eXamen.press

eXamen.press ist eine Reihe, die Theorie und Praxis aus allen Bereichen der Informatik für die Hochschulausbildung vermittelt.

Jürgen Paetz

Soft Computing in der Bioinformatik

Eine grundlegende Einführung und Übersicht

Mit 43 Abbildungen und 5 Tabellen

Springer

Jürgen Paetz

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

ISSN 1614-5216
ISBN-10 3-540-29886-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-29886-1 Springer Berlin Heidelberg New York

Springer ist ein Unternehmen von Springer Science+Business Media
springer.de

Printed in Germany

Satz: Druckfertige Daten des Autors
Herstellung: LE-TEX, Jelonek, Schmidt & Vöckler GbR, Leipzig
Umschlaggestaltung: KünkelLopka Werbeagentur, Heidelberg
Gedruckt auf säurefreiem Papier 33/3142 YL – 5 4 3 2 1 0

Allen Lesern – für Forschung, Anwendung und Lehre

Vorwort

Dieses Buch behandelt die Grundlagen des in der Forschung und Praxis bedeutenden Gebiets „Soft Computing“. Der Stoff aus den Themengebieten Datenanalyse, Neuronale Netze, Fuzzy-Logik, Maschinelles Lernen, evolutionäre Strategien und naturanaloge Algorithmen ist für eine bis zu vierstündige Vorlesung mit Übungen konzipiert. Teile einer zweistündigen Vorlesung „Soft Computing zur Wissensextraktion“ und einer dreistündigen Vorlesung „Soft Computing für Bioinformatiker“ wurden für dieses Buch überarbeitet und ergänzt. Das Buch enthält auch eine Einführung in die Matlab-Arbeitsumgebung, die für die Übungen verwendet werden kann. Im Anhang findet der Leser die wichtigsten Begriffe der Wahrscheinlichkeitsrechnung. Besonders interessant in diesem Buch ist die Darstellung der Anwendungsmöglichkeiten der Verfahren des Soft Computing innerhalb der Bioinformatik (Kapitel 1, 5, 8, 11, 18, 25, 31) und die Anwendung biologischer Prinzipien im Rahmen des Soft Computing (Kap. 26-30, 32-34), die selbst wiederum auf Probleme der Bioinformatik angewendet werden können. Auch wurden die Chemieinformatik und einige medizinische Anwendungen berücksichtigt. Speziell zu diesen Anwendungsthemen wurden (neben anderen) über 80 aktuelle Originalarbeiten (davon über 60 aus den Jahren ab 2000) eingearbeitet, so dass ein weiterführendes Studium dieser Arbeiten ermöglicht wird. Aufgrund der Detailfülle der Originalarbeiten, sowohl aus informatorischer als auch aus biologischer Sicht, können viele Arbeiten nur in verkürzter Form wiedergegeben werden. Insgesamt wird dem Leser ein Angebot von über 250 Literaturangaben und über 60 Aktivierungselementen gemacht. Dennoch können viele Themen nur im Sinne des Buches einführend und überblicksartig behandelt werden. Dem interessierten Leser sei das ergänzende Studium ausgewählter Originalliteratur daher empfohlen. In jedem Kapitel werden zu Beginn die Lehrziele genannt und am Ende wird der Lehrstoff kurz repetiert. Zum Abschluss sei allen gedankt, die das Buchprojekt in verschiedenster Art und Weise unterstützt haben.

Obertshausen, *Jürgen Paetz*
Dezember 2005

Inhaltsverzeichnis

Kapitel 1

Grundlagen der Datenanalyse – Bioinformatik

1

1

1 Grundlagen der Datenanalyse – Bioinformatik

Zum Auftakt dieses Buches werden wir zuerst gebräuchliche Begriffe und Sachverhalte aus dem Bereich der allgemeinen Datenanalyse wiedergeben. Die Inhalte werden bei der Anwendung von Methoden des Soft Computing in der Bioinformatik immer wieder verwendet werden.

Wie zu Beginn jeden Kapitels werden die Lehrziele aufgelistet. Am Ende des Kapitels wird der Stoff nocheinmal zusammengefasst.

Lehrziele:
- Kennenlernen der allgemeinen Grundbegriffe der Datenanalyse wie Datensatz und Attribut,
- unterscheiden können zwischen symbolischen, numerischen und strukturellen Daten,
- Problemstellungen der Bio- und Chemieinformatik nennen können,
- den Begriff „Bioinformatik“ für unsere Zwecke einordnen können,
- wichtige Literaturquellen kennen,
- die Idee der Clusterung im Vergleich zur Klassifikation verstanden haben,
- die Problemstellung der Merkmalswahl kennen,
- den Sinn und Zweck von Regeln zur Wissensgenerierung angeben können,
- den Zusammenhang zwischen Generalisierung und der Verwendung von Trainings- und Testdaten kennen,
- Wiederholung der grundlegenden Begriffe der Wahrscheinlichkeitsrechnung (mittels Anhang A).

1.1 Datenanalyse

Ausgangspunkt der **Datenanalyse** ist ein **Datensatz** (kurz: **Daten**), d.h. eine Ansammlung von **Datentupeln**. Ein Datensatz sei hier ein Zeilenvektor, der aus Werten, z.B. Messungen, verschiedener Variablen (**Attribute**) besteht. Solch einen Wert nennen wir **Attributausprägung**. Ein Beispiel für einen Datensatz mit Datentupeln ist in der Tabelle 1.1 angegeben. In der Regel werden Datenmengen in **Datenbanken** gespeichert. Sie können aber auch zur Datenanalyse in **Dateien** vorliegen. Bei einer Genexpressionsanalyse stünden in den Zeilen der Tabelle 1.1 Gene und in den Spalten reellwertige Expressionslevel.

Tabelle 1.1. Beispiel für einen Datensatz

	Farbe	Beruf	Temperatur	Beurteilung
Datentupel 1	rot	Metzger	5.2	gut
Datentupel 2	grün	Bioinformatiker	17.4	hervorragend
Datentupel 3	blau	Bäcker	19.6	gut

In der folgenden Auflistung beschreiben wir, welche **Datenarten** existieren.

- **Symbolische Daten.** Farbe, Beruf, Beurteilung, Nukleotide, DNA-Sequenzen.
- **Numerische und metrische Daten.** Temperatur, Anzahl eines Nukleotids in einer Sequenz; eine Hausnummer ist nur numerisch, da man mit ihr nicht sinnvoll rechnen kann.
- **Strukturelle Daten.** Moleküle, Graphen.

Attributausprägungen des Attributs „Beurteilung“ sind anordbar; diese Datenart bezeichnet man als **ordinal**. Aus strukturellen Daten können durch Umcodierung numerische oder symbolische abgeleitet werden.

Wenn wir einen Datensatz in einer (Bio-)Datenbank gespeichert haben, dann stellt sich Frage nach der Auswertung. Grundsätzliche Auswertungsmethoden sind:

- **Datenbankabfragen.** Z.B. mit SQL oder mit der Programmiersprache BioPython (http://biopython.org/documentation/),
- **Data Mining zur Informationsgewinnung bzw. Datenanalyse.** Auffinden „versteckter“ Sachverhalte,
- **Neuentwicklung einer Methode.** Um einen Sachverhalt zu analysieren, für den es noch keine Auswertungsmethode gibt.

Folgende Probleme der Bioinformatik (oder Chemieinformatik) kann man als Probleme der Datenanalyse auffassen (vgl. auch [DS02]):

- **Sequenzanalyse.** (Optimales) einfaches oder multiples Alignment,
- **Strukturvorhersage.** Sekundär-, Tertiär- oder Quartärstruktur,
- **Evolution.** Phylogenetische Analyse,
- **Genomics.** Genexpression,
- **Proteomics.** Proteinklassifikation und Analyse von Proteininteraktionen,
- **Systembiologie.** Bestimmung metabolischer Netzwerke,

- **Simulation.** Z.B. Ökosysteme oder Pflanzenwachstum,
- **Molekulare Biologie.** Docking von Molekülen,
- **Virtuelles Screening.** Suche nach Molekülen,
- **QSAR.** Quantitative Structure-Activity Relationships.

Literaturhinweise auf Meilensteine der Bioinformatik zwischen 1869 und 1981 findet man in [Tri00], ein Glossar zum Thema „Computational Drug Design" in [WCG+97]. Die wichtigsten abstrahierten Aufgaben der Datenanalyse sind:

- **Clusterung.** Auffinden ähnlicher Datentupel,
- **Klassifikation.** Das richtige Zuordnen eines Datentupels zu einer bekannten Klasse,
- **Regelgenerierung.** Das Extrahieren von Wissen über die Datentupel),
- **Optimierung.** Das Erzielen optimaler Ergebnisse.

Bevor wir unser einleitendes Kapitel fortführen, soll festgelegt werden, was im Rahmen dieses Buches unter **Bioinformatik** (und **Chemieinformatik**) verstanden wird. Unter Bioinformatik (Chemieinformatik) verstehen wir im Rahmen dieses Buches die Weiterentwicklung vorhandener Methoden, die Neuentwicklung von Methoden und deren Anwendung auf biologische (chemische) Problemstellungen oder aber die Nutzbarmachung biologischer, chemischer Prinzipien für algorithmische Prinzipien der Informatik. Insbesondere wollen wir hier nicht nur die Anwendung vorhandener Software oder Skriptsprachen fördern (wie z.B. durch die Anwendung von Matlab im Anhang B), sondern auch die selbstständige Entwicklung von Methoden.

1.2 Literaturhinweise

1.2

An dieser Stelle liste ich einige allgemeine Zeitschriften, Konferenzen sowie Lehrbücher über Bioinformatik und Chemieinformatik auf, die zusammen mit den in den folgenden Kapiteln genannten Literaturreferenzen eine Basis zum Nachschlagen von Datenanalysemethoden- und Anwendungen bildet (ohne Anspruch auf Vollständigkeit). Auch in späteren Kapiteln werden solche Übersichten gegeben.

Auswahl an Zeitschriften (mit Aspekten Bio-/Chemieinformatik):

- Bioinformatics,
- IEEE/ACM Transactions on Computational Biology and Bioinformatics,

- Computational Biology and Chemistry (früher: Computers & Chemistry),
- Computers in Biology and Medicine,
- Applied Bioinformatics,
- Combinatorial Chemistry,
- Combinatorial Chemistry & High Throughput Screening,
- Journal of Chemical Information and Modeling,
- Journal of Computer Aided Molecular Design,
- QSAR and Combinatorial Science.

Wir weisen darauf hin, dass viele Querbezüge zur Medizin existieren, so dass auch in hier nicht genannten medizinischen Zeitschriften Aspekte der Bioinformatik zu finden sind.

Konferenzen:

- Computational Systems Bioinformatics Conference (CSB, z.B. in Stanford, CA, USA im Jahr 2004),
- German Conference on Bioinformatics (GCB, z.B. in Bielefeld im Jahr 2004 oder in Hamburg im Jahr 2005),
- IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB, z.B. in San Diego, CA, USA in den Jahren 2004 und 2005),
- International Conference on Intelligent Systems for Molecular Biology (z.B. in Glasgow, Schottland im Jahr 2004),
- International Conference on Research in Computational Molecular Biology (RECOMB, z.B. in 2004 in San Diego, CA, USA),
- International Symposium on Computational Life Science (CompLife, zum erstenmal im Jahr 2005 in Konstanz),
- International Symposium on Biological and Medical Data Analysis (ISMDA, z.B. in Barcelona, Spanien im Jahr 2004),
- Asia-Pacific Bioinformatics Conference (APBC),
- European Conference on Computational Biology (ECCB),
- International Conference on Systems Biology (ICSB),
- IEEE Symposium on Bioinformatics and Bioengineering (BIBE),
- Structure-Based Drug Design Conference, z.B. in Boston, MA, USA im Jahr 2004.

Lehrbücher und Webseiten:

- Dugas und Schmidt [DS02],
- Gasteiger [GE03a],

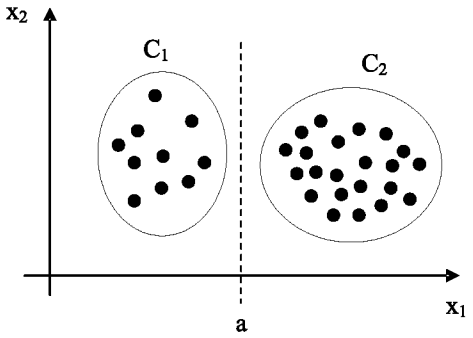

Abbildung 1.1. Zwei Cluster in einer zweidimensionalen Datenmenge

- Hansen [Han01],
- Lesk [Les02],
- Selzer, Marhöfer, Rohwer [SMR03],
- Hofestädt, Bioinformatik 2000, Forschungsführer [HS02],
- Merkl und Waack [MW02],
- Stockhausen [Sto95],
- Cartwright, Sztandera [CS03],
- Steger [Ste03],
- www.bioinformatik.de,
- http://bioinformatics.org,
- www.systembiologie.de,
- www.organic-computing.org.

1.3 Clusterung, Klassifikation und Regelgenerierung

1.3

Wir besprechen zuerst die Clusteraufgabe. Das Zusammenfassen gleichartiger Datentupel nennt man **Clusterung**, vgl. Abb. 1.1. C_1, C_2 heißen **Cluster**.

Bei der Clusteraufgabe in der Abb. 1.1 treten folgende Probleme auf: Bestimmung der Clusterregionen und die **Merkmalswahl** (engl.: **feature selection**) der relevanten Dimensionen.

Zur Lösung der Clusteraufgabe werden die **geometrischen** und **statistischen** Eigenschaften der Daten berücksichtigt. Das Cluster C_1 liegt getrennt vom Cluster C_2 (Trennlinie a) und im Cluster C_2 ist die **Datendichte** höher.

Tabelle 1.2. Beispiel für Datentupel

Temperatur T [°]	Blutdruck B [mmHg]	Zustand Z
38.0	140	krank
39.5	135	krank
36.8	110	gesund
36.9	115	gesund

Eine **Regel**, die ein Cluster beschreibt, wäre dann z.B.:

WENN $x_1 > a$ DANN x gehört zu Cluster C_1.

