Bis auf die Tatsache, daß im Bond-Diagramm die Topologie dargestellt wird, hat diese

Methode dieselben Nachteile wie das Hinton-Diagramm. Zusätzlich kommt noch hinzu, daß es schwierig ist, die Bias-Werte mit den Gewichten zu vergleichen, weil diese graphisch unterschiedlich dargestellt werden. Dieser Vergleich ist aber wichtig, wenn der Einfluß der Neuronaktivierungen beurteilt werden soll. Bei einer großen Anzahl der Neuronen wird dieses Diagramm ohnehin unübersichtlich. Die Aufteilung des Eingaberaums kann aus den Bond-Diagrammen genauso wenig wie aus den Hinton-Diagrammen abgelesen werden.

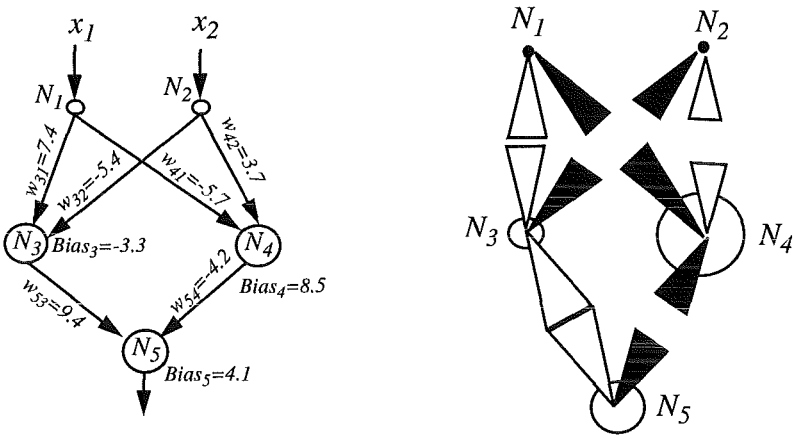


Abbildung 3.2: *Bond-Diagramm.*

3.2.3 Hyperplane-Diagramme

Bei einem Hyperplane-Diagramm [Munoro91] [Pratt91] werden alle Neuronen, unabhängig von ihrer tatsächlichen Aktivierungsfunktion, wie einfache Stufenneuronen behandelt (Aktivierungsfunktion s. Abbildung 1.3c, Seite 9). Derartige Neuronen teilen den Eingaberaum in zwei Bereiche auf, und zwar in einen mit der Aktivierung *1* und in einen mit einer Aktivierung *0*. Sie lassen sich als Trennhyperebenen mit einer Dimension um eins kleiner als die Dimension des Eingaberaumes visualisieren.

Hyperplane-Diagramme können Neuronen der ersten verdeckten Schicht eines vorwärtsgerichteten Netzes darstellen. Ein Defizit der Hyperebenen-Diagramme liegt darin, daß diese Methode ausschließlich zwei- oder dreidimensionale Eingaberäume darstellen kann. Bei höherdimensionalen Eingaberäumen können zwar verschiedene Projektionen ausgewählt werden, eine Interpretation der Daten wird aber zunehmend schwierig. Es gibt verschiedene Erweiterungen dieses Verfahrens, die sich vor allem auf die Verbesserung der Projektion anhand einer Datenanalyse konzentrieren. Als Datenanalyseverfahren wird die Principal Component Analysis (PCA) [Dennis91] [Elman89], Hierarchical Cluster Analysis (HCA) [Elman89] [Sejnowski87] oder Canonical Component Analysis (CCA) [Dennis91] [Wiles90] verwendet.

Der Nachteil dieser Methode liegt in der Approximation der Aktivierungsfunktionen durch Stufenfunktionen sowie darin, daß die Superposition der Neuronen überhaupt nicht berechnet wird. So kann die verteilt gespeicherte Information überhaupt nicht zum Vorschein kommen, und die für die Interpretation unverzichtbaren Klassentrennlinien können nicht angezeigt werden. Es ist lediglich die Aufteilung des Eingaberaumes aus der Sicht eines verdeckten Neurons visualisiert. Diese Darstellung suggeriert ebenfalls, daß MLPs mit einer verdeckten Schicht ausschließlich eine konvexe Aufteilung des Eingaberaumes vornehmen können. Es besteht hier keine Möglichkeit, Netze mit mehr als einer verdeckten Schicht zu visualisieren.

Die Erweiterungen des Verfahrens erlauben zwar eine Darstellung höherer Eingabe-Dimensionen durch eine lineare Analyse der Daten. Damit eignet sich dieses Verfahren je-

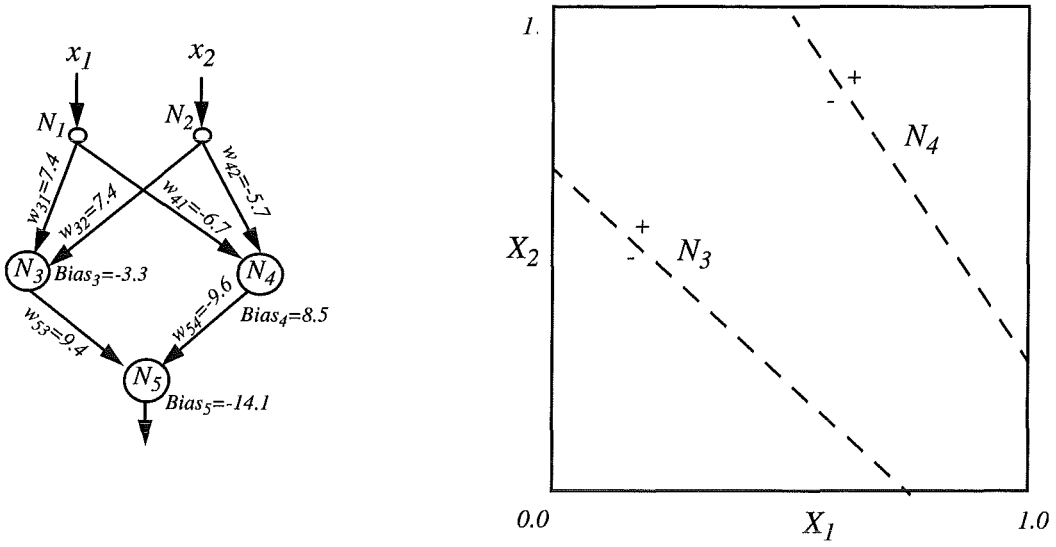


Abbildung 3.3: *Hyperplane-Diagramm des Netzes zur Realisierung der XOR-Funktion aus Abbildung 1.4 auf Seite 10.*

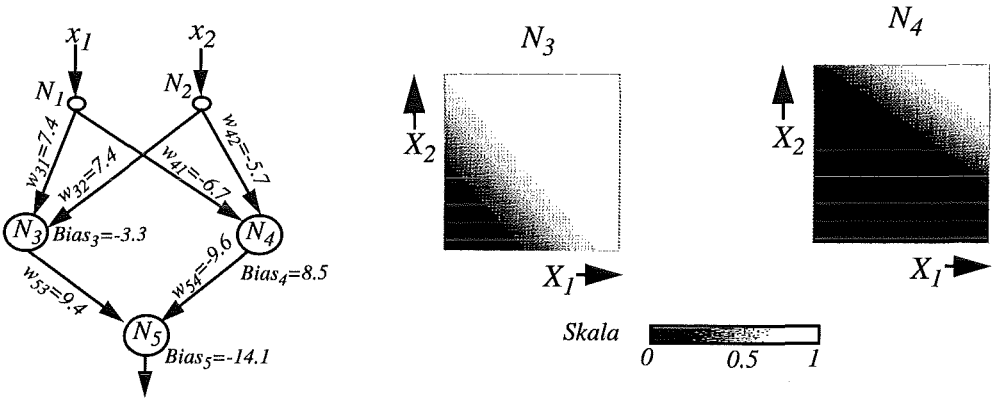


Abbildung 3.4: *Response-Function-Plots.*

doch nur für die Darstellung von Daten mit einer verhältnismäßig hohen stochastischen Ordnung.

3.2.4 Response-Function-Plots

In der Response-Function-Plots-Technik [Lang88] wird wie bei der Hyperplane-Methode die Entscheidungsfläche dargestellt. Die Approximation der Neuronen durch eine Stufenfunktion ist aber nicht mehr nötig, da das Netz wie eine Funktion aufgefaßt wird, deren Verlauf im Plot graphisch dargestellt wird. In einem Response-Plot kann immer nur ein Neuron dargestellt werden. Die Visualisierung ist nicht auf die Darstellung der Abhängigkeit von der Netzeingabe beschränkt, sondern es können beliebige Zusammenhänge untersucht werden. Lang und Witbrock [Lang88] zeigen, daß diese Plots für eine zweidimensionale Eingabe eine übersichtliche Darstellung bieten. Für die Darstellung höherdimensionaler Eingaben müssen ähnliche Maßnahmen getroffen werden wie beim Hyperplane-Diagramm.

Diese Methode hat im Vergleich zu den Hyperplane-Diagrammen den Vorteil, daß die Neuronen nicht mehr durch Stufenfunktionen approximiert werden müssen. Der Nachteil liegt darin, daß die Berechnungen von hoher Zeitkomplexität sein können. Die Re-

chenzeit wird um so länger, je kleiner die Abtastschritte und je größer die Eingabedimension bzw. Definitionsbereiche gewählt werden. Die Schrittweite ist a priori nicht bekannt, daher muß in der Regel eine genügend kleine Schrittweite angenommen werden.

3.2.5 Trajectory-Diagramme

Eine andere Methode der Visualisierung sind die von Wejchert und Tesauro [Wejchert90] entwickelten Trajectory-Diagramme. Während der Lernphase zeichnen sie die Trajektorie eines Neurons im Gewichteraum auf. Die Achsen des Diagramms sind durch die zum Neuron weisenden Gewichte definiert. Die Linienstärke entspricht dem momentanen Lernfehler. In Abbildung 3.5 ist das Trajectory-Diagramm des Neurons N_4 aus Abbildung 3.4 dargestellt. Es ist leicht zu erkennen, daß das Neuron ein lokales Minimum überwunden hat.

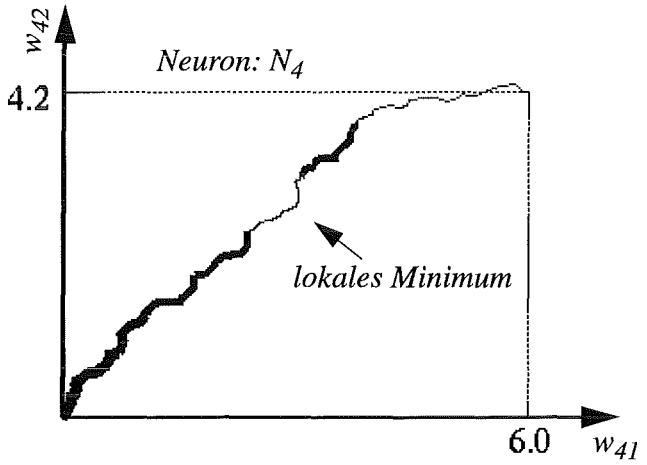


Abbildung 3.5: *Trajectory-Diagramm. Der Fehler am Ausgang des Netzes wird in Abhängigkeit von den Eingabe aufgetragen. Die Stärke der Linie gibt den Fehler an.*

Dieses Verfahren stellt die Fehlerhyperebene in einer etwas anderen Form dar. Es ist besonders dann zu empfehlen, wenn beurteilt werden soll, ob der Lernalgorithmus in einem lokalen Minimum stehengeblieben ist.

Der Nachteil dieser Methode liegt in der schlechten Darstellung hochdimensionaler Gewichtsräume, die Topologie wird nicht wiedergegeben, und die Partition des Eingaberaumes ist nicht zu erkennen.

3.2.6 Zusammenfassung

Die untere Tabelle zeigt eine Zusammenfassung der besprochenen Visualisierungsmethoden unter Berücksichtigung der Anforderungen aus Kapitel 3.1.

TABELLE 3. Bewertung der Visualisierungsmethoden

Visualisierungsmethode	Hinton	Bond	Hyperplane	Response-Function-Plots	Trajectory
Darstellung der vom Netz realisierten Funktion	Nein	Nein	Nein	Ja	Nein
Visualisierung der Beiträge beliebiger Neuronen	Ja	Ja	Nein[a]	Ja	Ja

TABELLE 3. Bewertung der Visualisierungsmethoden

Visualisierungsmethode	Hinton	Bond	Hyperplane	Response-Function-Plots	Trajectory
Visualisierung beliebiger Neuronen	Ja	Ja	Nein[a]	Ja	Ja
Darstellung anwendungsrelevanter Merkmale	Nein	Nein	Nein	Nein	Nein
Manipulation des Netzes durch Manipulation der Darstellung.	Nein	Nein	Nein	Nein	Nein
Miteinbeziehung der Daten in die Darstellung.	Nein	Nein	Nein	Nein	Nein
Darstellung hochdimensionaler Eingaberäume	Nein	Nein	Nein	Nein	Nein

a. Es können nur Neuronen der ersten verdeckten Schicht dargestellt werden

Diese Tabelle zeigt deutlich, daß keines dieser Verfahren eine Manipulation des Netzes zuläßt. Alle Visualisierungsverfahren können lediglich zweidimensionale Eingaberäume darstellen und sind daher unzureichend. Eine Einblendung von Trainings- und Testdaten ist bei keiner dieser Methoden vorgesehen. Die Darstellungen in Hinton- und Bond-Diagrammen sind alles andere als intuitiv. Sie fordern vom Betrachter eine große Vorstellungskraft, um die Bilder zu verstehen. Eine zusätzliche Darstellung von Daten würde diese Diagramme gänzlich unlesbar machen. Keines der Visualisierungsverfahren stellt die vom Netz realisierte Funktion zusammen mit den Beiträgen einzelner Neuronen dar. Die Response-Plot-Diagramme zeigen zwar die Netzfunktion an, liefern aber keine Aussage über die Beteiligung der jeweiligen Neuronen am Gesamtfunktionswert. Die Hyperplane-Diagramme zeigen im Gegensatz nur die Eingabeaufteilung aus der Sicht der verdeckten Neuronen; die vom Netz realisierte Funktion ist nicht dargestellt. Die doppelte Darstellung ist für die Interpretation sehr wichtig, weil sie die Neuronen erkennbar macht, welche für den Funktionsverlauf verantwortlich sind. Somit können einerseits relevante und andererseits überflüssige Neuronen gefunden und ggf. manipuliert oder entfernt werden. Damit eine Visualisierung eine echte Hilfe für die Qualitätsbeurteilung ist, müssen neben dem Netz auch die Daten in die Visualisierung miteinbezogen werden.

3.3 Visualisierung von hochdimensionalen Daten

Eine gute Visualisierung muß für den Betrachter leicht verständlich sein. Deswegen werden die hochdimensionalen Daten vorher so aufbereitet, daß sie sich anschließend leicht und übersichtlich auf einem zweidimensionalen Computerbildschirm darstellen lassen. Die meisten Aufbereitungsalgorithmen arbeiten zweistufig. Zuerst wird mit Hilfe einer Transformation eine kompakte Darstellung im hochdimensionalen Raum erzeugt, die dann in einen zweidimensionalen Raum, den Computerbildschirm, hineinprojiziert wird.

In den folgenden Abschnitten werden einige Methoden erläutert, welche auf statistischer Datenanalyse basieren und auch mit Hilfe von neuronalen Netzen realisiert wer-

den können. Damit ist auch eine schnelle Implementierung möglich. Die Auswahl wurde im Hinblick auf die Anwendung in der GINN und IGM getroffen.

3.3.1 Principal Component Analysis (PCA)

Als erste Methode wird die Lineare Principal Component Analysis (PCA) vorgestellt. Sie wird auch oft als Hauptkomponentenanalyse bezeichnet und wurde in der jetzigen Form bereits im Jahre 1993 von Hotelling entwickelt [Hotelling33]. Ihre Idee ist aber noch älter. Schon im Jahre 1901 verwendete sie Pearson für eine lineare Regression von Ebenen in einer vereinfachten Version [Pearson01].

PCA ist ein statistisches Verfahren, welches unter den Oberbegriff Faktoranalyse fällt [Diamantaras96]. Grob gesprochen findet eine PCA Abhängigkeiten innerhalb der multivariaten statistischen Variablen und ermöglicht somit deren kompaktere Beschreibung.

Wenn es Korrelationen unter statistischen Variablen der Dimension n gibt, dann genügen $m<n$ unabhängige Merkmale (Variablen) zu deren Beschreibung, ohne daß wesentliche Information[1] dadurch verloren geht. Als Faustregel gilt: Je größer die Korrelation unter den Variablen ist, desto weniger unabhängige Variablen werden für die Beschreibung benötigt. Die neuen Merkmale sind eine lineare Kombination aus den ursprünglichen. Dadurch läßt sich oft deren Bedeutung leicht festlegen.

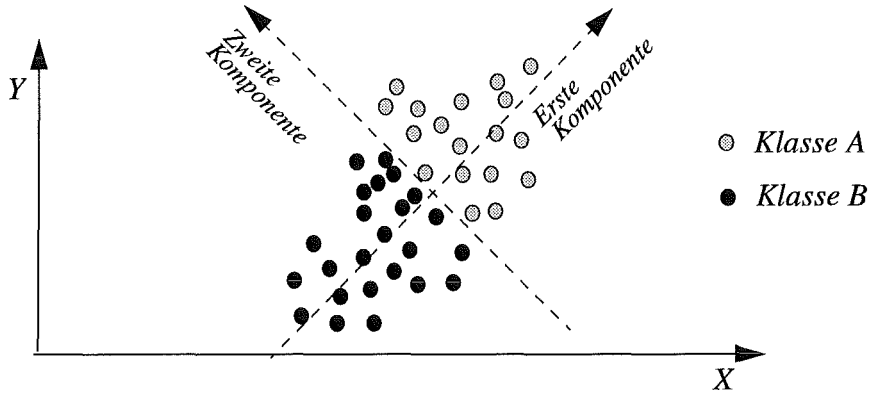


Abbildung 3.6: *Die gestrichelten Linien sind die Koordinaten nach einer PCA-Transformation. Die erste Komponente der PCA weist in die Richtung der größten Varianz, die zweite Komponente ist senkrecht zur ihr.*

Beispiel 3.1. Abbildung 3.6 zeigt ein Beispiel für eine PCA-Transformation. Die *42* Meßpunkte sind so angeordnet, daß aus der Achsensicht des ursprünglichen Koordinatensystems (durchgezogene Koordinatenachsen X und Y) keine Regularitäten zu erkennen sind, d. h. die Daten lassen sich nicht anhand nur einer Koordinate voneinander trennen. Es findet also keine Reduzierung der Dimension statt. Nach der PCA-Transformation sind die Koordinatenachsen (gestrichelte Linien) so gelegt worden, daß die zwei Klassen anhand der ersten Komponente gut separierbar sind. Zur Beschreibung des Problems genügt nur eine Koordinate. Es fand also eine Dimensionenreduzierung statt. ■

1. Im informationstheoretischen Sinne.

Bevor die PCA eingesetzt werden kann, müssen die Variablen mittelwertfrei gemacht werden. Dies kann geschehen, indem der Mittelwert von den Variablen abgezogen wird. Die PCA transformiert dann die Eingabevariablen $x=[x_1, ..., x_n]$ (mit dem Mittelwert $E\{x\}=0$ und der Kovarianzmatrix $R_x=E\{xx^T\}$) auf einen orthogonalen Merkmalsvektor $y=[y_1,...,y_m]$:

$$y = Wx, \tag{3.1}$$

wobei $m \leq n$ ist.

Bei der Berechnung der PCA wird also die Matrix W ermittelt. Ihre Spalten bilden eine orthonormale Basis eines Unterraums L, d. h. der Unterraum L wird durch die Transformationsmatrix W aufgespannt und es gilt: $WW^T=I$ und $L=span(W)$. Unter diesen Voraussetzungen ergibt eine Projektion von x auf den Unterraum L die Rekonstruktion $\hat{x}$ aus y. Formal ausgedrückt:

$$\hat{x} = W^T y = W^T W x, \tag{3.2}$$

wobei W^TW den Projektor bildet. Die PCA-Transformation minimiert den Rekonstruktionsfehler J_e im Sinne des kleinsten Fehlerquadrats. Dieser Fehler läßt sich schreiben als:

$$\begin{aligned} J_e &= E\{|x-\hat{x}|^2\} \\ &= E\{tr[(x-\hat{x})(x-\hat{x})^T]\} \\ &= tr(R_x) - \underbrace{tr(WR_xW^T)}_{J_v}, \end{aligned} \tag{3.3}$$

wobei tr (trace) die Spur, d. h. die Summe der Diagonalelemente der Matrix ist. Der Term J_v ist gleich der Varianz von y und gleichzeitig gleich der Varianz der Rekonstruktion $\hat{x}$, denn:

$$\begin{aligned} J_v = tr(WR_xW^T) &= E\{tr(yy^T)\} = \sum_{i=1}^{m} y_i^2 \\ = tr(W^TWR_xW^TW) &= E\{tr(\hat{x}\hat{x}^T)\} = \sum_{i=1}^{n} \hat{x}_i^2. \end{aligned} \tag{3.4}$$

Somit entspricht die Maximierung der Projektionsvarianz J_v der Minimierung des Rekonstruktionsfehlers J_e, d. h. PCA kann entweder als Varianzmaximierungs- oder als Rekonstruktionsfehlerminimierungsmethode angesehen werden [Diamantaras96].

Satz 3.1 (PCA)
Seien $\lambda_1, \lambda_2,..., \lambda_n$ in absteigender Reihenfolge geordnete Eigenwerte der Kovarianzmatrix R_x und $e_1, e_2,..., e_n$ die entsprechenden normalisierten Eigenvektoren. Die Transformationsmatrix, welche den Rekonstruktionsfehler J_e minimiert bzw. die Varianz J_v maximiert, hat die Form:

$$W_{Opt} = M[\pm e_1 \ \ldots \pm e_m]^T \tag{3.5}$$

wobei M eine beliebige quadratische orthogonale Matrix ist. Der minimale Rekonstruk-

tionsfehler J_e ist dann:

$$min\ J_e = \sum_{i=m+1}^{n} \lambda_i\ , für\ m<n \tag{3.6}$$

und die maximale Varianz J_v ist gleich:

$$\max J_v = \sum_{i=1}^{m} \lambda_i \tag{3.7}$$

Den Beweis zu diesem Satz findet man z. B. in [Diamantaras96].

Eine Verallgemeinerung der PCA-Transformation ist die Karhunen-Loève-Transformation, welche eine Erweiterung auf zeitkontinuierliche statistische Prozesse erlaubt [Karhunen46] [Loève48].

Beispiel 3.2. Abbildung 3.7 zeigt die PCA-Transformierte der Meßdaten aus der Datei cancer, die aus der Sammlung Proben1 [Prechelt94] stammen. Jedes Meßdatum hat acht Merkmale, so daß eine direkte Darstellung auf einem zweidimensionalen Bildschirm nicht möglich ist. Nach einer PCA-Transformation lassen sich aber diese Daten relativ gut darstellen. Die eine Klasse (Kreuze) repräsentiert bösartige, die andere (Kreise) die harmlosen Tumore. Diese beiden Klassen lassen sich anhand der PCA-Darstellung gut trennen. Die bösartigen Tumore bilden eine Häufung im rechten Teil des Bildes. Allerdings vermischen sich auch wenige Kreuze mit den Kreisen im mittleren Bildbereich.

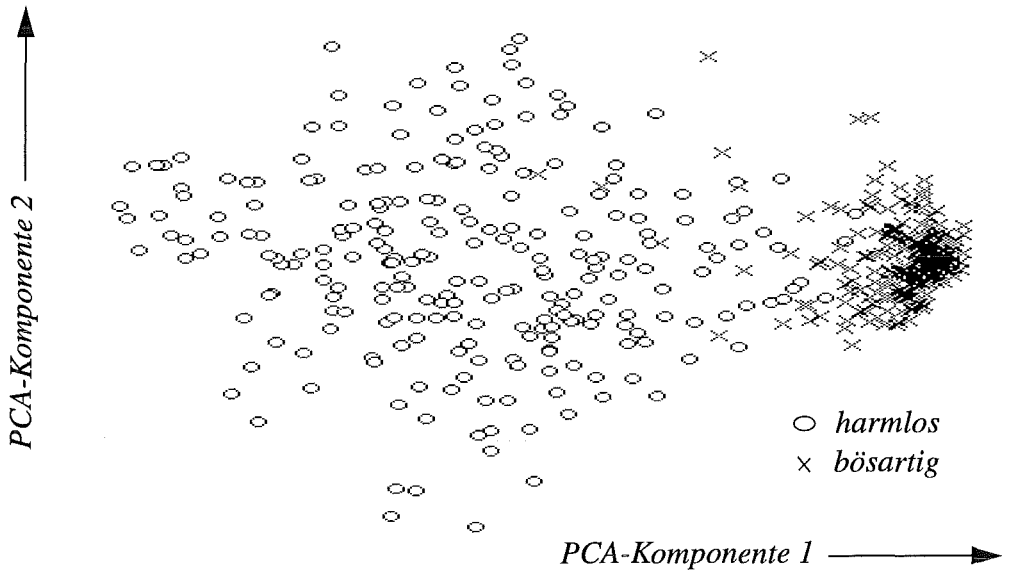


Abbildung 3.7: *Beispiel einer PCA-Transformation. Die Beispieldaten stammen aus einer Benchmarksammlung Proben1 [Prechelt94]. Beide Klassen, d. h. harmlose und bösartige Tumore, lassen sich gut trennen. Die bösartigen Tumore bilden eine Häufung im rechten Bildteil.*

■

Die Berechnung der PCA ist relativ aufwendig, da die Berechnung der Eigenvektoren einer Matrixinversion bedarf. Deshalb gibt es in der Literatur immer wieder Versuche, die Berechnung einer PCA für Darstellungszwecke mit anderen Methoden zu realisieren. Im Unterschied zur Statistik, wo die Achsen der PCA-Komponenten orthogonal sein müssen, genügt es bei einer Darstellung, wenn die Achsen einigermaßen senkrecht sind. Eigentlich wird nur verlangt, daß das Bild eine gute Darstellung der Daten bietet. In der Literatur gibt es mehrere Versuche, PCA mit Hilfe von neuronalen Netzen zu rea-

lisieren. Einige davon werden jetzt vorgestellt.

3.3.2 PCA realisiert mit neuronalen Netzen

In diesem Abschnitt wird eine Verbindung zwischen neuronalen Netzen und der PCA hergestellt. Da beide Verfahren den quadratischen Fehler minimieren, ist naheliegend, daß neuronale Netze die Komponenten einer PCA-Zerlegung berechnen könnten. Im Jahre 1982 stellten Oja und Karhunen [Oja82] [Karhunen82] ein Netz vor, welches die erste Komponente der PCA-Zerlegung berechnet (siehe Abbildung 3.8):

$$out = w^T x. \tag{3.8}$$

Das vorgeschlagenen Lernverfahren modifiziert die Gewichte nach der Regel:

$$w_{k+1} = w_k + \beta_k(out_k x_k - out_k^2 w_k). \tag{3.9}$$

Dies ist die linearisierte Version der normalisierten Hebbschen Lernregel ([Hebb49] Kapitel 4), wobei $out_k^2 w_k$ die Normalisierung ist. Unter zwei Bedingungen konvergieren die Ausgabe des Netzes gegen die erste Komponente und die Gewichte gegen den ersten Eigenvektor [Diamantaras96]:

1. Der statistische Prozeß $\{x_k\}$ ist im weitesten Sinne stationär, d. h. alle Mittelwerte sind von einer Verschiebung aller Beobachtungen unabhängig. Seine Autokorrelationsmatrix R_x besteht aus positiven und in absteigender Reihenfolge sortierten Eigenwerten: $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \ldots \geq \lambda_n > 0$, wobei der erste Eigenwert λ_1 ein einfacher Eigenwert ist.

2. Es gilt: $\lim_{k \to \infty} \beta_k = 0$ und $\sum_{k=0}^{\infty} \beta_k = \infty$.

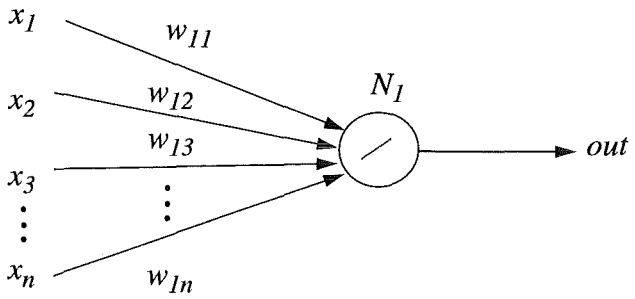


Abbildung 3.8: *Das neuronale Netz von Oja mit einem linearen Neuron (der schräge Strich in der Neuronendarstellung symbolisiert ein lineares Neuron) zur Berechnung der ersten Komponente einer PCA-Zerlegung.*

Die Berechnung aller Komponenten mit Hilfe eines neuronalen Netzes gestaltet sich schwieriger, da jeweils jedes Ausgabeneuron für eine Komponente stehen muß. Die Sortierung der Komponenten verlangt eine Konkurrenz unter den Ausgabeneuronen, was entweder durch vollständige laterale oder durch hierarchische Verbindungen der Ausgabeschicht erreicht werden kann.

Der Generalised Hebbian Algorithm (GHA) von Sanger [Sanger89] verwendet ein ähnliches Netz wie das Verfahren von Oja, berechnet aber nicht nur die erste Komponente, sondern alle Eigenvektoren in der Gewichtematrix. Das Netz von Földiák [Földiák89] spannt mit seinen Gewichten den Eigenvektorraum auf. Für die Berechnung der Komponenten wird aber eine Matrixinversion benötigt. Genau dies macht dieses Verfahren

nicht lokal. Hornik zeigte, daß dieses Verfahren numerisch instabil ist [Hornik92]. Diese unerwünschten Effekte können durch asymmetrische laterale Verbindungen beseitigt werden [Diamantaras92].

Eine bessere Lösung haben Rubner und Tavan [Rubner89] sowie Rubner und Schulten [Rubner90] vorgestellt (Abbildung 3.9). Ihre Lösung ist ein einschichtiges Netz mit lateralen Verbindungen zwischen den Ausgabeneuronen. Diese Verbindungen sind asymmetrisch, und die Neuronen sind hierarchisch geordnet. In dieser Topologie hat das erste Ausgabeneuron eine laterale Verbindung zu allen übrigen Ausgabeneuronen. Das zweite Ausgabeneuron ist, bis auf das erste, mit allen Ausgabeneuronen verbunden, das dritte ist, bis auf das erste und zweite, mit allen Ausgabeneuronen verbunden usw. Der Lernalgorithmus verwendet für die feedforward-Gewichte die normalisierte Hebbsche Regel:

$$\begin{aligned} \Delta w_{ij,\mathrm{k}} &= \beta out_{ik} x_{jk} \\ \tilde{w}_{ij,k+1} &= w_{ij,k} + \Delta w_{ij,k} \\ w_{ij,k+1} &= \frac{\tilde{w}_{ij,k+1}}{\|\tilde{w}_{ij,k+1}\|} \end{aligned} \tag{3.10}$$

und für die lateralen Gewichte die Anti-Hebbsche Regel:

$$\begin{aligned} \Delta c_{ij,k} &= -\beta y_{ik} out_{jk}, \quad \textit{für} \quad i > j \\ \tilde{c}_{ij,k+1} &= c_{ij,k} + \Delta c_{ij,k} \\ c_{ij,k+1} &= \frac{\tilde{c}_{ij,k+1}}{\|\tilde{c}_{ij,k+1}\|}, \end{aligned} \tag{3.11}$$

wobei k den Lernzyklus identifiziert. Dieses Netz konvergiert gegen die normalisierten PCA-Eigenvektoren [Diamantaras96]. Wegen der Normalisierung in der Hebbschen Regel ist das Verfahren nicht lokal. Es kann aber lokal werden, wenn anstelle der einfachen Normalisierung in (3.10) und (3.11) die linearisierte Version verwendet wird:

$$\begin{aligned} \Delta w_{ij,k} &= \beta(out_{ik} x_{jk} - out_{ik}^2 w_{ij,k}) \\ \Delta c_{ij,k} &= -\beta(out_{ik} out_{jk} - out_{ik}^2 w_{ij,k}), \quad \textit{für} \quad i > j. \end{aligned} \tag{3.12}$$

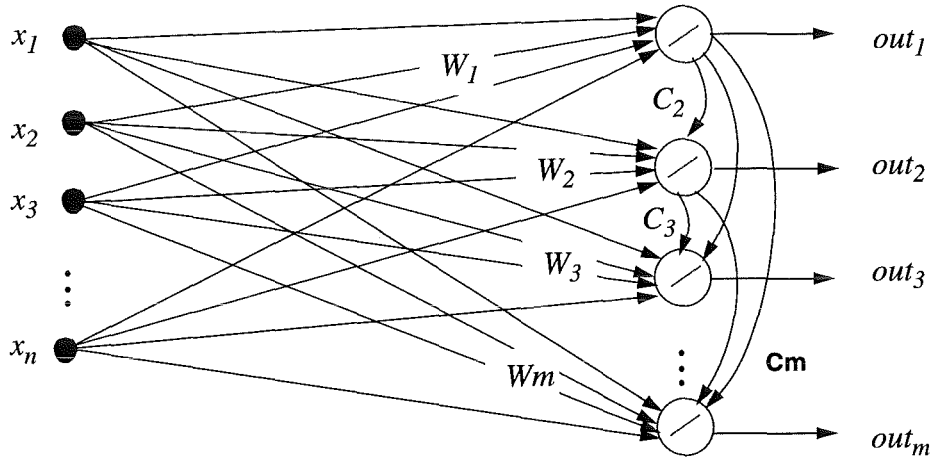


Abbildung 3.9: *Topologie des Rubner-Netzes zur Berechnung der PCA.*

Kung und Diamantaras [Kung90] verwenden in ihrem APEX (Adaptive Principal com-

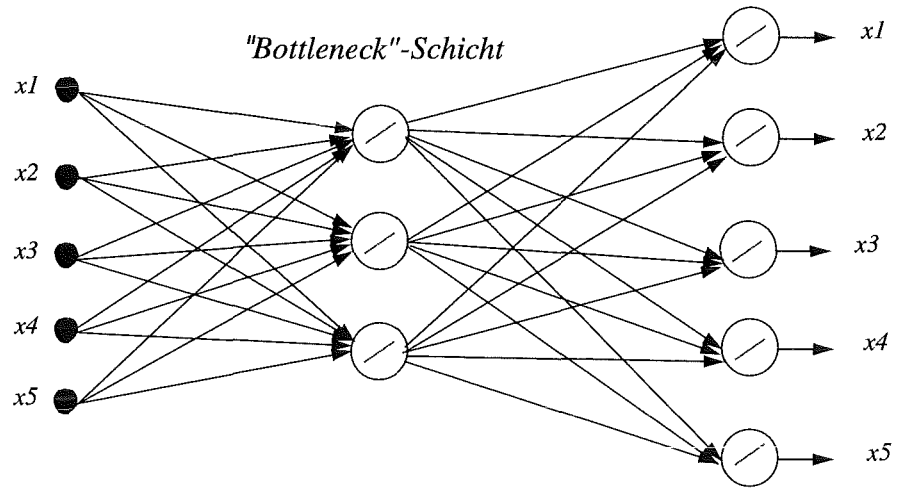


Abbildung 3.10: *Lineares Multilayer-Netz als PCA-Extraktor. Die Aktivitäten der "bottleneck"-Neuronen entsprechen den PCA-Komponenten.*

ponent EXtraction) Netz dieselbe Topologie wie Rubner, setzen aber ein anderes Lernverfahren ein. Sie benutzen Ojas Lernregel sowohl für die feedforward als auch für die lateralen Verbindungen.

Einen Vergleich der PCA-Netze findet man in [Diamantaras96]. Die Netze werden dort untersucht bezüglich der Lokalität des Lernverfahrens, der Genauigkeit der extrahierten Komponenten und der Fähigkeit, eine oder mehrere Komponenten zu berechnen.

TABELLE 4. Bewertung der PCA-Netze

	Lokalität	**Genaue Berechnung**	**Berechnung mehrerer Komponenten möglich**
Oja's Single Unit [Oja82]	ja	ja	nein
Földiák's Model [Földiák89]	nein	nein	ja
GHA [Sanger89]	nein	ja	ja
Rubner's Network [Rubner89]	ja	ja	ja
APEX Network [Kung90]	ja	ja	ja

3.3.3 Lineare Multilayer-Netze und PCA

Lineare Multilayer-Netze mit einer sog. *"bottleneck"-verdeckten Schicht* realisieren eine PCA-Zerlegung, wenn sie autoassoziativ arbeiten, d. h. wenn die Ausgabemuster des Trainingssatzes gleich den Eingabemustern sind. Eine "bottleneck"-verdeckte Schicht ist eine Netzschicht, die weniger Neuronen hat als die Eingabeschicht. Die Aktivitäten der "bottleneck"-Schicht entsprechen den PCA-Komponenten [Diamantaras96].

3.3.4 Nichtlineare PCA mit Multilayer-Perceptrons

Der Zweck der PCA ist die Erkennung von Korrelationen in den Eingabedaten und die Darstellung der Daten mit Hilfe weniger Merkmale, und zwar so, daß in der neuen Darstellung eine einfache Trennung der Klassen möglich ist. In der normalen PCA-Zerlegung hängen die gefundenen Merkmale linear von den ursprünglichen Koordinaten ab. Was passiert aber, wenn diese Abhängigkeit nichtlinear ist, so wie es der Fall in

Abbildung 3.11. Die nichtlineare PCA findet nichtlineare Korrelationen zwischen den

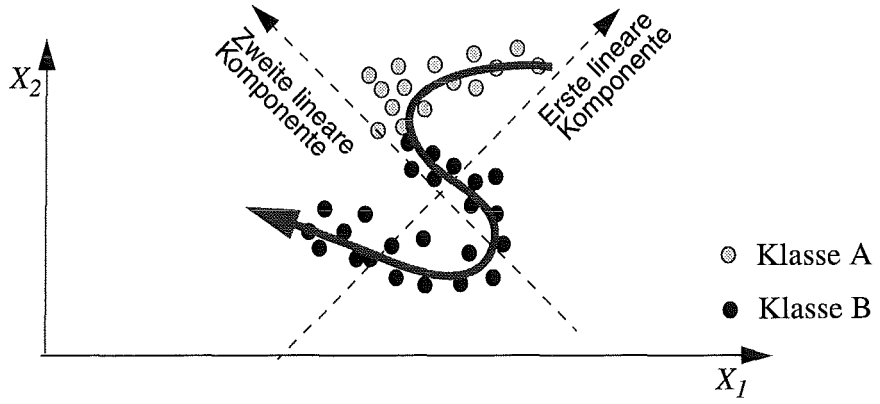


Abbildung 3.11: *Die gestrichelten Linien sind die Koordinaten nach einer linearen PCA-Transformation. Um die Muster zu trennen, wird aber eine nichtlineare Komponente gebraucht, die den Verlauf der dicken Linie hat.*

Eingabedaten und läßt sich mit einem Multilayer-Perceptron mit einer "bottleneck"-Schicht realisieren. Dieses Netz besitzt drei verdeckte Schichten, wobei die Aktivierungsfunktion der ersten und dritten verdeckten Schicht die nichtlineare logistische Funktion ist und die zweite verdeckte Schicht eine lineare Aktivierungsfunktion besitzt. Die zweite verdeckte Schicht ist auch die "bottleneck"-Schicht. Das Netz kann mit Backpropagation-Lernalgorithmus trainiert werden. Damit das Netz nicht überlernt, muß nach einer festen Anzahl von Lernepochen eine Crossvalidierung stattfinden. Dazu wird die Trainingsmenge zufällig in vier gleiche Gruppen aufgeteilt. Für das Lernen werden immer drei der vier Gruppen benutzt. Mit der vierten Gruppe wird das Ergebnis des Lernens überprüft. Dann wird eine andere Gruppe ausgelesen usw.

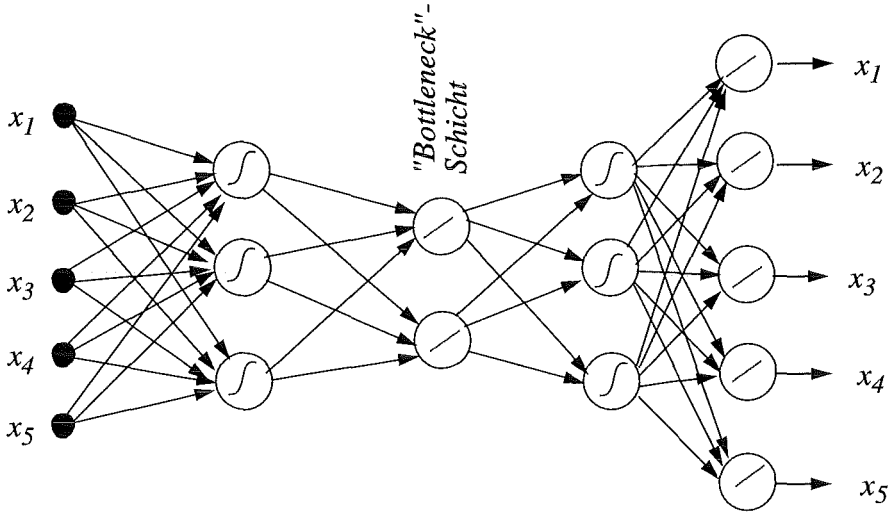


Abbildung 3.12: *Ein Netz mit drei verdeckten Schichten kann für die Berechnung der nichtlinearen PCA-Transformation eingesetzt werden. An der mittleren "Bottleneck"-Schicht werden die nichtlinearen Komponenten abgegriffen. Die Neuronen der ersten und dritten verdeckten Schicht haben eine nichtlineare Aktivierungsfunktion (gekennzeichnet durch ein ∫ in der Neuronendarstellung).*

Abbildung 3.13 zeigt die nichtlineare Transformation der *cancer*-Daten aus der Benchmark-Sammlung Proben1 [Prechelt94].

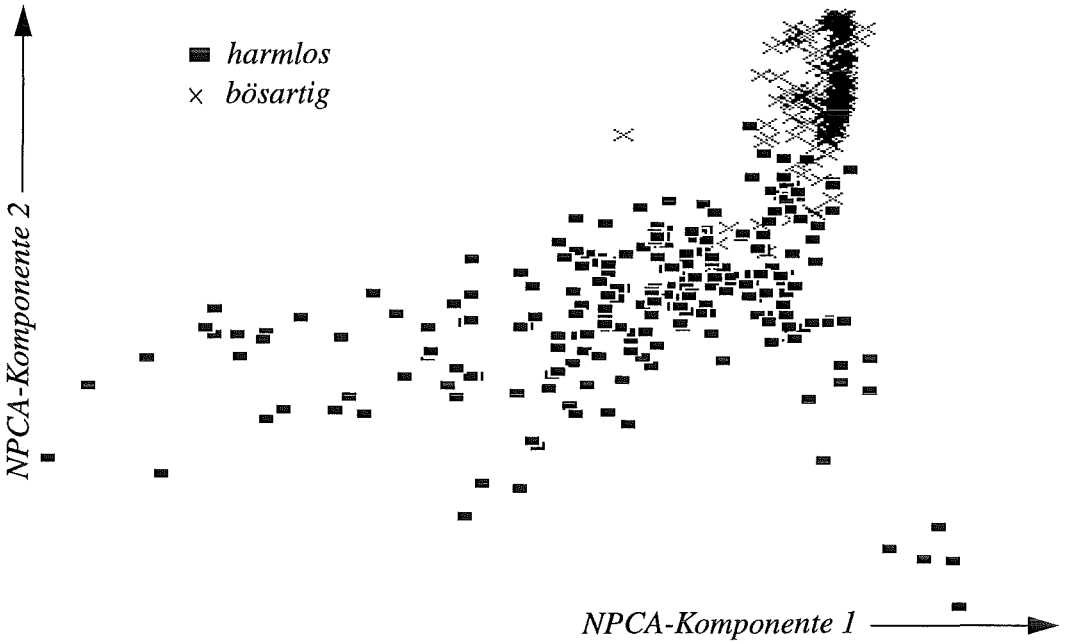


Abbildung 3.13: *Nichtlineare PCA-Transformation der Daten aus Abbildung 3.7.*

Die Darstellung in Abbildung 3.13 ist besser als die in Abbildung 3.7. Es gibt hier zwar auch einen Bereich, wo sich die Muster überlagern. Er ist aber viel kleiner als bei der linearen PCA. Die Erweiterung der PCA zur lokalen PCA (LPCA, s. Kapitel 4.3) wird die Situation der Visualisierung weiter verbessern. Die LPCA ist ein Teil der geometrischen Interpretation, die im folgenden Kapitel beschrieben wird.

3.4 Zusammenfassung

In diesem Kapitel wurden Merkmale aufgelistet, welche eine "ideale" Visualisierung ausmachen. Dann wurden bestehende Methoden der Netz- und Datenvisualisierung besprochen. Dabei sind die Schwachstellen der Verfahren aufgezeigt worden.

Die Betrachtungen sind der Ausgangspunkt für die Entwicklung von GINN. GINN wurde so entwickelt, daß sie die Vorteile aller hier besprochenen Verfahren hat; die Schwächen jedoch nicht. Im Unterschied zu den existierenden Verfahren ist GINN nicht nur eine Visualisierung, sondern erlaubt auch eine Manipulation und Optimierung eines neuronalen Netzes. In der weiteren Beschreibung dienen die hier besprochenen Methoden als Vergleich und als Ausgangspunkt der Erläuterungen. Es wird auch weitgehend die hier eingeführte Terminologie benutzt.

