# Vektorbasierte Neuronale Netze

Bernd Fritzke

# Vorwort

Dieses Buch handelt von adaptiven Systemen. Die Parameter dieser Systeme verändern sich unter dem Einfluß von Daten, welche „von außen“ vorgegeben werden. Der genaue Endzustand des Systems ist im Voraus nicht bekannt. Er entwickelt sich vielmehr durch einen Prozeß der Selbstorganisation, bei dem sich die vielen ähnlichen „Einheiten“, aus denen das System besteht, gegenseitig beeinflussen können. Oft ist der Einflußbereich jeder Einheit beschränkt auf eine kleine Anzahl von benachbarten Einheiten.

Die Einzelheiten des Selbstorganisationsprozesses entziehen sich einer zentralen Planung völlig. Zum einen ist der Anfangszustand des Systems in der Regel zufällig. Darüberhinaus unterliegen die Systemparameter und die präsentierten Daten einer statistischen Unschärfe, welche eine streng deterministische Fortrechnung des Systems verhindert.

Ein Beispiel für ein System mit solchen Eigenschaften ist das ökonomische Gefüge eines Staates. Trotz mehrerer Versuche, die es in dieser Richtung gegeben hat, läßt sich eine Volkswirtschaft offenbar nicht zentral planen.

Wie lassen sich dann überhaupt komplexe selbstorganisierende Systeme beeinflussen, oder sogar steuern? Die Interaktionen zwischen den Bestandteilen (oder *Einheiten*) des Systems hängt oft von globalen Parametern ab, die von außen vorgegeben und verändert werden können. Auch die grundsätzliche Struktur des Systems, die bestimmt, welche Einheit mit welchen anderen Einheiten interagiert, läßt sich vorgeben oder zumindest bestimmten Randbedingungen unterwerfen. Auf diese Weise wird ein Rahmen geschaffen, innerhalb dessen die Selbstorganisation abläuft.

Im Beispiel des Wirtschaftsgefüges wären globale Parameter Leitzinssätze oder Steuersätze. Eine gewisse Strukturierung, die bestimmt, wer mit wem benachbart ist und daher interagiert, ergibt sich aus der Topographie des Staates. Für Informationsaustausch gelten dabei (beeinflußt z.B. durch das Internet) ganz andere Nachbarschaftsbeziehungen als für Warenaustausch.

In bestimmten Situationen, etwa, wenn es eine höhere Einsicht erfordert, kann es durchaus sinnvoll sein, von außen in das selbstorganisierende System einzugreifen. Dies kann dazu dienen, das System aus einer Zwangslage zu befreien, die es wegen der lokalen Sichtweise seiner Einheiten nicht selbst verlassen kann. Solche Eingriffe können jedoch unvorhergesehene Konsequenzen haben (sie sind das einzige Werkzeug, das bei einem zentral gesteuerten System zur Verfügung steht) und sollten daher die Ausnahme von der Regel sein. Nach einem derartigen Eingriff re-organisiert sich das System durch eine Reihe von Interaktionen seiner Einheiten. Der Erfolg oder Mißerfolg des Eingriffs stellt sich erst dann heraus, wenn sich wieder ein Gleichgewichtszustand des Systems gebildet hat.

Ich hoffe, daß die Leserin bzw. der Leser an etlichen Stellen des folgenden Textes den Bezug der dargestellten Verfahren zu den obigen Absätzen erkennen wird und damit auch die Mächtigkeit des Prinzips der Selbstorganisation. Das ist das Hauptanliegen dieses Buches.

# Danksagung

An dieser Stelle möchte ich mich bei denen bedanken, die auf die eine oder andere Weise zum Entstehen dieses Buches beigetragen haben.

Herr Professor Dr. Christoph von der Malsburg hat durch seine Arbeiten zur Selbstorganisation bereits in den siebziger Jahren die Grundlagen für etliche der im folgenden vorgestellten Verfahren geschaffen. Diskussionen, die ich als Mitarbeiter seiner Arbeitsgruppe mit ihm führen konnte und Seminare und Vorträge, an denen er beiteiligt war, haben mir viele wertvolle Einsichten verschafft.

Herrn Professor Dr. Hans Jürgen Schneider danke ich für das Angebot, mich bei der Habilitation als inzwischen „Externer“ an der Technischen Fakultät der Universität Erlangen-Nürnberg zu unterstützen. Er war es, der mein Interesse für Graphen geweckt hat, welche in dem vorliegenden Buch eine wichtige Rolle spielen. Von der umfassenden Ausbildung im Bereich Programmiersprachen und effiziente Algorithmen, die ich als Student und Doktorand an seinem Lehrstuhl bekommen habe, konnte ich bei der praktischen Realisierung etlicher Lernverfahren profitieren. Von ihm kam auch die erste Anregung, sich „diese neuronalen Netze“ doch einmal anzusehen.

All meinen Kollegen am Institut für Neuroinformatik der Ruhr-Universität Bochum danke ich für ihre stete Diskussionsbereitschaft und ihren Humor, die beide die Entstehung dieses Buches gefördert haben.

Unserem Systemadministrator Michael Neef sei für technische Unterstützung aller Art gedankt und insbesondere dafür, daß man durch seine Umsicht nahezu den Eindruck bekommen konnte, daß Rechner und lokale Netzwerke nicht ausfallen können.

Ganz besonders aber danke ich meiner Frau Heike. Durch sie habe ich mein seelisches Gleichgewicht immer aufrecht erhalten können. Ihre Liebe, Toleranz und Fürsorge haben mir die Kraft gegeben, die ich für dieses Buch brauchte.

Dresden im Oktober 1998

Bernd Fritzke

# Inhaltsverzeichnis

# Teil I

# Einführung

# Kapitel 1

# Neuronale Netze und Lernen

## 1.1 Neuronale Netze

Die sogenannten „neuronalen Netze“ oder auch „künstlichen neuronalen Netze“ sind eine Gruppe von Verfahren, die schon mehrere Höhen (insbesondere die frühen sechziger Jahre und die späten achtziger Jahre) und Tiefen (die siebziger Jahre und frühen achtziger Jahre) erlebt haben. Vermutlich ist die reißerische Bezeichnung mit ein Grund dafür, daß an dieses Gebiet mehr Hoffnungen geknüpft wurden als an vergleichbare Gebiete aus der Mathematik oder der Statistik. Dies hat immer wieder zu Ernüchterung geführt, wenn die Grenzen einzelner Verfahren deutlich wurden. In den letzten Jahren hat jedoch eine gewisse Konsolidierung stattgefunden, die sich sich darin äußert, daß der Begriff neuronale Netze nicht mehr so sehr in Produktwerbungen auftaucht, sondern eher als eine Informationsverarbeitungstechnik unter anderen akzeptiert wird. Auch ist einiges zur theoretischen Fundierung der Methoden getan worden, wie unter anderem das Erscheinen mehrerer ausgezeichneter Überblicksmonographien zeigt [Haykin, 1994; Bishop, 1995; Ripley, 1996].

Es gibt eine große Anzahl von Verfahren, die den neuronalen Netzen zugeordnet werden, so daß eine präzise Definition des Begriffes nicht leicht ist. Viele Verfahren besitzen jedoch die folgenden Merkmale:

- Eine große Anzahl identischer Berechnungseinheiten wird verwendet.
- Zwischen den Berechnungseinheiten existieren (oftmals gewichtete) Verbindungen.
- Die Parameter der Verfahren werden schrittweise und unter Verwendung von Trainingsdaten adaptiert.
- Die Ausgabewerte der einzelnen Einheiten werden nichtlinear aus den Eingabewerten berechnet.

Der Zusammenhang zwischen neuronalen Netzen und echten Nervensystemen ist bei den meisten Verfahren schwach oder nur auf einem sehr hohen Abstraktionsniveau zu erkennen. Natürlich besteht auch das menschliche Gehirn aus vielen untereinander sehr ähnlichen Zellen, und es gibt eine erhebliche Anzahl von Verbindungen, deren Stärke sich durch Sinneserfahrungen verändern kann („synaptische Plastizität“). Vergleicht man jedoch die komplexen chemischen und elektrischen Vorgänge, die sich an einer einzelnen Synapse abspielen mit der Schlichtheit eines typischen formalen Neurons, so wird schnell klar, daß letzteres nur ein schwaches Zerrbild seines natürlichen Pendants sein kann. Es wird allerdings vielfach angenommen (oder erhofft), daß man beobachtete kollektive Phänomene auf der Ebene von Zellverbänden auch ohne die biologisch detaillierte Modellierung jedes Neurons reproduzieren kann.

Ein zentrales Forschungsgebiet sind dabei Modelle, die die Selbstorganisation von Verbindungsstrukturen im Gehirn erklären. Elementare Betrachtungen über die Anzahl der vorhanden Verbindungen und die dafür verfügbare Informationskapazität der genetischen Erbinformation lassen keinen Zweifel daran, daß der weitaus größte Teil dieser Strukturen nicht genetisch festgelegt wird, sondern sich durch Selbstorganisationsprozesse in frühen Entwicklungsphasen herausbildet.

Neben ihrer Rolle für die Hirnforschung haben die neuronalen Netze in den letzten Jahren einen festen Platz in einer Reihe von Anwendungsgebieten erobert. Bei vielen existierenden Anwendungen geht es um Signalverarbeitung, speziell um Mustererkennung (z.B. akustische Qualitätskontrolle von Motoren) oder Prozeßsteuerung (z.B. Steuerung der Walzkräfte bei der Stahlherstellung). Eine Anwendung im Umweltbereich ist die Prognose von Ozonkonzentrationen in Bodennähe. Auch Anwendungen in nichttechnischen Bereichen existieren, wie z.B. dem Bankwesen, wo neuronale Netze unter anderem für die Kreditwürdigkeitsbeurteilung von Kunden oder die Vorhersage von Aktienindizes eingesetzt werden.

Ein Vorteil, den neuronale Netze gegenüber manch anderen Verfahren haben, ist ihre Adaptivität (auch als Lernfähigkeit bezeichnet). Diese Adaptivität macht es den Netzen möglich, ihre Parameter mithilfe von präsentierten Beispieldaten selbst zu bestimmen und versetzt sie auch in die Lage, sich Veränderungen in den Daten anzupassen. Viele konventionelle Verfahren sind außerdem linear und daher nur bedingt geeignet für inhärent nichtlineare Probleme. Neuronale Netzmodelle sind in der Regel nichtlinear und damit prinzipiell in der Lage nichtlineare Zusammenhänge zu modellieren.

## 1.2 Lernen – momentan oder statistisch?

Genau wie der Begriff „Neuronale Netze“ selbst ist der Begriff „Lernen“ oft mit einer Reihe von Vorstellungen verbunden, die sehr hohe Erwartungen an „lernende Systeme“ begründen. In diesem Buch schließe ich mich der folgenden, eher nüchternen, Definition an:

> *A learning machine is any device whose actions are influenced by past experiences.*
>
> Nils Nilsson
> *Learning Machines*

Man beachte, daß diese Definition keinerlei Implikationen über kognitive Vorgänge macht und bei der „Lernmaschine“ kein Verständnis des gelernten Zusammenhangs oder der Problemstellung voraussetzt. Ein simpler gleitender Durchschnitt, der für eine bestimmte Zeitreihe wie etwa den Deutschen Aktienindex (DAX) jeweils das arithmetische Mittel der letzten 50 Werte ausgibt (siehe Abbildung 1.1), ist nach dieser Definition schon eine Lernmaschine. Zweifellos werden seine Aktionen (die jeweils angezeigten Durchschnittswerte) von seinen Erfahrungen (den bisher aufgetretenen Werten der Zeitreihe) beeinflußt. Ein derartiges System markiert eher das untere Ende der Komplexitätsskala für die im weiteren dargestellten Verfahren. Trotzdem enthält es schon wichtige Merkmale, welche auch etliche der später untersuchten Modelle kennzeichnen. Eines dieser Merkmale ist, daß jeder einzelne neu auftretende Datenwert (jede „Erfahrung“) die Ausgabe (oder auch das „Verhalten“) des Systems nur geringfügig beeinflußt. Dies ist kennzeichnend für das sogenannte *statistische Lernen*, auf das sich dieses Buch konzentriert. Es besteht in der Auswertung einer großen Anzahl von Beispiel- oder Trainingsdaten. Aus diesen werden im Laufe der Zeit bestimmte Parameter extrahiert, die das Verhalten des lernenden Systems beeinflussen. Ein Bereich, bei dem man annehmen kann, daß statistisches Lernen auch beim Menschen eine Rolle spielt, ist das Einüben komplexer Bewegungsabläufe wie etwa beim Tennis. Ein Anfänger im Tennisspiel muß den Weg des Balls extrem oft verfolgen und seine Bewegungen entsprechend korrigieren, bis er es schafft, ihm zugespielte (langsame) Bälle mit einiger Sicherheit ins (gegnerische) Feld zurückzuspielen. Die verbale Erklärung oder auch die einmalige Demonstration durch den Trainer reichen nicht aus, um dieses Niveau zu erreichen. Für einen fortgeschrittenen Spieler gilt ähnliches, wenn er sich das Ziel setzt, ihm schnell zugespielte Bälle präzise auf einen bestimmten Punkt im gegnerischen Feld zu retournieren. Andere Beispiele für überwiegend statistisches Lernen sind das Einüben von Kampfsportarten, Schreibmaschine schreiben oder fahrradfahren.

Es gibt allerdings Fälle, wo vermutlich eine ganz andere Form des Lernens stattfindet, die man als *momentanes Lernen* bezeichnen könnte. So benötigt ein Kind, das sich die Finger an einer Herdplatte verbrennt, in der Regel keine weiteren „Beispiele“ mehr, um sich zu merken, daß es im Umgang mit dem Herd lieber vorsichtig ist. Auch müssen wir eine bestimmte Person nicht mehrere hundert Male sehen, bevor wir sie wiedererkennen können. Im Gegenteil reicht normalerweise eine einzige Begegnung, um die Person das nächste Mal als bekannt einzuordnen (man spricht im Englischen deshalb auch von „one-shot learning“). Gerade die Fähigkeit zur zuverlässigen Gesichtserkennung ist natürlich extrem wichtig für unser gesamtes soziales Leben. Vermutlich nicht aus Zufall ist ein relativ großer Teil der menschlichen Hirnrinde speziell diesem Zweck gewidmet.

Wie schwierig das Erkennen von Gesichtern ist, das wir selbst so gut beherrschen, daß wir es oft gar nicht als Problem wahrnehmen, zeigt sich bei technischen Systemen zur Gesichtserkennung. Einige der Schwierigkeiten, mit denen diese Systeme zu kämpfen haben, um z.B. die drei Fotos in Abbildung 1.2 korrekterweise derselben Person zuzuordnen, sind Variationen von einem Bild zum anderen, beispielsweise in der Beleuchtung, der Pose, der scheinbaren Größe,

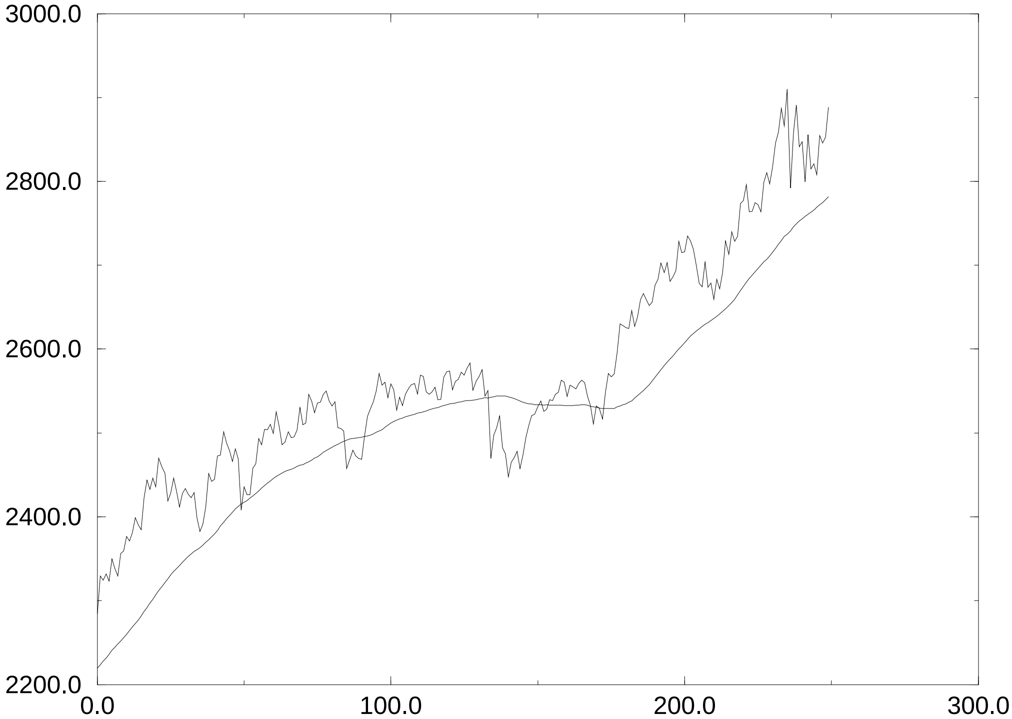



Abbildung 1.1: Aktienindex DAX von 1996 und gleitender 50-Tage Durchschnitt. Die 250 Börsentage des Jahres sind fortlaufend dargestellt.

oder der Frisur. Trotz erheblicher Fortschritte auf diesem Anwendungsgebiet ist die menschliche Fähigkeit zur Gesichtserkennung noch nicht annähernd erreicht und es ist unklar, ob dies jemals der Fall sein wird. Es gibt viele weitere Beispiele von nicht-statistischem Lernen. So reicht uns etwa die aus Büchern erworbene Information, daß ein Tiger gefährlich ist und die Erinnerung an dabei gesehene Abbildungen aus, um im Falle einer tatsächlichen Begegnung mit so einem Tier die Lage als bedrohlich einzustufen. Einem ausschließlich statistisch lernenden System würde diese Erkenntnis erst nach etlichen Begegnungen kommen, sofern es das Glück hätte, diese alle zu überleben.

In diesem Buch soll es ausschließlich um statistisches Lernen gehen. Diese Art von Lernen hat den Vorteil, daß die betrachteten Systeme relativ einfach sind (verglichen z.B. mit dem visuellen System von Säugetieren als einem Beispiel für eher nicht-statistisches Lernen). Außerdem gibt es vergleichbare Verfahren und geeignete Qualitätsmaße aus der Statistik, die quantitative Analysen zulassen. Ein weiterer wichtiger Grund, sich mit statistischem Lernen zu befassen, ist natürlich die hohe Anzahl potentieller Anwendungen in den verschiedensten Sektoren.

## 1.3 Paradigmen statistischen Lernens

Es lassen sich verschiedene statistische *Lernparadigmen* unterscheiden. In manchen Situationen sind die zur Verfügung stehenden Daten nicht mit Klassenbezeichnungen oder zugehörigen Ausgabewerten versehen. In solchen Fällen

a) b) c)

Abbildung 1.2: Drei Bilder der Person Marylin Monroe

spricht man von *unüberwachtem Lernen*, da kein „Lehrer" zur Verfügung steht, der zu einem bestimmten Eingabedatum die korrekte oder erwünschte Ausgabe bereithält. Das Resultat eines Lernprozesses kann in diesen Fällen z.B. eine Aufteilung der Daten in Gruppen einander ähnlicher Datenpunkte sein (Clustering) oder eine Schätzung der Wahrscheinlichkeitsdichte, der die Daten gehorchen. Eine häufige Problemstellung aus diesem Bereich ist auch die *Vektorquantisierung* zum Zwecke der Datenkompression oder -übertragung. Dabei wird für eine aus Vektoren (z.B. Sprachdaten) bestehende Datenmenge eine wesentlich kleinere Vektormenge, das sogenannte Kodebuch, bestimmt, das die ursprünglichen Daten möglichst gut repräsentiert. In der Anwendung wird dann anstelle jedes Datenvektors nur noch der Index des am nächsten liegenden Kodebuchvektors übertragen, was im allgemeinen eine erhebliche Kompression bedeutet (allerdings verbunden mit einem gewissen Informationsverlust).

Ein weiteres Lernparadigma, das jedoch nicht so verbreitet ist wie das unüberwachte Lernen, ist das *Verstärkungslernen* (engl. reinforcement learning). Hierbei geht man davon aus, daß das lernende System fortlaufend Aktionen ausführt, die zunächst ohne jede Rückmeldung bleiben. Von Zeit zu Zeit jedoch bewertet ein „Lehrer" die Gesamtleistung des Systems, z.B. durch ein binäres Signal, das die Werte „gut" und „schlecht" annehmen kann. Das System kann nun diese Information nutzen, um seine internen Parameter zu adaptieren. Eine Schwierigkeit, die dabei auftritt, ist das sogenannte *credit assignment* Problem: es ist prinzipiell unklar, welche der vergangenen Aktionen dazu beigetragen haben, daß die Bewertung ihren augenblicklichen Wert annahm. Ein beliebtes Testproblem für Verstärkungslernen ist das Balancieren einer Stange oder auch mehrerer durch Gelenke verbundener Stangen. Einiges Aufsehen erregte auch das auf Verstärkungslernen beruhende Backgammon-Programm *TD-Gammon* [Tesauro, 1995], welches allein durch Spielen gegen sich selbst die Spielstärke menschlicher Weltklassespieler erreicht hat. Interessanterweise war TD-Gammon noch erheblich stärker als das einige Jahre zuvor entwickelte *Neurogammon*, welches mittels einer Datenbank von Spielzügen trainiert worden war, die vorher von menschlichen Meisterspielern bewertet wurden. Bereits Neurogammon hatte damit einen internationalen Computerwettbewerb für Backgammon gewinnen können [Tesauro, 1990].

Das am weitaus meisten verwendete statistische Lernparadigma schließlich ist das *überwachte Lernen.* Hierbei existiert für eine bestimmte Datenmenge zu jedem Eingabedatum auch die zugehörige Ausgabeinformation. Dies kann im Fall eines Klassifizierungsproblems eine Klassenbezeichnung sein wie etwa „Motor defekt“ oder „Motor nicht defekt“. Bei Regressionsproblemen ist die Ausgabe kontinuierlich und kann z.B. den DAX-Wert des nächsten Tages oder auch den Fettanteil in einer gerade mit Ultraschall durchmessenen Schweinehälfte bedeuten.

In diesem Buch werden sowohl unüberwachtes als auch überwachtes Lernen behandelt, nicht jedoch Verstärkungslernen, da dies von seinem Ansatz und den Problemen, für die es geeignet ist, zu verschieden von den beiden anderen Paradigmen ist.

## 1.4 Notation

Es wird versucht, in diesem Buch auf eine in sich konsistente Notation zurückzugreifen. Mengen bezeichnen wir stets mit kalligraphischen Buchstaben (z.B. $\mathcal{A}, \mathcal{D}$). Vektoren werden in fetter Schrift dargestellt und stellen grundsätzlich Spaltenvektoren dar (z.B. $\mathbf{w}, \boldsymbol{\xi}$). Zeilenvektoren werden als transponiert gekennzeichnet (z.B. $\mathbf{w}^{\mathrm{T}}$).

Die wichtigsten Symbole sind die folgenden:

| Symbol | Bedeutung |
|---|---|
| $c$ | Einheit (auch Neuron oder Knoten) eines vektorbasierten Netzwerks |
| $\mathcal{A}$ | Menge aller Einheiten eines Netzwerks |
| $N$ | Kardinalität von $\mathcal{A}$ (Netzwerkgröße) |
| $n$ | Dimension des Eingaberaumes |
| $m$ | Dimension des Ausgaberaumes/Anzahl der möglichen Klassen |
| $\boldsymbol{\xi}$ | Eingabesignal aus $\mathrm{R}^n$ |
| $\boldsymbol{\zeta}$ | Ausgabesignal aus $\mathrm{R}^m$ |
| $\zeta$ | Ausgabesignal aus $\mathrm{R}$ |
| $\boldsymbol{\zeta}^i$ | $i$-ter Zielvektor aus $\mathrm{R}^m$ |
| $\zeta^i$ | $i$-tes 1-dimensionales Ausgabesignal $\mathrm{R}$ |
| $\boldsymbol{\xi}^i$ | $i$-ter Eingabevektor aus $\mathcal{D}$ |
| $(\boldsymbol{\xi}^i, \boldsymbol{\zeta}^i)$ | (Ein/Ausgabepaar für überwachtes Lernen |
| $\mathcal{D}$ | finite Menge von Eingabevektoren (für unüberwachtes Lernen) oder von Ein-/Ausgabeparen (für überwachtes Lernen) |
| $M$ | Kardinalität von $\mathcal{D}$ (Größe der Datenmenge) |
| $\mathbf{w}_c$ | Referenzvektor der Einheit $c$ |
| $\mathcal{C}$ | Menge der topologischen Verbindungen (Kanten) zwischen den Netzwerkknoten |
| $\mathcal{R}_c$ | Voronoimenge von $c$ |
| $V_c$ | Voronoigebiet der Einheit $c$ |
| $p(\boldsymbol{\xi})$ | Wahrscheinlichkeitsdichte an der Stelle $\boldsymbol{\xi}$ |
| $s(\boldsymbol{\xi})$ | Gewinnereinheit bezüglich des Eingabesignals $\boldsymbol{\xi}$ |
| $E(\cdot)$ | Erwartungswertoperator |
| $E[\cdot]$ | Fehlermaß |
| $\mathcal{N}_c$ | Menge der direkten topologischen Nachbarn der Einheit $c$. |

# Kapitel 2

# Vektorbasierte Neuronale Netze – Grundlagen

Die im folgenden eingeführten Bezeichnungen und Definitionen gelten für *alle* in diesem Buch behandelten Verfahren und sind wichtig für das Verständnis der späteren Kapitel.

## 2.1 Netzwerkstruktur

Vektorbasierte neuronale Netze sind dadurch gekennzeichnet, daß ihr jeweiliger Zustand wesentlich durch eine Anzahl von $n$-dimensionalen Vektoren beschreibbar ist, wobei $n$ die Dimension des Eingaberaums ist. Diese Vektoren werden als *Referenzvektoren* bezeichnet. Jedes *Netzwerk* besteht aus einer Menge

$$\mathcal{A} = \{c_1, \ldots, c_N\}$$

von $N$ formalen Neuronen (auch *Einheiten* genannt). Jeder Einheit $c \in \mathcal{A}$ ist der Referenzvektor

$$\mathbf{w}_c \in \mathrm{R}^n \tag{2.1}$$

zugeordnet, der die *Position* dieser Einheit im Eingaberaum $\mathrm{R}^n$ darstellt. Eine andere Interpretation von $\mathbf{w}_c$ wäre der *Prototyp* für die Eingabesignale, durch welche die Einheit $c$ maximal aktiviert wird.

Es gibt eine (möglicherweise leere) Menge

$$\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$$

von *Verbindungen* zwischen den Einheiten des Netzwerks. Diese Verbindungen sind symmetrisch, d.h. es gilt

$$(i,j) \in \mathcal{C} \iff (j,i) \in \mathcal{C}.$$

Im Gegensatz zu den Verbindungen, die bei mehrlagigen Perzeptrons verwendet werden (siehe dazu Kapitel 4), sind diese Verbindungen nicht gewichtet. Ihr einziger Zweck ist die Definition einer topologischen Struktur, die einige Modelle verwenden, um die Adaption des Gewinners (siehe unten) auf in der Struktur benachbarte Einheiten auszudehnen.

Für eine Einheit $c$ bezeichnen wir mit $\mathcal{N}_c$ die Menge ihrer direkten topologischen Nachbarn:

$$\mathcal{N}_c = \{i \in \mathcal{A} | (c,i) \in \mathcal{C}\} \,. \tag{2.2}$$

## 2.2 Trainingsdaten

Die Trainingsdaten, die dem Netzwerk präsentiert werden, liegen entweder vor als *kontinuierliche Wahrscheinlichkeitsverteilung* mit der Wahrscheinlichkeitsdichte

$$p(\boldsymbol{\xi}),\ \boldsymbol{\xi} \in \mathrm{R}^n$$

oder als *endliche Menge*

$$\mathcal{D} = \{\boldsymbol{\xi}^1, \ldots .\boldsymbol{\xi}^M\},\ \boldsymbol{\xi}^i \in \mathrm{R}^n. \tag{2.3}$$

Den ersteren Fall kann man auch interpretieren als unendlich große Datenmenge. Es gibt einige Verfahren (z.B. LBG, siehe Abschnitt 6.1), die nur für eine endliche Datenmenge anwendbar sind.

## 2.3 Der Gewinner

Für ein gegebenes Eingabesignal $\boldsymbol{\xi}$ ist der *Gewinner* $s(\boldsymbol{\xi})$ definiert als diejenige Einheit, die den nächstgelegenen Referenzvektor hat:

$$s(\boldsymbol{\xi}) = \arg\ \min_{c\in\mathcal{A}} \|\boldsymbol{\xi} - \mathbf{w}_c\|. \tag{2.4}$$

Als Vektornorm $\|\cdot\|$ wird die Euklidische Vektornorm verwendet. Für jedes Eingabesignal $\boldsymbol{\xi}$ existiert definitionsgemäß genau ein Gewinner. Falls mehrere Einheiten $c_i$ aus $\mathcal{A}$ den gleichen minimalen Abstand zu $\boldsymbol{\xi}$ besitzen, wird die mit dem kleinsten Index ausgewählt und zum Gewinner erklärt. Dafür nehmen wir auf der Indexmenge eine beliebige Ordnung an. Eine andere prinzipielle Möglichkeit wäre, für Referenzvektoren und Eingabesignale generelle Positionen im $\mathrm{R}^n$ zu postulieren und die obengenannte Situation damit explizit auszuschließen.

Zur Vereinfachung der Notation bezeichnen wir den Gewinner in bestimmten Fällen einfach nur mit $s$ (unter Weglassung der Abhängigkeit von $\boldsymbol{\xi}$). Wenn nicht nur der Gewinner, sondern auch weitere Einheiten in der Nähe von $\boldsymbol{\xi}$ von Interesse sind, bezeichnen wir mit $s_i$ die $i$-nächste Einheit ($s_1$ ist der Gewinner, $s_2$ ist die zweitnächste Einheit, u.s.w.).

## 2.4 Voronoi-Tessellation und Delaunay-Triangulation

Für viele Aspekte vektorbasierter neuronaler Netze sind zwei fundamentale und eng miteinander verbundene geometrische Konzepte von Bedeutung: Die *Voronoi-Tessellation* und die *Delaunay-Triangulation* (siehe Abbildung 2.1). Wenn eine Menge von Vektoren $\mathbf{w}_1, \ldots, \mathbf{w}_N$ in $\mathrm{R}^n$ gegeben ist, dann ist das

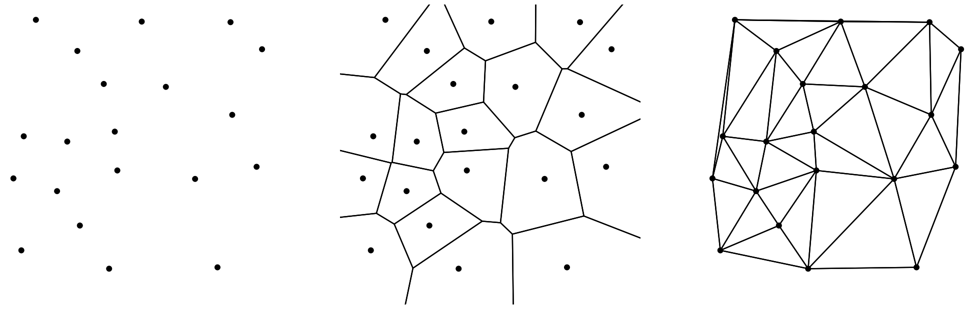



Abbildung 2.1: a) Punktmenge in $\mathrm{R}^2$, b) korrespondierende Voronoi-Tessellation, c) korrespondierende Delaunay-Triangulation.

*Voronoigebiet* $V_i$ eines Elementvektors $\mathbf{w}_i$ definiert als die Menge derjenigen Punkte aus $\mathrm{R}^n$, für die $\mathbf{w}_i$ der nächste Vektor ist:

$$V_i = \{\boldsymbol{\xi} \in \mathrm{R}^n |\, i = \arg\, \min_{j \in \{1,\ldots,N\}} \|\boldsymbol{\xi} - \mathbf{w}_j\|\}.$$

Wie schon bei der Definition des Gewinners ordnen wir Datenvektoren, die mehrere nächste Referenzvektoren haben, dem Referenzvektor zu, der den kleinsten Index besitzt. Jedes Voronoigebiet ist eine konvexe Region des $\mathrm{R}^n$, d.h. es gilt

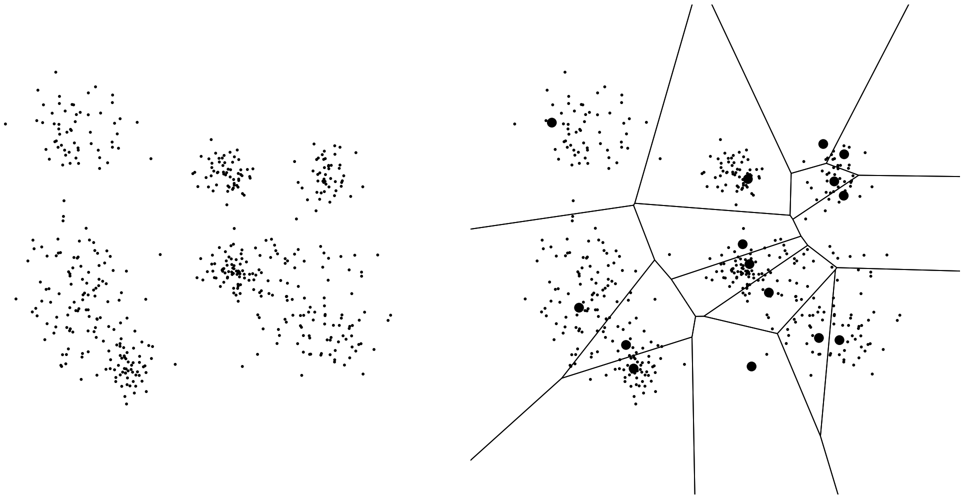
$$(\boldsymbol{\xi}^1 \in V_i \wedge \boldsymbol{\xi}^2 \in V_i) \quad \Rightarrow \quad (\boldsymbol{\xi}^1 + \alpha(\boldsymbol{\xi}^2 - \boldsymbol{\xi}^1) \in V_i) \quad \text{für } \alpha \in \mathrm{R}, 0 \leq \alpha \leq 1.$$

Die durch alle Voronoigebiete gebildete Partition des $\mathrm{R}^n$ wird *Voronoi-Tessellation* genannt (manchmal auch Dirichlet-Tessellation). Effiziente Verfahren zur Berechnung der Voronoi-Tessellation sind nur für zwei-dimensionale Daten bekannt [Preparata & Shamos, 1990; Mehlhorn, 1984]. Das Konzept selbst ist jedoch übertragbar auf beliebige Dimensionen.

Verbindet man alle Paare von Vektoren, deren Voronoigebiete eine gemeinsame Kante aufweisen, so erhält man einen speziellen Graphen, die Delaunay-Triangulation. Wie jede $n$-dimensionale Triangulation besteht auch die Delaunay-Triangulation aus $n$-dimensionalen Simplices (jeder Simplex bestehend aus $n+1$ Knoten mit vollständiger Verbindung). Sie ist aber gegenüber anderer Triangulationen ausgezeichnet dadurch, daß sie die einzige Triangulation ist, bei der die umschließende Hypersphere für jedes Simplex nur die Ecken des Simplex enthält und keine anderen Elemente der gegebenen Vektormenge. Von Omohundro [1990] wurde außerdem bewiesen, daß die Delaunaytriangulation diejenige Triangulation ist, die optimal für Funktionsinterpolation geeignet ist. Eines der im folgenden dargestellten Verfahren, die Hebb'sche Wettbewerbslernregel (Abschnitt 8.2), generiert einen Untergraph der Delaunaytriangulation der im wesentlichen beschränkt ist auf die Gebiete des $\mathrm{R}^n$, die Eingabedaten enthalten, wo also gilt, daß $p(\boldsymbol{\xi}) > 0$.

Wir definieren zur Vereinfachung einiger Formulierungen das *Voronoigebiet* $V_c$ *einer Einheit* $c$, $c \in \mathcal{A}$, als das Voronoigebiet ihres Referenzvektors:

$$V_c = \{\boldsymbol{\xi} \in \mathrm{R}^n | s(\boldsymbol{\xi}) = c\} \tag{2.5}$$



a) Eingabedaten b) Voronoitessellation

Abbildung 2.2: Eine Menge $\mathcal{D}$ von Eingabedaten ist dargestellt (a) und die Voronoitessellation, die sich ergibt, wenn eine Menge von Referenzvektoren im Eingaberaum positioniert wird (b). Für jede Einheit $c \in \mathcal{A}$ enthält die Voronoimenge $\mathcal{R}_c$ genau die Datenpunkte aus $\mathcal{D}$, die in dem Voronoigebiet $V_c$ der Einheit $c$ liegen.

Weiterhin bezeichnen wir im Falle einer endlichen Datenmenge $\mathcal{D}$ die Menge $\mathcal{R}_c$ aller Datenvektoren aus $\mathcal{D}$, für die die Einheit $c$ Gewinner ist, als *Voronoimenge* von $c$:

$$\mathcal{R}_c = \{\boldsymbol{\xi} \in \mathcal{D} | s(\boldsymbol{\xi}) = c\}. \tag{2.6}$$

Die Voronoimenge von $c$ enthält genau diejenigen Datenpunkte aus $\mathcal{D}$, die im Voronoigebiet $V_c$ von $c$ liegen. Da es für jeden Datenvektor einen eindeutigen Gewinner gibt, bilden die Voronoimengen eine Partition von $\mathcal{D}$, d.h.,

$$\bigcup_{c \in \mathcal{A}} \mathcal{R}_c = \mathcal{D}, \tag{2.7}$$

und

$$\mathcal{R}_i \cap \mathcal{R}_j = \emptyset \quad \text{für } i, j \in \mathcal{A} \text{ und } i \neq j.$$

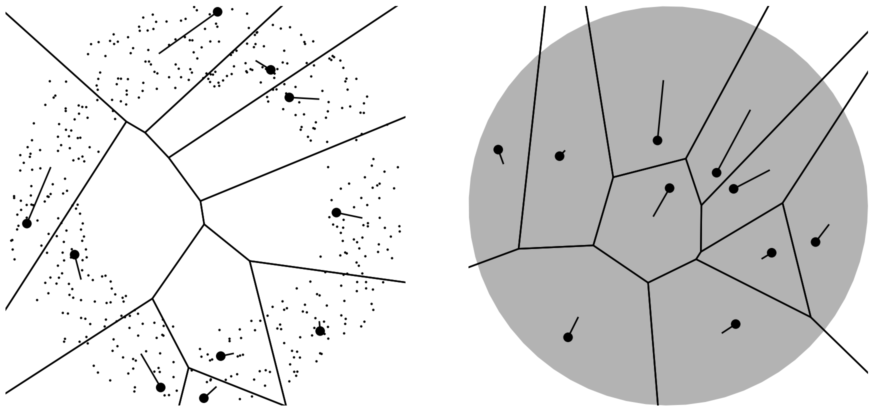
In Abbildung 2.2 ist für eine gegebene Datenmenge $\mathcal{D}$ und eine Menge von Referenzvektoren die resultierende Partition von $\mathcal{D}$ in Voronoimengen gezeigt.

Voronoigebiet und Voronoimenge sind wichtig für die Charakterisierung des Erwartungswertes der Signale, für die eine bestimmte Einheit $c$ Gewinner ist. Im Falle einer kontinuierlichen Wahrscheinlichkeitsverteilung ist dieser bedingte Erwartungswert, welcher gleichzeitig der *Zentroid* oder Massenschwerpunkt der Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ im Voronoigebiet $V_c$ ist, gegeben als

$$E(\boldsymbol{\xi} | \boldsymbol{\xi} \in V_c) = \frac{\int_{V_c} p(\boldsymbol{\xi}) \boldsymbol{\xi} d\boldsymbol{\xi}}{\int_{V_c} p(\boldsymbol{\xi}) d\boldsymbol{\xi}}$$

und im Falle einer endlichen Datenmenge $\mathcal{D}$ als

$$E(\boldsymbol{\xi} | \boldsymbol{\xi} \in \mathcal{R}_c) = \frac{\sum_{\boldsymbol{\xi} \in \mathcal{R}_c} p(\boldsymbol{\xi}) \boldsymbol{\xi}}{\sum_{\boldsymbol{\xi} \in \mathcal{R}_c} p(\boldsymbol{\xi})}. \tag{2.8}$$



a) endliche Datenmenge　　b) kontinuierliche W-Verteilung

Abbildung 2.3: Bedingte Erwartungswerte für das nächste Eingabesignal (Zentroide der Wahrscheinlichkeitsdichte im jeweiligen Voronoigebiet). Die fett gezeichneten Punkte sind die augenblicklichen Referenzvektoren. Die angehängten Segmente deuten auf den bedingten Erwartungswert für das nächste Eingabesignal der jeweiligen Einheit. a) Endliche Datenmenge $\mathcal{D}$ mit gleichwahrscheinlichen Elementen, generiert aus einer Wahrscheinlichkeitsverteilung mit ringförmiger Struktur. Die Erwartungswerte sind exakt berechnet gemäß Gleichung (2.9). b) Kontinuierliche Gleichverteilung in einem kreisförmigen Gebiet. Die Erwartungswerte sind angenähert durch Auswertung von 100.000 zufällig generierten Eingabesignalen.

Für den oft gegebenen Fall, daß alle Elemente der Menge $\mathcal{D}$ gleichwahrscheinlich sind, vereinfacht sich Gleichung (2.8) zu

$$E(\boldsymbol{\xi}|\boldsymbol{\xi} \in \mathcal{R}_c) = \frac{1}{|\mathcal{R}_c|} \sum_{\boldsymbol{\xi} \in \mathcal{R}_c} \boldsymbol{\xi}. \tag{2.9}$$

Abbildung 2.3 illustriert an zwei Beispielen die bedingten Erwartungswerte für das jeweils nächste Eingabesignal.

Die genannten Erwartungswerte geben bei den später behandelten Lernverfahren die zu erwartende Richtung der Adaption an. Bei einigen Verfahren (LBG und LBG-U) werden die Erwartungswerte auch direkt verwendet als neue Positionen der Referenzvektoren.

## 2.5 Aktivierungsfunktionen

Manchmal ist es sinnvoll, eine weniger harte Entscheidung zu treffen, als sie das Feststellen eines einzigen Gewinners darstellt. In solchen Fällen wird oft für jede Einheit des vektorbasierten neuronalen Netzwerks eine kontinuierliche *Aktivierung* berechnet, die von der Entfernung zwischen Eingabevektor und Referenzvektor abhängt und in der Regel auch von weiteren Parametern. Der Bereich des Eingaberaumes, aus dem diejenigen Signale stammen, welche bei einem Neuron eine (wesentliche) Aktivierung hervorrufen, wird *Aktivierungs-*

*region*[1] genannt. Eine oft verwendete Aktivierungsfunktion bei vektorbasierten neuronalen Netzen ist die Gauß'sche Glockenkurve. Die Aktivierung für die Einheit $c$ hat dabei die Form

$$\mathrm{D}_c(\boldsymbol{\xi}) = \exp\left(-\frac{\|\boldsymbol{\xi} - \mathbf{w}_c\|^2}{2\sigma_c^2}\right). \tag{2.10}$$

Der Breitenparameter $\sigma_c$ gibt die Ausdehnung der Glockenkurve für die Einheit $c$ an. Ein gegebenes Eingabesignal $\boldsymbol{\xi}$ bewirkt für alle Einheiten des Netzwerks eine von Null verschiedene Aktivierung (es gelte stets $\sigma > 0$). Allerdings sind in der Regel nur wenige Einheiten stark aktiviert.

Wenn alle Einheiten den *gleichen* Breitenparameter $\sigma$ haben, dann ist von zwei beliebigen Einheiten diejenige stärker aktiviert, die näher am Eingabesignal liegt. In diesem Fall ist der Bereich des Eingaberaums, für den eine bestimmte Einheit $c$ am stärksten von allen aktiviert ist, im wesentlichen[2] ihr Voronoigebiet.

Variiert der Parameter $\sigma$ von Einheit zu Einheit, ist nicht mehr allein die Distanz zwischen Eingabesignal $\boldsymbol{\xi}$ und Referenzvektor $\mathbf{w}_c$ entscheidend sondern vielmehr der Quotient aus dieser Distanz und dem Wert $\sigma_c$. Eine einfache Beziehung zu den Voronoigebieten existiert dann nicht mehr.

---

[1] Das Äquivalent aus der Biologie wäre das *rezeptive Feld* einer Nervenzelle, also z.B. der Bereich der Retina (Netzhaut), auf den ein Lichtreiz fallen muß, um ein bestimmtes Neuron in der primären Sehrinde zu aktivieren.

[2] Für $\sigma \to 0$ konvergiert der Bereich, in dem eine Einheit $c$ mehr als alle anderen Einheiten aktiviert ist, genau gegen ihr Voronoigebiet $V_c$.

# Kapitel 3

# Datenvorverarbeitung

Vektorbasierte neuronale Netze setzen die Existenz einer $n$-dimensionalen Wahrscheinlichkeitsverteilung

$$p(\boldsymbol{\xi}), \boldsymbol{\xi} \in \mathrm{R}^n$$

oder einer endlichen Datenmenge aus $n$-dimensionalen Vektoren

$$\mathcal{D} = \{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^M\},\ \boldsymbol{\xi}^i \in \mathrm{R}^n \tag{3.1}$$

voraus. Diese Trainingsdaten fallen in der Regel nicht „vom Himmel“, sondern entstehen durch einen Vorverarbeitungsprozeß aus einer – oft viel höherdimensionaleren – Wahrscheinlichkeitsverteilung

$$p(\mathbf{x}), \mathbf{x} \in \mathrm{R}^l$$

oder Datenmenge

$$\mathcal{D}_{\mathrm{orig}} = \{\mathbf{x}^1, \ldots, \mathbf{x}^M\}, \mathbf{x}^i \in \mathrm{R}^l,$$

die man als *Rohdaten* bezeichnen kann. Diese Rohdaten stammen direkt von Sensoren wie Kameras, Mikrofonen, Druckmeßgeräten oder Thermometern oder von anderen Datenquellen wie etwa Datenbanken oder Börseninformationsdiensten. Rohdaten können kontinuierlich oder diskret sein und sind mit verschiedensten Maßeinheiten behaftet.

Die Vorverarbeitung dieser Daten ist oft von entscheidender Bedeutung für den Erfolg der weiteren Verarbeitungsschritte; dort gemachte Fehler können in der Regel nicht mehr korrigiert werden. Nur in Ausnahmefällen kann man auf eine Vorverarbeitung gänzlich verzichten und die Rohdaten unverändert als Trainingsdaten verwenden.

Im weiteren Verlauf dieses Buches wird davon ausgegangen, daß eine geeignete Vorverarbeitung bereits durchgeführt wurde und die zur Verfügung stehenden Daten das Resultat dieser Vorverarbeitung sind. Zur Erläuterung werden im folgenden jedoch exemplarisch einige grundlegende Vorverarbeitungstechniken vorgestellt und diskutiert. Die Vorverarbeitung für ein reales Problem beinhaltet normalerweise eine Kombination mehrerer dieser (und weiterer) Techniken.

## 3.1 Skalierung

Oft ist eine Skalierung der einzelnen Komponenten der Datenvektoren sinnvoll. Abhängig von der Bedeutung der einzelnen Komponenten und den verwendeten Maßinheiten können sie völlig unterschiedliche Wertebereiche haben. Bei einer Anwendung aus dem Bereich der Medizin kann z.B. jeder Vektor $\mathbf{x}$ die Daten eines erwachsenen Patienten enthalten. Hier könnten dann einzelne Komponenten für das Gewicht in Kilogramm stehen und die Anzahl der Thrombozyten (Gerinnungsparameter) pro Mikroliter Blut. Der normale Wertebereich für das Gewicht wäre 50-100 und der für die Thrombozyten 150000-350000. Ohne Vorverarbeitung bildet jede dieser Komponenten eine der Koordinaten des Vektorraumes, in dem vektorbasierte neuronale Netze Distanzen messen. Im genannten Fall würde die Komponente „Gewicht“ von der Komponente „Thrombozyten“ fast völlig dominiert werden. Läßt man die Daten unverändert, gibt aber das Gewicht in Milligramm an, so kehrt sich die Lage um und die Gewichtskomponente ist plötzlich dominierend.

Offenbar ist es sinnvoll, sich von eher zufälligen Einflüssen wie der Wahl der Maßeinheiten unabhängig zu machen. Ein übliches Mittel ist eine Skalierung aller Komponenten derart, daß sie eine gemeinsame Varianz haben, z.B. den Wert Eins. Meist wird aus Darstellungsgründen zusätzlich noch von jeder Komponente der Mittelwert abgezogen, so daß die transformierten Daten in jeder Komponente den Mittelwert Null und die Varianz Eins haben. Für die $i$-te Komponente $x_i$ des Datenvektors $\mathbf{x}$ ergibt sich so die Transformation

$$x'_i = \frac{x_i - \mu_i}{\sigma_i},$$

wobei $\mu_i$ der Mittelwert und $\sigma_i$ die Standardabweichung der $i$-ten Komponente aller Vektoren ist (im Falle einer endlichen Datenmenge) oder einer hinreichend großen Stichprobe (im Falle einer kontinuierlichen Verteilung).