Das Zuordnen von Attributausprägungen zu einer Klasse nennt man **Klassifikation**. Für ein Beispiel betrachten wir die Tabelle 1.2. Die **Einheiten** sind in eckigen Klammern angegeben. Man beachte, dass wir immer Dezimalpunkte verwenden.

Die Klassifikationsentscheidung soll durch eine **Regel** angegeben werden. Möglichkeiten, die uns sofort einfallen, sind:

a) WENN ($T = 38.0$ UND $B = 140$) ODER ($T = 39.5$ UND $B = 135$) DANN $Z =$ krank.

Analog kann man für „$Z =$ gesund“ eine Regel bilden. Das Problem dieser Regel ist die fehlende **Generalisierung**, d.h. unbekannte, neue Daten werden nicht berücksichtigt. Wir wissen z.B. nicht, was passiert, wenn $T = 38.5$ ist.

b) WENN $T > b_1 := 37.5$ UND $B > b_2 := 120$ DANN $Z =$ krank.

Probleme im Zusammenhang mit Regeln, die gelöst werden müssen, sind: Die Wahl von b_1 und b_2, die Bestimmung der Anzahl der Regeln und die Überwachung des **Fehlers**, den eine Regelmenge verursacht, falls sich die Daten **überlappen**, vgl. Abb. 1.2. Betrachten wir die **Regelmengen** $\mathcal{R}_1$ und $\mathcal{R}_2$:

$\mathcal{R}_1$: WENN $x_1 < a$ DANN $x =$ Kreuz,
WENN $x_1 \geq a$ DANN $x =$ Kreis.

verursacht 3 Fehler.

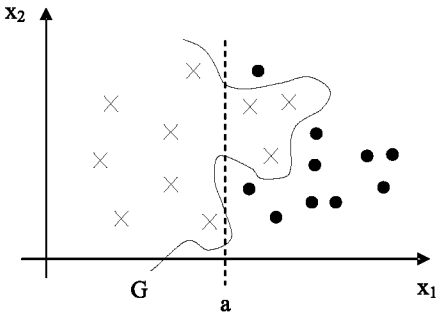

Abbildung 1.2. Überlappende Daten zweier Klassen (Kreis und Kreuz) mit einer möglichen (auch sinnvollen?) **Klassengrenze**

$\mathcal{R}_2$: WENN x_1 links von G DANN x = Kreuz,
WENN x_1 rechts von G DANN x = Kreis.

verursacht keinen Fehler. Allerdings müssen wir bemerken:

a) Die Regelmenge $\mathcal{R}_2$ ist nicht verständlich. Wir fordern daher also die **Verständlichkeit** von Regeln für den Anwender. Aus Regeln soll **Wissen** werden.

b) Die Regelmenge $\mathcal{R}_2$ generalisert möglicherweise nicht gut.

Wir fordern auch, dass die Regeln in einer angemessenen **Laufzeit** erzeugt werden können. Probleme bezüglich der Laufzeit sind:

- viele Datentupel im Datensatz, z.B. 1 Million,
- viele Attribute jedes Datentupels, z.B. 1000 und/oder,
- viele verschiedene Attributausprägungen pro Attribut, z.B. 100 und/oder,
- viele Überlappungen in der Datenmenge. Dann sind verschiedene Datenbereiche nicht gut unterscheidbar.

Wir geben ein Beispiel zur Problematik vieler Attribute. Eine Dimension habe 10 Attributausprägungen a_i. Bei Gleichwahrscheinlichkeit der Attributausprägungen gilt: $P(a_i) = \frac{1}{10} = 0.1$ (P bezeichnet die **Wahrscheinlichkeit**). Bei zwei Dimensionen mit jeweils 10 Attributausprägungen a_i bzw. b_j sind bereits $10 \cdot 10 = 100$ **Kombinationen** möglich. Ein Tupel (a_i, b_j) hat bei Gleichwahrscheinlichkeit nur eine Auftrittswahrscheinlichkeit von $P((a_i, b_j)) = \frac{1}{100} = 0.01$, usw. für 3, 4, 5, ... Dimensionen. Grob gesagt:

„Bei vielen Dimensionen wird der Raum leerer“. Die Konsequenz daraus ist, dass bei vielen Dimensionen mehr Daten zur Datenanalyse benötigt werden (exponentielles Wachstum). Dieser Sachverhalt wurde 1958 von Bellmann unter dem Begriff „Fluch der Dimensionen“ formuliert.

Kommen wir zurück zur Klassifikation (mit integrierter Regelgenerierung). Eine Regelmenge zur Klassifikation, ein sog. **Klassifikator**, soll auch unbekannte, neue Daten einordnen können. Um aber bereits vorab (also ohne neue Daten) zu testen wie gut unbekannte Daten eingeordnet werden, führt man die Cluster-, Klassifikations- bzw. Regelerzeugungsverfahren auf **Trainingsdaten** durch, das sind z.B. 50% aller Daten, und testet die Güte des Ergebnisses (durch Messung des Fehlers) auf den anderen 50% der Daten, den sog. **Testdaten**.

Bei unterschiedlicher, zufälliger Einteilung der Daten in Trainings- und Testdaten kommen unterschiedliche Ergebnisse heraus. Deshalb sind **Versuchswiederholungen** und die Angabe von Mittelwerten und Standardabweichungen der Testergebnisse sinnvoll.

Man beachte, dass bei der Auswertung sensibler Daten wie aus dem Life Science-Bereich die **Ethik** und der **Datenschutz** beachtet werden muss.

Aktivierungselement: Schauen Sie in der Literatur oder im Internet nach, was Sie zu der Problematik „Fluch der Dimensionen“ finden.

Aktivierungselement: Welche weiteren Probleme der Datenanalyse (aus der Bioinformatik) kennen Sie außer den im Text genannten noch?

Aktivierungselement: Wiederholen Sie die Grundbegriffe der Wahrscheinlichkeitsrechnung aus dem Anhang A.

Aktivierungselement: Erarbeiten Sie sich bei Interesse begleitend zur Buchlektüre Kenntnisse der Matlab-Analyse- und Programmierumgebung im Anhang B.

In diesem ersten Kapitel wurde ein allgemeiner Grundstein für die Datenanalyse gelegt. Im Verlauf des Buches werden wir viele Problemstellungen aus der Bioinformatik kennenlernen, denen eine Clusterung, Klassifikation

oder Regelgenerierung als Kernaufgabe zugrunde liegt. Zum Einstieg wurde eine Auswahl an wichtigen Literaturquellen genannt, in denen man auch Soft Computing-Methoden in der Bioinformatik finden kann. Im Idealfall kannten wir bereits das wichtigste Handwerkszeug aus der Wahrscheinlichkeitsrechnung oder haben es uns mit Hilfe des Anhangs A erarbeitet, um für das Soft Computing von dieser Seite aus gerüstet zu sein. Bei vielen Anwendungen, z.B. zur Strukturvorhersage mit neuronalen Netzen, verwendet man eine Aufteilung der Daten in Trainings- und Testdaten, um eine Generalisierung der getroffenen Aussagen zu erreichen. Erzielt man auf den Testdaten deutlich andere Ergebnisse als auf den Trainingsdaten, so ist dem Ergebnis zu misstrauen.

Kapitel 2

2

Übersicht und Einordnung

2

2 Übersicht und Einordnung

Wir erklären als Erstes, wie man auf einfache Weise clustern, klassifizieren oder Regeln generieren kann. Wir erläutern die Begriffe **Intelligente Datenanalyse** und **Soft Computing**. Anschließend geben wir einen ersten Überblick über die großen Themen dieses Buches. Vorab formulieren wir die Lernziele des zweiten Kapitels.

Lehrziele:
- Die Idee des linkage-Clusterverfahrens nachvollziehen können,
- den Algorithmus des k-means-Clusterverfahrens angeben können,
- die Idee des k-nearest-neighbour-Verfahrens verstehen,
- verstehen, dass statistische Kennzahlen für Daten wie der Mittelwert Schätzungen sind, die ungenau sein können,
- die wichtigen Arbeitsgebiete Statistik, Intelligente Datenanalyse, Data Mining, Soft Computing und Künstliche Intelligenz einordnen können,
- sich einen zeitlichen Überblick über die Entwicklung neuronaler Netze zu schaffen,
- die grundlegende Idee der Fuzzy-Logik verstehen,
- den Begriff des Maschinellen Lernens merken,
- die grundlegende Entwicklung der evolutionären Algorithmen nachvollziehen können,
- den Hintergrund von naturanalogen Algorithmen verstehen.

2.1 Statistische und intelligente Datenanalyse – Soft Computing

Die klassische Vorgehensweise in der Datenanalyse ist der Einsatz **statistischer** Verfahren, z.B. Standardverfahren aus Statistikprogrammen wie z.B. SPSS, SAS oder S-Plus. Zur Clusterung können die folgenden Verfahren angewendet werden.

Hierarchische Clusterverfahren (**linkage-Verfahren**) basieren auf Abstandsberechnungen zwischen den Datentupeln. Nahe beieinanderliegende Daten werden zu einem Cluster zusammengefasst und in einem Baum, einem sog. **Dendrogramm**, dargestellt. Eine größere Höhe im Baum bedeutet einen höheren Abstand, vgl. die Abb. 2.1. In diesem Beispiel können die Elemente a und b sowie c und d zu Clustern zusammengefasst werden, da ihre Abstände im Datenraum gering waren. Die Abstände zwischen a, b und

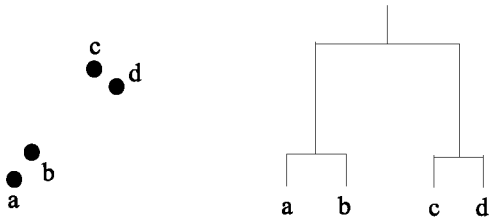

Abbildung 2.1. Vier Elemente im Datenraum und im Dendrogramm

c, d sind größer, so dass man nicht alle Elemente in einem Cluster zusammenfasst. Was zu einem Cluster gehört bleibt aber zum Teil eine Frage der Interpretation (Subjektivität der Clusterung). Geclustert werden können z.B. Microarray-Daten aus der Bioinformatik.

Beim **k-means-Verfahren** wird k als Anzahl der Cluster vorgegeben. Das Verfahren kann nicht die Anzahl der Cluster adaptieren. Jedes Datentupel wird je nach Abstand zu einem der k Cluster zugeordnet. Während der Anwendung des Verfahrens kann sich die Zuordnung eines Datentupels zu einem Cluster ändern.

2.1 Algorithmus 2.1 (k-means-Verfahren, wiedergegeben nach [Bac96])

1) Berechne oder wähle k initiale Clusterzentren c_j.

2) Ordne ein Sample x demjenigen Clusterzentrum c_j zu, zu dem die quadrierte euklidische Distanz minimal ist, d.h. $j = \min_{1,\ldots,k} d(x, c_j)^2$.

3) Neuberechnung der Clusterzentren (i-te Dimension): $c_j^{(i)} = \frac{\sum_g x_g^{(i)}}{\sharp\{x_g\}}$, wobei x_g alle dem Clusterzentrum zugeordneten Samples sind.

4) Hat sich im Schritt 2 die Zuordnung der Elemente verändert, so endet der Algorithmus, ansonsten gehe zu Schritt 2.

□

Verschiedene Clusterverfahren können verschiedene Ergebnisse liefern. Deshalb ist es wichtig, dass ein Datenanalyst weiß, ob ein Ergebnis sinnvoll ist oder nicht. Beim k-means-Verfahren ist z.B. ein sinnvolles k für die Anzahl der vermuteten Cluster zu wählen.

Aktivierungselement: Suchen Sie in der Literatur oder im Internet nach Anwendungen hierarchischer Clusterverfahren und des k-means-Verfahrens.

Aktivierungselement: Sehen Sie in der Literatur nach, wie man einen Abstand (mathematisch vornehmer: **Metrik**) formal definieren kann.

Beim **k-nearest-neighbour-Verfahren** zur Klassifikation wird ein Datentupel derjenigen Klasse zugeordnet, für die der Abstand zwischen dem Datentupel und dem Mittelwert aus k Datentupeln der Klasse am geringsten ist.

Statistische Verfahren setzen oft **Modellannahmen** voraus, wobei wir unter einem **Modell** eine Vereinfachung und eine Formalisierung der Wirklichkeit verstehen wollen. Oft wird bei statistischen Verfahren eine Annahme über die **Verteilung** der Daten derart gemacht, dass eine Normalverteilung vorausgesetzt wird. Die Verfahren sind weniger gut geeignet, um mit **nichtlinearen** Klassengrenzen und starken **Überlappungen** umzugehen. Die Verfahren liefern keine Regeln, d.h. sie liefern kein explizites Wissen.

Die statistische Literatur ist sehr umfangreich. Bereits vor über 100 Jahren wurden Methoden vorgeschlagen, die heute noch Verwendung finden. Bedenken Sie beispielsweise, dass Sie bereits beim Ausrechnen eines **arithmetischen Mittelwertes** eine statistische **Schätzung** des formalen **Erwartungswertes** vornehmen. Beispiel: Wenn Sie zweimal würfeln und zufällig zweimal hintereinander eine 6 würfeln, so würden Sie das arithmetische Mittel zu $(6 + 6)/2 = 6$ berechnen. Das wäre eine schlechte Schätzung des Erwartungswertes. Würfeln Sie 100mal, so werden Sie z.B. als Mitttelwert 3.52 herausbekommen, eine viel bessere Schätzung des tatsächlichen Erwartungswertes 3.5. Allgemein gilt: Je mehr Versuchswiederholungen bzw. Daten man zur Verfügung hat, desto besser wird eine Schätzung.