Kapitel 4

Geometrische Interpretation

Dieses Kapitel beschreibt die geometrische Interpretation. Alle Algorithmen werden zuerst für einen einfachen Fall erläutert und dann auf einen allgemeinen erweitert. So wird zuerst die Interpretation eines einzelnen Neurons beschrieben. Danach wird sie auf eine Überlagerung von zwei Neuronen ausgedehnt und diese schließlich auf beliebig viele Neuronen erweitert. Genauso wird verfahren, wenn es um die Visualisierung neuronaler Netze mit mehreren verdeckten Schichten geht. Zuerst wird die Visualisierung für eine verdeckte Schicht erklärt und dann auf mehrere erweitert. Alle Algorithmen werden zuerst für den Fall erläutert, daß die neuronalen Netze nur zwei Eingabeneuronen haben (der zweidimensionale Eingaberaum läßt sich ohne zusätzliche Dimensionenreduktion darstellen). Die Erweiterung auf mehrere Eingabeneuronen ist sehr komplex, so daß diesem Thema ein eigener Abschnitt gewidmet wird, wo die Methode der Visualisierung mit Hilfe der lokalen PCA (LPCA) beschreiben wird.

Neuronale Netze lassen sich einheitlich visualisieren, wenn sie für Klassifikationsaufgaben genutzt werden. Für einen Klassifikator ist nur der Verlauf der sog. *Klassentrennlinie* von Bedeutung. Diese Trennlinie trennt den Eingaberaum des Klassifikators in zwei Klassen auf. Um die Arbeitsweise des Klassifikators zu verstehen genügt es nun, den Verlauf dieser Linien zu untersuchen. Manchmal ist der Bereich (der sog. *Sicherheitsstreifen*) in der Umgebung dieser Linie von Bedeutung.

Die Schwierigkeit bei der Visualisierung neuronaler Netze liegt darin, daß aus der Darstellung zu sehen sein soll, welche Neuronen an der Bildung der Klassentrennlinie beteiligt sind. Ferner soll diese Visualisierung Hinweise geben, welche Neuronen manipuliert werden müssen, um einen bestimmten Trennlinienverlauf zu erhalten. Das alles verlangt, daß die Visualisierung nicht ein einfacher Funktionsgraph, sondern eine Netzparameter berücksichtigende Darstellung ist. Da die Transferfunktion eines Neurons einen Verlauf hat, welcher nur in Übergangsbereich (Sicherheitsstreifen), d. h. dort wo sich der Funktionswert von *0* auf *1* ändert (s. Abbildung 1.3 auf Seite 9), eine signifikante Wertänderung aufweist, muß dieser Bereich stärker berücksichtigt werden als die Bereiche, in denen der Funktionswert nahezu konstant ist.

Die Klassentrennlinie ist die Grenze zwischen zwei Bereichen des Definitionsbereiches, welche unterschiedliche Klassen repräsentieren. Ihre Dimension hängt von der Dimension des Eingaberaumes ab. Sie ist:

- im Falle eines eindimensionalen Eingaberaums ein *Punkt*,
- bei einer zweidimensionalen Eingabe eine *Linie*,
- im dreidimensionalen Raum eine *Fläche* und
- allgemein: in einem n-dimensionalen Eingaberaum eine *(n-1)-dimensionale Hyperebene.*

Das Problem bei der Darstellung der Klassentrennlinie ergibt sich erst dann, wenn eine Klassentrennlinie in einem drei- und höherdimensionalen Eingaberaum dargestellt werden soll. Dann muß sie in einer Form repräsentiert werden, die es erlaubt, sie auf einem zweidimensionalen Bildschirm darzustellen. Die Frage der Darstellung hochdimensionaler Eingaberäume auf einem zweidimensionalen Bildschirm wird erst im Kapitel über die LPCA behandelt. Die folgenden Erläuterungen beziehen sich auf einen zweidimensionalen Eingaberaum.

4.1 Geometrische Interpretation neuronaler Netze

Die primäre Aufgabe der geometrischen Interpretation ist die Visualisierung der Lösung, welche von einem neuronalen Netz gefunden wurde. In der Visualisierung wird der Verlauf der Klassentrennlinie sowie die Lage der verdeckten Neuronen dargestellt. Die Darstellung ist derart beschaffen, daß man daraus die Arbeitsweise eines neuronalen Netzes beurteilen, seine Lösung verstehen und ggf. korrigieren kann. Die Manipulation des Netzes erfolgt direkt durch eine Manipulation der Darstellung. Man arbeitet quasi wie mit einem Graphik-Programm. Aus der graphischen Visualisierung folgt dann (meistens sofort), wie die Neuronen manipuliert werden sollen. Insbesondere kann man auf diese Weise die bessere Generalisierungsfähigkeit eines Menschen ausnutzen. Wenn man eine Lösung sieht und erkennt, daß die Klassentrennlinie einen anderen Verlauf hat, als das erwartet wird, dann kann dieses Fehlverhalten durch eine Manipulation der Darstellung behoben werden. Die Änderungen werden in Echtzeit angezeigt. Damit eine derartige Vorgehensweise möglich ist, bedarf es einer adäquaten Visualisierung. Es stellen sich dabei zwei Fragen. Die erste ist: was wird visualisiert? Und die zweite: wie wird visualisiert?

Was wird visualisiert? Dargestellt wird das ganze Netz, eine bestimmte Schicht oder ein bestimmtes Neuron. In den folgenden Abschnitten wird die Visualisierung von Netzen mit keiner, einer und zwei verdeckten Schichten beschrieben. Dabei entspricht die Visualisierung eines Netzes mit keiner verdeckten Schicht der Visualisierung eines einzelnen Neurons.

Annahmen. In der weiteren Beschreibung werden (zwecks einer einfachen Erklärung) folgende Annahmen getroffen:

- es werden nur klassifizierende vorwärtsgerichtete Netze, ohne Rückkopplung, betrachtet,
- es gibt nur einen Klassentrennwert, welcher von Bedeutung ist (eine Erweiterung auf mehrere Klassentrennwerte ist trivial),
- das Netz hat nur eine Ausgabeneuron und
- alle Neuronen haben eine logistische Aktivierungsfunktion.

Die obigen Annahmen erlauben eine anschauliche Einführung in GINN, ohne daß viele Sonderfälle betrachtet werden müssen. Eine Erweiterung auf manche dieser Sonderfälle, wird zum Schluß gegeben. Die Beschreibung beginnt mit der Visualisierung des Grundbausteins jedes neuronalen Netzes, dem Neuron.

4.1.1 Visualisierung eines einzelnen Neurons (oder neuronalen Netz ohne verdeckte Schichten)

Bevor mit der Erläuterung der Visualisierung begonnen wird, sei hier noch einmal daran erinnert, daß ein Neuron mit einer logistischen Aktivierungsfunktion lediglich eine lineare Aufteilung seines Eingaberaumes vornehmen kann (vergleiche Kapitel 1.4). Die Visualisierung muß deshalb eine Linie darstellen, welche den Eingaberaum in zwei Teile aufgeteilt. Entlang dieser Linie (der Klassentrennlinie) befindet sich ein Sicherheitsstreifen. Im Sicherheitsstreifen vollzieht sich allmählich der Übergang zwischen den zwei Klassen. Für eine Interpretation reicht die Darstellung mit Hilfe der Klassentrennlinie und des Sicherheitsstreifen völlig aus.

Der Sicherheitsstreifen ist der Abschnitt der Visualisierung, in welchem das Neuron am stärksten den Verlauf der Klassentrennlinie beeinflußt, da sich dort der Übergang zwischen den Klassen allmählich vollzieht. Um die Steilheit des Übergangs graphisch darzustellen, wird neben der Klassentrennlinie auch eine *0*- und eine *1*-Linie in die Visualisierung eingebettet. Da aber die logistische Funktion den Wert *0* bzw. *1* nie erreicht, werden nicht diese Höhenlinien dargestellt, sondern zwei Linien, die auf einer Ebene liegen, welche tangential im Wendepunkt die logistische Funktion schneidet (s. Abbildung 4.1: im *1*-dimensionalen Fall, Abbildung 4.2: im *2*-dimensionalen Fall). Diese Darstellungsart ist dadurch gerechtfertigt, daß der größte Teil des Übergangs fast linear ist. Auf diese Weise gibt diese vereinfachte Darstellung den Verlauf der logistischen Funktion am besten wieder. Der Abstand zwischen diesen Linien gibt die Steigung des linearen Übergangs wieder. Ist er steil, d. h. schneller Übergang zwischen den Klassen, dann sind die *0*- und *1*-Linien dicht beieinander (schmaler Sicherheitsstreifen); ist er flach, d. h. langsamer Übergang, dann sind die Linien weit voneinander entfernt (breiter Sicherheitsstreifen).

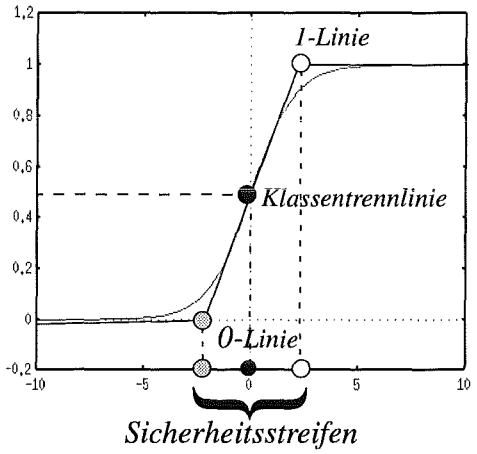


Abbildung 4.1: *Die Plazierung der dargestellten Höhenlinien. In der Visualisierung wird deren Projektion dargestellt (die Projektion auf die X-Achse ergibt 3 Punkte, welche das Neuron repräsentieren).*

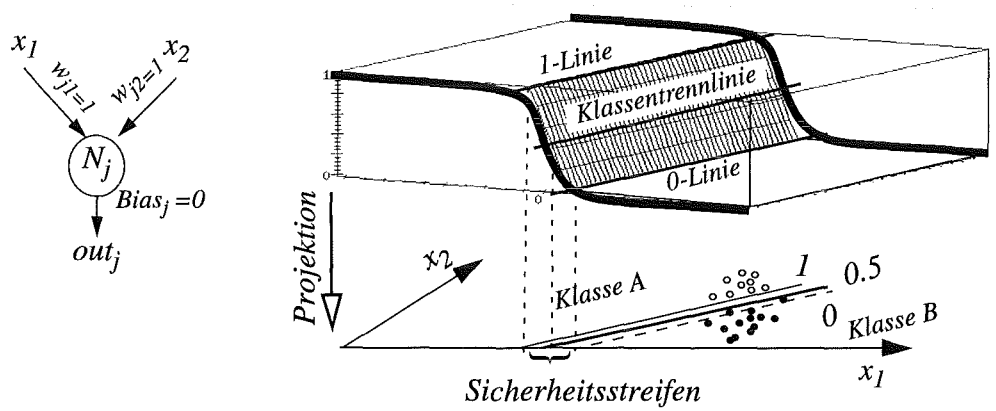


Abbildung 4.2: *Repräsentation eines Neurons in einem klassifizierenden Netz. Das Neuron wird durch die Klassentrennlinie (hier: Höhe = 0.5) visualisiert. Im folgenden ist die 0-Höhenlinie gestrichelt, die 1-Höhenlinie durchgezogen und die Klassentrennlinie fett eingetragen.*

Ein logistisches Neuron teilt seinen Eingaberaum durch eine gerade Trennlinie in zwei Klassen auf. Die Geradengleichung der Klassentrennlinie läßt sich aus den Neuronenparametern ausrechnen und als eine Höhenlinie darstellen (s. Abbildung 4.2). Im Falle der logistischen Aktivierungsfunktion sind alle Höhenlinien zueinander parallele Geraden. Die Formel für eine Höhenlinie mit der Höhe h lautet:

$$x_2 = -\frac{w_{j1}}{w_{j2}} x_1 - \frac{ln\left(\frac{1}{h} - 1\right) - Bias_j}{w_{j2}}. \qquad (4.1)$$

Die Indizierung entspricht der aus Abbildung 4.2.

Neuron als linearer Klassifikator. Weil ein Neuron seinen unmittelbaren Eingaberaum lediglich durch eine gerade Klassentrennlinie teilt (s. Abbildung 4.2), spricht man in diesem Zusammenhang auch von einer linearen Trennung oder einem linearen Klassifikator. Komplexere Trennlinien sind mit Hilfe eines einzelnen logistischen Neurons nicht zu realisieren [Minsky69]. Sie lassen sich aber mit Hilfe eines Netzes mit einer verdeckten Schicht bilden.

4.1.2 Visualisierung eines Netzes mit einer verdeckten Schicht

In einem neuronalen Netz mit einer verdeckten Schicht hängt die Darstellungsform eines Neurons von der Schicht ab, in welcher es sich befindet. Je nach Schicht muß also eine andere Darstellungsform gewählt werden.

Für alle Neuronen (ausgenommen die Eingabeneuronen) gilt aber: Es wird immer die Klassentrennlinie des Neurons visualisiert. Die Darstellung eines verdeckten Neurons der ersten verdeckten Schicht entspricht der eines einzelnen (s. oben). Die Klassentrennlinie ist also eine Gerade. Das Ausgabeneuron kombiniert mehrere verdeckte Neuronen, indem es deren Signale gewichtet summiert und die Nichtlinearität der Aktivierungsfunktion einbringt. Die Klassentrennlinie am Ausgang des Ausgabeneurons kann eine beliebige Kurve oder mehrere Kurven sein. Der Verlauf ergibt sich im wesentlichen aus der Kombination der Klassentrennlinien der verdeckten Neuronen. Nur in bestimmten Bereichen gibt es gewisse Abweichungen. Eine genaue Beschreibung des Verfahrens wird in den folgenden Abschnitten gegeben.

Im ersten Teil der Beschreibung wird erklärt, wie das Netz so visualisiert werden kann, daß auch die Beiträge aller verdeckten Neuronen angezeigt werden können. Dazu ist es zuerst nötig die Überlagerung von zwei verdeckten Neuronen einzuführen. Diese kann dann auf beliebig viele erweitert werden. Auf diese Weise wird es möglich sein, ein Neuron mit einer verdeckten Schicht zu visualisieren.

4.1.2.1 Überlagerung von zwei Neuronen

Ein Ausgabeneuron überlagert die Signale der Neuronen der verdeckten Schicht und erzeugt damit die Netzausgabe. Das Überlagern von zwei Neuronen ist nicht trivial, weil dabei eine Überlagerung von zwei nichtlinearen Funktionen stattfindet. Im Falle einer Klassifikation ist nicht der genaue numerische Wert von Interesse, sondern nur der Verlauf der Klassentrennlinie, so daß in diesem Fall eine Vereinfachung stattfinden kann. Die resultierende Klassentrennlinie ist dann im Prinzip eine Kombination aus den Klassentrennlinien der verdeckten Neuronen, bis auf den Bereich, wo sich die Sicherheitsstreifen überlagern. In diesem Bereich entsteht ein völlig neuer Verlauf. Dieser muß separat berechnet werden.

Für die Berechnung wird aus Zeitkomplexitätsgründen und wegen der geometrischen interaktiven Manipulation anstelle der nichtlinearen Aktivierungsfunktion ihre lineare Approximation genommen (s. Abbildung 4.3). Die Approximation wird mit Hilfe von drei Flächen realisiert. Die beiden waagerechten Halbebenen haben den konstanten Wert *0* bzw. *1*. Sie approximieren den fast konstanten Bereich der logistischen Funktion. Die Steigung des *0*-*1*-Übergangs (Sicherheitsstreifens) wurde so gewählt, daß die approximierende Fläche die logistische Funktion im Wendepunkt berührt. Für die Visualisierung und die Berechnung sind drei Linien von Bedeutung, und zwar die *0*- und die *1*-Linie sowie die Klassentrennlinie. Die *0*-Linie liegt dort, wo der Sicherheitsstreifen die *0*-Halbebene berührt, und die *1*-Linie liegt dort, wo der Sicherheitsstreifen die *1*-Halbebene berührt. Dazwischen, im Sicherheitsstreifen, liegt die Klassentrennlinie.

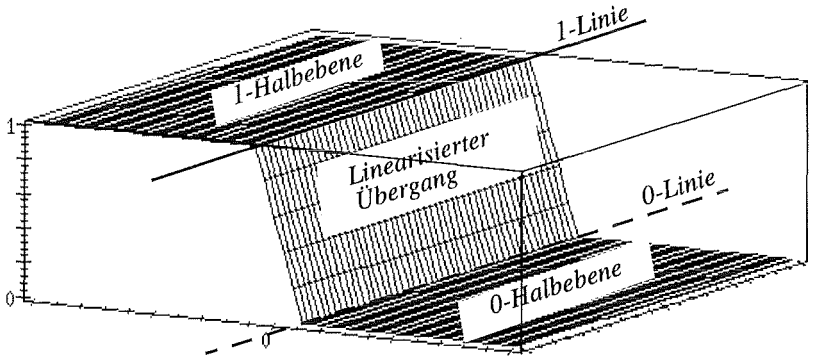


Abbildung 4.3: *Verwendete lineare Approximation der logistischen Funktion.*

Der Sicherheitsstreifen und die Halbebenen können in der Projektion durch Polygonzüge dargestellt werden (s. Projektion in Abbildung 4.2). Durch deren Überlagerung entstehen neue Polygonzüge. Es können sich verschiedene Bereiche der linearisierten Aktivierungsfunktion überlagern. Je nachdem, welche Bereiche es sind, gestaltet sich die Berechnung der Visualisierung unterschiedlich schwer. Folgende Fälle sind möglich:

- **Eine Halbebene überlagert eine Halbebene.** Das Ergebnis ist eine neue Halbebene, mit der Höhe gleich der Summe der Höhen der beiden Halbebenen (s. Abbildung 4.4).

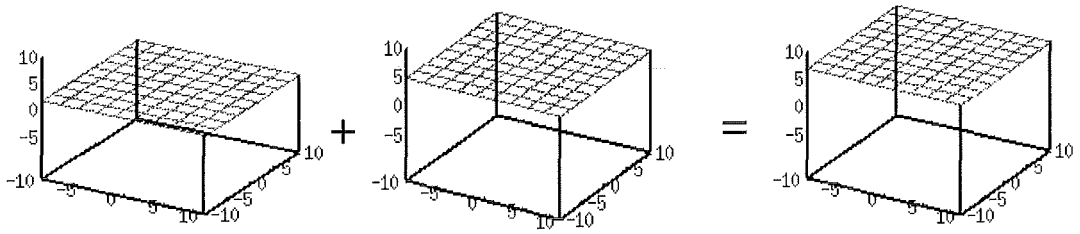


Abbildung 4.4: *Die Überlagerung von zwei Halbebenen.*

- **Eine Halbebene überlagert einen Sicherheitsstreifen.** Der Sicherheitsstreifen wird um die Höhe der Halbebene angehoben. Eine Trennlinie, welche sich auf dem linearen Übergang befindet, wird, je nach dem Vorzeichen der dazu addierten Höhe, nach unten (beim positiven Vorzeichen) bzw. nach oben (beim negativen Vorzeichen) verschoben. Diese Verschiebung ist relativ zur Fläche des linearen Übergangs und kommt deswegen zustande, weil der Übergang angehoben bzw. abgesenkt wird. Da

aber die Höhe der Klassentrennlinie (der Trennwert der Klassen am Netzausgang) unverändert bleibt, rutscht sie in der Darstellung nach oben bzw. nach unten (vergleiche Abbildung 4.5 d und e).

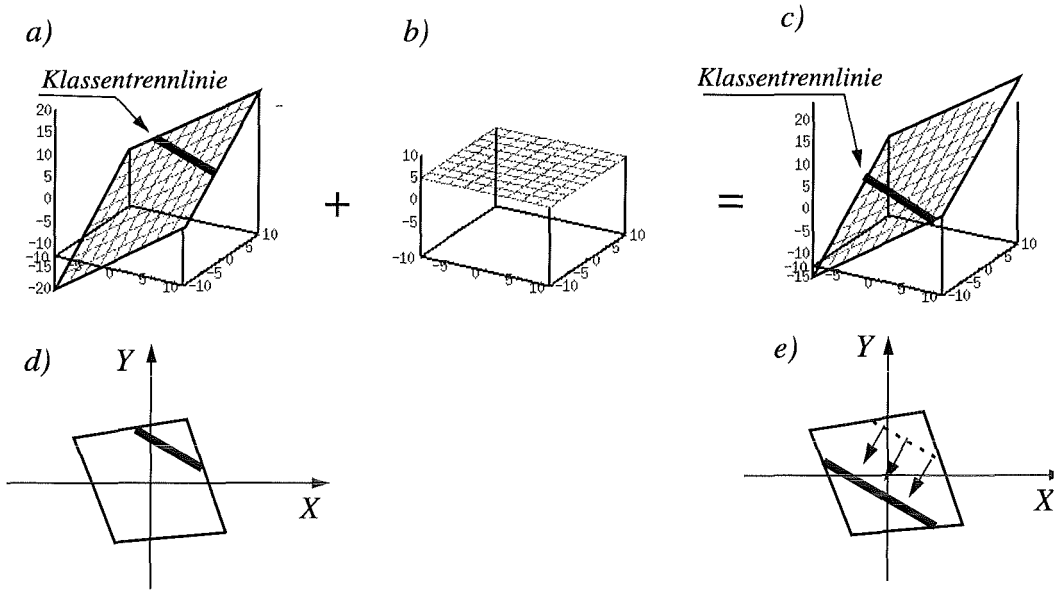


Abbildung 4.5: *Die Überlagerung von einem Sicherheitsstreifen und einer Halbebene. Bild a) zeigt den 3D-Plot eines Sicherheitsstreifens; Bild b) den 3D-Plot einer Halbebene; Bild c) den 3D-Plot des Ergebnisses der Überlagerung. Bilder d) und e) zeigen die Visualisierungen der Bilder a) bzw. c). Es ist deutlich zu sehen, daß sich die Lage der Klassentrennlinie verändert hatte.*

- **Zwei Sicherheitsstreifen überlagern sich.** In diesem Fall entsteht ein neuer Abschnitt des Sicherheitsstreifens. Seine Lage ergibt sich aus der vektoriellen Summe der beiden beteiligten Streifen.

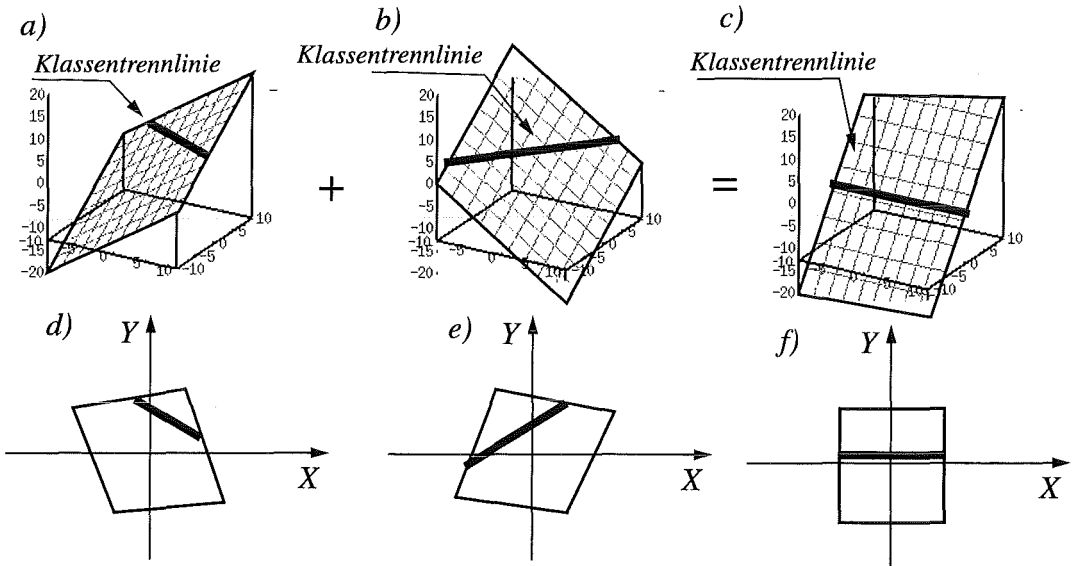


Abbildung 4.6: *Die Überlagerung von zwei linearen Übergängen. Bilder a) und b) zeigen die zu überlagernden linearen Übergänge, das Bild c) das Ergebnis der Überlagerung. Die entsprechenden Visualisierungen sind in den Bildern d), e) und f) dargestellt.*

Der Verlauf der Klassentrennlinie ändert sich bei dieser Überlagerung. Er verschiebt sich nicht nur, sondern es kommt auch zu einer Änderung seiner Richtung.

Diese drei Fälle resultieren aus einer Betrachtung der Linearkombination verdeckter Neuronen. Die Formel für diese Linearkombination lautet:

$$net_{out} = \sum_{k=1}^{m} w_{out,k}\sigma\left(\sum_{i=1}^{n} w_{ki}x_i + Bias_k\right) \tag{4.2}$$

mit:

x_i das i-te Eingabemerkmal,
w_{ki} das Gewicht vom i-ten Eingabeneuronen zum k-ten verdeckten Neuron,
$w_{out,k}$ das Gewicht vom k-ten verdeckten Neuron zum Ausgabeneuron,
σ eine sigmoide Aktivierungsfunktion (hier: die logistische Funktion) und
$Bias_k$ der Bias des k-ten Neurons.

Abhängig von den Eingabeaktivierungen x_i sind in manchen Bereichen alle oder mehrere Terme $w_{out,k}\sigma(\sum_{i=1}^{n} w_{ki}x_i)$ nahezu konstant. Diese Tatsache wurde bei der Visualisierung ausgenutzt. Diese Bereiche werden als konstante Halbebenen repräsentiert. Dadurch vereinfacht sich die Berechnung, da sich der wesentliche Aufwand auf die verbleibenden linearen Übergänge konzentriert. Aufgrund der Assoziativität und Kommutativität der Summation wird die Darstellung nicht auf einmal, sondern durch eine geschickte Zusammenfassung der Summanden berechnet, die eine einfache Reduzierung der Berechnung auf die zuvor definierten drei Überlagerungsfälle erlaubt. Dazu wird ein visualisierter Bereich in Teilbereiche (Polygonzüge) aufgeteilt. Die Aufteilung erfolgt entlang der *0*- und *1*-Linien der Neuronen. Für jeden der entstandenen Polygonzüge wird die Überlagerung unter Berücksichtigung der oben aufgelisteten drei Fälle separat berechnet. Der Verlauf der Klassentrennlinie wird ebenfalls in jeden dieser Polygonzüge separat eingetragen.

Beispiel 4.1. Abbildung 4.7 zeigt die Superposition von zwei Neuronen. Separat betrachtet teilt jedes Neuron den Eingaberaum diagonal auf (gestrichelte Linien in Abbildung 4.7b und Abbildung 4.7c). Die Überlagerung beider Neuronen ergibt jedoch eine trapezähnliche Aufteilung des Eingaberaumes (durchgezogene Linien in Abbildung 4.7b und Abbildung 4.7c. Im Bereich der Trapezspitze weicht der Trennlinienverlauf von einer Kombination der gestrichelten Linien ab. Dieser Bereich ist besonders bei Netzen mit nur einer verdeckten Schicht und kontinuierlichem Eingabebereich von großer Bedeutung, da auf diese Weise konkave Trennlinien gebildet werden können (s. Beispiel 4.2).

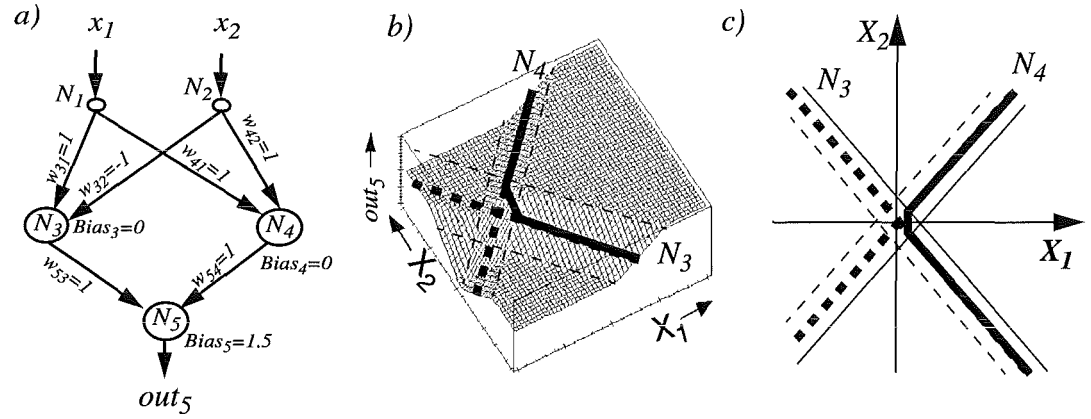


Abbildung 4.7: *Beispiel für die Summe von zwei Neuronen. Bild a) zeigt das visualisierte Netz, Bild b) den 3D-Plot der Netzausgabe und Bild c) die geometrische Visualisierung.*

■

4.1.2.2 Überlagerung von mehreren verdeckten Neuronen

Die Berechnung der Überlagerung mehrerer verdeckter Neuronen erfolgt nach denselben Schemata wie bei zwei Neuronen. Der Unterschied liegt in der komplexeren Aufteilung des Eingaberaumes. Es überlagern sich jetzt nicht nur zwei, sondern mehrere Bereiche. Die Berechnungen erfolgen schrittweise nach denselben Methoden wie oben. Sie können in beliebiger Reihenfolge durchgeführt werden, da die Operatoren kommutativ und assoziativ sind. Der Algorithmus ist in Kapitel 4.1.7 ausführlich erläutert. Um die Bedeutung der Überlagerung der linearen Übergänge zu demonstrieren, betrachten wir das sog. *Schachbrettproblem.*

Beispiel 4.2. Das Schachbrettproblem ist ein klassisches Klassifikationsproblem, das eine Erweiterung des XOR-Problems auf einen kontinuierlichen Wertebereich ist. Es geht dabei um die Aufteilung von Punkten anhand ihrer kartesischen Koordinaten in zwei Klassen, nämlich in eine weiße und in eine schwarze Klasse. Die Klassenaufteilung ist so gewählt, daß sie das kartesische Koordinatensystem in ein *2x2* großes schwarzweißes Schachbrettmuster aufteilt.

Dieses Problem wird mit einem neuronalen Netz mit zwei Eingabe-, vier verdeckten Neuronen und einem Ausgabeneuron gelöst. Die Parameter des Netzes sind: $w_{31}=-8$ $w_{32}=-8$, $Bias_3=9.99$, $w_{41}=-8$, $w_{42}=8$, $Bias_4=2.1$, $w_{51}=-8$, $w_{52}=-8$, $Bias_5=6.0$, $w_{61}=8$, $w_{62}=-8$, $Bias_6=1.8$, $w_{73}=50$, $w_{74}=-50$, $w_{75}=-50$, $w_{76}=--50$, $Bias_7=-50$. Diese Parameter stellen sich nach einer Trainingsphase und einer Manipulierung des Netzes (wie in Kapitel 5.1.1.4 beschrieben) ein.

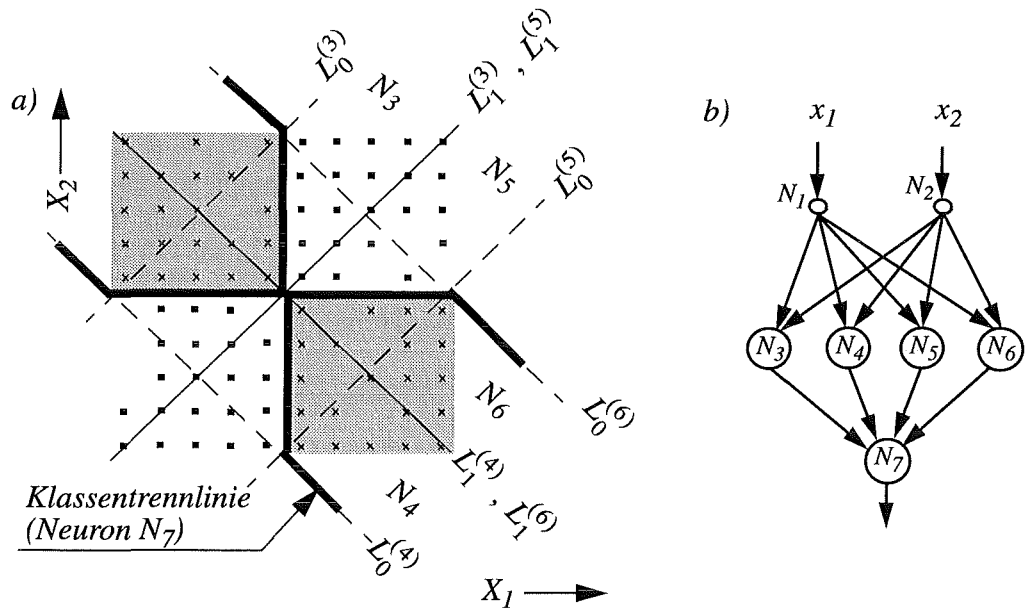


Abbildung 4.8: *Das Schachbrettproblem. Bild a): Die geometrische Visualisierung und Bild b): das verwendete Netz. Die Schreibweise:* $L_j^{(n)}$ *bezeichnet eine Linie der Höhe j des Neurons n.*

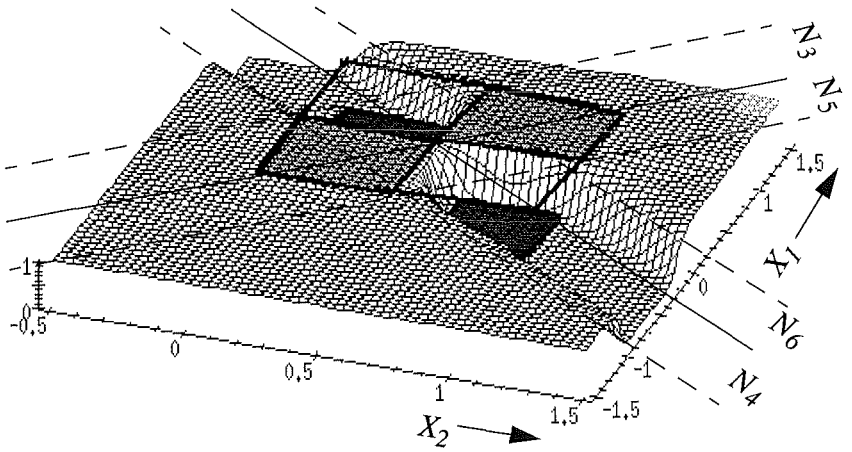


Abbildung 4.9: *Der Funktionsplot der Lösung aus Abbildung 4.8 mit eingetragenen 0- und 1-Höhenlinien der verdeckten Neuronen.*

■

In Beispiel 4.2 wird ein sog. *konkaves Problem* behandelt, d. h. es gibt Punkte einer Klasse (z. B. Klasse A), welche sich durch eine Linie verbinden lassen, die auch Muster der anderen Klasse (Klasse B) beinhaltet. Eine Trennung derartiger Klassen mit einem Netz, das nur eine verdeckte Schicht hat, kann nur auf eine Weise erreicht werden, welche den Sicherheitsstreifen ausnutzt. In unserem Beispiel liegt das Schachbrett genau dort, wo sich alle Sicherheitsstreifen schneiden, d. h. es entsteht durch die Überlagerung von vier Neuronen. Die Berechnung der Darstellung in Abbildung 4.8a wurde mit Hilfe der geometrischen Interpretation durchgeführt, indem die Methoden aus Kapitel 4.1.2 verwendet wurden. Das Beispiel rechtfertigt die Notwendigkeit einer komplizierten Berechnung der Überlagerung von Neuronen und zeigt gleichzeitig die Mächtigkeit eines vorwärtsgerichteten neuronalen Netzes mit einer verdeckten Schicht, nämlich die Fähigkeit, alle "normalen" Probleme (s. Kapitel 1.4) zu lösen (das Schachbrettproblem ist ein nichttriviales konkaves Problem).

In der bisherigen Beschreibung ist angenommen worden, daß die Klassentrennlinie korrekt eingezeichnet wird, ohne zu erklären, wie dies geschieht. In einer Visualisierung muß jedoch die Klassentrennlinie sowohl in die Darstellung der verdeckten Neuronen als auch in die Visualisierung des ganzen Netzes eingezeichnet werden. Das Verfahren dafür arbeitet im sog. *Rückwärtspaß*, in dem der Klassentrennwert vom Netzausgang bis zum Netzeingang zurückberechnet wird. Die Lage der Trennlinie eines Neurons hängt vom Wert dieser Rückberechnung am konkreten Neuron ab.

4.1.2.3 Einzeichnung der Klassentrennlinie

Die Visualisierung stellt die Klassentrennlinie dar. Sie ist für den großen Teil der Anwendungen, nämlich für die klassifizierenden Aufgaben, die wesentliche Information, welche eine Auskunft über die Funktionsweise eines Netzes gibt. Die Lage dieser Trennlinie hängt vor allem vom Schwellenwert (Klassentrennwert) t ab, welcher am Netzausgang die Muster zweier Klassen trennt. Die Position oder besser gesagt die Höhe h der Klassentrennlinie wird in einer Rückwärtsberechnung vom Ausgabeneuron zum Eingabeneuron ermittelt. Die Höhe ist der Klassentrennwert, bezogen auf ein konkretes Neuron oder allgemein einen Punkt im Netz, d. h. hat ein konkretes Neuron an seinem Ausgang eine Aktivierung gleich der Höhe h oder liegt diese an einem Punkt (z. B. einem Gewicht) an, dann wird am Netzausgang einen Aktivierung erzeugt, welche gleich dem Klassentrennwert t ist.

Die Höhe h einer Klassentrennlinie hängt vom Klassentrennwert t am Netzausgang ab. Für ein Neuron läßt sich diese durch das Auflösen der folgenden Formel nach h ermitteln:

$$t = act(h) = \frac{1}{1 + e^{-(h)}}. \tag{4.3}$$

Die Auflösung lautet:

$$h = -ln\left(\frac{1}{t} - 1\right). \tag{4.4}$$

Die Gleichung besagt nichts anderes, daß die Netzaktivierung des Neurons gleich h ist, wenn am Netzausgang der Trennwert t anliegt. Die Klassentrennlinie kann in den Eingaberaum des Neurons eingezeichnet werden. Die Geradengleichung lautet:

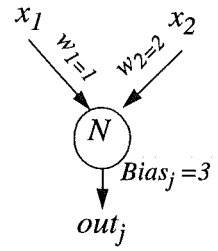
$$w_1 x_1 + w_2 x_2 - Bias = h,$$

mit:
w_1, w_2 den Gewichten von der Eingabeschicht,
x_1, x_2 den Merkmalen, welche am Neuron anliegen und
Bias dem Bias des Neurons.

Das Verhältnis $-w_2 / w_1$ bestimmt die Richtung der Geraden a, der Ausdruck $\frac{Bias + h}{w_2}$ die Verschiebung b entlang der x_1-Achse (beides gilt, falls auf der X-Achse das Merkmal x_1 aufgetragen ist).

Beispiel 4.3. Visualisierung eines Neurons mit den Gewichten $w_1=1$ und $w_2=2$ sowie dem *Bias* = *3*. Der Trennwert t am Netzausgang beträgt *0.5*. Als erstes wird die Höhe der Klassentrennlinie h nach der Formel (4.4) berechnet. Ihr Wert ist $h=0$. Die Richtung der Geraden ist $a=-2$ und die Verschiebung $b=1.5$. Die Gleichung der Trenngeraden ist somit: $x_2=-2x_1+1.5$.

a)



b)

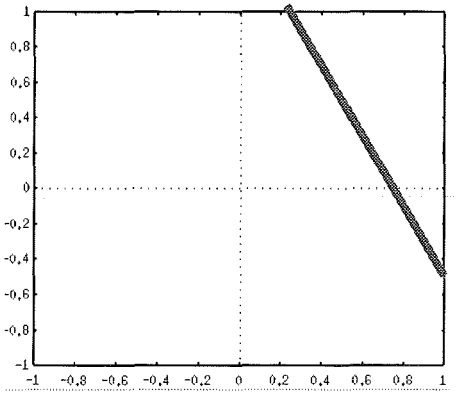


Abbildung 4.10: *Visualisierung eines Neurons. a): Das visualisierte Neuron. b): Die geometrische Visualisierung. Die fette Linie ist die Höhenlinie des Neurons N.*

■

Komplexer gestaltet sich das Einzeichen der Klassentrennlinie in die Neuronendarstellung der verdeckten Schicht. Die Rückwärtsrechnung muß jetzt über zwei Neuronen gehen, nämlich das Ausgabe- und das verdeckte Neuron. Weil in der Darstellung nur der Einfluß des betrachteten Neurons zu sehen sein soll, werden bei der Berechnung nur die Gewichte berücksichtigt, welche auf dem Pfad zum Ausgabeneuron liegen. Alle anderen können für die Dauer der Berechnung auf Null gesetzt werden.

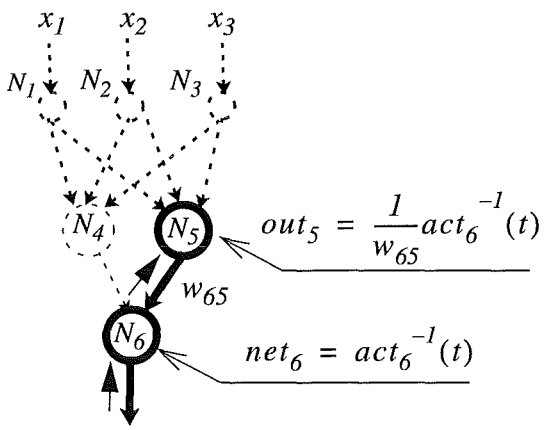


Abbildung 4.11: *Bei der Bestimmung der Höhe der Klassentrennlinie am Neuron N_5 wird entlang des Pfades von der Netzausgabe die Aktivierung des Trennwerts zurückberechnet.*

Beispiel 4.4. Abbildung 4.11 zeigt die Eintragung der Höhenlinie am Beispiel des verdeckten Neurons N_5. Die Höhe ist gleich der Aktivierung out_5. Die Berechnung bedeutet, daß die Aktivierung out_5 des Neurons N_5 gesucht wird, die am Netzausgang eine Aktivierung gleich dem Klassentrennwert t hervorruft. Dabei bleibt der Einfluß anderer Neuronen im Netz unberücksichtigt.

Im ersten Schritt der Berechnung wird der Klassentrennwert t durch das Neuron N_6 zurückpropagiert. Das Neuron N_6 verändert den Wert t gemäß der inversen Aktivierungsfunktion act_6^{-1} und liefert als Ergebnis die Netzaktivierung net_6 des Neurons N_6. Diese Aktivierung wird anschließend auf die Ausgabe out_5 des Neurons N_5 umgerechnet, indem die inverse Wirkung des Gewichts w_{65} (durch den Faktor $1/w_{65}$) berücksichtigt wird. Formal ergibt sich (für eine logistische Aktivierungsfunktion) die folgende Gleichung für die Berechnung der Höhenlinie:

$$out_5 = \frac{ln\left(\frac{1}{t} - 1\right) + Bias_6}{w_{65}}. \qquad (4.5)$$

■

4.1.3 Visualisierung eines Netzes mit zwei verdeckten Schichten

Die Visualisierung der Neuronen in der ersten und zweiten verdeckten Schicht ist analog zu der Visualisierung eines Netzes mit einer verdeckten Schicht, mit dem Unterschied, daß jetzt die Neuronen der zweiten verdeckten Schicht wie Ausgabeneuronen eines Netzes mit einer verdeckten Schicht dargestellt werden. Erst die Visualisierung des Ausgabeneurons gestaltet sich anders, da die Aufgabe eines verdeckten Neurons der zweiten verdeckten Schicht nicht nur die Linearkombination der Signale der Neuronen der ersten verdeckten Schicht ist, sondern auch die Weitertransformation der Netzeingabe. Die Sigmoidfunktion unterdrückt gewisse Bereiche der Netzaktivierung, so daß in vielen Fällen der Rechenaufwand für die Berechnung der Visualisierung nur unwesentlich anwächst. Die Aufgaben der Neuronen der zweiten verdeckten Schicht unterscheiden sich also etwas von der eines Ausgabeneurons eines Netzes mit einer verdeckten Schicht.

Die Neuronen der zweiten verdeckten Schicht bilden nicht nur eine Linearkombination der Neuronenausgaben der ersten verdeckten Schicht, sondern realisieren auch noch zusätzlich eine Schwellenfunktion. Die Schwellenfunktion bewirkt, daß manche Wertebereiche der Netzaktivierung auf *0* bzw. *1* abgebildet werden. Welche Bereiche es sind, hängt von den Gewichten zu der ersten verdeckten Schicht und dem Bias ab. Gleichgeschaltet werden diejenigen Wertebereiche, welche – nach der Gewichtung – im Sättigungsbereich der logistischen Funktion liegen.

Beispiel 4.5. In Abbildung 4.12 wurde das Netz aus Beispiel 4.1 um eine zweite verdeckte Schicht erweitert. Man sieht sofort – sowohl in dem Funktionsgraphen als auch in der geometrischen Visualisierung – daß viele der Polygonzüge im Vergleich zum Beispiel 4.1 zusammengefaßt wurden. Abbildung 4.12c zeigt die Visualisierung des Neurons N_5 der zweiten verdeckten Schicht. Zum Vergleich zeigt die Abbildung 4.12d die Visualisierung des Ausgabeneurons des Netzes aus Beispiel 4.1. Die Darstellung eines Neurons der zweiten verdeckten Schicht unterscheidet sich von der Darstellung eines Neurons der ersten verdeckten Schicht nur dadurch, daß die Klassentrennlinie einen komplizierteren Verlauf hat (keine Gerade mehr) und somit eine kompliziertere Aufteilung des Eingaberaumes vornimmt.