Manchmal existiert Vorwissen über die Daten, welches besagt, daß einzelne Komponenten eine größere Bedeutung haben als andere. In solchen Fällen können die jeweiligen normierten Komponenten noch mit entsprechenden Faktoren $f_i$ multipliziert werden:

$$x'_i = f_i \frac{x_i - \mu_i}{\sigma_i}$$

Vektorbasierte neuronale Netze sind sehr empfindlich bezüglich der Skalierung der Daten, da sie Distanzen in dem ihnen vorgegebenen Vektorraum messen. Es sei angemerkt, daß andere Netzwerkarchitekturen (z.B. mehrschichtige Perzeptrons) die Fähigkeit besitzen, durch Wahl geeigneter Gewichte selbst eine Skalierung der Daten vorzunehmen. Somit wäre eine Vorskalierung der Daten nicht mehr unbedingt erforderlich.

## 3.2 Ausblenden von Komponenten

Es müssen nicht alle Komponenten eines ursprünglichen Vektors $\mathbf{x}_i$ in den korrespondierenden Datenvektor $\boldsymbol{\xi}^i$ übernommen werden. Eine kleinere Anzahl von

Komponenten verringert den Berechnungsaufwand und ist daher anzustreben. Allerdings müssen die reduzierten Daten noch die Lösung des gegebenen Problems zulassen. Die optimale Unterauswahl von Komponenten zu treffen ist ein extrem schwieriges Problem, da es bei $l$-dimensionalen ursprünglichen Datenvektoren eine sehr große Anzahl ($2^l$) von möglichen Untermengen dieser Komponenten gibt. *A priori* Wissen kann hier helfen, eine Auswahl zu treffen. Manchmal wird vor der Ausblendung von Komponenten eine Koordinatentransformation durchgeführt (siehe Abschnitt 3.4).

## 3.3 Merkmalsextraktion

Eine weitere Möglichkeit, die Anzahl der Koordinaten zu reduzieren, ist die Berechnung von linearen oder nichtlinearen Kombinationen der Komponenten von $\mathbf{x}_i$, sogenannten *Merkmalen*. Anstelle der ursprünglichen Daten werden dann nur noch diese Merkmale weiterverwendet. Ein Beispiel aus dem Bereich der Bildverarbeitung ist in Abbildung 3.1 dargestellt. Die dort verwendeten sogenannten Gabor-Filter sind richtungs- und frequenzsensitiv und sind in ähnlicher Form im visuellen Kortex von Katzen nachgewiesen worden [Jones & Palmer, 1987]. Die komplette Vorverarbeitung des Gesichtserkennungssystems des Instituts für Neuroinformatik an der Ruhr-Universität Bochum basiert auf Gaborfiltern[1]. Es gibt ein große Vielzahl anderer möglicher Filter (z.B. Gradientenfilter), die zur Kantendetektion eingesetzt werden können oder auch Transformationen, wie die diskrete Fouriertransformation, die diskrete Kosinustransformation oder die Hadamardtransformation. Besonders oft wird die im folgenden Absatz beschriebene Hauptkomponentenanalyse mit anschließender Koordinatentransformation sowie Ausblendung einzelner Komponenten verwendet.

## 3.4 Hauptkomponentenanalyse

Als Hauptkomponentenanalyse (englisch principal component analysis, PCA) wird ein Verfahren zur Bestimmung derjenigen zueinander orthogonalen Raumrichtungen bezeichnet, entlang derer eine gegebene Datenmenge (hier $\mathcal{D}_{\mathrm{orig}}$) die größten Varianzen aufweist. Diese Richtungen werden als Hauptkomponenten bezeichnet und sie entsprechen genau den Eigenvektoren, die zu den Eigenwerten $\lambda_i$ der Kovarianzmatrix

$$\mathbf{\Sigma} = \frac{1}{M} \sum_{\mathbf{x} \in \mathcal{D}_{\mathrm{orig}}} (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}}$$

[1]Das Bochumer Gesichtserkennungssystem verwendet elastische markierte Graphen zum Finden und Vergleichen von Gesichtern [Vorbrüggen et al., 1990; Wiskott, 1995] und stellt eine algorithmische Variante der Dynamic Link Architecture [Malsburg, 1981] dar. Bei internationalen Vergleichstests hat dieses System bisher mehrfach hervoragend abgeschnitten. Es ist ein Beispiel für nicht-statistisches Lernen, da Gesichter nur einmal präsentiert werden müssen, um wiedererkannt zu werden. In die komplexe Architektur des Systems ist an verschiedenen Stellen a priori Wissen über das spezielle zu lösende Problem eingegegangen, um die bestmögliche Leistungsfähigkeit zu erreichen. Man kann anmerken, daß auch das visuelle System von Menschen und Säugetieren das komplexeste und am stärksten *a priori* festgelegte neuronale Netzwerk des jeweiligen Organismus ist.

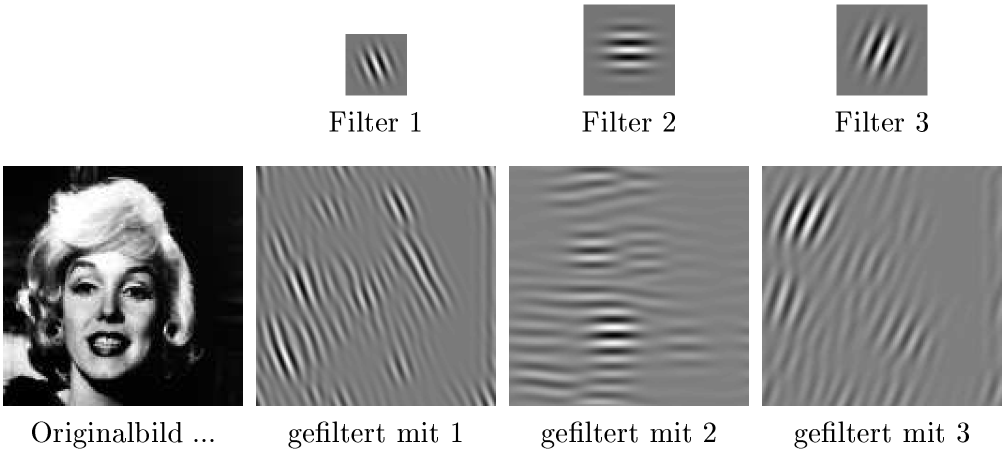
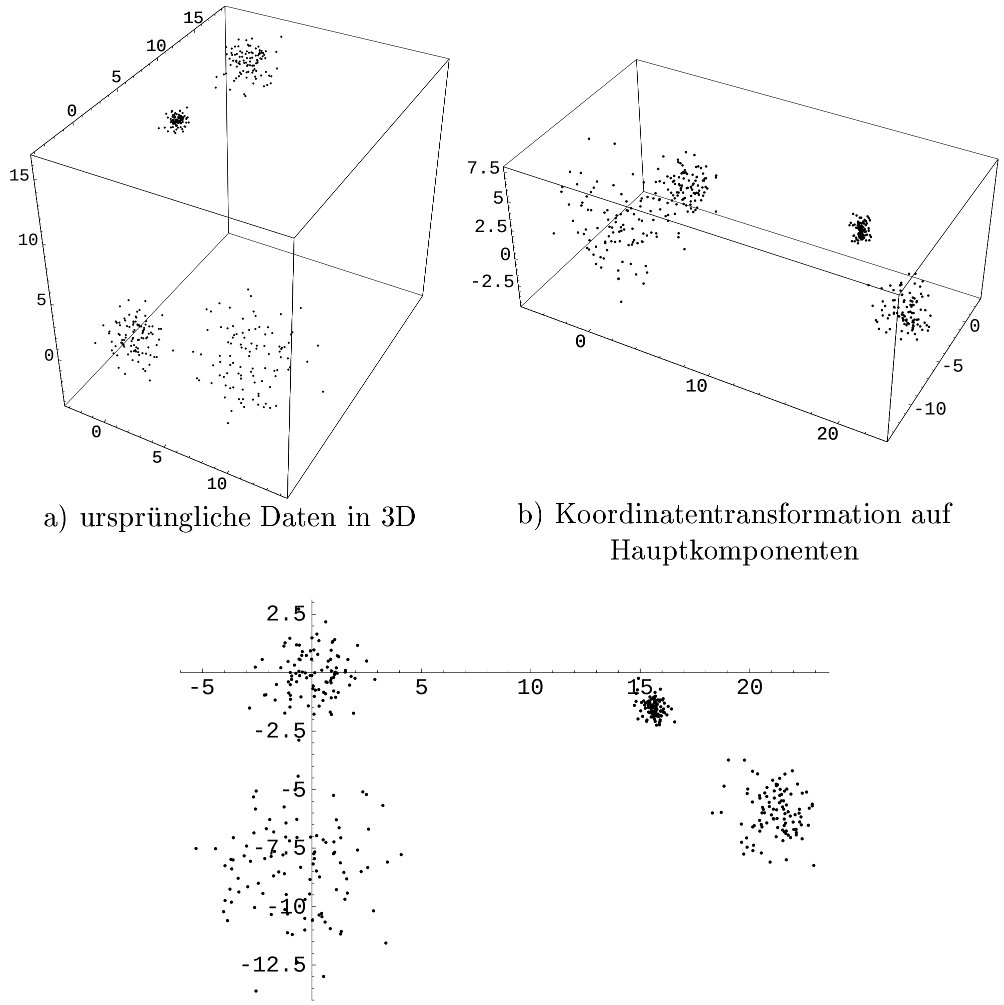



Abbildung 3.1: Beispiel für Bildvorverarbeitung. Die obere Reihe zeigt drei lineare Filter (Gaborfilter) von unterschiedlicher Orientierung und Frequenz. Die untere Reihe zeigt ein Originalbild und das Ergebnis der Konvolution der Filter mit diesem Bild. Man kann erkennen, daß die Filter besonders gut auf Bildinhalte bestimmter Richtung und Ortsfrequenz ansprechen (wie dies auch bestimmte richtungssensitive Zellen im visuellen Kortex von Säugetieren tun) und derartige Informationen aus dem ursprünglichen Bild extrahieren. Ein Bildpunkt des Originalbilds kann anschliessend durch einen Vektor von Filterantworten an dieser Stelle repräsentiert werden.

der gegebenen Datenmenge gehören ($M$ ist die Anzahl der Elemente in $\mathcal{D}_{\text{orig}}$ und $\boldsymbol{\mu}$ ist der Mittelwert aller Elemente). Jeder Eigenwert $\lambda_i$ gibt die Varianz entlang der Richtung seines zugeordneten Eigenvektors an. Bei vielen hochdimensionalen Datensätzen sind etliche der Eigenwerte extrem klein, was bedeutet, daß sich die $l$-dimensionalen Daten im wesentlichen in einem linearen Unterraum des $\mathrm{R}^l$ befinden, der z.B. die Dimension $n$, mit $n < l$, hat. Dies kann ausgenutzt werden, indem man eine lineare Koordinatentransformation macht so, daß die ersten $n$ Koordinatenrichtungen mit den Eigenvektoren zusammenfallen, die zu den $n$ größten Eigenwerten gehören. Die anderen $(l - n)$ Koordinaten werden vernachlässigt (siehe Abbildung 3.2). Dadurch wird die Dimensionalität der Daten reduziert bei minimaler Veränderung im Sinne eines quadratischen Fehlerkriteriums. Bei der praktischen Durchführung der Hauptkomponentenanalyse kann es bei manchen Verfahren (z.B. dem Gram-Schmidt'schen Orthogonalisierungsverfahren) zu numerischen Problemen kommen, da die Kovarianzmatrix $\boldsymbol{\Sigma}$ singulär sein kann. Eine empfehlenswerte Methode, die solche Probleme vermeidet, ist die sogenannte Singular-Value-Decomposition [Press et al., 1992, Kapitel 2.6].

Obwohl die Hauptkomponentenanlyse oft zur Dimensionsreduktion von Daten verwendet wird, ist es sinnvoll, sich der Einschränkungen dieser Technik bewußt zu sein. Insbesondere ist die Hauptkomponentenanlyse eine *lineare* Technik und deswegen nur in der Lage, die *linearen* Unterräume zu finden, in denen sich die gegebenen Daten im wesentlichen befinden. Solche linearen Unterräume entstehen durch lineare Korrelationen zwischen einzelnen Datenkomponenten.




a) ursprüngliche Daten in 3D

b) Koordinatentransformation auf Hauptkomponenten

c) Reduktion auf 2D

Abbildung 3.2: Datenreduktion mit Hilfe der Hauptkomponentenanalyse. Von den ursprünglichen Daten (a) werden die Eigenvektoren der Kovarianzmatrix berechnet. Multplikation der Daten mit der Matrix der Eigenvektoren ergibt transformierte Daten, deren Koordinatenvektoren mit den Eigenvektoren übereinstimmen (b). Durch Weglassen der Koordinaten (hier nur einer einzigen), die zu den kleinsten Eigenwerten korrespondieren, wird eine Datenreduktion bei minimalem Informationsverlust erreicht (c).

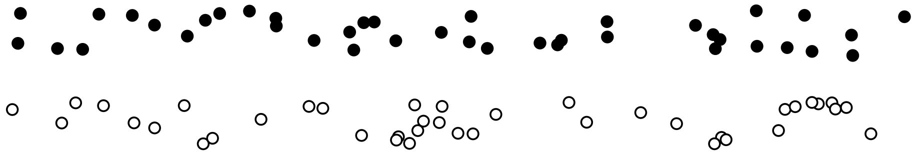


Abbildung 3.3: Datensatz für ein Klassifizierungsproblem mit zwei Klassen. Die größte Varianz der Daten ist entlang der Horizontalen, was eine Hauptkomponentenanalyse auch detektieren würde. Eine Beschränkung auf die horizontale Koordinate würde jedoch das vorliegende Klassifizierungsproblem gerade besonders kompliziert machen, da die Klassen sich bei diesem Datensatz nur entlang der Vertikalen einfach voneinander trennen lassen.

Sind die Korrelationen zwischen Komponenten des ursprünglichen Datenvektors $\mathbf{x}$ jedoch nichtlinear, entstehen nichtlineare (gekrümmte) Unterräume, die die Daten beherbergen. Diese können von der Hauptkomponentenanalyse nicht detektiert werden.

Ein weiteres mögliches Problem hängt damit zusammen, daß die Hauptkomponentenanalyse eine unüberwachte Technik ist in dem Sinne, daß sie bei markierten Daten, die etwa für jeden Datenvektor $\mathbf{x}$ eine Klassenbezeichnungen mitführen, die Markierungen (oder Ausgabewerte) ignoriert und nur mit den Datenvektoren selbst arbeitet. Die dann oft folgende Beschränkung auf die Datenrichtungen mit größter Varianz impliziert die Annahme, daß die anderen Richtungen für die gestellte Aufgabe (z.B. Klassifizieren von neuen Daten) nicht so relevant sind. Es ist sehr leicht, sich Datensätze vorzustellen, bei denen diese Annahme verletzt ist, bei denen also Komponenten mit geringer Varianz Informationen enthalten, die für die Klassifikation sehr wesentlich sind. Eine Weglassung dieser Komponenten nach einer Hauptkomponentenanalyse kann im Einzelfall das gestellte Problem unlösbar machen. Ein schematisches Beispiel ist in Abbildung 3.3 dargestellt.

## 3.5 Wandlung der Datendarstellung

Für eine gegebene Information gibt es stets unterschiedliche Arten der Darstellung, die – obwohl prinzipiell äquivalent – unterschiedlich gut geeignet sein können für die beabsichtigte Verarbeitung. Bei einem Klassifizierungsproblem mit $m$ verschiedenen Klassen mögen etwa die Daten vorliegen als $n$-dimensionale Eingabevektoren mit einer Klassenbezeichnung, die als Zahl aus dem Bereich 1 bis $m$ gegeben ist. Im allgemeinen ist es nicht ratsam, diese Kodierung so zu belassen und von einem adaptiven Verfahren die Ausgabe eines Wertes von 1 bis $m$ zu verlangen. Dies würde nämlich implizieren, daß die Klasse 1 der Klasse 2 ähnlicher ist als z.B. der Klasse 6. Ist dies nicht der Fall, sollte man unbedingt eine Kodierung wählen, die sicherstellt, daß in der verwendeten Metrik je zwei Klassenbezeichnungen den gleichen „Abstand“ voneinander haben. Bei Klassifizierungsproblemen mit $m$ Klassen bietet sich eine 1-aus-$m$ Kodierung an, die die Klassenbezeichnung $k$ als einen $m$-dimensionalen Vektor darstellt, der an der $k$-ten Stelle eine Eins und sonst nur Nullen hat. Bezüglich der Euklid'schen

Vektornorm haben dann je zwei Klassenbezeichnungen den gleichen Abstand zueinander ($\sqrt{2}$). Sollte jedoch Vorwissen existieren über die Verwandtschaft verschiedener Klassen zueinander, kann durchaus eine Kodierung gewählt werden, die dieses Vorwissen widerspiegelt. Dadurch wird es wahrscheinlicher, daß bei Fehlklassifizierungen zumindest eine „ähnliche“ Klasse als Ergebnis geliefert wird.

# Kapitel 4

# Abgrenzung zu mehrschichtigen Perzeptrons

An dieser Stelle soll ein kurzer Vergleich von vektorbasierten neuronalen Netzwerken mit den sogenannten *mehrschichtigen Perzeptrons* [Rumelhart et al., 1986a] (MLP) gemacht werden. Dieser Netzwerktyp, der als Sonderfall auch die einschichtigen Perzeptrons umfaßt, dominiert zur Zeit in Anwendungen und in der Fachliteratur. Da einzelne Neuronen dieses Netzwerktyps stets ein Skalarprodukt ihres Gewichtsvektors mit dem ihnen gegebenen Eingabevektor bilden (siehe unten), kann man diese Netzwerke auch als *skalarproduktbasierte Netzwerke* bezeichnen, wogegen die vektorbasierten neuronalen Netze auch als *distanzbasierte neuronale Netze* bezeichnet werden könnten.

Mehrschichtige Perzeptrons werden ausschließlich für überwachtes Lernen eingesetzt, d.h. sie setzen die Existenz einer Trainingsdatenmenge

$$\mathcal{D} = \{(\boldsymbol{\xi}^1, \boldsymbol{\zeta}^1), (\boldsymbol{\xi}^2, \boldsymbol{\zeta}^2), \ldots, (\boldsymbol{\xi}^M, \boldsymbol{\zeta}^M)\}$$

voraus, die aus $M$ Paaren von Eingabevektor $\boldsymbol{\xi}^i \in \mathrm{R}^n$ und Zielvektor $\boldsymbol{\zeta}^i \in \mathrm{R}^m$ besteht. Ein vordergründiges Ziel besteht dann darin, solche Parameterwerte für die Netzwerke zu finden, daß bei Präsentation des Eingabevektors $\boldsymbol{\xi}^i$ die Netzwerkausgabe $S(\boldsymbol{\xi})$ dem Zielvektor möglichst genau, also z.B. mit kleinem quadratischen Fehler

$$E = \frac{1}{2} \sum_{i=1}^{M} \|S(\boldsymbol{\xi}^i) - \boldsymbol{\zeta}^i\|^2,$$

entspricht. Das eigentliche Ziel ist jedoch, wie in Kapitel 10 detailliert diskutiert wird, die Modellierung der unbekannten Relation

$$\boldsymbol{\zeta} = f(\boldsymbol{\xi}),$$

der die Trainingsdaten unterliegen.

## 4.1 Netzwerkstruktur

Ein mehrschichtiges Perzeptron besteht aus mehreren Schichten formaler Neuronen (siehe Abbildung 4.1a). Jede Schicht ist mit der darüberliegenden vollstän-

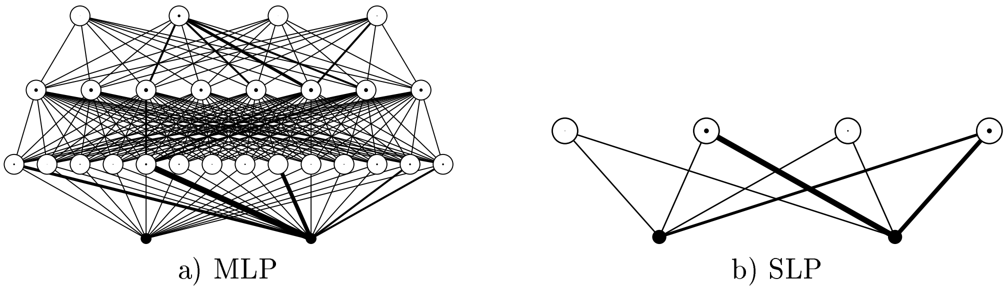
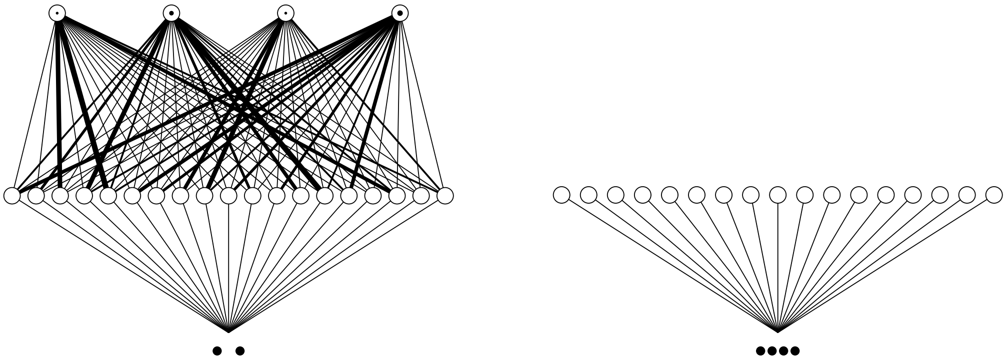



Abbildung 4.1: Skalarproduktbasierte Netzwerke. a) Mehrschichtiges Perzeptron (multi-layer perceptron, MLP) mit zweidimensionalen Eingabedaten, zwei versteckten Schichten der Größe 14 und 8 sowie einer Ausgabeschicht mit 4 Einheiten. Die unterschiedlich starken Verbindungen symbolisieren die gelernten Gewichte des Netzes. Der Punkt innerhalb der Neuronendarstellung steht für den Bias-Wert (siehe Text). b) Einschichtiges Perzeptron (single-layer perceptron, SLP) mit zweidimensionalen Eingabedaten und einer Ausgabeschicht mit 4 Einheiten.

dig verknüpft durch gewichtete Verbindungen. Die unterste Schicht ist die Eingabeschicht. Die Anzahl ihrer Einheiten ist gleich der Dimensionalität $n$ des Eingabevektors $\boldsymbol{\xi}$. Die Eingabeschicht spielt eine gewisse Sonderrolle, da hier keinerlei Verarbeitung stattfindet, sondern nur die Eingabewerte zur Verfügung gestellt werden. Daher wird sie normalerweise auch nicht mitgezählt, wenn die Anzahl der Schichten eines Netzwerks spezifiziert wird. Es folgen eine oder mehrere sogenannte *versteckte Schichten.* Jedes Neuron in diesen Schichten empfängt Eingangssignale von der jeweils darunterliegenden Schicht und sendet sein Ausgangssignal an jedes Neuron der folgenden Schicht. Die oberste Schicht schließlich ist die Ausgabeschicht, deren Größe der Dimensionalität $m$ des Zielvektors $\boldsymbol{\zeta}$ entspricht und deren Ausgabewerte die Komponenten des Ausgabevektors darstellen.

*Einschichtige Perzeptrons* besitzen keine versteckte Schicht und daher wird ihre gesamte Topologie durch die Dimensionen der Ein- und der Ausgabedaten bestimmt (Abbildung 4.1b).

Bei vektorbasierten Netzwerken muß man zwischen den überwacht lernenden und den unüberwacht lernenden unterscheiden. Ein typischer Vertreter der überwacht lernenden vektorbasierten Netze ist das RBF-Netz (Abbildung 4.2a, siehe auch Abschnitt 12.1). Es hat genau eine versteckte Schicht von Einheiten, die durch gewichtete Verbindungen mit den Ausgabeeinheiten verknüpft sind. Im Unterschied zu den mehrschichtigen Perzeptrons gibt es jedoch keine gewichteten Verbindungen von der Eingabeschicht. Der Grund liegt in den verschiedenen formalen Neuronenmodellen, die beim Perzeptron auf einem Skalarprodukt beruhen und beim vektorbasierten Netzwerk auf einer Vektordistanz. In Abbildung 4.2 ist dies dadurch symbolisiert, daß vom Eingabevektor nur eine Verbindung zu jeder Einheit der nächsten Schicht dargestellt ist. Diese Verbindung steht für die Weiterleitung des unveränderten Eingabevektors zu den lokalen Einheiten der nächsten Schicht, die anschließend die Distanz zwischen ihrem Referenzvektor und dem Eingabesignal berechnen und daraus ihre



a) RBF-Netzwerk

b) unüberwacht lernendes vektorbasiertes Netzwerk

Abbildung 4.2: Vektorbasierte (oder distanzbasierte) Netzwerke. Bei überwacht lernenden Netzwerken, wie z.B. einem RBF-Netzwerk (a), existiert eine Ausgabeschicht, deren Größe durch die Dimension $m$ der gegebenen Ausgabedaten festgelegt wird (hier 4). Unüberwacht lernende vektorbasierte Netzwerke (b) haben nur eine Schicht lokaler Neuronen, aber keine eigentliche Ausgabeschicht. Die Dimension des Eingabevektors bei dem in (b) dargestellten Netzwerk ist 4. Sowohl für überwacht als auch für unüberwacht lernende vektorbasierte Netzwerke wird das Eingabesignal jeweils allen lokalen Einheiten zugeführt, die dann die Vektordistanz zu ihrem jeweiligen Referenzvektor berechnen und daraus ihre Aktivierung ableiten.

Aktivierung ableiten.

Bei unüberwachten vektorbasierten Netzwerken existiert keine separate Ausgabeschicht (siehe Abbildung 4.2b). Allerdings kann der komplette Vektor von Aktivierungen der lokalen Einheiten als Ausgabewert interpretiert werden.

## 4.2 Neuronenmodell

Ein einzelnes formales Neuron eines mehrschichtigen Perzeptrons ist in Abbildung 4.3 dargestellt. Die Funktion, die es aus dem Eingabevektor $\boldsymbol{\xi} \in \mathrm{R}^n$ und einem Gewichtsvektor $\mathbf{w} \in \mathrm{R}^n$ berechnet, ist ein Skalarprodukt, dessen Ergebnis einer sogenannten *Aktivierungsfunktion* g(·) zugeführt wird:

$$y_j(\boldsymbol{\xi}) = g(\mathbf{w}_j^T \boldsymbol{\xi} + w_{j0}) \tag{4.1}$$

Dabei ist $j$ der Index des Neurons (im allgemeinen werden mehrere verwendet) und $w_{j0}$ ist der sogenannte *Bias* (siehe unten). Die Aktivierungsfunktion g(·) ist in der Regel monoton ansteigend und nichtlinear. Mögliche Realisierungen sind eine Schwellwertfunktion (Abbildung 4.4)

$$g(x) = \begin{cases} 1 & : \quad x \geq 0 \\ 0 & : \quad x < 0 \end{cases} \tag{4.2}$$

oder eine Sigmoidfunktion (Abbildung 4.5)

$$g(x) = \frac{1}{1 + e^{-x}}. \tag{4.3}$$

Welcher Art ist die „Verarbeitung“, die ein solches Neuron durchführt? Der Eingabevektor $\boldsymbol{\xi}$ läßt sich zerlegen in zwei Komponenten

$$\boldsymbol{\xi} = \boldsymbol{\xi}^{col} + \boldsymbol{\xi}^{ort},$$

von denen eine ($\boldsymbol{\xi}^{ort}$) orthogonal zum Gewichtsvektor $\mathbf{w}_j$ ist:

$$\mathbf{w}_j^T \boldsymbol{\xi}^{ort} = 0.$$

Damit läßt sich das Skalarprodukt $\mathbf{w}_j^T \boldsymbol{\xi}$ ausdrücken als

$$\begin{aligned} \mathbf{w}_j^T \boldsymbol{\xi} &= \mathbf{w}_j^T \boldsymbol{\xi}^{col} + \mathbf{w}_j^T \boldsymbol{\xi}^{ort} \\ &= \mathbf{w}_j^T \boldsymbol{\xi}^{col} \end{aligned}$$

und sein Ergebnis ist nur abhängig von der zu $\mathbf{w}_j$ *kollinearen* Komponente von $\boldsymbol{\xi}$. Auf jedem Punkt einer (Hyper-)Ebene E, die normal zu $\mathbf{w}$ ist, hat das Skalarprodukt den gleichen Wert, und auf derjenigen Normalenebene zu $\mathbf{w}_j$, die durch den Koordinatenursprung verläuft, hat es den Wert Null. Diese Ebene unterteilt den $\mathrm{R}^n$ in zwei Halbräume, wobei das Skalarprodukt in einem dieser Halbräume positiv und im anderen negativ ist.

Wird als Aktivierungsfunktion die Schwellwertfunktion (4.2) verwendet, so wird jeder Eingabevektor $\boldsymbol{\xi}$ entweder auf Null oder auf Eins abgebildet. Durch Addition einer *Bias*[1] genannten Konstanten $w_{j0}$ zum Skalarprodukt kann die Grenze zwischen den auf Null und auf Eins abgebildeten Halbräumen, die eigentlich durch den Koordinatenursprung geht, beliebig entlang des Eingabevektors verschoben werden. Wie man sich leicht überlegt [Bishop, 1995, Seite 78], ist der Abstand der Hyperebene vom Koordinatenursprung gegeben durch

$$l_j = -\frac{w_{j0}}{\|\mathbf{w}_j\|}.$$

Adaptive Verfahren, wie die Backpropagation-Lernregel [Rumelhart et al., 1986b], die geeignete Werte für die Gewichte bestimmen, können in gleicher Weise auch den Bias lernen. Zur Vereinfachung der Notation wird der Datenvektor $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)^T$ deshalb oft um eine Komponente $\xi_0$ erweitert, die konstant auf Eins gesetzt wird und deren zugeordnetes Gewicht der Bias ist (siehe Abbildung 4.3).

## 4.3 Aktivierungsregionen

Als *Aktivierungsregion* eines formalen Neurons bezeichnet man den Bereich im Eingaberaum $\mathrm{R}^n$, aus dem ein Eingabesignal stammen muß, um das betreffende Neuron zu aktivieren. Das Äquivalent aus der Biologie wäre das rezeptive Feld eines Neurons.

[1] Manchmal wird auch $-w_0$ als *Schwellwert* bezeichnet, da das Skalarprodukt diesen Wert überschreiten muß, damit die Ausgabe des Neurons von Null auf Eins wechselt.

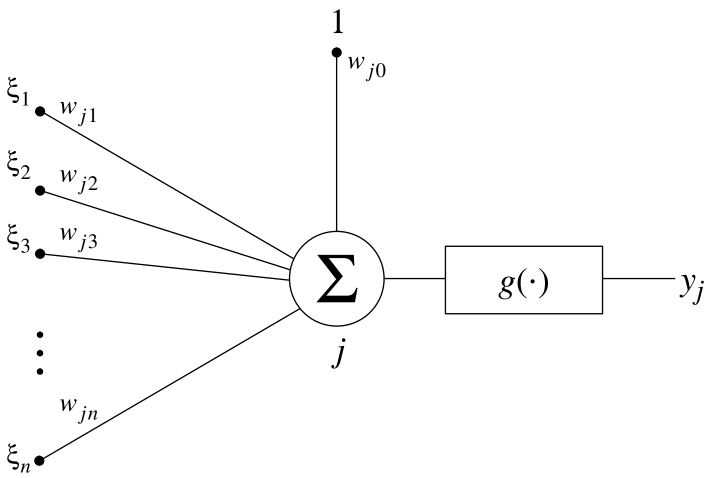



Abbildung 4.3: Formales Neuron eines mehrschichtigen Perzeptrons.

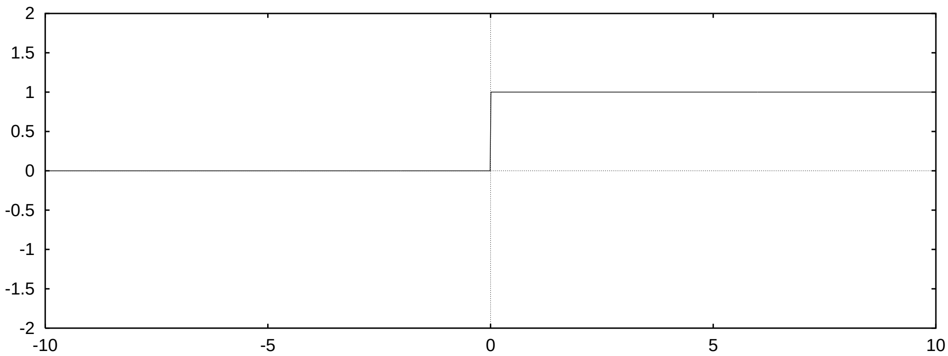



Abbildung 4.4: Schwellwertfunktion

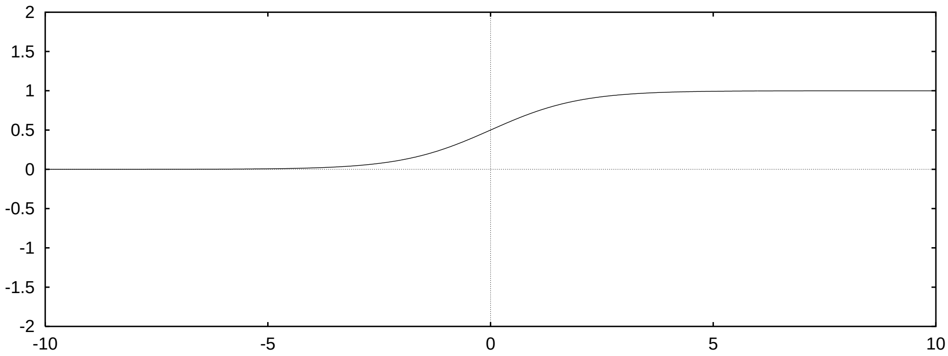
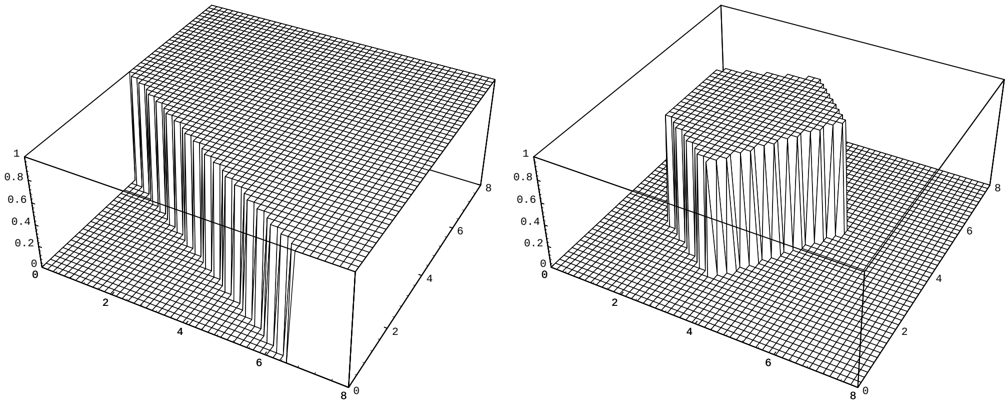



Abbildung 4.5: Sigmoidfunktion



a) Perzeptron b) vektorbasiertes Netzwerk

Abbildung 4.6: Aktivierungsregionen bei Schwellwertaktivierung. Der Eingaberaum ist zweidimensional. a) Aktivierungsregion eines skalarproduktbasierten Neurons mit Schwellwertaktivierungsfunktion. b) Aktivierungsregion eines vektorbasierten (bzw. distanzbasierten) Neurons.

Vergleichen wir nun die Aktivierungsregion eines Neuron mit Schwellwertaktivierungsfunktion mit der einer Einheit eines vektorbasierten neuronalen Netzwerks. Diese wird nur dann aktiviert, wenn sie für den Datenvektor der Gewinner ist und hat folgende Aktivierungsfunktion:

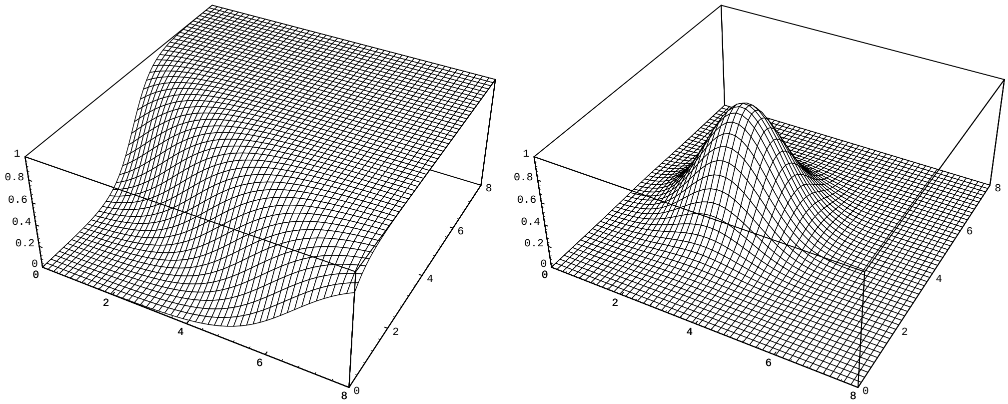
$$g_c(\boldsymbol{\xi}) = \begin{cases} 1 & : \text{ falls Einheit } c \text{ Gewinner ist,} \\ 0 & : \text{ sonst.} \end{cases} \tag{4.4}$$

Das Neuron mit Schwellwertaktivierungsfunktion (4.2) wird aktiviert von Signalen, die aus einem unendlich großen Halbraum im $\mathrm{R}^n$ kommen können. Das Antwortverhalten ist *nichtlokal.* Bei der Einheit des vektorbasierten neuronalen Netzwerks hingegen mit Aktivierungsfunktion (4.4) muß das Eingabesignal aus dem Voronoigebiet dieser Einheit stammen, damit die Einheit aktiviert wird. Dies ist ein konvexes und (meistens) endliches Teilgebiet des $\mathrm{R}^n$. Das Antwortverhalten ist *lokal.* Abbildung 4.6 gibt Beispiele für die jeweiligen Aktivierungsregionen im Fall eines zweidimensionalen Eingaberaums.

Ganz ähnliche Überlegungen lassen sich auch mit einer sigmoiden Aktivierungsfunktion (4.3) auf der Seite des Perzeptrons und einer kontinuierlichen, aber lokalisierten Aktivierungsfunktion wie der Gauß'schen Glockenkurve

$$g_c(\boldsymbol{\xi}) = \exp\left(-\frac{\|\boldsymbol{\xi} - \mathbf{w}_c\|^2}{2\sigma}\right) \tag{4.5}$$

auf der Seite der vektorbasierten neuronalen Netzwerke anstellen. Die Aktivierung der Neuronen ist nun kontinuierlich und nimmt die Extremwerte Null und Eins nur im Grenzfall an. Die Grundaussage bleibt die gleiche: ein skalarproduktbasiertes Neuron eines Perzeptrons wird von Signalen *wesentlich* aktiviert (z.B. so, daß die Aktivierung größer als ein Wert $\epsilon$ ist), die fast aus dem gesamten Eingaberaum kommen können. Ein distanzbasiertes Neuron eines vektorbasierten Netzwerks wird dagegen nur von Signalen aus einer begrenzten Region um seinen Referenzvektor herum wesentlich aktiviert (siehe Abbildung 4.7).



a) Perzeptron b) vektorbasiertes Netzwerk

Abbildung 4.7: Aktivierungsregionen bei kontinuierlicher Aktivierung. a) Aktivierungsregion eines skalarproduktbasierten Neurons mit sigmoider Aktivierungsfunktion wie es in mehrschichtigen Perzeptrons verwendet wird. b) Aktivierungsregion eines Neurons mit Gauß'scher Aktivierungsfunktion wie es in bestimmten vektorbasierten Netzwerken (z.B. Netzwerken radialer Basisfunktionen) verwendet wird.

Es sei angemerkt, daß sich durch Kombination mehrerer Schwellwert- oder Sigmoidneuronen auf höheren Schichten des Netzwerks durchaus konvexe Aktivierungsregionen bilden lassen. Es bedarf hierbei aber einer speziellen Anordnung der Hyperebenen der beteiligten Neuronen, während bei vektorbasierten neuronalen Netzen die lokalen Aktivierungsregionen von vornherein vorhanden sind.

## 4.4 Datenrepräsentation in inneren Schichten

Ein interessanter Aspekt ist auch die Repräsentation der Eingabedaten in der ersten versteckten Schicht eines mehrschichtigen Perzeptrons und innerhalb der (einzigen) Schicht lokaler Einheiten eines vektorbasierten Netzwerks.

Bei Perzeptrons ist (wie gerade dargelegt wurde) die Aktivierungsregion jedes Neurons dieser Schicht ein Halbraum des Eingaberaums. Die Lage dieses Halbraums wird vom Gewichtsvektor und vom Bias des Neurons bestimmt. Bei zufälliger Initialisierung dieser Werte wird ein Eingabesignal im Mittel die Hälfte aller Neuronen aktivieren. Dies kann man als *verteilte Datendarstellung* interpretieren. Bei vektorbasierten neuronalen Netzen wird von jedem Eingabesignal nur genau ein Neuron (bei Aktivierungsfunktion 4.4) oder wenige Neurone (bei Aktivierungsfunktion 4.5) aktiviert. Dies ist eine *lokalisierte Datendarstellung.*

## 4.5 Trainingsverfahren

Einschichtige Perzeptrons mit einer linearen Aktivierungsfunktion sind sehr einfach zu trainieren. Sie bestehen aus mehreren parallel geschalteten formalen Neuronen wie in Abbildung 4.3 dargestellt, wobei $g(\cdot)$ eine lineare Funktion ist.

Ohne Beschränkung der Allgemeinheit reicht es aus, eines dieser Neuronen zu betrachten. Für ein derartiges Neuron $j$ ist eine Trainingsdatenmenge

$$\mathcal{D} = \{(\boldsymbol{\xi}^1, \zeta_j^1), (\boldsymbol{\xi}^2, \zeta_j^2) \ldots (\boldsymbol{\xi}_j^M, \zeta_j^M)\}$$

gegeben, die aus $M$ Paaren von Eingabevektor $\boldsymbol{\xi}^i \in \mathcal{R}^n$ und Zielskalar $\zeta_j^i \in \mathrm{R}$ besteht. Dabei ist $\zeta_j^i$ die $j$-te Komponente des $m$-dimensionalen Zielvektors $\boldsymbol{\zeta}^i$.

Das Ziel ist dann, einen Gewichtsvektor $\mathbf{w}_j = (w_1, w_2, \ldots, w_n)^T$ zu finden, so daß die Ausgabe $y_j(\boldsymbol{\xi})$ des Neurons die Zielwerte $\zeta$ möglichst gut approximiert. In der Regel soll der quadratische Fehler

$$E[\mathbf{w}_j] = \frac{1}{2} \sum_{i=1}^{M} (y_j(\boldsymbol{\xi}^i) - \zeta_j^i)^2 \tag{4.6}$$

minimiert werden.

Wenn die Aktivierungsfunktion $g(\cdot)$ gleich der Identität gewählt wird und der Biasparameter als zusätzliche Komponente $w_{j0}$ in den Gewichtsvektor integriert wird (zusätzlich wird jeder Eingabevektor $\boldsymbol{\xi}^i = (\xi_1^i, \ldots, \xi_n^i)^T$ um eine Komponente $\xi_0^i$ erweitert, die immer auf den Wert Eins gesetzt wird), läßt sich die Ausgabe (4.1) des Neurons schreiben als das Skalarprodukt

$$y_j(\boldsymbol{\xi}) = \mathbf{w}_j^T \boldsymbol{\xi} = \sum_{i=0}^{n} w_{ji} \xi_i. \tag{4.7}$$

Jedes Ein-/Ausgabepaar $(\boldsymbol{\xi}_k^M, \zeta_k^M)$ definiert damit eine lineare Gleichung in $n+1$ Unbekannten:

$$\sum_{i=0}^{n} w_{ji} \xi_i^k = \zeta_j^k.$$

Da es im allgemeinen $M > n+1$ Gleichungen gibt, ist das Gleichungssystem überbestimmt und kann nicht exakt gelöst werden. Numerische Standardtechniken wie die Singular-Value-Decomposition gestatten allerdings die direkte Berechnung derjenigen Näherungslösung, die den Fehler (4.6) minimiert (siehe Anhang A.4).

Alternativ zu der direkten Berechnung mit den erwähnten Matrixtechniken kann auch ein iterativer Gradientenabstieg im $(n+1)$-dimensionalen Raum der Gewichte durchgeführt werden. Dazu muß die Fehlerfunktion (4.6) partiell nach jedem Gewicht abgeleitet werden. Der Gradient ist der Vektor der so erhaltenen partiellen Ableitungen. Eine wiederholte Veränderung des Gewichtsvektors $\mathbf{w}$ in Richtung des negativen Gradienten, d.h. eine Änderung der $k$-ten Komponente von $\mathbf{w}$ gemäß

$$\Delta w_k = -\eta \frac{\partial E}{\partial w_k},$$

führt schließlich zu einem lokalen Minimum der Fehlerfunktion, das im Falle eines linearen Neurons gleichzeitig das globale Minimum ist [Baldi & Hornik, 1989]. Ein Argument für die Durchführung dieses iterativen Verfahrens anstelle der direkten Lösung durch Matrixtechniken ist die Möglichkeit, den Gradientenabstieg durch Adaptionen anzunähern, die jeweils nach der Betrachtung eines einzigen Ein-/Ausgabepaars gemacht werden. Das resultierende Verfahren wird auch als *Delta-Regel* [Stone, 1986] bezeichnet und ermöglicht ein sogenanntes *Online-Lernen*, das auch bei sehr großen oder unendlichen Datenmengen einsetzbar ist (siehe Anhang A.1).

Bei mehrschichtigen Perzeptrons ist die Situation eine ganz andere. Durch die nichtlinearen Aktivierungsfunktionen der Neuronen in den versteckten Schichten ist auch die Netzwerkausgabe eine nichtlineare Funktion der Gewichte. Entsprechend existieren keine mathematischen Standardtechniken, um direkt Gewichte zu bestimmen, die den quadratischen Fehler minimieren. Daher müssen Gradientenabstieg ( z.B. das bekannt Backpropagation Verfahren [Rumelhart et al., 1986b]) oder andere nichtlineare Optimierungstechniken (wie das Verfahren der konjugierten Gradienten [Press et al., 1992]) angewendet werden, um geeignete Gewichte zu finden. Die Präsenz von lokalen Minima und weitere Eigenschaften der Fehlerfunktion im hochdimensionalen Gewichtsraum (die Dimension ist gleich der Anzahl aller Gewichte des Netzwerks) machen das Training erheblich langwieriger als bei einschichtigen Perzeptrons. Der potentielle Gewinn besteht in der größeren Klasse von Funktionen, die durch mehrschichtige Perzeptrons approximiert werden kann. Es wurde bewiesen [Hornik et al., 1989], daß mehrschichtige Perzeptrons sogenannte universelle Approximatoren darstellen, d.h. sie können nahezu beliebige Funktionen in einem kompakten Unterbereich des $\mathrm{R}^n$ beliebig genau annähern. Die entsprechenden Beweise setzen eine unbegrenzte Anzahl von Neuronen in den versteckten Schichten voraus und sind leider nicht konstruktiv. Die Eigenschaft der universellen Approximation teilen die mehrschichtigen Perzeptrons mit den Netzwerken radialer Basisfunktionen (Abschnitt 12.1), wie u.a. von Hartman et al. [1990] gezeigt wurde.

# Teil II

# Unüberwachtes Vektorbasiertes Lernen

# Kapitel 5

# Ziele unüberwachten Lernens

Wir wollen im folgenden unter unüberwachtem Lernen ausschließlich *Wettbewerbslernen* verstehen, da nur dieses Lernparadigma für die behandelten vektorbasierten Netzwerke relevant ist. Es sei aber angemerkt, daß z.B. auch das sogenannte *Hebb'sche Lernen*[1] dem unüberwachten Lernen zugeordnet wird, jedoch eine völlig andere Netzwerkarchitektur erfordert.

Für unüberwachtes Lernen gibt es mehrere mögliche Zielsetzungen, die meist nicht miteinander vereinbar sind. Im folgenden werden einige dieser Zielsetzungen dargestellt.