Eine schwierige Frage der Statistik lautet: Wie viele Daten benötigt man, damit die Ergebnisse **konfident** sind, d.h. dass sie mit einer hohen Wahrscheinlichkeit sinnvoll und vertrauenswürdig sind. So darf man z.B. nie leichtfertig statistische Schätzungen vornehmen, wenn die Dosis eines Medikaments von einer statistischen Berechnung abhängt. Bei der Rentenberechnung wurde z.B. die Bevölkerungsentwicklung falsch geschätzt; die Konsequenzen sind bekannt. Beim Challenger-Unglück wurde die Ausfallrate von Bauteilen falsch geschätzt, so dass es zum Absturz kam. Fazit: Datenanalyse ist eine sehr

verantwortungsvolle Tätigkeit und Sie sind oft der einzige Experte für eine bestimmte Problemstellung sind, so dass Ihnen und Ihren Ergebnissen vertraut wird.

In diesem Buch werden wir uns allerdings weniger der Statistik widmen, sondern vielmehr der „Intelligenten Datenanalyse". Zu den Verfahren der **Intelligenten Datenanalyse** (kurz: IDA) gehören diejenigen Datenanalysen, die (fast) ohne Modellannahmen *direkt aus den Daten* Wissen extrahieren können. Das soll bedeuten, dass sich ein Verfahren an die Daten **adaptiert**, oder genauer: Die **Parameter** eines Verfahrens adaptieren sich, z.B. der Radius eines Kreises. Verwandte Begriffe sind:

Data Mining. Hier handelt es sich um Auswertungsverfahren für sehr große Datenbanken. Die Schwierigkeit des Data Mining liegt in der Optimierung der Bearbeitungsgeschwindigkeit der Datenbankanfragen. Kann ein Verfahren sehr große Datenmengen verarbeiten, so heißt es **skalierbar** (engl.: scalable).

Soft Computing. Datenanalyse für überlappende Datenbereiche und ungenaue, unpräzise Daten. Solche Daten liegen in der Praxis oft vor.

Wir betrachten die folgenden Regeln:

I Klassisch) WENN $a = 5$ DANN Klasse X,
II Soft Computing) WENN $a \approx 5$ DANN $P(\text{Klasse } X) = p$.

Das ist einer Hauptunterschiede zur klassischen Logik innerhalb der Künstlichen Intelligenz (Abk.: KI). Selbstverständlich schließen sich die verschiedenen Ansätze nicht aus, sondern eine Kombination kann sinnvoll sein. Allerdings ist zunächst einmal nicht klar, wie man konkret diese ungenauen Angaben im Fall II) erhält. Klar ist aber, dass die Regel unter I) versagt, wenn nur ein einziges $a = 5$ zur Klasse Y gehört. Ein Beispiel für II) wäre: WENN Temperatur = hoch DANN Lawinengefahr = hoch.

Wegen der Komplexität der Problemstellungen arbeiten Verfahren der IDA oft **heuristisch**, d.h. sie finden nur **suboptimale** Lösungen einer evtl. theoretisch vorhandenen oder gar nicht möglichen theoretischen Lösung.

Die modernen Themen der IDA werden z.Z. auf vielen Konferenzen behandelt und es erscheinen sehr viele Fachbeiträge in Zeitschriften wie wir im Folgenden sehen werden. Es ist ein sehr forschungsaktives Gebiet, wenngleich die

Ursprünge z.T. bis zu 60 Jahren zurückliegen. Kommen wir also nun mit unserem Vorwissen über Datenanalyse zu der Vorstellung der großen Themen dieses Buches.

2.2 Neuronale Netze

Ursprünglich wurde versucht, die Gehirnstruktur nachzubilden und die Funktionsweise natürlicher Neuronen (= Gehirnzellen) nachzuahmen, so dass man die intelligenten Leistungen des Menschen verstehen kann. Heute existieren viele Varianten von NN, die abstrahiert Aufgaben der IDA lösen. Es wird aber auch weiterhin Gehirnforschung betrieben. Wir geben eine Kurzübersicht über die Historie von NN:

- **1943.** Erste Formalisierung eines künstlichen Neurons durch McCullough-Pitts,
- **1949.** Hebb'sche Lernregel,
- **1958.** Erstes funktionstüchtige NN (das sog. Perzeptron, Ziffernerkennung) durch Rosenblatt,
- **1969.** Krise: Minsky/Papert weisen Schwächen des Perzeptrons nach, d.h. das Lernverfahren des Perzeptrons selbst einfache Sachverhalte nicht lernen kann, z.B. die XOR-Funktion,
- **1972.** Kohonen's Korrelationsmatrix-Assoziativspeicher,
- **1973.** Konzepte der Selbstorganisation durch von der Malsburg,
- **1974.** Backpropagation durch Werbos' Dissertation,
- **1976.** ART-Netze durch Grossberg,
- **1982.** Hopfield-Netze,
- **1982.** Selbstorganisation und Kohonen-Netze durch Kohonen,
- **1982.** Lernen einer Principal Component Analysis (PCA) durch Oja,
- **1983.** Boltzmann-Maschine durch Kirkpatrick et al.,
- **1983.** Reinforcement-Lernen durch Barto et al.,
- **1986.** Neuentdeckung von Backpropagation durch Rumelhart, Hinton und Williams,
- **1986.** Nettalk von Sejnowski (lernte Aussprache von englischen Wörtern),
- **1988-90.** RBF-Netze durch Broomhead/Lowe, Poggio als allgemeine Funktionsapproximatoren,
- **1992.** Support Vector Machines durch Vapnik et al.,
- **1994.** Erste Formalisierung der Independent Component Analysis durch Comon,
- **1991-95.** Neuro-Fuzzy-Netze von Halgamuge, Jang, Nauck/Kruse, Kosko, Berenji, Huber/Berthold,

- **90er Jahre.** Spikende Neuronennetze werden verstärkt untersucht, z.B. durch Maass, Pratt, Judd, Gerstner,
- **1990-95.** Knowledge-Based Neurocomputing durch Towell/Shavlik, Fu,
- **90er Jahre bis heute.** Viele Varianten und Anwendungen, NN werden auf breiter Basis „hoffähig"; viele Lehrbücher und Dissertationen erscheinen, verschiedene Methoden werden kombiniert (**hybride Systeme**), Anwendungen in der Chemie- und Bioinformatik und in der Medizin (**Life Science** oder **Lebenswissenschaften**).

Das nächste wichtige Thema in unserem Buch ist die Fuzzy-Technologie, die in den letzten Jahren verstärkt mit dem Prinzip der neuronalen Netze verbunden wurde.

2.3

2.3 Fuzzy-Technologie

Die Fuzzy-Theorie wurde 1965 von Lotfi Zadeh („The Berkeley Initiative in Soft Computing") begründet [Zad65]. Weitere wichtige Arbeiten sind [Zad73] über die Analyse komplexer Systeme und Entscheidungsprozesse, [Zad81] über Possibilitätstheorie und „Soft Data Analysis", [Zad96] über „Computing with Words" und [Zad00] über eine Theorie der „Perceptions" basierend auf dem „Computing with Words". Einen historischen Rückblick gibt [Sei05].

Aktivierungselement: Suchen Sie die Homepage von L. Zadeh. Versuchen Sie herauszufinden, wie viele Ehrendoktortitel Zadeh verliehen wurden.

Was ist aber nun eine Fuzzy-Menge? Schauen wir uns dazu kurz die Naive Mengenlehre an und definieren uns eine Menge $M := \{5, 7, 9, 11, 13\}$. Es ist z.B. $5 \in M$ und $7 \in M$, aber $12 \notin M$. Im Rahmen der Naiven Mengenlehre ist ein Element immer in einer Menge enthalten oder nicht, d.h. es gilt $P(a \in M) = 1$ oder $P(a \in M) = 0$.

Zadeh hatte die Idee, auch $P(a \in M) = p$ für $]0, 1[$ zuzulassen, also eine *probabilistische Zugehörigkeit* eines Elements zu einer Menge. Das hat erstaunliche Konsequenzen. Man kann z.B. quantitativ formalisieren, was „Die Temperatur ist hoch" bedeutet. Die „Fuzzy-Waschmaschine" regelt die Trübung der Lauge durch Fuzzy-Technologie.

Die Fuzzy-Theorie wurde bis heute stark ausgebaut. Interessant für uns sind Anwendungen der FT und die Kombination mit NN.

2.4 Maschinelles Lernen

2.4

Es folgt ein Zitat aus dem Buch von Keller [Kel00, S. 17]: „Mit Lernen bezeichnet man adaptive Veränderungen der Fähigkeiten eines Systems, um die gleichen oder ähnliche Aufgaben, die aus der gleichen Population hervorgehen, beim nächsten Mal effizienter und effektiver behandeln zu können (Verbesserung der Performanz)".

Die Ära des Maschinellen Lernens beginnt im Prinzip in den Anfängen der Künstlichen Intelligenz 1956. Zu dieser Zeit wurde auch die Programmiersprache LISP entwickelt. In den 60/70er Jahren wurden die ersten Expertensysteme entwickelt, z.B. DENDRAL (Ermittlung von Strukturformeln chemischer Verbindungen durch Interpretation ihrer Massenspektrogramme) oder MYCIN (diagnostiziert bakterielle Infektionen und gibt Empfehlungen zur medikamentösen Behandlung). Beim Expertensystem wird das Wissen vorgegeben und neues Wissen aus bekanntem Wissen abgeleitet, aber nicht direkt aus den Daten neu erzeugt.

Der Begriff ML ist in unserem Buch ähnlich zu gebrauchen wie der Begriff der IDA. Wir werden die wichtigsten Verfahren des ML kennenlernen, die durch die Gebiete NN, FT und evolutionäre Algorithmen noch nicht abgedeckt werden, insbesondere Verfahren, die symbolische Daten verarbeiten können, aber auch Inferenzmethoden (Schlussfolgerungsmechanismen) der Logik, die aus bekannten Tatsachen Folgerungen erstellen. Auch werden Verfahren vorgestellt, die Regeln aus Daten erzeugen können.

2.5 Evolutionäre Strategien

2.5

Die evolutionären Strategien sind geeignet zum optimalen Lösen von abstrakt formulierten Aufgabenstellungen. Wir fragen uns zuerst, wie man überhaupt eine Aufgabe *optimal* lösen kann ohne gleich an evolutionäre Strategien zu denken. Ein klassischer Algorithmus aus der Optimierungstheorie ist der **Simplex-Algorithmus** von Danzig (1949) aus dem Gebiet der **linearen Programmierung**. Grundsätzlich geht es um die Optimierung einer **Zielfunktion** unter **Nebenbedingungen** (engl.: constraints).

2.2 **Beispiel 2.2** Gegeben sei eine Zielfunktion ZF $f(x) = 15x_1 + 45x_2 = Max!$. Dabei seien 15 und 45 Gewinneinheiten für Produkteinheiten x_1, x_2. Die Nebenbedingungen können sein: $10x_1 + 5x_2 \leq 600, 20x_1 + 10x_2 \leq 800, 10x_2 \leq 300, x_1 \geq 8$. Das Besondere ist, dass das Optimum immer auf einer **Ecke** eines Simplex liegt, was man im zweidimensionalen grafisch darstellen kann, vgl. die Abb. 2.2.

□

Aktivierungselement: Berechnen Sie die Koordinaten der beiden Punkte in der Abb. 2.2.

Mittlerweile gibt es stark verfeinerte Verfahren, z.B. für *ganzzahlige* Probleme, deren Lösungen nur ganze Zahlen sein dürfen ohne Nachkommastellen. Ist das Problem sehr komplex oder sind die Nebenbedingungen nicht linear, so kann man obiges Verfahren nur als erste Näherung verwenden. Fogel et al. haben 1965 die Evolution von endlichen Automaten zur Vorhersage von Zeitreihen verwendet. Schwefel hat 1965 in seiner Diplomarbeit versucht, die Theorie der biologischen **Evolution** auf das schwierige Optimierungsproblem der Optimierung der Windkanaleigenschaften einer Flugzeugdüse unter Variation der Form angewendet. Mit großem Erfolg! Die Hypothese dabei war, dass die Natur im Laufe der Evolution Organismen optimal anpasst. Dieses Prinzip kann man auf abstrakte Optimierungsprobleme übertragen.

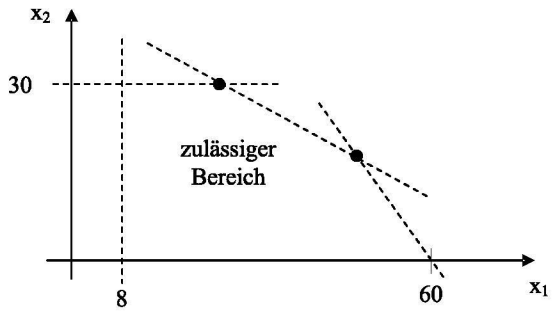

Abbildung 2.2. Beispiel für ein lineares Optimierungsproblem

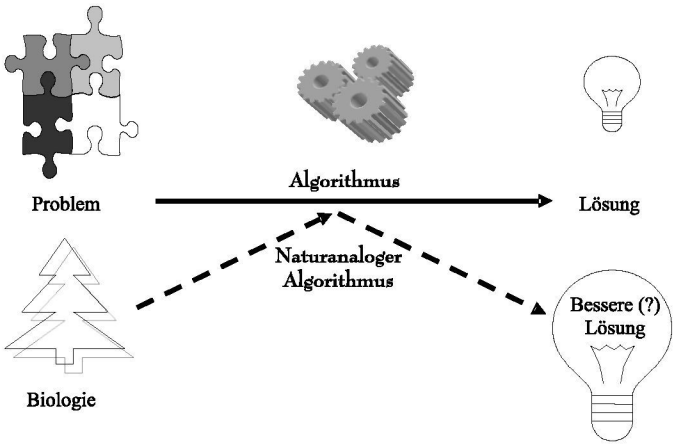

Abbildung 2.3. Sinn und Zweck naturanaloger Algorithmen

Stärker durch die **Genetik** motiviert sind Hollands **genetische Algorithmen** (1969), die aber im Prinzip Ähnliches leisten. Mittlerweile betrachtet man evolutionäre und genetische Strategien gemeinsam und spricht auch vereinheitlichend von evolutionären Strategien (Abk.: ES).