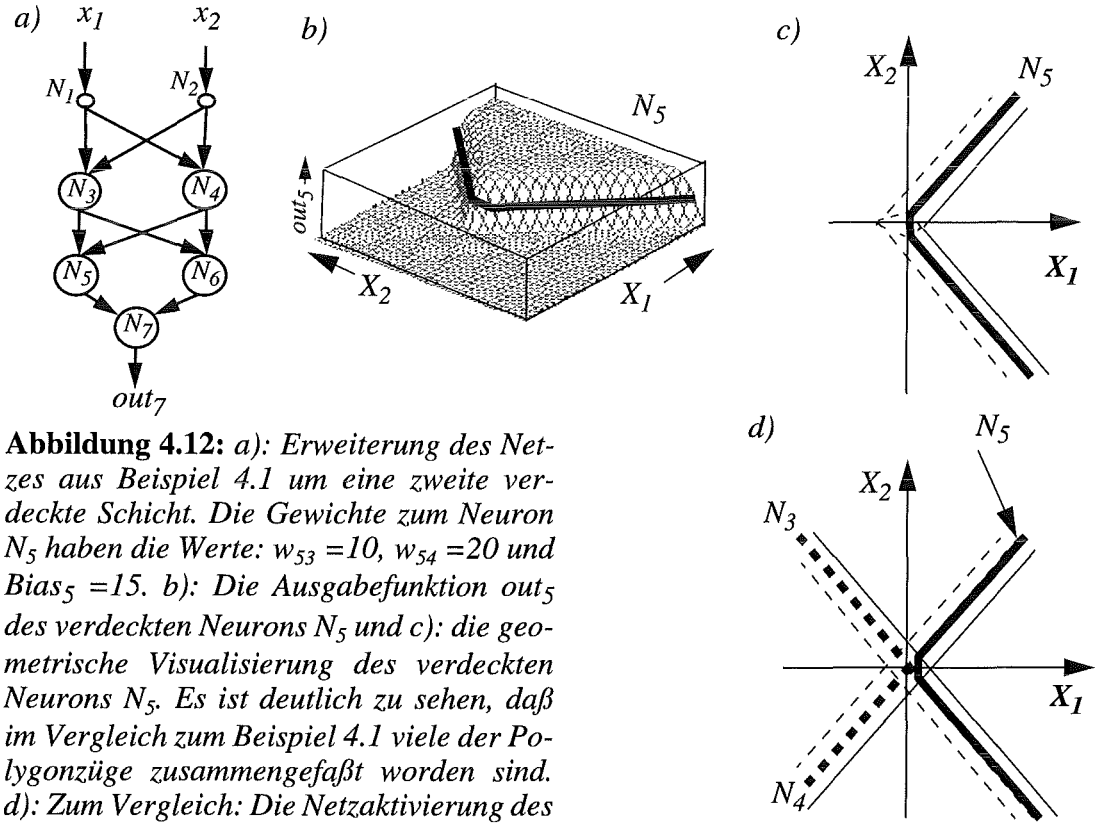


Abbildung 4.12: *a): Erweiterung des Netzes aus Beispiel 4.1 um eine zweite verdeckte Schicht. Die Gewichte zum Neuron N_5 haben die Werte: w_{53} =10, w_{54} =20 und $Bias_5$ =15. b): Die Ausgabefunktion out_5 des verdeckten Neurons N_5 und c): die geometrische Visualisierung des verdeckten Neurons N_5. Es ist deutlich zu sehen, daß im Vergleich zum Beispiel 4.1 viele der Polygonzüge zusammengefaßt worden sind. d): Zum Vergleich: Die Netzaktivierung des Neurons N_5.*

■

Das Ausgabeneuron eines Netzes mit zwei verdeckten Schichten überlagert die Signale der Neuronen der zweiten verdeckten Schicht. Bei der Berechnung muß nur berücksichtigt werden, daß die Funktionen der verdeckten Neuronen komplexer sind, d. h. bei der Überlagerung werden jetzt neben Halbebenen auch Polygonzüge überlagert. Die Berechnungsregeln bleiben jedoch dieselben. Wie jetzt ein Netz visualisiert werden kann, zeigt das nächste Beispiel.

Beispiel 4.6. Visualisierung eines Netzes mit zwei verdeckten Schichten (Netz aus Abbildung 4.12). Die Gewichte zum verdeckten Neuron N_6 sind: $w_{63} = 15$, $w_{64} = 10$ und der $Bias_6 = -12$. Die Gewichte zum verdeckten Neuron N_7 sind: $w_{75} = 1$, $w_{76} = 1$ und der $Bias_6 = 0.5$. Die Überlagerung im Neuron N_7 addiert die Neuronen N_5 und N_7. Aufgrund der Gewichte zum Ausgabeneuron wird das Neuron N_5 gespiegelt, so daß als endgültige Lösung sich die Klassenaufteilung ergibt, wie sie im rechten Teil des Bildes dargestellt ist.

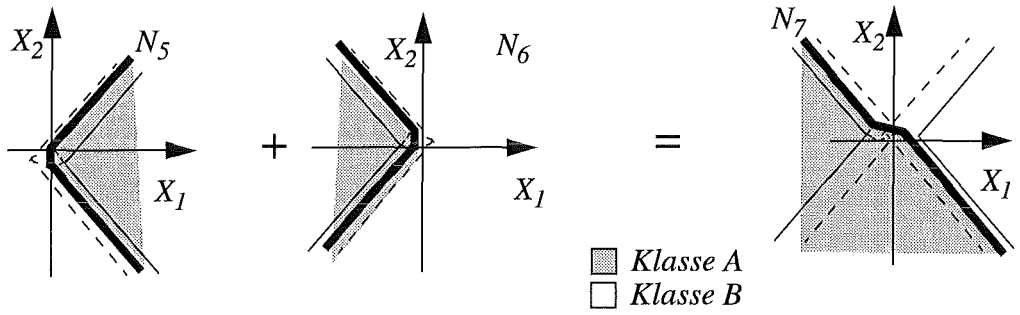


Abbildung 4.13: *Die Überlagerung der Neuronen N_5 und N_6 durch das Neuron N_7 (Netztopologie: s. Abbildung 4.12).*

■

4.1.4 Visualisierung von drei und mehr verdeckten Schichten

In jeder weiteren verdeckten Schicht eines neuronalen Netzes wird eine gewichtete Linearkombination der Aktivierungsfunktionen der Neuronen der vorhergehenden Schicht gebildet, d. h. es finden weitere Transformationen des Eingaberaumes statt. Aus der Sicht eines Neurons bedeutet das nichts anderes, als daß in bezug auf die Netzeingabe alle Neuronen der vorhergehenden Schicht eine komplexere Aktivierungsfunktionen haben. Somit kann derselbe Algorithmus verwendet werden wie für die Berechnung der Höhenlinie der ersten verdeckten Schicht. Der Unterschied liegt zum einen in der Aufteilung des Eingaberaumes, welcher wegen einer komplexeren Aktivierungsfunktion der vorhergehenden Neuronen komplizierter ausfällt, und zum anderen in der Schwellenoperation, welche durch die logistische Funktion realisiert wird. Der Berechnungsaufwand erhöht sich zwar proportional zur Anzahl der Neuronen. Er ist aber keine prinzipielle Hürde für die Visualisierung mehrschichtiger Netze. Als Beispiel wird in diesem Abschnitt die Visualisierung eines Netzes mit zwei verdeckten Schichten beschrieben.

4.1.5 Generelle Vorgehensweise

Die obigen Ausführungen zeigen, wie man ein Netz mit keiner, einer und zwei verdeckten Schichten visualisieren kann. Um die Funktion eines Neurons N_i in einem beliebigen mehrschichtigen Netz darzustellen, bleiben zunächst die Beiträge aller anderen Neuronen in der betrachteten Schicht unberücksichtigt. Die Höhe der Klassentrennlinie h für das betrachtete Neuron wird berechnet. Der Verlauf dieser Trennlinie wird im Eingaberaum mittels einer stückweise linearen Approximation berechnet (analog wie oben). In der Praxis ist diese Visualisierung allerdings sehr rechenintensiv und eine Echtzeitvisualisierung recht langsam. Da aber neuronale Netze mit mehr als zwei verdeckten Schichten selten anzutreffen sind, wird in dieser Richtung kein weiterer Aufwand getrieben.

4.1.6 Interpretation der Neuronen

Bisher wurde nur erläutert, wie man ein neuronales Netz bzw. ein konkretes Neuron darstellen kann. Die Bedeutung der Neuronen für die Lösung ist aber das eigentliche Ziel einer Interpretation. In diesem Abschnitt wird beschrieben, wie die Neuronen, abhängig von der Schicht, interpretiert werden können. Eine Interpretation soll vor allem helfen, die innere Repräsentation des Netzes zu verstehen, um Aussagen über die Qualität zu formulieren oder Korrekturen vornehmen zu können. Die Beschreibung beschränkt sich hier jedoch auf den Fall einer kontextfreien Klassifikation, sie kann aber in den meisten Fällen auf einen konkreten Fall übertragen werden, wenn man die Bedeutung der Klassen in der Interpretation berücksichtigt. Die Beschreibung arbeitetet sich Schicht für Schicht durch. Für jede Schicht wird die Interpretation in zwei Fällen angegeben, näm-

lich für den Fall, daß ein Neuron das Ausgabeneuron ist und für den Fall, daß das Neuron ein verdecktes Neuron ist. Zusätzlich wird diskutiert, wann ein Neuron ein sog. *überflüssiges Neuron* ist, d. h. ein Neuron, welches im Definitionsbereich des Problems keinen Einfluß auf die Lösung hat. Ein derartiges Neuron kann aus dem Netz entfernt werden (Topologieoptimierung). Eine Beschreibung der Optimierungsalgorithmen ist in Kapitel 5.2 gegeben.

4.1.6.1 Neuronen der Eingabeschicht

Diese Neuronen führen keine Operationen durch, sie sind lediglich Eingabedatenspeicher. Aus diesem Grunde werden sie auch nicht separat dargestellt. Sie können aber als die Achsen des Koordinatensystems interpretiert werden, welche den Eingaberaum aufspannen und in dem die Visualisierung (ggf. transformiert) dargestellt wird.

4.1.6.2 Neuronen der ersten verdeckten Schicht

Wie in Kapitel 4.1.1 erläutert wurde, teilen die Neuronen der ersten verdeckten Schicht den Eingaberaum durch eine gerade Linie in zwei Halbebenen auf (im Falle der logistischen Aktivierungsfunktion) und sind somit relativ einfach zu interpretieren. Die richtige Lage (Höhe) der Trennungsgeraden wird unter Berücksichtigung des Klassentrennwertes am Netzausgang berechnet. Dadurch kann der Beitrag eines verdeckten Neurons zur Gesamtlösung bestimmt werden. Wenn man sich die Visualisierung genauer ansieht (z. B. Abbildung 4.7 auf Seite 43), dann erkennt man schnell, daß eine Veränderung des Trennlinienverlaufs nur durch den Übergangsbereich der Aktivierungsfunktion bewirkt werden kann, d. h. die Trennlinie ändert ihren Verlauf nur dann, wenn sie den Sicherheitsstreifen zwischen der *0*- und der *1*-Höhenlinie schneidet. Daraus kann man folgern, daß ein Neuron der ersten verdeckten Schicht ein überflüssiges Neuron ist, wenn sein Sicherheitsstreifen außerhalb des Definitionsbereiches liegt. Das ist eine sehr einfache Bedingung, die automatisch überprüft werden kann.

Beispiel 4.7. In Abbildung 4.14 ist die Interpretation von zwei Neuronen der ersten verdeckten Schicht dargestellt. Jedes Neuron teilt den Eingaberaum durch eine gerade Trennlinie (jeweils die mittlere Linie in der Darstellung des Neurons) in zwei Klassen auf. Das Neuron N_2 ist ein überflüssiges Neuron, da seine Trennlinie und der Sicherheitsstreifen außerhalb des Definitionsbereiches liegen. Dieses Neuron kann entfernt werden. Der konstante Beitrag muß aber, durch eine Veränderung des Bias-Wertes von Neuronen der nachfolgenden Schicht, berücksichtigt werden. ■

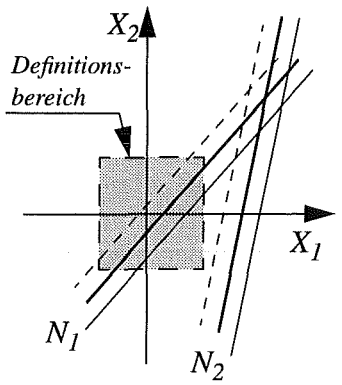


Abbildung 4.14: *Interpretation der Neuronen der ersten verdeckten Schicht als Klassifikatoren.*

4.1.6.3 Neuronen der zweiten verdeckten Schicht (das Ausgabeneuron eines Netzes mit einer verdeckten Schicht)

Neuronen der zweiten verdeckten Schicht werden genauso wie Neuronen der ersten verdeckten Schicht durch die Klassentrennlinie und den Sicherheitsstreifen dargestellt. Sie

realisieren aber eine komplexere Aufteilung der Netzeingabe, indem sie diese noch einmal transformieren. Die logistische Aktivierungsfunktion realisiert eine Sättigungsschwelle (s. Abbildung 1.3). Dadurch fallen Wertebereiche zusammen. Für die Interpretation kann diese Transformation durch eine kompliziertere Netzaktivierungsfunktion der Neuronen der ersten verdeckten Schicht ausgedrückt werden. Der Klassentrennwert wird in der Visualisierung dieser komplizierteren Netzaktivierungsfunktion mit Hilfe einer Höhenlinie visualisiert. Ihre Lage hängt zum einen vom Klassentrennwert am Netzausgang und zum anderen von den Gewichten ab.

Beispiel 4.8. Abbildung 4.15 zeigt die Interpretation von zwei verdeckten Neuronen der zweiten verdeckten Schicht. Beide Neuronen realisieren eine Überlagerung der verdeckten Neuronen der ersten verdeckten Schicht auf eine andere Weise. Ob ein Neuron in der zweiten verdeckten Schicht überflüssig ist, kann auf dieselbe Art festgestellt werden, wie das bei der ersten verdeckten Schicht der Fall war. ■

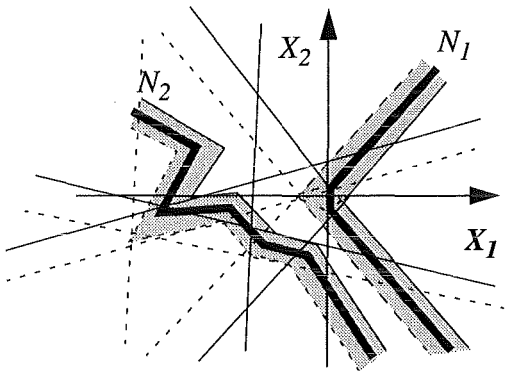


Abbildung 4.15: *Visualisierung von zwei verdeckten Neuronen der zweiten verdeckten Schicht. Die schraffierten Bereiche entsprechen den linearen Übergängen. Die übrigen Linien zeigen die 0- bzw. 1-Höhenlinien der Neuronen der ersten verdeckten Schicht.*

Hat das neuronale Netz nur eine verdeckte Schicht dann ist bereits die komplexere Aktivierungsfunktion mit eingezeichneten Höhenlinien die Interpretation des Ausgabeneurons. Bei Netzen mit einer zweiten verdeckten Schicht, muß noch die Rücktransformation des Klassentrennwertes bei der Interpretation berücksichtigt werden, d. h. jedes Neuron der zweiten verdeckten Schicht bildet den Klassentrennwert weiter auf die Neuronen der ersten verdeckten Schicht ab.

Eine weitere Aufgabe der Neuronen in der zweiten verdeckten Schicht ist die Schwellenwertoperation. Dabei wird die Netzaktivierung auf den Wertebereich *[0, 1]* abgebildet, d. h. Netzaktivierungswerte, welche weit entfernt vom Bias-Wert liegen, werden – je nach Neuronenparametern – auf einen der diskreten Werte *0* bzw. *1* abgebildet. Damit wird auch die lokale Wirkung des "schrägen" Übergangsbereiches verstärkt bzw. gleichgeschaltet.

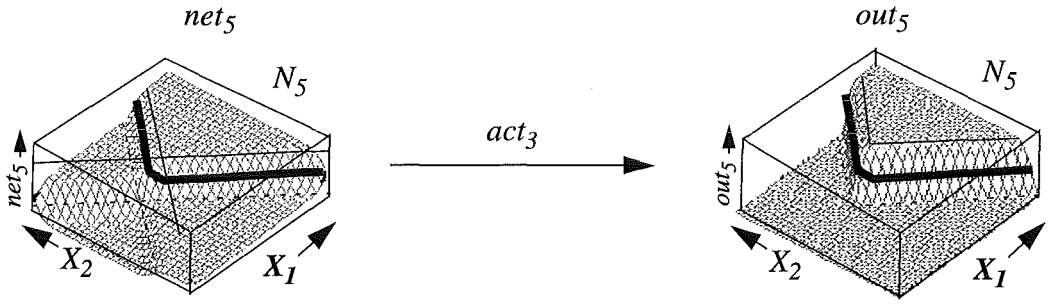


Abbildung 4.16: *Die Wirkung der logistischen Aktivierungsfunktion als Schwellenwertoperator.*

4.1.6.4 Neuronen der dritten verdeckten Schicht oder das Ausgabeneuron eines

zweischichtigen Netzes
Die Interpretation der Neuronen der dritten oder einer höheren Schicht ist schwieriger. Der Eingaberaum wird durch Neuronen der ersten und der zweiten Schicht transformiert und kann durch komplexe Aktivierungsfunktionen der Neuronen der zweiten verdeckten Schicht ausgedrückt werden. Die verschiedenen Höhen der Neuronen kommen einerseits durch die Rückberechnung des Klassentrennwertes an den Neuronen der dritten Schicht, andererseits durch die Gewichtung zwischen der zweiten und dritten Schicht zustande. Die Neuronen der dritten Schicht fassen damit mehrere komplexe Aufteilungen des Eingaberaumes zusammen, z. B. nicht zusammenhängende Einzelgebiete. Durch eine dritte Schicht ist es möglich, kompliziertere Klassentrennlinien mit wenigen Neuronen der ersten verdeckten Schicht zu realisieren. Es ist ebenfalls einfacher, ein Netz mit zwei verdeckten Schichten zu konstruieren, da die Neuronen der zweiten Schicht so angelegt werden können, daß sie eine bekannte Basisfunktion annähern (z. B. Wavelets oder Rechtecke), aus der sich die gesuchte Netzfunktion leicht analytisch zusammensetzen läßt.

4.1.6.5 Neuronen der vierten und weiteren verdeckten Schichten
Netze mit dieser Topologie werden selten eingesetzt. Deshalb werden sie in dieser Arbeit nicht weiter untersucht. Prinzipiell aber lassen sich solche Netze genauso wie Netze mit zwei verdeckten Schichten interpretieren.

4.1.7 Der Algorithmus
Im vorherigen Abschnitt wurde das Prinzip der Visualisierung erläutert. Jetzt wird genauer auf den Algorithmus eingegangen. Damit die Beschreibung eindeutig und verständlich ist, werden die verwendeten Begriffe zuerst beschrieben. Für das weitere Verständnis des Algorithmus ist es auch wichtig, sich die verschiedenen Abkürzungen zu merken. Die Grundbausteine der Visualisierung sind: die Höhenlinie, der Schnittpunkt, das Segment und der Polygonzug. Für eine einheitliche Berechnung sorgen spezielle Begrenzungslinien.

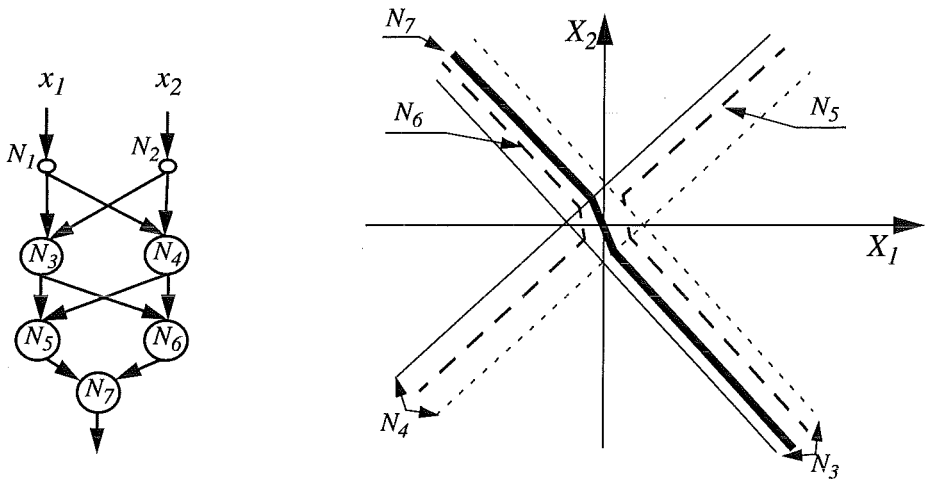


Abbildung 4.17: *Darstellung eines Ausgabeneurons eines Netzes mit zwei verdeckten Schichten bzw. eines Neurons der dritten verdeckten Schicht. In der Visualisierung sind auch Neuronen der ersten und zweiten verdeckten Schicht dargestellt.*

4.1.7.1 Höhenlinie

Wie schon in den bisherigen Erläuterungen dargelegt wurde, ist eine Höhenlinie ein wichtiger Baustein in der GINN. Es werden fast ausschließlich die charakteristischen Höhenlinien des Funktionsgraphen bzw. Hilfslinien dargestellt und in die Visualisierung eingetragen. Eine Höhenlinie kann entweder die Ausgabe eines bestimmten Neurons oder des gesamten Netzes repräsentieren. Eine gerade Linie wird immer in der Normalform:

$$L_h^{(n)} : A_h^{(n)} x + B_h^{(n)} y + C_h^{(n)} = 0 \qquad (4.6)$$

angegeben, wobei A, B und C Koeffizienten sind, der Index n die Neuronennummer und h die Höhe der Linie. Die meisten Höhenlinien befinden sich an der Grenze von zwei Bereichen (z. B. die *0*- oder die *1*-Linie), oder sie liegen im Sicherheitsstreifen (Klassentrennlinie). Die *0*-Linie wird in der Darstellung immer gestrichelt, die *1*-Linie immer durchgehend und die Klassentrennlinie immer fett dargestellt (s. Abbildung 4.18). Andere Höhenlinien werden als dünne Linien angezeigt. Spezielle Linien, die den Definitionsbereich einschränken werden als dünne gestrichelte Linien gezeichnet. Sie haben keine Höhe und keine Bedeutung bei der Interpretation und dienen lediglich der Effizienzsteigerung und Vereinheitlichung des Algorithmus.

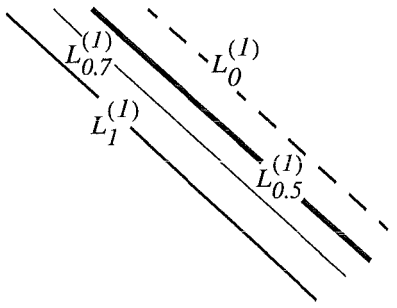


Abbildung 4.18: *Darstellung des Neurons N_1. Dargestellt werden die 0-, 0.5-, 0.7- und 1-Höhenlinie. Die 0.5 Linie ist die Klassentrennlinie.*

In einer graphischen Darstellung schneiden sich die Linien in mehreren Schnittpunkten. Die Schnittpunkte sind wesentlich für die Berechnung der geometrischen Interpretation. Wichtig dabei ist zu wissen, welche Linien an den Schnittpunkten beteiligt sind.

4.1.7.2 Der Schnittpunkt

Ein Schnittpunkt zwischen der Linie $L_i^{(m)}$ und $L_j^{(n)}$ wird mit $P_{L_j^{(n)} L_i^{(m)}}$ bzw. $P_{L_i^{(m)} L_j^{(n)}}$ notiert (d. h. die Indizes geben die sich schneidenden Linien an; die Reihenfolge der Indizes kann beliebig sein). Zum Teil werden in der Arbeit die komplizierten Indizes für Schnittpunkte durch eine einfache Notation P_1, P_2, P_3, ... ersetzt, wenn klar ist, welche Linien sich im Schnittpunkt schneiden. Die Schnittpunkte teilen die Linien in Segmente auf.

4.1.7.3 Das Segment

Ein Segment wird durch zwei Schnittpunkte definiert. Es liegt immer auf einer Höhenlinie. Ein Segment mit den Endpunkten $P_{L_j^{(n)} L_i^{(m)}}$ und $P_{L_i^{(m)} L_l^{(k)}}$ wird mit $S_{L_j^{(n)} L_i^{(m)} L_l^{(k)}}$ bezeichnet. Die Reihenfolge der Indizes ist so gewählt, daß der mittlere Index die Linie angibt, auf welchem das Segment liegt. Der erste Index steht für die Linie, welche das Segment am Ende mit der kleineren *Y*-Koordinate schneidet, und der dritte gibt die Linie am Ende mit der größeren *Y*-Koordinate an (s. Abbildung 4.19). Zwecks einer Vereinfachung können die komplizierten Indizes durch Buchstaben ersetzt werden, z. B.: S_A, S_B, S_C usw., bzw. durch Indizes der Endpunkte: S_{ij}, wenn die Endpunkte mit P_i und P_j notiert werden. Sich berührende Segmente können hintereinander angereiht werden und Polygonzüge bilden. Diese Polygonzüge dienen der Interpretation direkt als Visualisierungselemente.

4.1.7.4 Der Polygonzug

Polygonzüge werden, ähnlich wie Segmente, durch ihre Ecken eindeutig beschrieben, z. B. ein Polygonzug, der die Eckpunkte $P_{L_I^{(A)}L_{II}^{(B)}}$, $P_{L_{II}^{(B)}L_{III}^{(C)}}$, $P_{L_{III}^{(C)}L_{IV}^{(D)}}$, $P_{L_{IV}^{(D)}L_V^{(E)}}$, $P_{L_V^{(E)}L_{VI}^{(F)}}$, $P_{L_{VI}^{(F)}L_I^{(A)}}$ und $P_{L_I^{(G)}L_{II}^{(A)}}$ hat, wird als $Y_{L_I^{(A)}L_{II}^{(B)}L_{III}^{(C)}L_{IV}^{(D)}L_V^{(E)}L_{VI}^{(F)}}$ notiert. Die Indizes geben also die Linien an, aus welchen der Polygonzug besteht. Wenn es klar oder unwesentlich ist, aus welchen Linien ein Polygonzug besteht, dann kann ein Polygonzug mit Y_I, Y_{II}, Y_{III} usw., bzw. mit $Y_{1,2,3,\ldots,k}$, bezeichnet werden (falls die beteiligten Eckpunkte P_1, P_2, P_3, ... , P_k sind).

Wegen des eingeschränkten Visualisierungsbereiches (s. Kapitel 4.1.7.5) kommen im Algorithmus nur geschlossenen Polygonzüge vor.

Beispiel 4.9. In Abbildung 4.19 sind drei Höhenlinien $L_0^{(1)}$ (Neuron N_1 Höhe *0*), $L_0^{(2)}$ (Neuron N_2 Höhe *0*) und $L_1^{(3)}$ (Neuron N_3 Höhe *1*) eingetragen. Diese Linien schneiden sich in drei Punkten: $P_{L_0^{(1)}L_0^{(2)}}$, $P_{L_0^{(2)}L_1^{(3)}}$ und $P_{L_0^{(1)}L_1^{(3)}}$. Sie definieren drei Segmente, und zwar: $S_{L_1^{(3)}L_0^{(1)}L_0^{(2)}}$, $S_{L_0^{(1)}L_1^{(3)}L_0^{(2)}}$ sowie $S_{L_1^{(3)}L_0^{(2)}L_0^{(1)}}$. Der einzige geschlossene Polygonzug ist somit $Y_{L_1L_2L_3}$.

■

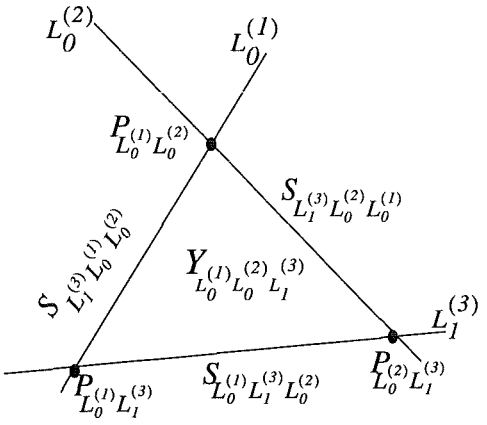


Abbildung 4.19: *Verwendete Bezeichnungen für Punkt, Segment und Polygonzug.*

Wenn man sich die Polygonzüge im dreidimensionalen Raum vorstellt, dann hat jeder Polygonzug eine bestimmte räumliche Lage, d. h. seine Fläche verläuft unter einem bestimmten Winkel zur *XY*-Ebene. Ausschlaggebend für die Lage eines Polygonzuges ist die Höhe seiner Eckpunkte. Die Unterschiede in den Höhen bestimmen also, wie ein Polygonzug in der Darstellungsebene angeordnet ist.

4.1.7.5 Einschränkung des Visualisierungsbereiches

Bevor mit dem ersten Berechnungsschritt begonnen wird, wird der Visualisierungsbereich eingeschränkt. Er wird gleich dem Definitionsbereich gesetzt. Die Einschränkung dient zum einen der Effizienzsteigerung und zum anderen einer Vereinheitlichung des Algorithmus. Der Algorithmus wird vereinheitlicht, weil es jetzt keine Sonderfälle mehr gibt, die durch offene Polygonzüge entstehen. Die Effizienzsteigerung ergibt sich daraus, daß jetzt nicht mehr die Darstellung für den ganzen $\boldsymbol{R^2}$ berechnet werden muß, sondern nur für den Definitionsbereich. Technisch gesehen wird die Beschränkung durch spezielle Rahmenlinien vollzogen, welche mit R_i bezeichnet werden (s. Abbildung 4.20). Diese Linien haben keine Bedeutung für die Interpretation.

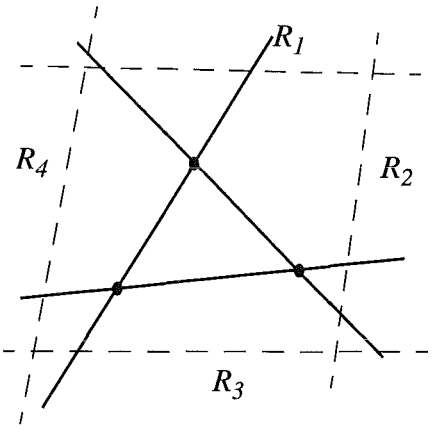


Abbildung 4.20: *Einzeichnung der Rahmenlinien.*

Beispiel 4.10. Die Darstellung aus Abbildung 4.19 wurde mit vier Rahmenlinien R_1, R_2, R_3 und R_4 beschränkt. Das Ergebnis zeigt Abbildung 4.20. Damit beinhaltet der Visualisierungsbereich keine offenen Polygonzüge mehr.
■

4.1.7.6 Die Berechnung

Die Berechnungen der Darstellung erfolgen Schicht für Schicht von der Eingabe- bis zu der Ausgabeschicht. Dabei werden die Schichten der Reihe nach visualisiert. Das am Bildschirm dargestellte Bild beinhaltet dann, je nach der Einstellung der Visualisierungsoptionen, nur die Visualisierung bestimmter Schichten. Für die Berechnung wird aber eine temporäre Darstellung aller Schichten bis zu der dargestellten Schicht benötigt.

Für eine beliebige Schicht sieht der Algorithmus vor:

- Einzeichnung der *0-/1*-Linien der (verdeckten) Neuronen
- Bestimmung der Schnittpunkte zwischen den *0-/1*-Linien
- Aufteilung in Segmente
- Zusammenfassung der Segmente zu Polygonzügen
- Berechnung der räumlichen Polygonzuglage
- Einzeichnung der Klassentrennlinie der visualisierten Schicht.

In den folgenden Abschnitten werden diese Schritte ausführlich erläutert.

4.1.7.7 Einzeichnung der 0-/1-Linien der verdeckten Neuronen

Die verdeckten Neuronen sind durch ihre *0*- und *1*-Linien visualisiert. Der Linienverlauf läßt sich aus den Neuronenparametern berechnen. Jedes verdeckte Neuron hat eine logistische Transferfunktion der Form:

$$\sigma(x, y) = \frac{1}{1 + e^{-w_1 x - w_2 y - Bias}}. \qquad (4.7)$$

Sie wird durch eine lineare Approximation angenähert. Die Geradengleichungen für die *0*-Linie lautet: $L_0^{(n)}: A_0^{(n)} x + B_0^{(n)} y + C_0^{(n)} = 0$, und die für die *1*-Linie $L_1^{(n)}: A_1^{(n)} x + B_1^{(n)} y + C_1^{(n)} = 0$.

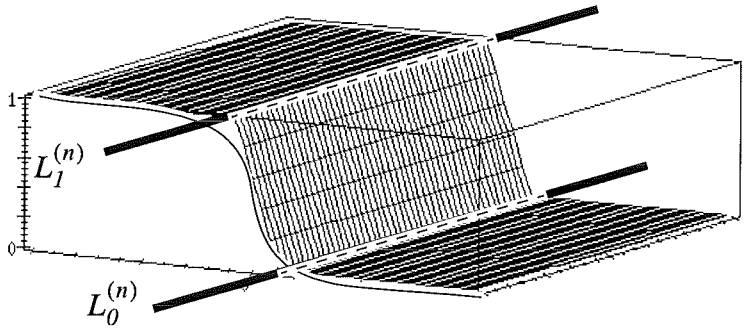


Abbildung 4.21: *Verlauf der 0- und 1-Linie in der Neuronapproximation. Die 1-Linie liegt an der Grenze der 1-Halbebene mit dem Sicherheitsstreifen, und die 0-Linie liegt an der Grenze zwischen der 0-Halbebene und dem Sicherheitsstreifen. Der lineare Übergang hat die Steigung der Sigmoidfunktion im Wendepunkt.*

Bestimmt werden müssen noch die Koeffizienten der Geradengleichungen. Dazu wird die Steigungsrichtung des Sicherheitsstreifens benötigt. Der Sicherheitsstreifen liegt zwischen den gesuchten Geraden. Die gesuchten Geraden befinden sich an den Grenzen der Halbebenen mit dem linearen Übergang. Der lineare Übergang ist so angelegt, daß er die Steigung der logistischen Funktion im Wendepunkt hat. Die Geradengleichung des Wendepunktes ergibt sich aus der Auflösung von:

$$\frac{\partial^2}{\partial(x,\,y)}\sigma(x,\,y) = \frac{w_1 w_2 e^{-w_1 x - w_2 y - Bias}\left(e^{-w_1 x - w_2 y - Bias} - 1\right)}{\left(1 + e^{-w_1 x - w_2 y - Bias}\right)^3} = 0\,. \tag{4.8}$$

zu:

$$w_1 x + w_2 y + Bias = 0\,. \tag{4.9}$$

Die Steigung im Wendepunkt ist damit gleich:

$$\begin{aligned} \frac{\partial}{\partial x}\sigma(x,\,y)\big|_{w_1 x + w_2 y + Bias = 0} &= \frac{1}{4}w_1 \\ \frac{\partial}{\partial y}\sigma(x,\,y)\big|_{w_1 x + w_2 y + Bias = 0} &= \frac{1}{4}w_2 \end{aligned} \tag{4.10}$$

und die Gleichung des linearen Übergangs lautet dann:

$$s(x,\,y) = \frac{1}{4}w_1 x + \frac{1}{4}w_2 y + \frac{1}{4}Bias + \frac{1}{2}. \tag{4.11}$$

Die Begrenzungslinien des Sicherheitsstreifens sind die gesuchten *1*- bzw. *0*-Linien. Sie sind der Schnitt mit den waagerechten *1*- und *0*-Halbebenen:

$$\begin{aligned} L_0^{(n)}\colon\ & \frac{1}{2}w_1^{(n)} x + \frac{1}{2}w_2^{(n)} y + \frac{1}{2}Bias^{(n)} + 1 = 0 \\ L_1^{(n)}\colon\ & \frac{1}{2}w_1^{(n)} x + \frac{1}{2}w_2^{(n)} y + \frac{1}{2}Bias^{(n)} - 1 = 0 \end{aligned} \tag{4.12}$$

Beide Höhenlinien begrenzen also den Sicherheitsstreifen und sind parallel so versetzt, daß deren Abstand umgekehrt proportional zu der Steigung des linearen Überganges ist. Die Breite des Sicherheitsstreifens indiziert somit die Steigung des linearen (und somit auch des nichtlinearen) Überganges. Ein steiler Übergang, d. h. ein schmaler Sicher-

heitsstreifen, bedeutet eine "sichere" Trennung der Klassen; ein breiter Sicherheitsstreifen bedeutet hingegen eine "unsichere" Trennung der Klassen.

Die Koeffizienten $A_0^{(n)}, B_0^{(n)}, C_0^{(n)}, A_1^{(n)}, B_1^{(n)}$ und $C_1^{(n)}$ der Geradengleichungen werden direkt aus den Parametern eines verdeckten Neurons n (mit den Gewichten $w_1^{(n)}, w_2^{(n)}$ und dem Bias $Bias^{(n)}$) bestimmt:

$$L_0^{(n)}: \underbrace{\tfrac{1}{2}w_1^{(n)}}_{A_0^{(n)}} x + \underbrace{\tfrac{1}{2}w_2^{(n)}}_{B_0^{(n)}} y + \underbrace{\tfrac{1}{2}\cdot Bias^{(n)} + 1}_{C_0^{(n)}} = 0 \tag{4.13}$$

$$L_1^{(n)}: \underbrace{\tfrac{1}{2}w_1^{(n)}}_{A_1^{(n)}} x + \underbrace{\tfrac{1}{2}w_2^{(n)}}_{B_1^{(n)}} y + \underbrace{\tfrac{1}{2}\cdot Bias^{(n)} - 1}_{C_1^{(n)}} = 0\,. \tag{4.14}$$

Während der Berechnung müssen lediglich *6* Multiplikationen und *2* Additionen durchgeführt werden. Daraus ergibt sich die Komplexität zu:

$$(6+2)O(1) \in O(1). \tag{4.15}$$

4.1.7.8 Bestimmung der Schnittpunkte von 0-/1-Linien innerhalb des Visualisierungsbereiches

Die *0-/1*-Linien verschiedener verdeckten Neuronen verlaufen selten parallel, sie schneiden sich und bilden somit Schnittpunkte. Dieser Berechnungsschritt bestimmt diese Schnittpunkte und trägt sie in eine sog. *Schnittpunktliste* ein. Der Algorithmus macht keinen Unterschied zwischen Linien verschiedener Höhen (*0*- und *1*-Linien) bzw. verschiedenen Linientypen (Höhen- und Rahmenlinien). Ein Schnittpunkt zwischen zwei Linien $L_h^{(n)}$ und $L_l^{(n)}$ ergibt sich aus:

$$\begin{cases} L_h^{(n)} : A_h^{(n)} x + B_h^{(n)} y + C_h^{(n)} = 0 \\ L_l^{(m)} : A_l^{(m)} x + B_l^{(m)} y + C_l^{(m)} = 0 \end{cases} \tag{4.16}$$

zu:

$$\begin{cases} x = \dfrac{B_h^{(n)} C_l^{(m)} - C_h^{(n)} B_l^{(m)}}{A_h^{(n)} B_l^{(m)} - A_l^{(m)} B_h^{(n)}} \\ y = -\dfrac{A_h^{(n)} C_l^{(m)} - C_h^{(n)} A_l^{(m)}}{A_h^{(n)} B_l^{(m)} - A_l^{(m)} B_h^{(n)}}. \end{cases} \tag{4.17}$$

Eintragung in die Schnittpunktliste. Für weitere Berechnungen werden nur Punkte benötigt, welche sich innerhalb des Visualisierungsbereiches befinden. Deswegen wird durch einen Test festgestellt, ob ein neuer Punkt in die Schnittpunktliste eingetragen werden soll. Schnittpunkte außerhalb des Visualisierungsbereich bleiben bei weiteren

Berechnungen unberücksichtigt und brauchen nicht in die Schnittpunktliste eingetragen werden. Ein weiteres Problem ergibt sich daraus, daß der Schnittpunktalgorithmus in einem Schritt nur den Schnittpunkt zweier Linien finden kann. Mehrfache Schnittpunkte müssen deswegen erkannt und zu einem einzigen zusammengefaßt werden.

Zusammenfassung von Mehrfachschnittpunkten. Weil der Algorithmus nur Schnittpunkte zwischen zwei Linien findet, faßt ein zusätzlicher Vergleich Mehrfachschnittpunkte, d. h. Schnittpunkte mit identischen Koordinaten, zusammen.

Algorithmus 4.1

Eingabe:
$L=\{L_{j_1}^{(i_1)}, L_{j_2}^{(i_2)}, ..., L_{j_n}^{(i_n)}\}$, eine Menge von Linien.

Ausgabe:
SP, eine Menge von Schnittpunkten.

Methode:

```
// Berechne die Schnittpunkte
FOR k = 1 TO n DO
    FOR l = k TO n DO
        // Berechne den Schnittpunkt zwischen dem Linienpaar
        sp=berechneSchnittpunkt(L_{j_k}^{(i_k)}, L_{j_l}^{(i_l)}) // Nach Formel (4.17)
        // Trage den Schnittpunkt in die sortierte Schnittpunktliste ein
        // Prüfe vorher, ob er existiert und im Visualisierungsbereich liegt
        IF sp.existiert() AND sp.imDefBereich() THEN
            SP.sortAppend(sp)
        ENDIF;
    END;
END;

// Fasse mehrfache Schnittpunkte zusammen
FOR s=0 TO SP.size() DO
    FOR t=s TO SP.size() DO

        // Wenn zwei Schnittpunkte gleich sind, dann fasse sie zusammen
        IF SP.elementAt(t) == SP.elementAt(s) THEN
            SP.merge(SP.elementAt(s), SP.elementAt(t));
        ENDIF;
    END;
END;
```

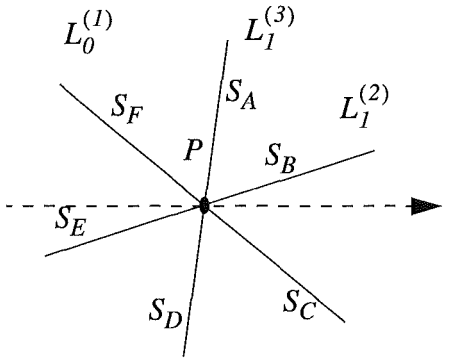
Komplexität:
Die Komplexität der Schnittpunktberechnung ist von der Ordnung $O(n^2)$:

$$\underbrace{O(n)(O(n)+O(1)+O(n))}_{1.\ Doppelschleife} + \underbrace{O(n)(O(n)+O(1))}_{2.\ Doppelschleife} \in O(n^2). \qquad (4.18)$$

Wie schon zu Beginn geschildert wurde, teilen Schnittpunkte die Linien in Segmente auf. Der nächste Berechnungsschritt bestimmt diese Segmente.

4.1.7.9 Bestimmung der sich aus den Schnittpunkten ergebenden Segmente

Die in Kapitel 4.1.7.8 berechneten Schnittpunkte teilen die Höhenlinien in Segmente auf. Um die Segmente zu finden, wird jede Linie von ihrem Ende mit der kleineren *Y*-Koordinate zum Ende mit der größeren *Y*-Koordinate, d. h. aufwärts, durchgegangen und dabei segmentiert. Waagerechte Linien werden definitionsgemäß von links nach rechts durchgegangen. Alle Segmente, welche sich in einem Schnittpunkt berühren, werden nach dem Winkel zur waagerechten Achse aufsteigend sortiert und in eine dem Schnittpunkt zugeordnete Winkelliste *WL* eingetragen.



$WL(P) = \{(S_B, 30^o), (S_A, 80^o), (S_F, 135^o), (S_E, 210^o), (S_D, 260^o), (S_C, 315^o)\}$

Abbildung 4.22: *Die Segmente, welche sich in einem Schnittpunkt treffen, werden in die Winkelliste WL(P) in der Reihenfolge der Winkel mit der waagerechten Achse eingetragen.*

Beispiel 4.11. Abbildung 4.22 zeigt einen Schnittpunkt, in dem sich sechs Segmente S_A, S_B, S_C, S_D, S_E und S_F berühren. In die Winkelliste *WL(P)* werden sie (sortiert nach dem Winkel zur waagerechten Achse) eingetragen. Den kleinsten Winkel hat das Segment S_B, den nächst größeren S_A, dann S_F usw. Die Listenelemente enthalten zwei Informationen, nämlich die Segmentidentifikation S_i und den Winkel α zur waagerechten Achse. Diese Informationen werden in weiteren Algorithmusschritten benötigt.
■

Algorithmus 4.2

Eingabe:

$L=\{L_{j_1}^{(i_1)}, L_{j_2}^{(i_2)}, ..., L_{j_n}^{(i_n)}\}$ (eine Menge von Linien) und $SP=\{P_1, P_2, ..., P_m\}$ (eine Menge von Schnittpunkten).

Ausgabe:

S (eine Menge von Segmenten) und $WL(P_i)$ (eine Winkelliste).