## 5.1 Fehlerminimierung

Ein häufiges Ziel ist die Minimierung der durchschnittlichen Distanz zwischen Eingabesignal $\boldsymbol{\xi}$ und dem Referenzvektor $\mathbf{w}_{s(\boldsymbol{\xi})}$ der Gewinnereinheit $s(\boldsymbol{\xi})$. Im Falle einer kontinuierlichen Eingabeverteilung $p(\boldsymbol{\xi})$ ist die Aufgabe die Minimierung des Fehlers

$$E[p(\boldsymbol{\xi}), \mathcal{A}] = \sum_{c \in \mathcal{A}} \int_{V_c} \|\boldsymbol{\xi} - \mathbf{w}_c\|^2 p(\boldsymbol{\xi}) d\boldsymbol{\xi}, \tag{5.1}$$

wobei $V_c$ das Voronoigebiet der Einheit $c$ ist.

Für den Fall, daß die Daten als endliche Datenmenge $\mathcal{D}$ vorliegen, läßt sich der zu minimierende Fehler ausdrücken durch

$$E[\mathcal{D}, \mathcal{A}] = 1/|\mathcal{D}| \sum_{c \in \mathcal{A}} \sum_{\boldsymbol{\xi} \in \mathcal{R}_c} \|\boldsymbol{\xi} - \mathbf{w}_c\|^2, \tag{5.2}$$

wobei $\mathcal{R}_c$ die Voronoimenge von $c$ ist.

Eine typische Anwendung, bei der Fehlerminimierung angestrebt wird, ist *Vektorquantisierung* [Linde et al., 1980]. Dabei werden Daten über einen Kommunikationskanal begrenzter Bandbreite übertragen, indem für jeden Datenvektor nur der Index des nächstgelegenen Referenzvektors übertragen wird. Es wird angenommen, daß die Menge aller Referenzvektoren, die in diesem Zusammenhang auch *Kodebuch* genannt wird, sowohl dem Sender als auch dem Empfänger

[1] Nicht zu verwechseln mit dem Hebb'schen Wettbewerbslernen, das in einem späteren Kapitel dargestellt wird.

bekannt ist. Daher kann der Empfänger die übertragenen Indizes benutzen, um die entsprechenden Kodebuchvektoren zu finden. Bei der Vektorquantisierung tritt ein Informationsverlust auf, da der Empänger anstelle eines Datenvektors nur den nächstgelegenen Referenzvektor erhält. Der Erwartungswert dieses Fehlers ist gegeben durch Gleichung (5.1) bzw. (5.2). Andererseits sind die durch Vektorquantisierung erzielbaren Kompressionsraten extrem hoch, da die Menge der zu übertragenden Information nicht von der Dimensionalität der Daten abhängt, sondern nur von der Größe des Kodebuchs. Beispielsweise läßt sich bei einem Kodebuch der Größe 256 jeder Vektor mit einem Byte übertragen. Wenn weiterhin die Datenvektoren eine Dimensionalität von 25 haben und jede Komponente eine doppelt genaue Fließkommazahl ist (8 Byte), ergibt sich ein Kompressionsfaktor von 200.

## 5.2 Entropiemaximierung

In manchen Situationen sollen die Referenzvektoren derart verteilt werden, daß jeder Referenzvektor die gleiche Chance hat, Gewinner für ein zufällig gewähltes Eingabesignal $\boldsymbol{\xi}$ zu sein:

$$P(s(\boldsymbol{\xi}) = c) \;=\; \frac{1}{|\mathcal{A}|} \quad (\forall c \in \mathcal{A}). \tag{5.3}$$

Wenn wir die Generierung eines Eingabesignals und anschließende Abbildung auf die nächste Einheit aus $\mathcal{A}$ als Zufallsexperiment deuten, bei dem eine Zufallsvariable $X$ einen Wert $x \in \mathcal{A}$ annimmt, so entspricht Gleichung (5.3) gerade einer Maximierung der *Entropie* nach Shannon [1948]

$$H(X) = -\sum_{x \in \mathcal{A}} P(x)\log(P(x)) = E(\log(\frac{1}{P(x)})), \tag{5.4}$$

wobei $E(\cdot)$ der Erwartungswertoperator ist. Der dabei angenommene Maximalwert von (5.4) ist $\log(|\mathcal{A}|)$ and trägt je nach Basis des Logarithmus die Einheit *Bits* (falls die Basis 2 ist) oder *Nats* (falls die Basis $e$ ist).

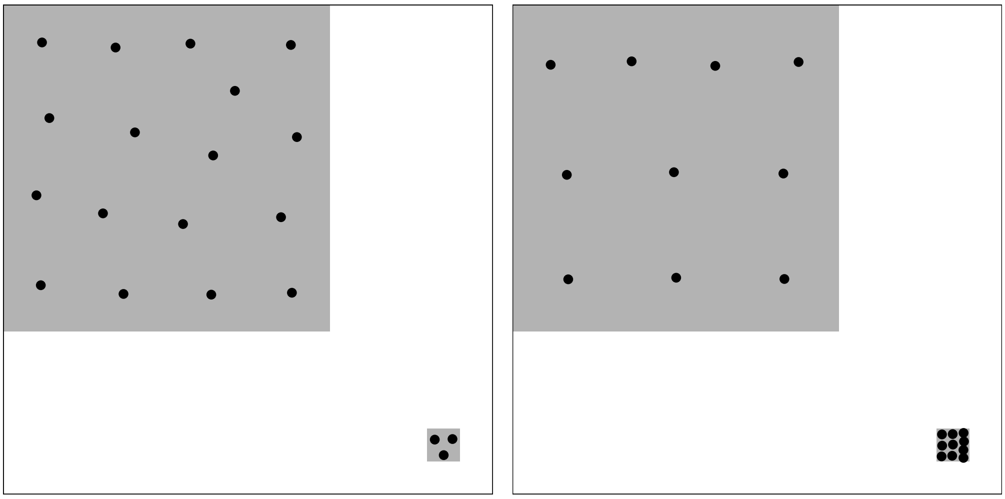
Im Falle einer kontinuierlichen Wahrscheinlichkeitsdichte ist (5.3) äquivalent zu

$$\int_{V_c} p(\boldsymbol{\xi})d\boldsymbol{\xi} = \frac{1}{|\mathcal{A}|} \quad (\forall c \in \mathcal{A}). \tag{5.5}$$

Im Falle einer endlichen Datenmenge $\mathcal{D}$ ist die Bedingung (5.3) dann erfüllt, wenn die Voronoimengen aller Einheiten jeweils gleich viele Elemente enthalten:

$$\frac{|\mathcal{R}_c|}{|\mathcal{D}|} = \frac{1}{|\mathcal{A}|} \quad (\forall c \in \mathcal{A}). \tag{5.6}$$

Eine exakte Gleichheit der Mengenkardinalitäten kann natürlich nur ereicht werden, wenn die Anzahl $|\mathcal{D}|$ der Datenvektoren ein ganzahliges Vielfaches der Anzahl der Referenzvektoren ist. Im allgemeinen ist daher bei endlichen Datenmengen die Maximierung der Entropie nur bis auf Diskretisierungseffekte erreichbar.



a) Fehlerminimierung: MSE=0.0024, Entropie = 2.657 (88.7%)

b) Entropiemaximierung: MSE=0.0038, Entropie = 2.990 (99.8%)

Abbildung 5.1: Beispiel für die Unvereinbarkeit von Entropiemaximierung und Fehlerminimierung. Die Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ ist in beiden Quadraten uniform, ist jedoch derart gewählt, daß in jedes Quadrat genau die Hälfte der Wahrscheinlichkeitsmasse fällt. 20 Referenzvektoren wurden auf zwei verschiedene Weisen verteilt, wobei jeweils der MSE (mittlerer quadratischer Fehler) und die Shannon-Entropie für 20000 zufällig erzeugte Signale gemessen wurde. Der theoretische Maximalwert für die Entropie ist in diesem Fall $\ln(20) = 2.996$. Die Verteilung zur Minimierung des Fehlers (a) positioniert nur wenige Referenzvektoren in dem Quadrat hoher Wahrscheinlichkeitsdichte und erreicht dadurch einen Quantisierungsfehler, der um mehr als 20% geringer ist als bei (b). Andererseits erreicht die Verteilung in (b), bei der die Dichte der Referenzvektoren in etwa proportional zu $p(\boldsymbol{\xi})$ ist, 99.8% der theoretisch maximalen Entropie, während in (a) nur 88.7% erreicht werden.

Wenn die Bedingung (5.3) erfüllt ist, liefert die lokale Punktdichte der Referenzvektoren in der Umgebung eines Vektors $\boldsymbol{\xi}$ einen Schätzwert für die Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ an dieser Stelle. Ein weiterer wichtiger Aspekt der Entopiemaximierung ist die Fehlertoleranz: bei Ausfall eines Referenzvektors ist der Anteil der Datenvektoren, die einem neuen Referenzvektor zugeordnet werden, beschränkt auf $1/|\mathcal{A}|$.

Entropiemaximierung und Fehlerminimierung lassen sich im allgemeinen nicht miteinander vereinbaren. Insbesondere bei nicht-uniformen Wahrscheinlichkeitsverteilungen ergeben sich unterschiedliche optimale Referenzvektorpositionen wie Abbildung 5.1 an einem Beispiel zeigt.

## 5.3 Dichteschätzung

Die Schätzung von Wahrscheinlichkeitsdichten ist ein Problem, daß in vielen Zusammenhängen auftritt. So erfordert z.B. bei Klassifizierungsproblemen die

Anwendung des *Bayes'schen Theorems*

$$P(C_k|\boldsymbol{\xi}) = \frac{p(\boldsymbol{\xi}|C_k)P(C_k)}{p(\boldsymbol{\xi})} \tag{5.7}$$

die Schätzung der bedingten Wahrscheinlichkeitsdichten $p(\boldsymbol{\xi}|C_k)$. Die $C_k$ sind dabei die verschiedenen Klassen, die auftreten können und die $P(C_k)$ sind die *a priori* Wahrscheinlichkeiten dieser Klassen, welche in der Regel unschwer abgeschätzt werden können durch die Anteile der verschiedenen Klassen an einer gegebenen Trainingsdatenmenge.

Eine Möglichkeit zur Schätzung von $p(\boldsymbol{\xi}|C_i)$ ist, von den Trainingsdaten diejenigen auszuwählen, die zur Klasse $C_i$ gehören und eine Schätzung der (nun unbedingten) Wahrscheinlichkeitsdichte dieser Daten vorzunehmen. Ein konventionelles Verfahren dazu ist das sogenannte Parzen-Window Verfahren. Dabei wird für jeden Datenpunkt ein Schätzwert generiert, indem in einer lokalen Umgebung fester Größe (dem Parzen-Window) die Anzahl der in dieser Umgebung vorhandenen Datenpunkte durch das Volumen der Umgebung geteilt wird. Bei großen Datenmengen ist dies ein erheblicher Aufwand. Vektorbasierte neuronale Netzwerke bieten die Möglichkeit, *typische* Repräsentanten für die Datenpunkte zu finden. Die Generierung der Schätzwerte muß dann nur noch für diese Repräsentanten durchgeführt werden.

Ein formales Maß für die Ähnlichkeit zweier Wahrscheinlichkeitsdichten $p(\boldsymbol{\xi})$ und $\tilde{p}(\boldsymbol{\xi})$ ist die *Kullback-Leibler-Divergenz*

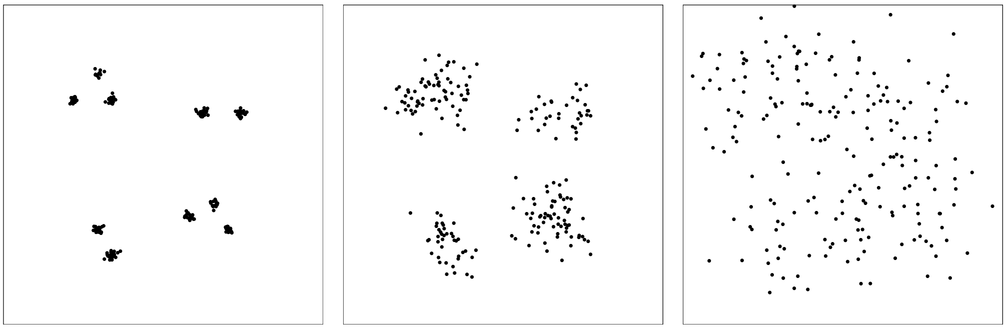
$$L = -\int p(\boldsymbol{\xi}) \ln \frac{\tilde{p}(\boldsymbol{\xi})}{p(\boldsymbol{\xi})}\, d\boldsymbol{\xi},$$

für die stets gilt, daß $L \geq 0$ mit Gleichheit dann und genau dann, wenn die zwei Wahrscheinlichkeitsdichten identisch sind. Natürlich läßt sich die Kullback-Leibler-Divergenz zwischen Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ und Schätzung $\tilde{p}(\boldsymbol{\xi})$ nur dann exakt berechnen, wenn $p(\boldsymbol{\xi})$ bekannt ist.

## 5.4 Clustering

Unter Clustering versteht man im allgemeinen die Aufteilung einer Datenmenge in Gruppen, sog. Cluster, derart, daß die Daten innerhalb eines Clusters sich ähnlich sind (geringe Intraclustervarianz) und sich die Daten aus unterschiedlichen Clustern unähnlich sind (hohe Interclustervarianz). Im Unterschied zur Klassifizierung gibt es beim Clustering keinerlei Informationen über etwaige Klassenzugehörigkeiten. Einziger Anhaltspunkt für die zu treffende Unterteilung ist das für die Daten definierte Distanzmaß. Oft sind Clusteringprobleme nur schwer präzise zu definieren (siehe Abbildung 5.2).

Man kann verschiedene konkrete Aufgabenstellungen unterscheiden, die z.B. darin differieren, ob die Anzahl der gewünschten Cluster *a priori* festgelegt wird oder Resultat des Verfahrens selbst ist. Clustering ist stark verwandt mit dem Problem der Vektorquantisierung. Einen guten Überblick über existierende Clusteringverfahren bietet Jain & Dubes [1988].



a) $\sigma = 0.01$ b) $\sigma = 0.05$ c) $\sigma = 0.14$

Abbildung 5.2: Beispiel für die unscharfe Definition des Clusteringproblems. Die Daten wurden aus 10 unterschiedlich zentrierten Normalverteilungen entnommen. Durch Variation der gemeinsamen Standardabweichung $\sigma$ dieser Verteilungen ergeben sich nacheinander verschiedene subjektive Aufteilungen in Cluster, die mit sich veränderndem $\sigma$ kontinuierlich ineinander übergehen.

## 5.5 Datenvisualisierung

Ein Problem, welches beim Umgang mit hochdimensionalen Daten immer wieder auftaucht, ist ihre Visualisierung. Einige vektorbasierte neuronale Netzwerke haben eine Netzwerkstruktur fester und niedriger Dimensionalität $k$ (z.B. ein zweidimensionales Gitter) und realisieren deshalb eine Abbildung der $n$-dimensionalen Daten auf diese diskrete $k$-dimensionale Struktur. Oft bleiben dabei Ähnlichkeitsbeziehungen in den ursprünglichen Daten enthalten, d.h. im Eingaberaum benachbarte Daten werden auf identische oder benachbarte Elemente des Netzwerks abgebildet. Das niedrigdimensionale Netzwerk kann graphisch dargestellt werden und die Daten können visualisiert werden, indem sie bei demjenigen Netzwerkknoten dargestellt werden, auf den sie abgebildet werden.

Entstammen die Daten einem niedrigdimensionalen Unteraum des $\mathrm{R}^n$, so besteht die Möglichkeit, daß das vektorbasierte Netzwerk diesen Unterraum findet. Wie die Abbildungen 5.3 und 5.4 zeigen, können auf diese Weise auch nichtlineare Unterräume gefunden werden im Gegensatz zur Hauptkomponentenanalyse (Kapitel 3), die nur lineare Unterräume identifizieren kann. Es ist jedoch oft schwer zu beurteilen, wie informationserhaltend die so durchgeführte Dimensionsreduzierung ist. Da es nur für spezielle Fälle bijektive Abbildungen von hoch- in niedrigdimensionale Räume gibt, kann es eine allgemeine Lösung auch nicht geben. Darüberhinaus können die gefundenen Abbildungen stark von der Initialisierung der Verfahren abhängen. Das gezeigte Beispiel ist entsprechend nicht unbedingt ein typisches, sondern eher ein ausgewählt gutes.

Im Bereich der Datenvisualisierung stehen die vektorbasierten Netzwerke mit fester Netzwerkdimensionalität in Konkurrenz zu anderen Ansätzen wie *Sammon's Mapping* [Sammon, 1969] oder *Generative Topographic Mapping* [Bishop et al., 1998]. Eine Frage, die in diesem Zusammenhang immer wieder auftaucht, ist, inwieweit die Abbildung der hochdimensionalen Daten in einen nied-

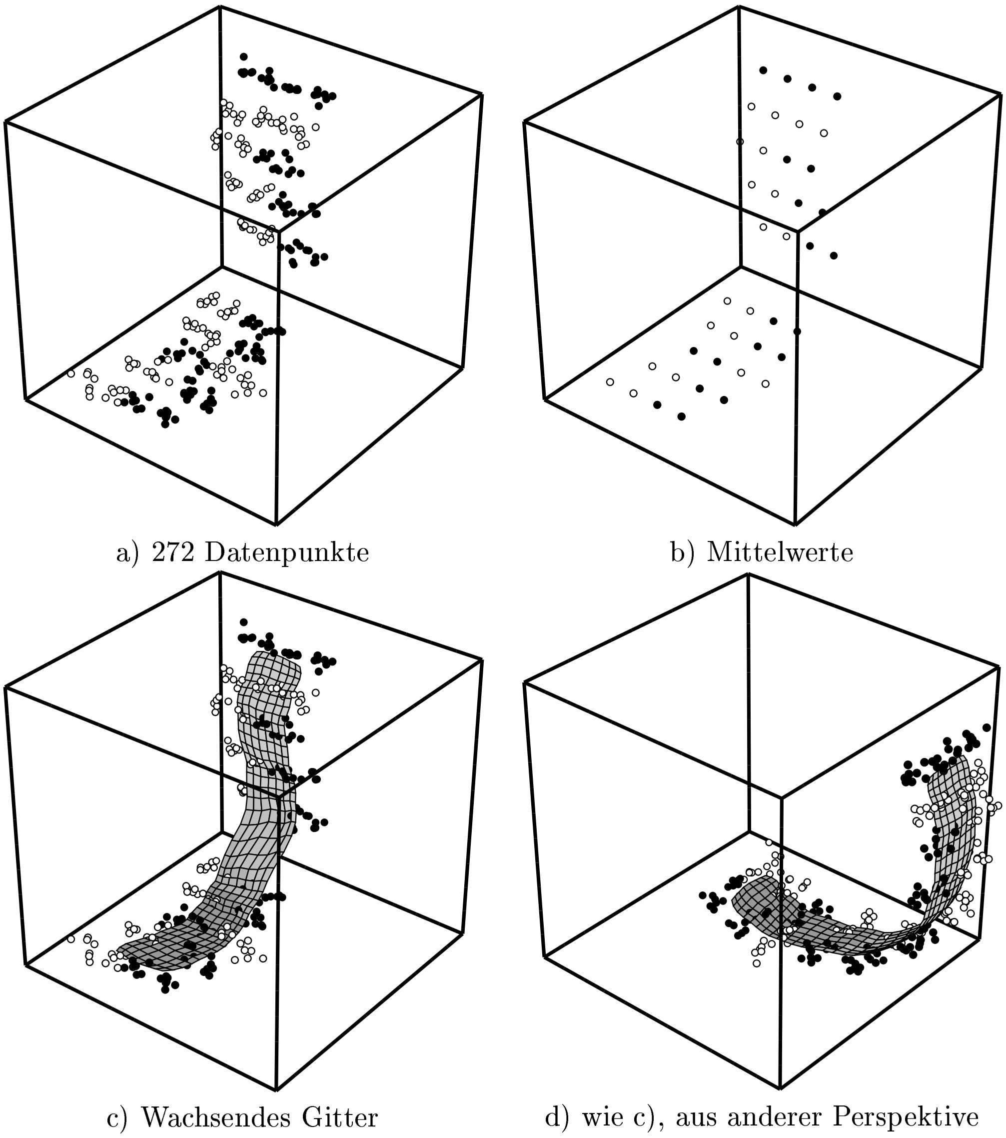



Abbildung 5.3: Ein zweidimensionales selbstorganisierendes Netzwerk (Wachsendes Gitter, siehe Abschnitt 9.3) hat sich an eine dreidimensionale Datenmenge angepaßt. Die Daten (a) gehören zu zwei Klassen und wurden aus einer Mischverteilung von 40 Normalverteilungen generiert, deren Mittelwerte auf zwei benachbarten Seitenflächen des Einheitswürfels liegen (b). Das Resultat eines datengesteuerten Wachstums einer zweidimensionalen Rechteckstruktur ist in (c) und (d) dargestellt.

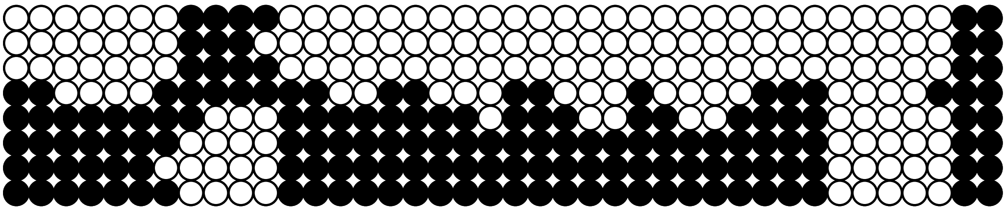


Abbildung 5.4: Visualisierung der Daten aus Abbildung 5.3 a) mit Hilfe der zweidimensionalen Netzwerkstruktur. Jeder Knoten des Netzwerks ist abhängig von der Klasse des nächsten Datenpunktes in schwarz oder weiss dargestellt.

rigerdimensionalen Raum *topologieerhaltend* ist, d.h. inwieweit Ähnlichkeitsbeziehungen in den ursprünglichen Daten nach der Projektion noch erhalten bleiben. Mit der exakten Definition von Topologieerhaltung haben sich eine ganze Reihe von Autoren mit unterschiedlichen Ergebnissen beschäftigt [Bauer & Pawelzik, 1992; Villmann et al., 1997; Goodhill & Sejnowski, 1997].

# Kapitel 6

# Hartes Wettbewerbslernen für endliche Datenmengen

Unter „hartem Wettbewerbslernen“ Verfahren, bei denen jedes Eingabesignal nur die Adaption des Gewinners beeinflußt und nicht zusätzlich die Adaption weiterer Einheiten.

Zwei grundsätzliche Varianten lassen sich unterscheiden. Die erste Variante (z.B. LBG) wertet zunächst alle Eingabevektoren aus, bevor irgendein Referenzvektor verändert wird. Das erfordert zwingend eine endliche Eingabedatenmenge. Derartige Verfahren werden auch als *Batch-Verfahren*[1] bezeichnet.

Bei der zweiten Variante (z.B. $k$-Means) wird nach jedem Eingabesignal sofort eine Adaption des Gewinners vorgenommen. In diesem Fall ist es nicht erforderlich, daß die Eingabedatenmenge endlich ist, sondern die Eingabesignale können auch als potentiell unendliche Folge dem Verfahren zugeführt werden. Man spricht in diesem Fall auch von *Online-Verfahren*[2].

Ein generelles Problem, das bei hartem Wettbewerbslernen auftreten kann, sind sogenannte *tote Einheiten* (engl. dead units). Damit werden Einheiten bezeichnet, die – z.B. durch ungeeignete Initialisierung – für kein Eingabesignal Gewinner werden und deshalb ihre Position stets beibehalten (siehe Abbildung 6.1). Diese Einheiten tragen nichts zu dem Zweck bei, dem das Netzwerk dient (z.B. Fehlerminimierung) und sind deshalb nicht genutzte Ressourcen, die es zu vermeiden gilt. Eine übliche Vorgehensweise, die das Entstehen toter Einheiten weitgehend verhindert, ist die Initialisierung aller Referenzvektoren mit Signalen gemäß $p(\boldsymbol{\xi})$ (bzw. aus der endlichen Datenmenge $\mathcal{D}$). Diese Strategie stellt sicher, daß sich zumindest anfangs keine Einheit in einem Gebiet mit

[1] Diese Bezeichnung ist abgeleitet vom „batch processing“ (deutsch Stapelverarbeitung), einer Betriebsart für Rechner, bei der jedes Programm komplett mit allen Eingabedaten in eine Warteschlange eingereiht und zu einem späteren Zeitpunkt ausgeführt wird. Die Programme dürfen dabei keine weiteren Eingriffe erfordern. Ein typisches Beispiel wäre die Berechnung der endgültigen Einzelbilder eines computergenerierten Spielfilms.

[2] Diese Bezeichnung stammt her vom „online processing“, einer Betriebsart für Rechner, bei der während der Ausführung eines Programms Daten durch einen Benutzer eingegeben und/oder abgelesen werden müssen. Typische Beispiele dafür wären das Flugbuchungssystem in einem Reisebüro, ein Chatbereich im Internet oder auch eine aus der Distanz ausgeführte chirurgische Operation, bei der die Bewegungen des Operateurs über ein Netzwerk auf eine Roboterhand übertragen werden.

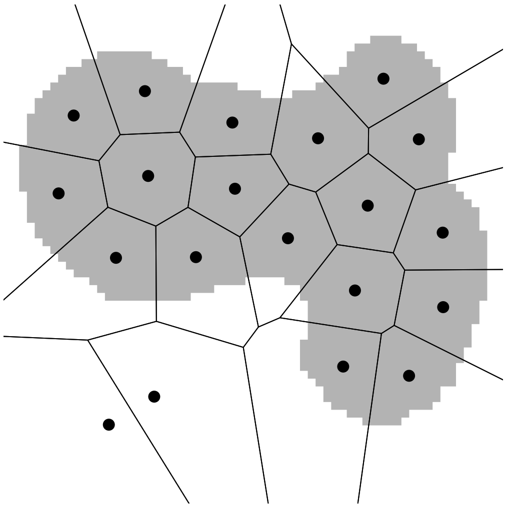


Abbildung 6.1: Tote Einheiten. Zwei Einheiten (links unten) des gezeigten Netzwerks haben keine Chance, Gewinner zu werden, da sich ihre Voronoigebiete nicht mit dem (grau schattierten) Gebiet positiver Wahrscheinlichkeitsdichte überschneiden.

Wahrscheinlichkeitsdichte Null befindet.

Das folgende Problem allerdings bleibt bestehen: Wenn die Referenzvektoren zufällig gemäß $p(\boldsymbol{\xi})$ initialisiert werden, ist ihre zu erwartende lokale Dichte proportional zu $p(\boldsymbol{\xi})$. Für bestimmte Zielsetzungen ist dies ungünstig. Sei etwa das gesetzte Ziel Fehlerminimierung und sei die Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ stark lokal variabel. Dann ist es (im Sinne der gewählten Zielfunktion) günstig, Gebiete hoher Wahrscheinlichkeitsdichte mit weniger Referenzvektoren zu kodieren, als es die Proportionalität zu $p(\boldsymbol{\xi})$ angeben würde. Ein Beispiel dafür war in Abbildung 5.1 zu sehen. Noch deutlicher wird es, wenn man die Fläche des kleineren Quadrats, aus dem nach Definition 50% der Signale kommen, gegen Null gehen läßt. Im Grenzfall reicht ein einziger Referenzvektor, um alle von dort kommenden Signale fehlerfrei zu kodieren. Die Proportionalität zu $p(\boldsymbol{\xi})$ würde jedoch erfordern, die Hälfte der verfügbaren Referenzvektoren dort zu positionieren.

Entsprechend ist auch bei anderen Zielsetzungen eine Initialisierung gemäß $p(\boldsymbol{\xi})$ oft ungünstig. Allerdings wird sie häufig trotzdem verwendet, da es schwierig ist *a priori* bessere Anfangspositionen zu finden. Die ausschließlich lokalen Adaptionen, die mit hartem Wettbewerbslernen vorgenommen werden, sind jedoch oft nicht in der Lage, ungünstige Initialisierungen zu korrigieren. So würde bei einer zufälligen Initialisierung gemäß der in Abbildung 5.1 gezeigten Verteilung fast immer eine gleiche Anzahl von Vektoren in beiden Quadraten sein. Dies kann durch hartes Wettbewerbslernen nicht geändert werden.

Es sei angemerkt, daß Verfahren des weichen Wettbewerbslernens (Kapitel 8 und 9), bei denen jedes Eingabesignal die Adaption mehrerer Einheiten des Netzwerks bewirkt, in der Regel viel weniger auf eine gute Initialisierung angewiesen sind. Verfahren, die die Anzahl der Einheiten variieren (Abschnitt 8.4, 9.2 und 9.3), haben darüberhinaus die Möglichkeit, die Positionen der Einheiten speziell der gestellten Aufgabe anzupassen.

## 6.1 LBG

Das LBG-Verfahren [Linde et al., 1980], benannt nach seinen Autoren Linde, Buzo und Gray, ist eine Methode zur Bestimmung eines *Kodebuches* (Menge von Referenzvektoren) zum Zwecke der Vektorquantisierung (siehe auch Abschnitt 5.1).

Das Grundprinzip von LBG ist, mehrfach jeden Referenzvektor $\mathbf{w}_c$ zum Schwerpunkt seiner Voronoimenge $\mathcal{R}_c$ zu bewegen. Die theoretische Basis dafür ist, daß eine *notwendige* Voraussetzung für die Minimierung des Quantisierungsfehlers

$$E[\mathcal{D},\mathcal{A}] = 1/|\mathcal{D}| \sum_{c\in\mathcal{A}} \sum_{\boldsymbol{\xi}\in\mathcal{R}_c} \|\boldsymbol{\xi} - \mathbf{w}_c\|^2$$

durch eine Menge von Referenzvektoren $\{\mathbf{w}_c | c \in \mathcal{A}\}$ die Erfüllung der sogenannten *Zentroidbedingung* durch jeden einzelnen Referenzvektor ist [Gray, 1992]. Die Zentroidbedingung besagt, daß jeder Referenzvektor im Massenschwerpunkt (Zentroid) der Wahrscheinlichkeitsdichte in seinem Voronoigebiet liegen muß (siehe dazu auch Seite 14 und Abbildung 2.3). Im Falle einer endlichen Eingabedatenmenge und bei Verwendung der Euklidischen Vektordistanz reduziert sich diese Bedingung zu

$$\mathbf{w}_c = \frac{1}{|\mathcal{R}_c|} \sum_{\boldsymbol{\xi}\in\mathcal{R}_c} \boldsymbol{\xi}, \tag{6.1}$$

also genau dazu, daß jeder Referenzvektor im Schwerpunkt der Datenvektoren seiner Voronoimenge $\mathcal{R}_c$ liegt. Die rechte Seite von Gleichung (6.1) entspricht genau der rechten Seite von Gleichung (2.9), d.h. der Zentroid einer Einheit $c$ ist genau der bedingte Erwartungswert für die Eingabesignale, für die $c$ der Gewinner ist.

Wenn man alle Referenzvektoren gemäß (6.1) auf den Schwerpunkt ihrer Voronoimenge verschoben hat, ist man jedoch im allgemeinen noch nicht fertig, da sich durch die Verschiebung der Referenzvektoren auch die Voronoigebiete und damit potentiell auch die Voronoimengen verändert haben. Deshalb wird dieser Schritt solange wiederholt, bis sich keine Veränderung der Voronoimengen mehr ergibt.

Das komplette LBG-Verfahren ist das folgende:

1. Initialisiere die Menge $\mathcal{A}$ mit $N$ Einheiten $c_i$

   $$\mathcal{A} := \{c_1, c_2, \ldots, c_N\}.$$

   Initialisiere die Referenzvektoren aller Einheiten:

   $$\mathbf{w}_c := (\text{zufällig aus } \mathcal{D}) \qquad (\forall c \in \mathcal{A})$$

2. Speichere den bisherigen Fehler:

   $$E_{\mathrm{LBG}} := E[\mathcal{D},\mathcal{A}]$$

3. Bestimme die Voronoimenge $\mathcal{R}_c$ für jede Einheit $c \in \mathcal{A}$.

4. Setze den Referenzvektor jeder Einheit auf den Mittelwert ihrer Voronoimenge:
$$\mathbf{w}_c := \frac{1}{|\mathcal{R}_c|} \sum_{\boldsymbol{\xi} \in \mathcal{R}_c} \boldsymbol{\xi} \quad (\forall c \in \mathcal{A}).$$

5. Wenn der aktuelle Fehler kleiner als der bisherige ist, d.h., wenn
$$E[\mathcal{D}, \mathcal{A}] < E_{\mathrm{LBG}},$$
fahre fort mit Schritt 2.

6. Ergebnis ist das aktuelle Kodebuch $\{\mathbf{w}_c | c \in \mathcal{A}\}$.

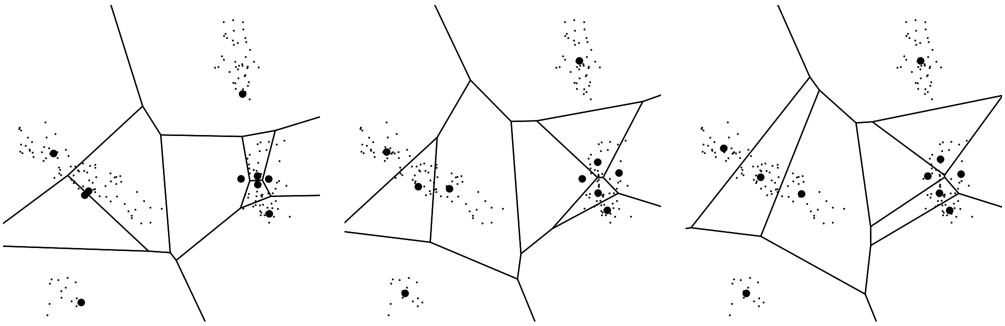
Die Schritte 3 und 4 werden als *Lloyd-Iteration* bezeichnet. Jede Lloyd-Iteration senkt entweder den Quantisierungsfehler oder läßt ihn unverändert [Gray, 1992]. Letzteres bedeutet, daß das Verfahren terminiert ist. Abbildung 6.2 zeigt ein Beispiel einer LBG-Simulation.

Die durch LBG erreichte Endkonfiguration stellt im allgemeinen nur ein *lokales* Minimum der Fehlerfunktion dar. Welches lokale Minimum erreicht wird, hängt sehr von den initialen Positionen der Referenzvektoren ab. Insbesondere, wenn die zu quantisierende Datenmenge $\mathcal{D}$ mehrere deutlich voneinander getrennte Cluster enthält, ist LBG nicht in der Lage, die ursprüngliche Aufteilung der Referenzvektoren auf Cluster zu verändern. Stattdessen werden in jedem Cluster die dort positionierten Referenzvektoren solange adaptiert, bis dort (lokal) die Zentroidbedingung (6.1) erfüllt ist. Ist dies für alle Cluster der Fall, ist LBG terminiert.

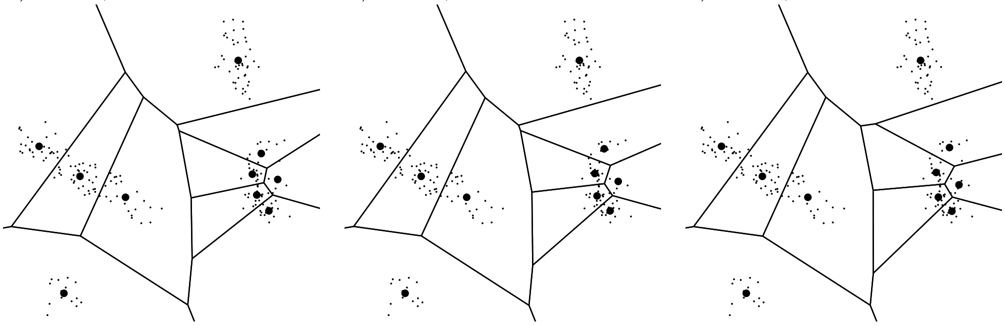
Bei Datenmengen, die deutliche Cluster enthalten, entscheidet sich also schon bei der Initialisierung, welches lokale Minimum LBG bestenfalls erreichen kann. Bei zufälliger Initialisierung durch Elemente der Datenmenge kann dies zu sehr schlechten Ergebnissen führen, insbesondere, wenn die existierenden Cluster eine stark unterschiedliche Punktdichte aufweisen (siehe Abbildung 6.3).

Ein denkbarer Weg, um mit LBG ein möglichst gutes Kodebuch zu erhalten, wären mehrfache Durchläufe und die Auswahl des besten dabei erhaltenen Kodebuchs. Abgesehen von dem erheblichen Rechenaufwand würde dies jedoch nicht das gerade genannte Problem bezüglich der unterschiedlichen Punktdichten lösen. Bei der in Abbildung 6.3 gezeigten Datenmenge wäre eine zufällige Initialisierung, bei der mehr als 10 Vektoren außerhalb des dichten Clusters landen, extrem unwahrscheinlich. Daher wäre eine impraktikabel hohe Anzahl von Versuchen notwendig, um mit hoher Wahrscheinlichkeit zumindest einmal eine derartige Initialisierung zu erreichen.

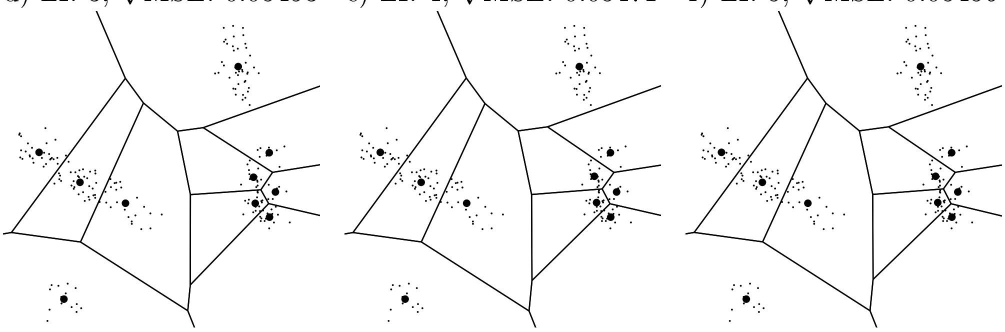
Besser wäre ein *zielgerichtetes* Verfahren, welches ein mit LBG erzeugtes Kodebuch schrittweise verbessert. Im nächsten Abschnitt wird ein solches Verfahren vorgestellt, das in vielen Fällen deutlich bessere Kodebücher als LBG liefert und gleichzeitig eine wesentlich geringere Abhängigkeit von der Initialisierung aufweist.



a) LI: 0, $\sqrt{\mathrm{MSE}}$: 0.09073 b) LI: 1, $\sqrt{\mathrm{MSE}}$: 0.06152 c) LI: 2, $\sqrt{\mathrm{MSE}}$: 0.05655

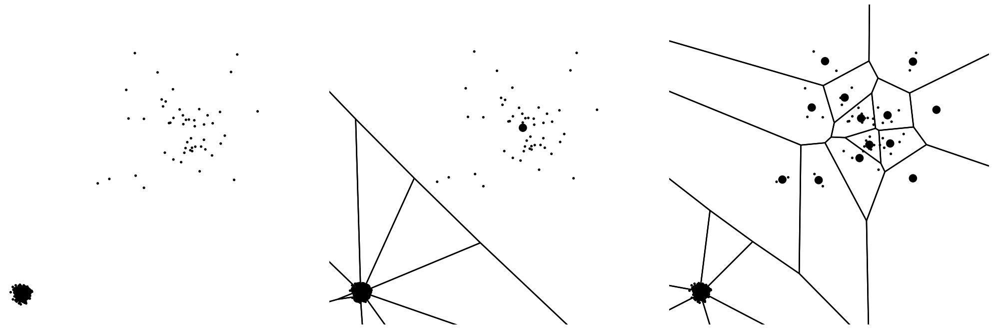


d) LI: 3, $\sqrt{\mathrm{MSE}}$: 0.05493 e) LI: 4, $\sqrt{\mathrm{MSE}}$: 0.05471 f) LI: 5, $\sqrt{\mathrm{MSE}}$: 0.05456



g) LI: 6, $\sqrt{\mathrm{MSE}}$: 0.05451 h) LI: 7, $\sqrt{\mathrm{MSE}}$: 0.05450 i) LI: 8, $\sqrt{\mathrm{MSE}}$: 0.05450

Abbildung 6.2: LBG-Simulation. Die Datenmenge $\mathcal{D}$ besteht aus 200 Datenpunkten, die in vier Clustern angeordnet ist. In (a) sind die initialen Referenzvektoren (LI = Lloyd-Iteration) gezeigt. Die übrigen Abbildungen (b)-(g) zeigen jeweils die Referenzvektoren nach der angegebenen Anzahl von Lloyd-Iterationen. Nach 7 Iterationen hat LBG ein lokales Minimum erreicht (h), was dadurch erkannt wird, daß der Fehler in der folgenden Iteration nicht mehr weiter sinkt (i). Der angegeben Fehlerwert $\sqrt{\mathrm{MSE}}$ ist die Wurzel aus dem mittleren quadratischen Fehler und ist ein Maß für die durchschnittliche Entfernung von Referenzvektor und Datenpunkt. Zum Vergleich: Die Daten füllen in etwas das Einheitsquadrat aus.



a) Datenmenge $\mathcal{D}$ mit 1000 Elementen — b) LBG-Kodebuch, $\sqrt{\text{MSE}}$: 0.001 — c) „besseres“ Kodebuch, $\sqrt{\text{MSE}}$: 0.00009

Abbildung 6.3: Beispiel für eine Verteilung, bei der LBG sehr ungünstige Kodebücher generiert. $\sqrt{\text{MSE}}$ bezeichnet die Quadratwurzel aus dem mittlerem quadratischen Fehler. Von den 1000 Datenpunkten (a) liegen 95% in einem lokal sehr begrenzten Cluster, während sich die restlichen 5% über ein wesentlich größeres Cluster verteilen. Bei der zufälligen Initialisierung von 20 Kodebuchvektoren lagen (genau der vorliegenden Wahrscheinlichkeit entsprechend) 19 Vektoren in dem Cluster mit der großen Punktdichte und ein Vektor in dem breiter gestreuten Cluster. LBG terminiert (nach 17 Lloyd-Iterationen) zu einer Konfiguration, die an dieser Aufteilung nichts änderte und die einen hohen Quantisierungsfehler aufwies (b). Mit einer alternativen Verteilung der Referenzvektoren, die stark von der Punktdichte der gegebenen Datenmenge abweicht, hätte in diesem Fall ein um eine ganze Größenordnung geringerer Quantisierungsfehler erreicht werden können (c).

## 6.2 LBG-U

Das im folgenden beschriebene Verfahren [Fritzke, 1997b] ist eine Generalisierung des LBG-Verfahrens. Die wesentliche Neuerung ist dabei die *nicht-lokale* Verschiebung eines einzelnen Referenzvektors, nachdem LBG terminiert ist. Diese Verschiebung entspricht einem Sprung im Raum aller möglichen Kodebücher hin zu einer neuen Position. Dort kann wiederum LBG verwendet werden, um ein neues (und hoffentlich besseres) lokales Minimum zu finden.

Nehmen wir im folgenden an, daß für eine Datenmenge $\mathcal{D}$ von LBG ein Kodebuch gefunden wurde, also eine Menge von Referenzvektoren $\{\mathbf{w}_c | c \in \mathcal{A}\}$, mit einem korrespondierenden Quantisierungsfehler $E[\mathcal{D}, \mathcal{A}]$.

Wie läßt sich dieses Kodebuch verbessern, ohne ein völlig neues zu erzeugen? Eine Möglichkeit ist, Kodebuchvektoren zu identifizieren, die wenig zur Reduzierung des Quantisierungsfehlers beitragen und diese auf neue Positionen im Eingaberaum zu verschieben. Um diese Idee zu quantifizieren, vergleichen wir für jede Einheit $c$ den derzeitigen Quantisierungsfehler

$$E[\mathcal{D}, \mathcal{A}]$$

mit dem Fehler

$$E[\mathcal{D}, \mathcal{A} \setminus \{c\}],$$

der dann auftreten würde, wenn wir die Einheit $c$ entfernten. Die Differenz dieser beiden Terme gibt genau den Beitrag an, den $c$ zur Reduzierung des Quantisierungsfehlers leistet. Daher definieren wir die *Nützlichkeit* (engl. utility) $U$ einer Einheit $c$ als

$$U(c) = E[\mathcal{D}, \mathcal{A} \setminus \{c\}] - E[\mathcal{D}, \mathcal{A}]. \tag{6.2}$$

Offensichtlich beeinträchtigt die Entfernung einer Einheit $c$ nur die Kodierung der Datenvektoren, für die $c$ Gewinner ist, also die Elemente der Voronoimenge $\mathcal{R}_c$. Diese Vektoren würden nach der Entfernung von $c$ auf die jeweils zweitnächste Einheit abgebildet werden, während sich für alle anderen Datenvektoren nichts änderte. Daher läßt sich Gleichung (6.2) äquivalent umformen zu

$$U(c) = \sum_{\boldsymbol{\xi} \in \mathcal{R}_c} \|\boldsymbol{\xi} - \mathbf{w}_{s_2(\boldsymbol{\xi})}\|^2 - \|\boldsymbol{\xi} - \mathbf{w}_c\|^2, \tag{6.3}$$

wobei $s_2(\boldsymbol{\xi})$ die für das jeweilige Eingabesignal $\boldsymbol{\xi}$ zweitnächste Einheit ist.

Eine Einheit mit einer Nützlichkeit $U = 0$ könnte demnach entfernt werden, ohne den Quantisierungsfehler zu erhöhen[3]. Darüberhinaus ermöglicht das Maß $U$ die Unterscheidung mehr oder weniger nützlicher Einheiten.

Um nun ein von LBG erzeugtes Kodebuch zu verbessern, könnte man die Einheit $a$ mit der geringsten Nützlichkeit $U$ an einen anderen Platz im Eingaberaum verschieben und LBG verwenden, um ausgehend von dieser neuen Konfiguration ein neues lokales Minimum der Fehlerfunktion zu finden. Wohin jedoch sollte $a$ (bzw. der Referenzvektor $\mathbf{w}_a$) verschoben werden?

Ohne weiteres Wissen über die Verteilung der Daten in $\mathcal{D}$ ist die naheliegendste Strategie, $a$ bei derjenigen Einheit $b$ zu plazieren, bei der zur Zeit der meiste Quantisierungsfehler (summiert für alle dieser Einheit zugeordneten Datenpunkte) auftritt. Dort könnte die Einheit $a$ wesentlich nützlicher sein als an ihrer augenblicklichen Position. Um dies formal ausdrücken zu können, bezeichnen wir mit $E[c]$, $c \in \mathcal{A}$, den Fehler, der für die Datenvektoren in der Voronoimenge von $c$ auftritt:

$$E[c] = \sum_{\boldsymbol{\xi} \in \mathcal{R}_c} \|\boldsymbol{\xi} - \mathbf{w}_c\|^2. \tag{6.4}$$

Nachdem LBG terminiert ist, wird also die Einheit $a$ mit minimaler Nützlichkeit bestimmt

$$a := \arg\min_{c \in \mathcal{A}} U(c)$$

und die Einheit $b$ mit größtem assozierten Fehler:

$$b := \arg\max_{c \in A} E[c]$$

Um zu erreichen, daß die Voronoimenge von $b$ aufgeteilt wird zwischen den Einheiten $a$ und $b$, ist es sinnvoll, $\mathbf{w}_a$ nicht gleich $\mathbf{w}_b$ zu setzen, sondern noch

[3] Wie man sich leicht überlegt, würden zwei Einheiten, die gleiche Referenzvektoren haben, beide die Nützlichkeit Null besitzen. Allerdings kann nur eine (beliebige) von ihnen entfernt werden, ohne den Quantisierungsfehler zu erhöhen. Danach muß das Nützlichkeitsmaß neu berechnet werden, wobei sich für die verbleibende Einheit in der Regel eine Nützlichkeit verschieden von Null ergeben wird.

einen sehr kleinen zufälligen Versatzvektor zu addieren. Dieser Vektor hat den einzigen Zweck, $\mathbf{w}_a$ und $\mathbf{w}_b$ unterschiedlich zu machen, und verhindert so, daß eine von beiden Einheiten Gewinner für alle Daten in $\mathcal{R}_b$ wird. Bei gleichen Abständen würde ja nach Voraussetzung die Einheit mit dem kleinsten Index zum Gewinner erklärt. Daher wäre bei $\mathbf{w}_a = \mathbf{w}_b$ entweder $a$ oder $b$ Gewinner für *alle* Daten aus $\mathcal{R}_b$. Um zu garantieren, daß der Versatzvektor klein ist in Relation zum Durchmesser des Voronoigebiets, kann man sich überlegen, daß man mit Hilfe der schon berechneten Größe $E[\mathbf{w}_b]$ die Standardabweichung

$$\sqrt{E[\mathbf{w}_b]/|\mathcal{R}_b|}$$

der Elemente der Voronoimenge $\mathcal{R}_b$ ausdrücken kann. Dies läßt sich (wegen $|\mathcal{R}_b| \leq |\mathcal{D}|$ nach unten abschätzen durch den Ausdruck

$$\sqrt{E[\mathbf{w}_b]/|\mathcal{D}|},$$

der nur noch von der Anzahl aller Datenvektoren abhängt und nicht mehr von der Anzahl der Datenvektoren im Voronoigebiet von $\mathbf{w}_b$. Um den endgültigen Versatzvektor festzulegen, wählt man nun eine kleine Konstante $\epsilon$ und einen zufälligen Vektor $\mathbf{u}$ von der $n$-dimensionalen Einheitskugel, der die Richtung des Versatzvektors angibt.