Anwendungen im Rahmen dieses Buches sind das Auffinden optimaler Klassifikatoren, Clusterer oder Regelerzeugungsverfahren. Teilweise werden die Verfahren spezifisch für die Anwendungen aus dem Gebiet der Bioinformatik zugeschnitten. In der Ingenieursdisziplin werden oftmals sog. **Regler** durch ES optimiert, die beispielsweise das Klima eines Gewächshauses regeln können.

2.6 Naturanaloge Algorithmen

2.6

Unter **naturanalogen Algorithmen** (Abk.: NAA) verstehen wir solche, die durch biologische Prinzipien geprägt sind, wie z.B. die evolutionären Algorithmen, die Schwarmalgorithmen (erdacht von Eberhart/Kennedy 1995) aber auch Ameisenkolonie-Algorithmen (eingeführt von Dorigo et al. ab 1991), Künstliche Immunsysteme oder gar Künstliches Leben. Die Abb. 2.3 verdeutlicht den Sinn und Zweck NAA für die Zwecke der Datenanalyse. Zum einen sollen durch die biologischen Grundprinzipien Methoden zur intelligenten Datenanalyse verbessert oder gar erst ermöglicht werden, und zum anderen sollen durch die verbesserten Methoden die Anwendungsprobleme selbst, z.B. innerhalb der Bioinformatik, gelöst werden.

Zusammenfassend kann die Aufgabe eines Bioinformatiker innerhalb eines **interdisziplinären** Teams das Erstellen einer Systemumgebung für die Datenanalyse sein, z.B. die effiziente Verwaltung einer Datenbank oder der Betrieb eines Hochleistungsrechnerclusters, der Entwurf spezieller Datenanalysealgorithmen, die Anwendung von Softwaretools für spezielle Problemstellungen, die Bereitstellung einer „Mensch-Maschine-Schnittstelle", z.B. die Visualisierung der Ergebnisse bzw. das übersichtliche Darstellen von Zahlenmaterial, Bildern, usw., oder auch die rechnergestützte Simulation von Biosystemen.

Wir widmen uns im Folgenden insbesondere den intelligenten Verfahren des Soft Computing unter besonderer Berücksichtigung der Anwendungen aus der Bioinformatik.

Fragen im betriebswirtschaftlichen Sinne des **Wissensmanagements** werden hier nicht behandelt. Es ist aber nicht unerheblich, eine Datenanalyse und den damit verbundenen Personal- und Materialbedarf sorgfältig zu planen und den potenziellen Nutzen der Analysen abzuschätzen. Insbesondere in der Bioinformatik sind die Kostenanforderungen an Material, Daten und Personal hoch.

Aktivierungselement: Recherchieren Sie doch mal was alles notwendig ist, um ein Projekt, sei es in einem Unternehmen oder innerhalb der Forschung, erfolgreich durchzuführen.

Aktivierungselement: Erklären Sie, was eine Klassifikation ist. Was ist eine Clusterung? Was versteht man unter dem Problem der Regelgenerierung?

Aktivierungselement: Erklären Sie kurz, was unter den Gebieten NN, FT, ML, ES und NAA zu verstehen ist.

Wichtige Arbeitsgebiete, die mit dem Soft Computing in Zusammenhang stehen, sind die Statistik, die Intelligente Datenanalyse, das Data Mining und die Künstliche Intelligenz. Aus dem Bereich der Statistik haben wir die Schätzung des Mittelwertes und das linkage-Clusterverfahren kennengelernt. Das k-means-Clustern und das k-nearest-neighbour-Verfahren sind einfache Beispiele für intelligente Datenanalysen. Data Mining beschäftigt sich vor allem mit sehr großen Datenmengen und der dazugehörigen Datenbanktechnologie. Die Künstliche Intelligenz basiert auf logischen Schlussfolgerungen

und kann am ehesten mit dem Maschinellen Lernen in Bezug gesetzt werden. Im Bereich des Soft Computing werden vor allem Daten auswertbar, die ungenau definiert sind (Fuzzy-Logik) oder die nicht exakt ausgewertet werden können, z.B. aufgrund von Überlappungen verschiedenartiger Daten (Neuronale Netze). Viele Ergebnisse zu komplexen Problemstellungen können mit evolutionären Algorithmen optimiert werden. Eine sehr vielversprechende Weiterentwicklung sind hier auch die naturanalogen Algorithmen, die z.B. das Verhalten von Ameisen modellieren.

Beginnen wir nun mit dem ersten Schwerpunkt unseres Buches: Neuronale Netze und deren Anwendungen.

Kapitel 3

Neuronale Netze: Grundlagen

3

3

3 Neuronale Netze: Grundlagen

Nach der Auflistung der Lehrziele dieses Kapitels und einer allgemeinen Literaturübersicht wird erklärt, wie neuronale Netze aufgebaut sind, was ein Neuron als elementare Einheit eines neuronalen Netzes ist, und wir lernen das Perzeptron als ein einfaches neuronales Netz kennen.

Lehrziele:
- Einen Überblick über wichtige Literatur zu neuronalen Netzen bekommen,
- verstehen, was ein künstliches Neuron im Vergleich zu einem biologischen ist und wie es arbeitet,
- die Topologie eines neuronalen Netzes angeben können, insbesondere die eines Feedforward-Netzes,
- die einzelnen Bestandteile eines Neurons angeben können,
- verschiedene Aktivierungsfunktionen kennen,
- die XOR-Funktion als Netz modellieren können,
- die Delta-Lernregel als Formel angeben können,
- überwachtes, unüberwachtes und verstärkendes Lernen voneinander abgrenzen können,
- die Funktionsweise eines Perzeptrons verstehen und zugehörige Probleme nennen können,
- wissen, was unter der dritten Generation von Neuronen gemeint ist,
- die Idee des spikenden Neurons kennen.

3.1 Literaturübersicht

Bevor wir die Frage beantworten, was ein neuronales Netz (Abk.: NN) ist, listen wir an dieser Stelle einige wichtige Konferenzen und Zeitschriften sowie Lehrbücher auf.

Konferenzen:

- International Joint Conference on Neural Networks (IJCNN),
- International Conference on Artificial Neural Networks (ICANN),
- International Conference on Neural Networks (ICNN),
- International Conference on Neural Information Processing (ICONIP),
- European Symposium on Artificial Neural Networks (ESANN),
- Workshop on Self-Organizing Maps (WSOM).

Zeitschriften:

- IEEE Transactions on Neural Networks,
- Neurocomputing,
- Neural Networks,
- Neural Computation,
- Neural Computing & Applications,
- Neural Processing Letters,
- International Journal of Neural Systems,
- Journal of Computational Neuroscience,
- Network: Computation in Neural Systems,
- Neural Computing Surveys,
- Journal of Chemical Information and Computer Sciences (u.a. mit Anwendungen NN in der Chemieinformatik),
- Bioinformatics (u.a. mit Anwendungen NN in der Bioinformatik).

Lehrbücher:

- Zell [Zel94],
- Brause [Bra95],
- Haykin [Hay99],
- Rojas [Roj93],
- Bishop [Bis95],
- Kohonen [Koh97].

Aktivierungselement: Bei weitergehendem Interesse an einzelnen Themen empfehlen wir ergänzende Literatur zur Vertiefung zu konsultieren. Auch können Sie zu vielen Themen mit Hilfe einer Internet-Suchmaschine leicht ergänzende Materialien finden.

Aktivierungselement: Ein NN-Tool, das frei verfügbar ist, ist der sog. *Stuttgarter Neuronale Netze Simulator* (SNNS). Sie finden die Software im Internet. Wenn Sie etwas mehr Zeit haben, können Sie ihn ausprobieren.

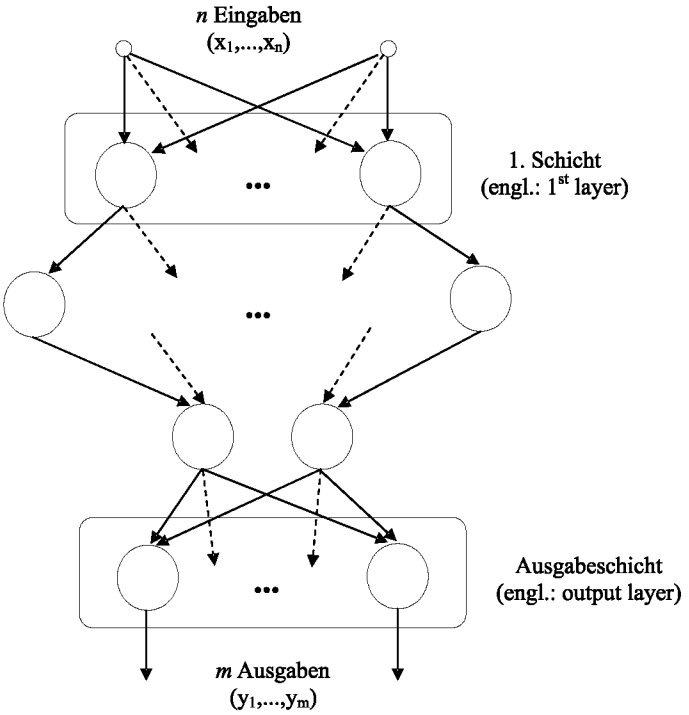

Abbildung 3.1. Ein FF-Netz

3.2 Das Neuronale Netz

3.2

Wir erklären den Aufbau eines neuronalen Netzes. Ein **neuronales Netz** ist aufgebaut aus **Neuronen**, den elementaren Verarbeitungseinheiten des Netzes. Die Struktur, d.h. die Neuronen und deren Verbindungen, werden als **Netztopologie** bezeichnet. Das Verfahren, mit dem die verbundenen Neuronen eine Aufgabenstellung „lernen", heißt **Lernregel**. Wer Tupelnotationen zu schätzen weiß, kann ein NN als einen Tupel $(M, \mathcal{G}, f)$ definieren mit M als die Menge der Neuronen, $\mathcal{G}$ als den Verbindungsgraphen und f der Funktion der Lernregel.

Als Erstes wollen wir sog. **FF-Netze** (Abk. für: feed forward) definieren. Das sind vorwärtsgerichtete Netze bzw. Graphen. Man fordert oft, dass auch Kanten vom Knoten zum selben Knoten verboten sind. Ein solches Netz ist in der Abb. 3.1 wiedergegeben. Die Neuronen sind üblicherweise in **Schichten** organisiert. $N^{(i)}$ sei ein Neuron (oder auch die Menge aller Neuronen) in der i-ten Schicht.

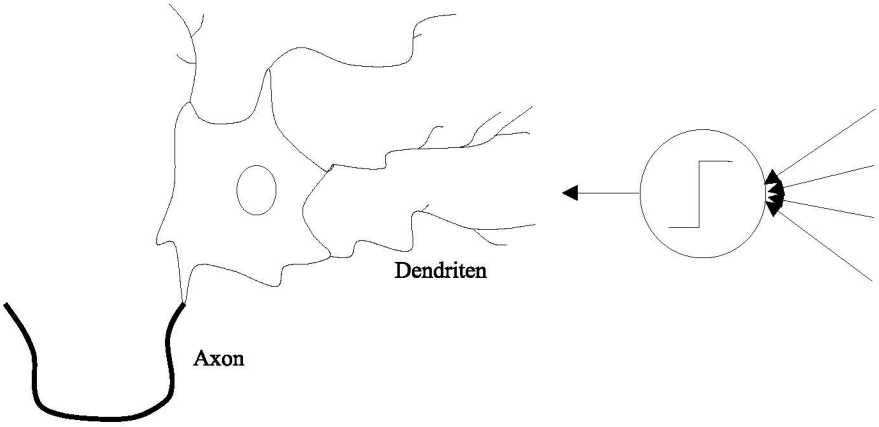

Abbildung 3.2. Links: biologisches Neuron; rechts: künstliches Neuron

Sind alle Neuronen $N^{(i)}$ einer Schicht i mit allen Neuronen $N^{(i+1)}$ einer Schicht $i+1$ verbunden, so heißt das NN **vollständig verbunden**. Eine Verbindung eines Neurons $N^{(i)}$ mit einem Neuron $N^{(j)}$, $j > i+1$, heißt **shortcut**. Netze sind nicht unbedingt vollständig verbunden. Ein NN mit Rückwärtskanten heißt **FB-Netz** (Abk. für: feedback), d.h. es existiert mindestens eine gerichtete Kante von einem Neuron $N^{(i)}$ zu einem Neuron $N^{(j)}$ mit $j < i$.

In der Literatur wird auch oft die Eingabe unten aufgezeichnet und die Ausgabe oben; manchmal ist der Informationsfluss auch seitwärts. Interessanterweise reicht es oft aus, nur zwei Schichten zu verwenden, dazu später mehr.

In der Abb. 3.2 vergleichen wir ein **biologisches Neuron** mit einem **künstlichen** Neuron (engl.: artificial neuron). Das biologische Neuron empfängt seine Signale über die **Dendriten**. Die Reizweiterleitung findet über das **Axon** statt. Der **Zellkern** verarbeitet die ankommenden Reize.

Aktivierungselement: Mehr über biologische Neuronen können Sie z.B. in [Zel94, Kap. 1 bis 4] oder ausführlicher in [CR03, Kap. 48, 49] nachlesen, vgl. aber auch den Abschnitt 3.6.

Anstelle der elektrischen Potenziale in biologischen Neuronen treten in künstlichen Neuronen reelle Zahlenwerte. Die Funktionalität des Nervennetzes kann sicher nur bedingt auf die eines künstlichen neuronalen Netzes übertragen werden. Insbesondere funktioniert ein künstliches neuronales Netz ohne echte chemische Neurotransmitter.

3.3 Das Neuron

3.3

Wir beschäftigen uns nun mit dem Aufbau eines künstlichen Neurons, bevor wir dann die Lernverfahren „Perzeptron-Lernen“ und im nächsten Kapitel „Backpropagation“ kennenlernen werden.