Methode:

// Finde für jede Linie alle Schnittpunkte, welche zu ihr gehören

// Gehe alle Linien durch
FOR $l=1$ **TO** n **DO**

// Gehe alle Schnittpunkte durch, welche auf der Linie $L_{j_l}^{(i_l)}$ liegen

FOR ALL $P_k \in L_{j_l}^{(i_l)}$ **DO**

// Füge Schnittpunkt P_k in eine nach Y-Werten sortierte Liste ein

$S_{tmp}.sortInsert(P_k);$

END;

// Bilde alle Segmente, die sich aus den Schnittpunkten ergeben

```
        FOR i=1 TO S_tmp.size() DO

            // Hole zwei aufeinanderfolgende Schnittpunkte...
            sp_1 = S_tmp.elementAt(i-1);
            sp_2 = S_tmp.elementAt(i);

            // ... und fasse sie zu einem Segment zusammen
            S.append(sp_1,sp_2);
            WL.sortInsert(angle(S.lastElement()));
        END;
END;
```

Komplexität:

Die Komplexität der Segmentbildung beträgt $O(n^3)$, denn:

$$\underbrace{O(n)}_{\substack{\text{Alle Linien}\\ \text{durchgehen}}} \underbrace{O(n-1)}_{\substack{\text{Alle Punkte}\\ \text{einer Linie}\\ \text{durchgehen}}} \underbrace{O(n-1)}_{\substack{\text{Sortiertes Einfügen}\\ \text{der Punkte}}} + \underbrace{O(n)(O(1)+O(1)+O(n))}_{\text{Segmentbildung}} \in O(n^3).$$

Nachdem die Segmente gefunden wurden, werden sie zu Polygonzügen zusammengefaßt, die dann die Darstellungselemente bilden.

4.1.7.10 Bestimmung der Polygonzüge

Ein Polygonzug – so wie er in dieser Arbeit definiert ist – ist eine endliche, geschlossene Folge von Segmenten, die keine Zyklen enthält. Ein Polygonzug kann durch die Angabe der Segmente oder durch die Angabe der Eckpunkte eindeutig beschrieben werden. Es gibt verschiedene Algorithmen, die es erlauben, einen Polygonzug zu finden.
Die einfachste Methode, alle Polygonzüge zu finden, läßt sich aus der Graphentheorie übernehmen.

Beispiel 4.12. In Abbildung 4.23 ist eine Aufteilung des Eingaberaumes durch zwei verdeckte Neuronen dargestellt (linkes Bild) und der daraus resultierende Graph (rechtes Bild). Um alle Polygonzüge zu finden, muß die Adjazenzmatrix des Graphen n mal mit sich selbst multipliziert werden. Bei der i-ten Multiplikation werden alle Pfade mit der Länge i errechnet. Übertragen auf die Polygonzugbildung bedeutet es, daß i aneinander gereihte Segmente gefunden werden. Allerdings ergeben sich wegen des nicht gerichteten Graphen sehr viele Pfade, die auf demselben Wege hin und her laufen. Deshalb findet dieser Algorithmus viele überflüssige Polygonzüge. Damit diese erkannt werden können, ist eine zusätzliche Buchführung erforderlich. Der Aufwand für diese zusätzliche Buchführung ist der Grund, weshalb sich dieser Algorithmus für die GINN nicht eignet.

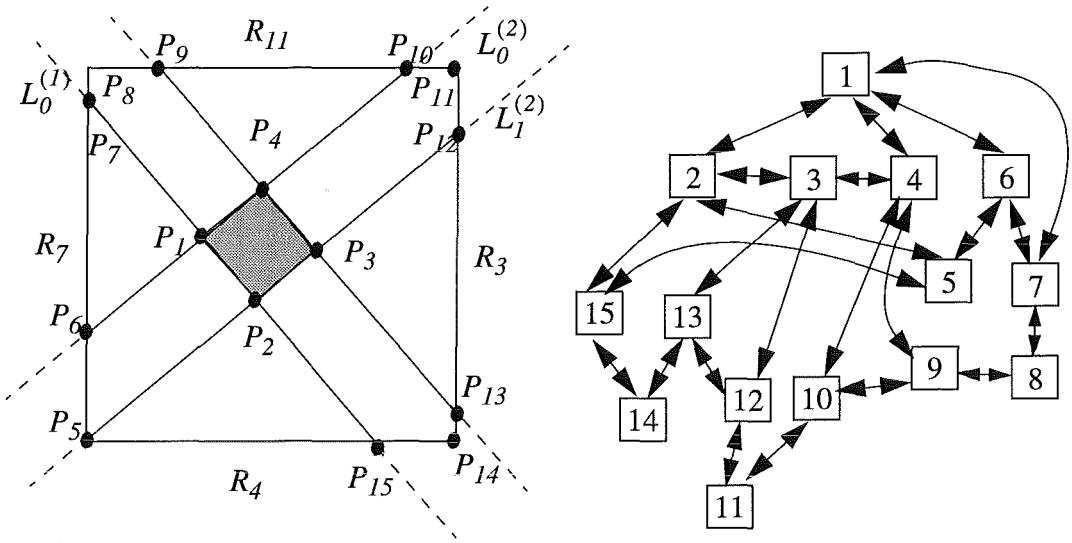


Abbildung 4.23: *Eine Aufteilung des Eingaberaumes durch zwei Neuronen (linkes Bild) und die Darstellung als Graph (rechtes Bild). In dieser Darstellung sind die Punkte die Knoten und die Segmente die Kanten des Graphen. Polygonzüge entsprechen Zyklen im Graph mit mehr als zwei Knoten. Der schraffierte Polygonzug im linken Bild entspricht dem Zyklus 1-2-3-4-1.*

■

Die graphentheoretische Methode findet alle möglichen Polygonzüge, die sich aus der Lage der Kanten und Ecken ergeben, d. h. auch diese, welche für die Visualisierung nicht gebraucht werden. Aus Effizienzgründen ist sie also in ihrer ursprünglichen Form für die geometrische Interpretation nicht geeignet. Eine Abwandlung, die eine Suche in einem sog. *LP-transformierten* Raum durchführt, ist besser geeignet.

Die Ineffizienz der graphentheoretischen Methode (in der Anwendung in der GINN) ist durch die vollständige Suche nach allen möglichen Polygonzügen bedingt. Diese Ineffizienz kann durch ein intelligentes Traversieren des Graphen behoben werden. Dabei muß in jedem Knoten entscheiden werden, wohin die nächste Kante gelegt wird. Diese Entscheidung ist eng mit der Beschaffenheit eines Polygonzuges verbunden, welcher in der Visualisierung gebraucht wird. Ein derartiger Polygonzug darf keine Kantenverbindung in seinem Inneren haben. Diese Bedingung läßt sich zuerst sehr schlecht in einen Algorithmus umsetzen, da das Innere eines Polygonzuges erst am Ende der Suche bekannt ist, d. h. erst dann, wenn der ganze Polygonzug gefunden wurde. In der Literatur findet man Algorithmen, welche die sog. *Basiszyklen* finden [Kun89]. Sie wurden in dieser Arbeit erweitert und den Bedürfnissen angepaßt. Eine elegante Lösung dieses Problems kann im sog. *LP-transformierten Raum* angegeben werden. In diesem Raum wird eine Linie durch einen L-Punkt und ein Punkt durch eine P-Linie dargestellt. Die LP-Transformationsvorschrift lautet:

$$L_i^{(n)}: \quad A_i^{(n)} x + B_i^{(n)} y + C_i^{(n)} \longrightarrow \left(-\frac{A_i^{(n)}}{B_i^{(n)}}, -\frac{C_i^{(n)}}{B_i^{(n)}} \right) \tag{4.19}$$

$$P_{L_{h_1}^{(n_1)} L_{h_2}^{(n_2)}} : (a, b) \longrightarrow v = -a \cdot u + b$$

Beispiel 4.13. Ein Beispiel für eine LP-Transformation zeigt Abbildung 4.24. Die vier Linien werden auf vier L-Punkte abgebildet; die vier Punkte hingegen auf vier P-Linien.

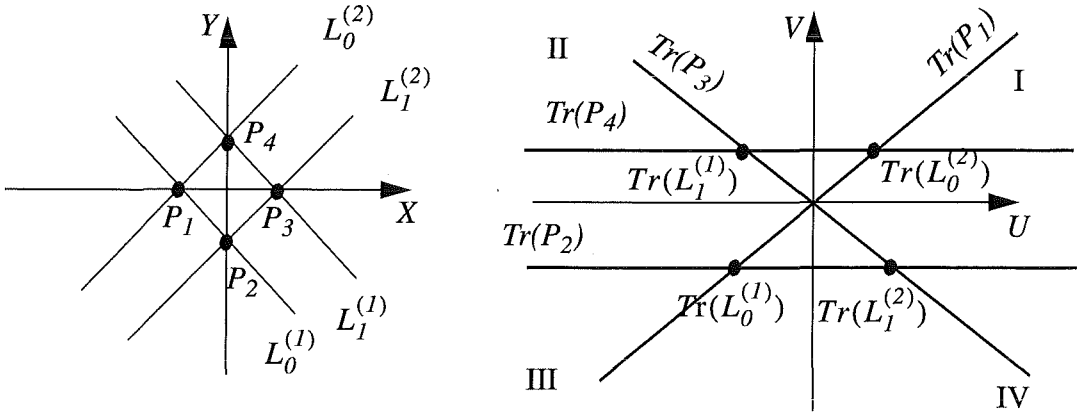


Abbildung 4.24: *Beispiel für einer LP-Transformation. Das linke Bild zeigt die Linien und Punkte im Originalraum, das rechte Bild dieselben Linien und Punkte nach der LP-Transformation. Die gesuchten Polygonzüge entsprechen im transformierten Raum Polygonzügen, deren Ecken die L-Punkte sind und keine L-Punkte in seinem Inneren beinhalten.*
■

Das Traversieren beginnt mit einem beliebigen L-Punkt. Der Algorithmus tastet sich entlang einer P-Linie zum nächsten L-Punkt weiter. Dabei wird der nächste L-Punkt so gewählt, daß in dem entstehenden Polygonzug kein anderer L-Punkt eingeschlossen wird. In jedem L-Punkt wird als nächste diejenige P-Linie gewählt, welche sich nach folgenden Regeln ergibt:

1. Liegt der L-Punkt im I. oder IV. Quadranten des kartesischen Koordinatensystems, dann drehe die Richtung gegen den Uhrzeigersinn bis zur nächsten P-Linie und bewege dich auf der P-Linie zum nächsten L-Punkt.
2. Liegt der L-Punkt im II. oder III. Quadranten, dann drehe die Richtung des Traversierens im Uhrzeigersinn bis zur nächsten P-Linie und bewege dich auf der P-Linie zum nächsten L-Punkt.

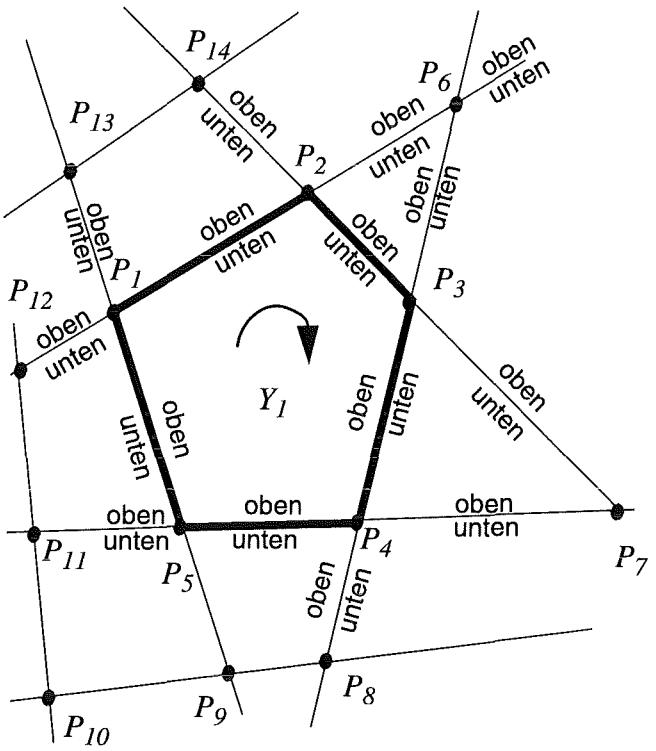
Führt man dieses Traversieren weiter durch, dann gelangt man nach endlich vielen Schritten wieder zum Ausgangspunkt und findet somit einen geschlossenen Polygonzug im Originalraum. Da der Visualisierungsbereich durch spezielle Begrenzungslinien eingerahmt ist, ist jeder im Visualisierungsbereich gefundene Polygonzug geschlossen. Die Polygonzugsuche im transformierten Raum ist wegen der Hin- und Hertransformation relativ rechenintensiv. Deswegen wird der Algorithmus in den Originalraum übertragen und dort durchgeführt.

Im Originalraum geht eine Linie durch zwei Punkte; im transformierten Raum wird ein Punkt durch zwei Linien definiert. Aus diesem Dualismus folgt, daß im transformierten Raum ein Segment durch zwei P-Linien definiert ist, die sich in einem L-Punkte schneiden. Der L-Punkt ist das Bild einer Geraden, deren Bestandteil ein Segment ist, welches durch die zwei P-Linien definiert ist. Daraus ergibt sich, daß alle für die Visualisierung gesuchten Polygonzüge im transformierten Raum auch Polygonzüge (sog. *LP-Polygonzüge*) sind. Die Bedeutung der Ecken und Kanten vertauscht sich jedoch (dem Dualismus entsprechend). Die gesuchten LP-Polygonzüge dürfen keine L-Punkte in ihrem Inneren einschließen. Obgleich die theoretische Überlegung im transformierten Raum geschieht, soll der Algorithmus (der Effizienz wegen) im Originalraum arbeiten.

Für die Polygonsuche muß also ein effizienter Graphentraversierungsalgorithmus entwickelt werden, der nur die benötigten Polygonzüge findet. Sowohl im Original als auch im transformierten Raum werden die Segmente und Schnittpunkte wie gewohnt als Kanten und Knoten eines Graphen dargestellt. Im transformierten Raum entsprechen die P-Linien den Kanten und die L-Punkte den Knoten des Graphen. Das Traversieren soll von einer Kante ausgehend durch den Graphen wandern und dabei den gesuchten LP-Polygonzug markieren (als zurückgelegten Weg). Eine modifizierte Version des Trace-Algorithmus [Dohmen91] erledigt diese Aufgabe. Während des Traversierens tastet er sich im bzw. entgegen dem Uhrzeigersinn an den P-Linien entlang. In jedem L-Punkt wechselt er zu einer neuen P-Linie. Das Traversieren wird solange fortgesetzt, bis der Ausgangspunkt der Wanderung erreicht wird. Somit ist einer der Polygonzüge gefunden worden. Das Traversieren wird für alle möglichen P-Linien wiederholt. So werden beide Polygonzüge gefunden, an denen eine P-Linie beteiligt ist.

Eine Berechnung im Originalraum ist möglich, wenn die dualen Schritte durchgeführt werden. Im Originalraum vertauschen sich jedoch die Rollen von Schnittpunkten und Segmenten: die Schnittpunkte bilden jetzt die Knoten und die Segmente die Kanten. Der Traversierungsalgorithmus geht von einem Segment aus und versucht, den Polygonzug im Uhrzeigersinn durchzuwandern. Jedes Segment wird in ein oberes und ein unteres aufgeteilt. Jedes dieser Teile gilt als eine separate Kante.

Als erstes wird ein Segment ausgewählt und das Traversieren mit seinem unteren Teil begonnen. Es wird der Winkel des Segments und das Segmentende bestimmt, welches die Wanderung im Uhrzeigersinn fortsetzt. Der Algorithmus tastet sich zu diesem Ende hin und sucht nach dem nächsten Segment. Das nächste Segment ist dasjenige, welches den nächstgrößten Winkel (mit der waagerechten Richtung) im Endpunkt bildet. Die Winkelberechnungen erfolgen modulo 360^o. Dieses Segment wird zum nächsten Segment im Polygonzug addiert. Zu bestimmen bleibt nun noch, ob der obere oder der untere Teil des Segments genommen wird. Dieses wird anhand des Winkels entschieden. Liegt dieser Winkel im Bereich 90^o-270^o, dann wird der obere Teil des Segments genommen, ansonsten der untere. Jedes durchgegangene Segment wird auch entsprechend markiert, damit es nicht doppelt traversiert wird. Diese Prozedur wird solange wiederholt, bis das Ausgangssegment wieder erreicht wird. Dann wird dasselbe mit dem oberen Teil des Ausgangssegments gemacht. Nachdem alle Segmente bearbeitet werde, sind alle Polygonzüge gefunden.



Winkellisten:

$WL(P_1)=\{(S_{1,2}, 30), (S_{1,13}, 110), (S_{1,12},210), (S_{1,5}, 290)\}$

$WL(P_2)=\{(S_{2,6}, 30), (S_{2,14}, 120), (S_{1,2},210), (S_{2,3}, 300)\}$

$WL(P_3)=\{(S_{3,6}, 60), (S_{3,2}, 120), (S_{3,4},240), (S_{3,7}, 300)\}$

$WL(P_4)=\{(S_{4,3}, 60), (S_{4,5}, 170), (S_{4,8},240), (S_{4,7}, 350)\}$

$WL(P_5)=\{(S_{5,1}, 110), (S_{5,11}, 170), (S_{5,9}, 290), (S_{5,4}, 350)\}$

Abbildung 4.25: *Die Aufteilung des Eingaberaumes durch 8 Höhenlinien (links) und die Winkellisten der Schnittpunkte (rechts). Während des Traversieren des Graphen geht der Algorithmus den gesuchten Polygonzug im Uhrzeigersinn durch. Dabei tastet er sich an den Segmenten entlang, bis der Ausgangspunkt erreicht ist.*

Beispiel 4.14. Im Graphen aus Abbildung 4.25 soll der Polygonzug Y_1 gefunden werden. Der Algorithmus wählt als erstes ein Anfangssegment, z. B. $S_{1,2}$ (es ist das Segment zwischen den Punkten P_1 und P_2). Als aktuelles Segmentteil wird sein unterer Teil ("unten") genommen. Im ersten Schritt wird der nächste traversierte Punkt bestimmt. Dieser Punkt muß einer von den Endpunkten des Segments sein. Welcher es ist, wird anhand des Winkels und der Steigung der Geraden entschieden, auf welcher das Segments liegt. In diesem Fall ist es der Punkt mit dem größeren y-Wert, d. h. P_2 (weil der Winkel im Bereich von 0^o-90^o liegt und die Steigung positiv ist). Der Algorithmus wandert also zum Punkt P_2. Jetzt muß ein neues aktuelle Segment gefunden werden. Dazu wird in der Winkelliste des Punktes P_2 das Segment $S_{1,2}$ gefunden und das nächste in der Liste als aktuelle Segment genommen, d. h. $S_{2,3}$. Ob der obere ("oben") oder untere ("unten") Teil des Segments genommen wird, entscheidet sich anhand des Winkels. Der Winkel 300^o liegt im Bereich 270^o-90^o(modulo 360^o), deswegen wird der untere Teil ("unten") des Segments genommen. Nächster Punkt ist diesmal das untere Ende des Segments $S_{2,3}$, da seine Steigung negativ ist und der Winkel im Bereich 0^o-90^o liegt (modulo 360^o gerechnet). Der Algorithmus geht somit zum Punkt P_3. Hier wird dieselbe Prozedur durchgeführt wie beim Punkt P_2. Allerdings liegt jetzt der Winkel des neu bestimmten Segments $S_{3,4}$ im Bereich 90^o-270^o. Deswegen wird für weitere Berechnungen sein oberer Teil ("oben") genommen. Als nächstes werden die Punkte P_4, P_5 und P_1 durchgegangen. Im Punkt P_1 angekommen, merkt der Algorithmus, daß ein Zyklus geschlossen wurde und beendet sein Traversieren. Der Polygonzug $Y_{1,2,3,4,5}$ ist somit gefunden.
■

Algorithmus 4.3

Eingabe:

Eine Menge von Schnittpunkten $SP = \{ P_{L_{h_1} L_{h_2}}^{(n_1)(n_2)}, \ldots, P_{L_{h_{k-1}} L_{h_k}}^{(n_{k-1})(n_k)} \}$

Segmenten $S = \{ S_{L_{h_{l_1}} L_{h_{l_2}}}^{(n_{l_1})(n_{l_2})}, \ldots, S_{L_{h_{l_{m-2}}} L_{h_{l_{m-1}}}}^{(n_{l_{m-2}})(n_{l_{m-1}})}, L_{h_{l_m}}^{(n_{l_m})} \}$

und die Winkellisten der Schnittpunkte

$WL = \{ (S_{L_{h_{l_1}} L_{h_{l_2}} L_{h_{l_3}}}^{(n_{l_1})(n_{l_2})(n_{l_3})}, \omega_{L_{h_{l_1}} L_{h_{l_2}} L_{h_{l_3}}}^{(n_{l_1})(n_{l_2})(n_{l_3})}), \ldots, (S_{L_{h_{l_{m-2}}} L_{h_{l_{m-1}}} L_{h_{l_m}}}^{(n_{l_{m-2}})(n_{l_{m-1}})(n_{l_m})}, \omega_{L_{h_{l_{m-2}}} L_{h_{l_{m-1}}} L_{h_{l_m}}}^{(n_{l_{m-2}})(n_{l_{m-1}})(n_{l_m})}) \}$.

Ausgabe:

Eine Liste *Y* mit allen Polygonzügen, die sich aus der Aufteilung des Raumes ergeben.

Methode:

In der Beschreibung werden folgende Bezeichnungen verwendet:

1. Eine Variable für die Speicherung eines Segments hat ein Feld mit dem Namen *teil*. In dem Feld wird eine Markierung für den oberen (*oben*) bzw. unteren (*unten*) Teil eines Segments gespeichert.
2. Auf das *i*-te Element der Winkelliste kann mit *WL.elementAt(i)* zugegriffen werden. Alle Elemente haben ein Feld mit dem Namen *segment* und eins mit dem Namen *winkel*.

```
// Diese Schleife geht alle Segmente durch und ruft für jedes
// noch nicht markierte Segment die Funktion traverse() auf
FOR i = 2 TO m-1 DO
        S_akt = S_{L_{h_{l_{i-1}}} L_{h_{l_i}}}^{(n_{l_{i-1}})(n_{l_i})} L_{h_{l_{i+1}}}^{(n_{l_{i+1}})}
        // Oberes Teil markiert?
        IF NOT istMarkiert(S_akt.teil, oben) THEN
                traverse(S_akt.oben);
        ENDIF;

        // Unteres Teil markiert?
        IF NOT istMarkiert(S_akt.teil, unten) THEN
                traverse(S_akt.unten);
        ENDIF;
END;
```

Die Funktion *traverse*(S_{akt}):

```
// Traversiere so lange, bis ein markiertes Segment erreicht wird

WHILE S_akt.teil ≠ markiert DO
        // Markiere den aktuell traversierten Teil des Segments
        mark(S_akt.teil );

        // Finde das Ende, zu welchem sich der Algorithmus bewegt
        // Bestimme dazu die Steigung und den Winkel des Segments
```

$$steigung = -\frac{A_{h_{l_{i+1}}}^{(n_{l_{i+1}})}}{B_{h_{l_{i+1}}}^{(n_{l_{i+1}})}};$$

$winkel = atanh(steigung)\ MOD\ 180^o;$

// Gehe zu dem gefundenen Ende
IF $steigung \geq 0$ **AND** $winkel \in [0°, 90°]$ **THEN**
    $Pkt = P_{L_{h_{l_i}}}^{(n_{l_i})} L_{h_{l_{i+1}}}^{(n_{l_{i+1}})}$; // oberes Ende
ELSE
    $Pkt = P_{L_{h_{l_{i-1}}}}^{(n_{l_{i-1}})} L_{h_{l_i}}^{(n_{l_i})}$; // unteres Ende
ENDIF;

// Suche in der Winkelliste nach nächstem Segment
FOR $j=1$ **TO** m **DO**
    IF $WL.elementAt(j).segment = S_{akt}$ **THEN**
        $segmentWinkel = WL.elementAt(j+1\ MOD\ k).winkel;$
        $S_{akt} = WL.elementAt(j+1\ MOD\ k).segment;$

        // Wähle oberen bzw. unteren Teil des Segments
        // (je nach Winkel)
        IF $segmentWinkel \in [90°, 270°]$ **THEN**
            $S_{akt}.teil = unten;$
        ELSE
            $S_{akt}.teil = oben;$
        ENDIF;

        // Traversiere weiter mit neuem Segment
        BREAK;
    ENDIF;
END;
END;

Komplexität:

Die Anzahl der Aufrufe ist höchstens so groß wie die Anzahl der Segmente. Die Anzahl der Segmente hängt aber von der Zahl der Schnittpunkte ab. Deren Zahl hängt wiederum mit der Zahl der Linien zusammen. Die Anzahl der Linien ist doppelt so groß wie die Zahl der Neuronen n (jedes Neuron wird durch eine *1*- und eine *0*-Linie dargestellt). Die Zahl der Schnittpunkte beträgt somit maximal $\sum_{i=1}^{2n-1} i = \frac{(2n-1)(2n-2)}{2}$. Die Anzahl der Segmente ist dann maximal $\sum_{i=1}^{2n} 2i-1 = (2n)^2$. Die Komplexität des Verfahrens setzt sich aus der Summe der Operationen zusammen, die für die einzelnen Schritte nötig sind. Die Komplexität ist gleich:

$$2 \cdot O((2n)^2) + O(n) \in O(n^2). \tag{4.20}$$

Der erste Summand entspricht der Zahl der Segmentteile ("oben", "unten") und der zweite der maximalen Anzahl der Segmente, die sich in einem Punkt berühren können.

4.1.7.11 Berechnung der Eckpunktehöhen der Polygonzüge

Um die Klassentrennlinie in die Polygonzüge einzutragen, muß die Höhe der Eckpunkte bekannt sein. Die Höhe könnte einfach durch das Propagieren eines speziellen Musters berechnet werden. Dies ist aber nicht immer möglich, denn welches Muster am Netzeingang angelegt werden muß, um genau den gesuchten Eckpunkt zu treffen, ist in vielen Fällen schwer zu bestimmen (z. B. bei hohen Dimensionen). In der geometrischen Interpretation wird daher die Höhe der Eckpunkte aus den Netzparametern berechnet. Die Approximation der logistischen Funktion ist so aufgebaut, daß auf einer Seite des Sicherheitsstreifens der Aktivierungswert gleich *1* und auf der anderen gleich *0* ist. Nur auf dem Sicherheitsstreifen selbst ändert sich der Wert entsprechend den Koordinaten des Punktes. Um die Aktivierung eines Eckpunktes zu bestimmen, muß lediglich festgestellt werden, auf welcher Seite des Sicherheitsstreifens sich dieser Punkt befindet. Liegt der Eckpunkt auf dem Sicherheitsstreifen, dann muß bestimmt werden, welche Aktivierung dieser Punkt hat. Der genaue Wert ergibt sich aus dem Verhältnis der Abstände zu der *0*- und *1*-Linie. Für jedes verdeckte Neuron wird sein Beitrag bestimmt und der entsprechende Betrag zur Gesamtaktivierung aufaddiert.

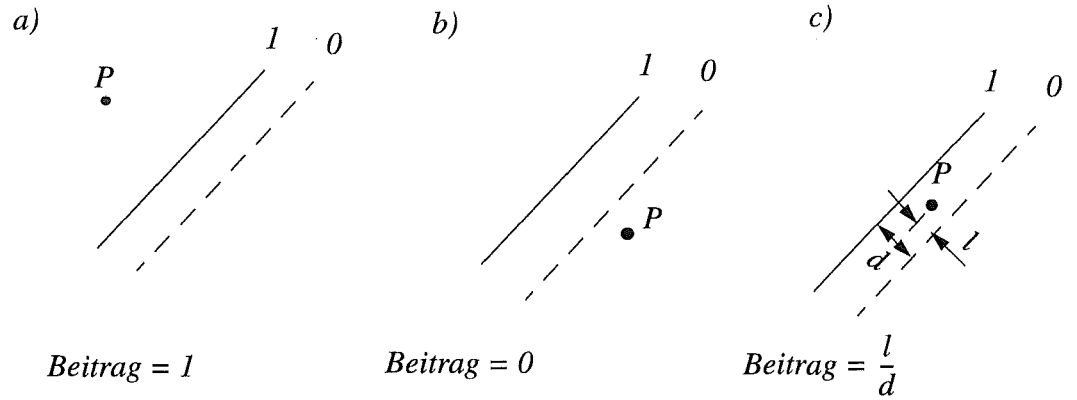


Abbildung 4.26: *Je nach der Position des Eckpunktes (in Bezug auf die 0- und 1-Höhenlinien des Neurons) wird ein Beitrag von 1 (Bild a), 0 (Bild b) oder, der Lage entsprechend, ein Wert zwischen 0 und 1 zur Höhe dazu addiert (Bild 3 c).*

Algorithmus 4.4

Für jeden Eckpunkt muß der folgende Algorithmus durchgeführt werden.

Eingabe:

Ein Eckpunkt $P=(x,y)$, Neuronenliste $N = \{n_1, n_2, ..., n_k\}$.

Ausgabe:

Die Höhe des Eckpunktes h.

Methode:

$h = 0$;
FOR $i=0$ **TO** k **DO** // Beiträge aller Neuronen ermitteln
    $neuron = n_i$;
    IF *P(x, y) liegt auf der 1-Halbebene* **THEN**
        $h = h + 1$;
    ELSE IF *P(x, y) liegt auf dem Sicherheitsstreifen* **THEN**

$$h = h + \frac{d(P(x, y), \text{0-Höhenlinie})}{d(\text{0-Höhenlinie, 1-Höhenlinie})};$$

    // d(,) liefert den Abstand zw. einem Punkt und einer Linie bzw. zw.
    // zwei Linien.

ENDIF;
END;
RETURN(*h* **);**

Komplexität:
Die Berechnung muß für jeden Eckpunkt und jedes Neuron durchgeführt werden. Daraus ergibt sich die Komplexität zu:

$$O(n)O\left(\frac{(2n-1)(2n-2)}{2}\right) \in O(n^3) . \tag{4.21}$$

4.1.7.12 Eintragen der Klassentrennlinie

Die geometrische Interpretation visualisiert ein klassifizierendes neuronales Netz durch die Darstellung der Klassentrennlinie. Deswegen muß die Trennlinie in die Visualisierung des interpretierten Neurons eingetragen werden. Da die Darstellung der Funktion, welche von einem bestimmten Neuron realisiert wird, aus vielen Polygonzügen besteht, wird in jeden dieser Polygonzüge die Klassentrennlinie separat eingetragen.

Um die Klassentrennlinie einzuzeichnen, muß ihre Höhe bekannt sein. Die Höhe wird durch eine Rückberechnung des Klassentrennwerts am Netzausgang auf die Aktivierung des interpretierten Neurons bestimmt. Dazu wird der Klassentrennwert rückwärts durch das Netz durchpropagiert. Jedes passierte Gewicht wirkt auf das rückpropagierte Signal mit seinem Kehrwert, jedes Neuron mit der inversen Aktivierungsfunktion. Die Berechnung setzt sich also aus der wiederholten Anwendung der folgenden Formel zusammen:

$$th_{S_{i+1}} = \frac{1}{w_{S_i, S_{i+1}}} ln\left(\frac{1}{th_{S_i}} - Bias_i\right) , \tag{4.22}$$
$$th_{S_0} = ktw.$$

mit:

ktw	der Klassentrennwert am Ausgabeneuron,
$S_0, \ldots, S_k$	eine Folge von Indizes der Neuronen, welche zwischen dem Ausgabeneuron und dem interpretierten Neuron liegen und
$w_{S_i, S_{i+1}}$	das Gewicht vom S_{i+1}-ten zum S_i-ten Neuron.

Jede solche Berechnung erfordert zwei Divisionen, eine Multiplikation, eine Subtraktion und eine Logarithmusberechnung pro Neuron. Da alle diese Operationen eine Komplexität $O(1)$ haben, ergibt sich die Gesamtkomplexität für ein Neuron der k-ten Schicht in einem m-schichtigen Netz zu:

$$2(m-k+1)O(1) + O(1) + O(1) + O(1) \in O(m-k) . \tag{4.23}$$

Der Eintrag der Klassentrennlinie in die Darstellung eines Neurons einer weiteren Schicht ist komplexer, als das der Fall bei den verdeckten Neuronen der ersten Schicht ist. Die Klassentrennlinie muß in jeden Polygonzug der Darstellung separat eingetragen werden. Die einzelnen Abschnitte ergeben die gesamte Klassentrennlinie. Ein Abschnitt der Klassentrennlinie schneidet einen Polygonzug und muß dort eingezeichnet werden, wenn:

Fall 1. der Polygonzug mindestens zwei Eckpunkte hat, von denen ein Eckpunkt über bzw. auf und der andere unter bzw. auf dem Trennwert liegt,

Fall 2. einer der Eckpunkte die Höhe der Klassentrennlinie hat.

Liegt die Klassentrennlinie zwischen zwei Ecken eines Polygonzuges (Fall 1), dann wird eine Höhenlinie in den Polygonzug eingetragen, die dem Klassentrennwert entspricht; berührt der Polygonzug nur mit einer Ecke den Klassentrennwert (Fall 2), dann wird der entsprechende Punkt als Teil der Trennlinie markiert.

Um eine Höhenlinie in den Polygonzug einzutragen, werden zwei Punkte auf dem Polygonzug mit der Höhe der Klassentrennlinie gesucht und durch eine gerade Strecke verbunden.

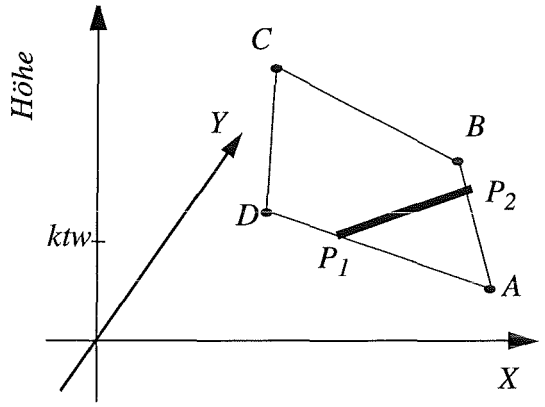


Abbildung 4.27: *Eintragung des Klassentrennwertes in einen Polygonzug. Der Klassentrennwert ktw liegt zwischen den Höhen der Punkte A und D sowie A und B. Die Klassentrennlinie entsteht durch die Verbindung der Punkte P_1 und P_2, welche auf den Segmenten $\overline{AD}$ und $\overline{AB}$ liegen und die Höhe von ktw haben.*

Algorithmus 4.5

Für jeden Polygonzug muß der folgende Algorithmus durchgeführt werden.

Eingabe:

Polygonzug Y mit Ecken $A_1=(x_1, x_2, h_1)$, $A_2=(x_2, y_2, h_2)$, ..., $A_n=(x_n, y_n, h_n)$ und Klassentrennwert *ktw*.

Ausgabe:

Anfang $P_1=(X_1, Y_1)$ und Endpunkt $P_2=(X_2, Y_2)$ der eingezeichneten Höhenlinie.

Methode:

$k = 1;$
FOR $i=0$ **TO** $i<n$ **DO**
  IF *ktw zwischen Höhe(A_i) und Höhe($A_{(i+1) \; MOD \; n}$)* **OR**
    ktw zwischen Höhe(A_i) und Höhe($A_{i+1 \; MOD \; n}$) **THEN**
  IF *Kante waagerecht* **AND** *Kantenhöhe == ktw* **THEN**
    $X_1 = X_2 = x_i;$
    $Y_1 = Y_2 = y_i;$
    RETURN;
  ENDIF;
  IF $h_i > h_{(i+1) \; MOD \; n}$ **THEN**
    $lowX = x_{(i+1) \; MOD \; n};$
    $lowY = y_{(i+1) \; MOD \; n};$
    $lowH = h_{(i+1) \; MOD \; n};$

$highX = x_i$;
$highY = y_i$;
$highH = h_i$;
ELSE
$highX = x_{(i+1) \; MOD \; n}$;
$highY = y_{(i+1) \; MOD \; n}$;
$highH = h_{(i+1) \; MOD \; n}$;
$lowX = x_i$;
$lowY = y_i$;
$lowH = h_i$;
ENDIF;

$$x_k = lowX + \frac{(highX - lowX)(ktw - lowH)}{highH - lowH};$$

$$y_k = lowY + \frac{(highY - lowY)(ktw - lowH)}{highH - lowH};$$

$k=k+1$;
END;

// Wenn der Polygonzug nur in einem Eckpunkt den Wert ktw annimmt
IF $k==1$ **THEN**
$x_2 = x_1$;
$y_2 = y_1$;
END;

Komplexität:
Jeder Eckpunkt eines Polygonzuges muß betrachtet werden, d. h. die Komplexität beträgt $O(n^2)$.

4.2 Visualisierung hochdimensionaler Eingaberäume

In praktischen Anwendungen haben neuronale Netze (meistens) einen hochdimensionalen Eingaberaum, d. h. sie haben viele Eingabeneuronen. Solche Netze lassen sich in den meisten Fällen nicht direkt durch eine Projektion auf zwei ausgewählte Dimensionen visualisieren, sondern sie müssen erst durch eine geschickte Transformation so abgebildet werden, daß eine Projektion auf nur zwei transformierte Koordinaten eine gute Darstellung sowohl der Daten als auch des Netzes ergibt. Die Visualisierung des Netzes kann auf zwei verschiedene Weisen erfolgen. Zum einen kann die Klassentrennlinie im hochdimensionalen Raum als Hyperebene berechnet und nach einer Transformation sowie einer Projektion im zweidimensionalen Raum dargestellt werden. Zum anderen können zuerst die Neuronen der ersten verdeckten Schicht in einen zweidimensionalen Raum transformiert und hineinprojiziert werden und erst dann die Klassentrennlinienberechnung durchgeführt werden. Beide Methoden haben Vor- und Nachteile, die später noch diskutiert werden. Zuerst aber werden die beiden Methoden angegeben, um so der Diskussion eine Grundlage zu geben.

4.2.1 Berechnung der Visualisierung vor der Projektion

Die Berechnung der Klassentrennlinie in einem hochdimensionalen Raum erfolgt auf ähnliche Weise wie für einen zweidimensionalen Raum. Als erstes werden die Polygonzüge berechnet, die sich aus der Aufteilung des Eingaberaumes ergeben. Dazu wird der Algorithmus 4.3 (Kapitel 4.1.7.10) verwendet.

Im Unterschied zum Algorithmus 4.3 können sich jetzt mehr als zwei Hyperpolygonzüge in einer Kante (Segment) berühren, da sich jetzt auch z. B. Flächen schneiden können. Deswegen wird der Algorithmus so erweitert, daß anstelle der Begriffe "oben" und "unten" alle Ebenenlagen für die Suchrichtungen treten. Im dreidimensionalen Fall wird z. B. die Winkellage der Ebenen gespeichert, in der die Polygonzüge liegen, d.h. wenn in einem Segment vier Polygonzüge zusammenkommen, dann merkt sich der Algorithmus vier verschiedene Winkel. Die Polygonzugsuche erfolgt dann innerhalb einer Ebene nach dem Algorithmus 4.3.

Neue Algorithmusschritte. In einem n-dimensionalen Eingaberaum wird der Klassentrennwert als eine n-1-dimensionale Hyperebene dargestellt. Die Berechnung dieser Hyperebene erfolgt durch das Einzeichnen in einen n-dimensionalen Hyperpolygonzug (analog zum Einzeichnen einer Linie in einen Polygonzug, s. Kapitel 4.1.7.12). Der Hyperpolygonzug entsteht durch eine sukzessive Zusammenlegung von *2-, 3-,..., n-1*-dimensionalen Hyperpolygonzügen. Für die Visualisierung von neuronalen Netzen mit einer hochdimensionalen Eingabe bedeutet es, daß ein neuer Berechnungsschritt erforderlich ist, der diese sukzessive Zusammenlegung von Hyperpolygonzügen durchführt. Um das Problem zu verdeutlichen, betrachten wir die Zusammenlegung von zweidimensionalen Polygonzügen zu dreidimensionalen Hyperpolygonzügen (Volumenelementen).

Beispiel 4.15. Betrachten wir die Polygonzüge aus Abbildung 4.28, die in der Mitte einen Würfel bilden. Jeder dieser Polygonzüge berührt mit seinen Kanten andere Polygonzüge. Die Zusammenlegung beginnt mit dem Polygonzug Y_1. Dieser Polygonzug ist ein Quadrat und hat vier Kanten (Segmente). Der Algorithmus geht von einem Segment aus und tastet sich im Inneren des Würfels an den Innenwänden weiter, bis der ganze Würfel gefunden wird. In unserem konkreten Fall ergibt sich folgendes Traversieren:

TABELLE 5. Traversieren der Polygonzüge aus Abbildung 4.28

Ausgangs-polygonzug[a]	Segment[b]	Nächster Polygonzug schon mal besucht?[c]	Aktion
Y_1	$S_{1,2}$	nein (Y_2)	markiere Y_2; gehe zu Y_2;
Y_2	$S_{1,8}$	nein (Y_6)	markiere Y_6; gehe zu Y_6;
Y_6	$S_{4,1}$	ja (Y_1)	nächste Kante von Y_6;
Y_6	$S_{4,7}$	nein (Y_4)	markiere Y_4; gehe zu Y_4;
Y_4	$S_{7,5}$	nein (Y_3)	markiere Y_3, gehe zu Y_3;
Y_3	$S_{7,8}$	ja (Y_6)	nächste Kante von Y_3;
Y_3	$S_{8,6}$	ja (Y_2)	nächste Kante von Y_3;
Y_3	$S_{6,5}$	nein (Y_5)	markiere Y_5; gehe zu Y_5;
Y_5	$S_{5,3}$	ja (Y_4)	nächste Kante von Y_5;
Y_5	$S_{3,2}$	ja (Y_1)	nächste Kante von Y_5;
Y_5	$S_{2,6}$	ja (Y_2)	nächste Kante von Y_5;
Y_5	$S_{6,5}$	ja (Y_3)	gehe zurück zu Y_3;
Y_3	$S_{1,2}$	ja (Y_1)	gehe zurück zu Y_4;
Y_4	$S_{5,3}$	ja (Y_5)	nächste Kante von Y_4;
Y_4	$S_{4,3}$	ja (Y_1)	nächste Kante von Y_4;

TABELLE 5. Traversieren der Polygonzüge aus Abbildung 4.28

Ausgangs-polygonzug[a]	Segment[b]	Nächster Polygonzug schon mal besucht?[c]	Aktion
Y_4	$S_{4,7}$	ja (Y_6)	gehe zurück zu Y_6;
Y_6	$S_{7,8}$	ja (Y_3)	nächste Kante von Y_6;
Y_6	$S_{8,1}$	ja (Y_2)	gehe zurück zu Y_1;
Y_1	$S_{2,3}$	ja (Y_5)	nächste Kante von Y_1;
Y_1	$S_{3,4}$	ja (Y_4)	nächste Kante von Y_1;
Y_1	$S_{4,1}$	ja (Y_6)	fertig;

a. In diesem Polygonzug befindet sich der Algorithmus gerade (Anordnung s. Abbildung 4.28)

b. Dieses Segment wird betrachtet

c. Dieser Polygonzug grenzt an den Ausgangspolygonzug mit der Kante "Segment"

Der Algorithmus beginnt mit einem Segment eines beliebigen (noch nicht besuchten) Hyperpolygonzuges, geht dann zum anderen (zweiten) Hyperpolygonzug, welcher sich mit dem Ausgangshyperpolygonzug (in dem Ausgangssegment) berührt. Aus dem zweiten Hyperpolygonzug wird ein neues Segment gewählt und geprüft, ob der Hyperpolygonzug, welcher an dieses Segment angrenzt, schon mal besucht wurde. Ist das nicht der Fall, dann fährt der Algorithmus mit dem angrenzenden Hyperpolygonzug fort. Ansonsten wird ein neues Segment gewählt. Diese Prozedur geht solange weiter, bis alle Segmente durchprobiert sind. Dann kehrt der Algorithmus zum Ausgangshyperpolygonzug zurück und versucht es dort mit einem weiteren Segment. Sind alle Segmente des Ausgangshyperpolygonzuges bearbeitet, dann ist der Algorithmus fertig, und der Hyperpolygonzug ist gefunden. ■

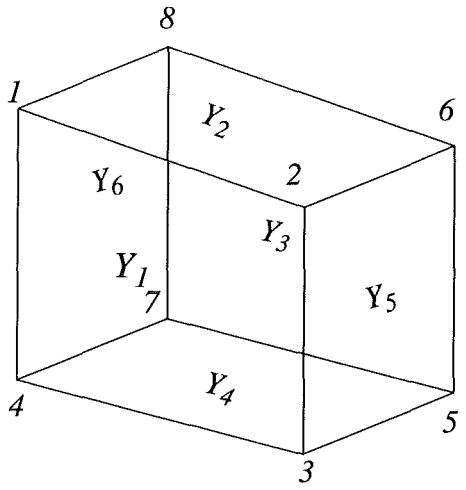


Abbildung 4.28: *Anordnung der Polygonzüge im Würfel. Y_1 ist der vordere, Y_3 der hintere, Y_5 der rechte, Y_6 der linke, Y_2 der obere und Y_4 der untere Polygonzug. Die Nummern 1 bis 8 bezeichnen die Eckpunkte.*

Komplexität. Bei einer k-dimensionalen Eingabe muß dieses Paradigma k-1-mal wiederholt werden. Die Komplexität erhöht sich also um den Faktor k-1 in Bezug auf die Komplexität aus Algorithmus 4.3, d. h. die Komplexität beträgt $O(kn^2)$ (n ist die Zahl der Neuronen).

4.2.2 Berechnung der Visualisierung nach der Projektion

Eine Berechnung der Visualisierung nach der Projektion ist nur dann möglich, wenn die Transformation, welche bei der Dimensionenreduktion benutzt wurde, linear ist. Für die Durchführung dieser Berechnung werden die Neuronen der ersten verdeckten Schicht in den neuen Raum transformiert und dann auf zwei Dimensionen projiziert. Das Ergebnis ist eine Neuronendarstellung, welches im transformierten Raum einen zweidimensionalen Merkmalsraum hat. Alle weiteren Berechnungen erfolgen mit denselben Algorithmen wie in Kapitel 4.1.7.6 beschrieben.