Schließlich wird der Referenzvektor $w_a$ verändert gemäß

$$\mathbf{w}_a := \mathbf{w}_b + \mathbf{u}\,\varepsilon\sqrt{E[\mathbf{w}_b]/|\mathcal{D}|}.$$

Diese Abfolge (Durchführung von LBG bis zur Terminierung und anschließende Verschiebung eines Referenzvektors) wird wiederholt, solange sich die Kodebücher, zu denen LBG konvergiert, verbessern im Sinne eines reduzierten Fehlers $E[\mathcal{D}, \mathcal{A}]$. Ist dies nicht mehr der Fall, wird als Ergebnis das bisher beste Kodebuch verwendet (d.h. das vorletzte).

Das gesamte Verfahren LBG-U ist das folgende:

1. Initialisiere die Menge $\mathcal{A}$ mit $N$ Einheiten $c_i$

$$\mathcal{A} := \{c_1, c_2, \ldots, c_N\}.$$

   Initialisiere die Referenzvektoren aller Einheiten:

$$\mathbf{w}_c := (\text{zufällig aus } \mathcal{D}) \qquad (\forall c \in \mathcal{A})$$

2. Speichere den bisherigen Fehler:

$$E_{\mathrm{LBG-U}} := E[\mathcal{D}, \mathcal{A}]$$

3. Speichere das bisher beste Kodebuch:

$$\mathcal{K}_{\mathrm{LBG-U}} := \{\mathbf{w}_c | c \in \mathcal{A}\}$$

$$fertig := \text{FALSE}$$

4. WHILE ($\neg$ *fertig*) REPEAT:
   - Führe LBG durch bis sich der Fehler $E[\mathcal{D}, \mathcal{A}]$ nicht mehr ändert.
   - Speichere den aktuellen Fehler
   $$E_{\mathrm{LBG}} =: E[\mathcal{D}, \mathcal{A}]$$
   - IF ($E_{\mathrm{LBG}} >= E_{\mathrm{LBG-U}}$)
     THEN
     $$fertig := \mathrm{TRUE}$$
     ELSE
     - Speichere aktuell besten Fehler:
     $$E_{\mathrm{LBG-U}} := E_{\mathrm{LBG}}$$
     - Speichere aktuell bestes Kodebuch:
     $$\mathcal{K}_{\mathrm{LBG-U}} := \{\mathbf{w}_c | c \in \mathcal{A}\}$$
     - Bestimme am wenigsten nützliche Einheit $a$:
     $$a := \arg\min_{c \in \mathcal{A}} U(c)$$
     - Bestimme Einheit $b$ mit größtem zugeordneten Fehler
     $$b := \arg\max_{c \in A} E[c]$$
     - Wähle Zufallsvektor:
     $$\mathbf{u} = (\text{zufällig aus der } n\text{-dimensionalen Einheitskugel})$$
     - Positioniere die Einheit $a$ im Eingaberaum direkt neben $b$:
     $$\mathbf{w}_a := \mathbf{w}_b + \mathbf{u}\,\varepsilon\sqrt{E[\mathbf{w}_b]/|\mathcal{D}|}$$
5. Das Ergebnis ist das Kodebuch $\mathcal{K}_{\mathrm{LBG-U}}$.

Abbildung 6.4 zeigt ein Beispiel einer LBG-U-Simulation. Das gefundene Kodebuch weist einen deutlich geringeren Fehler auf als das, welches in Abbildung 6.2 von LBG erzeugt wurde. Dies liegt nicht etwa an einer günstigeren Initialisierung, wie man an dem Kodebuch sieht, zu dem LBG-U anfangs konvergiert (Abbildung 6.4b) und das das Endresultat von LBG darstellen würde. In der Tat weist dieses Kodebuch einen größeren Fehler auf, als das von LBG in Abbildung 6.2 gefundene. Erst die darauffolgenden nichtlokalen Verschiebungen einzelner Vektoren durch LBG-U führen zu einem insgesamt besseren Kodebuch. Auch das Kodebuch in Abbildung 6.3c), welches dort ohne Herkunftsangabe als bessere Alternative gezeigt wurde, ist mit LBG-U generiert worden.

In [Fritzke, 1997b] wurden systematische Untersuchungen mit LBG und -LBG-U durchgeführt. Der verwendete Datensatz (Abbildung 6.5) war durchaus

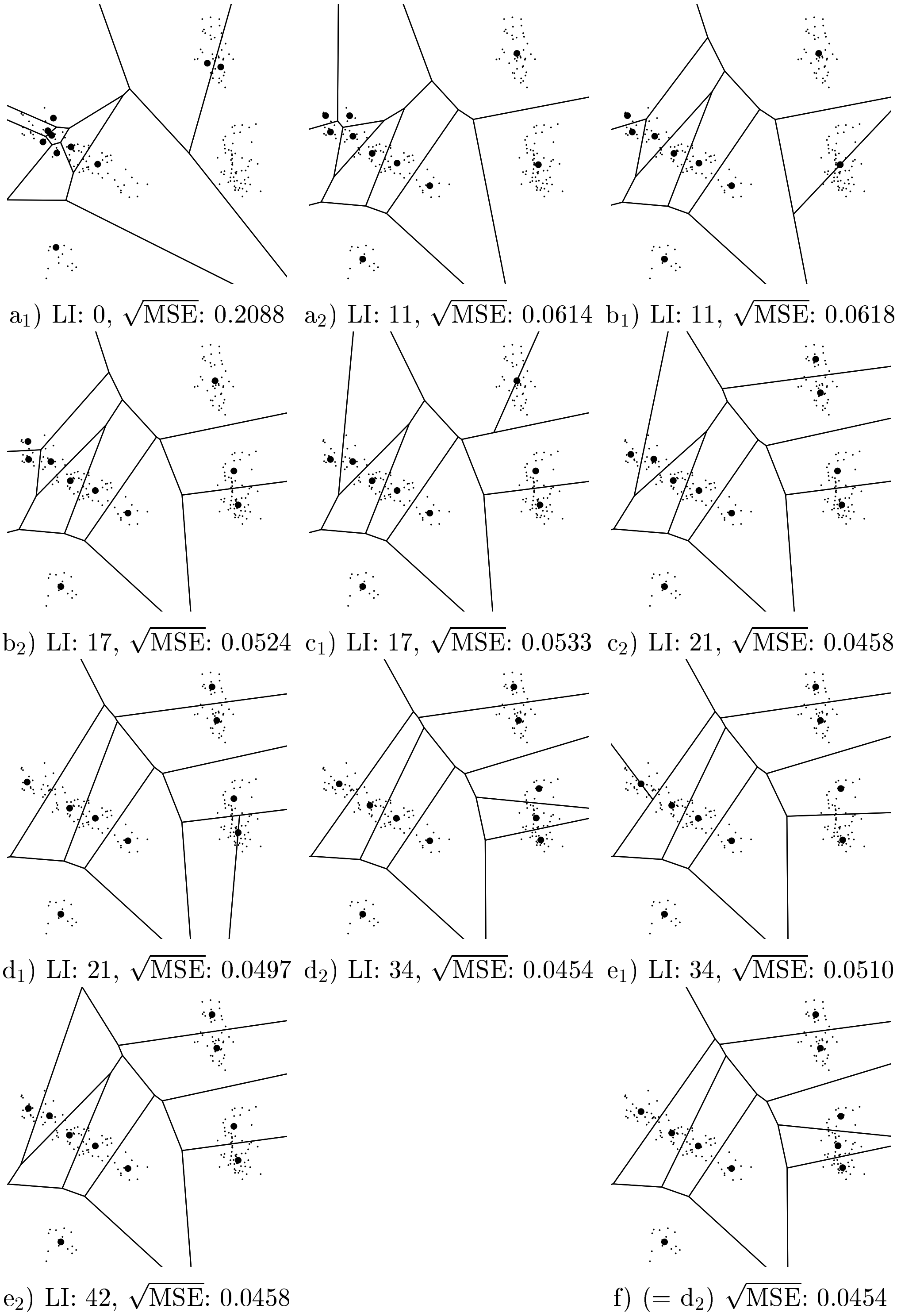



Abbildung 6.4: LBG-U-Simulation. LI steht für Lloyd-Iteration. (a$_1$) zeigt den Initialzustand und (a$_2$) das Ergebnis von LBG, welches in diesem Fall zu einem noch höheren Quantisierungsfehler als die Simulation in Abbildung 6.2 führt. In (b$_1$),(c$_1$),(d$_1$) und (e$_1$) ist jeweils die Situation nach der Verschiebung einer Einheit gezeigt und in (b$_2$),(c$_2$),(d$_2$) und (e$_2$) das lokale Minimum, zu dem LBG anschließend konvergierte. Da in (e$_2$) ein höherer Fehler auftrat als in (d$_2$), wird als Endresultat (f) von LBG-U die in (d$_2$) erreichte Konfiguration gewählt.

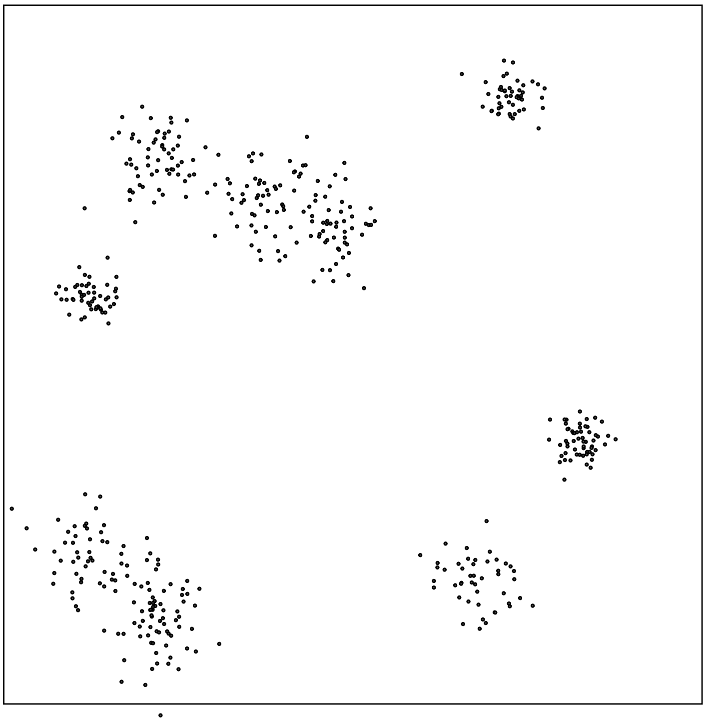


Abbildung 6.5: In [Fritzke, 1997b] verwendeter Datensatz zum Vergleich von LBG und LBG-U. Ein Datenpunkt liegt außerhalb des gezeigten Einheitsquadrates.

geeignet für LBG. Zwar waren die Daten in Clustern angeordnet, jedoch ohne große Unterschiede in der Wahrscheinlichkeitsdichte der verschiedenen Cluster (wie z.B. bei dem in Abbildung 6.3 gezeigten Datensatz vorhanden). Trotzdem wurden von LBG-U signifikant bessere Kodebücher gefunden (Abbildung 6.6).

LBG-U läßt sich interpretieren als eine Strategie, gezielte Sprünge im Raum aller möglich Kodebücher durchzuführen. Diese Sprünge erlauben das Verlassen lokaler Minima, die sich nach Terminierung von LBG ergeben. Der konzeptuelle Rahmen des harten Wettbewerbslernens wird durch diese Sprünge verlassen. Sie beseitigen in hohem Maße die Abhängigkeit von der Initialisierung, die harte Wettbewerbslernverfahren sonst kennzeichnet.

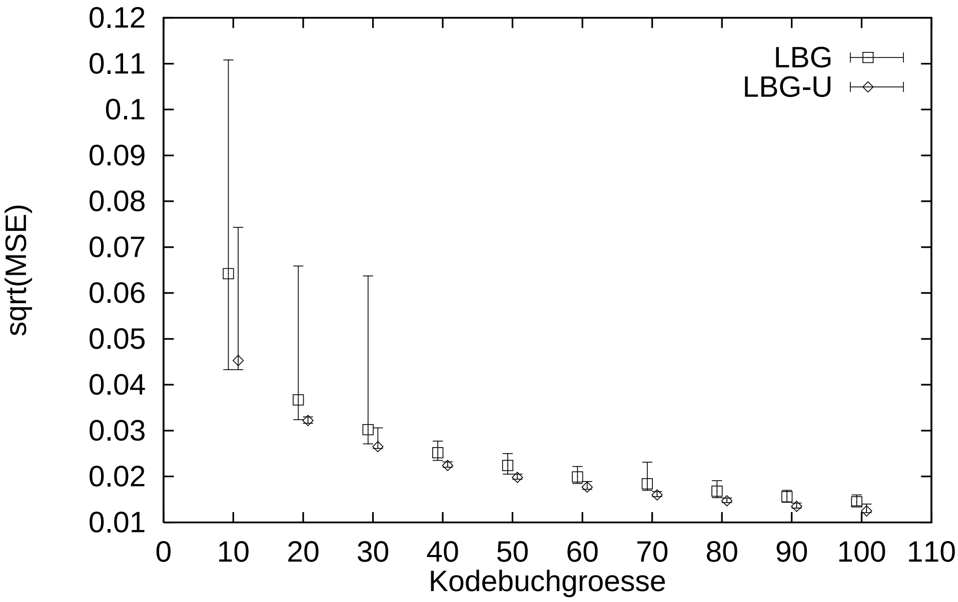



Abbildung 6.6: Vergleich von LBG und LBG-U anhand des Datensatzes aus Abbildung 6.5. Die Quadratwurzel aus mittlerem quadratischen Fehler (hier bezeichnet mit RMSE) ist über der Kodebuchgröße aufgetragen. Auch Minimum und Maximum des Fehlers wurden jeweils angegeben. Zehn Kodebücher der Größe 10 bis 100 wurden generiert. Pro Kodebuchgröße wurden jeweils 100 Simulationen durchgeführt (d.h., insgesamt 1000 Simulationen). Das LBG-U-Verfahren ermittelte für diese Daten nicht nur durchgängig bessere Codebücher als LBG, auch die Spanne zwischen bestem und schlechtestem Kodebuch war erheblich geringer. Dies hängt vermutlich damit zusammen, daß die von LBG-U gefundenen Lösungen alle relativ nahe am (unbekannten) globalen Optimum liegen

# Kapitel 7

# Hartes Wettbewerbslernen für große und unendliche Datenmengen

## 7.1 *Online*-Lernen: Grundprinzip

Bei vielen praktischen Problemen ist die Datenmenge $\mathcal{D}$ so groß, daß Batch-Verfahren, die definitionsgemäß die vollständige Auswertung aller Datenpunkte vor einer Adaption erfordern, unhandlich werden. Im Extremfall liegen die Daten als Sequenz potentiell unendlicher Länge vor (z.B. wenn die Daten aus einem kontinuierlich laufenden Sensor stammen), was die Verarbeitung durch Batch-Verfahren völlig unmöglich macht.

Ein Ausweg in dieser Situation sind *Online*-Verfahren, also Methoden, bei denen nach jedem Datensignal oder nach einer geringen Anzahl von Datensignalen eine Adaption der Systemparameter stattfindet. *Online*-Verfahren setzen nicht mehr die Existenz einer endlichen Datenmenge $\mathcal{D}$ voraus, sondern nur noch die Möglichkeit, beliebig viele Datensignale gemäß einer (im allgemeinen unbekannten) Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ zu erzeugen.

Im Bereich des harten Wettbewerbslernens existiert eine ganze Reihe verschiedener Online-Verfahren. Der grundsätzliche Ablauf ist dabei stets der folgende:

1. Initialisiere die Menge $\mathcal{A}$ mit $N$ Einheiten $c_i$

   $$\mathcal{A} := \{c_1, c_2, \ldots, c_N\}.$$

   Initialisiere die Referenzvektoren aller Einheiten:

   $$\mathbf{w}_c := (\text{zufällig gemäß } p(\boldsymbol{\xi})) \qquad (\forall c \in \mathcal{A})$$

   Initialisiere den Zeitparameter: $\quad t := 0$

2. Generiere ein Eingabesignal: $\quad \boldsymbol{\xi} := (\text{zufällig gemäß } p(\boldsymbol{\xi}))$

3. Bestimme den Gewinner $s = s(\boldsymbol{\xi})$ für das aktuelle Eingabesignal.

4. Inkrementiere den Zeitparameter: $t := t + 1$
5. Adaptiere den Referenzvektor des Gewinners mit der zeitabhängigen Lernrate $\epsilon(t)$:
$$\mathbf{w}_s := \mathbf{w}_s + \epsilon(t)(\boldsymbol{\xi} - \mathbf{w}_s)$$
6. Wenn $t < t_{\max}$, fahre fort mit Schritt 2.

Eine konkrete Realisierung des dargestellten Grundschemas ergibt sich durch Festlegung der Zeitabhängigkeit für die Lernrate $\epsilon(t)$. In den folgenden Abschnitten werden einige wichtige Möglichkeiten dafür dargestellt.

## 7.2 Konstante Lernrate

Eine besonders einfache Form des harten Wettbewerbslernens ergibt sich durch die Verwendung einer *zeitlich konstanten* Lernrate:

$$\epsilon(t) = \epsilon_0, \ (0 \leq \epsilon_0 \leq 1). \tag{7.1}$$

In diesem Fall ist jeder Referenzvektor $\mathbf{w}_c$ ein exponentieller gleitender Durchschnitt der Eingabesignale, für die seine Einheit $c$ bisher Gewinner gewesen ist. Sei $\boldsymbol{\xi}_1^c, \boldsymbol{\xi}_2^c, \ldots, \boldsymbol{\xi}_t^c$ die (Unter-)Folge der Eingabesignale, für die eine bestimmte Einheit $c$ der Gewinner war. Ihr Referenzvektor $\mathbf{w}_c$ nimmt dann nacheinander die folgenden Werte an (Beweis von Gleichung (7.2) in Anhang A.2):

$$\begin{aligned}
\mathbf{w}_c(0) &= (\text{zufällig gemäß } p(\boldsymbol{\xi})) \\
\mathbf{w}_c(1) &= (1-\epsilon_0)\mathbf{w}_c(0) + \epsilon_0 \boldsymbol{\xi}_1^c \\
&\vdots \\
\mathbf{w}_c(t) &= (1-\epsilon_0)^t \mathbf{w}_c(0) + \epsilon_0 \sum_{i=1}^{t} (1-\epsilon_0)^{t-i} \boldsymbol{\xi}_i^c.
\end{aligned} \tag{7.2}$$

Aus Gleichung (7.2) geht hervor, daß der Einfluß vergangener Signale auf den augenblicklichen Wert von $\mathbf{w}_c$ exponentiell schnell abnimmt (Siehe auch Abbildung 7.1). Der Einfluß des jeweils letzten Signals, für das eine Einheit Gewinner ist, ist stets gleich $\epsilon_0$. Dies hat zwei direkte Konsequenzen. Zum einen bleibt das System immer adaptiv und ist daher im Prinzip in der Lage, auch nichtstationären Wahrscheinlichkeitsverteilungen zu folgen. Andererseits gibt es keine Konvergenz, da auch nach vielen Eingabesignalen das zuletzt aufgetretene Signal noch eine erhebliche Adaption verursacht.

Ein typisches Verhalten eines derartigen Systems ist das folgende: Die Referenzvektoren driften von ihren Ausgangspositionen zu *quasi-stationären* Positionen, nämlich den Zentroiden ihrer Voronoigebiete, wo sie um ein dynamisches Gleichgewicht fluktuieren. *Bessere* quasi-stationäre Positionen im Sinne eines mittleren quadratischen Fehlers werden mit kleineren Lernraten erreicht, da in diesem Fall die Fluktuationen um den Zentroid des Voronoigebiets geringer

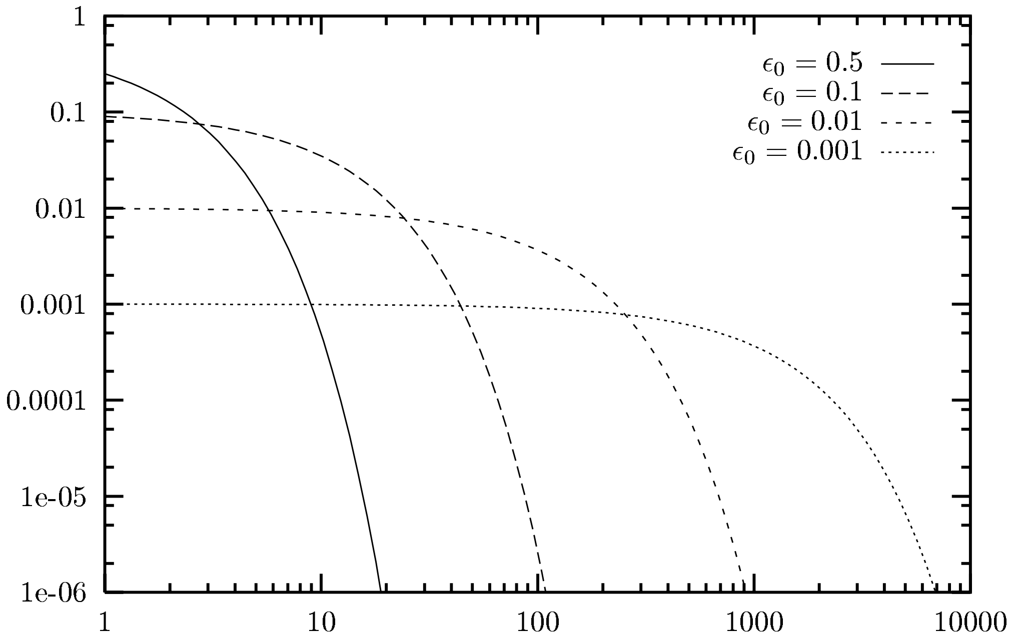



Abbildung 7.1: Einfluß des Eingabesignals $\boldsymbol{\xi}^0$ auf den Vektor des Gewinners $s$ aufgetragen als Funktion der Anzahl der Eingabesignale (inklusive $\boldsymbol{\xi}^0$), für die $s$ Gewinner ist. Die Abhängigkeit für verschiedene konstante Lernraten $\epsilon_0$ ist gezeigt. Der jeweilige Schnittpunkt mit der $x$-Achse gibt an, wann der Einfluß von $\boldsymbol{\xi}_0$ unter den Wert $10^{-6}$ gefallen ist. Für $\epsilon_0 = 0.5$ beispielsweise ist dieser Punkt schon nach 10 weiteren Signalen erreicht.

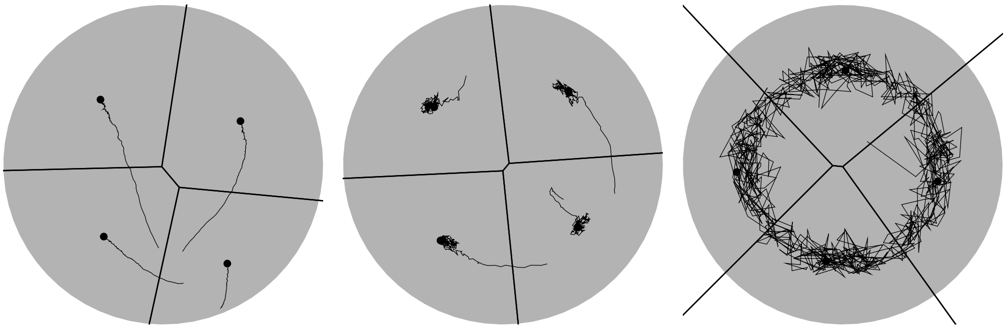
sind. *Schnelleres* Erreichen der quasi-stationären Positionen erfordert hingegen größere Lernraten. Abbildung 7.2 zeigt den Effekt von drei verschiedenen Lernraten für ein einfaches Beispiel.

## 7.3 $k$-means

Anstatt die Lernrate zeitlich konstant zu lassen, kann man sie auch schrittweise absenken (eine Erhöhung der Lernrate wird normalerweise nicht in Betracht gezogen). Eine besonders interessante Art, dies zu tun, ist, für jede Einheit $c$ des Netzwerks eine separate Lernrate $\epsilon_c(t)$ einzuführen und diese gemäß der harmonischen Folge zu modifizieren:

$$\epsilon_c(t) = \frac{1}{t} \tag{7.3}$$

Der Zeitparameter $t$ steht dabei für die Anzahl der Signale, für die die Einheit $c$ Gewinner gewesen ist. Diese resultierende Methode ist unter dem Namen *k-means* bekannt, was soviel heißt wie „$k$ Mittelwerte“. Diese Bezeichnung ist in einem bestimmten Sinne sehr angemessen, da jeder Referenzvektor $\mathbf{w}_c$ stets der Mittelwert aller Eingabesignale $\boldsymbol{\xi}_1^c, \boldsymbol{\xi}_2^c, \ldots, \boldsymbol{\xi}_t^c$ ist, für die die Einheit $c$ bisher Gewinner war (Beweis von Gleichung 7.4 in Anhang A.3):



a) $\epsilon_0 = 0.001$ b) $\epsilon_0 = 0.01$ b) $\epsilon_0 = 0.1$

Abbildung 7.2: Hartes Wettbewerbslernen mit konstanter Lernrate. Dargestellt ist, wie vier Referenzvektoren durch 20000 Signale aus der gezeigten Gleichverteilung adaptiert werden. Jeweils nach 100 Signalen wurde die augenblickliche Position eingezeichnet. Drei verschiedene Lernraten wurden verwendet.

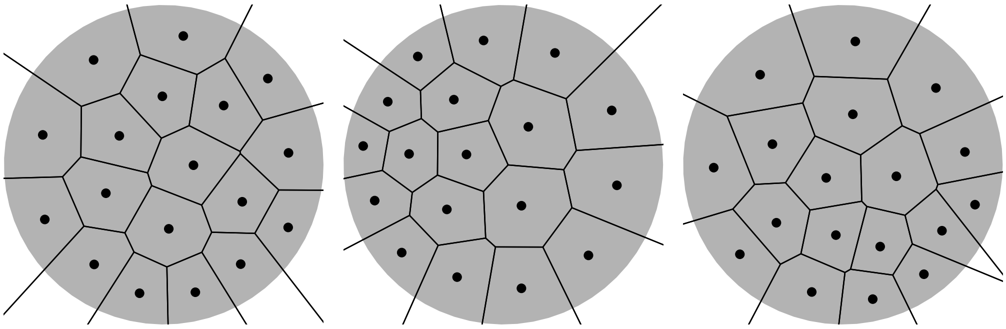
$$\begin{aligned}
\mathbf{w}_c(0) &= (\text{zufällig gemäß } p(\boldsymbol{\xi})) \\
\mathbf{w}_c(1) &= \mathbf{w}_c(0) + \epsilon_c(1)(\boldsymbol{\xi}_1^c - \mathbf{w}_c(0)) \\
&= \boldsymbol{\xi}_1^c \\
&\vdots \\
\mathbf{w}_c(t) &= \frac{1}{t}\sum_{i=1}^{t} \boldsymbol{\xi}_i^c \qquad (7.4)
\end{aligned}$$

Für den Sonderfall, daß nur ein einziger Referenzvektor existiert (also $k = |A| = 1$), ist die $k$-means Lernrate optimal, denn bei eine endlichen Stichprobe ist bekanntermaßen deren Mittelwert der beste Schätzwert für den Erwartungswert der zugrundeliegenden Wahrscheinlichkeitsverteilung. Dieser Zusammenhang gilt aber nicht mehr, wenn es mehrere Referenzvektoren gibt. In diesem (Normal-)Fall kann die Folge der Eingabesignale, für die eine Einheit irgendwann Gewinner war, durchaus Elemente enthalten kann, die außerhalb des *derzeitigen* Voronoigebietes $V_c$ dieser Einheit liegen. Der Grund dafür ist, daß jede Adaption eines Referenzvektors einige Voronoigebiete verändert und Eingabesignale, für die $c$ Gewinner gewesen ist, können daher später aus dem Voronoigebiet „herausfallen“.

Wie man sich leicht überlegt, gibt es für $k$-means keine eigentliche Konvergenz, da die harmonische Reihe divergiert:

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{i} = \infty. \qquad (7.5)$$

Allerdings ist bewiesen worden [MacQueen, 1965], daß $k$-means *asymptotisch* zu einer Konfiguration konvergiert, wo jeder Referenzvektor zusammenfällt



a) b) c)

Abbildung 7.3: $k$-means. Dargestellt sind drei verschiedene Konfigurationen, die nach jeweils 40000 Eingabesignalen erreicht wurden. Die Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ ist uniform in dem gezeigten kreisförmigen Gebiet. Trotzdem sind die erreichten Konfigurationen relativ unterschiedlich (Vergleiche dagegen Abbildung 7.5)

mit dem Erwartungswert der Eingabesignale in seinem Voronoigebiet, also dem Zentroid des Voronoigebiets:

$$E(\boldsymbol{\xi}|\boldsymbol{\xi} \in V_c) = \int_{V_c} \boldsymbol{\xi} p(\boldsymbol{\xi}) d\boldsymbol{\xi}. \tag{7.6}$$

Da dieses Resultat jedoch nur für den Grenzfall unendlich vieler Eingabesignale gilt, können die Resultate von $k$-means in endlich langen Simulationen davon abweichen. In der Tat scheint dies ein ernstzunehmendes Problem darzustellen, wie Untersuchungen von Darken & Moody [1990] zeigen. Abbildung 7.3 zeigt Simulationsresultate von $k$-means für eine einfache Verteilung.

## 7.4 Exponentiell abnehmende Lernrate

Eine weitere Möglichkeit, die Lernrate abzusenken, wurde von Ritter et al. [1991] im Kontext selbstorganisierender Karten vorgeschlagen. Der Vorschlag ist, eine exponentielle Abhängigkeit gemäß

$$\epsilon(t) = \epsilon_i (\epsilon_f / \epsilon_i)^{t/t_{\max}}, \tag{7.7}$$

zu verwenden, wobei $\epsilon_i$ und $\epsilon_f$ die Initial- bzw. Endwerte der Lernrate sind und $t_{\max}$ die Gesamtanzahl der Adaptionsschritte.

In Abbildung 7.4 wird diese Lernrate für einen bestimmten Parametersatz verglichen mit der harmonischen Folge, die als Lernrate von $k$-means verwendet wird. Insbesondere zu Beginn der Simulation ist die exponentiell abfallende Lernrate viel größer als die harmonische Folge. Dies bedeutet, daß alte Signale schnell wieder „vergessen“ werden, also die spätere Position der Referenzvektoren nicht wesentlich beeinflussen. Auch bei konstanten Lernraten spielen frühere Signale keine wichtige Rolle, wie in Abschnitt 7.2 gezeigt wurde. Hier jedoch wird die Lernrate im Laufe der Zeit herabgesenkt, so daß es schließlich zu einer

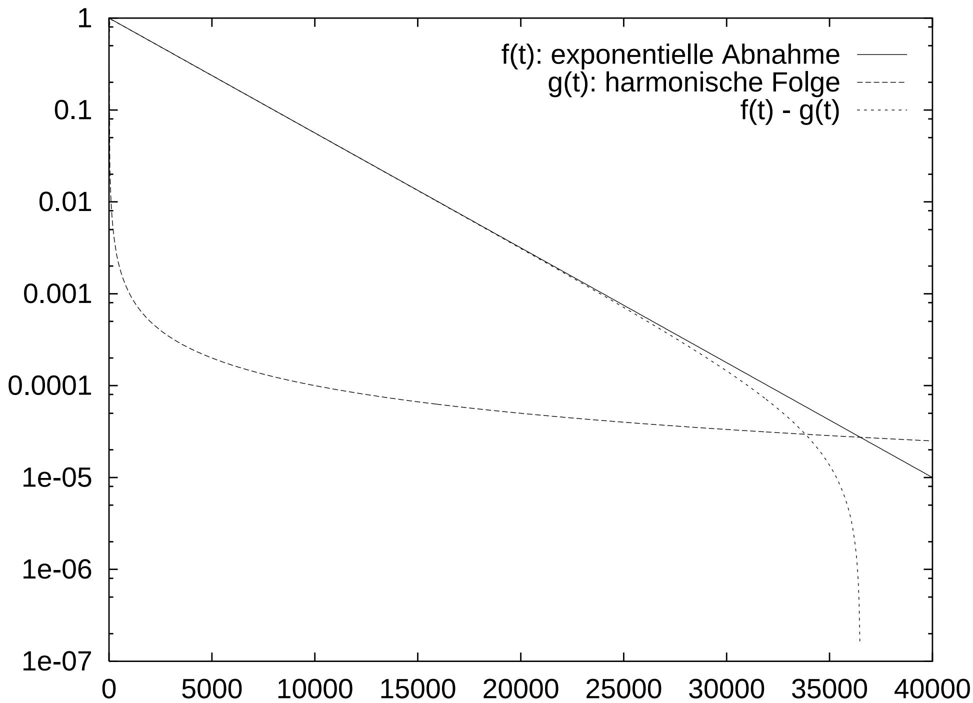



Abbildung 7.4: Vergleich der exponentiell abnehmenden Lernrate $f(t) = \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{\max}}$ und der bei $k$-means als Lernrate verwendeten harmonischen Folge $g(t) = 1/t$ für einen bestimmten Satz von Parametern ($\epsilon_i = 1.0$, $\epsilon_f =$ 1E-5, $t_{\max} = 40000$). Die gezeigte Differenz zwischen den Lernraten kann als statistisches Rauschen interpretiert werden, das dem System hinzugefügt und dann graduell wieder entfernt wird.

Konvergenz des Systems kommt. Diese Vorgehensweise hat gewisse Parallelen zu „Simuliertem Abkühlen“ (engl. simulated annealing) [Kirkpatrick et al., 1983]. Simuliertes Abkühlen fügt einem System statistisches Rauschen hinzu (oft gesteuert durch einen „Temperatur“-Parameter) welches im Laufe der Zeit immer geringer wird. Die Konsequenz ist, daß das System, während es versucht einen Fehler zu minimieren, lokale Minima wieder verlassen kann. Wenn das System „abgekühlt“ wird, verschwindet diese Fähigkeit und es wird schließlich nur noch die eigentliche Minimierungsstrategie (z.B. Gradientenabstieg) verfolgt. Die lokalen Minima, die mit simuliertem Abkühlen erreicht werden, sind oft wesentlich besser, als die mit reinem Gradientenanstieg erreichten.

Schon Darken & Moody [1990] haben $k$-means mit verschieden anderen Lernraten verglichen, die anfangs größer als die harmonische Folge waren und kamen zu dem Ergebnis, daß $k$-means nur für den Fall $k = 1$ besser ist.

In Abbildung 7.5 sind Simulationsresultate für eine Gleichverteilung in einem Kreis gezeigt, wobei eine expnentiell sinkende Lernrate verwendet wurde. Vergleicht man diese mit den entsprechenden Resultaten für $k$-means (Abbildung 7.3) so wird deutlich, daß die exponentiell sinkende Lernrate bei diesem Problem sowohl bessere als auch konsistentere Ergebnisse liefert. Während die mit $k$-means erzielten Ergebnisse stark voneinander abweichen, sind sich die

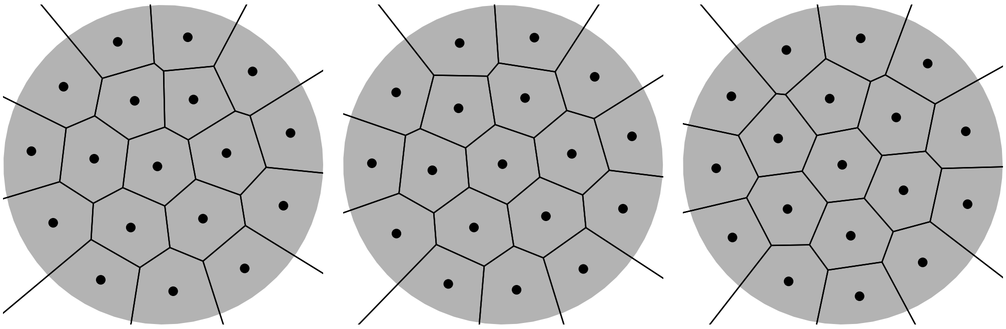



Abbildung 7.5: Hartes Wettbewerbslernen mit exponentiell absinkender Lernrate (Parameter wie in Abbildung 7.4). Dargestellt sind drei verschiedene Konfigurationen, die – ausgehend von unterschiedlichen zufälligen Initialkonfigurationen – nach jeweils 40000 Eingabesignalen erreicht wurden. Die Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ ist uniform in dem gezeigten kreisförmigen Gebiet. Die erreichten Konfigurationen sind sich sehr ähnlich (Vergleiche dagegen Abbildung 7.3)

mit der exponentiell abnehmenden Lernrate erzielten sehr ähnlich.

# Kapitel 8

# Weiches Wettbewerbslernen ohne feste Netzwerkdimension

In diesem und dem folgenden Kapitel geht es um *weiches Wettbewerbslernen*. Darunter werden Wettbewerbslernverfahren verstanden, bei denen nicht ausschließlich der Gewinner bezüglich des aktuellen Eingabesignals adaptiert wird, sondern weitere, eventuell sogar alle, Einheiten des Netzwerks.

## 8.1 Neuronales Gas

Dieses Verfahren [Martinetz & Schulten, 1991] ist dadurch gekennzeichnet, daß für jedes Eingabesignal $\boldsymbol{\xi}$ alle Einheiten des Netzwerks nach ihrem Abstand zu $\boldsymbol{\xi}$ sortiert werden. Anschließend wird jede Einheit mit einer Stärke adaptiert, die von der Position innerhalb der so erzeugten „Rangliste" abhängt. Die Einheit an der Spitze der Rangliste wird am stärksten adaptiert. Es gibt dabei zwei verschiedene Zeitabhängigkeiten. Einerseits ist die Lernrate definiert als

$$\epsilon(t) := \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{\max}} \qquad (0 \le t \le t_{\max}),$$

variiert also monoton zwischen einem Anfangswert $\epsilon_i$ und einem – grundsätzlich kleineren – Endwert $\epsilon_f$.

Andererseits geht als weiterer Faktor in die Adaption eine Größe $h(t,k)$ ein, die als *Nachbarschaftsreichweite* angesehen werden kann, die von der Zeit $t$ und von der Position $k$ der jeweils betrachteten Einheit in der Rangliste abhängt. Für den Gewinner hat $k$ den Wert 1, für die weiteren Einheiten jeweils 2,3, usw. Die Definition von $h(t,k)$ lautet

$$h(t,k) := \exp\left(\frac{1-k}{\lambda(t)}\right),$$

wobei $\lambda(t)$ definiert ist als

$$\lambda(t) := \lambda_i(\lambda_f/\lambda_i)^{t/t_{\max}} \qquad (0 \le t \le t_{\max}).$$

Die Größe $\lambda(t)$ beeinflußt, wie stark eine Einheit an einer bestimmten Position in der Rangliste bezüglich des aktuellen Eingabesignals adaptiert wird. Der

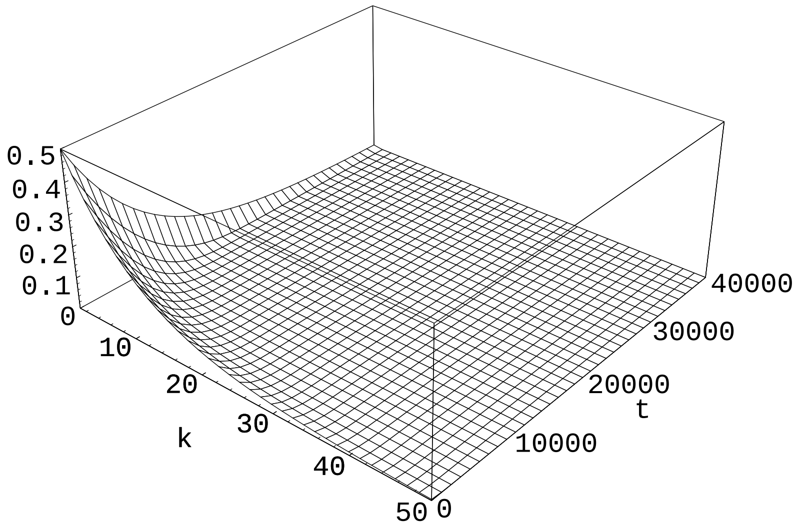



Abbildung 8.1: Zeitliche Entwicklung der Gesamtadaptionsstärke $\epsilon(t)h(t,k)$ in Abhängigkeit von Ranglistenplatz $k$. Parameter: $t_{\max} = 40000$, $\epsilon_i = 0.5$, $\epsilon_f = 0.005$, $\lambda_i = 10$, $\lambda_f = 0.01$

Anfangswert $\lambda_i$ wird stets größer als der Endwert $\lambda_f$ gewählt, so daß am Anfang einer Simulation viele Einheiten zusammen mit dem Gewinner angepaßt werden, am Ende jedoch nur noch der Gewinner selbst.

Die Gesamtadaptionsstärke der $k$-ten Einheit in der Rangliste zu einem Zeitpunkt $t$ ist das Produkt von Lernrate $\epsilon(t)$ und Nachbarschaftsreichweite $h(t,k)$. Der Gewichtsvektor dieser Einheit wird entsprechend gemäß

$$\Delta \mathbf{w}_{s_k} := \epsilon(t)\, h(t,k)\, (\boldsymbol{\xi} - \mathbf{w}_{s_k})$$

adaptiert.

Abbildung 8.1 zeigt für typische Parameter die Adaptionsstärke in Abhängigkeit von Ranglistenplatz $k$ und Zeit $t$.

Das komplette Verfahren „Neuronales Gas“ ist im folgenden dargestellt:

1. Initialisiere die Menge $\mathcal{A}$ mit $N$ Einheiten $c_i$

   $$\mathcal{A} := \{c_1, c_2, \ldots, c_N\}.$$

   Initialisiere die Referenzvektoren aller Einheiten:

   $$\mathbf{w}_c := (\text{zufällig gemäß } p(\boldsymbol{\xi})) \qquad (\forall c \in \mathcal{A})$$

   Initialisiere den Zeitparameter: $t := 0$

2. Generiere ein Eingabesignal: $\boldsymbol{\xi} := (\text{zufällig gemäß } p(\boldsymbol{\xi}))$

3. Sortiere alle Elemente von $\mathcal{A}$ gemäß ihrer Distanz zu $\boldsymbol{\xi}$, d.h., finde eine Sequenz $s_1, s_2, \ldots, s_N, (s_i \in \mathcal{A})$, derart, daß

   $$\|\boldsymbol{\xi} - \mathbf{w}_{s_1}\| \leq \|\boldsymbol{\xi} - \mathbf{w}_{s_2}\| \leq \ldots \leq \|\boldsymbol{\xi} - \mathbf{w}_{s_N}\|$$

4. Adaptiere die Referenzvektoren gemäß

$$\Delta \mathbf{w}_{s_k} := \epsilon(t) \cdot h(t,k) \cdot (\boldsymbol{\xi} - \mathbf{w}_{s_k}) \quad \text{für } k = 1, \ldots, N$$

mit den folgenden Zeitabhängigkeiten:

$$\begin{aligned} \epsilon(t) &:= \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{\max}}, \\ \lambda(t) &:= \lambda_i(\lambda_f/\lambda_i)^{t/t_{\max}}, \\ h(t,k) &:= \exp((1-k)/\lambda(t)) \end{aligned}$$

5. Inkrementiere den Zeitparameter: $t := t + 1$

6. Wenn $t < t_{\max}$, fahre fort mit Schritt 2.

Für die zeitabhängigen Parameter müssen geeignete Startwerte ($\lambda_i$, $\epsilon_i$) und Endwerte ($\lambda_f$, $\epsilon_f$) gewählt werden.

Abbildung 8.2 zeigt eine Simulation dieses Modells für eine ringförmige Wahrscheinlichkeitsverteilung. Die folgenden (von Martinetz et al. [1993] angegebenen) Parameter wurden verwendet: $\lambda_i = 10$, $\lambda_f = 0.01$, $\epsilon_i = 0.5$, $\epsilon_f = 0.005$, $t_{\max} = 40000$.

Martinetz [1992] konnte zeigen, daß die Adaptionsschritte, die bei dem Verfahren „Neuronales Gas" vorgenommen werden, im Mittel einem Gradientenabstieg auf einer *Energie- bzw. Kostenfunktion* entsprechen. Dies ist insofern bemerkenswert, da z.B. für die selbstorganisierenden Merkmalskarten von Kohonen (siehe Kapitel 9.1) bewiesen wurde, daß eine Energiefunktion für dieses Verfahren nicht existiert [Erwin et al., 1992]. Die Energiefunktion, die durch das Verfahren „Neuronales Gas" minimiert wird ist übrigens *nicht* der Quantisierungsfehler (5.1), zumindest nicht solange noch $\lambda \neq 0$ ist, d.h., solange Nachbareinheiten des Gewinners mitadaptiert werden.

## 8.2 Hebb'sches Wettbewerbslernen

Hebb'sches Wettbewerbslernen [Martinetz & Schulten, 1991; Martinetz, 1993] stellt eine elegante und einfache Methode dar, eine existierende Menge $\mathcal{A}$ von Einheiten (mit zugeordneten Referenzvektoren im $\mathrm{R}^n$) mit einer topologischen Struktur zu versehen, die oft in hohem Maße die Topologie der Eingabedatenverteilung $p(\boldsymbol{\xi})$ wiederspiegelt.

Der zentrale Schritt des Verfahrens ist, für das aktuelle Eingabesignal $\boldsymbol{\xi}$ den Gewinner $s_1$ und die zweitnächste Einheit $s_2$ zu bestimmen und dann eine Kante zwischen $s_1$ und $s_2$ einzufügen (sofern sie nicht bereits existiert). Die Referenzvektoren bleiben dabei völlig unverändert. Martinetz [1993] konnte zeigen, daß jede so eingefügte Kante eine Kante der zu den Referenzvektoren korrespondierenden Delaunaytriangulation ist. Der entstehende Graph ist also ein Untergraph der Delaunaytriangulation. Da Kanten nur in der Nähe von Eingabesignalen generiert werden, ist der erzeugte Untergraph im wesentlichen beschränkt auf Gebiete, wo $p(\boldsymbol{\xi}) > 0$. Insbesondere muß sich das Voronoigebiet eines der Endpunkte jeder Kante überschneiden mit einem Bereich des $\mathrm{R}^n$,

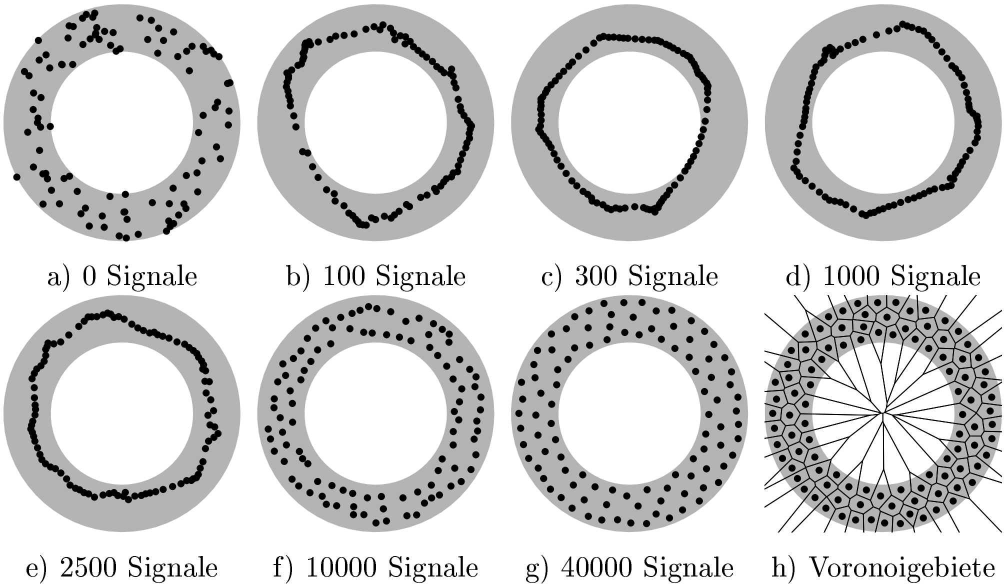



Abbildung 8.2: Neuronales Gas. Simulationssequenz mit einer Wahrscheinlichtsverteilung, die uniform in einem ringförmigen Gebiet ist. a) Initialzustand b-f) Zwischenzustände. g) Endkonfiguration. h) zur Endkonfiguration korrespondierende Voronoitessellation. Die anfangs starke Nachbarschaftsadaption führt zu einer lokalen Konzentration der Referenzvektoren. Wenn die Nachbarschaftsadaption geringer wird, verteilen sich die Vektoren gleichmäßig über das Gebiet positiver Wahrscheinlichkeitsdichte.

wo $p(\boldsymbol{\xi}) > 0$ gilt. In Abbildung 8.3 ist gezeigt, wie einige Eingabesignale zu entsprechenden Kanten führen.