Definition 3.1 (Neuron) 3.1

Schauen wir uns die Bestandteile eines Neurons an:

a) **Aktivierungszustand.** $z_i(t)$ (z.B. $\{+,-\}$, $[0,1]$, $[-1,1]$, $\mathbb{Z}$ oder $\mathbb{R}$)

b) **Aktivierungsfunktion.** $z_i(t+1) := f(z_i(t), v_i(t), Q)$ mit v_i als **Propagierungsfunktion**, meistens als **gewichtete Eingabe**

$$v_i(t) := \sum_j w_{ij} x_j(t), \quad w_{ij} = w_{ij}(t) \tag{1}$$

Die w_{ij} heißen **Gewichte** und können in einer Matrix angeordnet werden. x_i ist die i-te **Netzeingabe**. Q sei eine Menge von Parametern. Ein Beispiel für Parameter sind **Schwellwerte**.

Gebräuchliche Aktivierungsfunktionen sind: $f_1 \equiv \mathrm{id}$, f_2 lineare Funktion mit Sättigung, f_3 binäre Schwellwertfunktion, f_4 Sinus $sin(x)$ mit Sättigung, f_5 logistische Funktion $\frac{1}{1+e^{-x}}$, f_6 Tangens hyperbolicus $\tanh(x)$, vgl. auch [Zel94, S. 77] und die Skizze in der Abb. 3.3.

c) **Ausgabefunktion.** $y_i := g(z_i)$ mit y_i als **Netzausgabe**. Oft wird $g \equiv \mathrm{id}$ gewählt, so dass die Ausgabe gleich der Aktivierung ist.

d) **Lernregel.** Die Lernregel beinhaltet meistens die Modifikation der Gewichte und ist der interessanteste Bestandteil des Neurons.

□

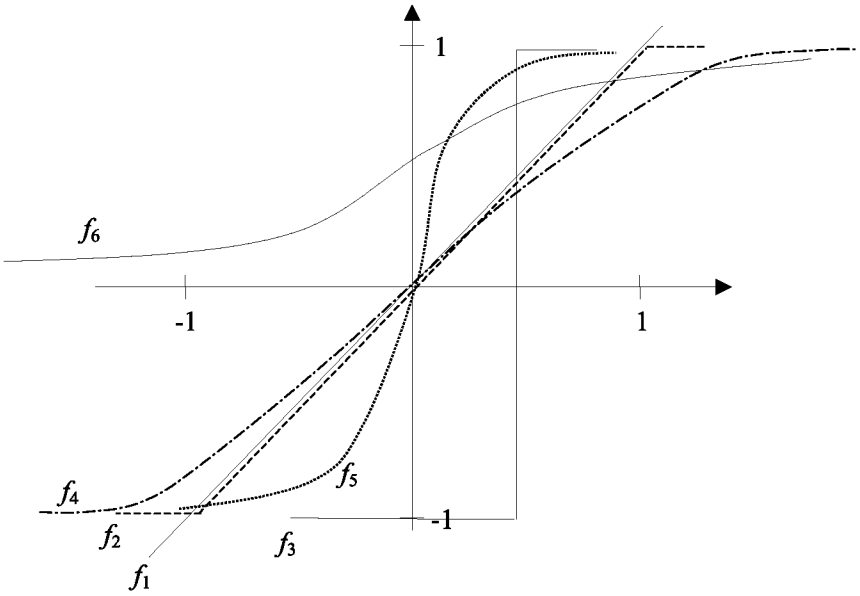

Abbildung 3.3. Aktivierungsfunktionen

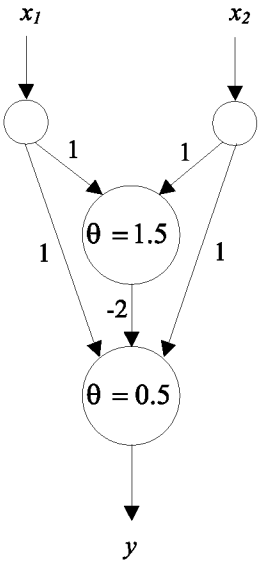

Abbildung 3.4. Ein Netz, das die XOR-Funktion repräsentiert

Beispiel 3.2 (XOR-Funktion, vgl. Abb. 3.4) 3.2
Es ist $g \equiv \mathrm{id}$, $v_i(t) = \sum_j w_{ij} x_j(t)$ und z_i die **Schwellwertfunktion** mit Schwellwert θ:

$$z_i(t) := \begin{cases} 1 & , \ v_i(t) \geq \theta \\ 0 & , \ \text{sonst} \end{cases} \tag{2}$$

□

Aktivierungselement: Überlegen Sie durch Anlegen der Eingaben $(0,1)$, $(1,0)$, $(0,0)$ und $(1,1)$ für (x_1, x_2), dass das Netz tatsächlich die XOR-Funktion repräsentiert.

3.4 Die Lernregel 3.4

Grundsätzlich kann das **Lernen** daraus bestehen, Neuronen oder Verbindungen einzufügen oder zu löschen, die Parameter oder die Funktionsvorschriften zu verändern oder (sehr häufig) die Gewichte zu verändern. Die **Hebbsche Lernregel** kann man wie folgt formulieren: Erhält die Zelle j eine Eingabe der Zelle i und sind beide Zellen gleichzeitig stark aktiviert, dann erhöhe das Gewicht w_{ij}. In mathematischer Schreibweise erhält man:

$$\Delta w_{ij} = \eta x_i z_j \ , \tag{3}$$

$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$. Dabei bezeichnet η eine konstante Lernrate, z.B. aus $]0,1]$.

Eine Erweiterung der Hebbschen Lernregel ist die **Delta-Regel**. Sei y_j die tatsächliche Ausgabe und $\bar{y}_j$ die erwartete Ausgabe („teacher"), dann können wir die Delta-Lernregel formulieren als

$$\Delta w_{ij} = \eta x_i (\bar{y}_j - y_j) = \eta x_i \delta_j \ . \tag{4}$$

Später werden wir sehen, dass die Backpropagation-Lernregel ein Spezialfall der Delta-Lernregel ist.

Aktivierungselement: Ist die Aktivierungsfunktion f linear, so benötigt man nicht mehrere Neuronenschichten. Warum?

Aktivierungselement: Leiten Sie die in der Abb. 3.3 dargestellten Aktivierungsfunktionen ab, d.h. bilden Sie deren erste Ableitungen.

In der folgenden Definition unterscheiden wir drei Arten von Lernverfahren:

3.3 **Definition 3.3 (Lernverfahren)**
a) **Überwachtes Lernen.** (engl.: supervised learning) Die Ausgabe wird vorgegeben, vgl. die Delta-Lernregel. Bei der Klassifikation kann x einen Eingabevektor und k eine Klasseninformation der zugehörigen Ausgabe bezeichnen. Zum überwachten Lernen gehört das Backpropagation-Lernen (Kap. 4) und das Lernen der RBF-Netze (Kap. 6).

b) **Unüberwachtes Lernen.** (engl.: unsupervised learning) Das Netz adaptiert sich selbstständig an die Daten, z.B. bei einer Clusterung. Hierzu gehören die Kohonen-Netze (Kap. 9).

c) **Verstärkendes Lernen.** (engl.: reinforcement learning) Dem Netz wird zu einer Eingabe und einer Ausgabe nur mitgeteilt, ob die Ausgabe richtig oder falsch ist (nur eine qualitative, aber keine quantitative Information); es wird nicht gesagt, wie richtig oder wie falsch die Ausgabe ist.

□

Das Lernen kann im Batch-Modus **offline** stattfinden, d.h. alle Eingaben müssen dem Netz bekannt sein oder aber **online**, d.h. eine Eingabe nach der anderen wird dem Netz präsentiert. Es kann nötig sein, die gesamte Eingabemenge mehrmals, evtl. in unterschiedlicher Reihenfolge, zu präsentieren. Je eine Wiederholung der Datenpräsentation nennt man eine **Epoche**.

Bei einem Lernen mit NN treten die folgenden Fragestellungen auf:

- Wie formuliert man die Lernaufgabe, z.B. die Eingabe und die Vorgabe für die Ausgabe?
- Wie initialisiert man die Parameter?

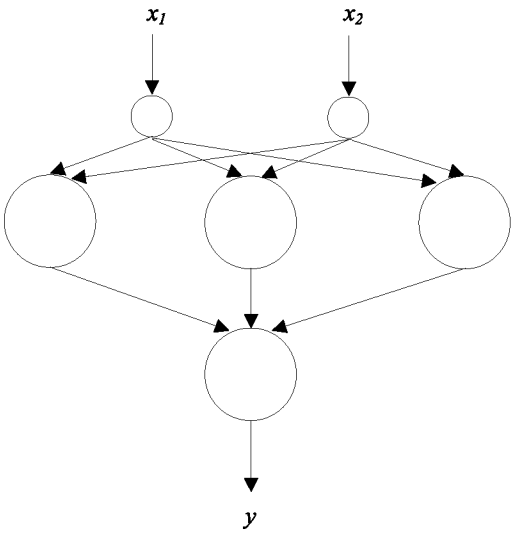

Abbildung 3.5. Netztopologie eines Perzeptrons

- Wie lernt man? Welche Lernregel verwendet man?
- Wie modifiziert man die Gewichte?
- Wie gibt man die Güte des Ergebnisses an?
- Wie belegt man zum Einen die Korrektheit und zum Anderen die Konvergenz des Lernverfahrens?
- Wie viele Schichten an Neuronen und wie viele Neuronen in den Schichten benötigt man?

Wir werden in den folgenden Abschnitten und Kapiteln Lösungen für diese Fragestellungen kennenlernen.

3.5 Das Perzeptron

3.5

In diesem Abschnitt stellen wir das **Perzeptron** (engl.: perceptron) vor, vgl. die Abb. 3.5. Für $v_i(t) = \sum_j w_{ij} x_j$ definieren wir

$$y_i = z_i = \begin{cases} 1 & \text{, falls } v_i \geq \theta \\ 0 & \text{, sonst} \end{cases} . \tag{5}$$

Den **Perzeptron-Lernalgorithmus** kann man mit $w = w_{ij}$, $z = \bar{y}_j$ (teacher) wie folgt formulieren:

WENN berechnete Ausgabe y_j ungleich gewünschter Ausgabe z DANN
WENN $y_j = 1$, $z = 0$ DANN Gewicht $w := w - y_i$; % ($y_i = 0$ oder 1)
WENN $y_j = 0$, $z = 1$ DANN Gewicht $w := w + y_i$; % ($y_i = 0$ oder 1)

Wir wissen noch nicht, was ein Perzeptron lernen kann. Eine Antwort darauf gibt das Perzeptron-Konvergenztheorem.

3.5.1 **Satz 3.5.1 (Perzeptron-Konvergenztheorem, 1962)**
Der Perzeptron-Lernalgorithmus konvergiert in endlich vielen Schritten, falls das Perzeptron eine binäre Funktion *repräsentieren* kann.

□

Nun kann aber ein einstufiges Perzeptron nur **linear separierbare** Mengen repräsentieren, d.h. Mengen, die sich durch **Hyperebenen** trennen lassen. Das Problem, dass sich dadurch ergibt, formuliert der folgende Satz.

3.5.2 **Satz 3.5.2 (Satz von Widner, 1960)**
Es gibt bei wachsender Eingabedimension n prozentual immer weniger binäre, linear separierbare Funktionen unter allen binären Funktionen.

□

Aktivierungselement: Wie viele Funktionen und linear separierbare Funktionen gibt es im Fall $n = 1, 2$? Beispiele für $n = 2$ sind in der Abb. 3.6 wiedergegeben.

Verwendet man zweistufige Perzeptronen, so kann man konvexe Polygone durch Annäherung einer Menge durch Polygonschnitte repräsentieren. Mit dreistufigen Perzeptronen kann man die Mengendifferenzen beliebiger konvexer Polygone, also beliebige Mengen, approximieren.

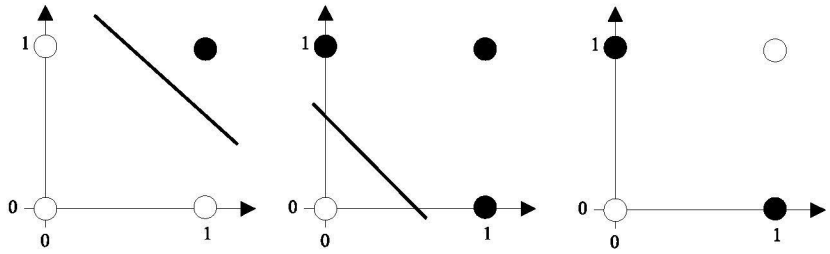

Abbildung 3.6. Linear separierbare Mengen AND und OR, nicht linear separierbare Menge XOR

Aktivierungselement: Machen Sie sich den grundsätzlichen Aufbau eines NN noch einmal klar (Netztopologie, Neuron, Lernregel).

3.6 Die dritte Generation

3.6

Die ersten Neuronen funktionierten mit *binären* Eingaben, so z.B. das erste Neuron, das McCullough-Pitts-Neuron, von 1943. Netzwerke, die auf solchen Neuronen basieren, bezeichnet man als Netzwerke der ersten Generation. Netzwerke der zweiten Generation verwenden sigmoidale Aktivierungsfunktionen und lassen reelle Eingaben zu. Dazu zählt man die für Backpropagation verwendeten Neuronen. Es wird aber implizit angenommen, dass immer die aktiven Eingaben synchron auftreten, damit ein Neuron aktiv wird. Diese Annahme ist biologisch nicht plausibel. Vielmehr muss man nach neueren Erkenntnissen annehmen, dass auch die zeitliche Dimension bei biologischen Neuronen eine Rolle spielt. An dieser Stelle nennen wir einige Eigenschaften biologischer Neuronen und kommen dann zu den sog. *spikenden Neuronen* (engl.: spiking neurons), die als die dritte Generation von Neuronen angesehen wird.

- $1cm^3$ des menschlichen Gehirns enthält 50 Millionen Nervenzellen und insgesamt ca. 10^{10} Nervenzellen.
- Synaptische Enden („Terminalien“) geben chemische Botenstoffe ab.
- Die Übertragung erfolgt von präsynaptischen Zellen auf postsynaptische.
- *Ganglien* sind eine Ansammlung von Nervenzellkörpern mit ähnlicher Funktion.