4.2.3 Reduktion der Dimensionalität (Projektion) vor oder nach der Berechnung der Visualisierung

Die Transformation im hochdimensionalen Raum ist vom Algorithmus her komplexer. Sie hat aber den Vorteil, daß sie keine Einschränkungen bezüglich der Transformation stellt. Die Transformation kann entweder linear oder nichtlinear sein, sie kann in Form einer Gleichung oder als Programm (z. B. neuronales Netz, s. Kapitel 3.3.2 bzw. Kapitel 3.3.4) vorliegen. Der Nachteil ist, daß die Darstellung in manchen Fällen nicht durch algebraische, sondern durch eine numerische Berechnung erfolgt. Das induziert, daß für die Manipulation des Netzes auf jeden Fall eine inverse Transformation vorliegen muß.

Eine Berechnung der Visualisierung nach der Projektion hat den Vorteil, daß sie viel effizienter ist. Denn neben der kleineren Berechnungskomplexität der Visualisierung müssen hier lediglich nur die Neuronen der ersten verdeckten Schicht transformiert werden. Der Einschränkung, daß lineare Transformationen verwendet werden dürfen, kann aber durch die Anwendung einer stückweise linearisierten Approximation einer nichtlinearen Transformation behoben werden. Eine besonders gut für die Visualisierung neuronaler Netze geeignete Transformation ist die Local Principal Component Analysis (LPCA).

In dieser Arbeit fiel die Entscheidung für die zweite Methode, da sie sich besser für die Durchführung einer Manipulation eignet. Die eigens dafür entwickelte LPCA gleicht die Nachteile gegenüber der ersten Methode aus.

4.3 Lokale PCA

Die Idee der lokalen PCA ist recht einfach: Verschiedene Bereiche des visualisierten Bereiches werden mit einer separaten PCA-Transformation dargestellt. Dadurch werden Bereiche, in denen die Richtung der größten Varianz in eine andere Richtung als die Varianz der gesamten Daten verläuft, aus stückweise verschiedenen Richtungen optimal dargestellt. Wenn die PCA in bestimmten Regionen des Definitionsbereichs keine zufriedenstellenden Ergebnisse liefert, kann sie durch eine andere lineare oder auch nichtlineare Transformation ersetzt werden. Wesentlich ist nur die lokale Anwendung der Transformationen, da nur dadurch eine gute Darstellung (auch nichtlinearer Datenverteilungen) erreicht werden kann. Die lokalen Transformationen werden in sog. *Transformationsfenstern* durchgeführt. Die Wahl der Fenster ist entscheidend für die Güte der LPCA.

Auswahl der Transformationsfenster. Bei einer Visualisierung ist der Benutzer die höchste Instanz, welche letztendlich entscheidet, ob die angezeigte Darstellung optimal ist. Deswegen wählt auch der Benutzer selbst die Fenster aus, indem er einen Teil des visualisierten Bildes markiert. Eine automatische Fensterwahl nach dem Prinzip der Clusteranalyse kann ebenfalls eingesetzt werden. Sie ist aber nur eine Hilfe, welche in Anspruch genommen werden kann.

Auswahl der Transformation. In jedem Fenster kann eine andere Transformation der Daten und des Netzes vorgenommen werden. Die Transformation kann zum einen datenbezogen sein, d. h. die Transformationsvorschrift wird aus der Datenverteilung ausgerechnet (PCA, CCA etc.), oder sie wird so realisiert, daß das neuronale Netz die Projektionsrichtung angibt. Dabei werden die Richtungen bevorzugt, in welchen die großen Trennflächen senkrecht zur Darstellungsebene liegen. Diese Richtungen können bestimmt werden, indem die Normalen der Trennflächen aufsummiert werden und so die Richtung der Visualisierung bestimmen. Die gesuchte Visualisierungsrichtung ist senkrecht zum Summenvektor. Wählt man das Netz als Bezugspunkt der Visualisie-

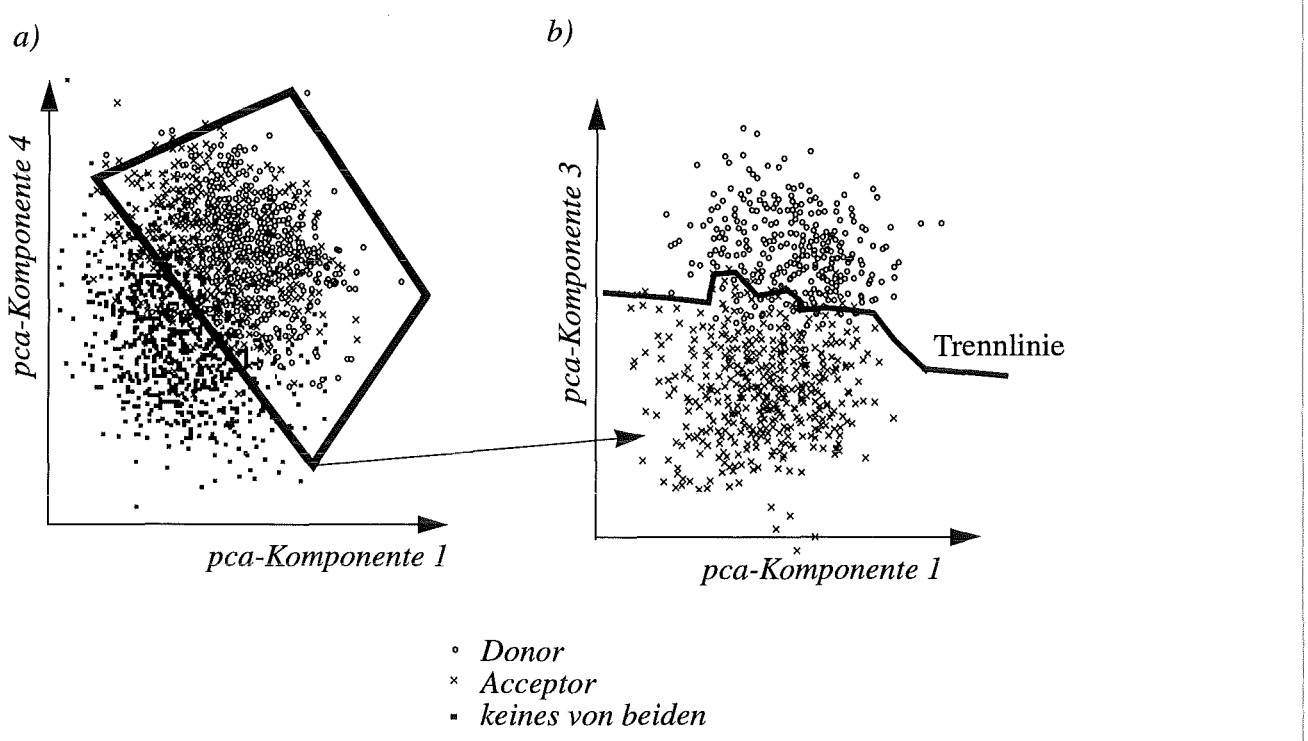


Abbildung 4.29: *Ein 180-dimensionales Problem, visualisiert mit der lokalen PCA. Bild a): Nach einer gewöhnlichen PCA-Transformation mit allen Mustern; Bild b): nach einer (lokalen) Transformation des Fensters. Das Fenster wurde in allen Dimensionen beschränkt. Die Beschränkung in nicht dargestellte Richtungen erfolgt mit Hilfe des Cluster-Algorithmus.*

rung, so muß man sich im Klaren sein, daß sich die Visualisierungsperspektive während des Lernvorgangs ändern kann. Deshalb ist dieser Bezugspunkt nur auf angelernte Netze empfehlungswert. Im weiteren bestimmt ausschließlich die Datenverteilung die Projektionsrichtung.

Beispiel 4.16. Mit der LPCA wurde ein DNA-Benchmark [Prechelt94] visualisiert. Die Eingabedimension beträgt *180*. Nach einer ersten PCA-Transformation ergibt sich die Darstellung aus Abbildung 4.29a. Nachdem der schlecht dargestellte Bereich durch ein Fenster markiert und erneut PCA-transformiert wurde, konnten die Muster im gesamten Definitionsbereich gut voneinander getrennt werden.
■

4.4 Zusammenfassung

Es wurde die geometrische Interpretation neuronaler Netze beschrieben. Die Erläuterungen begannen mit dem einfachsten Element, dem Neuron, und endeten mit der Erläuterung der Visualisierung eines Netzes mit zwei und mehr verdeckten Schichten. Dabei wurde die Betonung vor allem auf neuronale Netze mit einer und zwei verdeckten Schichten gelegt (da diese am häufigsten genutzt werden). Beschrieben wurde nicht nur die Visualisierung, sondern auch die Bedeutung der Darstellungselemente, wie Klassenlinien oder Sicherheitsstreifen. Weil eine einfache Projektion für die Darstellung hochdimensionaler Eingaberäume nicht ausreicht, wurde die lokale PCA (LPCA) eingeführt, welche eine gute Visualisierung derartiger Räume erlaubt. Wie mehrmals betont wurde, ist die GINN so entwickelt worden, daß sie leicht eine Manipulation und Optimierung der Netzstruktur ermöglicht. Diese beiden Aspekte werden im nächsten Kapitel beschrieben.

Kapitel 5

Manipulation von neuronalen Netzen und Topologie-Optimierung

Als Motivation für dieses Kapitel sei die folgende Beobachtung herangezogen: Eine Lösung, die von einem neuronalen Netz gefunden wird, entspricht nur selten den Vorstellungen des Anwenders. Der Grund dafür liegt in den unvollständigen Trainingsmustern und/oder im Lernverfahren. Die meisten Lernalgorithmen realisieren lediglich eine Approximation der Lerndaten und stellen keinen Bezug zur späteren Anwendung her. Bei Klassifikationsaufgaben zum Beispiel ist aber der Approximationsfehler ohne größere Bedeutung. Worauf es ankommt, ist der Verlauf der Klassentrennlinie. Das folgende Beispiel demonstriert einen solchen Fall.

Beispiel 5.1. Ein neuronales Netz soll eine Klassenaufteilung realisieren, welche die Form eines Rechtecks hat. Das Netz ist in Abbildung 5.1a dargestellt und seine geometrische Visualisierung in Abbildung 5.1b. Aus der Visualisierung kann entnommen werden, daß diese Aufgabe schlecht gelöst wurde. Anstatt mit Hilfe von vier verdeckten Neuronen die vier Kanten des Rechtecks zu bilden, versucht das Netz, die Klassenwerte zu approximieren und achtet dabei nicht auf die Form der Trennlinie. Daß diese konkrete Lösung gefunden wurde, liegt an der Initialisierung der Neuronen und dem gewählten Lernalgorithmus (Backpropagation).

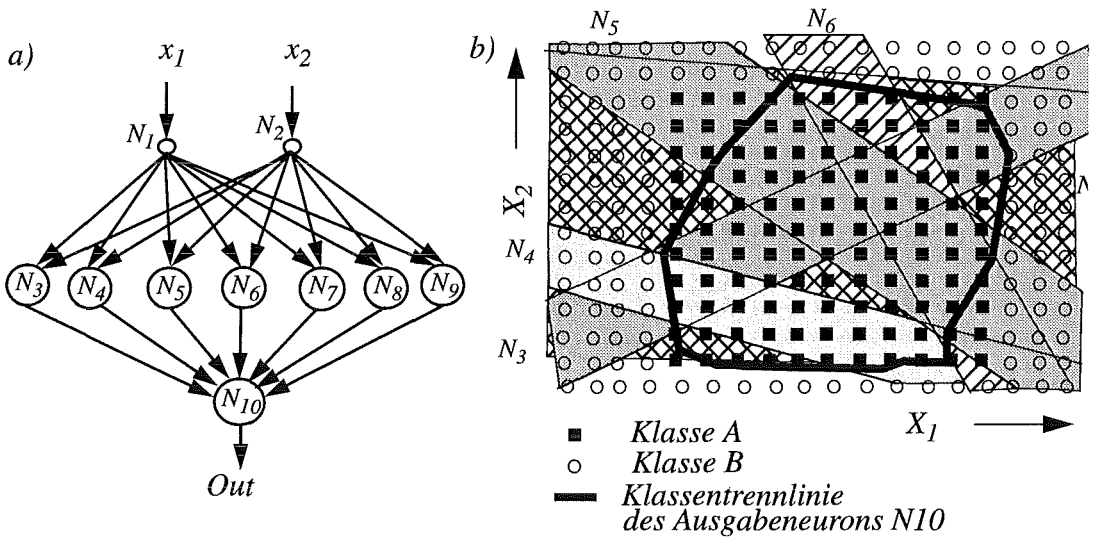


Abbildung 5.1: *Das neuronale Netz aus a) löst ein Klassifikationsproblem nicht so, wie es erwartet wird (Bild b). Die schraffierten Flächen stellen den Verlauf der Sicherheitsstreifen der Neuronen N_3 bis N_7 dar. Die Neuronen N_8 und N_9 liegen außerhalb des Definitionsbereiches und haben somit keinen Einfluß auf die Lösung.*

Durch die Manipulation von vier und das Ausschneiden von drei Neuronen kann die vom Anwender gewünschte Lösung herbeigeführt werden. Bei der Manipulation wurden die Parameter der Neuronen N_3, N_4, N_5 und N_6 so verändert, daß ihre Aktivierungsfunktion einen fast stufenartigen Verlauf hat.

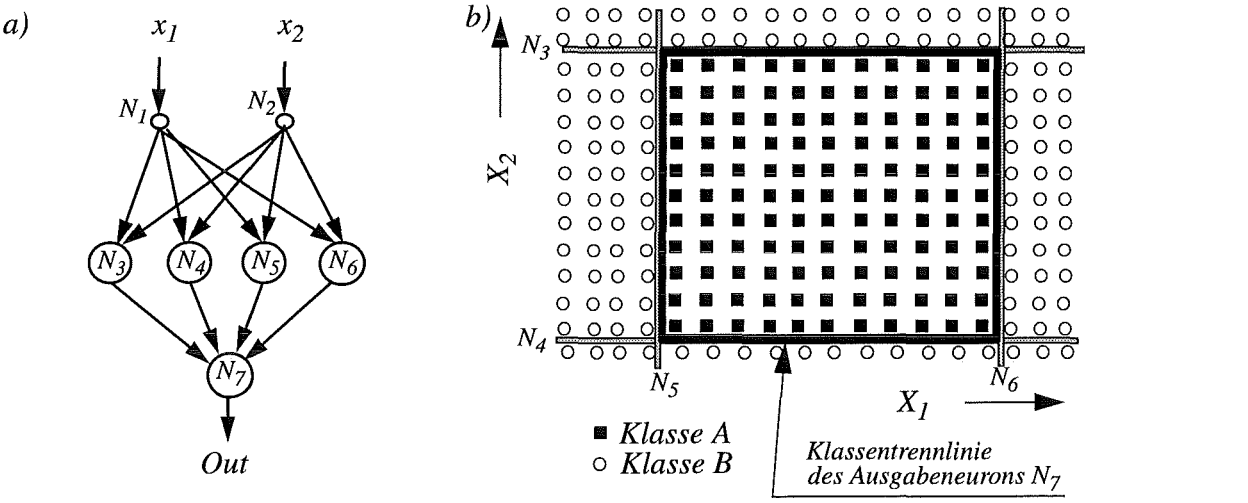


Abbildung 5.2: *Nach der Manipulation des Netzes, d. h. dem Entfernen der Neuronen N_7, N_8 und N_9 und einer Veränderung der Neuronenparameter von N_3, N_4, N_5 und N_6 liefert das neuronale Netz die gewünschte Lösung.*

■

Eine Netzmanipulation soll soweit wie möglich auf eine intuitive Weise erfolgen, d. h. der Benutzer soll nicht darauf angewiesen sein, die Gewichte und Bias-Werte einzeln zu ändern, sondern er soll, z. B. mit Hilfe einer Maus, die Neuronen graphisch manipulieren können, indem er ähnlich wie mit einem Zeichenprogramm arbeitet. In den folgenden Abschnitten werden die Manipulationsalgorithmen beschrieben.

5.1 Manipulation

Die Parameter eines Netzes (Gewichte und Bias-Werte) sollen nicht von Hand verändert werden, sondern aus einer Manipulation der Visualisierungselemente. Daß dabei viele Parameter gleichzeitig verändert werden müssen, sollte dem Benutzer verborgen bleiben. Er soll seinerseits nur graphische (geometrische) Manipulationen auf den Neuronendarstellungen oder auf der Klassentrennlinie durchführen müssen. Es stehen ihm mehrere elementare Manipulationen zur Verfügung. Komplexe Manipulationen entstehen durch das Hintereinanderausführen elementarer Manipulationen (Macros), z. B. kann eine Drehung um einen beliebigen Punkt *A* durch eine Verschiebung zum Koordinatenursprung, eine Drehung um den Koordinatenursprung und eine Zurückverschiebung zum Punkt *A* realisiert werden. In dieser Arbeit sind affine Abbildungen wie Drehung, Verschiebung und Skalierung elementar. Damit eine einheitliche Terminologie geschaffen wird, ist in den nächsten Abschnitten immer dann, wenn von einer Manipulation eines Neurons die Rede ist, die Manipulation seiner *0*- und *1*-Linie gemeint. Z. B. bedeutet das Einfügen eines Neurons das gleichzeitige Einfügen seiner *0*- und *1*-Linie. Damit jede Manipulation möglich ist, gibt es eine gewisse Anzahl von Elementaroperationen. Sie wurden so gewählt, daß ihre Komposition eine beliebige Manipulation erlaubt. Ihre Wahl folgte aus der Durchführbarkeitsstudie der Operationen von GINN.

5.1.1 Elementare Manipulationen

Alle elementaren Manipulationen werden auf der Darstellung der Neuronen durchgeführt und erst dann in die Parameter der Neuronen umgerechnet. Es gibt viele mögliche Definitionen einer elementaren Manipulation. In dieser Arbeit werden neben den drei affinen Abbildungen (Drehen, Verschieben und Skalieren) noch zusätzlich das Einfügen und Löschen von Neuronen bzw. Neuronenschichten in die Menge der elementaren Manipulationen aufgenommen. Es gelten folgende Manipulationen als elementar:

- Drehung um einen Winkel α um den Koordinatenursprung,
- Verschiebung um einen Vektor $\boldsymbol{T}$,
- Skalierung,
- Drehung, Verschiebung und Skalierung eines Teils der Klassentrennlinie,
- das Einfügen von Neuronen,
- das Entfernen von Neuronen,
- das Einfügen einer neuen Schicht und
- das Entfernen einer bestehenden Schicht.

Wie in Kapitel 4 ("Geometrische Interpretation") erläutert wurde, wird ein Neuron der ersten verdeckten Schicht durch zwei Linien, nämlich die *0*- und *1*-Linien visualisiert, ein Neuron der zweiten verdeckten Schicht hingegen durch den von ihm realisierten Teil der Klassentrennlinie, der als Polygonzug dargestellt wird. Operationen auf Neuronen der ersten und zweiter verdeckten Schicht beziehen sich immer auf ihre Repräsentationen.

Die Operationen werden vom Benutzer auf dem Bildschirm durchgeführt und sind deswegen für ihn zweidimensional. Die manipulierten Komponenten entsprechen aber nicht direkt den Eingabedimensionen, da sie in den meisten Fällen mit einer Transformation (z. B. PCA oder LPCA) vorher abgebildet wurden. Damit die Manipulation trotzdem korrekt abläuft, muß das manipulierte Bild zuerst in den Originalbereich zurücktransformiert und dann manipuliert werden. Nach der Rücktransformation kann es

aber vorkommen, daß nicht nur zwei, sondern mehrere Dimensionen des Eingaberaums manipuliert werden müssen. Das ist dann der Fall, wenn z. B. eine transformierte Komponente eine Funktion von zwei oder mehr Eingabedimensionen ist. In diesem Fall führt die geometrische Interpretation diese Manipulation so aus, daß die wenigsten Eingabedimensionen verändert werden. Der Benutzer kann aber auch entsprechende Nebenbedingungen formulieren. Zum einen kann er fordern, daß die Änderung nur durch die Manipulation in bestimmten Dimensionen erfolgt. Zum anderen kann er festlegen, daß sich während der Manipulation nur bestimmte Gewichte oder Bias-Werte ändern bzw. nicht ändern dürfen. Die Art der Berechnung hängt im wesentlichem von der topologischen Lage eines Neurons ab. Danach richtet sich die Beschreibung. Wie das schon bei der Erläuterung der GINNder Fall war, wird auch hier die Erklärung zuerst nur für zweidimensionale Eingaberäume gegeben und dann auf höhere Dimensionen erweitert.

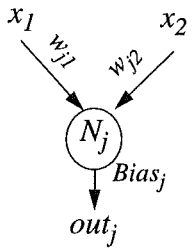
5.1.1.1 Operationen auf Neuronen der ersten verdeckten Schicht

Neuronen der ersten verdeckten Schicht führen die allererste Aufteilung des Eingaberaumes durch. Deshalb sind deren Ausgaben die Bausteine, aus welchen die Neuronen weiterer Schichten ihre Klassentrennlinien zusammensetzen. Operationen auf diesen Neuronen haben somit eine fundamentale Wirkung auf die Arbeitsweise des Netzes. Sie resultieren direkt aus Manipulationen, welche auf den Neuronen durchgeführt werden, oder sie sind Nebeneffekte von Manipulationen, die auf Neuronen einer weiteren Schicht durchgeführt wurden, z. B. kann die Drehung eines Neurons der zweiten verdeckten Schicht (genauer gesagt: der von ihm gebildeten Klassentrennlinie) durch die Drehung aller Neuronen der ersten verdecken Schicht realisiert werden, die an seiner Klassentrennlinienbildung beteiligt sind. Aus diesem Grunde sind die Operationen auf Neuronen der ersten verdeckten Schicht besonders wichtig. Sie werden jetzt der Reihe nach beschrieben.

Drehung. Die Drehung ist eine der am häufigsten genutzten Operationen. Sie kann sich auf den Verlauf der Klassentrennlinie im gesamten Definitionsbereich auswirken, d. h. nicht nur dort, wo es gewünscht wird. Das resultiert aus der Form dieser Trennlinie (einer Geraden). Der Benutzer soll deswegen die Möglichkeit haben, die Drehfreiheit der Neuronendarstellung einzuschränken, und zwar so, daß sie über einen bestimmten Bereich nicht herauskommt.

Hier wird nur die Drehung um den Koordinatenursprung beschrieben. Eine Drehung um einen beliebigen Punkt kann als eine Zusammensetzung von der Verschiebung des Drehpunktes zum Koordinatenursprung, einer Drehung um denselben und zurück realisiert werden.

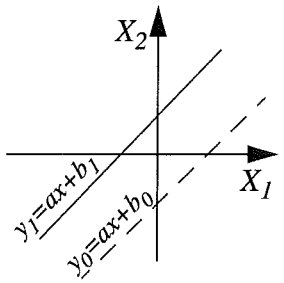
Bei einer Drehung handelt es sich um eine affine Abbildung, welche die *0*- und *1*-Linien eines Neurons um einen Punkt rotiert, ohne daß sich der Abstand zwischen der *0*- und der *1*-Linie ändert. Dieser Abstand entspricht der Steigung des linearen Übergangs (Sicherheitsstreifens). Damit dieser konstant bleibt, müssen die Gewichte zur Eingabeschicht und der Bias-Wert entsprechend dem Drehwinkel geändert werden. Um die Formel zu bestimmen, betrachten wir den Einfluß der Netzparameter auf die Lage der *0*- und *1*-Linie. Im zweidimensionalen Fall lauten die Liniengleichungen:



$$a = -\frac{w_{j1}}{w_{j2}}$$

$$b_0 = \frac{2 - Bias_j}{w_{j2}}$$

$$b_1 = \frac{-2 - Bias_j}{w_{j2}}.$$



Soll eine Drehung stattfinden mit der Randbedingung, daß der Abstand zwischen der *0*- und der *1*-Linie unverändert bleibt, dann müssen sich die Steigung a und die Verschiebungen b_0 bzw. b_1 ändern. In Neuronenparametern ausgedrückt: die Gewichte und der Bias-Wert müssen sich ändern. Aus der Geometrie ist bekannt, daß bei einer Drehung um einen Winkel α jeder Punkt $(x_1, x_2)^T$ mit einer Drehmatrix D der Form

$$D = \begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix}$$

multipliziert wird.

Wendet man diese Matrix auf die Gleichungen der *0*- und *1*-Linie an, dann ergeben sich folgende Formeln für diese Linien nach der Drehung:

$$y_1 = -\frac{sin(\alpha) + \left(-\frac{w_{j1}}{w_{j2}}\right)cos(\alpha)}{-\frac{w_{j1}}{w_{j2}}sin(\alpha) - cos(\alpha)}x - \frac{\frac{2 - Bias_j}{w_{j2}}}{-\frac{w_{j1}}{w_{j2}}sin(\alpha) - cos(\alpha)}$$

$$y_0 = -\frac{sin(\alpha) + \left(-\frac{w_{j1}}{w_{j2}}\right)cos(\alpha)}{-\frac{w_{j1}}{w_{j2}}sin(\alpha) - cos(\alpha)}x - \frac{\frac{-2 - Bias_j}{w_{j2}}}{-\frac{w_{j1}}{w_{j2}}sin(\alpha) - cos(\alpha)}.$$

Diese Formel muß noch in die Neuronenparameter umgerechnet werden.

Die Umrechnung liefert nicht direkt die neuen Gewichte w'_{j1} und w'_{j2}, sondern einen Quotienten q. Dieser Quotient gibt an, mit welchem Faktor das Gewicht w_{j2} multipliziert werden muß, um die gewünschte Drehung zu erhalten, ohne daß sich dabei der Abstand d zwischen den *0*- und *1*-Linien verändert. Das neue Gewicht w'_{j1} wird dann aus dem Gewicht w'_{j2} und dem Drehwinkel ausgerechnet. Wenn man berücksichtigt, daß das Gewicht w_{j1} der Steigung in X_1- und das Gewicht w_{j2} der Steigung in X_2-Richtung proportional ist (s. Formel (4.10), Seite 56), dann läßt sich der Faktor q als ein Quotient der Projektionen des Ab-

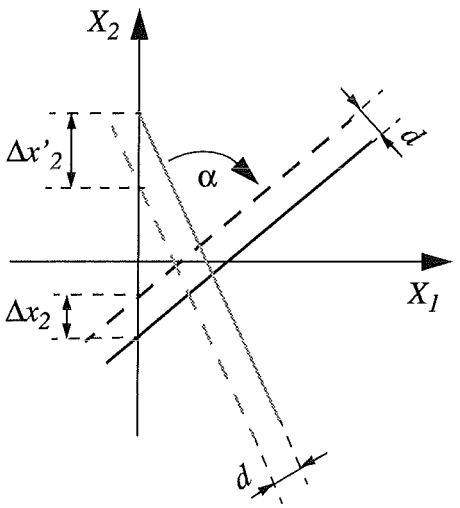


Abbildung 5.3: *Drehung eines Neurons.*

stands d auf die X_2-Achse (vor und nach der Drehung) berechnen. In Abbildung 5.3 sind diese Projektionen mit Δx_2 (vor der Drehung) bzw. $\Delta x'_2$ (nach der Drehung) bezeichnet:

$$\Delta x_2 = \frac{4}{w_{j2}}$$

$$\Delta x'_2 = \frac{4}{w_{j2}\left(-\frac{w_{j1}}{w_{j2}} sin(\alpha) - cos(\alpha)\right)}.$$

Der gesuchte Faktor q ist somit:

$$q = \frac{dx_2}{dx'_2} = -\frac{w_{j1}}{w_{j2}} sin(\alpha) - cos(\alpha),$$

das Gewicht w'_{j2} (nach der Drehung) beträgt dann:

$$w'_{j2} = w_{j2} q$$

und das Gewicht w'_{j1} (nach der Drehung):

$$w'_{j1} = -\frac{sin(\alpha) + cos(\alpha)\left(-\frac{w_{j1}}{w_{j2}}\right)}{\left(-\frac{w_{j1}}{w_{j2}} sin(\alpha) - cos(\alpha)\right)} w'_{j2}.$$

Verschiebung. Die Verschiebung eines Neurons der ersten verdeckten Schicht gehört, neben der Drehung um den Koordinatenursprung, zu den wichtigsten affinen Abbildungen. Die Verschiebung wird durch eine Änderung des Bias-Wertes erreicht. Sie kann in der X_1- oder in der X_2-Richtung erfolgen. Bei einer Verschiebung in X_1-Richtung wird der Bias-Wert gemäß der folgenden Formel verändert:

$$Bias' = Bias + w_{j1} T_{X_1}.$$

Wobei *Bias'* der Bias-Wert nach der Verschiebung und T_{x_1} der Betrag des Verschiebungsvektors in X_1-Richtung ist. Analoges ergibt sich für eine Verschiebung in die X_2-Richtung:

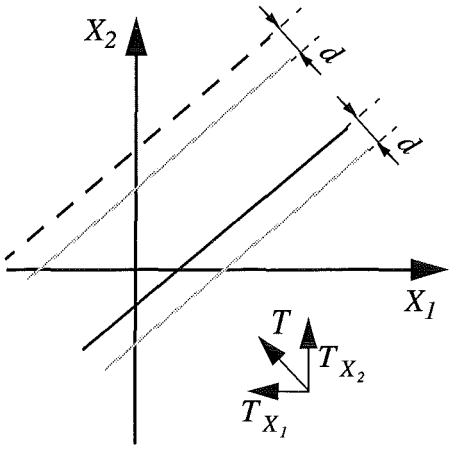


Abbildung 5.4: *Verschiebung eines Neurons um den Vektor T. Die Verschiebung setzt sich aus der Verschiebung entlang der X_1- und der X_2-Achse zusammen.*

$$Bias' = Bias + w_{j1} T_{X_2}.$$

Eine Verschiebung in einer beliebigen Richtung setzt sich zusammen aus Verschiebungen in X_1- und X_2-Richtung.

Eine andere Art der Manipulation ist die Skalierung. Sie erlaubt das Vergrößern und Verkleinern des Abstands zwischen den Klassentrennlinien.

Skalierung der Sicherheitsstreifens. Wie schon vorher erwähnt wurde, entspricht der Abstand zwischen der *0*- und der *1*-Linie der Steigung des linearen Übergangs (er wird

auch als Sicherheitsstreifen bezeichnet). Der Abstand kann vergrößert oder verkleinert werden. Das hat zur Folge, daß sich zum einen der Überlagerungsbereich des Neurons ändert, was bei der Überlagerung mit einem anderen Neuron zu einer kürzeren "abgeschnittenen Ecke" führt. Zum anderen werden die Höhenlinien dichter, was z. B. bei Klassifikationsaufgaben von Bedeutung ist.

In einer geometrischen Darstellung wird Skalierung durch das Verändern des Abstandes zwischen der *0*- und der *1*-Linie erreicht. Dies geschieht, indem die beiden Gewichte w_{j1} und w_{j2} zu w'_{j1} bzw. w'_{j2} so verändert werden, daß ihr Verhältnis unverändert bleibt, d. h.:

$$\frac{w_{j1}}{w_{j2}} = \frac{w'_{j1}}{w'_{j2}} = const.$$

Dadurch bleibt die Richtung des Neurons bestehen. Es gibt drei mögliche Arten einer Skalierung. Abbildung 5.5 zeigt diese Fälle.

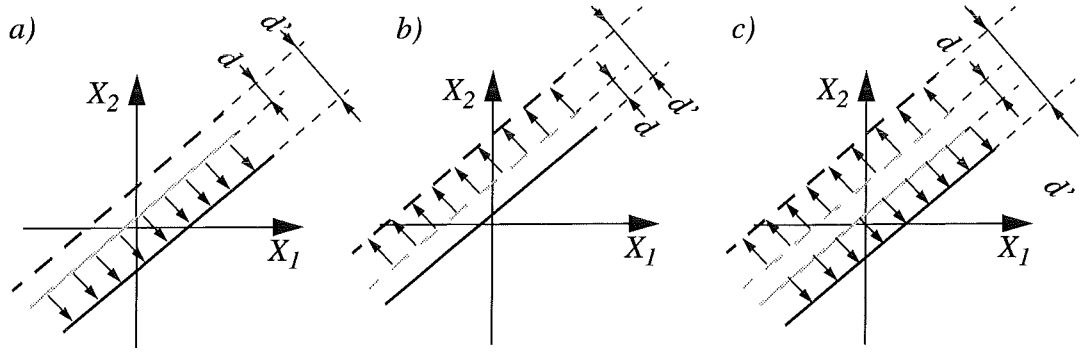


Abbildung 5.5: *Drei verschiedene Möglichkeiten für Skalierung: a) und b): eine der Linien wird weggezogen, c): beide Linien werden gleichermaßen weggeschoben.*

In allen drei Fällen wird der Abstand zwischen den Linien durch eine Änderung der Neuronenparameter erreicht und erst dann das Neuron an die gewünschte Position verschoben. Da die Verschiebung vorher schon beschrieben wurde, wird hier nur die Manipulation des Linienabstandes erläutert.

Um die Steigung in X_1-Richtung und X_2-Richtung um den Faktor $k=d'/d$ zu verändern, werden die Gewichte mit dem Faktor k multipliziert:

$$w'_{j1} = kw_{j1}$$
$$w'_{j2} = kw_{j2},$$

wobei w_{j1}, w_{j2} die Gewichte vor der Manipulation und w'_{j1} und w'_{j2} die Gewichte nach der Manipulation sind.

Als nächstes wird die Änderung der Stärke des Einflusses eines Neurons zur Lösung beschreiben. Diese Manipulation ist eine andere Art der Skalierung.

Änderung des Beitrags zur Lösung. Die Stärke der Verbindung zur nächsten Schicht bestimmt den Beitrag eines Neurons zur Gesamtlösung. Ist der Absolutbetrag des Gewichts klein, dann ist der Beitrag des Neurons zur Gesamtlösung in der Regel klein; ist er hingegen groß, dann ist auch der Beitrag in der Regel groß. Die Änderung des Beitrags wird also durch eine Änderung des Gewichts zum Ausgabeneuron vollzogen.
Eine Änderung des Gewichts um den Faktor k bewirkt gleichzeitig eine Änderung der

Steigung des linearen Übergangs um diesen Faktor, ohne daß sich der Abstand zwischen der *0*- und der *1*-Linie ändert. Im Unterschied zur Änderung der Steigung des linearen Übergangsbereiches, so wie es im vorherigen Abschnitt beschrieben wurde, wirkt sich diese Änderung auf den Beitrag des konstanten Bereiches (der *1*-Halbebene) zur Gesamtlösung aus. Das kann genutzt werden, um den Verlauf der Klassentrennlinie zu verändern. Eine andere Art der Manipulation ist die Änderung der Netztopologie. Die Netztopologie kann durch Einfügen bzw. Löschen eines Neurons erreicht werden.

Das Einfügen eines Neurons. Das Einfügen eines Neurons erfolgt durch das Einbringen seiner *0*- und *1*-Höhenlinie in die Visualisierung. Nachdem das Neuron plaziert wurde, wird es in die Netzstruktur eingefügt. Bei hochdimensionalen Räumen muß die Lage nicht nur in den gerade dargestellten, sondern in allen übrigen Dimensionen bestimmt werden. Die Gewichte zu Neuronen der zweiten verdeckten Schicht werden nach dem Einfügen auf *0* gesetzt.

Das Entfernen eines Neurons. Beim Entfernen eines Neuron, wird das Neuron aus der Netzstruktur gelöscht. Ein Neuron kann aber auch temporär gelöscht werden, indem alle Gewichte zu der zweiten verdeckten Schicht auf *0* gesetzt werden. Ein auf diese Weise gelöschtes Neuron kann dann wieder leicht in das Netz eingefügt werden.

5.1.1.2 Operationen auf Neuronen der zweiten verdeckten Schicht

Auf Neuronen der zweiten verdeckten Schicht können die gleichen Manipulationen wie auf Neuronen der ersten Schicht durchgeführt werden. Die Auswirkung und Realisierung sieht teilweise etwas anders aus, da manche dieser Manipulationen schon durch Manipulation der ersten Schicht bewältigt werden können. Zusätzlich zu den Operationen, die den Verlauf der ganzen Klassentrennlinie verändern, kann in der zweiten verdeckten Schicht auch ein Teil der Klassentrennlinie manipuliert werden.

Drehung um den Koordinatenursprung. Wenn ein Neuron der zweiten verdeckten Schicht um einen Winkel α um den Koordinatenursprung gedreht werden soll, dann müssen zuerst die Neuronen der ersten verdeckten Schicht gefunden werden, aus welchen die Klassentrennlinie des Neurons der zweiten verdeckten Schicht im Definitionsbereich kombiniert wird. Danach kann das Neuron der zweiten verdeckten Schicht gedreht werden, indem die vorher bestimmten Neuronen der ersten verdeckten Schicht einzeln um den Koordinatenursprung gedreht werden.

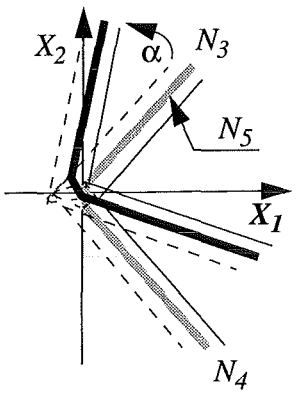


Abbildung 5.6: *Drehung eines Neurons der zweiten verdeckten Schicht.*

Verschiebung. Eine Verschiebung einer Neuronenlinie der zweiten verdeckten Schicht wird, ähnlich wie die Drehung, durch das Verschieben aller Neuronenlinien der ersten verdeckten Schicht erreicht, welche zur Bildung der Klassentrennlinie des Neurons im Definitionsbereich beitragen.

Änderung des Beitrags zur Lösung. Genauso, wie das der Fall bei Neuronen der ersten verdeckten Schicht war, bestimmt die Stärke der Verbindung zur dritten Schicht den Beitrag eines Neurons der zweiten verdeckten Schicht zur Gesamtlösung. Durch die Änderung des Gewichts ändert sich auch dieser Beitrag.

Einfügen eines Neurons. Da ein Neuron der zweiten verdeckten Schicht eine komplexere Aufteilung des Eingaberaumes vornimmt und immer eine Kombination aus Aufteilungen der Neuronen der ersten verdeckten Schicht ist, ist sein Einfügen immer mit einer Auswahl und zumeist dem Einfügen von neuen Neuronen in der ersten verdeckten Schicht verbunden. Das Einfügen kann deswegen auf zwei verschiedene Arten erfolgen:

1. Beim Einfügen des Neurons werden die vorhandenen Neuronen der ersten verdeckten Schicht genutzt, um die Klassentrennlinie zu bilden. In diesem Fall müssen neue Gewichtsverbindungen zur ersten und zur dritten verdeckten Schicht geschaffen werden. Die Gewichte zu Neuronen der dritten verdeckten Schicht werden mit *0* initialisiert.
2. Das Neuron soll einen vorgegebenen Verlauf der Klassentrennlinie realisieren. Dazu müssen unter Umständen zusätzliche Neuronen in die erste verdeckte Schicht eingefügt werden, deren Kombination dann den gewünschten Verlauf ergibt. Die Parametrisierung der eingefügten Neuronen erfolgt entweder, indem sie separat so angelernt werden, daß dabei der gewünschte Verlauf entsteht, oder die benötigten Neuronen werden von Hand eingefügt. In beiden Fällen kann noch anschließend geprüft werden, ob sich bereits einige der benötigten Neuronen in der ersten verdeckten Schicht befinden.

5.1.1.3 Änderung eines Teils der Klassentrennlinie

Oft gestaltet sich die Manipulation eines neuronalen Netzes viel einfacher, wenn nicht die Neuronendarstellung, sondern die Klassentrennlinie manipuliert wird. Der Verlauf der Klassentrennlinie kann (1) durch eine Manipulation der vorhandenen und (2) durch das Einfügen neuer Neuronen verändert werden.

Die Verschiebung einer Klassentrennlinie parallel zur *0*- bzw. *1*-Linie eines Neuron, ist die erste Korrektur, die durch eine Manipulation vorhandener Neuronen realisiert werden kann.

Verschiebung parallel zur 0- und 1-Linie. Eine parallele Verschiebung kann auf zwei Arten erfolgen und zwar durch: (1) die Verschiebung des ganzen Neurons oder (2) eine Änderung der Steigung des linearen Überganges. Dadurch kann sich die Klassentrennlinie lediglich im Bereich zwischen der *0*- und der *1*-Linie verschieben. Das hat den Vorteil, daß sich diese Änderung nur lokal auswirkt, weil sie diese Korrektur nur auf den Bereich innerhalb des Sicherheitsstreifens bezieht.

Die erste Methode (Verschiebung des ganzen Neurons) wurde bereits in Kapitel 5.1.1.1 ausführlich beschrieben. Die zweite Methode (Verschiebung durch eine Änderung der Steigung des linearen Übergangs) kann entweder durch eine Änderung der Gewichte zur Eingabe- oder zu der zweiten verdeckten Schicht vorgenommen werden.

Als erstes wird die Verschiebung durch Änderung der Gewichte zur Eingabeschicht erläutert. In diesem Fall wird die Steigung des linearen Übergangs so manipuliert, daß die gewünschte Verschiebung stattfindet. Abbildung 5.7 zeigt die Idee. Um die Klassentrennlinie (einen Punkt mit der Höhe *ktw*) zu verschieben, wir die Steigung des Sicherheitsstreifens verändert (hier: flacher gemacht). Dadurch rutscht die Klassentrennlinie nach rechts. Soll die Klassentrennlinie nach links verschoben werden, dann muß die Steigung des Sicherheitsstreifen vergrößert werden.

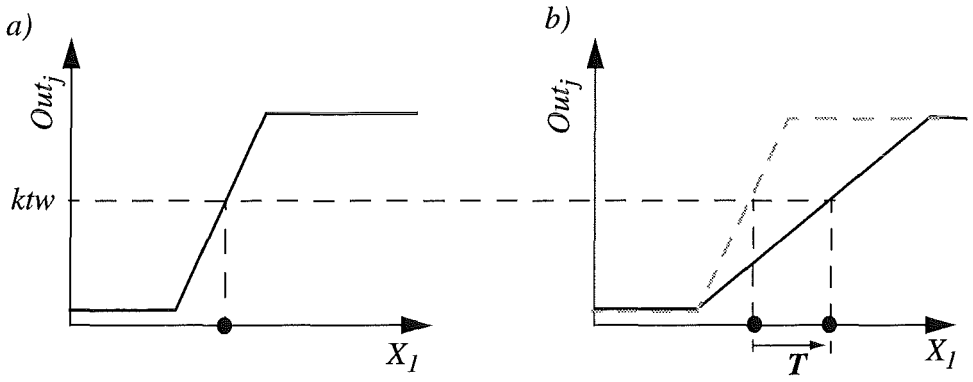


Abbildung 5.7: *Durch eine Änderung der Steigung des linearen Übergangs wird eine Verschiebung der Klassentrennlinie um den Vektor* ***T*** *erreicht. ktw ist der Klassentrennwert am Ausgabeneuron.*

Damit diese Manipulation graphisch erfolgen kann, muß eine Umrechnung der Trennlinienverschiebung in Änderungen der Netzparameter stattfinden. Die Änderungen der Gewichte w_{j1}, w_{j2} und des Bias können folgendermaßen berechnet werden:

$$w'_{j1} = \frac{4h_j}{\Delta x_1}$$

$$w'_{j2} = \frac{4h_j}{\Delta x_2}$$

$$Bias'_j = -2 - \frac{4h_j\left(\frac{1}{2} + Bias\right)}{w_{j1}\Delta x_1 + w_{j2}\Delta x_2}$$

mit:

Δx_1	Verschiebung der Klassentrennlinie in X_1-Richtung
Δx_2	Verschiebung der Klassentrennlinie in X_2-Richtung
h_j	Auf das manipulierte Neuron N_j umgerechneter Klassentrennwert
w_{j1}, w_{j2}	Gewichte zwischen Eingabeschicht und erster verdeckter Schicht vor der Manipulation
w'_{j1}, w'_{j2}	Gewichte zwischen Eingabeschicht und erster verdeckter Schicht nach der Manipulation
Bias	Bias-Wert vor der Manipulation
Bias'	Bias-Wert nach der Manipulation

Beispiel 5.2. Durch eine Änderung der Steigung des linearen Bereichs wurde ein Teil der Klassentrennlinie verschoben, welcher parallel zu der *0*- und *1*-Linie des Neurons N_1 verläuft. Damit die Manipulation lokal bleibt, wird das Neuron an der alten *1*-Linie ausgerichtet, d. h. die *0*-Linie wird weggezogen (s. Abbildung 5.8).
■

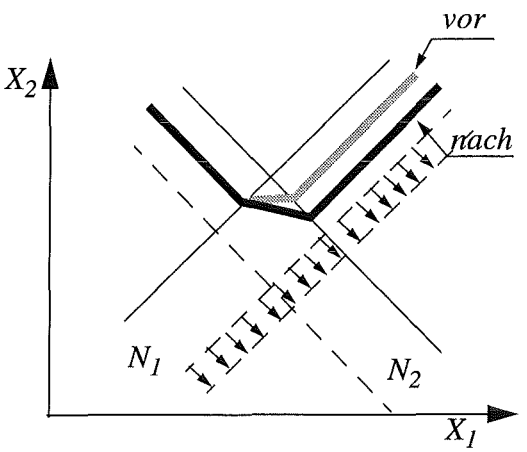


Abbildung 5.8: *Verschiebung eines Teils der Klassentrennlinie. Das Bild zeigt den Verlauf vor und nach der Manipulation.*

Eine andere Variante ergibt sich, wenn durch eine Änderung des Gewichts zu der zweiten verdeckten Schicht die Steigung des linearen Übergangs verändert wird. Eine Änderung des Gewichts zur zweiten verdeckten Schicht um den Faktor k staucht bzw. streckt die Aktivierungsfunktion des verdeckten Neurons der ersten verdeckten Schicht, ohne den Abstand zwischen der *0*- und *1*-Linie zu verändern (s. Abbildung 5.9). Dadurch verschiebt sich auch die Klassentrennlinie. Diese Verschiebung ist aber auf den Bereich zwischen der *0*- und der *1*-Linie beschränkt.
Der Faktor k läßt sich aus der Formel:

$$k = \frac{w_{j1}x_1 + w_{j2}x_2}{w_{j1}(x_1 + \Delta x_1) + w_{j2}(x_2 + \Delta x_2)}$$

berechnen. Die Gewichte sind dann gleich:

$$w'_{j1} = kw_{j1}$$
$$w'_{j2} = kw_{j2}.$$

Dabei sind Δx_1 und Δx_2 die Komponentenvektoren der Klassentrennlinienverschiebung in X_1- bzw. X_2-Richtung.