Hebb'sches Wettbewerbslernen verändert die Referenzvektoren nicht und ist deshalb auch nicht in der Lage, Positionen für sie zu finden. Es gibt zwei grundsätzliche Arten, Hebb'sches Wettbewerbslernen einzusetzen. Einerseits kann es im Anschluß an ein Verfahren, welches Referenzvektoren positioniert, verwendet werden, um eine Topologie zu generieren. Eine andere Möglichkeit ist, parallel zur Adaption der Referenzvektoren die Topologie durch Hebb'sches Wettbewerbslernen zu erzeugen. In diesem Fall ändert sich jedoch die zu den Referenzvektoren korrespondierende Delaunaytriangulation ständig und Kanten, die zu einem früheren Zeitpunkt durch Hebb'sches Wettbewerbslernen eingefügt wurden, bleiben daher nicht notwendig Kanten der Delaunaytriangulation. Daher sind hier Mechanismen erforderlich, die ein Entfernen von Kanten zulassen (siehe Abschnitte 8.3 und 8.4).

Im folgenden beschreiben wir die Variante von Hebb'schem Wettbewerbslernen, die von einer sich nicht ändernden Menge von Referenzvektoren ausgeht. Diese Menge kann zufällig initialisiert werden (wie hier angenommen) oder auch von einem separaten Verfahren, welches die Vektoren positioniert.

1. Initialisiere die Menge $\mathcal{A}$ mit $N$ Einheiten $c_i$

$$\mathcal{A} := \{c_1, c_2, \ldots, c_N\}.$$

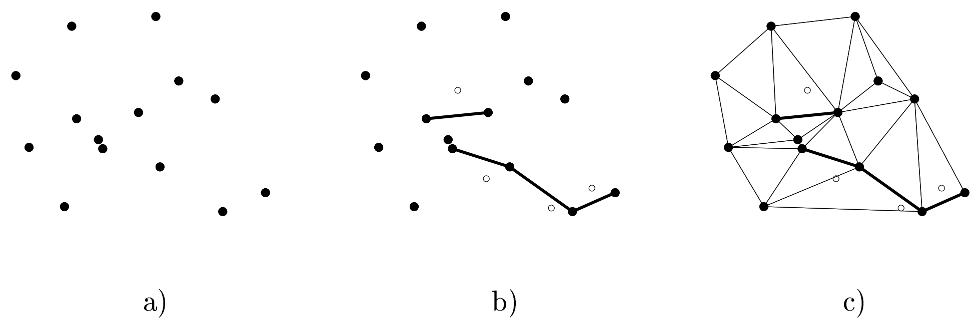



Abbildung 8.3: Hebb'sches Wettbewerbslernen. a) Referenzvektoren in $\mathrm{R}^2$, b) 4 Eingabesignale (weiß) und die durch Hebb'sches Wettbewerbslernen erzeugten Kanten c) vollständige Delaunaytriangulation.

Initialisiere die Referenzvektoren aller Einheiten:

$$\mathbf{w}_c := (\text{zufällig gemäß } p(\boldsymbol{\xi})) \qquad (\forall c \in \mathcal{A})$$

Initialisiere die Verbindungsmenge $\mathcal{C}$ , $\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$, mit der leeren Menge:

$$\mathcal{C} := \emptyset$$

Initialisiere den Zeitparameter: $t := 0$

2. Generiere ein Eingabesignal: $\boldsymbol{\xi} := (\text{zufällig gemäß } p(\boldsymbol{\xi}))$

3. Bestimme die Gewinnereinheit $s_1$ und die zu $\boldsymbol{\xi}$ zweitnächste Einheit $s_2$.

4. Falls noch keine Verbindung zwischen $s_1$ und $s_2$ existiert, schaffe eine:

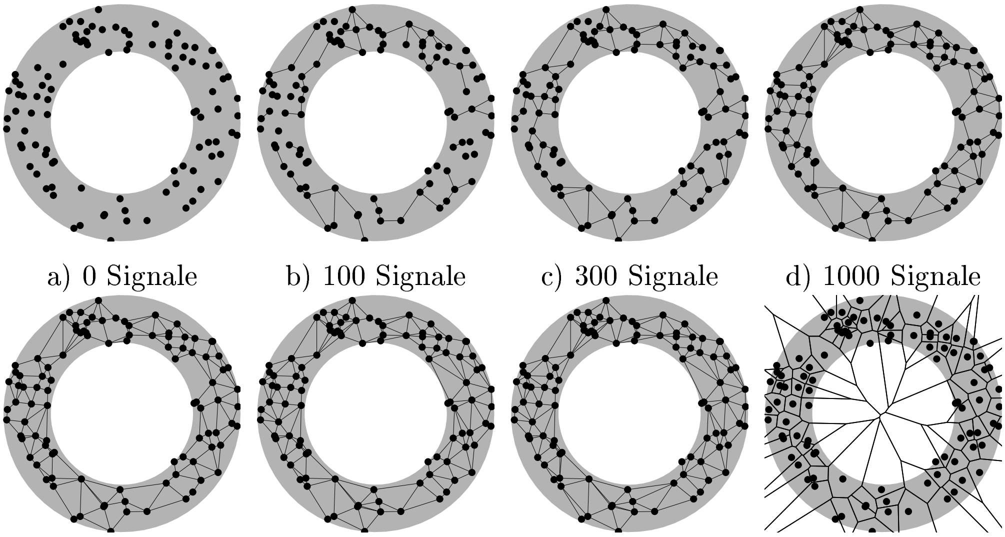
$$\mathcal{C} := \mathcal{C} \cup \{(s_1, s_2)\}$$

5. Inkrementiere den Zeitparameter: $t := t + 1$

6. Wenn $t < t_{\max}$, fahre fort mit Schritt 2.

Abbildung 8.4 zeigt eine Simulation dieses Modells für eine ringförmige Wahrscheinlichkeitsverteilung.

Der einzige Parameter dieses Verfahrens ist die Anzahl $t_{\max}$ der Eingabesignale. Je größer $t_{\max}$ gewählt wird, desto wahrscheinlicher ist es, daß eine Kante, die gemäß der (unbekannten) zugrundeliegenden Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ präsent sein müßte, auch erzeugt wird. Es gibt allerdings Konstellationen der Referenzvektoren (z.B. wenn vier Referenzvektoren *fast* auf einem Kreis liegen), wo es extrem unwahrscheinlich ist, daß eine bestimmte Kante durch Hebb'sches Wettbewerbslernen „gefunden" wird.




e) 2500 Signale f) 10000 Signale g) 40000 Signale h) Voronoigebiete

Abbildung 8.4: Hebb'sches Wettbewerbslernen. Simulationssequenz mit einer Wahrscheinlichtsverteilung, die uniform in einem ringförmigen Gebiet ist. a) Initialzustand b-f) Zwischenzustände. g) Endkonfiguration. h) zur Endkonfiguration korrespondierende Voronoitessellation. Da sich die Position der Referenzvektoren überhaupt nicht ändert, ist dieses Verfahren in Isolation sehr abhängig von der Initialisierung.

## 8.3 Neuronales Gas mit Hebb'schem Wettbewerbslernen

Dieses Modell [Martinetz & Schulten, 1991, 1994] ist im wesentlichen (aber nicht ganz) eine Superposition von Neuronalem Gas und Hebb'schen Wettbewerbslernen. Die Adaption der Referenzvektoren wird komplett durch das Verfahren „Neuronales Gas“ gemacht. Parallel dazu wird eine Topologie mittels Hebb'schem Wettbewerbslernen erzeugt. Der Grund, die Kombination dieser beiden Verfahren hier separat zu beschreiben, ist ein Mechanismus zum Löschen von Kanten, der als neues Element hinzukommt und der auch bei dem später beschriebenen Verfahren „Wachsendes Neuronales Gas“ zum Einsatz kommt.

Warum müssen Kanten gelöscht werden? Hebb'sches Wettbewerbslernen erzeugt Kanten, die gleichzeitig auch Kanten sind in der zu den Referenzvektoren korrespondierenden Delaunaytriangulation. Diese Triangulation kodiert die Ähnlichkeitsbeziehungen für bestimmte Zielsetzungen (z.B. für das Suchen nächster Nachbarn eines Punktes) in optimaler Weise. Daher ist es erstrebenswert, in dem erzeugten Graphen auch nur solche Kanten zu haben, die auch in der Delaunaytriangulation enthalten sind. Durch das Adaptieren der Referenzvektoren aber, das durch das Verfahren „Neuronales Gas“ vorgenommen wird, ändert sich die jeweilige zu den Referenzvektoren korrespondierende Voronoitriangulation. Dies bedeutet, daß eingefügte Kanten zu einem späteren Zeitpunkt eventuell nicht mehr zur aktuellen Voronoitriangulation gehören. Solche Kanten zu identifizieren und damit ihre Entfernung möglich zu machen, ist das Ziel des im folgenden vorgestellten *Alterungsprozesses für Kanten.*

Jeder Kante $(c_1, c_2)$ zwischen zwei Einheiten $c_1$ und $c_2$, $(c_1 \in \mathcal{A}, c_2 \in \mathcal{A})$, wird ein Parameter $\mathrm{age}_{(c_1,c_2)}$ zugeordnet, der ihr *Alter* darstellt. Beim Erzeugen der Kante durch Hebb'sches Wettbewerbslernen wird dieser Parameter auf Null gesetzt und später jedesmal dann, wenn die Endpunkte der Kante Gewinner $s_1$ und zweitnächste Einheit $s_2$ für das augenblickliche Eingabesignal sind:

$$\mathrm{age}_{(s_1,s_2)} := 0$$

Ein Alterung findet dadurch statt, daß für jedes Eingabesignal das Alter aller vom Gewinner $s_1$ ausgehenden Kanten um 1 erhöht wird:

$$\mathrm{age}_{(s_1,c)} := \mathrm{age}_{(s_1,c)} + 1 \quad (\forall c \in \mathcal{N}_{s_1})$$

Dabei ist $\mathcal{N}_{s_1}$ die Menge der topologischen Nachbarn der Einheit $s_1$. Diese *lokale* Alterung verhindert, daß durch Signale einem Bereich des Eingaberaums, wo eine hohe Wahrscheinlichkeitsdichte ist, ein globaler Zeitzähler hochgesetzt wird, der dann zu Löschung von Kanten in Gebieten mit geringer Signaldichte führt.

Wenn eine Kante ein Maximalalter $T(t)$ erreicht hat, wird sie wieder entfernt. Das Maximalalter ist zeitabhängig definiert als

$$T(t) := T_i(T_f/T_i)^{t/t_{\max}} \qquad (0 \leq t \leq t_{\max}).$$

Der Anfangswert $T_i$ wird *kleiner* gewählt als der Endwert $T_f$, d.h., die Elimination von Kanten erfolgt am Anfang einer Simulation schneller, als am Ende. Dies ist sinnvoll, da das „Neuronale Gas“ Verfahren am Anfang starke Modifikationen der Referenzvektoren vornimmt, was zu vielen ungültigen Kanten führt, die möglichst schnell beseitigt werden sollen. Am Ende einer Simulation hingegen ist ein großes Maximalalter $T(t)$ sinnvoll, um zufällige Löschungen von Kanten zu verhindern, die durchaus in der Delaunaytriangulation enthalten sind.

Das komplette Verfahren „Neuronales Gas mit Hebb'schem Wettbewerbslernen“ ist im folgenden dargestellt:

1. Initialisiere die Menge $\mathcal{A}$ mit $N$ Einheiten $c_i$

$$\mathcal{A} := \{c_1, c_2, \ldots, c_N\}.$$

   Initialisiere die Referenzvektoren aller Einheiten:

$$\mathbf{w}_c := (\text{zufällig gemäß } p(\boldsymbol{\xi})) \qquad (\forall c \in \mathcal{A})$$

   Initialisiere die Verbindungsmenge $\mathcal{C}$ , $\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$, mit der leeren Menge:

$$\mathcal{C} := \emptyset$$

   Initialisiere den Zeitparameter: $t := 0$

2. Generiere ein Eingabesignal: $\boldsymbol{\xi} := (\text{zufällig gemäß } p(\boldsymbol{\xi}))$

3. Sortiere alle Elemente von $\mathcal{A}$ gemäß ihrer Distanz zu $\boldsymbol{\xi}$, d.h., finde eine Sequenz $s_1, s_2, \ldots, s_N, (s_i \in \mathcal{A})$, derart, daß

$$\|\boldsymbol{\xi} - \mathbf{w}_{s_1}\| \leq \|\boldsymbol{\xi} - \mathbf{w}_{s_2}\| \leq \ldots \leq \|\boldsymbol{\xi} - \mathbf{w}_{s_N}\|$$

4. Adaptiere die Referenzvektoren gemäß

$$\Delta \mathbf{w}_{s_k} := \epsilon(t) \cdot h(t,k) \cdot (\boldsymbol{\xi} - \mathbf{w}_{s_k}) \quad \text{für } k = 1, \ldots, N$$

mit den folgenden Zeitabhängigkeiten:

$$\begin{aligned} \epsilon(t) &:= \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{\max}}, \\ \lambda(t) &:= \lambda_i(\lambda_f/\lambda_i)^{t/t_{\max}}, \\ h(t,k) &:= \exp((1-k)/\lambda(t)) \end{aligned}$$

5. Falls noch keine Verbindung zwischen $s_1$ und $s_2$ existiert, schaffe eine:

$$\mathcal{C} := \mathcal{C} \cup \{(s_1, s_2)\}$$

Setze das Alter der Verbindung zwischen $s_1$ und $s_2$ auf Null („verjünge" die Verbindung):

$$\text{age}_{(s_1,s_2)} := 0.$$

6. Erhöhe das Alter aller von Einheit $s_1$ ausgehenden Kanten:

$$\text{age}_{(s_1,c)} := \text{age}_{(s_1,c)} + 1 \qquad (\forall c \in \mathcal{N}_{s_1})$$

($\mathcal{N}_{s_1}$ ist die Menge aller direkten topologischen Nachbarn von $s_1$).

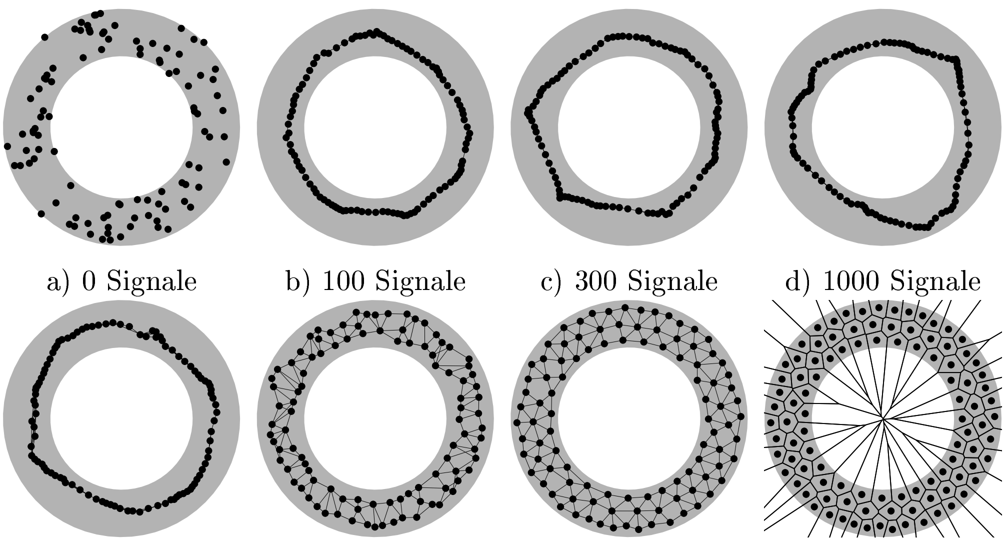
7. Entferne Kanten mit einem Alter größer als ein Maximalwert $T(t)$, wobei

$$T(t) := T_i(T_f/T_i)^{t/t_{\max}}.$$

8. Inkrementiere den Zeitparameter: $t := t + 1$

9. Wenn $t < t_{\max}$, fahre fort mit Schritt 2.

Abbildung 8.5 zeigt eine Simulation dieses Modells für eine ringförmige Wahrscheinlichkeitsverteilung. Die folgenden (von Martinetz et al. [1993] angegebenen) Parameter wurden verwendet: $\lambda_i = 10$, $\lambda_f = 0.01$, $\epsilon_i = 0.5$, $\epsilon_f = 0.005$, $t_{\max} = 40000$, $T_i = 20$, $T_f = 200$. Es sei angemerkt, daß es grundsätzlich immer möglich ist, die beschriebene Kombination von neuronalem Gas und Hebb'schem Wettbewerbslernen mit Alterung von Kanten durch ein zweistufiges Vorgehen zu ersetzen. Dafür wird zuerst neuronales Gas verwendet, um die Referenzvektoren zu positionieren. Anschließend werden die Referenzvektoren „eingefroren" und eine Topologie wird mittels Hebb'schem Wettbewerbslernen erzeugt. Ein Altern der Kanten ist in diesem Falle nicht notwendig, da garantiert ist, daß jede generierte Kante auch Kante in der Delaunaytriangulation bleibt, da sich die Referenzvektoren nicht mehr ändern.



a) 0 Signale b) 100 Signale c) 300 Signale d) 1000 Signale

e) 2500 Signale f) 10000 Signale g) 40000 Signale h) Voronoigebiete

Abbildung 8.5: Neuronales Gas mit Hebb'schem Wettbewerbslernen. Simulationssequenz mit einer Wahrscheinlichtsverteilung, die uniform in einem ringförmigen Gebiet ist. a) Initialzustand b-f) Zwischenzustände. g) Endkonfiguration. h) zur Endkonfiguration korrespondierende Voronoitessellation. Die Referenzvektoren werden durch das neuronale Gas bewegt. Gleichzeitig werden Kanten durch Hebb'sches Wettbewerbslernen erzeugt. Werden diese Kanten eine Weile nicht „aufgefrischt", werden sie wieder gelöscht.

## 8.4 Wachsendes Neuronales Gas

Ein Problem, das viele neuronale Netzwerkmodelle aufweisen, ist, daß *a priori* eine Netzwerkgröße festgelegt werden muß, die sich dann während des Trainings nicht mehr ändert. Je nach Problemstellung und gegebenen Daten kann sich aber später herausstellen, daß zuwenig oder zuviele Einheiten verwendet wurden. Um eine günstige Netzwerkgröße zu bestimmen, sind daher oft zahlreiche Simulationsläufe mit unterschiedlichen Netzwerkgrößen erforderlich.

Das im folgenden beschriebene Modell „Wachsendes Neuronales Gas“ [Fritzke, 1995a] vermeidet die Notwendigkeit der *a priori* Festlegung einer Netzwerkgröße durch einen datengesteuerten Wachstumsprozeß. Ausgehend von einem sehr kleinen Netzwerk werden nach und nach neue Einheiten hinzugefügt. Der Wachstumsprozeß wird beendet, wenn ein Haltekriterium – wie etwa das Unterschreiten eines bestimmten mittleren Quantisierungsfehlers – erfüllt ist.

Ein Kennzeichen des Wachstumsprozesses ist, daß neue Einheiten nicht etwa zufällig eingefügt werden (also mit zufällig gewählten Referenzvektoren), sondern daß während vorangegangener Adaptionsschritte *akkumulierte statistische Information*[1] $E$ benutzt wird, um eine geeignete Position innerhalb des Netzwerks und im Eingaberaum zu bestimmen. Diese Vorgehensweise basiert auf drei Grundannahmen:

- Die akkumulierte statistische Information $E$ ist eine Fehlergröße, die reduziert werden soll, d.h.,
$$\sum_{c \in \mathcal{A}} E_c \quad min!$$
- Das Einfügen einer neuen Einheit $r$ bewirkt insgesamt eine Absenkung des Fehlers, d.h.,
$$\sum_{c \in \mathcal{A}} E_c > \sum_{c \in \mathcal{A} \cup \{r\}} E_c$$
- Die stärkste Absenkung des Fehlers ist zu erwarten, wenn die neue Einheit $r$ in der Nähe der Einheit $q$ eingefügt wird, die bisher den größten Fehler akkumuliert hat, für die also gilt:
$$q := \arg\max_{c \in \mathcal{A}} \mathrm{E}_c.$$

Eine Zielsetzung, die all diese Voraussetzungen erfüllt, ist die Reduktion des Quantisierungsfehlers mit einem vektorbasierten Netzwerk. Wenn in diesem Fall jede Einheit den Quantisierungsfehler der Signale akkumuliert, für die sie Gewinner ist, so ist die Summe aller Werte $E_c$ genau der Quantisierungsfehler für alle aufgetretenen Eingabesignale. Die Einführung einer neuen Einheit kann den zu erwartenden Quantisierungsfehler nur herabsetzen. Ohne weiteres Vorwissen über die Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ der Eingabesignale ist die Einfügung einer neuen Einheit am erfolgversprechendsten in der Nähe der Einheit $q$, die bisher den größten akkumulierten Fehler aufweist. Es ist aber auch so, daß eine

[1] Dieses Prinzip wurde zuerst vorgeschlagen im Modell „Wachsende Zellstrukturen“ [Fritzke, 1992, 1994a], das auch in Abschnitt 9.2 beschrieben ist.

Einfügung an identischer Position im Eingaberaum, d.h., mit $w_r = w_q$, keinerlei Verbesserung bringen würde, da keines der Eingabesignale, die bisher auf die Einheit $q$ abgebildet wurden (also im Voronoigebiet von $q$ lagen), eine nähere Einheit finden kann. Es ist daher naheliegend, die neue Einheit zwischen der Einheit $q$ mit dem größten akkumulierten Fehler und einer benachbarten Einheit zu positionieren. Im Verfahren „Wachsendes Neuronales Gas“ wird unter den Nachbarn von $q$ stets derjenige Nachbar $f$ mit dem größten akkumulierten Fehler ausgewählt, d.h.,

$$f := \arg\max_{c \in \mathcal{N}_q} \mathrm{E}_c$$

und die neue Einheit wird auf halbem Weg zwischen $q$ und $f$ im Eingaberaum plaziert:

$$w_r := (w_q + w_r)/2.$$

Die Fehlerwerte von $q$ und $f$ werden reduziert um einen Bruchteil $\alpha$. Das Ziel dabei ist, diese Werte den (geringeren) Fehlern anzupassen, die nach der Einfügung der Einheit $r$ zu erwarten sind[2]. Aus demselben Grund wird der neuen Einheit $r$ auch ein initialer Fehlerwert zugeordnet, der interpoliert wird von den (bereits reduzierten) Fehlerwerten von $q$ und $f$. Die Verbindungstopologie wird durch Hebb'sches Wettbewerbslernen mit Kantenalterung erzeugt. Im Gegensatz zu dem Verfahren „Neuronales Gas mit Hebb'schem Wettbewerbslernen“ ist das Maximalalter, das eine Kante erreichen darf, zeitlich konstant.

Die Adaption der Referenzvektoren erfolgt durch eine Adaption des Gewinners $s_1$ sowie der direkten Nachbarn von $s_1$ innerhalb der Netzwerktopologie. Die Adaptionsstärke ist zeitlich konstant genau wie die Größe der Nachbarschaft, die sich stets auf die direkten Nachbarn beschränkt.

Der vollständige Algorithmus „Wachsendes Neuronales Gas“ ist der folgende:

1. Initialisiere die Menge $\mathcal{A}$ mit zwei Einheiten $c_1$ und $c_2$

   $$\mathcal{A} := \{c_1, c_2\}.$$

   Initialisiere die Referenzvektoren beider Einheiten:

   $$\mathbf{w}_c := (\text{zufällig gemäß } p(\boldsymbol{\xi})) \qquad (\forall c \in \mathcal{A}).$$

   Initialisiere die Verbindungsmenge $\mathcal{C}$ , $\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$, mit der leeren Menge:

   $$\mathcal{C} := \emptyset$$

2. Generiere ein Eingabesignal: $\boldsymbol{\xi} := (\text{zufällig gemäß } p(\boldsymbol{\xi}))$

[2] Alternativ könnte man auch nach jeder Einfügung einer neuen Einheit alle Fehlerwerte löschen und völlig neue Werte in den darauffolgenden Adaptionsschritten akkumulieren. Diese Strategie, die u.a. bei dem Modell „Wachsendes Gitter“ in Abschnitt 9.3 verfolgt wird, vermeidet die (heuristische) Umverteilung der Fehlerwerte, erfordert andererseits eine Anzahl von Adaptionsschritten *pro Einfügeschritt*, die proportional zur jeweiligen Netzwerkgröße ist. Zahlreiche Simulationen haben jedoch ergeben, daß diese Vorgehensweise keinen meßbaren Vorteil ergibt, so daß in der Regel mit Umverteilung der Fehlerwerte gearbeitet wird, was eine deutlich schnellere Konstruktion des Netzwerks erlaubt.

3. Bestimme die Gewinnereinheit $s_1$ und die zu $\boldsymbol{\xi}$ zweitnächste Einheit $s_2$.

4. Falls noch keine Verbindung zwischen $s_1$ und $s_2$ existiert, schaffe eine:

$$\mathcal{C} := \mathcal{C} \cup \{(s_1, s_2)\}$$

   Setze das Alter der Verbindung zwischen $s_1$ und $s_2$ auf Null („verjünge“ die Verbindung):

$$\text{age}_{(s_1,s_2)} := 0.$$

5. Addiere den quadratischen Abstand zwischen Eingabesignal und Gewinner zu einer lokalen Fehlervariable:

$$\Delta \mathrm{E}_{s_1} := \|\boldsymbol{\xi} - \mathbf{w}_{s_1}\|^2$$

6. Adaptiere die Referenzvektoren des Gewinners und seiner direkten topologischen Nachbarn:

$$\begin{array}{rcll} \Delta \mathbf{w}_{s_1} & := & \epsilon_b(\boldsymbol{\xi} - \mathbf{w}_{s_1}) & \\ \Delta \mathbf{w}_i & := & \epsilon_n(\boldsymbol{\xi} - \mathbf{w}_i) & \qquad (\forall i \in \mathcal{N}_{s_1}) \end{array}$$

7. Erhöhe das Alter aller von Einheit $s_1$ ausgehenden Kanten:

$$\text{age}_{(s_1,c)} := \text{age}_{(s_1,c)} + 1 \qquad (\forall c \in \mathcal{N}_{s_1})$$

   ($\mathcal{N}_{s_1}$ ist die Menge aller direkten topologischen Nachbarn von $s_1$).

8. Entferne Kanten mit einem Alter größer als $a_{max}$. Wenn dies zu Einheiten ohne Kanten führt entferne auch diese.

9. Wenn die Anzahl der bisher erzeugten Eingabesignale ein ganzzahliges Vielfaches eines Parameters $\lambda$ ist, erzeuge eine neue Einheit wie folgt:

   - Bestimme die Einheit $q$ mit maximalem akkumuliertem Fehler:

$$q := \arg\max_{c \in \mathcal{A}} \mathrm{E}_c$$

   - Bestimme unter den Nachbarn von $q$ die Einheit $f$ mit maximalem akkumuliertem Fehler:

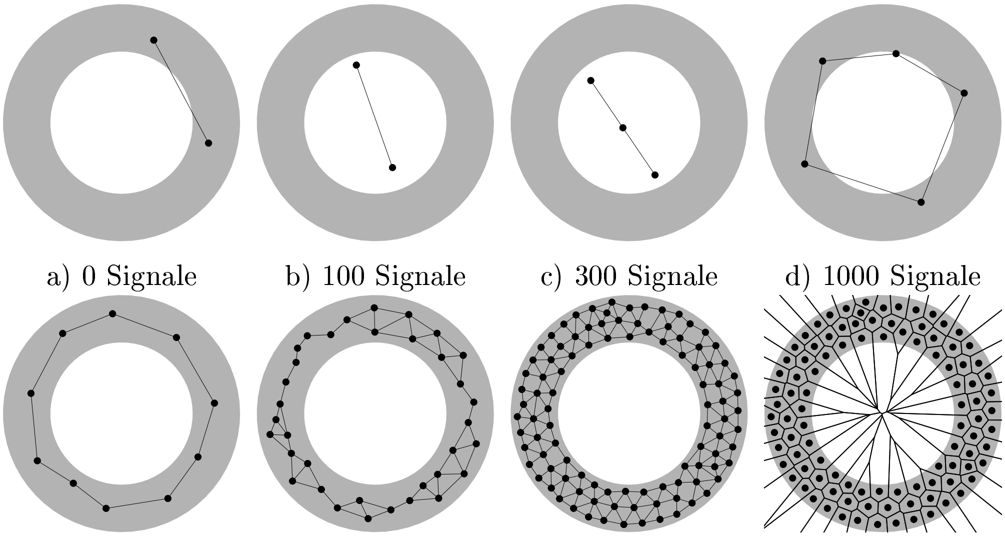
$$f := \arg\max_{c \in \mathcal{N}_q} \mathrm{E}_c$$

   - Füge eine neue Einheit $r$ zum Netzwerk dazu und interpoliere ihren Referenzvektor von $q$ und $f$:

$$\mathcal{A} := \mathcal{A} \cup \{r\}, \qquad \mathbf{w}_r := (\mathbf{w}_q + \mathbf{w}_f)/2.$$

   - Erzeuge je eine Kante zwischen der Einheit $r$ und den Einheiten $q$ und $f$. Lösche die Kante zwischen $q$ und $f$.

$$\mathcal{C} := \mathcal{C} \cup \{(r, q),\, (r, f)\}, \qquad \mathcal{C} := \mathcal{C} \setminus \{(q, f)\}$$



a) 0 Signale b) 100 Signale c) 300 Signale d) 1000 Signale

e) 2500 Signale f) 10000 Signale g) 40000 Signale h) Voronoigebiete

Abbildung 8.6: Wachsendes Neuronales Gas. Simulationssequenz mit einer Wahrscheinlichtsverteilung, die uniform in einem ringförmigen Gebiet ist. a) Initialzustand b-f) Zwischenzustände. g) Endkonfiguration. h) zur Endkonfiguration korrespondierende Voronoitessellation. Die Maximale Netzwerkgröße ist auf 100 gesetzt worden.

- Reduziere die Fehlervariablen von $q$ und $f$ um einen Bruchteil $\alpha$:

$$\Delta \mathrm{E}_q := -\alpha \mathrm{E}_q, \qquad \Delta \mathrm{E}_f := -\alpha \mathrm{E}_f$$

- Interpoliere die Fehlervariable von $r$ aus den Fehlervariablen von $q$ und $f$.

$$\mathrm{E}_r := (\mathrm{E}_q + \mathrm{E}_f)/2$$

10. Reduziere die Fehlervariablen aller Einheiten um einen Bruchteil $\beta$:

$$\Delta \mathrm{E}_c := -\beta \, \mathrm{E}_c \qquad (\forall c \in \mathcal{A}).$$

11. Wenn das Haltekriterium (z.B. Netzwerkgröße oder ein Qualitätskriterium) noch nicht erfüllt ist, fahre fort mit Schritt 2.

Abbildung 8.6 zeigt eine Simulation dieses Modells für eine ringförmige Wahrscheinlichkeitsverteilung. Die benutzten Parameter waren: $\lambda = 300$, $\epsilon_b = 0.05$, $\epsilon_n = 0.0006$, $\alpha = 0.5$, $\beta = 0.0005$, $a_{max} = 88$.

Kleine konstante Adaptionsraten werden bei diesem Verfahren dadurch möglich, daß neue Einheiten von existierenden interpoliert werden. Daher sind sie von Anfang an auf guten Positionen und müssen nicht, wie bei anderen Verfahren, längere Strecken durch Adaption zurücklegen.

Wenn das Ziel des Selbstorganisationsprozesses nicht Fehlerminimierung, sondern Entropiemaximierung ist, kann man Schritt 5 des oben beschriebenen Verfahrens ersetzen durch

5. Addiere 1 (eins) zu einer lokalen Fehlervariable:

$$\Delta \mathrm{E}_{s_1} := 1$$

Dies bewirkt, daß neue Einheiten vermehrt in der Nähe derjenigen Einheiten positioniert werden, die überproportional oft Gewinner gewesen sind und führt dadurch letztlich zu einer höheren Entropie als die fehlerbasierte Einfügung.

Eine Variante von „Wachsendes Neuronales Gas“, die besonders gut für nichtstationäre Verteilungn geeignet ist, ist in [Fritzke, 1997a] beschrieben. In dieser Variante wird das von LBG-U (siehe Kapitel 6.2) bekannte *Nützlichkeitsmaß* verwendet, um Einheiten auch wieder zu löschen, wenn sie an anderer Stelle mehr zum Reduzieren des Fehlers beitragen können.

# Kapitel 9

# Weiches Wettbewerbslernen mit fester Netzwerkdimension

Die in diesem Kapitel dargestellten Verfahren verbindet, daß sie alle eine Netzwerktopologie mit einer bestimmten und meist niedrigen Dimensionalität $k$ verwenden. So ist etwa im zweidimensionalen Fall die Topologie der selbstorganisierenden Merkmalskarten (Abschnitt 9.1) ein Rechteckgitter und die der wachsenden Zellstrukturen (Abschnitt 9.2) eine zweidimensionale Struktur aus Dreiecken. Viele der bisher dargestellten Verfahren haben gar keine Verbindungen zwischen den Netzwerkeinheiten und damit auch keine Topologie. Dies trifft z.B. für die Verfahren des harten Wettbewerbslernens in Kapitel 6 oder das neuronale Gas in Abschnitt 8.1) zu. Andere bereits behandelte Verfahren haben zwar Verbindungen, jedoch sind diese nicht auf eine bestimmte Dimensionalität beschränkt, sondern richten sich nach der Dimensionalität der Eingabedaten. Beispiele sind das Hebb'sche Wettbewerbslernen in Abschnitt 8.2, neuronales Gas mit Hebb'schen Wettbewerbslernen in Abschnitt 8.3 oder wachsendes neuronales Gas in Abschnitt 8.4.

Der wesentliche Vorteil, den eine $k$-dimensionale Topologie mit sich bringt ist, daß man das Netzwerk unabhängig von der Dimensionalität $n$ der Eingabedaten visualisieren kann, wenn nur $k$ hinreichend klein ist. Dies wiederum kann zur Visualisierung auch hochdimensionaler Eingabedaten verwendet werden, indem Repräsentanten der Datenvektoren an denjenigen Netzwerknoten dargestellt werden, die ihnen im Eingaberaum am nächsten sind.

Ein potentieller Nachteil einer Topologie fester Dimension ist, daß der Adaptionsprozeß dadurch zusätzlichen Randbedingungen und Einflüssen unterworfen wird, die für bestimmte Zielsetzungen – wie etwa Fehlerminimierung – hinderlich sein können.

## 9.1 Selbstorganisierende Merkmalskarten

Dieses Model stammt von Kohonen [1982] und basiert zum Teil auf früheren Arbeiten von Willshaw & von der Malsburg [1976]. Das Modell ähnelt dem (viel später entwickelten) neuronalem Gas (Abschnitt 8.1) insofern, als ebenfalls eine abnehmende Nachbarschaftsreichweite und eine abnehmende Lernrate verwendet werden. Ein wichtiger Unterschied allerdings ist die Netzwerktopologie, die ein zweidimensionales Gitter $(a_{ij})$ ist und die sich während des durchgeführten Adaptionssprozesses nicht ändert.

Die Distanz auf diesem Gitter wird benutzt, um zu bestimmen, wie stark die Einheit $r = a_{km}$ adaptiert wird, wenn die Einheit $s = a_{ij}$ der Gewinner ist. Als Distanzmaß verwenden wir im folgenden die (auch als Manhattan-Distanz bekannte) $\mathrm{L}_1$-Norm:

$$\mathrm{d}_1(r, s) = |i - k| + |j - m| \qquad \text{für } r = a_{km} \text{ und } s = a_{ij}.$$

Für die Nachbarschaftsreichweite existieren mehrere Möglichkeiten. Ritter et al. [1991] schlagen die folgende Gaußglocke vor, um die Adaption einer Einheit $r$ zu festzulegen, wenn $s$ der Gewinner für das augenblickliche Eingabesignal ist:

$$h(t, r, s) = \exp\left(-\frac{\mathrm{d}_1(r, s)^2}{2\sigma^2(t)}\right). \tag{9.1}$$

Dabei wird der Breitenparameter $\sigma$ der Gausglocke zeitlich variiert gemäß

$$\sigma(t) = \sigma_i(\sigma_f/\sigma_i)^{t/t_{\max}} \tag{9.2}$$

mit geeignetem Anfangswert $\sigma_i$ und Endwert $\sigma_f$.

Die Lernrate $\epsilon(t)$ wird wie beim neuronalen Gas zeitlich verändert gemäß

$$\epsilon(t) = \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{\max}} \tag{9.3}$$

für Anfangswert $\epsilon_i$ und Endwert $\epsilon_f$.

Die Gesamtadaptionsstärke einer Einheit $r$ ergibt sich durch Multiplikation von Lernrate und Nachbarschaftsreichweite, so daß der Referenzvektor von $r$ verändert wird gemäß

$$\Delta \mathbf{w}_r = \epsilon(t)\, h(t, r, s)\, (\boldsymbol{\xi} - \mathbf{w}_r). \tag{9.4}$$

In Abbildung 9.1 ist die zeitliche Entwicklung der Gesamtadaptionsstärke für typische Parameterwerte dargestellt.

Der vollständige Algorithmus „Selbstorganisierende Merkmalskarten“ ist der folgende:

1. Initialisiere die Menge $\mathcal{A}$ mit $N = N_1 \cdot N_2$ Einheiten $c_i$

   $$\mathcal{A} := \{c_1, c_2, \ldots, c_N\}.$$

   Initialisiere die Referenzvektoren aller Einheiten:

   $$\mathbf{w}_c := (\text{zufällig gemäß } p(\boldsymbol{\xi})) \qquad (\forall c \in \mathcal{A})$$

   Initialisiere die Verbindungsmenge $\mathcal{C}$ derart, daß sie ein rechteckiges $N_1 \times N_2$-Gitter bildet.

   Initialisiere den Zeitparameter: $\quad t := 0$

2. Generiere ein Eingabesignal: $\quad \boldsymbol{\xi} := (\text{zufällig gemäß } p(\boldsymbol{\xi}))$

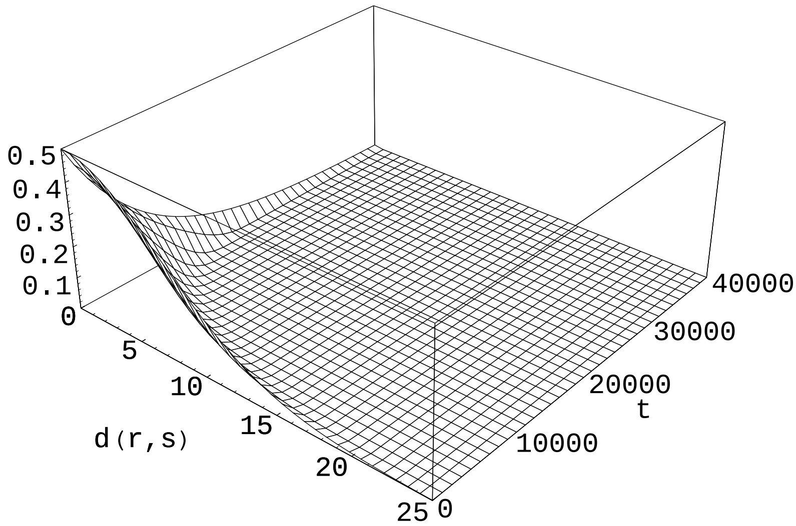



Abbildung 9.1: Zeitliche Entwicklung der Gesamtadaptionsstärke $\epsilon(t)h(t,r,s)$ in Abhängigkeit von dem Abstand $d(r,s)$ zwischen betrachteter Einheit $r$ und Gewinner $s$. Parameter: $t_{\max} = 40000$, $\epsilon_i = 0.5$, $\epsilon_f = 0.005$, $\sigma_i = 10$, $\sigma_f = 0.01$

3. Bestimme den Gewinner $s = s(\boldsymbol{\xi})$ für das aktuelle Eingabesignal.

4. Adaptiere jede Einheit gemäß

$$\Delta \mathbf{w}_r := \epsilon(t)\, h(t,r,s)\, (\boldsymbol{\xi} - \mathbf{w}_r),$$

   wobei $\epsilon(t)$ und $h(t,r,s)$ definiert sind wie in Gleichung (9.3) und (9.1) angegeben.

5. Inkrementiere den Zeitparameter: $t := t + 1$

6. Wenn $t < t_{\max}$, fahre fort mit Schritt 2.

Abbildung 9.2 zeigt eine Simulation dieses Modells für eine ringförmige Wahrscheinlichkeitsverteilung. Die benutzten Parameter waren: $\sigma_i = 3.0$, $\sigma_f = 0.1$, $\epsilon_i = 0.5$, $\epsilon_f = 0.005$, $t_{\max} = 10000$ und $N_1 = N_2 = 10$.

Das beschriebene Verfahren läßt sich analog anwenden für Rechteckgitter von allgemeiner Dimensionalität $k$. Von praktischer Bedeutung sind jedoch meist nur die Fälle $k \in \{1,2,3\}$, da nur diese Werte eine einfache Visualisierung des Netzwerks gestatten. Ein Beispiel, wo dreidimensionale Gitter eingesetzt wurden ist ein Robotikproblem, wo ein redundanter Roboter die inverse Kinematik in einem dreidimensionalen Arbeitsbereich lernte [Martinetz et al., 1990]. In Abbildung 9.3 sind Netzwerke verschiedener Dimensionalität gezeigt.

## 9.2 Wachsende Zellstrukturen

Dieses Netzwerkmodell [Fritzke, 1992, 1994a] hat wie die selbstorganisierenden Merkmalskarten im vorigen Abschnitt eine Topologie fester Dimensionalität $k$.

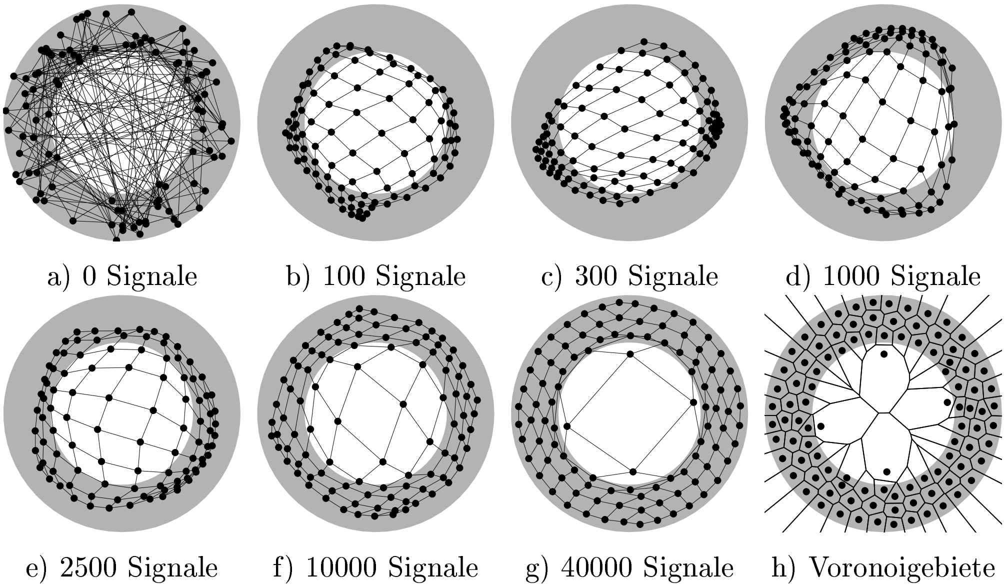


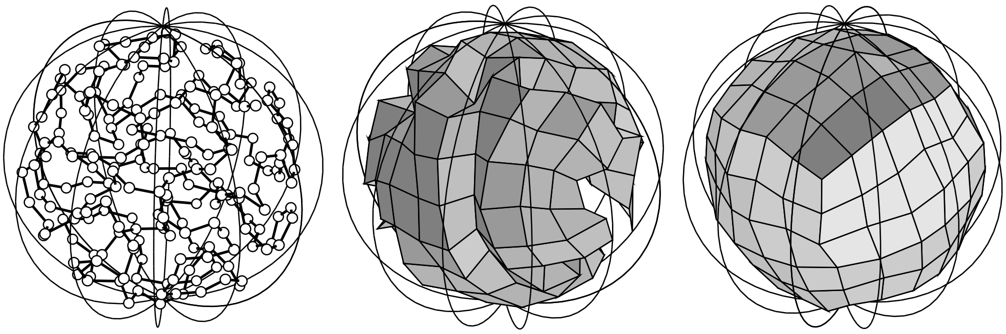

Abbildung 9.2: Selbstorganisierende Merkmalskarten. Simulationssequenz mit einer Wahrscheinlichtsverteilung, die uniform in einem ringförmigen Gebiet ist. a) Initialzustand b-f) Zwischenzustände. g) Endkonfiguration. h) zur Endkonfiguration korrespondierende Voronoitessellation. Wie beim neuronalen Gas führt die anfangs starke Nachbarschaftsadaption und die hohe Lernrate zu sehr heftigen und globalen Adaptionen, die im Laufe der Zeit geringer und lokaler werden.

Darüberhinaus sind beide Modelle jedoch sehr verschieden voneinander. So ist bei den wachsenden Zellstrukturen die Netzwerkstruktur nicht von vornherein festgelegt, sondern entsteht durch einen datengesteuerten Wachstumsprozeß. Der topologische Grundbaustein ist dabei ein $k$-dimensionaler Simplex, also ein Segment für $k = 1$, ein Dreieck für $k = 2$, ein Tetraeder für $k = 3$ u.s.w. Abgesehen von dieser speziellen Einschränkung der Topologie ist das Verfahren nahezu identisch zum (später entwickelten) wachsenden neuronalen Gas (Abschnitt 8.4). An dieser Stelle soll daher der Wachstumsprozeß auch nicht noch einmal motiviert werden, sondern es sei auf die entsprechende Diskussion auf Seite 74 verwiesen.

Genau wie beim wachsenden neuronalen Gas findet auch bei den wachsenden Zellstrukturen eine Adaption des Gewinners und seiner direkten topologischen Nachbarn statt und ebenfalls wird der quadratische Fehler lokal akkumuliert und für die Bestimmung der Positionen, wo neue Einheiten eingefügt werden sollen, verwendet.

Beim Einfügen neuer Einheiten gibt es gewisse Unterschiede zum wachsenden neuronalen Gas. Bei letzterem wird (wie beschrieben) jeweils eine neue Einheit $r$ zwischen der Einheit $q$ mit maximalem akkumuliertem Fehler eingefügt und zwischen demjenigen Nachbarn $f$ von $q$, der unter den Nachbarn den größten Fehler akkumuliert hat.

Bei den wachsenden Zellstrukturen wird $r$ stattdessen zwischen $q$ und dem Nachbarn $f$ von $q$ eingefügt, der im Eingaberaum am weitesten von $q$ entfernt



a) 216 Einheiten b) $18 \times 12$ Einheiten c) $6 \times 6 \times 6$ Einheiten

Abbildung 9.3: Selbstorganisierende Karten verschiedener Dimensionalität passen sich einer dreidimensionalen Wahrscheinlichkeitsverteilung an.

ist, für den also gilt:

$$f := \arg\max_{c \in \mathcal{N}_q} \|\mathbf{w}_q - \mathbf{w}_c\|$$

Der Grund für diese Modifikation ist, daß auch bei den wachsenden Zellstrukturen möglichst gleichseitige Dreiecke, bzw. Simplices entstehen sollen, damit die Annäherung an die Delaunaytriangulation möglichst gut ist. Im Gegensatz zum wachsenden neuronalen Gas gibt es aber hier keinen dem Hebb'schen Wettbewerbslernen entsprechenden Mechanismus, der dafür sorgt, daß überflüssige Kanten eliminiert und notwendige eingefügt werden. Daher ist es notwendig, von vornherein möglichst gleichseitige Simplices zu erzeugen, wozu die verwendete Strategie tendenziell führt.

Nachdem eine neue Einheit $r$ zwischen den Einheiten $q$ und $f$ eingefügt wurde, wird die Verbindungstopologie lokal ergänzt, so daß sich wieder eine Struktur aus $k$-dimensionalen Simplices ergibt. Wie man sich leicht überlegt, ist es dafür ausreichend, $r$ mit allen gemeinsamen Nachbarn von $q$ und $f$ zu verbinden. Alle Nachbarn der neuen Einheit $r$ werden dann in die Umverteilung der Fehlervariablen mit einbezogen. Beim wachsenden neuronalen Gas waren dies dagegen nur $q$ und $f$, da weitere Nachbarn nicht existierten, sonder erst durch nachfolgende Kanteneinfügungen gemäß dem Hebb'schen Wettbewerbslernen entstanden.

Die für das wachsende neuronale Gas auf Seite 75 gemachten Bemerkungen bezüglich der Möglichkeit, alle Fehlervariablen nach jeder Einfügung zu löschen und dafür mehr Adaptionsschritte pro Einfügung zu machen gelten hier völlig analog.