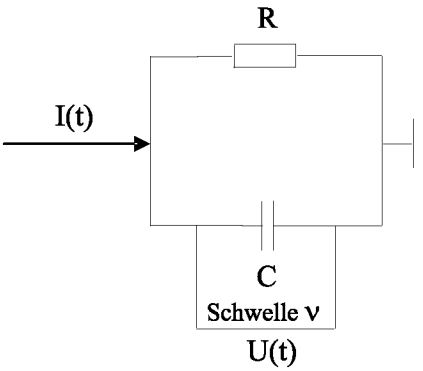

Abbildung 3.7. Modell eines Integrate-and-Fire-Neurons. R = Widerstand, C = Kapazität, I = Strom, U = Spannung

- Ein Nervensignal entsteht durch elektrische Spannungsänderungen über der Plasmamembran der Nervenzellen.
- Lebende Zellen haben ein Membranpotenzial („Ladungsdifferenz"), verursacht durch unterschiedliche Ionenkonzentrationen.
- Wenn Natriumionen in die Zelle strömen, wird diese positiver und dadurch aktiviert.
- Das Großhirn ist für die Sprache, das Riechen, das Hören, das Sehen, die Motorik und das Schmecken zuständig.
- In der Großhirnrinde sind die Areale ähnlicher Funktionsbereiche benachbart angeordnet, z.B. für die Schulter, den Arm, die Hand und die Finger.
- Unter *Kognition* versteht man den Erkenntnisgewinn durch bewusste Wahrnehmung.
- Menschen haben ein Kurzzeit- und ein Langzeitgedächtnis.
- 1998 wurde entdeckt, dass im adulten Gehirn Nervenzellen nachwachsen.
- Die zeitliche Dimension spielt bei der Kodierung von Informationen im Gehirn eine Rolle.

Die letztgenannte Eigenschaft ist interessant, da die Verwendung der zeitlichen Dimension hilft, Neuronen einzusparen. Zeitkodierende Neuronen, sog. **spikende Neuronen** [Bis98], [Maa97] können Aktivierungen asynchron über die Zeit aufsummieren. Es gibt Lernaufgaben, für die deutlich weniger spikende Neuronen als Neuronen der zweiten Generation benötigt werden. Netze aus spikenden Neuronen benötigen zur Erfüllung ihrer Aufgabenstellung im Gehirn weniger Platz.

Ein einfaches technisches Modell solch eines spikenden Neurons ist das **Integrate-and-Fire-Neuron**, das in der Abb. 3.7 zu sehen ist. Durch Stromimpulse über die Zeit wird der Kondensator geladen. Erreicht diese Spannung einen Schwellwert, so feuert das Neuron. Formal beschreiben wir ein Netz aus spikenden Neuronen (Abk.: SNN) wie folgt [Maa97].

Definition 3.4 (Spikendes Neuronales Netz) 3.4

Die folgenden Mengen und Abbildungen werden benötigt: V: Menge der spikenden Neuronen, $E \subset V \times V$: Synapsen, $w_{u,v}$: Gewichte ((u,v) Synapse), $\varepsilon_{u,v} : \mathbb{R}^+ \longrightarrow \mathbb{R}$: Antwortfunktion für alle Synapsen $(u,v) \in E$, $\theta_u : \mathbb{R}^+ \longrightarrow \mathbb{R}^+$: Schwellwertfunktion für jedes Neuron $v \in V$.

Sei $F_u \subset \mathbb{R}^+$ die Menge der Zeiten, zu denen von $u \in V$ gefeuert wird („firing times"). Dann ist das Potenzial von v zur Zeit t gegeben durch:

$$P_v(t) := \sum_{u:(u,v)\in E} \sum_{s\in F_u : s<t} w_{u,v} \cdot \varepsilon_{u,v}(t-s) \tag{6}$$

□

Bevor wir in den nächsten Kapiteln verschiedene Typen neuronaler Netze mit ihren Lernverfahren kennenlernen, haben wir in diesem Kapitel den Grundstein dafür gelegt. Besonders wichtig ist es, den Aufbau eines Neurons verstanden zu haben und zu wissen, wie Neuronen zu neuronalen Netzen verschaltet werden können. Die Delta-Lernregel und das Perzeptron sind die Klassiker zum Thema neuronale Netze und bilden die Grundlage der heute oft verwendeten Lernverfahren wie Backpropagation. Eine andere Art der Datenverarbeitung in Neuronen ist durch die spikenden Neuronen gegeben, die anhand der zeitlichen Abfolge von Außenimpulsen lernen.

Kapitel 4

Backpropagation

4

4

4 Backpropagation

Backpropagation ist ein **Gradientenabstiegsverfahren**, d.h. es verwendet die ersten Ableitungen der Fehlerfunktion $E(Q)$, Q Menge der Parameter, um in Richtung des steilsten Abstiegs ein (lokales) Minima der Fehlerfunktion zu suchen. Wir stellen das Backpropagation-Lernen und seine Eigenschaften vor. Da Backpropagation bereits ein sehr bekanntes und weit verbreitetes Verfahren ist, stellen wir im Rahmen dieses Buches gerade so viele Fakten vor, um zu verstehen, wie es zur Regelgenerierung und für die Anwendungen im Kap. 5 verwendet werden kann.

Lehrziele:
- Die Prinzipien der Fehlerberechnung und des Gradientenabstiegs kennen,
- die Idee der Herleitung der Backpropagation-Lernregel wissen,
- die wichtigsten Probleme des Backpropagation-Lernes verstehen und Verbesserungsvorschläge angeben können,
- insbesondere die Problematik des overfitting, d.h. einer Überanpassung eines Modells, beschreiben können,
- Varianten von Backpropagation kennen,
- die universelle Approximationseigenschaft kennen,
- die Aufgabe des Knowledge-Based Neurocomputing kennen,
- den Unterschied zwischen a-posteriori und in situ Regelgenerierung angeben können,
- das Klassifikations-Regelgenerierungs-Dilemma formulieren können,
- die Idee der dekompositionellen Regelextraktion kennen.

4.1 Das Backpropagation-Lernen

Bezeichne p den Index eines Trainingsmusters und sei y_{pj} die Ausgabe des Neurons j, dann ist $E(Q)$ gegeben durch

$$E(Q) := \sum_p E_p \ , \tag{7}$$

z.B. mit dem **quadratischen Abstand** (engl.: squared error)

$$E_p := \frac{1}{2} \sum_j (y_{pj} - \bar{y}_{pj})^2 \ . \tag{8}$$

Zur Wiederholung definieren wir den Gradienten. Der **Gradient** ist gegeben durch $\nabla f := (D_{x_1} f, \ldots, D_{x_n} f)$; eine andere Schreibweise für $D_x f$ ist $\frac{\partial f}{\partial x}$. Die

Richtung des negativen Gradientenvektors zeigt in die Richtung des steilsten Abstiegs. Daraus resultiert die **Gradienten-Lernregel** (offline-Version)

$$\Delta Q = -\eta \nabla E(Q) \ . \tag{9}$$

Online-Verfahren nach dem Prinzip des Gradientenabstiegs heißen auch **stochastische Gradientenverfahren**.

4.1 **Bemerkung 4.1 (Idee der Herleitung der Backpropagation-Lernregel)** Der Ausgangspunkt ist die Netzeingabe einer Zelle j bei Muster p. Bei mehrstufigen NN (die „mächtiger" sind als einstufige), gibt es keine Sollausgabe (engl.: teacher) $\bar{y}$ für Neuronen der **inneren** Schichten, d.h. für Schichten, die nicht Ausgabeschicht (und nicht die Eingabeschicht) sind. Deshalb wird die Fehlerberechnung von hinten (den Ausgabeneuronen) nach vorne (den Eingabeneuronen) durchgeführt. Das Teilergebnis der vorherigen Schicht wird weiterverwendet. Daraus resultiert der Name **Backpropagation**. Als Hilfsmittel wird dabei die **Kettenregel** verwendet und eine Fallunterscheidung in Ausgabeneuronen und innere Neuronen getroffen. Dabei kann man zur Wiederholung die Kettenregel (etwas salopp) als $(f(g))' = f'(g)g'$ bzw. $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$ formulieren. In [Zel94, S. 107-110] finden Sie eine ausführliche Herleitung der Backpropagation-Lernregel.

□

Aktivierungselement: Leiten Sie die untenstehende Backpropagation-Lernregel her.

Es resultiert die Backpropagation-Lernregel (online), die ein Spezialfall der Delta-Lernregel ist:

$$\Delta_p w_{ij} = \eta y_{pi} \delta_{pj} \tag{10}$$

mit

$$\delta_{pj} = \begin{cases} f'_{\text{act}}(v_{pj})(\bar{y}_{pj} - y_{pj}), & \text{falls } j \text{ Ausgabeneuron} \\ f'_{\text{act}}(v_{pj}) \sum_k \delta_{pk} w_{jk}, & \text{falls } j \text{ inneres Neuron} \end{cases} , \tag{11}$$

wobei f_{act} hier die Aktivierungsfunktion bezeichne.

4.2 Probleme beim Backpropagation-Lernen

Wir listen die negativen Eigenschaften des Backpropagation-Lernens zusammen mit Lösungsvorschlägen auf:

a) **Symmetry breaking.** Die Gewichte dürfen am Anfang nicht alle gleich groß gewählt werden. Lösung: Man wähle kleine Zufallswerte mit einem Zufallszahlengenerator.

b) Es kann vorkommen, dass gute Minima verlassen werden. Lösung: Die Wahl der Schrittweite ist entscheidend.

c) Backpropagation kann konvergieren, auch wenn das globale Minimum der Fehlerfläche nicht erreicht ist. Es bleibt in einem lokalen Minimum stecken. Lösung: Erneut ist die Wahl der Schrittweite entscheidend.

d) **Flache Plateaus.** Ist der Gradient lange Zeit fast Null, so kann das Verfahren auf einem solchen flachen Plateau steckenbleiben. Lösung: Einführung eines sog. **Momentum-Terms** $m_t = \alpha \Delta_p w_{ij}(t)$, $\alpha \in [0.2, 1[$, so dass die Lernregel lautet: $\Delta_p w_{ij}(t+1) = \eta y_{pi} \delta_{pj} + m_t$. Der Momentum-Term bewirkt eine Erhöhung der Gewichtsänderung auf flachen Plateaus.

e) **Oszillation.** In steilen Schluchten des „Fehlergebirges“ kann es zur Oszillation der Richtung des Gradienten kommen. Lösung: Es kann ebenfalls ein Momentum-Term verwendet werden.

f) **Flat spot elimination.** Ist die Ableitung fast gleich Null, so lernt das NN sehr langsam. Lösung: Ableitung künstlich auf z.B. 0.1 setzen.

g) **Weight decay.** Viele große Gewichte zerklüften die Fehlerfläche stark, so dass gute Minima schwerer zu finden sind. Lösung: Man lässt weniger große Gewichte zu durch Bestrafung großer Gewichte, d.h. durch überproportionale Bewertung des Fehlers großer Gewichte.

h) **Modellkomplexität.** Backpropagation kann sehr viele innere Neuronen und Verbindungen besitzen. Lösung: Man kann ggf. Verbindungen, die nahe bei Null liegen entfernen, (engl.: **prunen**). Anschließend wird das Netz **nachtrainiert**.

i) **Overfitting.** Trainiert man das NN, so wird der Fehler auf den Trainingsdaten immer kleiner, aber ab einem bestimmten Zeitpunkt wird der Fehler

auf den *Testdaten* nicht mehr kleiner, sondern größer. Lösung: Ab diesem Zeitpunkt spricht man von overfitting, d.h. an diesem Zeitpunkt muss das Lernen abgebrochen werden, da ansonsten eine unerwünschte Interpolation der Trainingsdaten erreicht wird. Manchmal verwendet man statt Trainings- und Testdaten auch Trainings-, **Validierungs-** und Testdaten: Tainingsdaten zum Trainieren, Validierungsdaten zur Fehlerberechnung, um das Lernen rechtzeitig abzubrechen und overfitting zu vermeiden und Testdaten, um die Leistungsfähigkeit des Netzes zu messen.

j) **Vorverarbeitung.** Man hat herausgefunden, dass eine Mittelwertzentrierung der Daten (Abziehen des Mittelwertes der Daten von den Daten je Dimension), eine Varianznormierung (dividieren durch die Varianz je Dimension) und eine Dekorrelierung (z.B. durch eine sog. **Hauptkomponentenanalyse**, engl. Principal Component Analysis, kurz: PCA; das ist das lineare Transformieren der Daten in die Richtungen der Eigenvektoren durch eine Transformationsmatrix) die Leistungsfähigkeit des Netzes verbessert.

k) **Fehlerabschätzung.** Wie robust und genau ist ein Lernresultat? Stellen Sie sich eine Brücke mit maximaler Belastbarkeit von 20 Tonnen vor. Es ist entscheidend, ob es 20 Tonnen ± 0.1 Tonne bedeutet oder 20 Tonnen ± 10 Tonnen. Dies berührt die Thematik der **Konfidenzintervalle**. Lösung: Versuchswiederholungen mit anschließender Berechnung des Mittelwerts und der Standardabweichung geben Aufschluss über Abweichungen.

l) **Lerngeschwindigkeit.** Backpropagation ist durch die o.g. Probleme ein langsames Lernverfahren. Lösung: vgl. den nächsten Abschnitt.

Aktivierungselement: Veranschaulichen Sie sich die Probleme, soweit es möglich ist.

Bei dieser Aufzählung wollen wir es belassen. Wir sehen im nächsten Abschnitt, dass man einige der Probleme mit (mathematischen) Tricks entschärfen kann. Wir sehen auch, dass das Netz positive Eigenschaften hat, so dass sich die Verwendung durchaus lohnt.