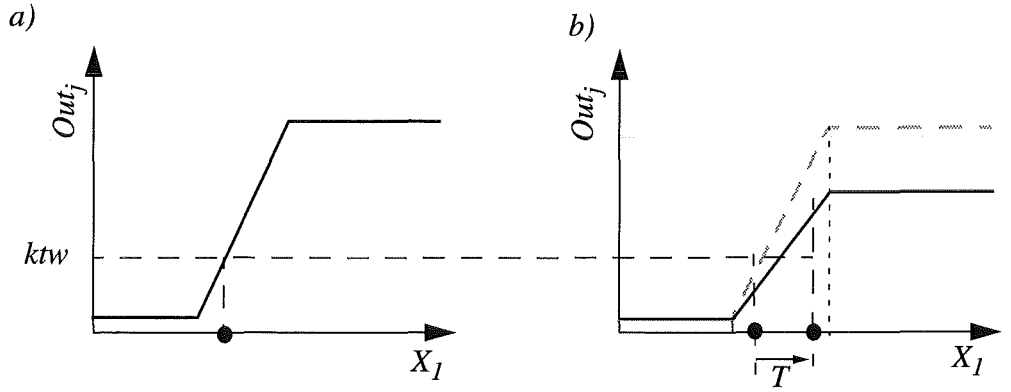


Abbildung 5.9: *Durch eine Änderung der Steigung des linearen Übergangs wird eine Verschiebung der Klassentrennlinie um den Vektor* **T** *erreicht.*

Als Nebeneffekt dieser Manipulation ergibt sich eine Änderung des Beitrags zur Gesamtlösung. Zwecks einer Kompensation müssen die Gewichte anderer Neuronen, die an derselben Klassentrennlinienbildung beteiligt sind, ebenfalls mit k multipliziert werden. Ansonsten verschiebt sich die Klassentrennlinie überall dort, wo die *1*-Hyperebene des manipulierten Neurons sie schneidet. Dies wird an dem folgenden Beispiel klar.

Beispiel 5.3. Der Klassentrennlinienverlauf in Abbildung 5.10 wurde durch eine Multiplikation des Gewichts w_{53} mit dem Faktor $k=3$ so verändert, daß jetzt der zur *0*- und *1*-Linie des Neurons N_3 parallele Klassentrennlinienabschnitt in der Darstellung nach unten (zur *0*-Linie) verschoben wurde. Ohne eine Zusatzkorrektur des Gewichts w_{54} um den Faktor k verschiebt sich auch der Teil, welcher parallel zur *0*- und *1*-Linie des Neurons N_4 verläuft.
■

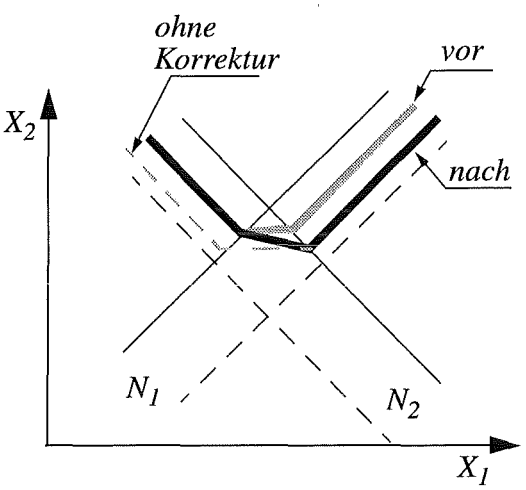


Abbildung 5.10: *Verschiebung eines Teils der Klassentrennlinie. Das Bild zeigt den Verlauf vor und nach der Manipulation sowie mit und ohne der Korrektur des Neurons N_2.*

Drehung eines Segments, das parallel zur 0- und 1-Linie verläuft. Um einen Teil der Klassentrennlinie zu drehen, der parallel zur *0*- und *1*-Linie eines Neurons verläuft, wird das betroffene Neuron der ersten verdeckten Schicht um einen festgelegten Punkt gedreht. Die Auswahl des Punktes hängt von der konkreten Problemstellung ab. In Abbildung 5.11 wurde der Abschnitt der Klassentrennlinie, welcher parallel zum Neuron N_1 verläuft, um den Punkt A gedreht. Die Operation erfolgt durch das Drehen des Neurons N_1 um den Koordinatenursprung und eine Verschiebung zum Punkt A.

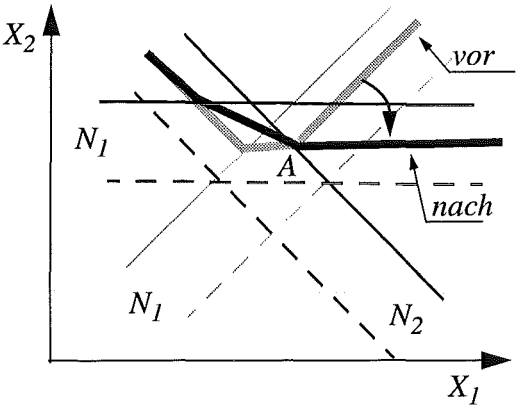


Abbildung 5.11: *Drehung eines Teils der Klassentrennlinie. Das Bild zeigt den Verlauf vor und nach der Manipulation.*

Verschiebung im Überlagerungsbereich. Eine Verschiebung der Klassentrennlinie im Überlagerungsbereich mehrerer Neuronen erfolgt, indem gleichzeitig alle an der Überlagerung beteiligten Neuronen manipuliert werden. Dabei ändert sich meistens auch der Verlauf der Klassentrennlinienteile, welche parallel zur *0*- und *1*-Linie der manipulierten Neuronen verlaufen. Bei der Berechnung werden zuerst die Schnittpunkte der Klassentrennlinie mit der *0*- und der *1*-Linie der beteiligten Neuronen ermittelt (in Abbildung 5.12 sind es die Punkte *A* und *B)*, und dann werden diese Punkte entlang der *0*- bzw. *1*-Linie verschoben, bis die gewünschte Verschiebung erfolgt. In Abbildung 5.12 wurde der Punkt *A* zum Punkt *A'* und der Punkt *B* zum Punkt *B'* verschoben. Dabei werden auch die Klassentrennlinienabschnitte der angrenzenden Neuronen – in Abbildung 5.12 die Neuronen N_1 und N_2 – verschoben.

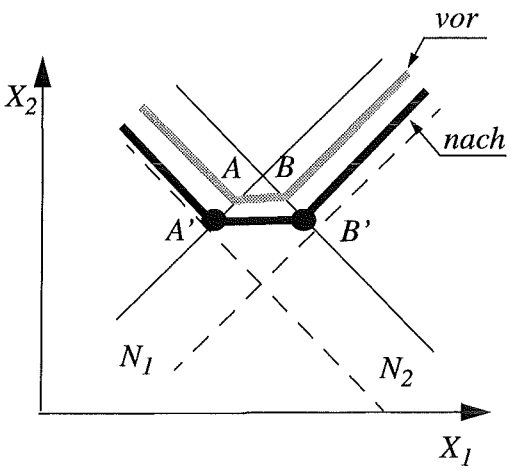


Abbildung 5.12: *Verschiebung eines Teils der Klassentrennlinie im Überlagerungsbereich. Das Bild zeigt den Verlauf vor und nach der Manipulation.*

Drehung im Überlagerungsbereich. Eine Drehung wird im Prinzip genauso wie eine Verschiebung realisiert. Der Unterschied liegt darin, daß jetzt die beiden Punkte *A'* und *B'* so plaziert werden müssen, daß sich daraus eine Drehung ergibt. Ein Beispiel für eine Drehung um den Punkt *B* zeigt Abbildung 5.13. Um diese Drehung zu vollziehen, wurde die Klassentrennlinie des angrenzenden Neurons N_2 soweit verschoben, bis sie durch den Punkt *A'* verläuft.

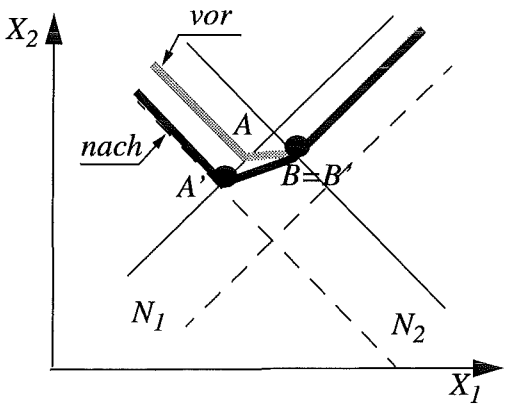


Abbildung 5.13: *Drehung eines Teils der Klassentrennlinie im Überlagerungsbereich. Das Bild zeigt den Verlauf vor und nach der Manipulation.*

5.1.1.4 Anwendung auf das Schachbrettproblem

Die Manipulationsmöglichkeiten werden anhand eines nichttrivialen Beispiels demonstriert.

Beispiel 5.4. Als Beispiel betrachten wir das Schachbrettproblem in Beispiel 4.2. Das Netz mit zwei Eingabe-, vier verdecktenNeuronen und einem Ausgabeneuron wurde in *2000* Zyklen mit Standard-Backpropagation trainiert. Als Ergebnis ergab sich eine Aufteilung des Eingaberaumes, wie sie in Abbildung 5.14 dargestellt ist.

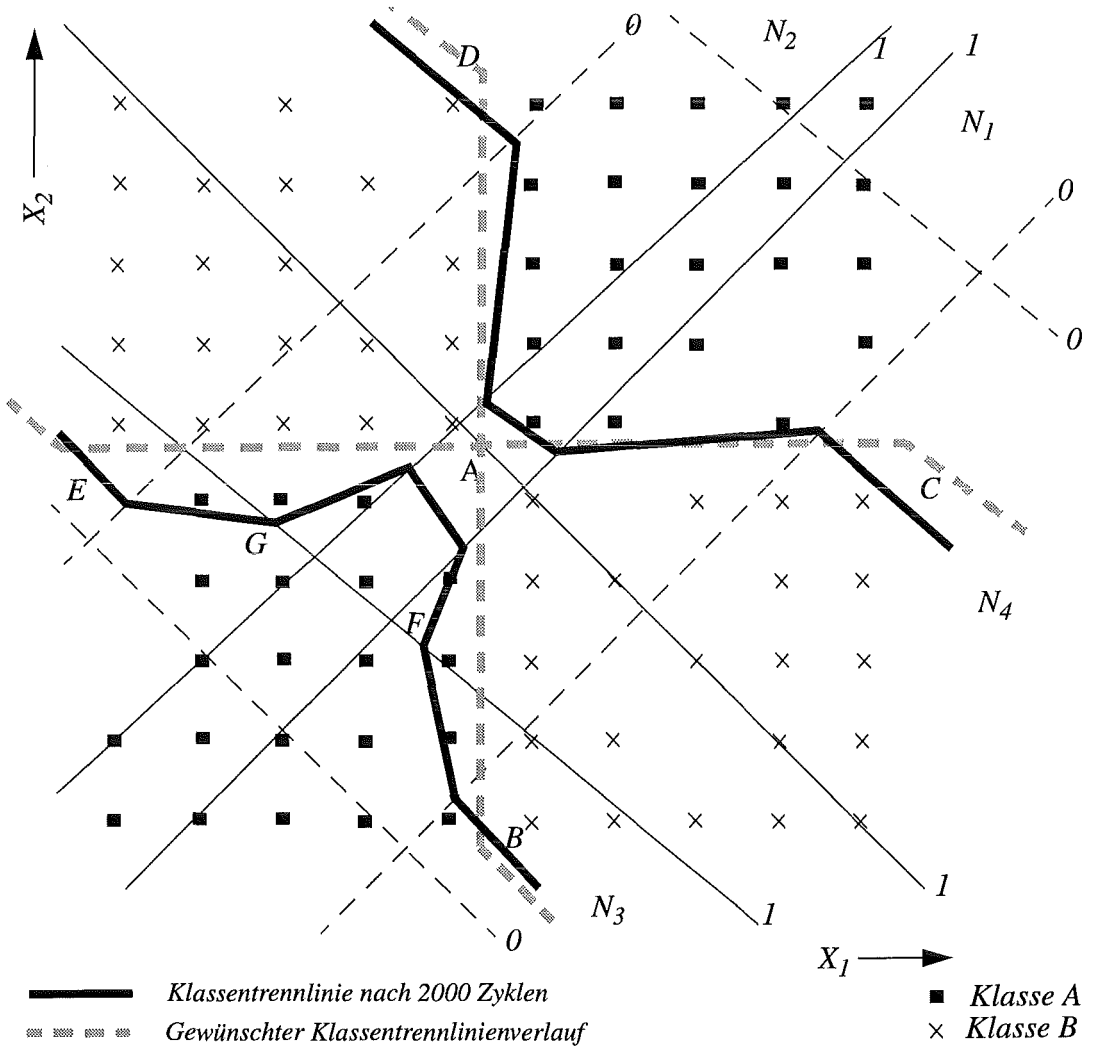


Abbildung 5.14: *Das neuronale Netz vor der Manipulation der Neuronen. Die Klassentrennlinie hat nicht die Form eines Schachbretts, und sie entspricht auch nicht unseren Vorstellungen von einer guten Lösung.*

Man erkennt sofort, daß der Verlauf der Klassentrennlinie nicht dem erwarteten entspricht. Es müssen folgende Änderungen vorgenommen werden: (1) die Klassifikation im Mittelpunkt des Bildes muß exakter werden; (2) die teilenden Bereiche zwischen den Feldern müssen gerade und länger gemacht werden; und (3) die Klassentrennlinie muß so verlaufen, daß alle von Netz falsch klassifizierten Muster in die richtige Klasse fallen. Die gestrichelte Linie in Abbildung 5.14 zeigt den gewünschten Verlauf. Er wurde durch folgende Manipulationen erreicht: (1) die Neuronen N_1 und N_2 werden soweit zueinander geschoben, bis der flache Bereich *A* verschwindet (vergl. Abbildung 5.14 und Abbildung 5.15); (2) die Klassentrennlinie wird in Bereichen, in denen sie parallel zu den Neuronen N_3 und N_4 verläuft, so verschoben (Bereiche *B, C, D, E*, Abbildung 5.14), daß die Klassenaufteilung die Form eines Schachbretts annimmt, und (3) die Bereiche *F* und *G* der Klassentrennlinie (Abbildung 5.14) werden so geändert, daß die falsch klassifizierten Muster in die richtige Klasse gelangen. Das Ergebnis der Manipulation zeigt Abbildung 5.15.

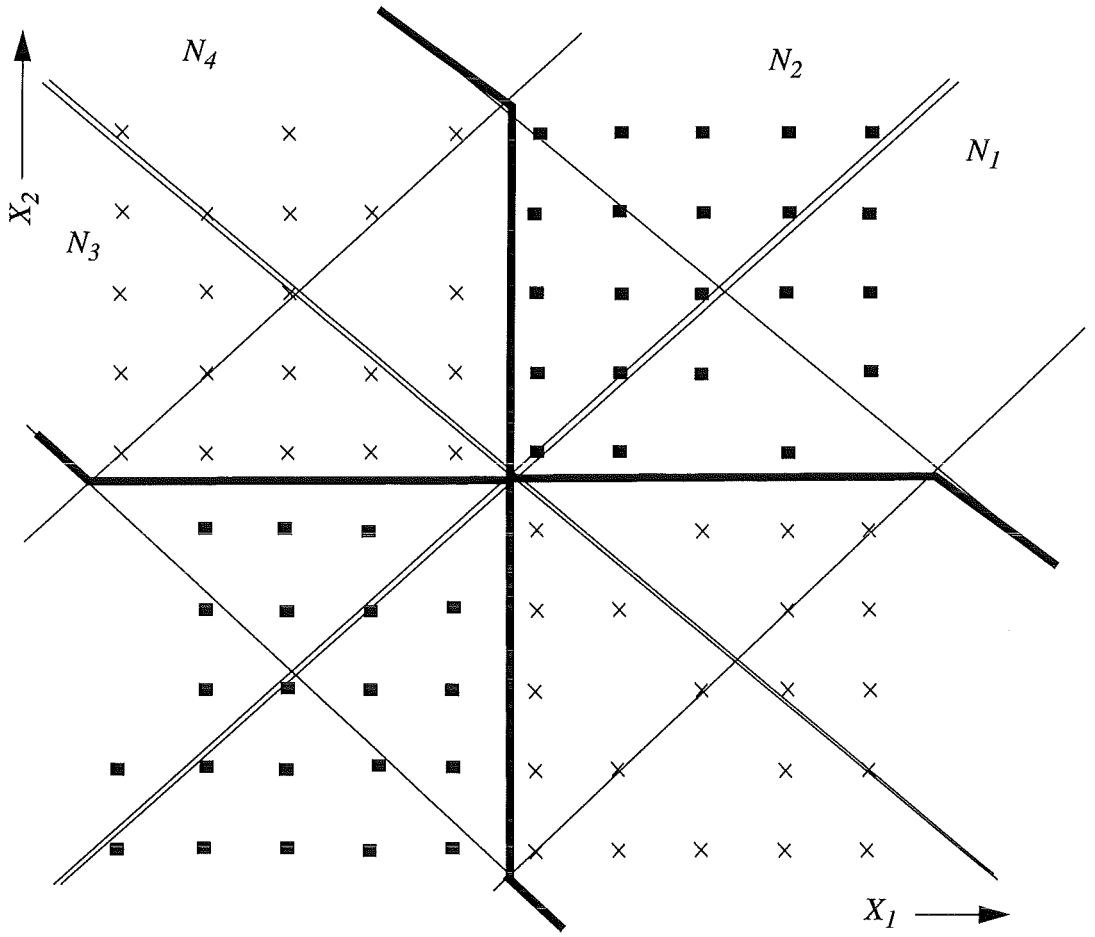


Abbildung 5.15: *Nach der Manipulation.*

■

5.1.2 Manipulation durch Teilnetzlernen

Bei dieser Manipulationsart werden Neuronen separat trainiert, welche im manipulierten Bereich des Definitionsbereiches liegen. Dabei werden aus dem gewünschten Verlauf der Klassentrennlinie Lernmuster generiert, die dann vom Teilnetz gelernt werden. Die Lösung muß am Ende mit der geometrischen Visualisierung kontrolliert werden. Dabei müssen ggf. kleine Ungenauigkeiten im Klassentrennlinienverlauf durch Neuronenmanipulation behoben werden. Der kritische Punkt ist das Einsetzen des Teilnetzes. Diese Methode findet ihren Einsatz vor allem dann, wenn grobe Änderungen des Klassentrennlinienverlaufs gewünscht werden, z. B. dann, wenn ein neuer Cluster eingefügt werden soll. Solche Änderungen lassen sich schwer durch die vorher beschriebenen Manipulationen der Klassentrennlinie erreichen. Besonders effizient ist diese Methode, wenn Netze mit zwei verdeckten Schichten manipuliert werden sollen. In solch einem Fall kann zuerst ein neues Teilnetz mit einer verdeckten Schicht so trainiert werden, daß es die geforderte Änderung lokal hervorruft und dann mit dem manipulierten Netz verbunden wird, indem ein zusätzliches Gewicht zum Ausgabeneuron eingefügt wird. Die lokale Wirkung kann dadurch erreicht werden, daß das Netz einen Beitrag nur in dem lokalen Bereich liefert (im übrigen Bereich liefert das Netz eine *0*).

Durch die Netzmanipulation kann die Leistung eines Netzes gesteigert werden, im Sinne einer qualitativ besseren Lösung. Ein anderer Aspekt ist die Frage, ob es unter den Neuronen sog. *überflüssige* gibt. Diese Frage kann durch eine Topologieoptimierung beantwortet werden.

5.2 Topologieoptimierung

Unter einer Topologieoptimierung wird in dieser Arbeit die Vereinfachung des Netzes durch das Entfernen von Neuronen oder ganzen Schichten verstanden.

Ein Neuron wird als überflüssig definiert und kann entfernt werden, wenn es keinen wesentlichen Beitrag zur Bildung der Klassentrennlinie im Definitionsbereich leistet. In Kapitel 4.1.6 wurde neben der Interpretation auch das Erkennen von überflüssigen Neuronen, welche außerhalb des Definitionsbereiches liegen und keinen Beitrag zur Klassentrennlinienbildung leisten, kurz beschrieben. In diesem Abschnitt werden genaue Regeln für das Aufspüren und Entfernen von überflüssigen Neuronen inner- und außerhalb des Definitionsbereichs beschrieben.

Bei der Topologieoptimierung wird meistens nach folgendem Schema vorgegangen: Zuerst wird ein Neuron gefunden, welches eine notwendige Bedingung für "überflüssig sein" erfüllt, und danach wird geprüft, ob das Neuron auch tatsächlich entfernt werden kann (hinreichende Bedingung).

5.2.1 Optimierung der ersten verdeckten Schicht

Bei der Suche nach überflüssigen Neuronen der ersten verdeckten Schicht müssen die Auswirkungen auf weitere Schichten aufs genaueste untersucht werden. Neuronen der ersten verdeckten Schicht teilen direkt den Eingaberaum. Das Entfernen eines Neurons dieser Schicht bringt in der Regel eine signifikante Vereinfachung der Topologie.

5.2.1.1 Aufspüren und Entfernen von überflüssigen Neuronen außerhalb des Definitionsbereiches

Neuronen, deren Sicherheitsstreifen außerhalb des Definitionsbereiches liegt, sind überflüssig, wenn deren Schnitt mit dem Definitionsbereich ein Polygonzug mit gleicher Höhe ist, d. h. im Definitionsbereich liegt nur der *0*- bzw. der *1*-Bereich des Neurons. Dies kann schon während der Visualisierung festgestellt werden, wenn über den Beitrag einzelner Neuronen zur Gesamtaktivierung Buch geführt wird. Ein überflüssiges Neuron ist dadurch charakterisiert, daß es einen konstanten Beitrag von *1* bzw. *0* leistet.

Das Entfernen eines Neurons kann unter Umständen eine Änderung des Klassentrennlinienverlaufes hervorrufen, da sich der Wert der Netzaktivierung der Neuronen der zweiten verdeckten Schicht um den konstanten Betrag des entfernten Neurons ändert. Um dem entgegenzuwirken, wird der Bias-Wert jedes Neurons der zweiten verdeckten Schicht um einen konstanten Beitrag verändert. Dieser Beitrag $\Delta Bias$ ist gleich:

$$\Delta Bias = w_{ij}c,$$

mit:

- w_{ij} Das Gewicht zwischen dem entfernten Neuron j und dem i-ten Neuron der zweiten Schicht und
- c: Konstanter Beitrag des entfernten Neurons (in der Regel *0* oder *1*).

Beispiel 5.5. Abbildung 5.16a zeigt die Visualisierung der ersten verdeckten Schicht mit drei verdeckten Neuronen. Das Neuron N_3 liegt außerhalb des Definitionsbereiches und leistet deswegen zur Netzaktivierung im Definitionsbereich nur einen konstanten Beitrag von *1*. Beim Entfernen dieses Neurons wird deswegen der Bias-Wert jedes Neurons der zweiten Schicht um den Betrag w_{j3} erhöht. Das Ergebnis ist in Abbildung 5.16b

zu sehen. Im Definitionsbereich ändert sich der Verlauf der Klassentrennlinie nach diesem Eingriff nicht.

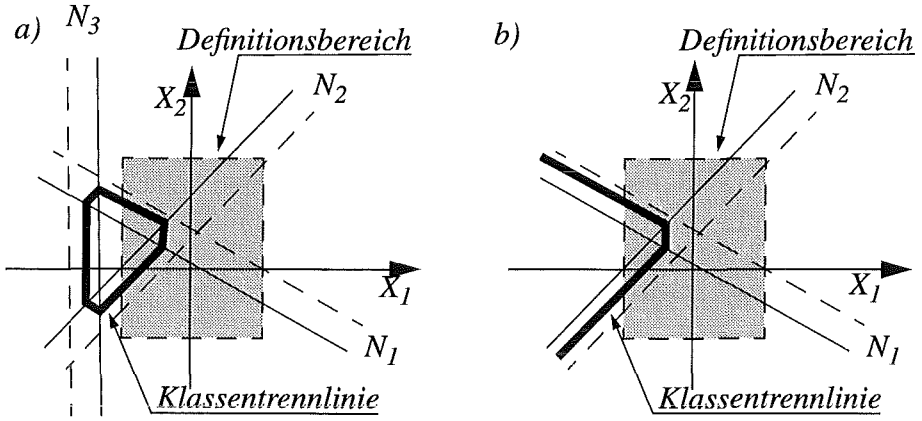


Abbildung 5.16: *Topologieoptimierung durch das Entfernen eines überflüssigen Neurons. Bild a) zeigt die Visualisierung vor und Bild b) nach der Optimierung.*

■

5.2.1.2 Aufspüren von Neuronen, die zwar keinen konstanten Beitrag zur Aktivierungsfunktion leisten, den Verlauf der Klassentrennlinie jedoch nicht ändern

Neuronen von diesem Typ können in zwei Kategorien aufteilt werden. Zum einen in Neuronen, deren Sicherheitsstreifen im Definitionsbereich die Klassentrennlinie nicht schneidet. Diese Neuronen lassen sich einfach entfernen. Zum anderen in Neuronen, deren Sicherheitsstreifen im Definitionsbereich sich mit der Klassentrennlinie schneidet. Um solche Neuronen zu entfernen, ist ein komplexerer Algorithmus notwendig.
Zuerst wird das Aufspüren und Entfernen von Neuronen beschrieben, deren Sicherheitsstreifen die Klassentrennlinie nicht schneidet. Diese Neuronen können aufgespürt werden, indem ihre Schnittpunkte – innerhalb des Definitionsbereiches – zwischen der *1*- bzw. *0*-Linie und der Klassentrennlinie bestimmt werden. Hat weder die *1*- noch die *0*-Linie einen Schnittpunkt mit der Klassentrennlinie, dann gehört das Neuron zu der gesuchten Klasse. Im Gegensatz zum obigen Fall müssen jetzt noch zusätzliche Tests durchgeführt werden, in denen der Einfluß des betrachteten Neurons auf beiden Seiten des Sicherheitsstreifens untersucht wird.

Je nach der Lage der *0*- und *1*-Linien wird ein Neuron auf unterschiedliche Weise aus dem Netz entfernt. Als erstes wenden wir uns dem Entfernen von Neuronen, zu deren Sicherheitsstreifen nur auf einer Seite der Klassentrennlinie liegt. Bei dieser Lage des Neurons können folgende Fälle auftreten:

Fall 1. Auf der anderen Seite des Sicherheitsstreifens liegen keine Neuronen, dann kann das Neuron genauso entfernt werden wie ein Neuron, das außerhalb des Definitionsbereichs liegt.

Beispiel 5.6. Die erste verdeckte Schicht eines neuronalen Netzes ist in Abbildung 5.17a visualisiert. Das Neuron N_3 ist ein überflüssiges Neuron, weil es nur auf einer Seite der Klassentrennlinie liegt und diese nicht schneidet. Es kann entfernt werden, indem zum Bias-Wert jedes Neurons der zweiten Schicht (welches auf der *1*-Halbebene des entfernten Neurons liegt) der entsprechende Beitrag, d. h. w_{j3}, addiert wird.

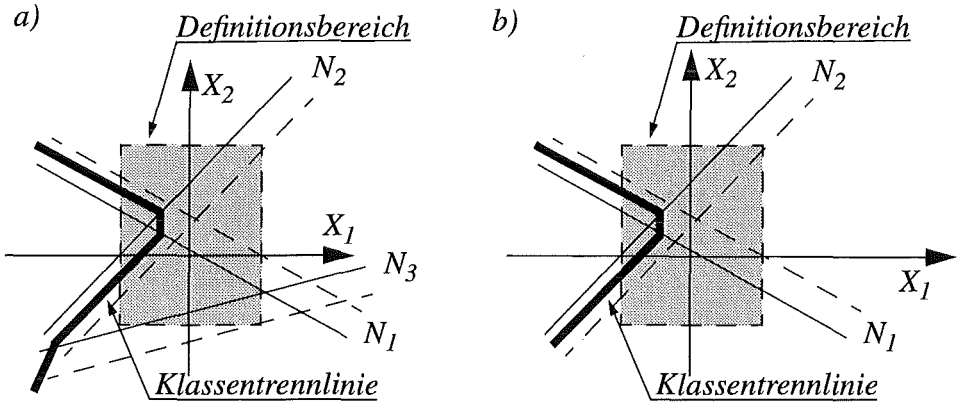


Abbildung 5.17: *Topologieoptimierung der ersten verdeckten Schicht. Bild a): Das Neuron N_3 liegt im Definitionsbereich, trägt aber dort nichts zur Bildung der Klassentrennlinie bei; Bild b): Nach dem Entfernen des Neurons N_3. Im Definitionsbereich hat sich die Klassenaufteilung nicht geändert, da eine Kompensation der 1-Halbebene durch Bias-Änderung erzwungen wurde.*

■

Fall 2. Auf der anderen Seite des Sicherheitsstreifens liegen weitere Neuronen. Es muß getestet werden, ob das Entfernen des Neurons die Bildung eines neuen Abschnitts der Klassentrennlinie nach sich zieht. Ein neuer Abschnitt könnte deswegen entstehen, weil das aufgespürte Neuron unterschiedliche Beiträge zur Gesamtaktivierung auf beiden Seiten des Sicherheitsstreifens leistet, solange es im Netz vorhanden ist. Dieser Unterschied resultiert aus den verschiedenen Beiträgen der *0*- und *1*-Halbebenen. Die Prüfung kann durch eine Neuberechnung der Visualisierung des betreffenden Bereiches erfolgen, nachdem das zu entfernende Neuron aus dem Netz temporär gelöscht wurde (alle Ausgangsgewichte des Neurons auf *0* setzen). Verändert sich dabei der Klassentrennlinienverlauf im Definitionsbereich nicht, dann kann das Neuron nach derselben Methode wie im Fall 1 aus dem Netz entfernt werden. Bildet sich dabei aber ein neuer Abschnitt der Klassentrennlinie, dann darf das Neuron nur dann entfernt werden, wenn der neue Teil der Klassentrennlinie durch eine Änderung des Bias-Wertes der Neuronen der zweiten verdeckten Schicht neutralisiert werden kann. Das ist nur dann der Fall, wenn die betroffenen Neuronen, die den neuen Teil der Klassentrennlinie erzeugen, sich nicht (im Definitionsbereich) mit dem Sicherheitsstreifens des aufgespürten Neurons schneiden.

Beispiel 5.7. Abbildung 5.18 und Abbildung 5.19 zeigen zwei Fälle, welche bei der Optimierung auftreten können. Im Netz aus Abbildung 5.18 wurde das Neuron N_3 als Kandidat für das Entfernen bestimmt; in Abbildung 5.19 das Neuron N_4. Im ersten Fall (Abbildung 5.18) kann das Neuron aus dem Netz entfernt werden; im zweiten Fall (Abbildung 5.19) aber nicht. Das liegt daran, daß bei der Entfernung des Neurons N_4 ein neuer Abschnitt der Klassentrennlinie entstehen würde, der nicht mehr durch eine Bias-Korrektur kompensiert werden könnte.

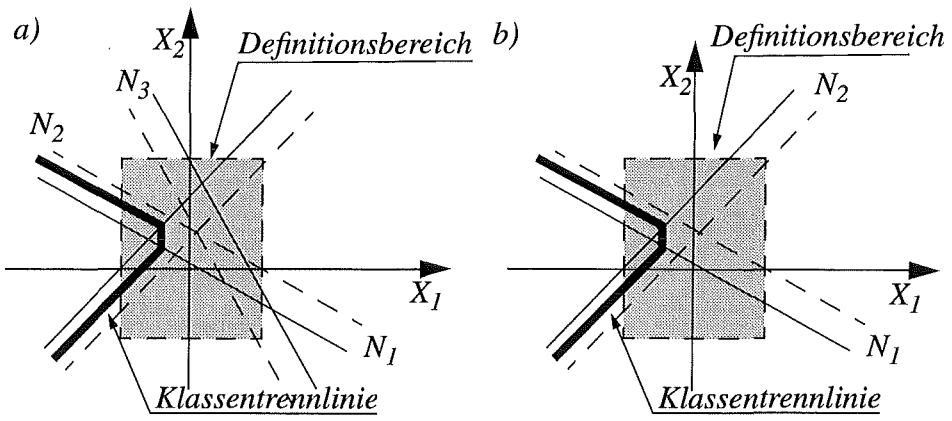


Abbildung 5.18: *Topologieoptimierung der ersten verdeckten Schicht möglich. Bild a): Das Neuron N_3 liegt im Definitionsbereich. Auf beiden Seiten seines Sicherheitsstreifens liegen im Definitionsbereich andere Neuronen. Bild b): Nach dem Entfernen des Neurons N_3.*

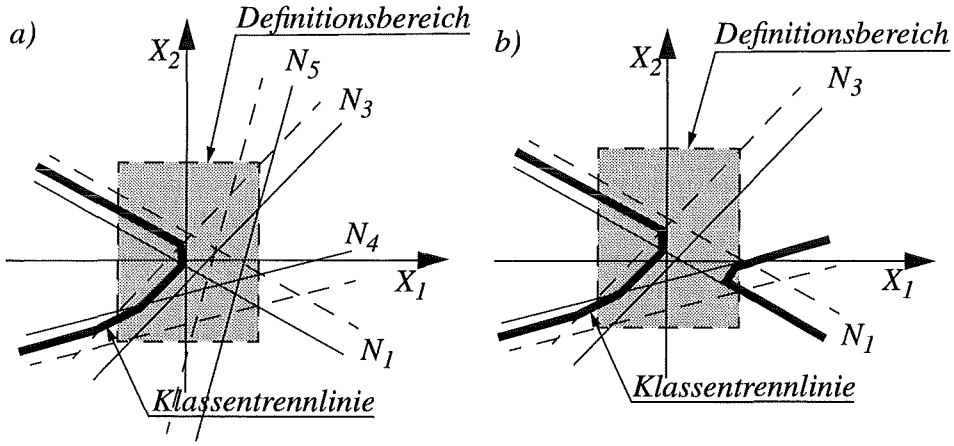


Abbildung 5.19: *Topologieoptimierung der ersten verdeckten Schicht nicht möglich. Bild a): Das Neuron N_5 liegt im Definitionsbereich. Auf beiden Seiten seines Sicherheitsstreifens liegen im Definitionsbereich andere Neuronen. Bild b): Nach dem Entfernen des Neurons N_5. Es entstand ein neuer Abschnitt der Klassentrennlinie, welcher durch eine Korrektur des Bias-Wertes nicht kompensiert werden kann. Das Neuron N_5 kann also nicht entfernt werden.*

■

Befindet sich die Klassentrennlinie auf beiden Seiten des Sicherheitsstreifens, dann ist eine Topologieoptimierung nur dann möglich, wenn die beiden Teile der Klassentrennlinie von verschiedenen Neuronen der zweiten verdeckten Schicht erzeugt werden.

Bei der Optimierung müssen folgende Fälle berücksichtigt werden:

Fall 1. Die Mengen der Neuronen, welche die Klassentrennlinie auf beiden Seiten des Sicherheitsstreifens bilden, sind disjunkt und schneiden sich nicht mit dem Sicherheitsstreifen. In diesem Fall kann durch eine unabhängige Änderung des Bias-Wertes der Neuronen der zweiten verdeckten Schicht das Entfernen eines Neuron kompensiert werden.

Beispiel 5.8. Aus dem neuronalen Netz aus Abbildung 5.20a wurde das Neuron N_5 entfernt. Das Ergebnis zeigt Abbildung 5.20b. Der Einfluß des entfernten Neurons N_5 wurde durch eine Bias-Korrektur der Neuronen N_6 und N_7 beseitigt.

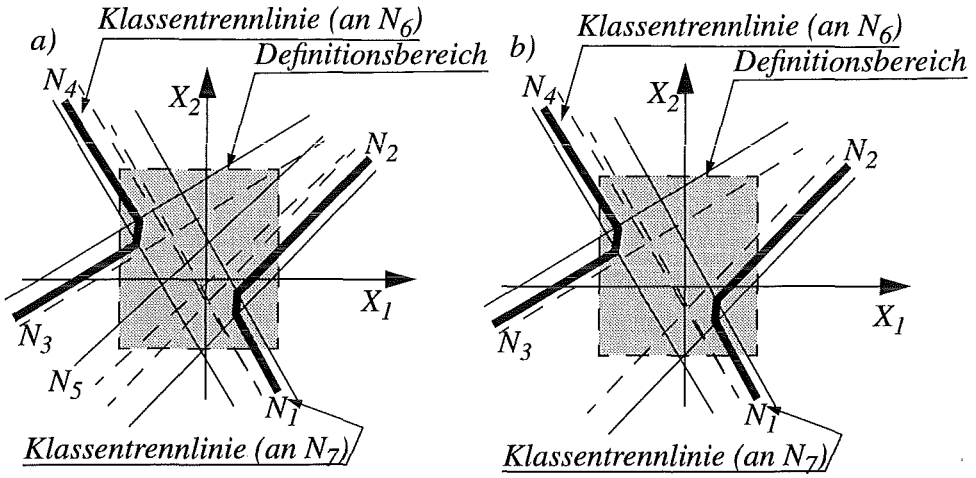


Abbildung 5.20: *Topologieoptimierung der ersten verdeckten Schicht. Bild a): Das Neuron N_5 liegt im Definitionsbereich zwischen zwei Abschnitten der Klassentrennlinie. Bild b): Visualisierung nach dem Entfernen des Neurons N_5.*

■

Fall 2. Die Menge der Neuronen, welche die Klassentrennlinie auf beiden Seiten des Sicherheitsstreifens bilden, sind nicht disjunkt. Ein Neuron darf nicht entfernt werden, da ansonsten die Änderung des Bias-Wertes eines Neurons (gemäß dem Beitrag des entfernten Neurons) auf einer Seite des Sicherheitsstreifens eine Änderung des Klassentrennlinienverlaufes auf der anderen Seite hinter sich ziehen würde.

5.2.1.3 Neuronen, welche die Klassentrennlinie schneiden, aber keinen Beitrag leisten

Solche Neuronen können nur dann keinen Beitrag zur Klassentrennlinie leisten, wenn: (1) die Gewichte zur nächsten Schicht gleich Null sind, oder (2) es andere Neuronen gibt, welche die Wirkung kompensieren.

Das Aufspüren im Fall 1 ist trivial. Im Fall 2 können die gesuchten Neuronen nur dann gefunden werden, wenn man berücksichtigt, daß die approximierten Neuronen – im Gegensatz zu logistischen Neuronen – eine abelsche Gruppe bilden.

5.2.2 Approximierte Neuronen der ersten verdeckten Schicht als abelsche Gruppe.

Auf einem beschränkten Definitionsbereich bilden die approximierten Neuronen eine abelsche Gruppe. Folgenden Tabelle zeigt die Eigenschaften und die Erfüllbarkeit durch logistische und approximierte Neuronen.

Eigenschaft	Logistische Neuronen	Approximierte Neuronen
Assoziativität: $\forall\ N_1, N_2, N_3 \in NN,$ gilt: $(N_1 + N_2) + N_3 = N_1 + (N_2 + N_3)$	ja	ja
Kommutativität: $\forall\ N_1, N_2 \in NN,$ gilt: $N_1 + N_2 = N_2 + N_1$	ja	ja
Existenz des Nullelements: $\exists\ 0 \in NN$, mit $N + 0 = N,\ \forall\ N \in NN$	nein	ja
Existenz eines inversen Elements: $\forall\ N \in NN,\ \exists\ -N \in NN$, so daß $N + (-N) = 0$	ja	ja

Für die Berechnung werden algebraische Operationen auf Neuronen durchgeführt, die dann als Ergebnis den Verlauf der Klassentrennlinie ergeben. Durch eine Zusammenfassung von Neuronen zu Nullneuronen können überflüssige Neuronen bestimmt werden.

5.2.2.4 Erkennung von ähnlichen Neuronen

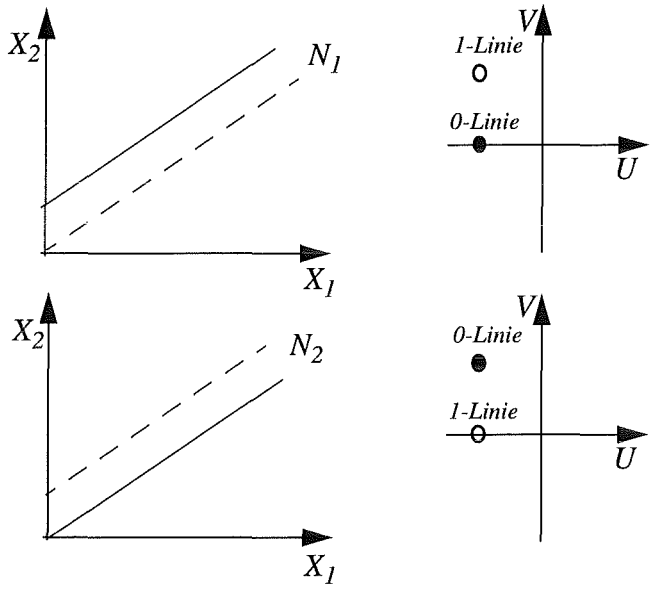
Im Sinne der obigen abelschen Gruppe sind Neuronen ähnlich, wenn sich deren geometrische Visualisierung überdeckt. Derartige Neuronen können sich – je nach den Gewichten zur zweiten verdeckten Schicht – aufheben, oder sie sind redundant. Die Erkennung von ähnlichen Neuronen läßt sich am einfachsten im LP-transformierten Raum durchführen. In diesem Raum sind alle Neuronen durch ihre *0*- und *1*-L-Punkte repräsentiert. Die notwendige Bedingung für ähnliche Neuronen ist die Deckung der L-Punkte. Die notwendige Bedingung für das Entfernen von ähnlichen Neuronen betrifft die Gewichte zu der zweiten verdeckten Schicht bzw. zum Ausgabeneuron. Bei gleichen Neuronen (*0*- und *1*-L-Punkte decken sich) müssen die Gewichte entgegengesetzte Vorzeichen haben, bei entgegengesetzten Neuronen gleiche. Dabei sind zwei Neuronen entgegengesetzt, wenn sich der *0*-L-Punkt eines Neurons mit dem *1*-L-Punkt des anderen Neurons überdeckt und umgekehrt.

5.2.2.5 Entfernen von ähnlichen Neuronen

Zu Nullneuronen zusammengefaßte Neuronen können ohne weiteres aus dem Netz entfernt werden. Beim Entfernen eines redundanten Neurons müssen alle Gewichte zur zweiter verdeckten Schicht auf das verbleibende Neuron umgelegt werden. Beim Entfernen eines gleichen Neurons werden die Gewichte des entfernten Neurons zu dem des verbleibenden aufaddiert und bei entgegengesetzten subtrahiert.

Beispiel 5.9. Im neuronalen Netz aus Abbildung 5.21 sind die Neuronen N_1 und N_2 ähnlich. Je nach dem Wert der Gewichte zum Ausgabeneuron können sie beide oder nur eines aus dem Netz entfernt werden. Bei der Entfernung nur eines Neurons müssen die Gewichte des verbleibenden um den Wert des Gewichtes des entfernten verringert werden.

Visualisierung *LP-Transformation* *Bedingungen für das Entfernen*



Gewicht zum Ausgabe-neuron j	**Aktion**
$w_{j1}=w_{j2}$	Beide Neuronen entfernen
sonst	Eines der beiden Neuronen kann entfernt werden.

Abbildung 5.21: *Zwei ähnliche und entgegengesetzte Neuronen. In der Visualisierung decken sich die 1- und 0-Linien und in der LP-Transformation die L-Punkte.*

■

In praktischen Anwendungen kommt es äußerst selten vor, daß es zwei oder mehrere Neuronen gibt, die sich ähnlich sind. Aus diesem Grunde kann anstelle einer vollständiger Gleichheit nur eine Ähnlichkeit (bis auf ein gewisses ε) gefordert werden. Um überflüssige Neuronen zu finden, wird dann die ε-Umgebung im LP-transformierten Raum auf ähnliche Neuronen durchsucht werden. Dabei kann die ε-Umgebung (je nach Bedarf) verschiedene Formen haben. Als ein anderes Maß an Ähnlichkeit kann aber auch die Korrelation von Neuronen im Definitionsbereich genommen werden. Dies erscheint deswegen als sinnvoll, weil dann Neuronen als ähnlich angesehen werden, wenn sie sich im Definitionsbereich fast decken, außerhalb aber auseinander liegen können.

5.2.2.6 Optimierung nach einer Projektion

Anstelle die Ähnlichkeit von Neuronen in allen Dimensionen zu betrachten, kann auch die Ähnlichkeit nach einer Projektion untersucht werden. Die Projektionsfläche kann dabei stückweise entlang der Klassentrennlinie verlaufen.