Das Verfahren „Wachsende Zellstrukturen“ ist das folgende:

1. Wähle eine Netzwerkdimensionalitat $k$.

2. Initialisiere die Menge $\mathcal{A}$ mit $N = k + 1$ Einheiten $c_i$

$$\mathcal{A} := \{c_1, c_2, \ldots, c_N\}.$$

Initialisiere die Referenzvektoren aller Einheiten:

$$\mathbf{w}_c := (\text{zufällig gemäß } p(\boldsymbol{\xi})) \qquad (\forall c \in \mathcal{A})$$

3. Initialisiere die Verbindungsmenge $\mathcal{C}$ derart, daß jede Einheit mit jeder anderen verbunden ist (dies ist die Topologie eines $k$-dimensionalen Simplex).

4. Generiere ein Eingabesignal: $\boldsymbol{\xi} :=$ (zufällig gemäß $p(\boldsymbol{\xi})$)

5. Bestimme den Gewinner $s = s(\boldsymbol{\xi})$ für das aktuelle Eingabesignal.

6. Addiere den quadratischen Abstand zwischen Eingabesignal und Gewinner zu einer lokalen Fehlervariable:

$$\Delta \mathrm{E}_s := \|\boldsymbol{\xi} - \mathbf{w}_s\|^2$$

7. Adaptiere die Referenzvektoren des Gewinners und seiner direkten topologischen Nachbarn:

$$\begin{aligned} \Delta \mathbf{w}_s &:= \epsilon_b(\boldsymbol{\xi} - \mathbf{w}_s) \\ \Delta \mathbf{w}_i &:= \epsilon_n(\boldsymbol{\xi} - \mathbf{w}_i) \qquad (\forall i \in \mathcal{N}_s) \end{aligned}$$

8. Wenn die Anzahl der bisher erzeugten Eingabesignale ein ganzzahliges Vielfaches eines Parameters $\lambda$ ist, erzeuge eine neue Einheit wie folgt:

   - Bestimme die Einheit $q$ mit maximalem akkumuliertem Fehler:

   $$q := \arg\max_{c \in \mathcal{A}} \mathrm{E}_c$$

   - Bestimme unter den Nachbarn von $q$ die Einheit $f$ mit größtem Abstand zu $q$:

   $$f := \arg\max_{c \in \mathcal{N}_q} \|\mathbf{w}_q - \mathbf{w}_c\|$$

   - Füge eine neue Einheit $r$ zum Netzwerk dazu und interpoliere ihren Referenzvektor von $q$ und $f$:

   $$\mathcal{A} := \mathcal{A} \cup \{r\}, \qquad \mathbf{w}_r := (\mathbf{w}_q + \mathbf{w}_f)/2.$$

   - Verbinde die neue Einheit $r$ mit allen gemeinsamen Nachbarn von $q$ und $f$:

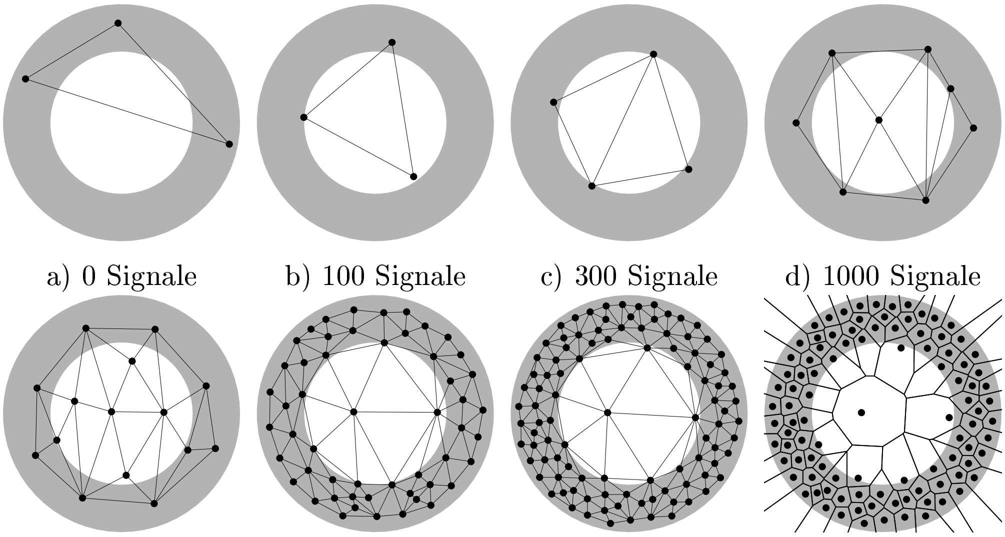
   $$\mathcal{C} := \mathcal{C} \cup \{(r,c)\} \quad \forall c \in \mathcal{N}_q \cap \mathcal{N}_f$$

   - Erzeuge je eine Kante zwischen der Einheit $r$ und den Einheiten $q$ und $f$. Lösche die ursprüngliche Kante zwischen $q$ und $f$.

   $$\mathcal{C} := \mathcal{C} \cup \{(r,q),\, (r,f)\}, \qquad \mathcal{C} := \mathcal{C} \setminus \{(q,f)\}$$

   - Reduziere die Fehlervariablen aller Nachbarn von $r$ um einen Bruchteil, der von der Anzahl der Nachbarn von $r$ abhängt:

   $$\Delta \mathrm{E}_i := -\frac{\alpha}{|\mathcal{N}_r|}\mathrm{E}_i \qquad (\forall i \in \mathcal{N}_r).$$



a) 0 Signale b) 100 Signale c) 300 Signale d) 1000 Signale

e) 2500 Signale f) 10000 Signale g) 40000 Signale h) Voronoigebiete

Abbildung 9.4: Wachsende Zellstrukturen. Simulationssequenz mit einer Wahrscheinlichtsverteilung, die uniform in einem ringförmigen Gebiet ist. a) Initialzustand b-f) Zwischenzustände. g) Endkonfiguration. h) zur Endkonfiguration korrespondierende Voronoitessellation. Per Konstruktion besteht die Netzwerkstruktur immer aus Hypertetrahedern (Dreiecken in diesem Fall).

- Initialisiere den Fehlerwert der Einheit $r$ mit dem Mittelwert ihrer Nachbarn:

$$\mathrm{E}_r := \frac{1}{|\mathcal{N}_r|} \sum_{i \in N_r} \mathrm{E}_i.$$

9. Reduziere die Fehlervariablen aller Einheiten um einen Bruchteil $\beta$:

$$\Delta \mathrm{E}_c := -\beta \, \mathrm{E}_c \qquad (\forall c \in \mathcal{A}).$$

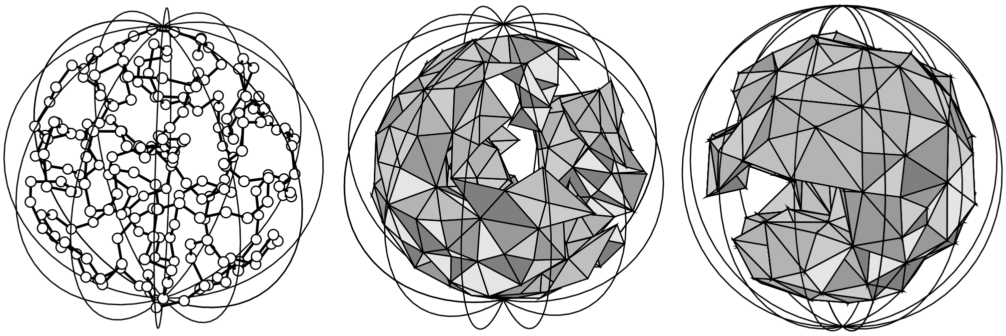
10. Wenn das Haltekriterium (z.B. Netzwerkgröße oder ein Performanzmaß) noch nicht erfüllt ist, fahre fort mit Schritt 4.

Abbildung 9.4 zeigt eine Simulation dieses Modells für eine ringförmige Wahrscheinlichkeitsverteilung. Die benutzten Parameter waren: $\alpha = 1.0$, $\varepsilon_b = 0.06$, $\varepsilon_n = 0.002$, $\beta = 0.0005$, $\lambda = 200$.

In Abbildung 9.5 sind Netzwerke verschiedener Dimensionalität gezeigt. Eindimensionale Wachsende Zellstrukturen sind von Fritzke & Wilke [1991] für die näherungsweise Lösung von kombinatorischen Optimierungsproblemen eingesetzt wurden.

## 9.3 Wachsende Gitter

Dieses Modell [Fritzke, 1995b] ist entwickelt worden, um die einfache Möglichkeit der Datenvisualisierung, die rechteckige Gitterstrukturen bieten, zu verknüpfen mit den Vorteilen eines inkrementellen Netzwerks. Diese Vorteile sind



a) 216 Einheiten, 1D b) 216 Einheiten, 2D c) 216 Einheiten, 3D

Abbildung 9.5: Wachsende Zellstrukturen verschiedener Dimensionalität passen sich einer dreidimensionalen Wahrscheinlichkeitsverteilung an.

insbesondere konstante Parameter und die Möglichkeit, ein geeignetes Netzwerk durch Wachstum zu erhalten. Die Gitterstruktur wurde abgeleitet von den selbstorganisierenden Merkmalskarten (Abschnitt 9.1) und die prinzipielle Art des Wachstums vom wachsenden neuronalen Gas (Abschnitt 8.4) bzw. von den wachsenden Zellstrukturen (Abschnitt 9.2).

Um stets eine rechteckige Gitterstruktur zu behalten, wird das Netzwerk initialisiert mit einem $k$-dimensionalen Hyperkubus und es werden immer komplette $k-1$-dimensionale Hyperebenen eingefügt. Im Fall $k=2$ ist die Anfangsstruktur ein $2 \times 2$-Gitter und es werden jeweils komplette Zeilen oder Spalten eingeführt.

Wie bei den anderen inkrementellen Modellen auch, wird eine Einfügung stets bei der Netzwerkeinheit gemacht, die bisher den größten Fehler akkumuliert hat. Anders als beim wachsenden neuronalen Gas oder den wachsenden Zellstrukturen wird die Fehlerinformation von allen Einheiten bei jeder Einfügung komplett gelöscht. Dies hat den Vorteil, daß sich eine heuristische Umverteilung des Fehlers erübrigt. Andererseits muß vor der nächsten Einfugung erst wieder genug statistische Information angesammelt werden, um über die nächste Einfügung entscheiden zu können. Daher wird eine Anzahl von Adaptionsschritten vor jeder Einfügung durchgeführt, die proportional zur augenblicklichen Netzwerkgröße ist. Bei den anderen inkrementellen Verfahren wird nur eine *konstante* Anzahl von Adaptionsschritten pro Einfügung durchgeführt.

Eine weitere Besonderheit dieses Netzwerkmodells (die allerdings auch für die anderen inkrementellen Modelle verwandt werden kann) ist die Existenz von zwei verschiedenen Phasen. In der *Wachstumsphase* sind alle Parameter konstant. In der *Feinabstimmungsphase* dagegen, wird die Lernrate kontinuierlich abgesenkt. Diese Phase beginnt, wenn das Netzwerk seine endgültige Größe erreicht hat, z.B. weil ein Fehlermaß weit genug gesunken ist. Sie dient dazu, bessere Positionen für die Referenzvektoren zu bestimmen, als dies mit einer konstanten Lernrate möglich ist. Allerdings wird die Nachbarschaftsadaption auch in der Feinabstimmungsphase nicht unterbunden, d.h., der resultierende Prozeß ist kein hartes Wettbewerbslernen.

Das komplette Verfahren „Wachsende Gitter“ ist im folgenden beschrieben:
*Wachstumsphase*

1. Setze die initiale Höhe und Breite des Netzwerks:
$$N_1 = 2, \quad N_2 = 2.$$
2. Initialisiere die Menge $\mathcal{A}$ mit $N = N_1 \cdot N_2$ Einheiten $c_i$
$$\mathcal{A} := \{c_1, c_2, \ldots, c_N\}.$$
Initialisiere die Referenzvektoren aller Einheiten:
$$\mathbf{w}_c := (\text{zufällig gemäß } p(\boldsymbol{\xi})) \qquad (\forall c \in \mathcal{A})$$
3. Initialisiere die Verbindungsmenge $\mathcal{C}$ derart, daß sie ein rechteckiges $N_1 \times N_2$-Gitter bildet.
Initialisiere den Zeitparameter: $t := 0$
4. Generiere ein Eingabesignal: $\boldsymbol{\xi} := (\text{zufällig gemäß } p(\boldsymbol{\xi}))$
5. Bestimme den Gewinner $s = s(\boldsymbol{\xi})$ für das aktuelle Eingabesignal.
6. Inkrementiere die Zählvariable des Gewinners: $\tau_s := \tau_s + 1$.
7. Inkrementiere den Zeitparameter: $t := t + 1$
8. Adaptiere jede Einheit gemäß $\Delta \mathbf{w}_r = \epsilon_0\, h(r, s)(\boldsymbol{\xi} - \mathbf{w}_r)$,
9. Wenn die Anzahl $t$ der Eingabesignale für die augenblickliche Netzwerkgröße ein ganzzahliges Vielfaches $\lambda_g$ dieser Netzwerkgröße erreicht, d.h., wenn
$$\lambda_g \cdot N_1 \cdot N_2 = t$$
mache das folgende:
   - Bestimme die Einheit $q$ mit maximalem $\tau$:
   $$q := \arg\max_{c \in \mathcal{A}} \tau_c.$$
   - Bestimme unter den Nachbarn von $q$ die Einheit $f$ mit größtem Abstand zu $q$:
   $$f := \arg\max_{c \in \mathcal{N}_q} \|\mathbf{w}_q - \mathbf{w}_c\|$$
   - Abhängig von der relativen Lage von $q$ und $f$ fahre fort mit einem von zwei Fällen:
   **Fall 1:** $q$ und $f$ sind in derselben *Zeile* des Gitters, d.h,
   $$q = a_{i,j} \wedge (f = a_{i,j+1} \vee f = a_{i,j-1}).$$

  - Füge eine neue Spalte mit $N_1$ Einheiten zwischen den Spalten von $q$ und $f$ ein.
  - Interpoliere die Referenzvektoren der neuen Einheiten von den Referenzvektoren ihrer direkten Nachbarn in der gleichen Zeile.
  - Erhöhe den Spaltenzähler: $N_2 := N_2 + 1$.

  **Fall 2:** $q$ und $f$ sind in derselben *Spalte* des Gitters, d.h,

$$q = a_{i,j} \wedge (f = a_{i+1,j} \vee f = a_{i-1,j}).$$

  - Füge eine neue Zeile mit $N_2$ Einheiten zwischen den Zeilen von $q$ und $f$ ein.
  - Interpoliere die Referenzvektoren der neuen Einheiten von den Referenzvektoren ihrer direkten Nachbarn in der gleichen Spalte.
  - Erhöhe den Zeilenzähler: $N_1 := N_1 + 1$.

- Setze alle lokalen Zählerwerte zurück: $\tau_c := 0 \quad (\forall c \in \mathcal{A})$.
- Setze den Zeitparameter zurück: $t := 0$.

10. Wenn die gewünschte Netzwerkgröße noch nicht erreicht ist, d.h., wenn

$$N_1 \cdot N_2 < N_{\text{min}},$$

fahre fort mit Schritt 2.

*Feinabstimmungsphase*

11. Generiere ein Eingabesignal: $\boldsymbol{\xi} :=$ (zufällig gemäß $p(\boldsymbol{\xi})$)

12. Inkrementiere den Zeitparameter: $t := t + 1$

13. Bestimme den Gewinner $s = s(\boldsymbol{\xi})$ für das aktuelle Eingabesignal.

14. Adaptiere jede Einheit $r$ gemäß

$$\Delta \mathbf{w}_r := \epsilon(t)\, h_{rs}(\boldsymbol{\xi} - \mathbf{w}_r)$$

wobei

$$\epsilon(t) := \epsilon_0(\epsilon_1/\epsilon_0)^{t/t_{\max}}$$

und

$$t_{\text{max}} := N_1 \cdot N_2 \cdot \lambda_f.$$

15. Wenn $t < t_{\text{max}}$ fahre fort mit Schritt 11

Abbildung 9.6 zeigt eine Simulation dieses Modells für eine ringförmige Wahrscheinlichkeitsverteilung. Die benutzten Parameter für die Wachstumsphase waren: $\lambda_g = 30$, $\sigma = 0.7, \epsilon_0 = 0.005$. Die Parameter für die Feinabstimmungsphase waren: $\sigma$ und $\epsilon_0$ unverändert, $\epsilon_1 = 0.005$, $\lambda_f = 100$, $N_{\text{min}} = 100$.

In Abbildung 9.7 sind Netzwerke verschiedener Dimensionalität gezeigt.

Ein den wachsenden Gitterns verwandtes Verfahren, das jedoch eine andere Einfügestrategie verfolgt und außerdem mit abnehmender Adaptionsstärke arbeitet, wurde von Bauer & Villmann [1995] vorgeschlagen.

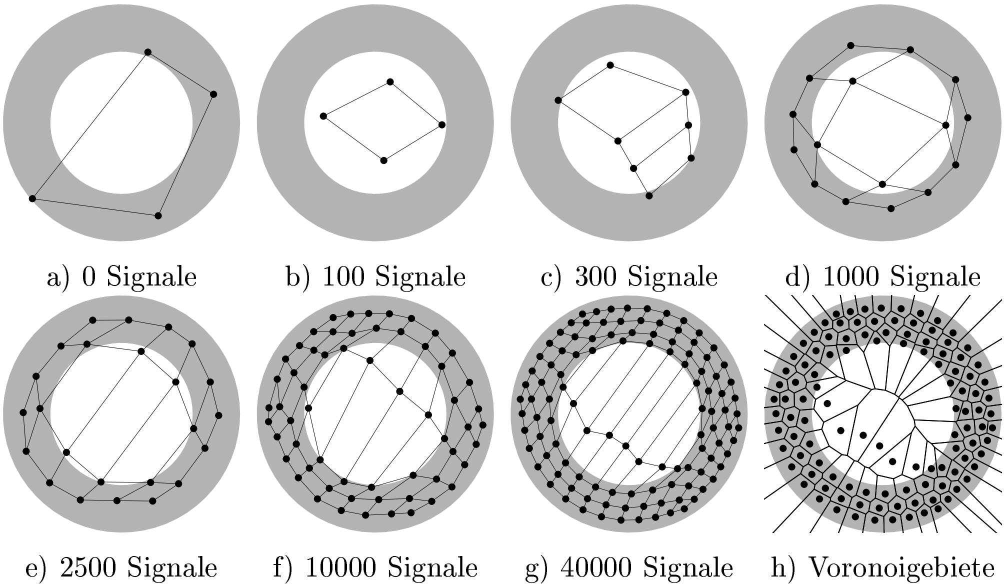



Abbildung 9.6: Wachsende Gitter. Simulationssequenz mit einer Wahrscheinlichtsverteilung, die uniform in einem ringförmigen Gebiet ist. a) Initialzustand b-f) Zwischenzustände. g) Endkonfiguration. h) zur Endkonfiguration korrespondierende Voronoitessellation. Die Topologie dieses wachsenden Netzwerks ist festgelegt auf ein rechteckiges Gitter.

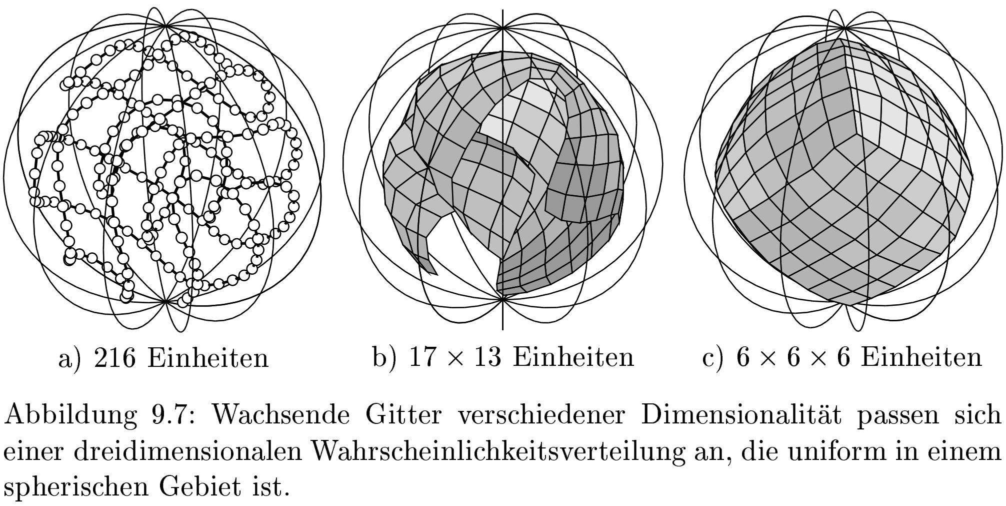



Abbildung 9.7: Wachsende Gitter verschiedener Dimensionalität passen sich einer dreidimensionalen Wahrscheinlichkeitsverteilung an, die uniform in einem spherischen Gebiet ist.

# Teil III

# Überwachtes Vektorbasiertes Lernen

# Kapitel 10

# Ziele überwachten Lernens

Bei vielen Problemstellungen sind außer den Eingabedaten $\boldsymbol{\xi} \in \mathrm{R}^n$ auch zugehörige erwünschte Ausgabedaten $\boldsymbol{\zeta} \in \mathrm{R}^m$ gegeben. Man kann sich vorstellen, daß ein „Lehrer" die Ausgabedaten zur Verfügung stellt. Daher wird von *überwachtem Lernen* gesprochen, wenn ein System mit solchen Daten adaptiert wird.

Die konkrete Aufgabe des lernenden Systems besteht darin, die zugrundeliegende Relation zwischen Ein- und Ausgabedaten zu lernen. Charakteristisch ist dabei, daß die Relation nicht explizit gegeben ist, sondern nur durch eine Menge von Paaren von Beispieldaten

$$\mathcal{D} = \{(\boldsymbol{\xi}^1, \boldsymbol{\zeta}^1), (\boldsymbol{\xi}^2, \boldsymbol{\zeta}^2), \ldots, (\boldsymbol{\xi}^M, \boldsymbol{\zeta}^M)\}.$$

Einige Beispiele für Einsatzgebiete überwachten Lernens sind die akustische Klassifizierung von Motoren in defekte und nicht-defekte, die Steuerung des Walzendrucks in einem Stahlwerk, die Prognose von Zeitreihen, die visuelle Detektion von Oberflächenfehlern aus Videoaufnahmen, oder die Bestimmung eines Flugzeugtyps anhand von Radardaten.

Diese Reihe ließe sich problemlos fortsetzen und illustriert die Bedeutung dieses Lernparadigmas. In Abhängigkeit von der Art der Ausgabedaten unterscheidet man zwei Arten des überwachten Lernens, die Funktionsapproximation (auch Regression genannt) und die Klassifikation, die im folgenden näher charakterisiert werden.

## 10.1 Funktionsapproximation

Bei der Funktionsapproximation (auch *Regression* genannt) sind die Ausgabewerte kontinuierlich. Ein Beispiel wäre die Schätzung des Fettanteils von Schweinehälften anhand von Ultraschallmeßdaten. Die Aufgabe des lernenden Systems besteht allgemein darin, die unbekannte Relation

$$\boldsymbol{\zeta} = f(\boldsymbol{\xi}),$$

die den Trainingsdaten zugrunde liegt, zu approximieren.

Ein grundsätzliches Problem rührt daher, daß die Trainingsdaten in fast allen praktischen Fällen mit statistischen Fehlern (Störungen, Rauschen) behaftet sind, die z.B. von Meßungenauigkeiten herrühren können. Wenn sich diese Fehler auf die Ausgabedaten beschränken (z.B. weil die Eingabedaten mit großer Präzision vorgegeben werden können), dann heißt das, daß die Ausgabedaten des Trainingsdatensatzes erzeugt werden gemäß

$$\boldsymbol{\zeta}^i = f(\boldsymbol{\xi}^i) + \epsilon,$$

wobei $\epsilon$ eine Zufallsgröße mit Erwartungswert Null ist, die oft als normalverteilt angenommen wird.

Ein System $S$, das die Trainingsdaten perfekt lernt, so daß die Bedingung

$$S(\boldsymbol{\xi}^i) = \boldsymbol{\zeta}^i \quad (\forall i \in \{1, \ldots, M\}\ )$$

erfüllt ist, hat auch das Rauschen $\varepsilon$ mitgelernt, das nach Voraussetzung keinerlei Information über die gesuchte Funktion $f(\boldsymbol{\xi})$ enthält. Man bezeichnet diese Situation als *Überanpassung* (engl. over-fitting). Da der Rauschterm bei jedem Datenpunkt anders ausfällt, ergeben verschiedene Datensätze, selbst wenn sie auf den gleichen Eingabevektoren bestehen, unter Umständen sehr verschiedene Zustände des Systems $S$. Dies Problem wird als das *Varianzproblem* bezeichnet und seine Lösung ist eine zentrale Fragestellung im Bereich der neuronalen Netzwerke, aber auch in der klassischen Statistik.

Eine Möglichkeit, die Varianz, in diesem Zusammenhang die Abhängigkeit des Ergebnisses vom gewählten Datensatz, zu verringern, ist die Reduktion der Freiheitsgrade des Systems $S$. Dies führt typischerweise dazu, daß die Trainingsdaten nicht mehr perfekt gelernt werden können, gleichzeitig werden die Ergebnisse für verschiedene Datensätze (die sich nur in ihrem Rauschterm unterscheiden) ähnlicher. Reduziert man jedoch die Freiheitsgrade des Systems zu stark, so ist es nicht mehr in der Lage, die Funktion $f(\boldsymbol{\xi})$ gut zu approximieren. Kennzeichnend dafür ist ein sogenannter *Bias*, eine *systematische* Abweichung von $S(\boldsymbol{\xi})$ und $f(\boldsymbol{\xi})$:

$$E(S(\boldsymbol{\xi})) \neq E(f(\boldsymbol{\xi})).$$

Dabei bezieht sich der Erwartungswert $E(\cdot)$ auf unterschiedliche Trainingsdatensätze, mit denen die Parameter des Systems $S$ bestimmt werden.

Generell ist bei einer begrenzten Datenmenge, wie sie in vielen Fällen vorliegt, eine Abwägung zwischen Bias und Varianz notwendig [Geman et al., 1992]. In Abbildung 10.1 ist das Bias/Varianz-Problem am Beispiel der Approximation einer univariaten Funktion durch Polynome gezeigt. Es existieren zwei Datensätze 1 und 2, die durch „Messungen“ der Funktion $f(\xi) = \xi^2$ an der gleichen Menge von Punkten entstanden sind. Dabei wurde das Meßergebnis von einem normalverteilten Rauschen überlagert, so daß die Datensätze sich in den Ausgabewerten etwas unterscheiden. Wird ein lineares Polynom an die Datensätze angepaßt, ergeben sich zwar für beide Datensätze ähnliche Parameter, aber die Approximation der zugrundeliegenden Funktion ist nicht gut. Ein Polynom zweiten Grades ergibt erwartungsgemäß eine sehr gute Modellierung der zugrundeliegenden Funktion. Gleichzeitig ähneln sich die Polynome für beide Datensatze sehr, d.h. die Abhängigkeit vom jeweiligen Datensatz ist relativ gering. Berechnet man jedoch die interpolierenden Polynome, die durch die Punkte in jeweils einem Datensatz definiert sind, ergeben sich für beide Datensätze ganz unterschiedliche Ergebnisse. Verwendet man diese Polynome, um den Wert von $f(\xi)$ an Stellen zu schätzen, die nicht in den Trainingsdatensätzen enthalten sind, kann es zu extrem großen Abweichungen sowohl von der Funktion $f(x)$ als auch zwischen den beiden Polynomen kommen. Ein grundsätzliches Problem ist, daß die Komplexität der Funktion, die einem gegebenen Datensatz zugrundeliegt, normalerweise nicht bekannt ist. Daher ist die Wahl der richtigen Komplexität (z.B. die Wahl einer quadratischen Funktion im Beispiel von Abbildung 10.1) nicht ohne weiteres möglich. Wenn hingegen sogenanntes *a priori* Wissen über die zugrundeliegende Funktion bekannt ist, kann die Ausnutzung dieser Information zu erheblich verbesserten Ergebnissen führen.

## 10.2 Musterklassifizierung

Bei der Musterklassifizierung ist das Ziel, das Eingabemuster $\boldsymbol{\xi}$ einer von endlich vielen Klassen möglichst korrekt zuzuordnen. Ein Beispiel ist ein System zur Erkennung handgeschriebener Ziffern auf Überweisungsformularen. Wenn die Ziffern durch vorangehende Arbeitschritte schon lokalisiert wurden (was selbst bereits ein kompliziertes Problem darstellen kann), muß jedes Eingabemuster einer von $(10 + k)$ Klassen zugeordnet werden, wobei $k$ für die Anzahl der Zeichen steht, die nicht Ziffern sind, also etwa Leerzeichen, Bindestrich, Punkt, Komma.

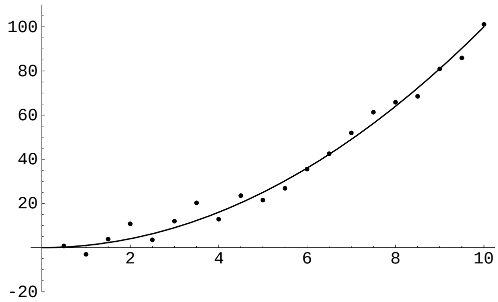
Im Gegensatz zur Funktionsapproximation ist die Zielinformation $\boldsymbol{\zeta}$ bei der Musterklassifizierung meist diskret, d.h. sie kann nur eine enliche (und oft kleine) Anzahl von Werten annehmen. Die Trainingsdaten bestehen aus einer Menge $\mathcal{D}$ von Eingabevektoren, denen jeweils eine Klassenbezeichnung beigegeben ist. Allerdings werden oft auch die Ausgaben eines adaptiven Systems $S$ nicht direkt als Klassenbezeichnungen interpretiert, sondern als *a posteriori* Wahrscheinlichkeiten $P(C_k|\boldsymbol{\xi})$ für die verschiedenen Klassen $C_k$. In diesem Fall verwendet man typischerweise eine Ausgabeeinheit pro auftretender Klasse und stellt sicher, daß sich die Ausgabewerte zu Eins ergänzen, so daß sie als Wahrscheinlichkeiten interpretierbar sind.

Dieses Vorgehen hat seine Grundlagen in der klassischen statistischen Mustererkennung [Duda & Hart, 1973] und basiert auf dem Bayes'schen Theorem

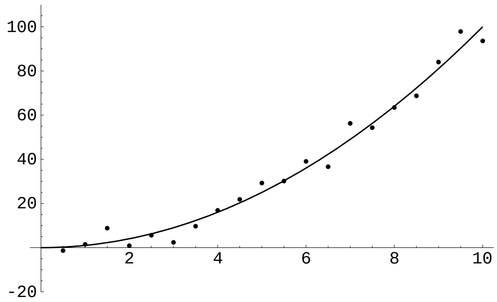
$$P(C_k|\boldsymbol{\xi}) = \frac{p(\boldsymbol{\xi}|C_k)\, P(C_k)}{p(\boldsymbol{\xi})}. \tag{10.1}$$

Dabei ist $p(\boldsymbol{\xi}|C_k)$ die bedingte Wahrscheinlichkeitsdichte von $\boldsymbol{\xi}$, gegeben, daß die Klasse $C_k$ vorliegt und $P(C_k)$ ist die *a priori* Wahrscheinlichkeit der Klasse $C_k$.

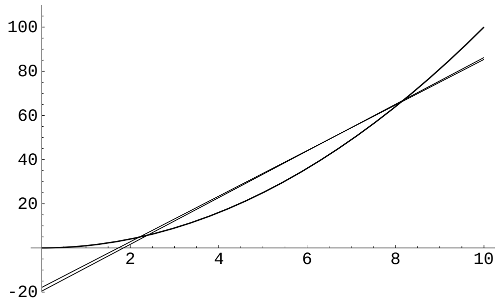
Die auf dem Bayes'schen Theorem aufbauende *Bayes'sche Regel* besagt, daß zur Minimierung des zu erwartenden Klassifizierungsfehlers, ein Eingabemuster $\boldsymbol{\xi}$ derjenigen Klasse zuzuordnen ist, die die größte *a posteriori* Wahrscheinlichkeit aufweist. Die Anwendung von Gleichung (10.1) erfordert eine Modellierung der bedingten Wahrscheinlichkeitsdichten $p(\boldsymbol{\xi}|C_k)$. Beim Einsatz neuronaler Netzwerke versucht man jedoch meist, die *a posteriori* Wahrscheinlichkeiten



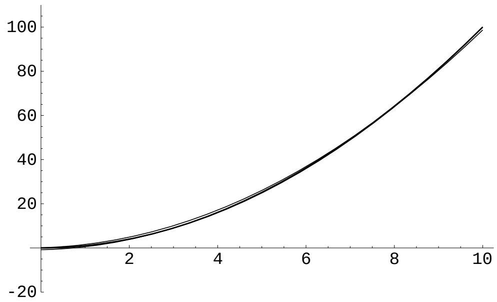

a) Datensatz 1



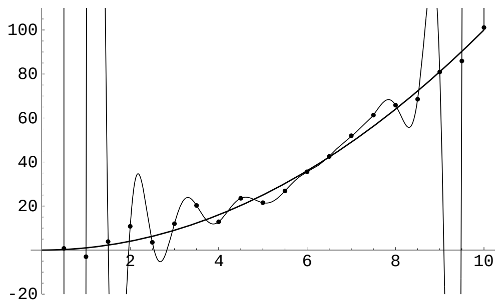

b) Datensatz 2



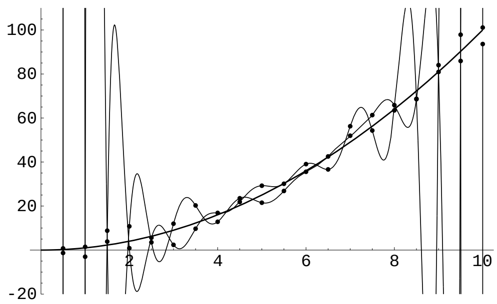

c) lineare Approximationen




d) quadratische Approximationen

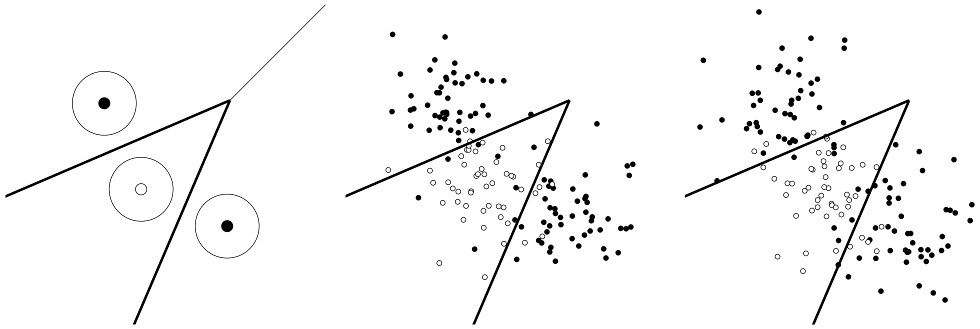



e) Polynominterpolation von Datensatz 1




f) Polynominterpolationen von Datensatz 1 und von Datensatz 2

Abbildung 10.1: Illustration des Bias/Varianz-Problems. Von einer univariaten Funktion $f(\xi)$ sind zwei Trainingsdatensätze von je 20 Punkten gegeben, die sich durch ein überlagertes Rauschen (z.B. von Meßungenauigkeiten herrührend) voneinander unterscheiden (a,b). Die Anpassung linearer Modelle an beide Datensätze ergibt zwar einheitliche Ergebnisse, aber die Approximation von $f(x)$ ist schlecht (c). Das lineare Modell hat einen großen Bias. Quadratische Modelle ergeben in diesem Fall eine sehr gute Anpassung an $f(\xi)$ und gleichzeitig ähnliche Ergebnisse für beide Datensätze (d). Hochgradige Polynome repräsentieren die Trainingsdaten jeweils perfekt (e), aber die Funktion $f(\xi)$ ist nicht gut repräsentiert und die Polynome für die beiden Datensätze unterscheiden sich erheblich (f). Das hochgradige Polynom hat einen Bias von Null, aber eine große Varianz.



a) W.-Verteilung b) Datensatz 1 c) Datensatz 2

Abbildung 10.2: Wahrscheinlichkeitsverteilung für ein Klassifizierungsproblem und zwei damit generierte Stichproben. Aus einer Wahrscheinlichkeitsverteilung, die aus der Überlagerung von drei Normalverteilungen mit gleicher Varianz besteht (a) wurden zwei verschiedene Datensätze zufällig generiert (b,c). Der optimale Bayesklassifikator kann für diese Verteilung aus der Voronoitessellation, die zu den Zentren der Normalverteilungen korrespondiert, berechnet werden (als fetter Winkel eingetragen).

direkt zu schätzen. Mit Hilfe der Bayes'schen Regel läßt sich aus den geschätzten *a posteriori* Wahrscheinlichkeiten dann direkt eine Klassifizierung für ein gegebenes Eingabemuster ableiten.

Ganz ähnlich wie bei der Funktionsapproximation tritt auch bei der Klassifizierung das Problem auf, Bias und Varianz gegeneinander abzuwägen. Wieviele freie Parameter, bzw. wieviel Flexibilität sollte man einem System $S$ (z.B. einem neuronalen Netzwerk) zubilligen, um die *a posteriori* Wahrscheinlichkeiten oder auch direkt die Grenzen zwischen verschiedenen Klassen zu modellieren?

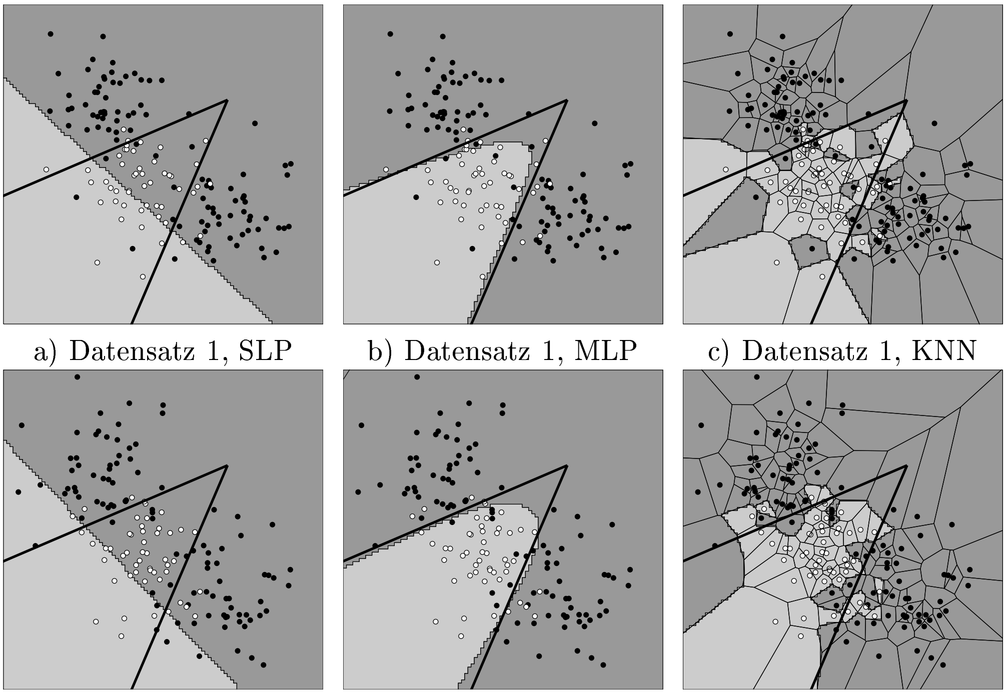
Wie bei der Funktionsapproximation können die Trainingsdaten verrauscht sein, was sich z.B. darin äußern kann, daß ein Teil der Daten mit falschen Klassenbezeichnungen versehen ist. Als zusätzliches Problem kommt bei der Klassifizierung jedoch hinzu, daß es Bereiche im Eingaberaum geben kann, die gar nicht eindeutig einer Klasse zuzuordnen sind, da mehrere Klassen $C_i$ hier eine von Null verschiedene *a posteriori* Wahrscheinlichkeit $P(C_k|\boldsymbol{\xi})$ besitzen. Selbst der theoretisch optimale Bayes-Klassifikator, der die Kenntnis der tatsächlichen *a posteriori* Wahrscheinlichkeiten voraussetzt und jedes Eingabemuster entsprechend der Bayes'schen Regel klassifiziert, würde hier einen von Null verschiedenen zu erwartenden Fehler haben, der auch als Bayes-Fehler bezeichnet wird.

Für das Training eines neuronalen Netzes mit einer begrenzten Datenmenge bedeutet dies, daß auch dann, wenn die Trainingsdatenmenge perfekt klassifiziert wird, dies keinerlei Garantie für eine gute Klassifizierung neuer Daten beinhaltet. Das eigentliche Ziel ist daher nicht, die Trainingsdaten perfekt zu lernen, sondern den Bayes-Klassifikator möglichst gut anzunähern. In Abbildung 10.2 und 10.2 ist die Bedeutung der richtigen Netzwerkkomplexität an einem Beispiel illustriert. Man beachte, daß die sehr guten Ergebnisse, die in diesem Fall mit dem mehrschichtigen Perzeptron erzielt wurden, einerseits davon herrühren, daß die Größe des Netzwerks genau so gewählt war, daß es den optimalen Klassifikator realisieren konnte, der in diesem Fall bekannt war. Hinzu kommt, daß die gezeigten Ergebnisse durchaus nicht die einzigen waren. In Simulationen mit diesem Netzwerk waren in der Tat die weitaus meisten Ergebnisse fast identisch zu den in Abbildung 10.3a) und 10.3a) gezeigten für das einschichtige Perzeptron, d.h., das Trainingsverfahren (Backpropagation) hat oft nur ein *lokales* Minimum der Fehlerfunktion gefunden. Hier sind jedoch absichtlich zwei „gute" Ergebnisse wiedergegeben um zu illustrieren, daß die Ergebnisse mit einem neuronalen Netzwerk sehr gut werden können, wenn man dem Netzwerk eine passende Komplexität gibt.

Da in fast allen praktischen Klassifizierungsproblemen die Wahrscheinlichkeitsverteilung der unterschiedlichen Klassen nicht bekannt ist, kennt man auch den Bayes-Klassifikator nicht. Deshalb hat man in der Regel *a priori* keine Anhaltspunkte wie komplex ein Netzwerk sein muß. Es sollte idealerweise gerade komplex genug sein, um den Bayes-Klassifikator approximieren zu können, jedoch nicht komplexer, da sonst die Gefahr der zu starken Anpassung an die meist begrenzte Trainingsdatenmenge besteht.

Es gibt zahlreiche Ansätze, die versuchen, das Problem der richtigen Netzwerkgröße/-komplexität zu lösen. Einer dieser Ansätze besteht darin, die verfügbaren Daten aufzuspalten in eine Trainings- und eine Validierungsmenge. Nur die Trainingsdaten werden dazu verwendet, die Parameter für immer komplexere Netzwerke zu bestimmen (oder, falls möglich, die Komplexität eines einzigen Netzwerks immer weiter zu steigern). Parallel dazu wird beobachtet, wie gut das trainierte Netzwerk die Validierungsdaten abbildet. Oft sinkt der Fehler anfangs sowohl bei den Trainings- als auch bei den Validierungsdaten. Ab einem bestimmten Zeitpunkt jedoch wird der Fehler für die Validierungsdaten nicht mehr geringer obwohl der Fehler für die Trainingsdaten noch sinkt. Dies kann als Zeichen für Überanpassung gedeutet werden und als Signal, die Komplexität des Netzwerks nicht mehr weiter zu steigern.

Die gleiche Strategie kann auch verwendet werden, um das Training eines Netzwerks zu beenden, bevor der Fehler auf den Trainingsdaten sein Minimum erreicht hat („early stopping"). Weitere Ansätze beginnen mit komplexen Netzwerken und reduzieren deren Komplexität schrittweise, z.B. durch Entfernen von Einheiten und Gewichten [Cun et al., 1990]. Weiterhin gibt es die Möglichkeit, Regularisierungsparameter zu verwenden, die z.B. einige Gewichte dazu zwingen ähnliche oder sehr kleine Werte anzunehmen. Mit solchen Maßnahmen wird die *effektive* Komplexität des Netzwerks verringert, obwohl die Struktur unverändert bleibt [Krogh & Hertz, 1992].



a) Datensatz 1, SLP b) Datensatz 1, MLP c) Datensatz 1, KNN

d) Datensatz 2, SLP e) Datensatz 2, MLP f) Datensatz 2, KNN

Abbildung 10.3: Bias/Varianz Abwägung für ein Klassifizierungsproblem. Mit jedem der beiden endlichen Datensätze aus Abbildung 10.2 wurden drei Modelle parametrisiert, bzw. trainiert: ein einschichtiges lineares Perzeptron (a,d), ein mehrschichtiges Perzeptron mit einer versteckten Schicht von zwei Einheiten (b,e), und der Nächster-Nachbar-Klassifikator wie in Abschnitt 11.1 beschrieben (c,f). Die linearen Netzwerke erzeugen zwar zwei ähnliche Klassifikatoren (geringe Varianz), schaffen es aber nicht, die Trainingsdaten mit kleinem Fehler abzubilden (hoher Bias). Die Architektur des mehrschichtigen Perzeptrons wurde in Kenntnis der Verteilung gewählt. Die erzeugten Klassifikatoren sind einander ähnlich (geringe Varianz) und approximieren gleichzeitig den Bayesklassifikator sehr gut (geringer Bias). Das Nächster-Nachbar-Verfahren klassifiziert die Trainingsdaten perfekt (kein Bias), erzeugt aber extrem unterschiedliche Klassifikationen für die beiden Datensätze (hohe Varianz).

# Kapitel 11

# Konventionelle statistische Verfahren für Klassifizierung und Regression

Die in diesem Kapitel beschriebenen Verfahren sind seit langem bekannte statistische Verfahren und stehen in enger Relation zu später beschriebenen Netzwerkmodellen. Sie sind nicht adaptiv, stellen also keine Lernverfahren dar, und benutzen immer wieder die gesamten verfügbaren Trainingsdaten zur Bearbeitung neuer Eingaben. Lernverfahren dagegen bilden mit Hilfe der Daten eine kompakte Repräsentation und verwenden danach ausschließlich diese Repräsentation.

## 11.1 $k$-Nächste-Nachbarn Methoden

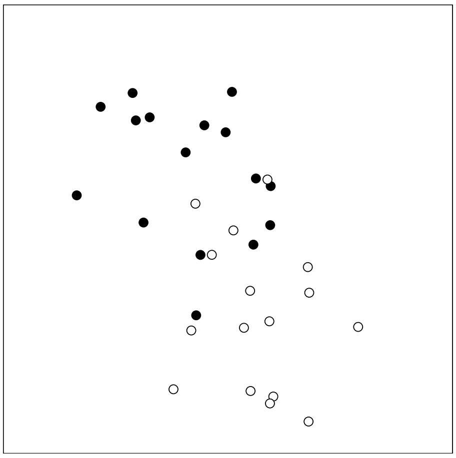
$k$-Nächste-Nachbarn-Methoden basieren auf Arbeiten von Fix & Hodges [1951] und können unter anderem für Klassifikation und Regression eingesetzt werden. Voraussetzung in jedem Fall ist die Existenz einer endlichen Datenmenge

$$\mathcal{D} = \{(\boldsymbol{\xi}^1, \zeta^1), (\boldsymbol{\xi}^2, \zeta^2), \ldots, (\boldsymbol{\xi}^M, \zeta^M)\},$$

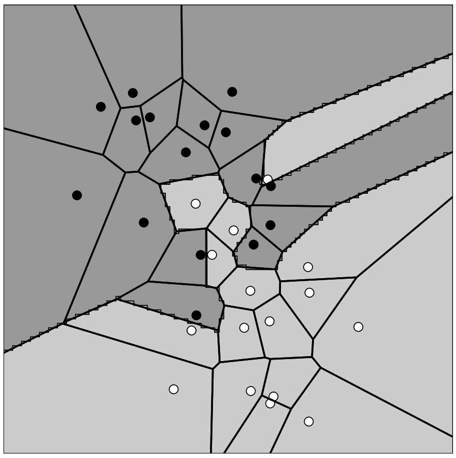
die aus Eingabedaten $\boldsymbol{\xi}^i \in \mathrm{R}^n$ besteht und zugehörigen Ausgabedaten $\zeta^i$, die im Falle von Klassifikationsproblemen Klassenbezeichnungen darstellen und im Falle von Regressionsproblemen erwünschte skalare Ausgabewerte. Die Generalisierung zu mehrdimensionalen Ausgaben $\boldsymbol{\zeta}^i$ im Falle von Regressionsproblemen ist trivial und wird hier aus Gründen der einfacheren Notation nicht gemacht.

Betrachten wir zunächst die *Klassifikation* mit einem $k$-Nächste-Nachbarn-Verfahren. Ein Vektor $\boldsymbol{\xi} \in \mathrm{R}^n$ wird klassifiziert, indem die $k$ nächsten Elemente der Datenmenge $\mathcal{D}$ bestimmt werden und $\boldsymbol{\xi}$ der Klasse zugeordnet wird, die unter diesen Elementen am häufigsten vertreten ist. Dabei kann es in bestimmten Fällen zu einem Gleichstand zwischen mehreren Klassen kommen, der z.B. durch eine Rückweisung von $\boldsymbol{\xi}$ behandelt werden kann.