4.3 Varianten und Eigenschaften

Um die Performanznachteile von Backpropagation auszugleichen, insbes. die Punkte c) und l) aus dem letzten Abschnitt, wurden viele Varianten entwickelt, z.B.

- **Quickprop.** lehnt sich an Newton-Verfahren an, bei denen die Steigung der Fehlerfunktion sukzessive durch Geraden angenähert wird.
- **Resilient Propagation.** (Abk.: RPROP) RPROP verwendet Ideen von Quickprop und benutzt eine separate Schrittweitensteuerung für jedes Neuron. Es verwendet außerdem ein **Manhattan-Training**, d.h. es spielt nur das Vorzeichen des Fehlersignals eine Rolle, aber nicht dessen genaue Höhe.
- **Newton-Methode.** Verwendung zweiter Ableitungen, was die Berechnung aufwändiger macht. Newton-Methoden stammen aus dem Gebiet der Numerischen Optimierung, siehe z.B. [Hay99, Abschnitt 4.18].
- **Quasi-Newton-Methode.** schätzt zweite Ableitungen ab wie z.B. der Levenberg-Marquardt-Algorithmus (in der Simulationsumgebung Matlab).
- **Konjugierte Gradienten.** Der Gradient zeigt zwar in die Richtung des steilsten Abstiegs, aber bei der Wahl dieser Richtung konvergiert das Netz nicht am schnellsten; durch die Verwendung zweier sog. konjugierter Gradienten kann die Konvergenzgeschwindigkeit erhöht werden, siehe z.B. [Hay99, Abschnitt 4.18]. Zwei Vektoren v und w heißen **konjugiert** durch eine Matrix A, wenn gilt: $v^T Aw = 0$.

Aktivierungselement: Bei Interesse können Sie sich genauer über die eine oder andere Variante informieren.

Wie man sieht hat das Verfahren Backpropagation einige Schwächen, obwohl es in der Praxis häufig eingesetzt wird. Diese Schwächen versucht man oft durch **Versuchswiederholungen** auszugleichen, d.h. durch experimentelles Bestimmen geeigneter Parameter wie die Lernrate, die Art der Aktivierungsfunktion und die Netztopologie.

Wir wollen aber noch die entscheidende positive Eigenschaft kennenlernen, die **universelle Approximationseigenschaft**, die im Wesentlichen besagt, dass eine stetige, nichtkonstante, monotone, beschränkte Funktion durch einen Backpropagation-Algorithmus beliebig genau approximiert werden kann.

Das ist leicht aufgeschrieben, aber nicht leicht zu beweisen. Ähnliche Sätze gibt es auch für andere Netztypen; i.d.R. genügen dafür bereits zwei Schichten, eine innere und eine Ausgabeschicht. Die universelle Approximationseigenschaft hat zur Folge, dass *nichtlineare* Klassifikationsaufgaben gelöst werden können, die aus den verschiedensten Anwendungsgebieten stammen können, z.B. diagnostische Probleme in der Medizin, Klassifikation von Kunden in der Betriebswirtschaft, Klassifikation von Molekülen in der Chemie oder Klassifikation von Sekundärstrukturtypen in der Biologie.

Im folgenden Abschnitt wird deutlich werden, dass diese Art von Lernen Schwächen bei der Interpretation des Ergebnisses, d.h. bei der Regelextraktion, verursacht.

Aktivierungselement: Probieren Sie einen Backpropagation-Algorithmus aus, z.B. mit einem Tool aus dem Internet (SNNS oder andere kleinere Programme) oder mit der Neural Network-Toolbox aus Matlab.

Aktivierungselement: Welche Klassifikationsaufgaben kennen Sie?

4.4

4.4 Regelgenerierung mit Feedforward-Netzen

Wir können nun mit einem trainierten FF-Netz klassifizieren. Für die Testdaten erhalten wir bei der Eingabe von Mustern $x_1, \ldots, x_p, \ldots, x_n$ in das NN die vom Netz berechneten Ausgaben $y_1, \ldots, y_p, \ldots, y_n$.

Dabei tritt das folgende Problem auf: Das NN berechnet zwar eine Ausgabe, aber wir wissen nur wie Eingabe und Ausgabe durch Gewichte verrechnet werden und haben keine **Erklärungskomponente** in Form von „WENN Eingabe = ... DANN Ausgabe = ... "-Regeln. Dies ist auf die **verteilte** Repräsentation des Wissens im NN zurückzuführen, die auf **Parallelprozessoren** die Arbeitsgeschwindigkeit erhöhen kann. Man sagt, das Netz zeige ein **black box-Verhalten**. Es hat etwas gelernt, aber es kann nicht erklären, was es gelernt hat.

Im Gebiet des **Knowledge-Based Neurocomputing** versucht man, aus solchen FF-Netzen Regeln zu erhalten. Dazu gibt es zwei Möglichkeiten:

a) **A-posteriori Regelgenerierung.** Erst wird das NN trainiert, dann werden Regeln extrahiert. Dabei liegt der Schwerpunkt des NN-Trainings auf der eigentlichen Trainingsaufgabe, z.B. der Fehlerminimierung der Klassifikationsaufgabe (also Minimierung des quadratischen Fehlers beim Lernen bzw. Minimierung der fehlklassifizierten Daten).

b) **In situ Regelgenerierung.** Man sucht eine Repräsentation der Neuronen und eine entsprechende Lernregel, die ermöglichen, das NN als Regelmenge direkt zu interpretieren. Das ist die eigentliche Idee des Soft Computing zur Wissensextraktion und führt später auf sog. Neuro-Fuzzy-Systeme.

An dieser Stelle formulieren wir das sog. **Klassifikations-Regelgenerierungs-Dilemma** (kurz: KR-Dilemma): Man kann i.Allg. nicht das optimale Klassifikationsergebnis *und* das optimale Ergebnis einer Regelgenerierung erhalten.

Das KR-Dilemma ist unter Punkt a) klar zu Gunsten der Klassifikation entschieden worden. A-posteriori kann man nur noch so gut Regeln extrahieren, wie es das für die Klassifikation austrainierte NN zulässt. Das KR-Dilemma birgt noch einen Fallstrick: Auch wenn es mehrere Möglichkeiten zur Beurteilung eines Klassifikationsfehlers gibt, kann man dennoch objektiv sagen, wann der Fehler klein und damit (fast) optimal ist. Bezüglich der Regelgenerierung ist zunächst einmal überhaupt nicht klar, was eine optimale Regel oder gar eine optimale Regelmenge ist. Am Anfang des Buches hatten wir aber bereits einige Kriterien für gute Regeln kennengelernt. Später werden wir versuchen, diese Kriterien zu quantifizieren.

Aktivierungselement: Versuchen Sie anhand eines zweidimensionalen Beispiels sich klar zu machen, warum man keine „optimalen“ Regeln finden kann und warum die Regelgenerierung nicht leicht mit einer optimalen Klassifikationsleistung zu verbinden ist.

Kommen wir nun zu den Möglichkeiten, a-posteriori Regeln aus FF-Netzen zu extrahieren. Unterscheiden kann man Punkt a) wiederum in

a1) **Verfahrensabhängige a-posteriori Regelextraktoren.** Also z.B. nur für mit Backpropagation trainierte Netze oder Netze mit spezieller Netztopologie oder speziellen Gewichten und

a2) **Verfahrensunabhängige a-posteriori Regelextraktoren.** Also weitestgehend beliebige FF-Netze, beliebige Lernregel, beliebige Gewichte.

Es existieren Ansätze aus den frühen 90er Jahren, z.B. von Shavlik, Fu oder Ishikawa, sog. **dekompositionelle** Ansätze, bei denen aus den Neuronen Regeln extrahiert und aggregiert werden. Diese wollen wir nicht alle diskutieren. Wir widmen uns direkt dem Punkt a2) und zwar zwei Ansätzen, kurz dem pädagogischen und im nächsten Abschnitt ausführlicher dem dekompositionellen.

Bei der Regelextraktion im **pädagogischen** Ansatz werden dem Netzwerk Samples präsentiert und die Ausgabe notiert. Die innere Struktur des Netzes wird *nicht* analysiert. Die Regeln werden mit hohem Aufwand anhand der Samples und der Netzausgabe gebildet. Im Prinzip muss dafür ein separates Regelgenerierungsverfahren verwendet werden, vgl. [CZ00b, S. 370-372].

4.5 Dekompositionelle Regelextraktion

Wir präsentieren die Ideen der dekompositionellen Regelextraktion nach Tsukimoto [Tsu00]. Wir setzen ein FF- oder FB-Netz mit monotonen, z.B. sigmoiden, Aktivierungsfunktionen voraus, wie es z.B. beim Backpropagation-Lernen üblich ist.

Wir betrachten zuerst nur diskrete Variablen, die o.B.d.A. durch Transformation als Vorverarbeitung jeweils den Wertebereich $\{0, 1\}$ haben. n sei die Anzahl von Variablen. Wir widmen uns hier dem Basis-Algorithmus, der die Neuronen des NN durch **Boolesche Funktionen**, also binärwertigen Funktionen, approximiert.

Ein Neuron im NN sei beschrieben durch $f(x_1, \ldots, x_n)$. Die $(f_i), i = 1, 2, 3, \ldots, 2^n$ seien die Werte der Neuronen. Die Wertebereiche der Neuronen sind $[0, 1]$ wegen der Verwendung sigmoider Aktivierungsfunktionen. $g(x_1, \ldots, x_n)$ stehe für eine Boolesche Funktion, und $(g_i), g_i = 0$ oder $g_i = 1, i = 1, \ldots, 2^n$ seien die Werte der Booleschen Funktion. Es ist beispielsweise $g_1 = g(0, 0)$ für $n = 2$.

Gegeben seien die Elementarattribute a_i, und g_i sei der zum Elementarattribut zugehörige Wert. Boolesche Funktionen sind gegeben durch die folgende ODER-Verknüpfung

$$g(x_1, \ldots, x_n) = \bigvee_{i=1}^{2^n} g_i a_i \quad . \tag{12}$$

Die Elementarattribute a_i sind mit $e(x_j) = x_j$ oder der Negation $\bar{x}_j$ gegeben durch die folgende UND-Verknüpfung

$$a_i = \bigwedge_{j=1}^{n} e(x_j) \quad (i = 1, \ldots, 2^n) \quad . \tag{13}$$

Ist $e(x_j) = x_j$, dann ist der Wert $x_j = 1$; ist $e(x_j) = \bar{x}_j$ dann ist der Wert $x_j = 0$.

Nun sei

$$g_i = \begin{cases} 1, & (f_i \geq 0.5) \\ 0, & (f_i < 0.5) \end{cases} \quad . \tag{14}$$

Die Boolesche Funktion ist mit obigem g_i gegeben durch die ODER-verknüpfte Formel (12).

Beispiel 4.2 **4.2**

Seien folgende Werte (f_i) des NN bei der Eingabe von $(x, y) = xy = 00, 01, 10, 11$ (mit zugehörigen Elementarattributen $\bar{x}\bar{y}, \bar{x}y, x\bar{y}, xy$) gegeben: $0.82, 0.78, 0.14, 0.21$. Die Werte der zugehörigen Booleschen Funktion lauten $g(x, y) = 1, 1, 0, 0$. Man erhält die Darstellung der Booleschen Funktion (UND = „+“):

$$g(x, y) = g(0, 0)\bar{x}\bar{y} + g(0, 1)\bar{x}y + g(1, 0)x\bar{y} + g(1, 1)xy \quad . \tag{15}$$

Dies kann reduziert werden zu

$$\begin{aligned} g(x, y) &= 1\bar{x}\bar{y} + 1\bar{x}y + 0x\bar{y} + 0xy & (16) \\ &= \bar{x}\bar{y} + \bar{x}y & (17) \\ &= \bar{x} \quad . & (18) \end{aligned}$$

Die Regel lautet hier also, dass die Ausgabe die Negation der Eingabe x ist.

□

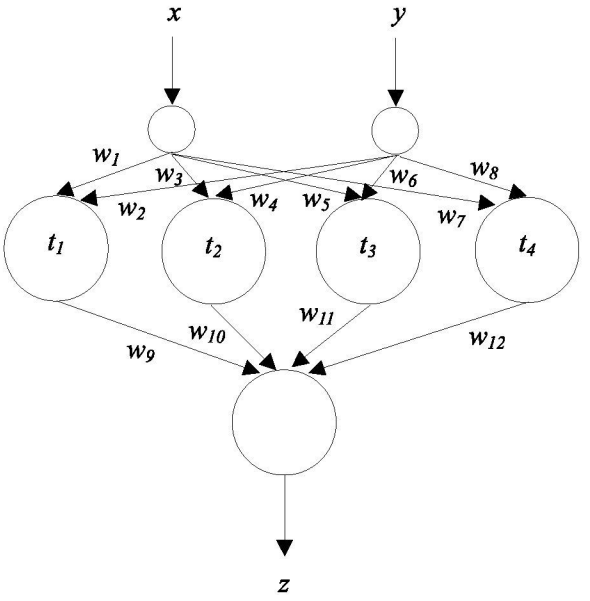

Abbildung 4.1. NN zum Beispiel 4.3

4.3 Beispiel 4.3 (vgl. Abb. 4.1)
Das NN habe die XOR-Funktion gelernt ($x\bar{y} \vee \bar{x}y$). Die w_i seien Gewichte, S eine sigmoide Funktion und die h_i die Bias-Werte. Es ist

$$t_1 = S(w_1 x + w_2 y + h_1) \tag{19}$$
$$t_2 = S(w_3 x + w_4 y + h_2) \tag{20}$$
$$t_3 = S(w_5 x + w_6 y + h_3) \tag{21}$$
$$t_4 = S(w_7 x + w_8 y + h_4) \tag{22}$$
$$z = S(w_g t_1 + w_{10} t_2 + w_{11} t_3 + w_{12} t_4 + h_5)\ . \tag{23}$$

Die Gewichte und Bias-Werte seien nach 1000 Trainingswiederholungen:

$w_1 = 2.51, w_2 = -4.80, w_3 = -4.90, w_4 = 2.83,$
$w_5 = -4.43, w_6 = -4.32, w_7 = -0.70, w_8 = -0.62,$
$w_9 = 5.22, w_{10} = 5.24, w_{11} = -4.88, w_{12} = 0.31$ und
$h_1 = -0.83, h_2 = -1.12, h_3 = 0.93, h_4 = -0.85, h_5 = -2.19.$

Wir berechnen exemplarisch die Werte für $t_1 = S(2.51x - 4.80y - 0.83)$:

$$t_1(1,1) = S(2.51 \cdot 1 - 4.80 \cdot 1 - 0.83) = S(-3.12) \quad (24)$$
$$t_1(1,0) = S(2.51 \cdot 1 - 4.80 \cdot 0 - 0.83) = S(1.68) \quad (25)$$
$$t_1(0,1) = S(2.51 \cdot 0 - 4.80 \cdot 1 - 0.83) = S(-5.63) \quad (26)$$
$$t_1(0,0) = S(2.51 \cdot 0 - 4.80 \cdot 0 - 0.83) = S(-0.83)\ . \quad (27)$$

Es ist $S(-3.12) \approx 0$, $S(1.68) \approx 1$, $S(-5.63) \approx 0$ und $S(-0.83) \approx 0$ und daher $t_1 \approx x\bar{y}$.
Man erhält: $t_2 \approx \bar{x}y$, $t_3 \approx \bar{x}\bar{y}$ und $t_4 \approx 0$. Das ergibt: $z = t_1 t_2 \vee \bar{t}_1 t_2 \bar{t}_3 \vee t_1 \bar{t}_2 \bar{t}_3$. Substituieren wir $t_1 = x\bar{y}$, $t_2 = \bar{x}y$ und $t_3 = \bar{x}\bar{y}$, so erhalten wir $z = x\bar{y} \vee \bar{x}y$, die XOR-Funktion.