Beispiel 5.10. In Abbildung 5.22 ist ein neuronales Netz visualisiert, welches mit Hilfe von zwei Neuronen eine gerade Aufteilung des Definitionsbereiches durchführt (Bild a). Anhand einer Projektion und algebraischer Operationen auf den Neuronen ist erkennbar, daß die Summe der beiden Neuronen im Definitionsbereich und in die projizierte Richtung einen konstanten Wert hat. Die beiden Neuronen können also durch ein einziges Neuron ersetzt werden, welches dieselbe Aufteilung des Definitionsbereiches durchführt (Bild c).

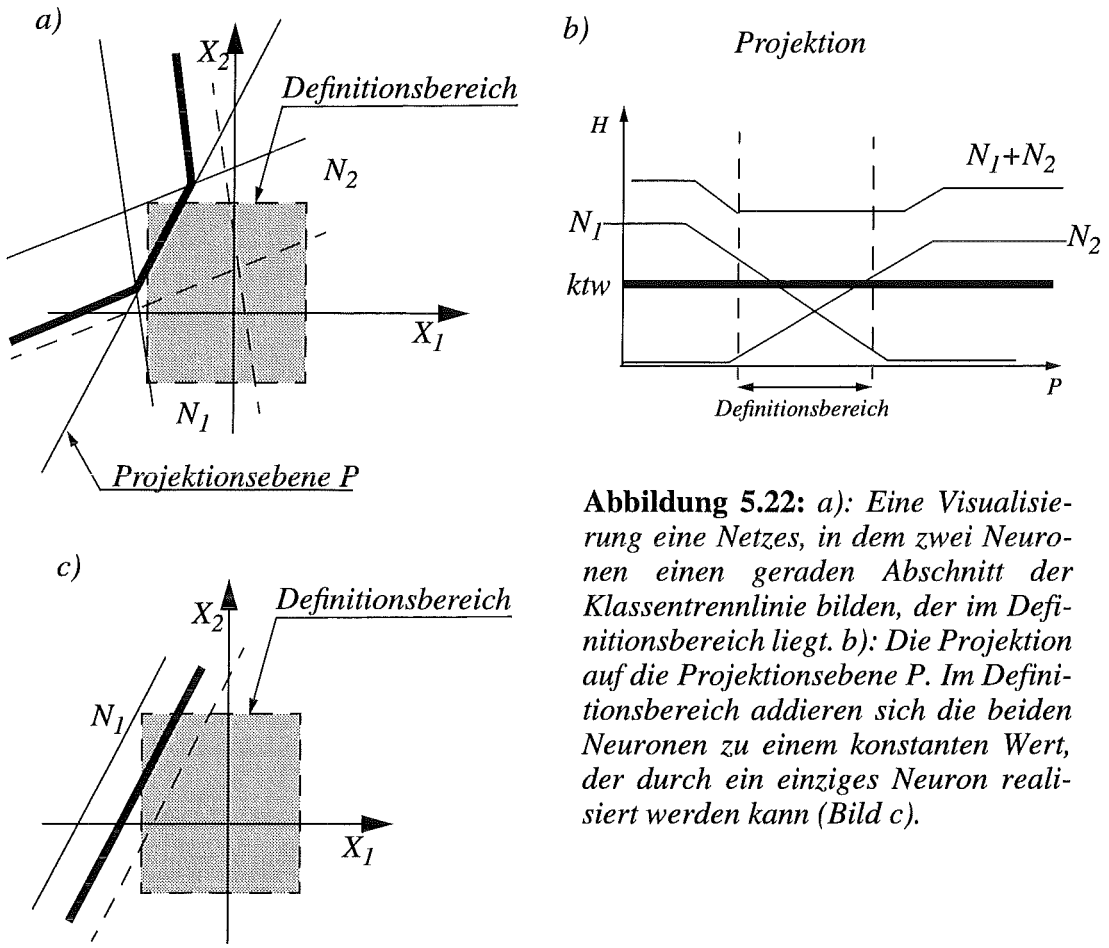


Abbildung 5.22: *a): Eine Visualisierung eine Netzes, in dem zwei Neuronen einen geraden Abschnitt der Klassentrennlinie bilden, der im Definitionsbereich liegt. b): Die Projektion auf die Projektionsebene P. Im Definitionsbereich addieren sich die beiden Neuronen zu einem konstanten Wert, der durch ein einziges Neuron realisiert werden kann (Bild c).*

■

Diese Methode findet Neuronen, welche die notwendige Bedingung für eine Reduktion erfüllen. Bevor ein Neuron gelöscht werden kann, muß allerdings mit den vorher beschriebenen Verfahren (Kapitel 5.2.1.1) geprüft werden, ob das Entfernen des Neurons neue Trennlinienabschnitte hinter sich bringt.

5.2.3 Optimierung der zweiten verdeckten Schicht

Bei der Optimierung der zweiten verdeckten Schicht wird im Prinzip genauso vorgegangen wie bei der Optimierung der ersten verdeckten Schicht. Der Unterschied liegt jetzt darin, daß anstelle von geraden Trennlinien in der Regel kompliziertere Polygonzüge auftreten.

5.3 Zusammenfassung

In diesem Kapitel wurden Methoden der Manipulation und der Optimierung neuronaler Netze vorgestellt, die eine wesentliche Verbesserung der Qualität einer neuronalen Lösung mit sich bringen. Die Manipulationen sind vor allem dafür geeignet, Justierungen der Klassentrennlinien vorzunehmen. Für größere Änderungen ist vor allem die Methode des Teilnetzetrainings geeignet.
Die Optimierungsverfahren bringen eine weitere Steigerung der Netzqualität, weil Neuronen, welche keinen Einfluß auf die Lösung haben oder überflüssig sind, aus dem Netz entfernt werden. Die beschriebenen Methoden der geometrischen Interpretation (GINN), der interaktiven geometrischen Manipulation (IGM) und die geometrischen Optimierungsverfahren (GONN) wurden in Form des Neuronale-Netze-Debuggers (NNDB) implementiert und getestet.

Kapitel 6

Neuronale-Netze-Debugger (NNDB)

Das primäre Ziel des NNDB war die Präsentation der Visualisierungs- und Manipulationsmöglichkeiten der geometrischen Interpretation sowie die Darstellung hochdimensionaler Daten. Die Visualisierung erfolgt so, wie sie in Kapitel 4 beschrieben wurde, die Manipulation und Optimierung nach den Methoden aus Kapitel 5. Die Daten werden mittels der LPCA auf zwei Dimensionen reduziert.

Damit der NNDB auf vielen Rechnerplattformen betrieben werden kann, wurde er in der objektorientierten und plattformunabhängigen Programmiersprache JAVA implementiert. Das Programm kann im Prinzip mit jedem Neuronale-Netze-Simulator arbeiten. Für die Anbindung ist lediglich die Bereitstellung eines Interfacemoduls notwendig, welches die internen Formate des Neuronale-Netze-Simulators in eine für den NNDB verständliche Form umwandelt. Der NNDB kann sowohl offline als auch online betrieben werden. Im offline-Betrieb müssen das Netz und die Trainings- bzw. Lernmuster in einer Datei vorliegen; im online-Betrieb werden die benötigten Daten durch eine Client-Server-Topologie vom Neuronale-Netze-Simulator übermittelt. Eine Animation des Lernvorganges kann dann ebenfalls erfolgen. Dazu wird in fest vorgegebenen Zeitabschnitten die Visualisierung neu berechnet. Die Bewegung der *0*- und *1*-Linien der Neuronendarstellung sowie die Formung der Klassentrennlinie wird dadurch sichtbar. Manipulation und Optimierung ist im offline-Betrieb oder im online-Betrieb möglich, nachdem der Netzsimulator angehalten wurde.

In folgenden Abschnitten werden die wichtigsten Merkmale und die Funktionalität des NNDB erläutert. Dabei wurde als Netzsimulator der SNNS (Stuttgarter-Neuronale-Netze-Simulator) gewählt. Wenn andere Simulatoren verwendet werden, dann sind – unter Umständen – einige der hier besprochenen Funktionen nicht verfügbar.

6.1 Anwendungsparadigma

Das Anwendungsparadigma zeigt die Abbildung 6.1. Der NNDB arbeitet parallel zum Neuronale-Netze-Simulator und zeigt online die Visualisierung des Netzes an. Er kann aber auch offline mit Netz- und Musterdateien arbeiten. Die Mensch-Maschine-Schnittstelle ist in eine Neuronale-Netz-Schnittstelle und in eine Lern- und Testdatenschnittstelle aufgeteilt.

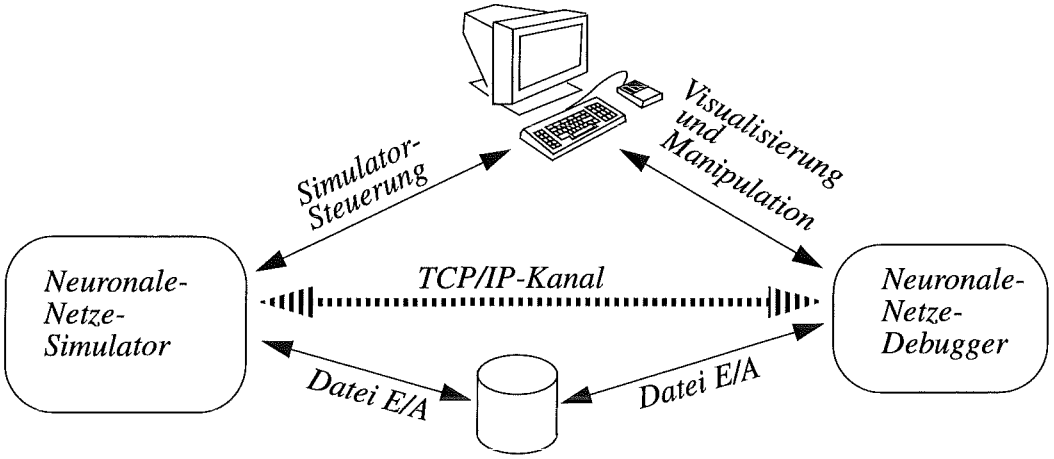


Abbildung 6.1: *Anwendungsparadigma für den Neuronale-Netze-Debugger NNDB.*

Neuronale-Netz-Schnittstelle. Über den Bildschirm und die Tastatur bzw. Maus kann sowohl der Simulator gesteuert als auch ein neuronales Netzes interpretiert und manipuliert werden. Das Netz kann entweder als Datei vorliegen oder über eine TCP/IP-Verbindung geladen werden. Wird das Netz mittels einer TCP/IP-Verbindung geladen, dann kann auch der Lernprozeß als eine Animation angezeigt werden. Dazu holt der Debugger alle t Millisekunden die Netzparameter aus dem Simulator und berechnet die Interpretation. Nach dem Anhalten der Animation kann das Netz manipuliert werden. Das korrigierte Netz wird dann zurück an den Simulator übermittelt (Datei oder TCP/IP-Kanal).

Lern- und Testdatenschnittstelle. Eine integrierte Datenbank (MSQL) und ein Kalkulationsprogramm (basierend auf dem UNIX-Programm gawk) bieten die Möglichkeit, Trainings- und Lerndaten zu bearbeiten. Der Benutzer kann Kriterien für die Auswahl der Daten angeben und zwischen verschiedenen Methoden der Crossvalidierung wählen. Diese Schnittstelle stellt auch die Daten mit Hilfe der LPCA dar.

Im folgenden werden die Neuronale-Netz-Schnittstelle und die Lern- und Testdatenschnittstelle etwas genauer beschrieben.

6.2 Die Neuronale-Netz-Schnittstellen

Die Neuronale-Netz-Schnittstelle hat die Aufgabe, das Netz zu visualisieren und eine Manipulation des Netzes zu ermöglichen. Abbildung 6.2 zeigt das Hauptfenster des NNDB. Im oberen Teil befinden sich zwei Karteimenüs, mit welchen die gewünschte Funktionalität auswählt werden kann. In der Mitte ist der Ausgabebereich, in dem das visualisierte Netz angezeigt wird. In der Fußzeile sind die Statusinformationen eingeblendet. Dort befindet sich auch die History-Liste, welche die Auswahl der zuletzt benutzten Projektionsparameter speichert. Der Benutzer wählt und manipuliert die Neuronen interaktiv mit der Maus. Manipuliert werden können: entweder ein Neuron, indem man seine Darstellung, d. h. der Verlauf der *0*- und der *1*-Linie, verändert oder die Klassentrennlinie, indem man ihren Verlauf korrigiert.

Die Parameter eines Neurons können in einem zusätzlichen Fenster angezeigt werden (rechts im Bild). Dieses Fenster enthält alle Gewichte und den Bias-Wert sowie zusätzlich eine Information "Optimizable", die angibt, ob das Neuron vom NNDB als überflüssig erkannt wurde. Welche Untersuchungen dabei durchgeführt werden, hängt von

Optimierungsoptionen ab. Im einfachsten Fall wird nur untersucht, ob das Neuron einen konstanten Beitrag im Definitionsbereich leistet (Testlevel 1). Diese Option ist auch (zeitlich gesehen) am schnellsten.

Nach der Manipulation kann das Netz entweder in einer Datei abgespeichert oder über einen TCP/IP-Kanal an den Simulator gesendet werden.

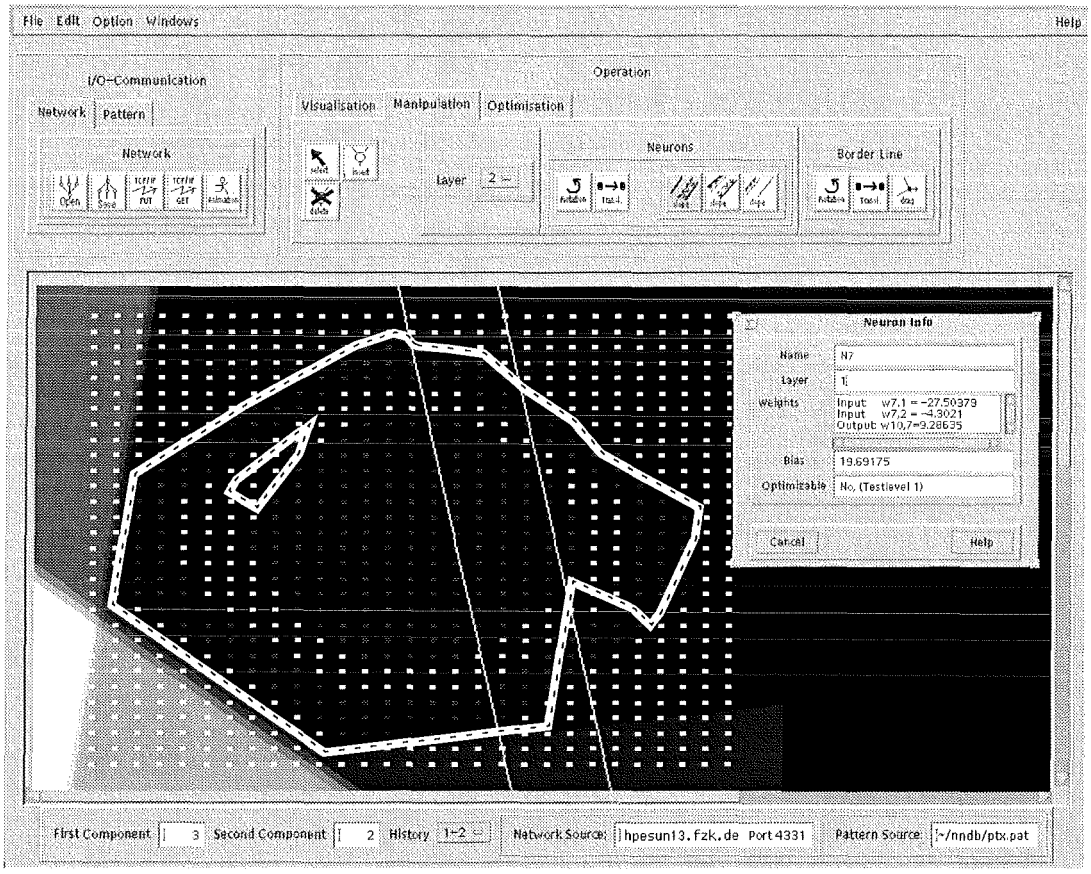


Abbildung 6.2: *Ansicht des NNDB während der Manipulation des Neurons N_7. Die Parameter des manipulierten Neurons sind in einem separaten Fenster dargestellt.*

Genauigkeit der Approximation. Die Genauigkeit der Approximation eines Neurons kann beliebig verfeinert werden, indem neue Interpolationspunkte entlang der logistischen Funktion eingefügt werden. Der erreichte Fehler wird im Fenster als "Error" angezeigt (s. Abbildung 6.3). Im Normalfall wird ein Neuron nur durch drei lineare Abschnitte dargestellt (s. Abbildung 4.3). Wenn aber eine genauere Approximation nötig ist, kann sie durch das Einfügen zusätzlicher Stützpunkte erreicht werden. Das ist vor allem bei einer quantitativen Qualitätsbeurteilung wichtig. In Abbildung 6.3 ist die Approximationsfunktion mit 7 Stützpunkten dargestellt.

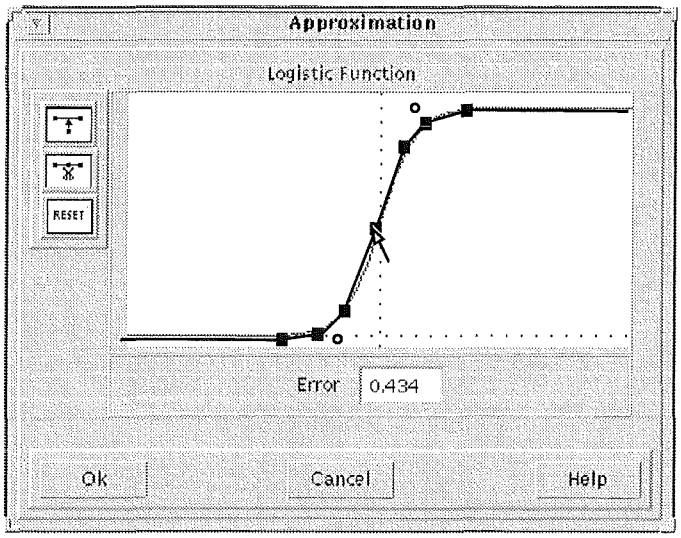


Abbildung 6.3: *In dem Fenster "Approximation" kann die Anzahl der Stützpunkte graphisch eingegeben werden. Die zwei kleine Kreise geben die Position der 0- und der 1-Linie an. Sie werden auch dann eingeblendet, wenn die Approximation durch mehrere lineare Abschnitte verfeinert wurde.*

Das neuronale Netz wird auch bei Projektionen immer in die Präsentation der Daten eingeblendet, d. h. die Richtung der Projektion gibt in den meisten Fällen die Datenverteilung vor. Sie wird von der Lern- und Testdatenschnittstelle berechnet.

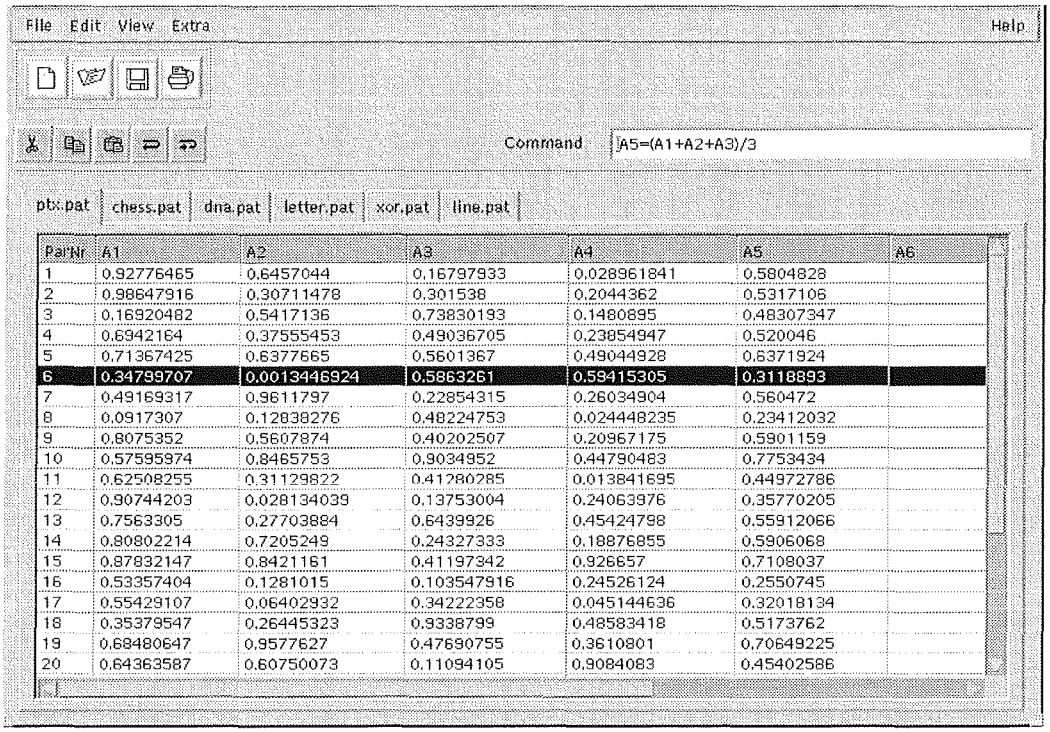
6.3 Die Lern- und Testdatenschnittstelle

Mit der Lern- und Testdatenschnittstelle können die Daten aus der Simulation bearbeitet werden. Zum einen ist es möglich, die Daten in einem Tabellenkalkulationsprogramm zu bearbeiten und zum anderen können die Daten unter Verwendung der lokalen PCA-Transformation dargestellt werden.

6.3.1 Datenverwaltung

Die Datenverwaltung erfolgt in Tabellen (s. Abbildung 6.4). In diesen Tabellen wird jedes Muster in einer Zeile abgespeichert. Durch Operationen auf den Spalten können neue Merkmale gebildet werden, z. B. ist in Abbildung 6.4 die Spalte *A5* der Mittelwert der Spalten *A1*, *A2* und *A3*. Neben der Zeilenoperationen können auch statistische Werte der Spalten (z. B. Mittelwert, Standardabweichung usw.) sowie gemischte Parameter (z. B. Korrelation, Kovarianz usw.) berechnet und graphisch angezeigt werden.

Die Muster, welche später als Daten für ein neuronales Netz verwendet werden, können aus dieser Tabelle, durch das Markieren von Spalten (Merkmale) und Zeilen (Muster), ausgewählt und zu einer neuen Datei zusammengefügt werden.



PatNr	A1	A2	A3	A4	A5	A6
1	0.92776465	0.6457044	0.16797933	0.028961841	0.5804828	
2	0.98647916	0.30711478	0.301538	0.2044362	0.5317106	
3	0.16920482	0.5417136	0.73830193	0.1480895	0.48307347	
4	0.6942164	0.37555453	0.49036705	0.23854947	0.520046	
5	0.71367425	0.6377665	0.5601367	0.49044928	0.6371924	
6	0.34799707	0.0013446924	0.58632261	0.59415305	0.3118893	
7	0.49169317	0.9611797	0.22854315	0.26034904	0.560472	
8	0.0917307	0.12838276	0.48224753	0.024448235	0.23412032	
9	0.8075352	0.5607874	0.40202507	0.20967175	0.5901159	
10	0.57595974	0.8465753	0.9034952	0.44790483	0.7753434	
11	0.62508255	0.31129822	0.41280285	0.013841695	0.44972786	
12	0.90744203	0.028134039	0.13753004	0.24063976	0.35770205	
13	0.7563305	0.27703884	0.6439926	0.45424798	0.55912066	
14	0.80802214	0.7205249	0.24327333	0.18876855	0.5906068	
15	0.87832147	0.8421161	0.41197342	0.926657	0.7108037	
16	0.53357404	0.1281015	0.103547916	0.24526124	0.2550745	
17	0.55429107	0.06402932	0.34222358	0.045144636	0.32018134	
18	0.35379547	0.26445323	0.9338799	0.48583418	0.5173762	
19	0.68480647	0.9577627	0.47690755	0.3610801	0.70649225	
20	0.64363587	0.60750073	0.11094105	0.9084083	0.45402586	

Abbildung 6.4: *Die tabellarische Verwaltung der Lern- und Trainingsdaten.*

6.3.2 Datenvisualisierung

Die Datenvisualisierung stellt die Daten graphisch dar. Da die Ausgabegeräte in der Regel nur zweidimensionale Bilder anzeigen können, ist eine Reduktion der Dimension notwendig. Im NNDB erfolgt die Dimensionenreduktion durch eine Projektion. Damit aber komplexe Datensätze übersichtlich visualisiert werden können, müssen sie vorher meistens transformiert werden. Der NNDB benutzt die lokale PCA-Transformation (s. Kapitel 5 und Abbildung 6.5). Im markierten Bereich überlagern sich die Muster derart, daß eine Beurteilung unmöglich ist. Deswegen wurde dieser Bereich im anderen PCA-Fenster dargestellt (s. Abbildung 6.6). In diesem Fenster (Window *w2*) ist eine gute Separierung möglich.

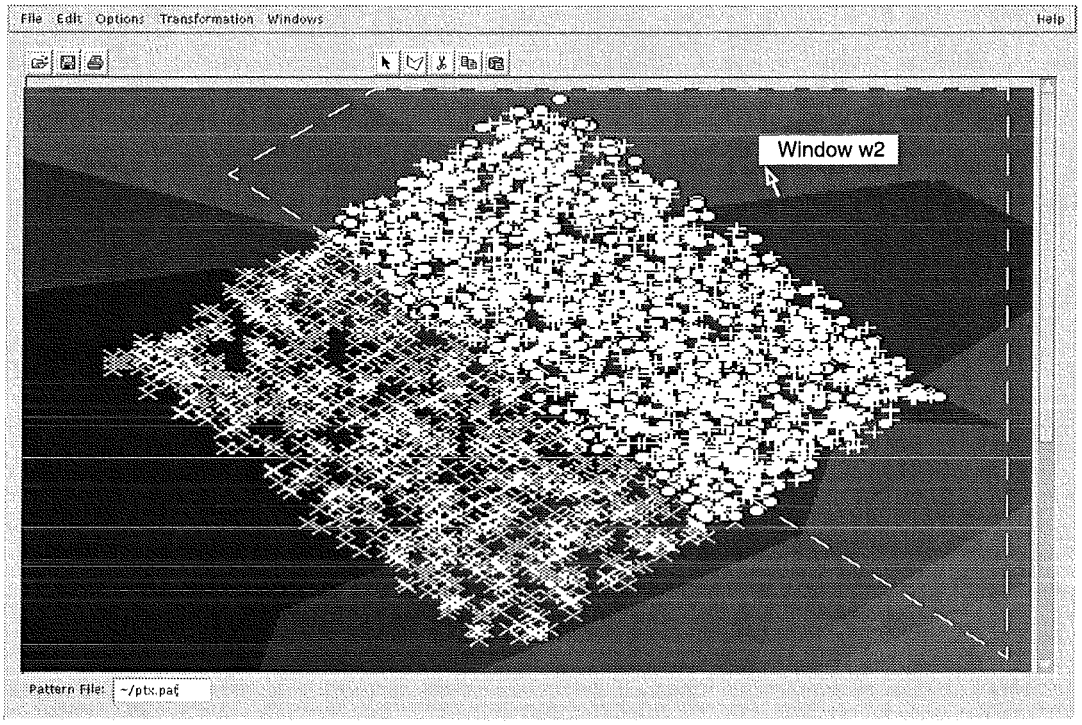


Abbildung 6.5: *Visualisierung von Daten. Der eingerahmte Bereich ist aus dem Blickpunkt der größten Varianz schlecht interpretierbar. Er wurde separat mit PCA transformiert. Das Ergebnis zeigt Abbildung 6.6.*

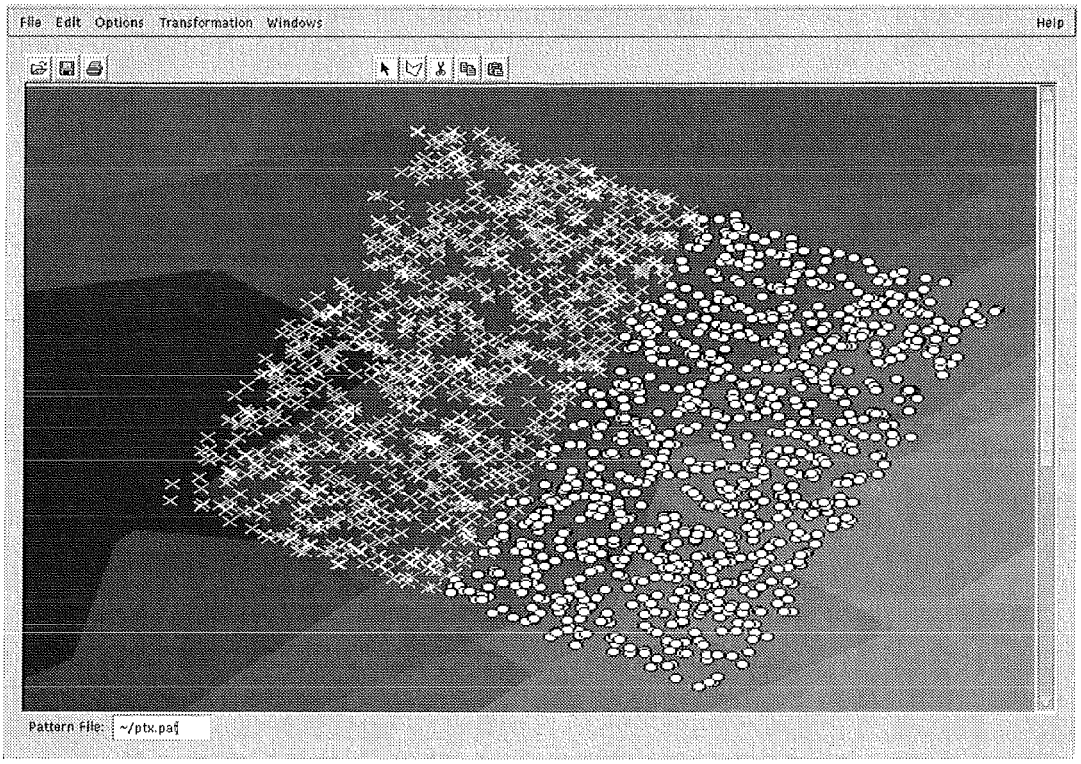


Abbildung 6.6: *Das in Abbildung 6.5 ausgewählte Fenster (Window w2) separat transformiert.*

6.4 Das Brustkrebs-Benchmark

Das Brustkrebs-Benchmark stammen aus der Sammlung "proben1", die vom Dr. L. Prechelt an der Universität Karlsruhe zur Verfügung gestellt wurde (Internetadresse: http://wwwipd.ira.uka.de/~prechelt).

Diese Daten enthalten eine Sammlung von Brustkrebsbeschreibungen, die anhand einer Mikroskopinspektion gemacht wurden. Jede Beschreibung besteht aus *9* Merkmalen. Die Ausgabe klassifiziert diese Muster in harmlose bzw. bösartige Tumore.

In Abbildung 6.7 ist die Visualisierung der Brustkrebsdaten und eines neuronalen Netzes mit *10* verdeckten Neuronen in der ersten verdeckten Schicht dargestellt. In dieser Klassifikation gibt es mehrere Fehlklassifikationen, d. h. bösartige Tumoren wurden als

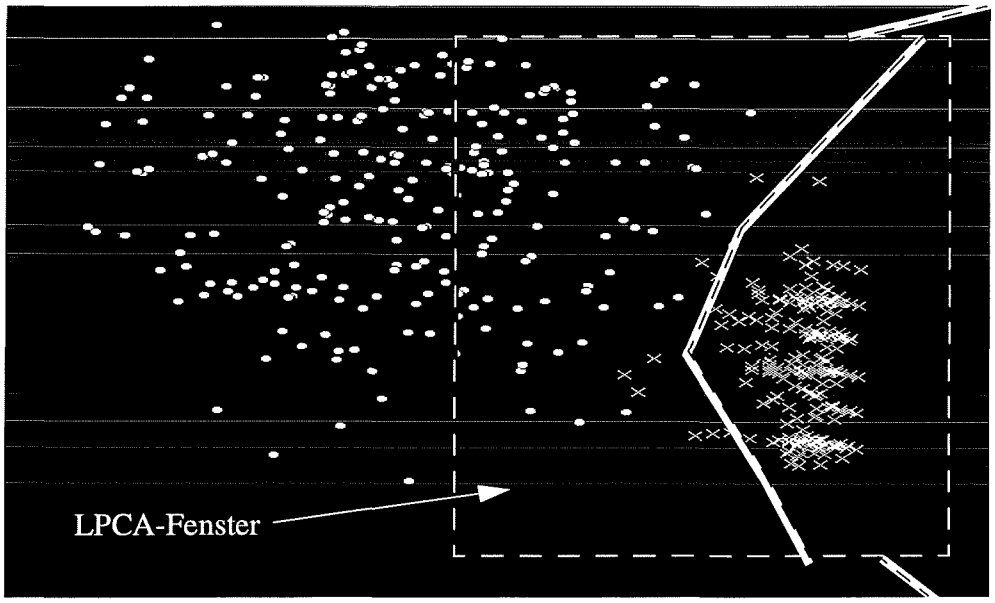


Abbildung 6.7: *Visualisierung der cancer1-Daten und eines neuronalen Netzes, das diese Daten separiert. Das LPCA-Fenster wurde dort angelegt, wo die PCA-Transformation keine gute Darstellung lieferte (vergl. Abbildung 6.8). Mit Kreisen sind bösartige und mit Kreuzen harmlose Tumore gekennzeichnet.*

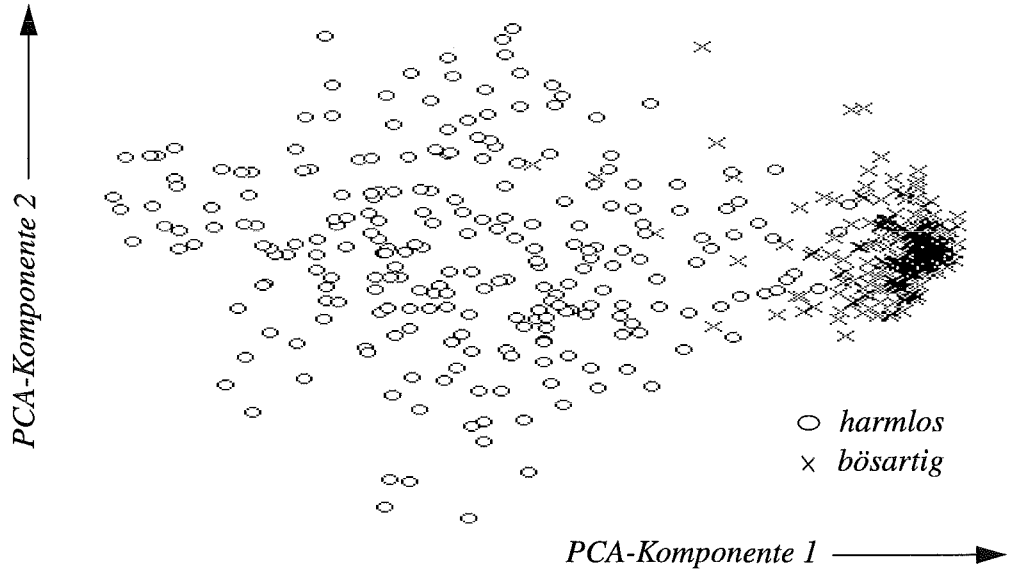


Abbildung 6.8: *Zum Vergleich: Visualisierung mit PCA-Transformation.*

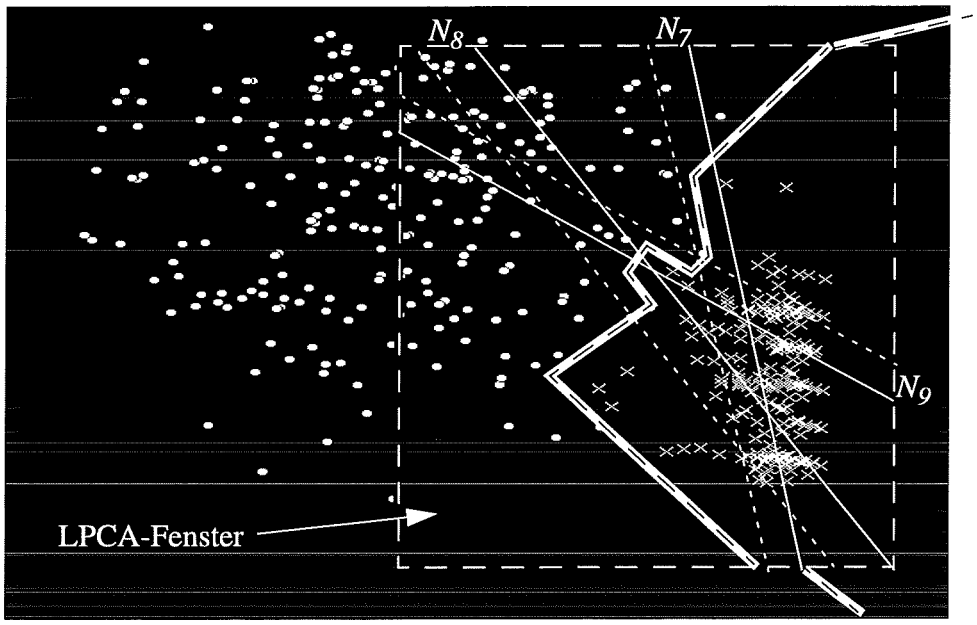


Abbildung 6.9: *Visualisierung der cancer1-Trainingsdaten und eines neuronalen Netzes nach der Manipulation und einer Darstellung durch die LPCA.*

harmlos klassifiziert. Diese Fehler sind für die Anwendung kritisch. Durch eine Manipulation des Netzes wurden die falsch klassifizierten Muster beseitigt. Um den gewünschten Verlauf der Klassentrennlinie zu erreichen, war es nötig, drei der Neuronen, welche vom Lernalgorithmus außerhalb des Definitionsbereiches geschoben wurden, in den Definitionsbereich zu versetzen und entsprechend anzuordnen (s. Abbildung 6.9). Diese Neuronen wurden so angeordnet, daß sie die Klassentrennlinie nur dort berühren, wo ihr Verlauf korrigiert werden sollte. Um die Qualität der Lösung nach der Manipulation zu testen, wurde ein Testdatensatz verwendet. Das Ergebnis zeigt die Abbildung 6.10. Die Testdaten wurden korrekt klassifiziert (bis auf einen bösartigen Tumor und acht harmlose Tumore). Ob die Ausreißer von Bedeutung sind, muß noch ein Experte beurteilen. Falls ja, dann kann er sie in die korrekte Klasse verschieben, indem er das Netz weiter manipuliert. Anhand der vorliegenden Daten könnte man aber vermuten, daß dieser Meßpunkt in den Testdaten falsch klassifiziert wurde, da die Lerndaten in diesem Bereich viele harmlose Tumore beinhalten (vergleiche: Abbildung 6.7).

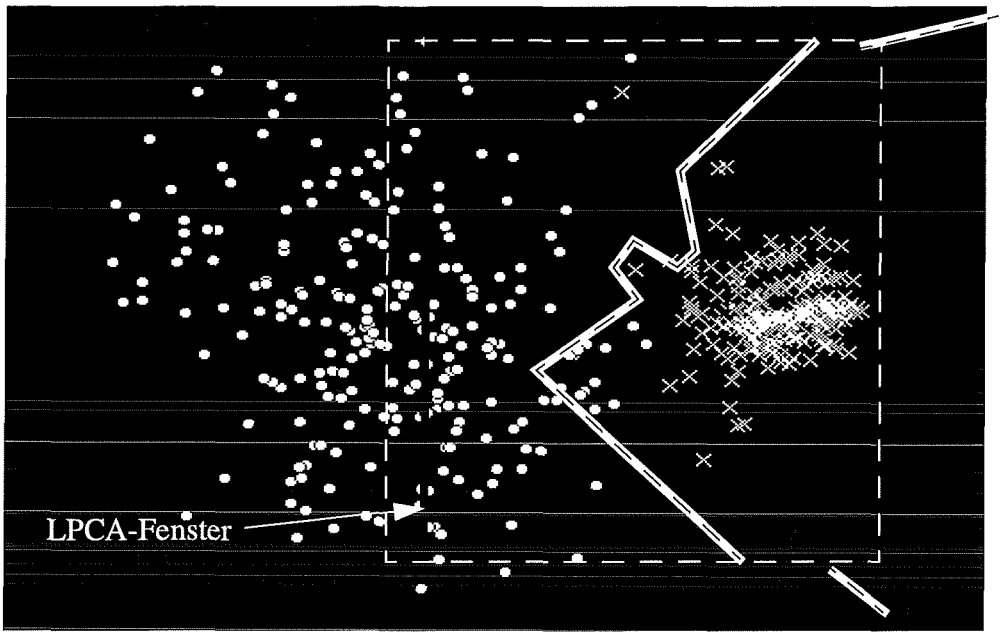


Abbildung 6.10: *Visualisierung der cancer1-Testdaten und eines neuronalen Netzes nach der Manipulation und einer Darstellung durch eine LPCA.*

6.5 Anwendung bei der Gasanalyse

Im Rahmen einer Gasanalyse wird am Forschungszentrum Karlsruhe eine künstliche Nase entwickelt. Das Meßsystem besteht aus acht Sensoren, von denen jeder die Konzentration eines unbekannten Gases mißt. Das System soll benutzt werden, um die Konzentration von drei bestimmten Gasen zu ermitteln, und zwar von: Benzol, Xylol und Oktan. Die Mustererkennung soll mit Hilfe eines neuronalen Netzes erfolgen, welches die Meßwerte der acht Sensoren auf die Konzentration der drei Gase abbildet. Das Problem bei der Entwicklung dieses Netzes ist, daß trotz der linearen Kennlinien der Sensoren das Netz nicht lernen wollte (der MSE blieb immer sehr hoch), obwohl sehr viele Netztopologien ausprobiert wurden. Um den Fehler zu fixieren, wurde der NNDB eingesetzt. Die Meßdaten wurden ohne das Netz visualisiert (s. Abbildung 6.11). Jeder Meßpunkt steht für eine bestimmte Konzentration von Benzol, Xylol und Oktan (die prozentualen Konzentrationen sind in den Klammern angegeben, z. B. bedeutet *(80, 5,0)* eine Konzentration von *80%* Benzol, *5%* Xylol und *0%* Oktan). Die gestrichelten Linien symbolisieren einen Meßvorgang. Nach einer Analyse der Visualisierung stellte sich heraus, daß, obwohl die acht Sensoren lineare Charakteristik haben, die erzielten Meßergebnisse nicht so angeordnet sind, wie man es erwarten würde, z. B. liegt die Konzentration *(100, 0, 33)* unter der von *(94, 40, 0)* und über der von *(64,0,15)*. Die Lage der Meßpunkte, bezogen auf die Konzentrationen der gemessenen Gase, ist also nichtlinear, obwohl sie es innerhalb einer Messung ist (was die gestrichelten Linien beweisen). Diese Erkenntnis führte dazu, daß die Merkmalsextraktion und die Konfiguration der Ausgabeneuronen geändert wurde. Zusätzlich wurde die Aufnahme neuer Daten veranlaßt, da die vorhandenen nicht ausreichend waren.

Das Beispiel zeigt, daß oft nur die Betrachtung der Meßdaten, die in einer klaren Form dargestellt werden, Schlüsse auf die Funktionsweise des Netzes erlauben.

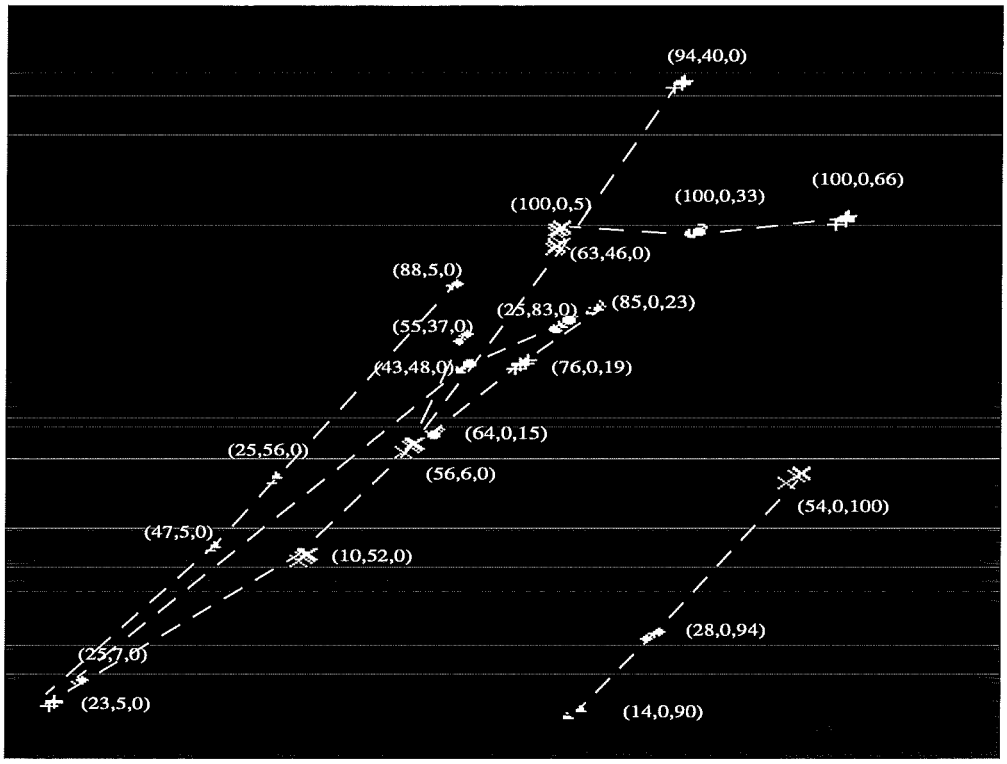


Abbildung 6.11: *Visualisierung der Gasanalyse-Daten. Die einzelnen Meßpunkte geben die Konzentrationen von Benzol, Xylol und Oktan in % an. Die gestrichelten Linien symbolisieren die einzelnen Messungen. Bei jeder Messung wurde die Konzentration eines der Gase auf Null gehalten.*

6.6 Zusammenfassung

In diesem Kapitel wurden die wichtigsten Merkmale des NNDB beschrieben. Der NNDB realisiert alle in dieser Arbeit besprochenen Verfahren. Darüber hinaus bietet er eine Datenbank und Merkmalsextraktion an. Seine Anwendung auf Benchmarkprobleme und der Einsatz in der Gasanalyse wurden beschrieben. Im nächsten Abschnitt folgt die bis dato wichtigste Anwendung in der Auswertesoftware des Rißprüfmolches.