Ein oft verwendeter Sonderfall ist $k = 1$, wobei man von der Nächster-Nachbar-Methode spricht. In diesem Fall wird $\boldsymbol{\xi}$ einfach der Klasse des nächsten



a) Datenmenge $\mathcal{D}$



b) Nächster-Nachbar Klassifikator

Abbildung 11.1: Nächster-Nachbar Klassifikator. Die Klassifizierung wird durch die Voronoitessellation definiert. Jeder Datenpunkt aus $\mathcal{D}$ bestimmt die Klasse für alle Eingabesignale, die in sein Voronoigebiet fallen.

Datenvektors zugeordnet. Anders formuliert, bestimmt die Klasse eines Datenvektors $\boldsymbol{\xi}^i$, wie Eingabevektoren im Voronoigebiet von $\boldsymbol{\xi}^i$ klassifiziert werden (siehe Abbildung 11.1). Wie man sich leicht überlegt, wirkt $k$ dabei als *Glättungsparameter*. Je größer der Wert von $k$ ist, desto mehr Datenvektoren einer Klasse $C_i$ müssen in einem lokalen Gebiet vorhanden sein, damit überhaupt Eingabevektoren dort als zur Klasse $C_i$ zugehörig klassifiziert werden. So lassen sich beispielsweise in dem Fall, daß es nur zwei mögliche Klassen $C_1$ und $C_2$ gibt, beliebig viele lokale Ansammlungen von $\lfloor (k-1)/2 \rfloor$ Datenvektoren der Klasse $C_2$ im Eingaberaum verteilen, ohne daß ein einziger Eingabevektor als zu der Klasse $C_2$ gehörig klassifiziert wird. Voraussetzung ist dabei nur, daß die lokalen Ansammlungen von Vektoren der Klasse $C_2$ voneinander durch hinreichend viele Vektoren der Klasse $C_1$ getrennt sind. Im Extremfall $k = M$ werden alle Eingabevektoren derjenigen Klasse zugeordnet, die in $\mathcal{D}$ am häufigsten vertreten ist.

Anstatt eine harte Mehrheitsentscheidung durchzuführen und direkt eine Klassifikation als Ergebnis zu produzieren, gibt es auch die Alternative, *a posteriori* Wahrscheinlichkeiten für die auftretenden Klassen zu schätzen. Sei $m$ die Anzahl der möglichen Klassen und seien die Bezeichnungen dieser Klassen die Zahlen $\{1, \ldots, m\}$. Dann wird die Klassenbezeichnung $\zeta^i$ jedes Trainingsvektors $\boldsymbol{\xi}^i$ kodiert in einem $m$-dimensionalen Zielvektor $\boldsymbol{\zeta}^i = (\zeta_1^i, \zeta_2^i, \ldots, \zeta_m^i)^T$ aus $m-1$ Nullen und einer Eins, dessen Komponenten gesetzt werden auf

$$\zeta_j^i = \delta_{j\zeta^i}, \qquad (\forall j \in 1, \ldots, m),$$

wobei $\delta$ das sogenannte Kronecker-Delta ist, das definiert ist als:

$$\delta_{ij} = \begin{cases} 1 & : \quad i = j \\ 0 & : \quad (\text{sonst}) \end{cases} \tag{11.1}$$

Diese Kodierung kann interpretiert werden als diskrete Wahrscheinlichkeitsverteilung über den möglichen Klassen. Da von den Trainingsdaten bekannt ist, welcher Klasse sie angehören, sind bis auf eine Komponente alle gleich Null. Für einen Eingabevektor $\boldsymbol{\xi}$ kann nun ein Vektor von *a posteriori* Wahrscheinlichkeiten ermittelt werden durch Mittelung der Zielvektoren der $k$ nächsten Nachbarn in der Datenmenge $\mathcal{D}$:

$$S(\boldsymbol{\xi}) := \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{\zeta}^{s_i(\boldsymbol{\xi})} \tag{11.2}$$

Dabei ist $s_i(\boldsymbol{\xi})$ der Index des $i$-nächsten Datenvektors und $\boldsymbol{\zeta}^{s_i(\boldsymbol{\xi})}$ entsprechend der Zielvektor dieses Datenvektors.

Für $k = 1$ ist dies offenbar äquivalent zu der oben beschriebenen harten Mehrheitsentscheidung, für größere $k$ hat diese Vorgehensweise jedoch den Vorteil, daß die Klassen, die nicht die Mehrheit bilden, im Ergebnis nicht mehr völlig unberücksichtigt bleiben, sondern zu entsprechend geschätzten *a posteriori* Wahrscheinlichkeiten führen. Offenbar läßt sich aus dem in Gleichung (11.2) berechneten Vektor mittels der Bayes'schen Regel (siehe Gleichung 10.1) sofort eine Klassifizierung ableiten. Umgekehrt lassen sich aus einer Klassifizierung jedoch keine *a posteriori* Wahrscheinlichkeiten gewinnen, so daß die „weichere“ Variante (11.2) vorzuziehen ist. Für den Extremfall $k = M$ ergibt sich für jeden beliebigen Eingabevektor das gleiche Ergebnis, nämlich der Vektor der relativen Häufigkeiten der Klassen in der Datenmenge $\mathcal{D}$.

Auch Funktionsapproximation kann mit $k$-Nächste-Nachbarn Methoden durchgeführt werden. Nach Voraussetzung sind dann die Zielwerte $\zeta^i$ Skalare. Die Ausgabe für einen Eingabevektor $\boldsymbol{\xi}$ ergibt sich dann als Mittelwert der Zielwerte der $k$ nächsten Nachbarn des Eingabesignals:

$$S(\boldsymbol{\xi}) = \frac{1}{k} \sum_{i=1}^{k} \zeta^{s_i(\boldsymbol{\xi})}$$

Wie man sich leicht überlegt, hat die so ermittelte Schätzung potentielle Unstetigkeitsstellen an den Stellen im Eingaberaum, an denen sich die Zusammensetzung der Menge der $k$ nächsten Nachbarn ändert. Im Falle $k = 1$ sind dies genau die Grenzen der Voronoigebiete der Datenvektoren $\boldsymbol{\xi}^i \in \mathcal{D}$ (siehe Abbildung 11.2). Im Fall $k = M$ ergibt sich für jeden Eingabevektor die gleiche Ausgabe, nämlich das arithmetische Mittel

$$S(\boldsymbol{\xi}) = \frac{1}{M} \sum_{i=1}^{M} \zeta^i$$

aller Zielwerte.

## 11.2 Kernelbasierte Verfahren

Ein in gewisser Weise zu $k$-Nächste-Nachbarn Methoden dualer Ansatz sind die sogenannten kernelbasierten Verfahren. Anstatt eine feste Anzahl $k$ von Nachbarn zu berücksichtigen, die je nach lokaler Punktdichte der Datenmenge

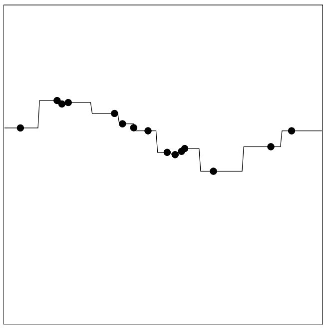


Abbildung 11.2: Regression mit dem Nächster-Nachbar Verfahren. An den Grenzen der Voronoigebiete ergeben sich Sprungstellen des Ausgabewertes.

$\mathcal{D}$ aus einem kleinen oder großen Gebiet des Eingaberaums stammen können, benutzen kernelbasierte Verfahren lokalisierte Filter fester Größe, sogenannte *Kernel*, um ein gewichtetes Mittel einer variablen Anzahl von Nachbarn zu berechnen:

$$S(\boldsymbol{\xi}) = \frac{\sum_{i=1}^{M} \zeta^i K(\boldsymbol{\xi}^i - \boldsymbol{\xi})}{\sum_{i=1}^{M} K(\boldsymbol{\xi}^i - \boldsymbol{\xi})} \tag{11.3}$$

Ein häufig verwendeter Kernel ist die $n$-dimensionale Normalverteilung

$$N(\boldsymbol{\mu}, \sigma, \boldsymbol{\xi}) := \frac{1}{(2\pi)^{n/2}\,\sigma} \exp\left(-\frac{||\boldsymbol{\xi} - \boldsymbol{\mu}||^2}{2\sigma^2}\right) \tag{11.4}$$

Zwar ist $N(\boldsymbol{\mu}, \sigma, \boldsymbol{\xi})$ im gesamten Eingaberaum von Null verschieden, kann jedoch wegen der endlichen Varianz $\sigma^2$ als lokalisiert bezeichnet werden. Wenn wir Gleichung (11.4) in Gleichung (11.3) einsetzen, erhalten wir den sogenannten Nadaraya-Watson-Schätzer [Nadaraya, 1964]:

$$S(\boldsymbol{\xi}) = \frac{\sum_{i=1}^{M} \zeta^i \exp(-||\boldsymbol{\xi}^i - \boldsymbol{\xi}||^2/\sigma^2)}{\sum_{i=1}^{M} \exp(-||\boldsymbol{\xi}^i - \boldsymbol{\xi}||^2/\sigma^2)}$$

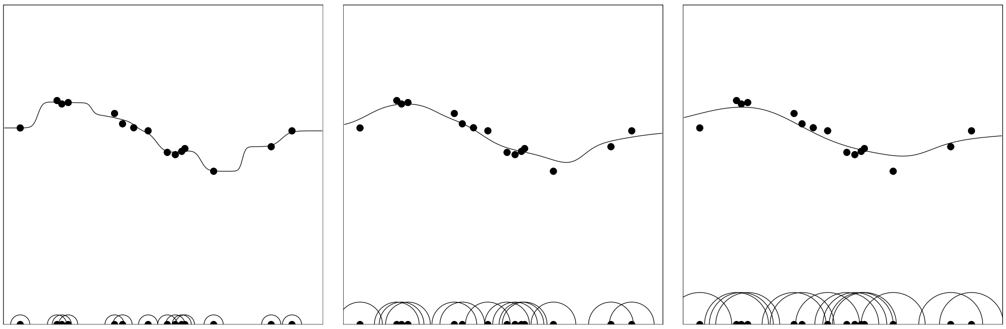
Der Parameter $\sigma$ dient dabei (vergleichbar dem $k$ bei $k$-nächster-Nachbar Methoden) als Glättungsparameter. Je kleiner $\sigma$ gewählt wird, umso mehr Gewicht erhalten die dem Eingabevektor $\boldsymbol{\xi}$ benachbarten Datenvektoren $\boldsymbol{\xi}^i$ und für den Grenzfall gilt, daß nur der nahegelegenste Datenvektor die Ausgabe entscheidet (oder die nahegelegensten, falls es mehrere in gleicher Entfernung gibt):

$$\lim_{\sigma \to 0} S(\boldsymbol{\xi}) = \boldsymbol{\zeta}^{s_1(\boldsymbol{\xi})}$$

Abbildung 11.3 gibt ein eindimensionales Beispiel für den Einfluß von $\sigma$.

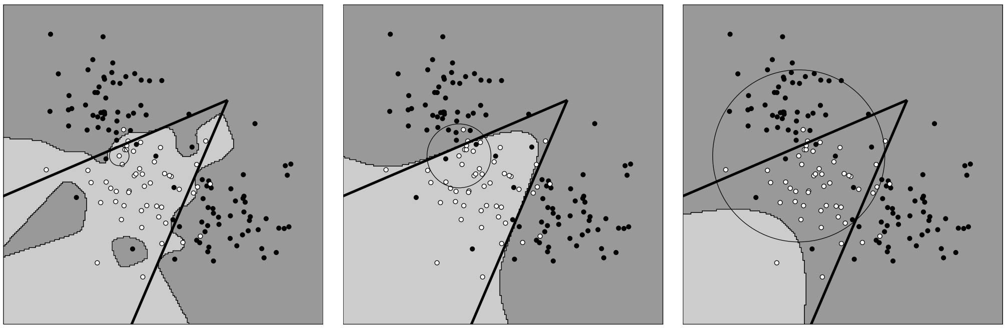
Umgekehrt nähert sich für wachsendes $\sigma$ die Ausgabe der kernelbasierten Verfahrens dem Durchschnitt aller Zielwerte:

$$\lim_{\sigma \to \infty} S(\boldsymbol{\xi}) = \sum_{i=1}^{M} \boldsymbol{\zeta}^i$$



a) $\sigma = 0.03$ b) $\sigma = 0.07$ c) $\sigma = 0.10$

Abbildung 11.3: Regression mit dem Nadaraya-Watson-Schätzer. Der gezeigte Datensatz wird mit verschiedenen Werten von $\sigma$ für die Schätzung der zugrundeliegenden Funktion verwendet.



a) $\sigma = 0.04$ b) $\sigma = 0.13$ c) $\sigma = 0.35$

Abbildung 11.4: Klassifikation mit dem Nadaraya-Watson-Schätzer. Der verwendete Datensatz ist derselbe wie in Abbildung 10.3b). Der eingezeichnete Winkel ist der optimale Bayes-Klassifikator. Für jeweils einen Datenpunkt ist zur Illustration ein Kreis mit Radius $\sigma$ dargestellt. Die verschiedenen Werte von $\sigma$ führen zu unterschiedlich guten Approximationen des Bayes-Klassifikators.

Die Klassifikation mit Kerneln funktioniert analog zu der Vorgehensweise bei $k$-nächster-Nachbar Methoden. Man kodiert die Klassenbezeichnungen $\zeta^i$ als $m$-dimensionalen Vektor $\boldsymbol{\zeta}^i$ von Wahrscheinlichkeiten und berechnet *a posteriori* Wahrscheinlichkeiten für einen Eingabevektor $\boldsymbol{\xi}$ gemäß:

$$S(\boldsymbol{\xi}) = \frac{\sum_{i=1}^{M} \boldsymbol{\zeta}^i \exp(-||\boldsymbol{\xi}^i - \boldsymbol{\xi}||^2/\sigma^2)}{\sum_{i=1}^{M} \exp(-||\boldsymbol{\xi}^i - \boldsymbol{\xi}||^2/\sigma^2)}$$

Abbildung 11.4 zeigt ein zweidimensionales Beispiel, in dem verschiedene Werte für $\sigma$ getestet werden.

# Kapitel 12

# Grundlegende Architekturen für vektorbasiertes überwachtes Lernen

In diesem Kapitel werden mehrere prinzipiell verschiedene Architekturen für überwachtes vektorbasiertes Lernen vorgestellt. Aus einer Kombination dieser Architekturen und den bereits vorgestellten unüberwacht lernenden Netzwerken läßt sich eine große Anzahl verschiedener Verfahren bilden, von denen im darauffolgenden Kapitel dann einige vorgestellt werden.

## 12.1 Radiale Basisfunktionen

Netzwerke radialer Basisfunktionen, sog. *RBF-Netze*, haben ihren Ursprung in der exakten Interpolation für eine Menge von Datenpunkten. Wir stellen daher kurz dar, wie sich eine gegebene Datenmenge

$$\mathcal{D} = \{(\boldsymbol{\xi}^1, \boldsymbol{\zeta}^1), (\boldsymbol{\xi}^2, \boldsymbol{\zeta}^2), \ldots, (\boldsymbol{\xi}^M, \boldsymbol{\zeta}^M)\}$$

mittels radialsymmetrischer Funktionen interpolieren läßt. Die am häufigsten verwendeten Funktionen für diesen Zweck sind Gauß-Funktionen der Form

$$\phi(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right),$$

aber etliche andere Funktionen können auch verwendet werden.

Um zwischen den Datenpunkten der Menge $\mathcal{D}$ interpolieren zu können, wird auf jedem Datenpunkt $j$ eine radialsymmetrische Funktion

$$\phi_j(\boldsymbol{\xi}) = \exp\left(-\frac{\|\boldsymbol{\xi}_j - \boldsymbol{\xi}\|^2}{2\sigma^2}\right)$$

zentriert und damit die Ausgabefunktion

$$S(\boldsymbol{\xi}) = \sum_{i=1}^{M} w_i \phi_i(\boldsymbol{\xi})$$

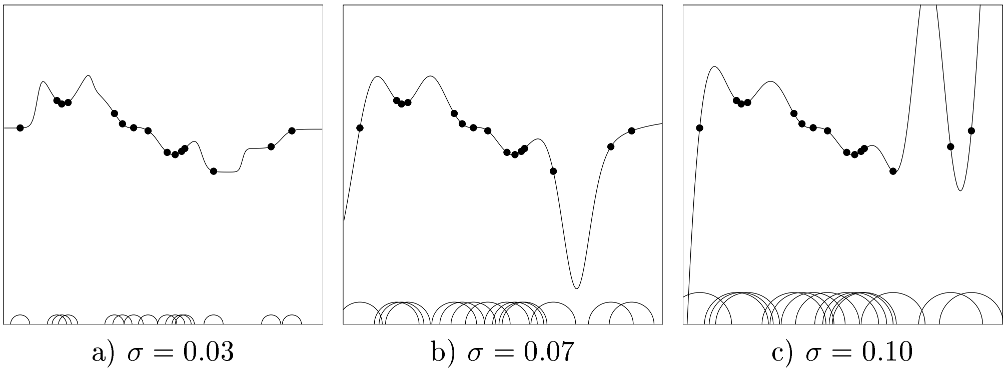



Abbildung 12.1: Exakte Interpolation mit Gaußfunktionen gleicher Breite $\sigma$. Im Gegensatz zum Nadaraya-Watson-Schätzer (vergleiche Abbildung 11.3) ergeben sich hier zwischen den Trainingsdatenpunkten Schätzwerte, die außerhalb des Bereichs der Ausgabewerte der Trainingsdaten liegen.

definiert. Die $w_i$ sind Gewichtsparameter.

Das gesetzte Ziel ist dann, Gewichte $w_i, i = 1, \ldots, M$ zu bestimmen derart, daß für alle Ein-/Ausgabepaare $(\boldsymbol{\xi}^i, \zeta^i)$ die Beziehung

$$S(\boldsymbol{\xi}^i) = \zeta^i \tag{12.1}$$

erfüllt ist.

Da sich für jedes der $M$ Ein-/Ausgabepaare aus (12.1) eine lineare Gleichung ergibt, erhält man ein lineares Gleichungssystem mit $M$ Gleichungen in $M$ Unbekannten. Vorausgesetzt, daß die Eingabevektoren $\boldsymbol{\xi}^i$ alle verschieden sind, sind die Gleichungen linear unabhängig und das Gleichungssystem kann eindeutig gelöst werden. Die resultierende Funktion $S(\boldsymbol{\xi})$ geht dann genau durch die Stützstellen $(\boldsymbol{\xi}^i, \zeta^i)$ und interpoliert dazwischen kontinuierlich und differenzierbar. Der Parameter $\sigma$, der die Breite der Gauß-Funktionen angibt, beeinflußt, wie glatt die Interpolation zwischen den Stützstellen verläuft (siehe Abbildung 12.1).

Wenn die Daten mit Rauschen behaftet sind führt die exakte Interpolation zu Ergebnissen, die stark von dem jeweils verwendeten Datensatz abhängen können (hohe Varianz) und die unbekannte erzeugende Funktion nicht gut wiedergeben (siehe auch Abbildung 10.1). Außerdem kann bei großen Datenmengen die Lösung des entsprechenden Gleichungssystems sehr aufwendig werden. Daher wurden, ausgehend von der geschilderten exakten Interpolation, einige Modifikationen vorgenommen, die zu den sogenannten *Netzwerken radialer Basisfunktionen* (RBF-Netze) führen [Moody & Darken, 1989; Shepherd & Broomhead, 1990; Poggio & Girosi, 1990]:

- Es wird eine Anzahl $N$ von lokale Funktionen verwendet, die wesentlich geringer ist als die Anzahl $M$ der gegebenen Datenpunkte

- Die Position der lokalen Funktionen im Eingaberaum muß nicht mehr mit jeweils einem der Datenpunkte zusammenfallen.

- Der Parameter $\sigma$ ist nicht mehr unbedingt gleich für alle lokalen Funktionen, sondern wird durch getrennt wählbare Parameter, $\sigma_i, i = 1, \ldots, N$, ersetzt.
- eine besondere Einheit, die sogenannte Bias-Einheit, wird eingeführt, deren Aktivierung $\phi_0$ immer gleich Eins ist und die einen von Null abweichenden Mittelwert der Zielwerte $\zeta^i$ ausgleichen kann.

Die Ausgabe des RBF-Netzes wird berechnet durch

$$y_j(\boldsymbol{\xi}) = \sum_{i=0}^{N} w_{ji}\phi_i(\boldsymbol{\xi}) \tag{12.2}$$

Es fällt die Ähnlichkeit mit der Gleichung 4.7 auf Seite 32 für ein lineares Neuron auf. In der Tat sind die Ausgabeeinheiten eines RBF-Netzwerks nichts anderes als einschichtige Perzeptrons. Allerdings sind die Eingaben nicht die $n$-dimensionalen Eingabevektoren selbst, sondern die $(N+1)$-dimensionalen Vektoren $\boldsymbol{\phi} = (\phi_0, \phi_2, \ldots, \phi_N)^T$ der Aktivierungen der lokalisierten Einheiten. Normalerweise gilt $N >> n$. Damit werden die ursprünglichen Daten nichtlinear in einen höherdimensionaleren Raum transformiert, welcher seinerseits den Eingaberaum für die folgenden linearen Einheiten bildet.

Gleichung 12.2 ergibt (wie bei der exakten Interpolation) für jedes der $M$ Datenpaare eine lineare Gleichung. Da nun aber die Anzahl der Gleichungen ($M$) größer ist als die Anzahl der Unbekannten ($N+1$) ist das resultierende Gleichungssystem überbestimmt und kann im allgemeinen nicht gelöst werden. Allerdings ist (ebenso wie bei den einschichtigen Perzeptrons in Kapitel 4) eine Lösung im Sinne eines minimalen quadratischen Fehlers

$$E[\mathbf{w}] = \frac{1}{2}\sum_{i=1}^{M}(y(\boldsymbol{\xi}) - \zeta^i)^2 \tag{12.3}$$

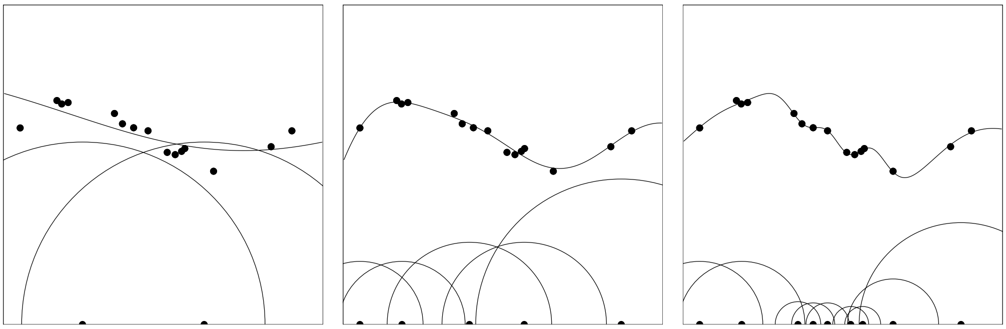
möglich durch Pseudoinversentechniken (siehe Anhang A.4).

In Abbildung 12.2 ist ein Beispiel für die Behandlung eines Regressionsproblems mit RBF-Netzwerken zu sehen.

## 12.2 Lokale konstante Abbildungen

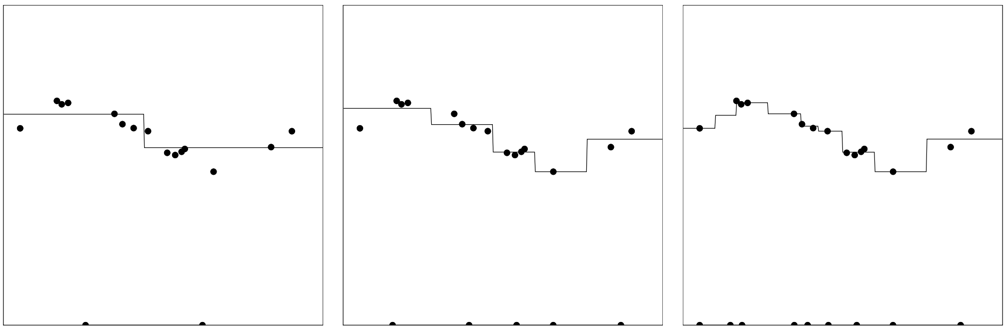
Lokale konstante Abbildungen ergeben sich als direkte Generalisierungen von $k$-Nächste-Nachbarn-Techniken. Anstatt alle gegebenen $M$ Ein-/Ausgabepaare für jedes neue Eingabesignal neu auszuwerten, wird eine Anzahl $N$, $N < M$, von Einheiten im Eingaberaum verteilt. Für jede Einheit $c$ wird die Menge der Ein-/Ausgabepaare aus der Datenmenge $\mathcal{D}$ bestimmt, deren Eingabevektoren in das Voronoigebiet $V_c$ fallen; dann wird der Ausgabewert für $c$ als das arithmetische Mittel der Zielwerte $\zeta^i$ dieser Ein-/Ausgabepaare errechnet.

Der Vorteil von der beschriebenen Methode gegenüber $k$-nächste-Nachbarn Methoden ist, daß die ursprünglichen Daten nach der Berechnung der $N$ Ausgabewerte nicht mehr gespeichert werden müssen. Die Anzahl $N$ der Einheiten ist ein Glättungsparameter und bestimmt die Genauigkeit, mit der eine gegebene Trainingsdatenmenge gelernt wird. Abbildung 12.3 zeigt ein Beispiel.



a) 2 Einheiten b) 5 Einheiten c) 9 Einheiten

Abbildung 12.2: Regression mit RBF-Netzwerken. Unterschiedlich viele lokale Einheiten wurden im Eingaberaum plaziert und mit den durch Kreise angedeuteten Breitenparametern $\sigma$ versehen. Optimale Gewichte wurden mit Pseudoinversentechniken berechnet.

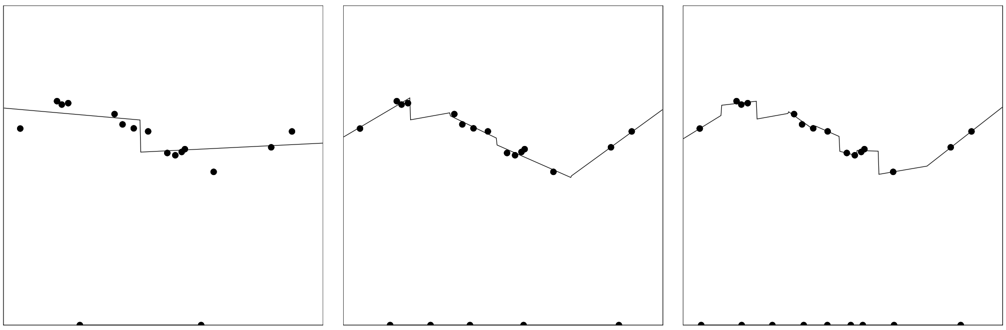


a) 2 Einheiten b) 5 Einheiten c) 9 Einheiten

Abbildung 12.3: Approximation einer Datenmenge durch lokal konstante Abbildungen.

## 12.3 Lokale lineare Abbildungen

Anstatt die Trainingsdatenpunkte innerhalb der Voronoigebiete einer Einheit durch einen konstanten Wert zu modellieren, lassen sich auch komplexere Modelle verwenden. Der nächsteinfachste Fall sind lineare Modelle. Dies führt zu einer Modellierung der zugrundeliegenden Relation $f(\boldsymbol{\xi})$ durch eine Anzahl $N$ von linearen Modellen, von den jedes auf eines der Voronoigebiete beschränkt ist, die sich durch die Verteilung von $N$ Zentren im Eingaberaum ergeben. Die Parameter der lokalen linearen Modelle lassen sich entspechend wieder durch Verwendung von Pseudoinversen oder durch iterativen Gradientenabstieg bestimmen. Abbildung 12.4 zeigt ein Beispiel.



a) 2 Einheiten  b) 5 Einheiten  c) 9 Einheiten

Abbildung 12.4: Approximation einer Datenmenge durch lokale lineare Abbildungen.

# Kapitel 13

# Beispiele konkreter Verfahren

Vom Prinzip her lassen sich sämtliche unüberwacht lernenden vektorbasierten Netzwerke mit den in Kapitel 12 vorgestellten generellen Architekturen für überwachtes Lernen kombinieren. Nicht alle Kombinationen sind sinnvoll und nur wenige sind tatsächlich untersucht worden. In diesem Kapitel werden exemplarisch zwei der möglichen Modelle vorgestellt.

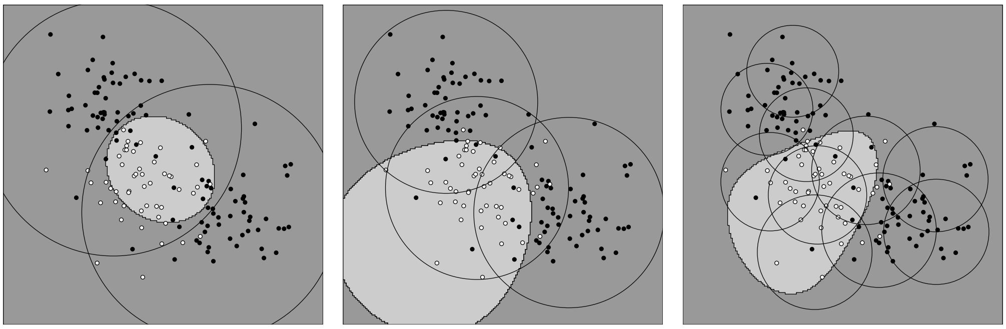
## 13.1 RBF-Netzwerke nach Moody und Darken

Moody & Darken [1989] schlagen für RBF-Netzwerke (Kapitel 12.1) ein Vorgehen im mehreren Phasen vor:

- Verwendung von LBG (Abschnitt 6.1) oder $k$-means (Abschnitt 7.3) zur Positionierung einer Menge von $N$ lokalen Einheiten innerhalb des Eingaberaumes.
- Wahl der Breitenparamter $\sigma_j$, $j = 1, \ldots, N$, durch eine Heuristik. Konkret wird vorgeschlagen, das $\sigma$ jeder Einheit proportional zum durchschnittlichen Abstand der $k$ nächsten Einheiten zu wählen.
- Bestimmung der Gewichte zu den Ausgabeeinheiten durch Pseudoinversentechniken (Anhang A.4)

Abbildung 13.1 zeigt ein Beispiel für Netzwerke verschiedener Größe, die für ein Klassifizierungproblem auf diese Weise erzeugt wurden. Ein offensichtlicher Vorteil der von Moody und Darken vorgeschlagenen Methode ist, daß alle Komponenten Standardtechniken sind, die keine inhärenten Schwierigkeiten bergen. Andererseits müssen wichtige Parameter, insbesondere die Netzwerkgröße voreingestellt werden. Die geeignete Netzwerkgröße hängt vom gegebenen Problem ab. Der ideale Anhaltspunkt wäre die Komplexität des Bayesklassifikators, die jedoch als unbekannt angesehen werden muß.

Für das Beispiel in Abbildung 13.1 sind schon drei Einheiten ausreichend. In Abbildung 13.2 ist ein komplizierteres Problem dargestellt, für das selbst 10 Einheiten nicht ausreichend sind. Das Resultat ist in diesem Fall ein hoher Fehler bereits auf den Trainingsdaten. Man kann dies interpretieren als einen



a) 2 Einheiten b) 3 Einheiten c) 10 Einheiten

Abbildung 13.1: RBF-Netzwerke nach Moody und Darken.

Bias, der auftritt, weil das Netzwerk von seiner Struktur her nicht in der Lage ist, den Bayesklassifikator hinreichend gut zu approximieren.

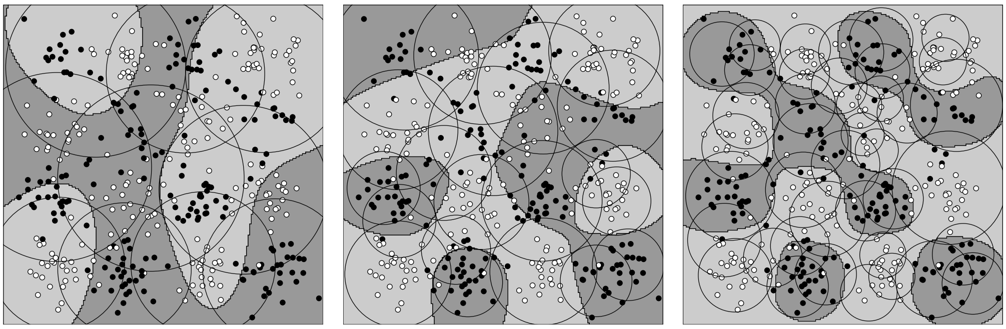
Man kann sich überlegen, daß bei diesem Problem mit 16 RBF-Einheiten eine optimale Lösung konstruierbar ist. Trotzdem wird selbst mit 20 Einheiten noch keine Konfiguration gefunden, die die Trainingsdaten gut klassifizieren kann (Abbildung 13.2 b). Der Grund dafür ist in der unüberwachten Positionierung der Zentren der Gaußfunktionen zu sehen, die durch LBG erfolgte. LBG positioniert die Zentren ohne die Klassenbezeichnungen zu beachten und ist außerdem stark von der jeweiligen Initialisierung abhängig.

Mit 40 Einheiten wird schließlich eine annehmbare Approximation des Bayesklassifikators gefunden (Abbildung 13.2 c). Jedoch kann man nicht davon ausgehen, daß stets ein kleines Mehrfaches der minimal nötigen Anzahl an RBF-Einheiten ausreicht, um mit dem Ansatz von Moody und Darken gute Ergebnisse zu erzielen. Die Initialisierung der Zentren hängt wegen der Verwendung von LBG von der unbedingten Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$ der Trainingsdaten ab. Daher lassen sich leicht Beispiele mit unterschiedlich dichten Clustern konstruieren, wo eine extrem hohe Anzahl von Einheiten notwendig wäre, damit LBG die RBF-Einheiten so positioniert, daß das Problem lösbar ist (siehe Abbildung 13.3).

## 13.2 Überwachtes Wachsendes Neuronales Gas

Das im folgenden beschriebene Verfahren SGNG (für „Supervised Growing Neural Gas“) [Fritzke, 1994b, 1996] ist keine einfache Kombination (oder „Hintereinanderschaltung“) eines unüberwacht lernenden Vektorquantisierungsverfahrens und einer überwacht lernenden Architektur. Vielmehr ist SGNG ein eigenständiges Verfahren, welches die Zielvektoren $\boldsymbol{\zeta}^i$ der gegebenen Ein-/Ausgabepaare $(\boldsymbol{\xi}^i, \boldsymbol{\zeta}^i)$ auswertet, um die lokalen Einheiten geeignet zu positionieren.

Die Grundidee ist sehr einfach. In dem ursprünglichen Verfahren „Wachsendes Neuronales Gas“ (Abschnitt 8.4) wird der akkumulierte Quantisierungsfehler benutzt, um neue Einheiten einzufügen. Dies war in dem vorausgesetzten Kontext der Vektorquantisierung angemessen, da das gesetzte Ziel die Reduzie-



a) 10 Einheiten b) 20 Einheiten c) 40 Einheiten

Abbildung 13.2: RBF-Netzwerke nach Moody und Darken. Dargestellt ist ein Klassifikationsproblem mit Daten, die aus 16 schachbrettartig verteilten Normalverteilungen stammen. Der Bayesklassifikator hätte in diesem Fall genau die Form eines $4 \times 4$-Schachbrettes. Mit 10 Einheiten können die Daten nicht ausreichend gelernt werden (a). Auch 20 Einheiten reichen in dieser Simulation für den RBF-Ansatz von Moody und Darken nicht aus, eine gute Approximation des Bayes-Klassifikators zu generieren (b). Mit 40 Einheiten wird schließlich ein recht gutes Ergebnis erzielt (c).

rung des quadratischen (Quantisierungs-)Fehlers

$$E = \frac{1}{2}\sum_{i=1}^{M} \|\boldsymbol{\xi}^i - \mathbf{w}_s\|^2$$

war. Bei überwachtem Lernen, z.B. Klassifikation, ergibt sich der interessierende Fehler aus der Differenz zwischen Netzwerkausgabe $S(\boldsymbol{\xi}^i)$ und Zielwert $\boldsymbol{\zeta}^i$:

$$E = \frac{1}{2}\sum_{i=1}^{M} \|S(\boldsymbol{\xi}^i) - \boldsymbol{\zeta}^i\|^2$$

Daher liegt es nahe, nicht mehr den Quantisierungsfehler

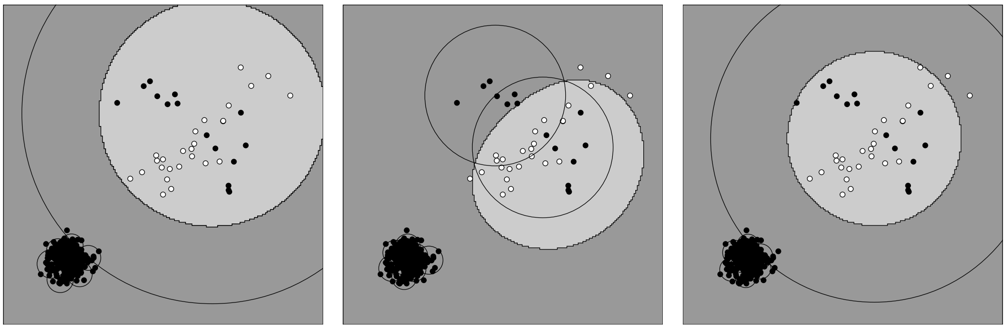
$$\|\boldsymbol{\xi}^i - \mathbf{w}_s\|^2$$

für das jeweilige Eingabesignal $\boldsymbol{\xi}^i$ lokal bei der Gewinnereinheit zu akkumulieren, sondern den *Ausgabefehler*

$$\|S(\boldsymbol{\xi}^i) - \boldsymbol{\zeta}^i\|^2$$

. In Abschnitt 8.4 wurden für die Größe die akkumuliert wird, mehrere Forderungen aufgestellt. Man kann sich überlegen, daß alle diese Forderungen für den Ausgabefehler erfüllt sind:

- *Die akkumulierte statistische Information E ist eine Fehlergröße, die reduziert werden soll.*
  Dies trifft für den Ausgabefehler zu.



a) 10 Einheiten b) 10 Einheiten c) 10 Einheiten

Abbildung 13.3: RBF-Netzwerke nach Moody und Darken. Dargestellt ist ein Klassifizierungsproblem mit großen Schwankungen der Wahrscheinlichkeitsdichte. Die Daten enthalten ein Cluster, das 90% aller Datenpunkte enthält und aus Datenpunkkten nur einer Klasse besteht. Die restlichen 10% der Daten verteilen sich auf mehrere breiter gestreute Cluster beider Klassen. Wenn LBG zufällig gemäß $p(\boldsymbol{\xi})$ initialisiert wird ist zu erwarten, daß im Mittel genau ein Vektor außerhalb des dichten Clusters ist. Dies reicht nicht aus, um ein RBF-Netzwerk zu bilden, das die gegebenen Daten lernen kann. Gezeigt sind die Ergebnissse von drei verschiedenen Simulationen.

- *Das Einfügen einer neuen Einheit bewirkt insgesamt eine Absenkung des Fehlers.*
  Auch dies ist zutreffend, denn die Einfügung einer neuen Einheit erhöht die Freiheitsgrade des Netzwerks und es kann sich den Trainingsdaten besser anpassen.

- *Die stärkste Absenkung des Fehlers ist zu erwarten, wenn die neue Einheit in der Nähe der Einheit eingefügt wird, die bisher den größten Fehler akkumuliert hat.*
  Die RBF-Einheit, die den größten Fehler akkumuliert hat, hat in ihrem Einflußbereich (in einer Umgebung von wenigen $\sigma$) vermutlich besonders viele Daten, die verschiedenen Klassen angehören, so daß keine geeigneten Gewichte zu den Ausgabeeinheiten gefunden werden können. Die Einfügung einer neuen Einheit in ihrer Nähe teilt diese Datenmenge auf und erhöht die Chance, daß geeignete Gewichte gefunden werden können. Eine besonders große Absenkung des Fehlers ist dort zu erwarten, wo zur Zeit besonders viel Fehler auftritt.

Das eigentlich Verfahren SGNG ergibt sich aus dem ursprünglichen „Wachsenden Neuronalen Gas“ wie folgt:

- Jeder lokale Einheit $c \in \mathcal{A}$ wird mit einer Gaußfunktion als Aktivierungsfunktion versehen.

- Der Breitenparameter $\sigma$ wird für jede Einheit proportional zum mittleren Abstand der direkten Nachbarn in der Netzwerktopologie gewählt.

- Entsprechend der Dimensionalität der Ausgabedaten werden gewichtete Verbindungen von allen lokalen Einheiten zu allen Ausgabeeinheiten erzeugt.

- Bei der Präsentation eines Ein-/Ausgabepaars $(\boldsymbol{\xi}^i, \boldsymbol{\zeta}^i)$ wird der Eingabevektor $\boldsymbol{\xi}$ verwendet, um gemäß der Lernregel für „Wachsendes Neuronales Gas" die Referenzvektoren der Gewinner und der direkten Nachbarn zu adaptieren.

- Die Differenz zwischen Netzwerkausgabe $S(\boldsymbol{\xi}^i)$ und Zielvektor $\boldsymbol{\zeta}^i$ wird benutzt, um die Gewichte zu den (linearen) Ausgabeeinheiten gemäß der Deltaregel (siehe Anhang A.1) zu adaptieren.

- Der quadratische Fehler $\|S(\boldsymbol{\xi}^i) - \boldsymbol{\zeta}^i\|^2$ für das augenblickliche Datenpaar wird lokal bei der Gewinnereinheit akkumuliert, also bei der RBF-Einheit, deren Zentrum dem Eingabesignal am nächsten liegt.

Das beschriebene Verfahren fügt neue lokale Einheiten dort im Eingaberaum ein, wo besonders viele Eingabesignale falsch klassifiziert werden (äquivalent bei Regressionsproblemen dort, wo Eingabesignale besonders fehlerhaft abgebildet werden). Dadurch wird der Fehler im Laufe der Zeit immer weiter abgesenkt. Bei einem endlichen Datensatz, der keine widersprüchlichen Daten enthält, sinkt der Fehler in der Regel auf Null. Spätestens, wenn die Anzahl $N$ der lokalen Einheiten gleich der Anzahl $M$ der Datenpunkte ist, ergibt sich für die Gewichte des Netzwerks ein lineares Gleichungssystem, das nicht mehr überbestimmt ist und somit exakt gelöst werden kann.

Wie jedoch in den Abschnitten 10.1 und 10.2 diskutiert wurde, ist das Ziel des Lernprozesses *nicht* das Lernen der Trainingsdaten, sondern die möglichst genaue Approximation des Bayes-Klassifikators. Da die Wahrscheinlichkeitsverteilung der Daten und damit auch der Bayes-Klassifikator normalerweise nicht bekannt sind, müssen andere Wege gefunden werden, eine Überanpassung an die Trainingsdaten zu vermeiden.

Eine oft verwendete Möglichkeit ist, (wie bereits früher erwähnt) die verfügbaren Daten aufzuspalten in einen Trainingsdatensatz und einen Validierungsdatensatz. Nur die Trainingsdaten werden verwendet, um das Netzwerk zu trainieren, parallel wird der Fehler beobachtet, den das Netzwerk auf den Validierungsdaten macht. Wenn dieser Fehler nicht mehr weiter sinkt, kann man annehmen, daß weiteres Training bzw. eine weitere Steigerung der Netzwerkgröße nur noch zu Überanpassung führt. Ein praktisches Problem dabei ist, festzustellen, wann der Fehler auf den Validierungsdaten „nicht mehr sinkt", da er während des Trainings statistischen Schwankungen unterliegt. Dies kann dazu führen, daß das Erreichen eines Fehlerplateaus erst bemerkt wird, wenn eine gewisse Überanpassung schon stattgefunden hat.

Eine denkbare Lösung dieses Problems im Falle von SGNG ist, durch Betrachtung von Verläufen des Validierungsfehlers bei zahlreichen verschiedenen Simulationen eine geeignete Netzwerkgröße zu bestimmen und anschließend Netzwerke bis zu genau dieser Größe wachsen zu lassen. Dies wurde durchgeführt für die Daten, die auch in Abbildung 13.2 verwendet wurden. Fünf

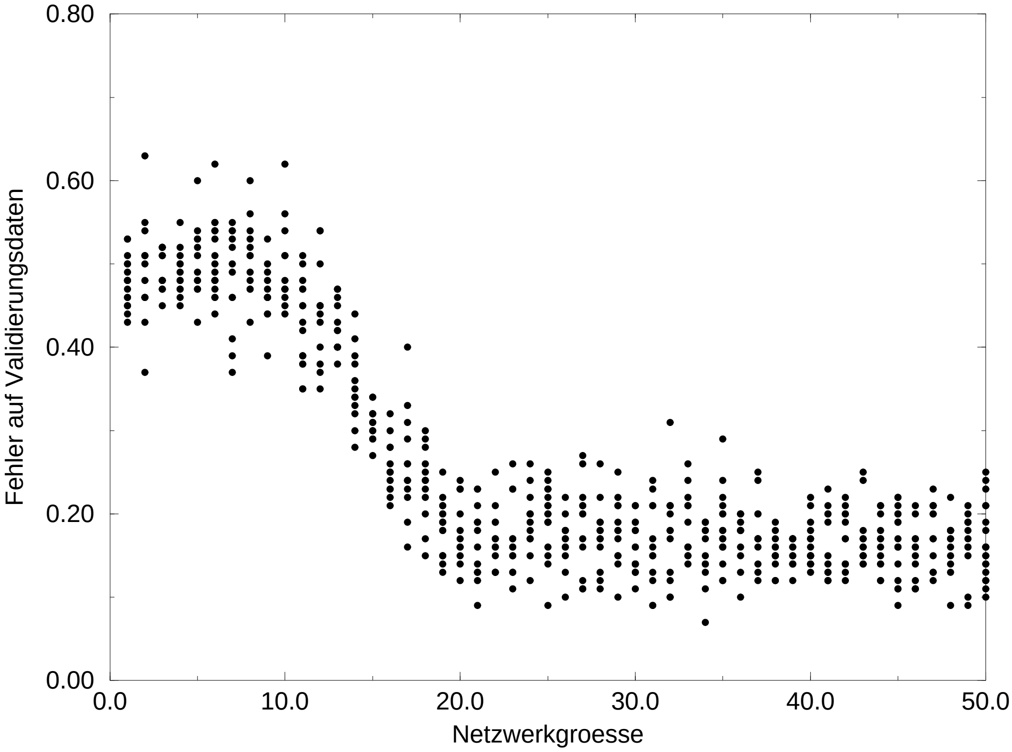
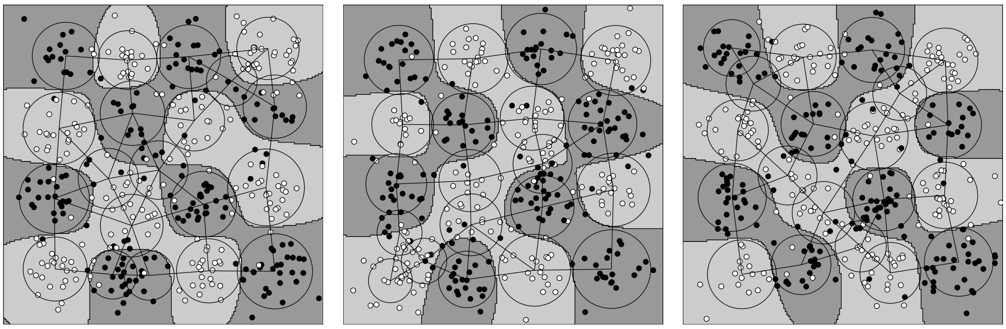



Abbildung 13.4: Überwachtes Wachsendes Neuronales Gas. Der Fehler auf den Validierungsdaten für den Datensatz aus Abbildung 13.2 wurde für fünf SGNG-Netzwerke gemessen. Für jedes Netzwerk und jede Netzwerkgröße wurden mehrere Male der Fehler gemessen (nach jeweils unterschiedlich vielen Adaptionsschritten). Daher sind für die einzelnen Netzwerkgrößen mehr als fünf Fehlerwerte zu sehen. Die Graphik zeigt deutlich, daß ab einer Netzwerkgröße von 20 der Fehler nicht mehr sinkt.

SGNG-Netzwerke wurden mit diesen Daten trainiert, wobei das Wachstum jeweils bei einer Größe von 50 Einheiten abgebrochen wurde. Parallel wurde der Fehler auf einer Validierungsdatenmenge beobachtet, die von der gleichen Verteilung erzeugt worden ist. Die Betrachtung des Validierungsfehlers, aufgetragen über der Netzwerkgröße (Abbildung 13.4), zeigt, daß ab einer Netzwerkgröße von 20 der Fehler nicht mehr zu sinken scheint.