Kapitel 7

Anwendung der geometrischen Interpretation bei der Erkennung von Rissen in Pipelines

Die geometrische Interpretation wurde erfolgreich bei der Visualisierung und Manipulation von neuronalen Netzen eingesetzt, die in einem Rißerkennungssystem, dem sog. *Rißprüfmolch*, arbeiten. Das System erkennt Längsrisse in Ölleitungen und ersetzt die bis dato nötige Wasserdruckprüfung, die eine erhebliche Menge an umweltbelastendem Sondermüll (Öl-Wasser-Gemisch) verursacht. Die Rißerkennung mit dem Rißprüfmolch kann während des normalen Betriebes der Ölleitung erfolgen. Die untersuchten Netze wurden in einem Offline-Auswertungssystem eingesetzt, welches Schweißnähte erkennt.

7.1 Der Rißprüfmolch

Der Rißprüfmolch (kurz: Molch) ist ein Gerät, welches in eine Ölleitung eingeführt und mittels des Öldrucks durch die Rohre durchgeschleust wird. Während des Durchlaufs nimmt er mit Hilfe mehrerer hundert Ultraschallsensoren Ultraschallbilder von der Rohrwand auf. Die Ultraschallsensoren senden unter fest vorgegebenen Winkeln und in festgelegter Reihenfolge Ultraschallimpulse in die Rohrwand ein (sog. *Schüsse*). Aufgrund seiner kleinen Wellenlänge wird das Ultraschallsignal von verschiedenen Reflektoren[1] gespiegelt und gelangt als Echo zurück zu den Ultraschallsensoren, die dann als Signalempfänger arbeiten. Das empfangene Echo wird in komprimierter Form auf Massenspeicher (DAT-Bänder) geschrieben. Abgespeichert werden dabei nur Ultraschallechos, welche von Reflektoren stammen. Die Anordnung der Sensoren und die

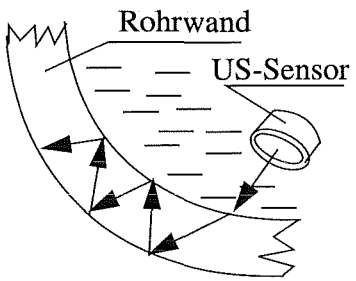


Abbildung 7.1: *Radiale Ausbreitung des Ultraschallsignals in der Rohrwand.*

1. Alle Schäden und Materialinhomogenitäten in einer Rohrwand sind Reflektoren

Reihenfolge der Schüsse ist so gewählt, daß kein Riß übersehen wird, der eine Mindestlänge aufweist. Die Datenauswertung erfolgt nach dem Lauf des Molchs im sog. *Offline-Betrieb*. Dafür werden die auf Massenspeichern gespeicherten Daten aufbereitet und ausgewertet. Ein nicht unwesentlicher Teil der Ultraschallbilder (bis zu 50%) beinhaltet die Längsschweißnaht. Das Schweißnahtecho ist dabei so stark, daß die Erkennung von Rissen aus solchen Bildern nicht möglich ist. Risse, die sich im Schweißnahtbereich befinden, müssen aus den Ultraschallbildern anderer Sensoren erkannt werden. Die Aufgabe der automatischen Auswertung besteht im wesentlichen aus der Ausblendung vom Ultraschallbildbereichen, die irrelevante Reflektoren zeigen, wie z. B. Schweißnähte.

Damit die Arbeitsweise des Schweißnahterkennungssystems besser verstanden werden kann, wird zuerst der Aufbau eines Ultraschallbildes erläutert.

7.2 Aufbau eines Ultraschallbildes

Während der radialen Ausbreitung in der Rohrwand wird das Ultraschallsignal von Reflektoren gespiegelt und erzeugt Echos. Ein Reflektor kann die Innenwand, die Außenwand oder eine Materialstörung im Rohr sein (z. B.: Riß, Schweißnaht, Einschluß etc.). Bei der Aufnahme der Ultraschallbilder wird die Echostärke zum gegebenen Zeitpunkt aufgenommen und im Bild als Spalte entlang der Signallaufzeitachse (engl. Time of Flight), farbig aufgetragen. Eine solche Spalte heißt ein A-Bild. Hintereinander angeordnete A-Bilder ergeben ein zweidimensionales B-Bild. Ein B-Bild wird von jedem Ultraschallsensor aufgenommen. Die X-Koordinate gibt die Position im Rohr und die Y-Koordinate die Signallaufzeit (Time of Flight) in der Rohrwand an. Die Farbe des Pixel (in der gedruckten Version der Grauwert) im Punkt $P(x,y)$ stellt die Signalamplitude des Echos dar.

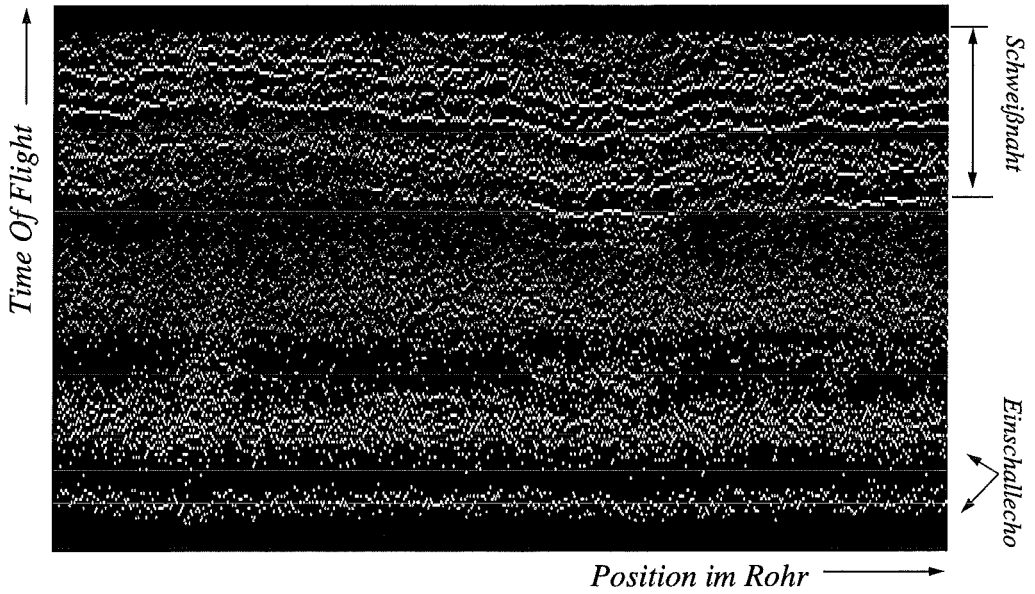


Abbildung 7.2: *Beispiel für ein B-Bild. Im oberen Bereich ist ein Schweißnahtecho zu sehen. Im unteren Bereich liegt das Einschallecho.*

Schweißnähte heben sich in Ultraschallbildern dadurch hervor, daß sie aus mindestens drei parallelen und korrelierenden Liniensegmenten bestehen, die eine Mindestlänge aufweisen und sich periodisch wiederholen. Konkrete Werte für die Parameter, wie Korrelation, Mindestlänge und Periode, sind als Regeln bekannt. In Abbildung 7.2 befindet sich eine Schweißnaht im oberen Teil des B-Bildes.

7.3 Das Schweißnahterkennungssystem

Die Schweißnahterkennung bearbeitet nur die Ultraschallbilder, welche höchstwahrscheinlich Schweißnähte enthalten. Diese Ultraschallbilder werden von einem anderen Teil der Auswertungssoftware zur Verfügung gestellt. Weil die Daten entlang des Rohres sequentiell aufgezeichnet worden sind, werden sie auch sequentiell an die Rißerkennungssoftware weiter gereicht. Diese speichert einen kontinuierlichen Ausschnitt des B-Bildes als ein Auswertungsfenster ab, das sich entlang des Rohres bewegt. Alle Bearbeitungsschritte beziehen sich immer auf einen Bildausschnitt, der sich im Fenster befindet.

Das Schweißnahterkennungssystem besteht aus mehreren Modulen. Die Aufgaben der Module reichen von der Datenverwaltung bis hin zur Klassifikation.

Die Datenverwaltung. Die Datenverwaltung stellt die Schnittstelle zur Außenwelt her, verwaltet Ultraschalldaten und wandelt sie ggf. in ein internes Format um. Die primäre Aufgabe besteht jedoch in der Sortierung der A-Bilder nach Sensoren und Position im Rohr und in der Ermittlung derjenigen Sensoren, welche über einen längeren Zeitintervall keine Daten geliefert haben. Solche Sensoren werden als ungültig erklärt und aus dem Speicher gelöscht. Diese Strategie reduziert den Speicherbedarf auf ein notwendiges Minimum.

Die Vorverarbeitung. Die Vorverarbeitung extrahiert aus den Bilddaten verschiedene Merkmale, die später als Eingabe für ein neuronales Netz dienen. Die Vorverarbeitung transformiert zuerst das Bild derart, daß es mit Hilfe einer einfachen Schwelle in Segmente zerlegt werden kann, die durch ein paar Merkmale einfach beschrieben werden können. Da im Falle der Schweißnahterkennung nach Linien gesucht wird, sollte das Ultraschallbild so zerlegt (segmentiert) werden, daß alle Linien hervorgehoben werden. Dies kann am besten mit Hilfe einer Wavelet-Transformation geschehen.

Die Wavelet-Transformation. Theoretisch gesehen zerlegt die Wavelet-Transformation jede Funktion in sog. *Wavelets*. Alle Wavelets sind selbstähnlich und entstehen durch eine Verschiebung und Stauchen bzw. Strecken (d. h. eine Skalierung) aus einem sog. *Basiswavelet* (auch *Motherwavelet* genannt). Jedes Wavelet läßt sich also durch die Angabe von einem Basiswavelet und zwei Parametern der Verschiebung und Skalierung, eindeutig beschreiben. Als Basiswavelet kann jede beliebige Funktion genommen werden, welche bestimmte Voraussetzungen erfüllt [Daub92]. Für die Erkennung von Linien eignen sich am besten die sog. *Gabor-Wavelets*. Sie sind so ausgelegt, daß sie Linien und Kanten sehr gut erkennen können, indem sie diese im transformierten Bild durch eine höhere Amplitude betonen. Eine der Eigenschaften von Gabor-Wavelets ist die Fähigkeit, Lücken in Linien zu schließen.

Das Schließen von Lücken ist ein wichtiges Merkmal bei der Segmentierung von Ultraschallbildern, weil in diesen Bildern viele Lücken auftreten. Die Schließung erfolgt dabei auf eine unscharfe Weise. Es wird nicht immer eine Lücke von k Pixel geschlossen, sondern die Entscheidung, wann eine Lücke geschlossen wird, hängt von der Anzahl der Pixel auf beiden Seiten der Lücke ab. In Abbildung 7.3 ist die Wavelet-Transformierte des B-Bildes aus Abbildung 7.2 dargestellt. Durch diese Transformation wurden kleinere Lücken geschlossen und Strukturen, die eine linienartige Form haben, hervorgehoben. Dieses Bild läßt sich mit einem Schwellenwert segmentieren.

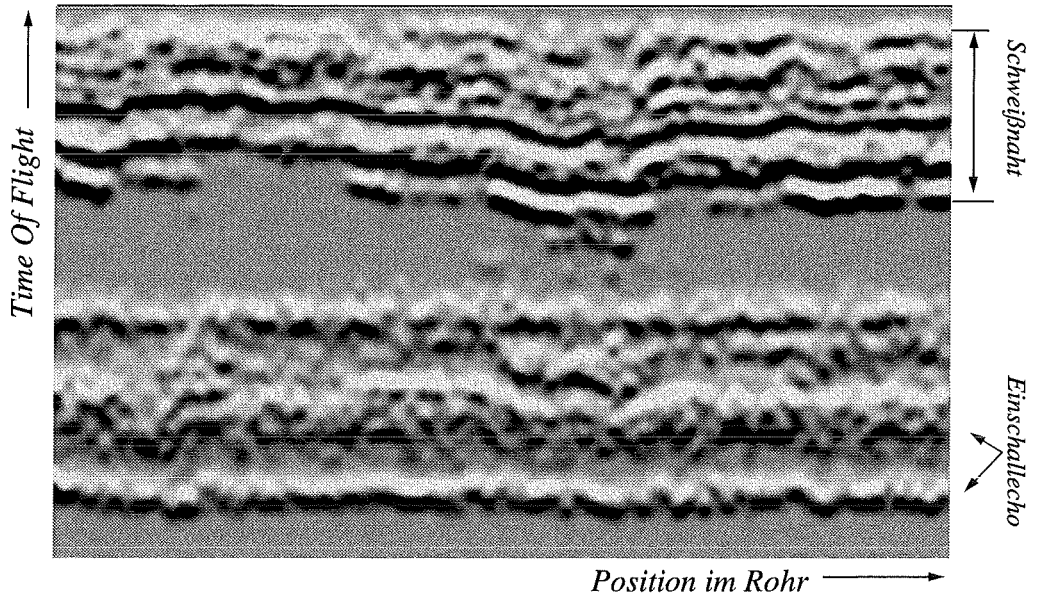


Abbildung 7.3: *Wavelet-Transformierte des B-Bildes aus Abbildung 7.2. Die Linienstruktur wurde durch die Transformation hervorgehoben.*

In Abbildung 7.4 ist das segmentierte B-Bild gezeigt. Durch den Vergleich der Abbildung 7.2 und Abbildung 7.4 läßt sich deutlich erkennen, daß die Zusammenfassung in Segmente annähernd nach menschlichem Empfinden geschah.

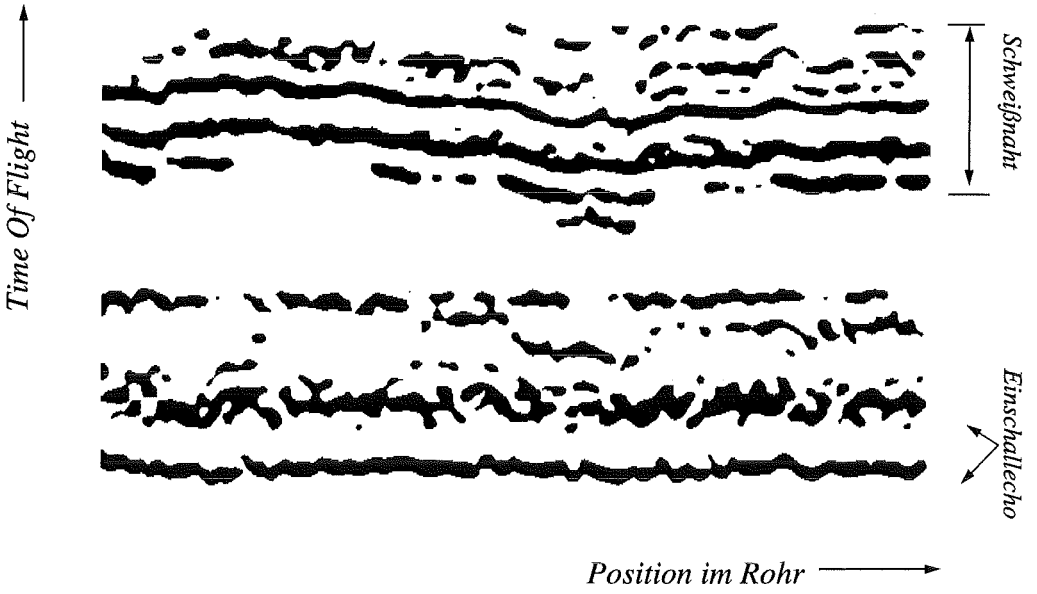


Abbildung 7.4: *Segmentiertes B-Bild aus Abbildung 7.2.*

Die Segmente haben verschiedene Formen. Für die weitere Bearbeitung werden allerdings nur linienartige Segmente benötigt. Deswegen wird im nächsten Abschnitt eine Merkmalsextraktion beschrieben, welche Segmente charakterisiert. Im übernächsten Abschnitt werden die Segmente anhand der Merkmale durch ein neuronales Netz klassifiziert.

Merkmalsextraktion. Damit die Segmente mit Hilfe eines neuronalen Netzes klassifiziert werden können, müssen sie vorher durch Merkmale, welche die Eingabe für das Netz sind, beschrieben werden.

Nach der Segmentierung wird für jedes einzelne Segment eine Reihe von Merkmalen berechnet, die anschließend als Eingabe für ein klassifizierendes neuronales Netz dienen. In diesem Abschnitt soll ein neuronales Netz (von mehreren) mit Hilfe der geometrischen Interpretation untersucht werden.

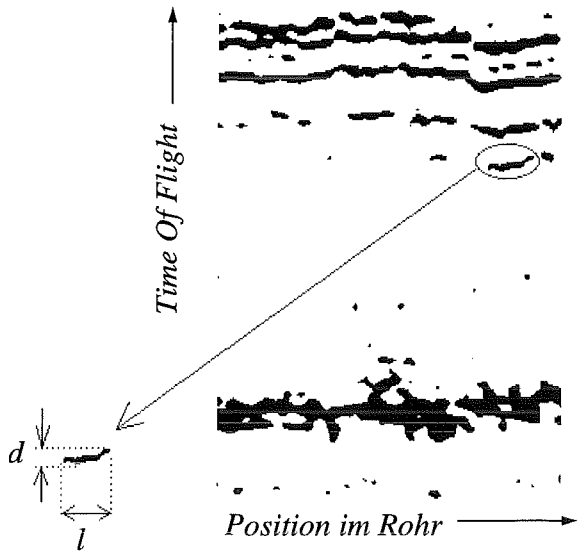


Abbildung 7.5: *Für jedes Segment wird ein Satz von Parametern bestimmt.*

Das untersuchte Netz soll Segmente in linienartige und nicht linienartige aufteilen. Dazu werden dem Netz der Reihe nach Merkmale aller Segmente präsentiert. Das Netz entscheidet dann, ob das präsentierte Segment linienartig ist oder nicht. Die Merkmale sind:

- Das Verhältnis Länge zu Breite, normiert auf die Fensterlänge
- Ein Index, welcher angibt, wie glatt die Ränder sind, d. h.: sind die Ränder stark ausgefranst, dann ist dieser Index groß; sind sie dagegen glatt, dann ist der Index klein.

Diese Merkmale wurden von zwei zusätzlichen neuronalen Netzen aus den statistischen Parametern der Segmente berechnet. Diese Netze werden aber hier nicht weiter erläutert.

Klassifikation. Die Klassifikation erfolgt mit einem neuronalen Netz, welches vier Neuronen in seiner ersten verdeckten Schicht hat. Dieses neuronale Netz wurde mit Rprop (einer Variante von Backpropagation, s. [Riedmiller93]) trainiert. Nach *3000* Lernschritten teilte das neuronale Netz den Eingaberaum gemäß Abbildung 7.6.

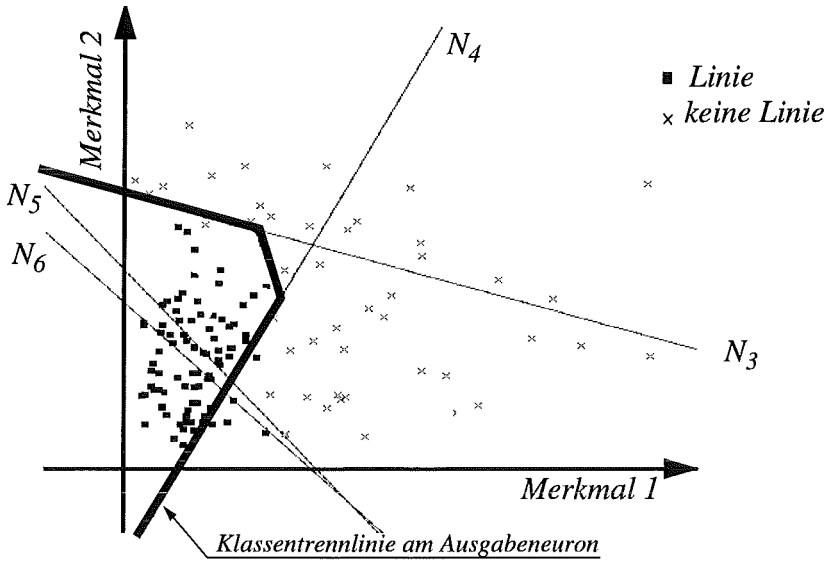


Abbildung 7.6: *Die Aufteilung des Merkmalsraums durch ein neuronales Netz mit vier verdeckten Neuronen.*

Es ist deutlich zu sehen, daß es Fehlklassifikationen gibt. Diese fehlklassifizierten Segmente sind in der unteren Tabelle aufgelistet.

Segment	**Klasse (Trainingsdaten)**	**Klassifikation durch Netz**
	keine Linie[a]	keine Linie
	Linie	keine Linie
	Linie	keine Linie
	Linie	keine Linie
	Linie	keine Linie
	Linie	keine Linie

a. Dieses Segment wurde vom Experten zuerst fälschlicherweise als "keine Linie" klassifiziert. Erst durch die Interpretation des Netzes wurde dieser Fehler bemerkt und korrigiert.

Die Fehler treten bei Segmenten auf, die von einem – auch ungeübten – Betrachter als linienartig eingestuft würden. Sie kommen in dem konkreten Fall deswegen zustande, weil die falsch klassifizierten Muster außerhalb des Clusters-Zentrum liegen. Da der Lernalgorithmus auf das Minimieren der Summe des quadratischen Fehlers ausgelegt ist (Backpropagation), gerät er in ein lokales Minimum und kommt von dort nicht her herraus. Für eine korrekte Funktionsweise des Systems ist eine korrekte Klassifikation dieser Segmente aber notwendig. Deshalb wurde das Netz so manipuliert, daß die falsch klassifizierten Muster in die Klasse "linienartig" fielen. Dazu war es notwendig, das Neuron N_6 zu entfernen und das Neuron N_5 etwas zu drehen. Das Ergebnis der Manipulation zeigt die Abbildung 7.7.

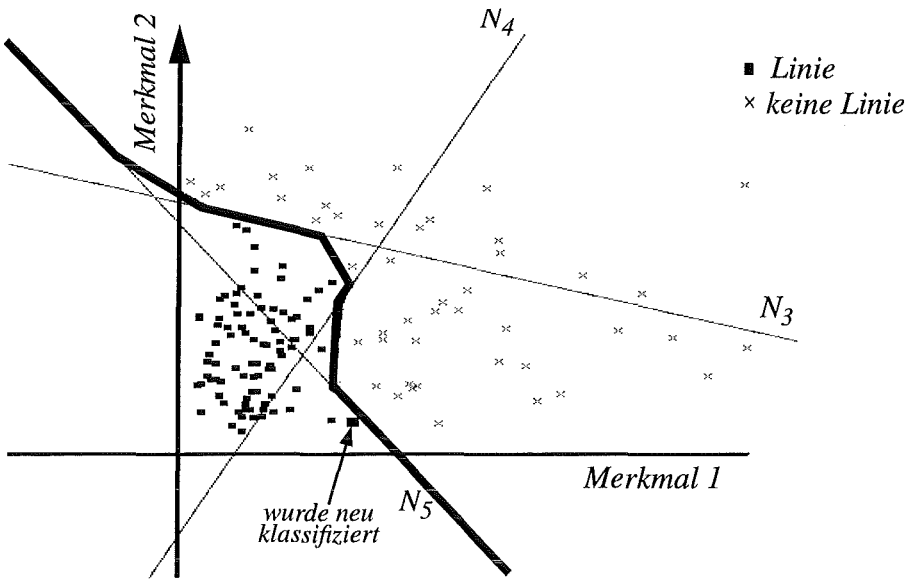


Abbildung 7.7: *Das neuronale Netz aus Abbildung 7.6 nach der Manipulation. Das Neuron N_6 wurde gelöscht und das Neuron N_5 etwas gedreht.*

Weitere Bearbeitungsschritte. In weiteren Schritten wird zuerst ein sog. *Referenzsegment* mit einem weiteren neuronalen Netz bestimmt. Das Referenzsegment wird dann mit anderen Segmenten korreliert und liefert den Korrelationskoeffizienten sowie die Periode der Linienstruktur. Anhand der Regeln für Korrelation, Periodizität und Mindestlänge wird entschieden, ob die gefundene Linienstruktur eine Schweißnaht ist. Diese Entscheidung kann ebenfalls mit einem neuronalen Netz getroffen werden. In der praktischen Anwendung hat sich jedoch eine regelbasierte Entscheidung als besser erwiesen, da sich die Parameter von Rohrleitung zur Rohrleitung stark unterscheiden.

7.4 Zusammenfassung

In diesem Kapitel wurde die bisher wichtigste Anwendung von GINN und IGM vorgestellt, nämlich der Einsatz bei der Bewertung von neuronalen Netzen, welche in der Rißerkennung des Rißprüfmolches arbeiten (diese Anwendung wird als "sicherheitsrelevant" eingestuft). Es hat sich gezeigt, daß der Einsatz von GINN und IGM wesentlich zur Verbesserung der Qualität des verwendeten Netzes beigetragen hat. Insbesondere konnten falsch klassifizierten Muster erkannt und der Verlauf der Klassentrennlinie so korrigiert werden, daß sie in die korrekte Klasse fielen. Es konnte sogar ein vom Experten falsch klassifiziertes Muster ausgemacht werden, welches dann umklassifiziert wurde. Dadurch wurden die Meßdaten konsistenter.

Kapitel 8

Zusammenfassung und Ausblick

Im Rahmen dieser Dissertation wurde ein Interpretationsverfahren für neuronale Netze entwickelt. Im Mittelpunkt stand dabei die Interpretation vorwärtsgerichteter Netze mit einer oder zwei verdeckten Schichten, welche in Klassifikationsapplikationen eingesetzt werden. Die Einschränkung auf vorwärtsgerichtete Netze hat sich als berechtigt erwiesen, da ca. *60%* aller Anwendungen diesen Netztyp verwenden. Eine Erweiterung auf andere Netztypen und mehrere Schichten ist dabei leicht realisierbar und in dem Verfahren berücksichtigt worden.

Die geometrische Interpretation deckt drei Themengebiete ab: zum ersten die simultane Visualisierung von hochdimensionalen Daten und neuronalen Netzen, zum zweiten die Manipulation eines trainierten Netzes und zum dritten die Topologieoptimierung.
Die generelle Vorgehensweise bei der Visualisierung eines Netzes kann in drei Punkten angegeben werden:

1. Um die Funktion eines Neurons N_i in einem beliebigen mehrschichtigen Netz darzustellen, werden zunächst alle anderen Neuronen in der gleichen Schicht ausgeblendet,
2. der Klassentrennwert auf die Netzaktivierung dieses Neurons zurückprojiziert und schließlich
3. der Verlauf der Klassentrennlinie bezüglich des Eingaberaums als eine Überlagerung der stückweise linearen Approximation der Neuronen der vorhergehenden Schichten berechnet.

Bei hochdimensionalen Eingaberäumen erfolgt die Darstellung des Netzes und der Lern- und Testdaten mit der sog. *Local Principal Component Analysis* (LPCA). Dieses neuartige Visualisierungsverfahren ist eine stückweise lineare Annäherung einer nichtlinearen PCA-Transformation. Ein Vorteil ergibt sich dabei vor allem für die graphische Manipulation des neuronalen Netzes. Da die im LPCA-Raum graphisch manipulierte Darstellung in Neuronenparameter umgerechnet werden muß, gestaltet sich diese Berechnung wegen der stückweise linearen Transformation besonders effizient und numerisch stabil. Gegenüber anderen Visualisierungsmethoden hat die geometrische Interpretation den Vorteil, daß sie gleichzeitig die Daten und das Netz in derselben Darstellung anzeigt. Der Anwender kann dann sehr einfach die Qualität der Lösung beurteilen. Auch ein ungeübter Benutzer kann sehr leicht falsche Lösungen erkennen, z. B.

dann, wenn die Aufteilung des Eingaberaumes sehr zerklüftet ist oder der Verlauf der Klassentrennlinie nicht dem erwarteten entspricht (z. B. linear anstatt v-förmig). Ausreißer werden automatisch erkannt und in der Darstellung angezeigt. Dies hat sich z. B. als sehr nützlich bei der Beurteilung eines neuronalen Netzes erwiesen, das bei der Schweißnahterkennung arbeitet und ein Teil eines automatischen Rißerkennungssystem ist.

Die Manipulation ist der andere große Bestandteil der geometrischen Interpretation. Sie erlaubt eine Veränderung der Netzparameter, und zwar nicht durch die Änderung einzelner Gewichte oder Bias-Werte, sondern durch die Manipulation der Visualisierung. Der Benutzer führt dabei affine Abbildungen auf der Darstellung durch, ähnlich wie in einem Zeichenprogramm. Er kann einzelne Neuronen, Neuronengruppen oder die Klassentrennlinie verändern. Die geometrische Interpretation rechnet diese Änderungen in neue Netzparameter um. Bestimmte Teile der Klassentrennlinie, Neuronen oder Neuronengruppen können verankert werden, um dadurch unerwünschte Nebeneffekte zu vermeiden. Das Einfügen und Löschen von Neuronen, Neuronengruppen oder Teilen der Klassentrennlinie bzw. neuronalen Schichten ist ebenfalls möglich.

Die Visualisierung zusammen mit der Manipulation eröffnet einem Anwender neue Wege für das Einbringen von nichtformalem und formalem Wissen in die Netzstruktur. Das Paradigma ist dabei: Sehen-Verstehen-Manipulieren. Der Benutzer betrachtet die Lösung, vergleicht sie mit der eigenen Vorstellung und korrigiert sie. Das Wissen, welches er dabei einbringt, braucht nicht vollständig zu sein. Es kann sich nur auf bestimmte Eingabedimensionen oder auf einen Ausschnitt des Definitionsbereiches beziehen. In den meisten Fällen helfen schon kleine Veränderungen, um die Lösung wesentlich zu verbessern.

Die Topologieoptimierung, welche auf Berechnungen in einer abelschen Gruppe basiert, kann überflüssige Neuronen aufspüren und sie aus dem Netz entfernen, ohne daß sich dabei die Lösung im Definitionsbereich ändert. Dieses Verfahren ist unabhängig von der betroffenen Netzschicht und reduziert dabei zu groß dimensionierte Netze.

Die geometrische Interpretation ist als Neuronale-Netze-Debugger (NNDB) implementiert worden. Dieses Programm realisiert die oben beschriebene Funktionalität und bietet darüber hinaus eine Animation des Lernvorganges und ein nützliches Werkzeug für tabellarische Datenverwaltung und Musterextraktion.

Die geometrische Interpretation, so wie sie in dieser Arbeit beschrieben wurde, legt den Grundstein für weitere Arbeiten auf diesem Gebiet. Bisherige Erfahrungen mit der geometrischen Interpretation zeigen, daß es durchaus möglich ist, eine bestimmte Menge von Neuronen als Bausteine zu nehmen, um daraus jede beliebige Lösung zu erzeugen. Umgekehrt können mit einer beschränkten Anzahl von Neuronen nur bestimmte Lösungen generiert werden. Daraus läßt sich z. B. ein neues hierarchisches Lernverfahren konstruieren, welches insbesondere auch Nebenbedingungen und das Wissen des Benutzers berücksichtigt.

Literaturverzeichnis

[Asfour95] Y. R. Asfour, G. A. Carpenter, S. Grossberg, Boston University, "Landsat Satellite Image Segmentation Using the Fuzzy ARTMAP Neural Network", World Congress on Neural Networks, Washington, D.C., July 17-21, 1995, pp. I-150-I-156.

[Azoff94] E. M. Azoff, "Neural Network Time Series Forecasting of Financial Markets", John Wiley, 1994.

[Ban93] M. Banerjee, M. K. Chakraborty, "Logic of Rough Sets", in V. S. Alagar et. al. (eds.) Incompleteness and Uncertainty in Information Systems, Proc. SOFTEKS Workshop on Incompleteness and Uncertainty in Information Systems, Concordia University, Montreal, Canada, 1993, Springer-Verlag, pp. 223-244.

[Bern96] A. Bernatzki, W. Eppler, H. Gemmeke, "Interpretation of Neural Networks for Classification Tasks", Fourth European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, September 2-5, 1996, pp. 1420-1424.

[Bron60] I. N. Bronstein, K. A. Semendjajew, "Taschenbuch der Mathematik", BSB B. G. Teubner Verlagsgesellschaft, Leipzig, 1960.

[Chen96] W.-Y. Chen, S.-H. Chen, C.-J. Lin, "A Speech Recognition Method Based on the Sequential Multi-Layer Perceptrons", Neural Networks, Pergamon, 1996, pp. 655-670.

[Craven91] M. W. Craven, J. W. Shavlik, "Visualizing Learning and Computation in Artificial Neural Networks", Machine Learning Research Group Working Paper 91-5, 1991.

[Daub92] I. Daubechies, "Ten Lectures on Wavelets", Society for Industrial and Applied Mathematics (SIAM), 1992.

[Dennis91] S. Dennis, S. Philips, "Analysis Tool for Neural Networks", Technical Report 207, Department of Computer Science, University of California, San Diego, 1991.

[Diamantaras96] K. I. Diamantaras, S. Y. Kung, "Principal Component Neural Networks, Theory and Applications", John Wiley & Sons, Inc, New York, 1996.

[Diamantaras92] K. I. Diamantaras, "Principal Component Learning Networks and Applications", Ph. D. thesis, Princeton University, 1992.

[Dohmen91] M. Dohmen, W. Oberschelp, "Mathematische Methoden für Bildverarbeitung und Computergraphik, Spezielle algorithmische Probleme der Computergraphik", Schriften zur Informatik und Angewandten Mathematik, interner Bericht am Lehrstuhl für Angewandte Mathematik insbesondere Informatik, Aachen 1991.

[Elman89] J. L. Elman, "Representation and Structure in Connectionist Models", Technical Report 8903, Center for Research in Language, University of California, San Diego, 1989.

[Eppler93] W. Eppler, "Vorstrukturierung Neuronaler Netze mit Fuzzy-Logik", Fortschrittberichte VDI, Reihe 10: Informatik/Kommunikationstechnik, Nr. 266, VDI Verlag GmbH, Düsseldorf 1993.

[Fahlman88] S. E. Fahlman, "An Empirical Study of Learning Speed in Backpropagation Networks", Carnegie-Mellon Computer Science Rpt. CMU-CS-88-162, 1988.

[Földiák89] P. Földiák, "Adaptive Network for Optimal Linear Feature Extraction", in Int. Joint Conf. Neural Networks, Washington DC, 1989, pp. 401-406.

[Hebb49] D. O. Hebb, "The Organization of Behavior", Wiley, New York, 1949.

[Hecht89] R. Hecht-Nilsen, "Neurocomputing", Addison-Wesley Publishing Company, 1989.

[Hinton86] G. E. Hinton, J. L. McClelland, D. E. Rumelhart, "Distributed Representations", in Rumelhart, D. E., J. L. McClelland [Eds.], Parallel Distributed Processing: Exportation in the Microstructure of Cognition, MIT Press, Cambridge, MA, 1986, pp. 77-109.

[Hopfield84] J. J. Hopfield, "Neurons with Graded Response have Collective Computational Properties Like those of Two-State Neurons", Proc. Natl. Acad. Sci., 81, 1984, pp. 3088-3092.

[Hopfield82] J. J. Hopfield,"Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proc. Natl. Acad. Sci., 79, 1982, pp. 2554-2558.

[Hornik92] K. Hornik, C.-M. Kaun, "Convergence Analysis of Local Feature Extraction Algorithms", Neural Networks, 2, pp. 229-240, 1992.

[Hotelling33] H. Hotelling, "Analysis of a Complex of Statistical Variables into Principal Components", J. Educ. Psychol., 24, pp. 498-520, 1933.

[Jájá92] J. Jájá, "An Introduction to Parallel Algorithms"Addison-Wesley, 1992, pp. 272-277.

[Karhunen46] K. Karhunen, "Zur Spektraltheorie stochastischer Prozesse", Ann. Acad. Sci. Fenn., 34, 1946.

[Karhunen82] J. Karhunen, E. Oja, "New Methods for Stochastic Approximation of Truncated Karhunen-Loève Expansion", Proc. 6th Int. Conf. on Pattern Recognition, Springer-Verlag, New York, 1982, pp. 550-553.

[Khalid95] M. Khalid, S. Omatu, R. Yusof, "Temperature Regulation with Neural Networks and Alternative Control Schemes", IEEE Transactions on Neural Networks, vol. 6, May 1995, pp. 572-582

[Kolmogorov57] A. N. Kolmogorov, "On the Representation of Continuous Functions of Many Variables by Superposition of Continuous Functions of One Variable and Addition", Dokl. Akad. Nauk UdSSR, 114, 1957, pp. 953-956.

[Köppen96] B. Köppen-Seliger, M. Schubert, P. M. Frank, "Recurrent Neural Networks for Fault Detection", Fourth European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, September 2-5, 1996, pp. 240-244.

[Kung90] S. Y. Kung, K. I. Diamantaras, "A Neural Network Learning Algorithm for Adaptive Principal Component EXtraction (APEX)", Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing, Albuquerque, April 1990, pp. 861-864.

[Kun89] A. Kunzmann, "Test synchroner Schaltwerke auf der Basis partieller Prüfpfade", VDI-Verl., II, 139 S.,Fortschrittberichte / VDI : Reihe 10; Nr. 117 , Düsseldorf, 1989, ISBN 3-18-141710-6

[Lang88] K. J. Lang, M. J. Witbrock, "Learning to Tell Two Spirals Apart", Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann, 1988, pp. 52-59.

[LeCun90] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, "Handwritten Digital Recognition with a Back Propagation Network", In Advances in Neural Information Processing Systems, vol. 2, San Mateo, CA. Morgan Kaufmann, 1990, pp. 396-404.

[Lehrbaß96] F. B. Lehrbaß, M. J. Peter, "DAX Future Trading with a Neural Network", Fourth European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, September 2-5, 1996, pp. 2145-2148.

[Loève48] M. Loève, "Functions aléatoires du second order", in P. Lèvy [Ed.] Suppl. to Processus Stochastique et mouvement Brownien, Gouthière-Villars, Paris, 1948.

[Minsky69] M. Minsky, S. Papert, "Perceptrons", MIT Press, Cambridge MA, 1969.

[Munoro91] P. Munoro, "Visualisations of 2-D Hidden Unit Space", Technical Report LIS035/IS91003, School of Library and Information Science, University of Pittsburgh, Pittsburgh, 1991.

[Oja82] E. Oja, "A Simplified Neuron Model as Principal Component Analyzer", J. Math. Biol., 15, 1982, pp. 267-273.

[Parker85] D. B. Parker, "Learning-Logic" Technical Report TR-47, Centre for Computational Res. in Economics and Management, Sci., MIT April 1985.

[Parker86] D. B. Parker, "A Comparison of Algorithms for Neuron-like Cells", Denker, J. [Ed.], Proc. Second Annual Conf. on Neural Networks for Computing, 151, American Inst. of Phisics, New York, 1986, pp. 327-332.

[Parker87] D. B. Parker, "Optimal Algorithms for Adaptive Networks: Second Order Back Propagation, Second Order Direct Propagation, and Second Order Hebbian Learning", Proc. of the Int. Conf. on Neural Networks, II, IEEE Press, New York, June 1987, pp. 593-600.

[Pearson01] K. Pearson, "On Lines and Planes of Closest Fit to System of Points in Space", Philos. Mag., Ser. 6, 2, 1901, pp. 559-572.

[Pomerleau89] D. A. Pomerleau, Alvinn, "An Autonomous Land Vehicle in a Neural Network", Advances in Neural Information Processing Systems, vol. 1, San Mateo, CA. Morgan Kaufmann, 1989, pp. 305-313.

[Pratt91] L. Y. Pratt, J. Mostow, "Direct Transfer of Learned Information among Neural Networks", Proceedings of the Ninth National Conference on Artifical Intelligence, Anaheim, CA, 1991, pp. 584-589.

[Prechelt94] L. Prechelt, "Proben1 – A Set of Neural Network Benchmark Problems and Benchmarking Rules", Fakultät für Informatik der Universität Karlsruhe, Technical Report 21/94, September 30, 1994.

[Rehkugler90] H. Rehkugler, T. Poddig, "Statistische Methoden versus Künstliche Neuronale Netzwerke zur Aktienkursprognose - eine vergleichende Studie", Bamberger Betriebswirtschaftliche Beiträge Nr. 73/1990, Otto-Friedrich-Universität Bamberg, 1990.

[Riedmiller93] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", Proc. ICNN, San Francisco, 1993.

[Ritter90] H. Ritter et al., "Neuronale Netze", Addison-Wesley, 1990.

[Rosenblatt58] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", Psychol. Rev., 65, 1958, pp. 386-408.

[Rubner89] J. Rubner, P. Tavan, "A Self-Organizing Network for Principal-Components Analysis", Europhys. Lett., 10(7), 1989, pp. 693-698.

[Rubner90] J. Rubner, K. Schulten, "Development of Feature Detectors by Self-Organisation", Biol. Cybernet., 62, 1990, pp. 193-199.

[Rumelhart86] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning Internal Representation by Error Propagation", in D. E. Rumelhart, J. L. McClelland [Eds.], "Parallel Distributed Processing: Exploration in the Microstructure of Cognition I", MIT Press, Cambridge MA, 1986, pp. 318-362.

[Rumelhart86a] D. E. Rumelhart, J. L. McClelland, "Parallel Distributed Processing: Exploration in the Microstructure of Cognition I & II", MIT Press, Cambridge MA, 1986.

[Said95] J. N. Said, K. Khorasani, C. Y. Suen, Concordia University, "A New Back-Propagation Learning Algorithm with Application to Unconstrained Handwritten Character Recognition", World Congress On Neural Networks Washington, D.C., July 17-21, 1995, pp. II-217-II-221.

[Sanger89] T. D. Sanger, "An Optimally Principle for Unsupervised Learning", in D. S. Touretzky, [Ed.], Advances in Neural Information Processing Systems, Morgan Kaufmann, San Mateo, CA, 1989, pp. 11-19.

[Schöneburg90] E. Schöneburg et al., "Neuronale Netzwerke: Einführung, Überblick und Anwendungsmöglichkeiten", Markt & Technik-Verlag, 1990.

[Sejnowski87] T. Sejnowski, C. Rosenberg, "Parallel Networks that Learn to Pronounce English Text", Complex Systems, 1987, pp.145-168.

[SNNS99] Online-Manual von: "Stuttgart Neural Network Simulator", http://www-ra.informatik.uni-tuebingen.de/forschung/snns/

[Tesauro89] G. Tesauro, T. J. Sejnowski, "A Parallel Network that Learns to Play Backgammon", Artifical Intelligence, 39, 3, 1989, pp. 357-390.

[Wejchert90] J. Wejchert, G. Tesauro, "Neural Network Visualisation", Advances in Neural Information Processing Systems, vol. 2, San Mateo, CA, Morgan Kaufmann, 1990, pp. 465-472.

[Werbos74] P. J. Werbos, "Beyond Regression", Doctoral Dissertation, Appl. Math., Harvard University, November 1974.

[White93] H. White, "Economic Prediction Using Neural Networks: The Case of IBM Daily Stock Returns." Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance. Eds. Robert Trippi and Efraim Turban. Probus Publishing Company. Chicago, IL. 1993. pp. 315-328.

[Wiles90] J. Wiles, J. Stewart, A. Bloesch, "Pattern of Activations are Operators in Recurrent Networks", Technical Report 189, Department of Computer Science, University of Queensland, Queensland, Australia, 1990.

[Zadeh65] L. A. Zadeh, "Fuzzy Sets, in Information and Control", Vol. 8, Academic Press New York, 1965.

[Zell94] A. Zell, "Simulation Neuronaler Netze", Addison-Wesley, 1994.

Lebenslauf

Persönliche Daten

Name	Andreas Alfred Bernatzki
Anschrift	An der Vogelhardt 8 76149 Karlsruhe Tel.: 07247/82-6032 Fax: 07247/82-3560
Geburtsdatum	11.02.1965
Geburtsort	Hindenburg
Familienstand	ledig
Staatsangehörigkeit	deutsch

Schulbildung

09/1972 - 06/1980	Grundschule in Hindenburg
09/1980 - 06/1984	Allgemeinbildendes Gymnasium in Hindenburg
07/1986 - 06/1987	Kolleg für Aussiedler aus osteuropäischen Ländern in Geilenkirchen

Akademische Bildung

10/1984 - 06/1985	Fakultät für Automatik und Informatik der Politechnika Slanska in Gleiwitz
10/1987 - 04/1994	Studium der Informatik an der RWTH Aachen
ab 10/1994	Wissenschaftlicher Mitarbeiter am Forschungszentrum Karlsruhe