Aufgrund dieser Beobachtung wurden mehrere Netzwerke trainiert, wobei die maximale Größe auf 20 festgelegt wurde. Um zu überprüfen, wie abhängig das Ergebnis von dem speziellen Datensatz ist (also wie hoch die Varianz ist), wurden auch anderere Datensätze verwendet, die alle von der Verteilung generiert wurden, aus der auch die Trainingsdaten stammten. Abbildung 13.5 zeigt drei typische Ergebnisse. Obwohl jeweils mit anderen Daten trainiert, sind die resultierenden Klassifikatoren einander sehr ähnlich und bilden auch den Bayes-Klassifikator gut nach. Man beachte, daß die gewählte Netzwerkgröße allein nicht auseichend ist, um ein RBF-Netzwerk für diese Datensätze erfogreich zu trainieren, wie das Beispiel des gleich großen RBF-Netzwerks in Abbildung 13.2 zeigt, dessen Zentren mit LBG positioniert wurden.

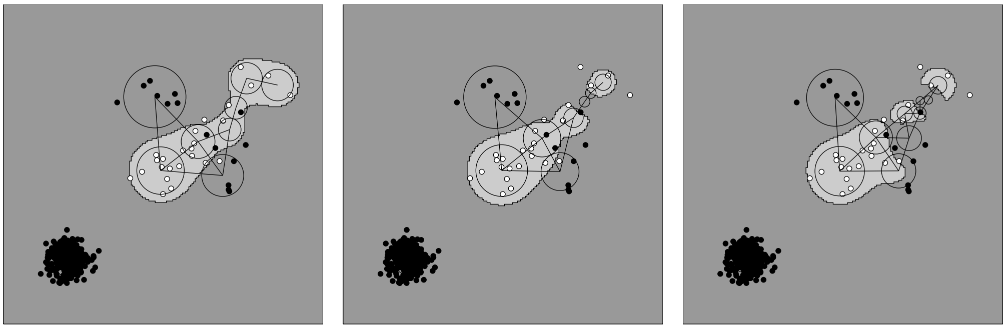


a) b) c)

Abbildung 13.5: Überwachtes Wachsendes Neuronales Gas. Dargestellt sind drei Ergebnisse von Netzwerken der Größe 20, die mithilfe der in Abbildung 13.4 gezeigten Meßergebnisse als geeignet bestimmt wurde. In allen drei Fällen hat das Training zu einem guten Ergebnis geführt.

Die fehlergetriebene Einfügestrategie von SGNG, verknüpft mit einem geeigneten Kriterium, um die Netzwerkgröße zu bestimmen, hat in diesem Beispiel zu Netzwerken fast minimaler Größe geführt, die ihre freien Parameter so geeignet einsetzten, daß der Bayes-Klassifikator gut approximiert wurde. In Abbildung 13.6 ist gezeigt, daß auch das Beispiel mit der extrem varierenden Wahrscheinlichkeitsdichte von SGNG befriedigend bearbeitet wird. Der Ansatz von Moody und Darken hatte für dieses Beispiel mit der gleichen Anzahl von Einheiten keinen guten Klassifikator produzieren können (Abbildung 13.2b).

Von Bruske & Sommer [1995] wurde unabhängig das zu SGNG recht ähnliche Modell „Dynamic Cell Structures“ vorgeschlagen, welches auch die Idee des Wachstums auf der Basis von Fehlerinformation [Fritzke, 1994a, 1991] mit der Topologieerzeugung durch Hebb'sches Wettbewerbslernen [Martinetz, 1993] verknüpft. Das „Dynamic Decay Adjustment“ von Berthold & Diamond [1995], welches wiederum auf „Restricted Coulomb Energy“ von Reilly et al. [1982] basiert, ist auch ein RBF-Modell, das die Anzahl der Einheiten schrittweise erhöht und dabei eine spezielle Heuristik zum Adaptieren der Radien (Decay) der RBF-Einheiten verwendet. Eine anderer Ansatz, RBF-Netwerke inkrementell aufzubauen, stammt von Platt [1991]. Im Gegensatz zu SGNG werden bei diesem Modell, genannt „Resource-Allocating Network“, neue RBF-Einheiten schon auf der Basis einzelner misklassifizierter Muster eingefügt und nicht auf der Basis von Fehlerwerten, die von einer ganzen Reihe von Eingabesignalen akkumuliert wurden.



a) b) c)

Abbildung 13.6: Überwachtes Wachsendes Neuronales Gas. Obwohl die Wahrscheinlichkeitsdichte der schwarzen Klasse an einer Stelle extrem hoch ist, schafft es das Verfahren in allen gezeigten Fällen durch fehlerbasierte Einfügungen, genug Einheiten an anderen Stellen zu positionieren, um die Trainigsdaten zu einem annehmbaren Grade zu lernen. Eine Methode, die wesentlich die Dichte der Daten kopiert (wie etwa $k$-means) hätte wesentlich mehr von den verfügbaren 20 Zentren bei dem schwarzen Cluster positioniert und hätte daher schlechte Vorrausetzungen für ein Separieren der Klassen geschaffen.

# Teil IV

# Ergänzungen und Schluß

# Kapitel 14

# Verwandte Verfahren

In diesem Kapitel werden zwei sehr verwandte überwacht lernende Verfahren dargestellt, die sich nicht ohne weiteres in das in diesem Buch gewählte Schema integrieren lassen, die aber auch den vektorbasierten Netzwerken zuzuordnen sind: LVQ und PSOM. Anschließend wird auf Supportvektormaschinen eingegangen, einen jüngeren Ansatz, der seine Ursprünge in der statistischen Lerntheorie hat und der in einigen Ausprägungen starke Bezüge zu vektorbasierten neuronalen Netzwerken aufweist.

## 14.1 Learning Vector Quantisation (LVQ)

LVQ ist ein überwacht lernendes vektorbasiertes Netzwerk für Klassifizierungsprobleme, welches von Kohonen [1984] vorgeschlagen wurde. Jedem Referenzvektor $\mathbf{w}_c$ wird eine Klassenbezeichnung $\boldsymbol{\zeta}_c$ zugeordnet. Das Training ist im wesentlichen hartes Wettbewerbslernen, allerdings mit einer Trainingsdatenmenge, die aus Paaren von Eingabevektor $\boldsymbol{\xi}$ und zugehöriger Klassenbezeichnung $\boldsymbol{\zeta}$ besteht. Die Klassenbezeichnung wird in der Weise verwendet, daß die Adaptionsrichtung genau *umgedreht* wird, wenn für ein Eingabesignal die Klasse $\boldsymbol{\zeta}_s$ des Gewinners $s$ nicht übereinstimmt mit der Klasse $\boldsymbol{\zeta}$ des Eingabesignals:

$$\Delta \mathbf{w}_s = \begin{cases} \epsilon(\boldsymbol{\xi} - \mathbf{w}_s) & \text{für} \quad \boldsymbol{\zeta}_s = \boldsymbol{\zeta} \\ -\epsilon(\boldsymbol{\xi} - \mathbf{w}_s) & \text{für} \quad \boldsymbol{\zeta}_s \neq \boldsymbol{\zeta} \end{cases} \tag{14.1}$$

Das Ziel, das auf diese Weise erreicht werden soll, ist eine Positionierung der Referenzvektoren nahe bei den Klassengrenzen der präsentierten Daten und damit eine möglichst gute Annäherung des Bayes-Klassifikators. Bei der Initialisierung der Referenzvektoren ist es sinnvoll, darauf zu achten, daß jeder Referenzvektor $\mathbf{w}_c$ anfangs in einem Gebiet liegt, das seiner Klasse $\boldsymbol{\zeta}_c$ zugeordnet werden kann. Wird dies nicht gemacht, ist der Referenzvektor eingeschlossen von Vektoren anderer Klassen und kann erratisch fluktuieren durch die andauernden Präsentationen von Signalen aus anderen Klassen, für die er Gewinner ist.

Mehrere Weiterentwicklungen von LVQ sind im Laufe der Zeit vorgeschlagen worden. Eine wesentliche ist LVQ2 [Kohonen, 1989], die Klassifikatoren generiert, die näher am Bayes-Klassifikator liegen, als die von LVQ generierten. Bei LVQ2 wird die Lernregel nur dann angewandt, wenn

1. der Eingabevektor $\boldsymbol{\xi}$ vom augenblicklichen Gewinner $s_1$ misklassifiziert wird

2. der zweitnächste Nachbar $s_2$ die korrekte Klasse hat; und

3. der Eingabevektor $\boldsymbol{\xi}$ hinreichend nahe an der Entscheidungsgrenze (normale Bisektorebene zwischen $\mathbf{w}_{s_1}$ und $\mathbf{w}_{s_2}$) liegt.

In diesem Fall werden *beide* Vektoren $\mathbf{w}_{s_1}$ und $\mathbf{w}_{s_2}$ adaptiert mit jeweils dem inkorrekten und dem korrekten Fall von Gleichung 14.1.

## 14.2 Parametrized Self-organizing Map (PSOM)

PSOM ist ein Regressionsverfahren, das von Ritter [1993] vorgeschlagen wurde. Eine bemerkenswerte Eigenschaft dieser Methode ist die Fähigkeit, aus wenigen Beispielen zu lernen. PSOM ist insbesondere geeignet für die Approximation von kontinuierlichen, glatten, nichtlinearen Abbildungen wie sie in der Robotik häufig auftreten. Zunächst wird eine kleine selbstorganisierende Karte (SOM) gelernt oder direkt positioniert, deren Knoten dann als Stützstellen für kontinuierliche Basisfunktionen dienen. Mit den Basisfunktionen (oft werden dafür Lagrangepolynome gewählt) wird dann eine gekrümmte Mannigfaltigkeit erzeugt, die die gewünschte Relation zwischen Ein- und Ausgabedaten realisiert. Im Gegensatz zu den ursprünglichen selbstorganisierenden Merkmalskarten erfordert die Suche nach dem Gewinner die Minimierung der Distanz zwischen Eingabevektor und einer kontinuierlichen Mannigfaltigkeit. Eine geeignete Methode, diese relativ aufwendige Suche durchzuführen, stammt von Levenberg und Marquardt [Press et al., 1992]. PSOM realisiert eine kontinuierliche Abbildung zwischen Ein- und Ausgabedaten. Dabei können die Rollen von Ein- und Ausgabedaten auch vertauscht oder anders verändert werden, was eine spezielle Eigenschaft von PSOM ist und die Verwendbarkeit dieses Verfahrens weiter steigert. Ein ausführliche Darstellung von PSOM und Erweiterungen findet sich in Walter [1996].

## 14.3 Supportvektormaschinen

Supportvektormaschinen [Schölkopf, 1997; Smola & Schölkopf, 1998; Schölkopf et al., 1998a,b; Schölkopf et al., 1995] sind eine Familie von Lernverfahren, die in jüngerer Zeit erhebliche Aufmerksamkeit erhalten haben. Supportvektormaschinen basieren auf der statistischen Lerntheorie, insbesondere auf den Arbeiten von Vapnik [Vapnik & Chervonenkis, 1971; Vapnik, 1995]. Im Gegensatz zu neuronalen Netzen wird bei Supportvektormaschinen nicht allein das *empirische Risiko* (der Fehler auf den Trainingsdaten) minimiert, sondern die Summe aus einer oberen Schranke des empirischen Risikos und aus einem Strafterm, der von der Komplexität des verwendeten Klassifizierers abhängt.

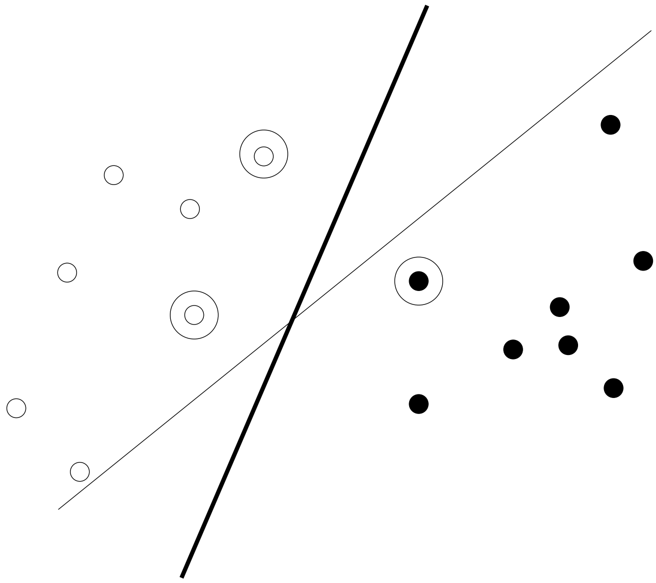


Abbildung 14.1: Lineare Supportvektormaschine. Die zwei Klassen (schwarz und weiß) sind linear separabel. Die Supportvektormaschine bestimmt diejenige Hyperebene (fett gezeichnet), die den maximalen Abstand von allen Trainingsdatenpunkten hat. Ihre Lage wird von wenigen Datenpunkten bestimmt, den sog. Supportvektoren (eingekreist). Zum Vergleich ist auch eine weitere mögliche trennende Hyperebene gezeigt, die aber nicht die Forderung des maximalen Abstandes erfüllt.

Bei *linearen Supportvektormaschinen* wird die lineare Separabilität der gegebenen Daten vorausgesetzt. Dann wird diejenige separierende Hyperebene (von den vielen möglichen) bestimmt, die den größten Abstand zu allen Trainingsdatenpunkten besitzt. Die Lage dieser Hyperebene wird eindeutig bestimmt durch wenige Trainingsdatenpunkte in der Nähe der Klassengrenze, die sogenannten *Supportvektoren* (siehe Abbildung 14.1). Durch die zusätzliche Forderung, daß die Supportvektoren von der Hyperebene auf einen Wert aus $\{+1, -1\}$ abgebildet werden, kann eine eindeutige kanonische Normalform der Hyperebene bestimmt werden. Die Bestimmung der Supportvektoren läuft auf die Lösung eines konvexen Optimierungsproblems hinaus.

Für den häufigen Fall, daß die Trainingsdaten nicht linear separierbar sind, lassen sich *nichtlineare Supportvektormaschinen* verwenden. Diese entstehen aus linearen Supportvektormaschinen dadurch, daß die Trainingsdaten zunächst unter Verwendung bestimmter Kernelfunktionen nichtlinear in einen höherdimensionaleren Raum projiziert werden, in dem sie dann linear separabel sind. In diesem Raum wird dann das oben beschriebene Verfahren zur Bestimmung der Supportvektoren durchgeführt. Dabei ist eigentlich die Bildung von Skalarprodukten in dem hochdimensionalen Raum notwendig, was sehr aufwendig wäre. Durch Verwendung einer geeigneten Kernelfunktion lassen sich aber diese Skalarprodukte auch direkt aus den ursprünglichen Daten berechnen.

Ein Zusammenhang von Supportvektormaschinen und vektorbasierten neuronalen Netzen ergibt sich dadurch, daß bei der Verwendung einer Gaussfunktion als Kernel eine Supportvektormaschine mathematisch sehr ähnlich zu einem RBF-Netzwerk ist [Schölkopf et al., 1997]. Allerdings werden die Positionen der Zentren und die Gewichte zu den Ausgabeeinheiten bei der Supportvektormaschine automatisch bestimmt. Die Zentren sind genau die Supportvektoren und die Gewichte ergeben sich aus der Lösung des konvexen Optimierungsproblems. Bei herkömmlichen RBF-Netzen hingegen (siehe Kapitel 13.1) werden die Zentren durch hartes Wettbewerbslernen bestimmt und die Gewichte durch anschließende Minimierung des quadratischen Fehlers.

# Kapitel 15

# Anwendungen vektorbasierter Neuronaler Netze

In diesem Abschnitt soll ein ein kursorischer Überblick der vielfältigen Anwendungen gegeben werden, die bereits mit vektorbasierten neuronalen Netzen realisiert wurden. Die Auflistung ist bei weitem nicht vollständig, sondern soll nur einige Beispiele geben.

Selbstorganisierende Merkmalskarten sind u.a. eingesetzt worden für

- Lernen von visoumotorischer Kontrolle eines Roboterarms [Martinetz et al., 1990]
- Spracherkennung [Kohonen, 1988; Torkkola et al., 1991]
- Explorative Datenanalyse [Ultsch & Siemon, 1990]
- Klassifikation von Hirntumoren [Vercauteren et al., 1990]
- Visualisierung semantischer Zusammenhänge in Sätzen [Ritter & Kohonen, 1989]
- näherungsweise Lösung kombinatorischer Optimierungsproblem [Angeniol et al., 1988; Favata & Walker, 1991]

Neuronales Gas ist u.a. verwendet worden für

- Prädiktion chaotischer Zeitreihen [Martinetz et al., 1993]
- Steuerung der Walzendrucks bei der Stahlherstellung

Wachsende Zellstrukturen und Wachsendes Neuronales Gas ist u.a. verwendet worden für

- Information Retrieval [Zavrel, 1995]
- Auswertung von Hochenergiephysikexperimenten [Kunze & Steffens, 1995; Berlich et al., 1996; Kunze, 1996]
- Computergraphik (Renderingverfahren) [Bohn, 1996]

- Selbstkalibrierung der Fixationsbewegungen eines Stereokamerakopfes [Pagel et al., 1998]

- Lernen der Kinematik eines redundanten Roboterarms [Maël, 1996]

LBG-U ist verwendet worden für

- Lernen von Objektstartpositinen in Videosequenzen [Loos et al., 1998]

Die älteren Verfahren LBG, $k$-Means und $k$-Nächste-Nachbarn sind für so viele Anwendungen eingesetzt worden, daß hier keine Auflistung gemacht wird. Gleiches gilt für konventionellen RBF-Netze nach Moody und Darken.

# Kapitel 16

# Zusammenfassung

Vektorbasierte Neuronale Netze (VBNN) sind Verfahren des *statistischen Lernens*, d.h. die Netzwerkparameter werden durch eine große Anzahl von Eingabesignalen adaptiert. Im Gegensatz dazu steht *momentanes Lernen* (engl. one-shot learning), das insbesondere in vielen Bereichen menschlichen Lernens eine Rolle zu spielen scheint.

Innerhalb des statistischen Lernens sind VBNN dem *Wettbewerbslernen* zuzuordnen. Der zentrale Schritt ist die Ermittlung des *Gewinners* für ein Eingabesignal. Dies kann als Wettbewerb der Einheiten des Netzwerks untereinander interpretiert werden. Da dieser Wettbewerb durch die Auswertung von (oft euklidischen) Vektordistanzen entschieden wird, sind VBNN abhängig von einer geeigneten Skalierung der Eingabedaten, was bei der *Datenvorverarbeitung* berücksichtigt werden muß.

Vergleicht man VBNN mit den bekannten mehrschichtigen Perzeptrons (MLPs), die häufig als die prototypischen „neuronalen Netze" angesehen werden, so ist ein wesentlicher Unterschied, daß MLPs in der Regel mittels Gradientenabstieg trainiert werden *ohne*, daß es dabei einen Wettbewerb zwischen den Einheiten des Netzwerks gibt, wie er bei VBNN die Regel ist. Das formale Neuron, das einem MLP zugrundeliegt, berechnet außerdem im wesentlichen ein *Skalarprodukt*, während ein Neuron eines VBNN eine *Vektordistanz* ermittelt. Während MLPs ausschließlich für überwachtes Lernen eingesetzt werden[1], können VBNN sowohl überwacht als auch unüberwacht lernen.

Bei unüberwachtem Lernen lassen sich verschiedene, sich in der Regel widersprechende, Zielsetzungen definieren, wie etwa Fehlerminimierung, Entropiemaximierung, Dichteschätzung, Clustering oder Datenvisualisierung.

Falls die gegebene Datenmenge endlich ist, können sogenannte *Batchverfahren* wie LBG oder LBG-U eingesetzt werden, um die Parameter des vektorbasierten neuronalen Netzes zu bestimmen. Sie haben den Vorteil der garantierten Konvergenz in endlich vielen Schritten. Darüberhinaus ermöglicht LBG-U die Bestimmung besserer Minima (im Sinne der Minimierung des Quantisierungsfehlers), was für Anwendungen in der Vektorquantisierung wichtig ist.

Bei sehr großen Datenmengen oder Daten, die kontinuierlich (z.B. von einem

[1] Mit wenigen Ausnahmen wie etwa dem Autoassoziatornetzwerk, das zur Datenkompression verwendet wird.

Sensor) geliefert werden, muß *Online-Lernen* verwendet werden. Das bedeutet, daß nach jedem einzelnen Eingabesignal eine Adaption einiger Parameter vorgenommen wird. Begnügt man sich damit, ausschließlich den Gewinner zu adaptieren (*hartes Wettbewerbslernen*), so ist sie wesentliche Entscheidung die über den Zeitverlauf der Adaptionsstärke. Zeitlich konstante Lernraten ermöglichen kontinuierliches Lernen und sind daher prinzipiell auch für nichtstationäre Signalverteilungen anwendbar. Allerdings gibt es keine Konvergenz (genau wegen der konstanten Lernrate). Abnehmende Lernraten dagegen führen bei stationären Verteilungen in der Regel zu geringerem Quantisierungsfehler. Generell sind harte Wettbewerbslernverfahren sehr abhängig von der Initialisierung der Referenzvektoren, da sie nur lokale Veränderungen vornehmen.

Wenn außer dem Gewinner für ein Eingabesignal auch weitere Einheiten adaptiert werden, spricht man von *weichem Wettbewerbslernen.* Verschiedene Verfahrensvarianten definieren dabei den Begriff „weitere Einheiten“ unterschiedlich, z.B. abhängig von der Distanz im Eingaberaum (Neuronales Gas) oder mittels einer auf den Einheiten des Netzwerks definierten Graphstruktur (Wachsendes Neuronales Gas).

Wenn eine Graphstruktur auf den Einheiten des Netzwerks verwendet wird und diese eine feste und niedrige Dimensionalität besitzt, kann das Netzwerk zur Datenvisualisierung verwendet werden. Existierende Verfahren, für die dies zutrifft, unterscheiden sich in der Art der Randbedingungen, die der Graphstruktur auferlegt werden. Die selbstorganisierenden Merkmalskarten verwenden eine feste Graphstruktur von der Gestalt eines rechteckigen Gitters. Die sog. wachsenden Gitter generieren ein rechteckiges Gitter durch sukzessives Einfügen von Zeilen und Spalten gesteuert durch die Eingabedaten. Sie sind auf diese Weise in der Lage, die Netzwerkstruktur den Daten unter den gegebenen Randbedingungen anzupassen. Die wachsenden Zellstrukturen erzeugen eine Struktur fester Dimensionalität, die aus Simplices (Dreiecken im zweidimensionalen Fall) besteht und sich den Daten noch besser anzupassen kann, ohne jedoch die vorgegebene Dimensionalität zu verlassen.

VBNN können auch für *überwachtes Lernen* eingesetzt werden, d.h. für Situationen, wo die Daten als Paare $(\boldsymbol{\xi}, \boldsymbol{\zeta})$ von Eingabsignal $\boldsymbol{\xi}$ und erwünschter Ausgabe $\boldsymbol{\zeta}$ vorliegen. Dabei besteht z.T. ein enger Zusammenhang zu klassischen Verfahren wie $k$-Nächste-Nachbarn oder Parzen-Windows. Man kann verschiedene Grundarchitekturen für vektorbasiertes überwachtes Lernen unterscheiden, u.a. RBF-Netze, lokal lineare und lokal konstante Abbildungen.

Prinzipiell läßt sich jede dieser Architekturen mit jedem unüberwacht lernenden vektorbasierten Netzwerk kombinieren. Daraus ergibt sich eine große Anzahl möglicher Verfahren, von den zwei exemplarisch dargestellt werden: RBF-Netzwerke nach Moody und Darken sowie überwachtes Wachsendes Neuronales Gas.

Anschliessend werden zwei sehr verwandte überwacht lernende Verfahren dargestellt, die sich nicht ohne weiteres in das gewählte Schema integrieren lassen, die aber auch den vektorbasierten Netzwerken zuzuordnen sind: LVQ und PSOM. Schließlich wird auf Supportvektormaschinen eingegangen, einen neueren in der statistischen Lerntheorie verwurzelten Ansatz, der in einigen Ausprägungen starke Bezüge zu vektorbasierten neuronalen Netzwerken aufweist.

Diverse Anwendungen vektorbasierter neuronaler Netze werden in einem weiteren Kapitel genannt.

Der Autor hofft, mit diesem Buch die vektorbasierten neuronalen Netze verständlich dargestellt, strukturiert und eingeordnet zu haben. Dieses Gebiet birgt noch viele weitere Anwendungsmöglichkeiten in Bereichen wie Robotik, Maschinenlernen, Datenanalyse und Data Mining, Datenkompression- und Visualisierung. Auch im theoretischen Bereich warten, gerade beim weichen Wettbewerbslernen, noch einige interessante mathematische Probleme auf ihre Lösung. Daher ist sowohl unter Anwendungs- als auch unter Forschungsgesichtspunkten, die Beschäftigung mit vektorbasierten neuronalen Netzen weiter lohnend.

# Anhang A

# Mathematischer Anhang

## A.1 Die Delta-Regel

Die Delta-Regel [Widrow & Hoff, 1960] ist ein Verfahren, um die Gewichte für ein lineares Neuron zu bestimmen, so daß der quadratische Fehler

$$E[\mathbf{w}] = \frac{1}{2}\sum_{i=1}^{M}(\mathbf{w}^T\boldsymbol{\xi}^i - \zeta^i)^2$$

minimiert wird

Es wird ein *Gradientenabstieg* im Raum der Gewichte durchgeführt. Der augenblickliche Gewichtsvektor wird in die Richtung verändert, in der die Fehlerfunktion am stärksten abfällt. Die Richtung *des stärksten Anstiegs* der Fehlerfunktion ist der sogen. Gradient von $E[\mathbf{w}]$:

$$\nabla_w E[\mathbf{w}] = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}\right]^T$$

Als *Gradientenabstieg* bezeichnet man entsprechend eine Bewegung im Raum $\mathrm{R}^n$ der Gewichte in Richtung des negativen Gradienten, d.h.:

$$\Delta\mathbf{w} = -\nabla_w E[\mathbf{w}]$$

Die Änderung eines einzelnen Gewichts $w_k$ ergibt sich zu:

$$\begin{aligned} \Delta w_k &= -\eta\frac{\partial E}{\partial w_k} && \text{(A.1)} \\ &= -\eta\sum_{i=1}^{M}\xi_k^i(\mathbf{w}^T\boldsymbol{\xi}^i - \zeta^i) && \text{(A.2)} \end{aligned}$$

Diese Gewichtsänderung erfordert eine Summation über alle Trainingsmuster. Wenn deren Anzahl sehr groß oder sogar unendlich ist, ist so eine Summation aufwendig bzw. unmöglich. Näherungsweise kann man dann eine Gewichtsänderung schon für ein einzelnes Musterpaar $(\boldsymbol{\xi}, \zeta)$ machen (Online Lernen):

$$\begin{aligned}
\Delta w_k &= -\eta \xi_k (\mathbf{w}^T \boldsymbol{\xi} - \zeta) && \text{(A.3)}\\
&= \eta \xi_k (\underbrace{\zeta - \mathbf{w}^T \boldsymbol{\xi}}_{\delta}) && \text{(A.4)}\\
&= \eta \xi_k \delta && \text{(A.5)}
\end{aligned}$$

## A.2 Beweis von Gleichung 7.2

**Satz:** Sei $\mathbf{w}_c$ der Referenzvektor einer Einheit $c$ des Netzwerks und sei $\boldsymbol{\xi}_1^c, \boldsymbol{\xi}_2^c, \ldots, \boldsymbol{\xi}_t^c$ die (Unter-)Folge der Eingabesignale für die die Einheit $c$ Gewinner ist, nachdem sie zufällig gemäß $p(\boldsymbol{\xi})$ initialisiert wurde. Erfolge die Adaption mit konstanter Lernrate $\epsilon_0$ gemäß

$$\mathbf{w}_c(t) := \mathbf{w}_c(t-1) + \epsilon_0 \left(\boldsymbol{\xi}_t^c - \mathbf{w}_c(t-1)\right) \tag{A.6}$$

Dann ist der Wert des Referenzvektors $\mathbf{w}_c$ zum Zeitpunkt $t$ gegeben durch:

$$\mathbf{w}_c(t) = (1-\epsilon_0)^t \mathbf{w}_c(0) + \epsilon_0 \sum_{i=1}^{t} (1-\epsilon_0)^{t-i} \boldsymbol{\xi}_i^c \tag{A.7}$$

Beweis (durch vollständige Induktion):

- t=1:

$$\mathbf{w}_c(1) = \mathbf{w}_c(0) + \epsilon_0 \left(\boldsymbol{\xi}_1^c - \mathbf{w}_c(0)\right) = (1-\epsilon_0)^1 \mathbf{w}_c(0) + \epsilon_0 \boldsymbol{\xi}_1^c (\text{erfüllt})$$

- $(t-1) \rightarrow t$: Gelte (A.7) für (t-1)

$$\begin{aligned}
\mathbf{w}_c(t) &= \mathbf{w}_c(t-1) + \epsilon_0 \left(\boldsymbol{\xi}_t^c - \mathbf{w}_c(t-1)\right) \qquad (\text{gemäß (A.6)})\\
&= (1-\epsilon_0)\mathbf{w}_c(t-1) + \epsilon_0 \boldsymbol{\xi}_t^c\\
&= (1-\epsilon_0) \underbrace{\left[(1-\epsilon_0)^{t-1}\mathbf{w}_c(0) + \epsilon_0 \sum_{i=1}^{t-1} (1-\epsilon_0)^{t-1-i} \boldsymbol{\xi}_i^c\right]}_{\text{Einsetzen der Annahme}} + \epsilon_0 \boldsymbol{\xi}_t^c\\
&= (1-\epsilon_0)^t \mathbf{w}_c(0) + \epsilon_0 \sum_{i=1}^{t} (1-\epsilon_0)^{t-i} \boldsymbol{\xi}_i^c \qquad \text{q.e.d}
\end{aligned}$$

## A.3 Beweis von Gleichung 7.4

**Satz:** Sei $\mathbf{w}_c$ der Referenzvektor einer Einheit $c$ des Netzwerks und sei $\boldsymbol{\xi}_1^c, \boldsymbol{\xi}_2^c, \ldots, \boldsymbol{\xi}_t^c$ die (Unter-)Folge der Eingabesignale für die die Einheit $c$ Gewinner ist, nachdem sie zufällig gemäß $p(\boldsymbol{\xi})$ initialisiert wurde. Erfolge die Adaption mit harmonisch abnehmender Lernrate

$$\epsilon_c(t) = \frac{1}{t}$$

gemäß

$$\mathbf{w}_c(t) := \mathbf{w}_c(t-1) + \epsilon_c(t)\,(\boldsymbol{\xi}_t^c - \mathbf{w}_c(t-1)) \tag{A.8}$$

Dann ist der Wert des Referenzvektors $\mathbf{w}_c$ zum Zeitpunkt $t$ gegeben durch:

$$\mathbf{w}_c(t) = \frac{1}{t}\sum_{i=1}^{t}\boldsymbol{\xi}_i^c \tag{A.9}$$

Beweis (durch vollständige Induktion):

- t=1:

$$\begin{aligned}\mathbf{w}_c(1) &= \mathbf{w}_c(0) + \epsilon_c(1)\,(\boldsymbol{\xi}_1^c - \mathbf{w}_c(0)) \\ &= \boldsymbol{\xi}_1^c \\ &= \frac{1}{1}\sum_{i=1}^{1}\boldsymbol{\xi}_i^c\end{aligned} \tag{A.10}$$

- $(t-1) \rightarrow t$: Gelte (A.9) für (t-1)

$$\begin{aligned}\mathbf{w}_c(t) &= \mathbf{w}_c(t-1) + \frac{1}{t}\,(\boldsymbol{\xi}_t^c - \mathbf{w}_c(t-1)) && \text{(gemäß (A.8))} \\ &= (1-\frac{1}{t})\left[\frac{1}{t-1}\sum_{i=1}^{t-1}\boldsymbol{\xi}_i^c\right] + \frac{1}{t}\boldsymbol{\xi}_t^c && \text{(Einsetzen der Annahme)} \\ &= \frac{1}{t}\sum_{i=1}^{t}\boldsymbol{\xi}_i^c \qquad \text{q.e.d}\end{aligned}$$

## A.4 Lösung von linearen Problemen der kleinsten Quadrate mit Singular Value Decomposition

Bemerkung: Die folgenden Ausführungen sind in relativ direkter Form aus Press et al. [1992] übernommen und sind hier zur Bequemlichkeit des Lesers wiedergegeben.

Sei ein überbestimmtes lineares Gleichungssystem

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{A.11}$$

mit $M$ Gleichungen in $N$ Unbekannten $x_i$, d.h. die Form der beteiligten Matrizen ist:

**A**: $M \times N$- Matrix

**x**: $N$-dim. Vektor

**b**: $M$-dim. Vektor

Da das Gleichungssystem überbestimmt ist, existiert keine exakte Lösung. Ersatzweise kann man das Ziel definieren, eine Belegung des Vektors $\mathbf{x}$ zu finden, die den quadratischen Fehler

$$E[\mathbf{x}] = (\mathbf{Ax} - \mathbf{b})^T(\mathbf{Ax} - \mathbf{b}) \tag{A.12}$$

minimiert. Eine Technik, die dies ermöglicht ist die *Singular Value Decomposition* (SVD).

**Satz**[Press et al., 1992, Kapitel 2.6]: Jede $M \times N$ Matrix $\mathbf{A}$ (M = Anzahl der Zeilen, N = Anzahl der Spalten) mit $M \geq N$ läßt sich zerlegen in das Produkt aus einer $M \times N$ spaltenorthogonalen Matrix $\mathbf{U}$, einer $N \times N$ Diagonalmatrix $\mathbf{W}$ mit nicht-negativen Elementen und der transponierten einer $N \times N$ Orthogonalmatrix $\mathbf{V}$.

SVD findet genau so eine Zerlegung und aus dieser läßt sich direkt die Näherungslösung für (A.11) ableiten mit minimalem Fehler (A.12). Die Form der beteiligten Matrizen wird durch folgendes Schema illustriert:

$$\begin{pmatrix} \\ \\ \\ \mathbf{A} \\ \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ \\ \mathbf{U} \\ \\ \\ \\ \end{pmatrix} \begin{pmatrix} w_1 & & & \\ & w_2 & & \\ & & \cdots & \\ & & \cdots & \\ & & & w_N \end{pmatrix} \begin{pmatrix} \\ \\ \mathbf{V}^T \\ \\ \\ \end{pmatrix} \tag{A.13}$$

Dabei gilt:

$$\begin{pmatrix} \\ \mathbf{U}^T \\ \\ \end{pmatrix} \begin{pmatrix} \\ \\ \mathbf{U} \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \mathbf{V}^T \\ \\ \end{pmatrix} \begin{pmatrix} \\ \mathbf{V} \\ \\ \end{pmatrix}$$

$$= \begin{pmatrix} \\ \mathbf{I} \\ \\ \end{pmatrix}$$

Wenn bei der Matrix $\mathbf{W}$ einige der Diagonalelemente Null sind, deutet das auf eine Singularität der Matrix $\mathbf{A}$ hin, d.h. $\text{rang}(\mathbf{A}) < N$.

Die Näherungslösung der Gleichung

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{A.14}$$

mit dem kleinsten quadratischen Fehler ergibt sich als

$$\mathbf{x} = \mathbf{V}[\mathrm{diag}(1/w_j)](\mathbf{U}^T \cdot \mathbf{b}) \tag{A.15}$$

Dabei wird für alle $w_j$, die den Wert Null haben anstelle von $(1/w_j)$ der Wert Null verwendet.

Das entsprechende Tableau ist

$$\left(\begin{array}{c} \\ \mathbf{x} \\ \\ \end{array}\right) = \left(\begin{array}{ccc} & & \\ & \mathbf{V} & \\ & & \end{array}\right)\left(\begin{array}{ccc} & & \\ & \mathrm{diag}(1/w_j) & \\ & & \end{array}\right)\left(\begin{array}{ccc} & & \\ & \mathbf{U}^T & \\ & & \end{array}\right)\left(\begin{array}{c} \\ \\ \mathbf{b} \\ \\ \\ \end{array}\right)$$

# Literaturverzeichnis


B. Angeniol, G. Vaubois, und J. Texier. Self-organizing feature maps and the traveling salesman problem. *Neural Networks*, 1(4):289–293, 1988.

P. Baldi und K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

H.-U. Bauer und K. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Transactions on Neural Networks*, 3(4): 570–579, 1992.

H.-U. Bauer und T. Villmann. Growing a hypercubical output space in a self-organizing feature map. TR-95-030, International Computer Science Institute, Berkeley, 1995.

R. Berlich, M. Kunze, und J. Steffens. A Comparison between the Performance of Feed Forward Neural Networks and the Supervised Growing Neural Gas Algorithm. In *5th Artificial Intelligence in High Energy Physics workshop*, Lausanne, 1996. World Scientific.

M. R. Berthold und J. Diamond. Boosting the performance of RBF networks with dynamic decay adjustment. In G. Tesauro, D. S. Touretzky, und T. K. Leen, editors, *Advances in Neural Information Processing Systems*, Band 7, Seiten 521–528, Cambridge, MA, 1995. MIT Press.

C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

C. M. Bishop, n. Markus Svens, und C. K. I. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998.

C.-A. Bohn. Efficiently representing the radiosity kernel through learning. In X. Pueyo und P. Schröder, editors, *Rendering Techniques '96*, Seiten 123–132, Wien, Austria, 1996. Springer-Verlag Wien. Proceedings of the 7th EUROGRAPHICS workshop on rendering in Porto, Portugal, June 17-19, 1996.

J. Bruske und G. Sommer. Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7(4):845–865, 1995.

Y. L. Cun, J. S. Denker, und S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, Seiten 598–605. Morgan Kaufmann Publishers, San Mateo, 1990.

C. Darken und J. Moody. Fast adaptive k-means clustering: Some empirical results. In *Proc. IJCNN*, Band II, Seiten 233–238. IEEE Neural Networks Council, 1990.

R. O. Duda und P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

E. Erwin, K. Obermayer, und K. Schulten. Self-organizing maps: ordering, convegence properties and energy functions. *Biological Cybernetics*, 67:47–55, 1992.

F. Favata und R. Walker. A study of the application of kohonen-type neural networks to the travelling salesman problem. *Biological Cybernetics*, 64:463–468, 1991.

E. Fix und J. L. Hodges. discriminatory analysis—nonparametric discrimination: consistency properties. Report 4, US Air Force School of Aviation Medicine, 1951.

B. Fritzke. Let it grow – self-organizing feature maps with problem dependent cell structure. In T. Kohonen, K. Mäkisara, O. Simula, und J. Kangas, editors, *Artificial Neural Networks*, Seiten 403–408. North-Holland, Amsterdam, 1991.

B. Fritzke. *Wachsende Zellstrukturen – ein selbstorganisierendes neuronales Netzwerk*. Dissertation, Universität Erlangen-Nürnberg, 1992.

B. Fritzke. Growing cell structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994a.

B. Fritzke. Fast learning with incremental RBF networks. *Neural Processing Letters*, 1(1):2–5, 1994b.

B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, und T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, Seiten 625–632. MIT Press, Cambridge MA, 1995a.

B. Fritzke. Growing grid – a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5):9–13, 1995b.

B. Fritzke. Automatic construction of radial basis function networks with the growing neural gas model and its relevance for fuzzy logic. In *Applied Computing 1996: Proceedings of the 1996 ACM Symposium on applied computing*, Seiten 624–627, Philadelphia, 1996. ACM.

B. Fritzke. A self-organizing network that can follow non-stationary distributions. In *ICANN'97: International Conference on Artificial Neural Networks*, Seiten 613–618. Springer, 1997a.

B. Fritzke. The LBG-U method for vector quantization – an improvement over LBG inspired from neural networks. *Neural Processing Letters*, 5(1):35–45, 1997b.

B. Fritzke und P. Wilke. FLEXMAP – A neural network with linear time and space complexity for the traveling salesman problem. In *Proc. of IJCNN-91*, Seiten 929–934, Singapore, 1991.

S. Geman, E. Bienenstock, und R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

G. J. Goodhill und T. J. Sejnowski. A unifying objective function for topographic mappings. *Neural Computation*, 9(6):1291–1303, 1997.

R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Press, 1992.

E. J. Hartman, J. D. Keeler, und J. M. Kowalski. Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, 2: 210–215, 1990.

S. Haykin. *Neural Networks – a comprehensive foundation.* MacMillan College Publishing Company, 1994.

K. Hornik, M. Stinchcombe, und H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

A. K. Jain und R. C. Dubes. *Algorithms for clustering data.* Prentice Hall, 1988.

J. P. Jones und L. A. Palmer. The two-dimensional spatial structure of simple receptive fields in cat striate cortex. *Journal of Neurophysiology*, 58:1187–1211, 1987.

S. Kirkpatrick, C. D. G. Jr., , und M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, 1983.

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

T. Kohonen. *Self-organization and associative memory.* Springer, Heidelberg, 1984.

T. Kohonen. The neural phonetic typewriter. *IEEE computer*, 21:11–22, 1988.

T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin, 1989.

A. Krogh und J. A. Hertz. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems 4*, Seiten 950–957, 1992.

M. Kunze. *Über die Anwendung Neuronaler Systeme in der Teilchenphysik.* Habilitationsschrift, Ruhr-Universität Bochum, 1996.

M. Kunze und J. Steffens. Growing Cell Structure and Neural Gas – Incremental Neural Networks. In *4th Artificial Intelligence in High Energy Physics workshop*, Pisa, 1995.

Y. Linde, A. Buzo, und R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communication*, COM-28:84–95, 1980.

H. S. Loos, B. Fritzke, und C. von der Malsburg. Positionsvorhersage von bewegten Objekten in großformatigen Bildsequenzen. In *Accepted for Proceedings des Workshop Dynamische Perzeption, Juni 18–19, Bielefeld, Germany.* Infix Verlag, 1998.

J. MacQueen. On convergence of $k$-means and partitions with minimum average variance. *Ann. Math. Statist.*, 36:1084, 1965. abstract.

E. Maël. A Hierarchical Network for Learning Robust Models of Kinematic Chains. In C. von der Malsburg, W. von Seelen, J. Vorbrüggen, und B. Sendhoff, editors, *ICANN'96: International Conference on Artificial Neural Networks*, Seiten 617–622. Springer, 1996.

C. v. d. Malsburg. The correlation theory of brain function. Internal Report 81-2, MPI Biophysical Chemistry, 1981.

T. M. Martinetz. *Selbstorganisierende neuronale Netzwerkmodelle zur Bewegungssteuerung.* Infix Verlag, 1992.

T. M. Martinetz. Competitive Hebbian learning rule forms perfectly topology preserving maps. In *ICANN'93: International Conference on Artificial Neural Networks*, Seiten 427–434, Amsterdam, 1993. Springer.

T. M. Martinetz, S. G. Berkovich, und K. J. Schulten. Neural-gas network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, 1993.

T. M. Martinetz, H. J. Ritter, und K. J. Schulten. three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Transactions on Neural Networks*, 1(1):131–136, 1990.

T. M. Martinetz und K. J. Schulten. A “neural-gas” network learns topologies. In T. Kohonen, K. Mäkisara, O. Simula, und J. Kangas, editors, *Artificial Neural Networks*, Seiten 397–402. North-Holland, Amsterdam, 1991.

T. M. Martinetz und K. J. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

K. Mehlhorn. *Data structures and algorithms.* Springer Publishing Company, 1984. Vol 1–3.

J. E. Moody und C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

E. A. Nadaraya. On estimating regression. *Theory of probability and its applications*, 9(1):141–142, 1964.

S. M. Omohundro. The Delaunay triangulation and function learning. Tr-90-001, International Computer Science Institute, Berkeley, 1990.

M. Pagel, E. Maël, und C. v. d. Malsburg. Self calibration of the fixation movement of a stereo camera head. *Autonomous Robots*, 5:355–367, 1998.

J. C. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

T. Poggio und F. Girosi. Networks for approximation and learning. In *Proceedings of the IEEE*, Seiten 1481–1497, 1990.

F. P. Preparata und M. I. Shamos. *Computational geometry.* Springer, New York, 1990.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, und B. P. Flannery. *Numerical Recipes in C: The art of scientific computing.* Cambridge University Press, Cambridge, 1992. (second edition).

D. L. Reilly, L. N. Cooper, und C. Elbaum. A neural model for category learning. *Biological Cybernetics*, 45:35–41, 1982.

B. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, 1996.

H. J. Ritter. Parametrized self-organizing maps. In *ICANN'93: International Conference on Artificial Neural Networks*, Seiten 568–575, Amsterdam, 1993. Springer.

H. J. Ritter und T. Kohonen. Self-organizing semantic maps. *Biological Cybernetics*, 61:241–254, 1989.

H. J. Ritter, T. M. Martinetz, und K. J. Schulten. *Neuronale Netze.* Addison-Wesley, München, 1991.

D. E. Rumelhart, G. E. Hinton, und R. J. Williams. Learning internal representations by error propagation. In R. D. E. und J. L. McClelland, editors, *Parallel Distributed Processing*, Band 1, Seiten 318–362. MIT Press, Cambridge, 1986a.

D. E. Rumelhart, G. E. Hinton, und R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986b.

J. W. Sammon, Jr. A non-linear mapping for data structure analysis. *IEEE Transactions on Computers*, 18:401–409, 1969.

B. Schölkopf, C. Burges, und V. Vapnik. Extracting support data for a given task. In U. M. Fayyad und R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining.* AAAI Press, Menlo Park, CA, 1995.

B. Schölkopf. *Support Vector Learning.* R. Oldenbourg Verlag, Munich, 1997.

B. Schölkopf, P. Knirsch, A. Smola, und C. Burges. Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces. In P. Levi, M. Schanz, R.-J. Ahlers, und F. May, editors, *Mustererkennung 1998 — 20. DAGM-Symposium*, Informatik aktuell, Seiten 124 – 132, Berlin, 1998a. Springer.

B. Schölkopf, P. Simard, A. Smola, und V. Vapnik. Prior knowledge in support vector kernels. In M. Jordan, M. Kearns, und S. Solla, editors, *Advances in Neural Information Processing Systems 10*, Seiten 640 – 646, Cambridge, MA, 1998b. MIT Press.

B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, und V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Trans. Sign. Processing*, 45:2758 – 2765, 1997.

C. Shannon. A mathematical theory of communication. *Bell Systems Technical Jornal*, 27:379–423, 623–656, 1948.

T. J. Shepherd und D. S. Broomhead. Nonlinear signal processing using RBF. In *Proceedings of SPIE – The International Society for Optical Engineering*, Band 1348, Seiten 51–61, Bellingham, 1990. Int Soc for Optical Engineering.

A. Smola und B. Schölkopf. On a kernel-based method for pattern recognition, regression, approximation and operator inversion. *Algorithmica*, 22:211 – 231, 1998.

G. O. Stone. An analysis of the delta rule and the learning of statistical associations. In R. D. E. und J. L. McClelland, editors, *Parallel Distributed Processing*, Band 1, Seiten 444–459. MIT Press, Cambridge, 1986.

G. Tesauro. Neurogammon wins computer olympiad. *Neural Computation*, 1: 321–323, 1990.

G. Tesauro. Temporal diffence learning and TD-Gammon. *Communications of the ACM*, 38(3), 1995.

K. Torkkola, J. Kangas, P. Utela, S. Kaski, M. Kokkonen, M. Kurimo, und T. Kohonen. Status report of the finnish phonetic typewriter project. In T. Kohonen, K. Mäkisara, O. Simula, und J. Kangas, editors, *Artificial Neural Networks*, Seiten 771–776. North-Holland, Amsterdam, 1991.

A. Ultsch und W. P. Siemon. Kohonens self-organizing feature maps for exploratory data analysis. In *International Neural Network Conference*, Seiten 305–308, Paris, 1990.

V. Vapnik. *The nature of statistical learning theory.* Springer Verlag, 1995.

V. N. Vapnik und A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16:264–280, 1971.

L. Vercauteren, G. Sieben, und M. Praet. The classification of brain tumours by a topological map. In *International Neural Network Conference*, Seiten 387–391, Paris, 1990.

T. Villmann, R. Der, M. Herrmann, und T. Martinetz. Topology preservation in self-organizing feature maps: exact definition and measurement. *IEEE TNN*, 8(2):256–266, 1997.

J. C. Vorbrüggen, C. v.d. Malsburg, und R. P. Würtz. Transputer system for the recognition of human faces by graph matching. In G. H. R. Eckmiller und G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, Seiten 37–41. Elsevier Science Publishers B.B. (North Holland), 1990.

J. Walter. *Rapid Learning in Robotics*. Infix Verlag, 1996.

B. Widrow und M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, Band 4, Seiten 96–104. IRE, New York, 1960.

D. J. Willshaw und C. von der Malsburg. How patterned neural connections can be set up by self-organization. In *Proceedings of the Royal Society London*, Band B194, Seiten 431–445, 1976.

L. Wiskott. *Labeled Graphs and Dynamic Link Matching for Face Recognition and Scene Analysis*, Band 53 von *Reihe Physik*. Verlag Harri Deutsch, Thun, Frankfurt a. Main, Germany, 1995. PhD thesis.

J. Zavrel. Neural Information Retrieval. Technical report, University of Amsterdam, Dept. of Computational Linguistics, 1995.

# Index