□

Aktivierungselement: Führen Sie im Bsp. 4.3 die Rechnungen für einen der anderen drei t_2, t_3, t_4 durch.

Wir geben einen Ausblick auf das weitere Vorgehen zur Regelextraktion. Die Regelgenerierung scheint gut zu klappen. Worin besteht also das eigentliche Problem? Das Problem liegt in der Laufzeit: Sie ist exponentiell. Die eigentliche Idee in [Tsu00] ist die Reduzierung der Laufzeit auf polynomielle Zeit durch ausschließliche Betrachtung von Termen niedrigerer Ordnung. Den Rahmen für die Zeitreduzierung bilden multilineare Funktionenräume. **Multilineare Funktionen** mehrerer Variablen sind linear in jeder Variable, wenn die übrigen Variablen als Konstanten angesehen werden. Auf die Herleitung und Darstellung des Algorithmus wird an dieser Stelle verzichtet, da weiteres Vorwissen über multilineare Funktionen benötigt wird, vgl. aber [Tsu00]. Die Erweiterung auf numerische Wertebereiche geschieht durch die Verwendung *stetiger Boolescher Funktionen*, die ebenfalls eine Boolesche Algebra bilden.

Neben der Regelextraktion existieren auch Ansätze, die Regeln A-priori zur Initialisierung *integrieren*, um Netze nur noch nachtrainieren zu müssen. Man erhofft sich dadurch ein besseres Lernverhalten.

Backpropagation ist eines der meist verwendeten neuronalen Lernverfahren für Feedforward-Netze. Das Verfahren beruht auf der Gradienten-Lernregel und der Anwendung der mathematischen Kettenregel. Mit einem Backpropagation-Netz kann man stetige Funktionen beliebig genau approximieren. Trotz seiner Beliebtheit und der universellen Approximationseigenschaft hat das klassische Backpropagation-Lernen einige Probleme beim Lernen wie langsame Konvergenz und die Tendenz zum Overfitting. Die langsame Konvergenz kann durch Varianten verbessert werden. Das Overfitting wird durch Verwendung von Trainings-, Validierungs- und Testdaten vermieden. Mit Backpropagation-Lernen trainierte Netze sind zunächst Black-Boxen, insofern sie den gelernten Inhalt nicht in Form von Regeln dem Anwender zur Verfügung stellen. Deshalb existieren Ansätze, die aus den gelernten Gewichten Regeln a-posteriori extrahieren können, die annähernd die Struktur des Netzes beschreiben. Grundsätzlich sind die möglichst optimale Klassifikationsleistung und eine möglichst optimale Regelgenerierung zwei nicht unbedingt miteinander vereinbare Lernziele. Eine optimale Klassifikation garantiert nicht unbedingt optimale Regeln und vice versa.

Kapitel 5

Backpropagation-Netze in der Bioinformatik

5

5

5 Backpropagation-Netze in der Bioinformatik

Nachdem wir Backpropagation als Lernverfahren für das Training von FF-Netzen eingeführt haben, gehen wir auf die prinzipiellen Anwendungsmöglichkeiten im Rahmen der Bioinformatik ein.

Lehrziele:
- Das Prinzip der Kodierung von Sequenzen verstehen und Kodierungsmöglichkeiten angeben können,
- die wichtigsten Güteindices nennen können wie beispielsweise die Sensitivität und die Spezifität,
- die Vorhersage der sekundären Proteinstruktur mit neuronalen Netzen beschreiben können,
- die Vorhersage von Spaltungsstellen kennen,
- die Aufgaben des Wirkstoffdesigns beschreiben können,
- die Begriffe Virtual Screening, Enrichment Factor und QSAR definieren können,
- den Sinn der Lipinski-Regeln angeben können,
- wissen, dass chemische Moleküle durch Zeichenketten kodiert werden,
- den Begriff des Deskriptors angeben können,
- ein Beispiel der Anwendung neuronaler Netze im Drug Design kennen,
- weitere Anwendungen nennen können.

5.1 Vorbereitung

Eine grundsätzliche Rolle bei einer Anwendung spielen

- die Merkmalswahl,
- die Datenkodierung und
- die Netzwerk- und Lernalgorithmenauswahl.

Als erstes Beispiel betrachten wir einige Merkmale von Aminosäuren:

- hydrophobe bzw. lipophile Eigenschaft,
- Volumen,
- Masse,
- Ladung,

- Oberflächenbeschaffenheit und
- struktureller Faltungsgrad (z.B. einer α-Helix).

Kodieren kann man diese Eigenschaften zum einen *direkt* in einem binären Merkmalsvektor, z.B. die Nukleotide als 1000 = A, 0100 = T, 0010 = G, 0001 = C und 0000 = „Leerstelle". Die Kodierung „BIN20" für A wäre dann $(1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$. Man kann noch eine Dimension für die Kennzeichnung eines Spacers in Regionen zwischen Proteinen einfügen („BIN21"-Kodierung). Eine andere direkte Kodierungsmöglichkeit ist „BIN4": 00 = A, 01 = T, 10 = G, 11 = C.

Es können auch kontinuierliche, reellwertige Merkmale, z.B. die Ladung, in einem Merkmalsvektor verwendet werden (*indirekte* Kodierung). Diese Merkmale können diskretisiert werden. Reellwertige Merkmale werden oft normalisiert, d.h. der Wertebereich wird in ein Intervall [0,1] oder [-1,1] transformiert. Sind eher globale Eigenschaften, z.B. einer Proteinsequenz, gefragt, so kann man die Häufigkeit der Kombinationen innerhalb eines Fensters mit festgelegter Subsequenzlänge betrachten und in einem Merkmalsvektor zusammenfassen.

Die Ausgabe kann man direkt oder indirekt kodieren. Hat man z.B. die beiden Ausgaben „ja" und „nein", so kann man direkt „ja" als 1 und „nein" als 0 kodieren. Man kann aber auch „ja" als zweidimensionalen Ausgabevektor (1,0) und „nein" als (0,1) kodieren. Auf die Netzwerk- und Lernalgorithmenauswahl sind wir bereits im Rahmen von FF-Netzen eingegangen, so dass wir im nächsten Abschnitt einige Anwendungen von Backpropagation-Lernen aus der Bioinformatik nennen werden.

An dieser Stelle wollen wir noch die üblichen *Güteindices* zur Beurteilung eines Lernverfahrens mit zwei Klassen, z.B. „positiv" und „negativ", vorstellen. Dazu seien:

- TP die Anzahl der korrekt vorhergesagten positiven Beispiele (true positive),
- TN die Anzahl der korrekt vorhergesagten negativen Beispiele (true negative),
- FP die Anzahl der falsch vorhergesagten positiven Beispiele (false positive),
- FN die Anzahl der falsch vorhergesagten negativen Beispiele (false negative).

Mit den vorangegangenen Definitionen können wir nun die Güteindices formulieren.

Definition 5.1 (Güteindices) **5.1**

a) Vorhersagegenauigkeit: (TP+TN) / (TP+TN+FP+FN),
b) Sensitivität: TP / (TP+FN),
c) Spezifität: TN / (TN+FP).

d) ROC/AUC: Trägt man für verschiedene Schwellwerte die Sensitivitätswerte gegen den Spezifitätswerten auf, so liegen diese Werte auf einer Kurve, der sog. *Receiver-Operating Characteristic* (Curve). Den Flächeninhalt $\in [0.5, 1.0]$ unter dieser Kurve bezeichnet man mit AUC (Abk. für: area under curve). Ist er 0.5, so ist kein Lernerfolg eingetreten. Ist er 1.0, so hat man die maximale Klassifikationsgüte erreicht. Liegt er ungewöhnlicherweise unter 0.5, so wäre das Lernverfahren schlechter als der triviale Klassifikator, der immer die Klasse mit der höchsten A-priori-Wahrscheinlichkeit annehmen würde.

e) Korrelationskoeffizient nach Matthews $\in [-1, 1]$:

$$\frac{(\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN})}{\sqrt{(\mathrm{TP} + \mathrm{FP}) \cdot (\mathrm{FP} + \mathrm{TN}) \cdot (\mathrm{TN} + \mathrm{FN}) \cdot (\mathrm{FN} + \mathrm{TP})}} \,. \tag{28}$$

□

Die Spezifität und Sensitivitätsangaben können für $n > 2$ Klassen verallgemeinert werden. Die jeweiligen Angaben über korrekt und falsch klassifizierte Beispiele werden dann in einer $(n \times n)$-Matrix eingetragen.

Aktivierungselement: Es seien TP = 45, FP = 20, TN = 30 und FN = 5 gegeben. Berechnen Sie den Matthews-Korrelationskoeffizienten und bewerten Sie das erhaltene Ergebnis.

Im Folgenden geben wir stellvertretend für viele Anwendungen das grundsätzliche Prinzip der Anwendung von Backpropagation-Lernen wieder. Für weitere, oftmals sehr spezielle Anwendungen verweisen wir auf die genannte Literatur.

5.2 Strukturvorhersage

Ein Klassiker unter den Anwendungen neuronaler Netzwerke ist die *Vorhersage der sekundären Proteinstruktur*. Qian und Sejnowski [QS88] haben eine BIN21-Kodierung verwendet, um mittels Perzeptron- und Backpropagation-Lernen die Klassen „α-Helix" (1,0,0), „β-Blatt" (0,1,0) und „Spirale" (0,0,1) (engl.: helix, sheet, coil) vorherzusagen, oder alternativ die beiden Klassen „Helix" (1,0), „keine Helix" (0,1). Die Proteinsequenz wurde mit Fenstern der Länge 7 bis 17 (=Aminosäurekodierungen) abgetastet, so dass die Eingabedimensionen 7 mal 21 = 147 bis 17 mal 21 = 357 betrugen. Verschiedene Anzahlen von versteckten Neuronen wurden verwendet. Die Vorhersagegenauigkeit betrug für drei Zustände ca. 65%. Die Vorhersagegenauigkeit konnte von Rost und Sander [RS93] auf über 70% gesteigert werden.

5.3 Vorhersage von Spaltungsstellen – Antivirale Medikamente

Um anti-virale Medikamente zu entwickeln ist es notwendig, Wissen über die Spaltung der viralen Polyproteine in ihre funktionalen Sequenzabschnitte zu erhalten. In [NWRY02] wird mit Hilfe von NN versucht, von bekannten Spaltungsstellen (engl.: cleavage sites) auf solche von unbekannten der HIV- und Hepatitis C-Viren (Abk.: HCV) und zu schließen. Letztere verursachen schwere Leberschäden. Virale Protease spaltet dabei die Polyproteine. Ziel ist es durch Medikamente (Inhibitoren) die Aktion der viralen Protease zu hemmen. Erschwert wird dies durch virale Mutationen und durch die notwendige Vermeidung der Veränderung der normalen menschlichen Protease-Funktionen.

Zur Klassifikation von HCV-Sequenzen wurden 168 spaltbare Sequenzen und 752 Gegenbeispiele verwendet. Bei der Klassifikation der HIV-Sequenzen wurden 114 spaltbare Sequenzen und 248 Gegenbeispiele verwendet. Als Ergebnis erhielt man in beiden Fällen im Mittel eine Erkennungsleistung von über 90% (92.50% für HCV und 96.25% für HIV). Die Ergebnisse waren ca. 5-7% besser als die mit Entscheidungsbäumen erzielten. Die Aminosäuren ergeben für HIV 8 mal 20 gleich 160 binär kodierte Eingaben. Da man für HCV fünf Moleküle zusätzlich kodiert hat, ergeben sich für HCV 10 mal 25 gleich 250 Eingaben. Zum Trainieren wurde Backpropagation-Lernen mit Momentum-Term aus SNNS verwendet.

Der trainierte Entscheidungsbaum für die HCV-Sequenzen lieferte die folgende beste Regel, wenn die Stellen 1 bis 6 linksseitig und die Stellen 7 bis 10 rechtsseitig der Spaltung liegen:

WENN Stelle 6 IST Cysteine DANN Spaltungsaktivität (83.1% Konfidenz)

An dieser Stelle wurde das black-box-Verhalten des neuronalen Netzes angeführt, um zusätzlich Regeln aus einem Entscheidungsbaum zu gewinnen. Es ist ein Beispiel für das Regelgenerierungs-Klassifikations-Dilemma; mit dem neuronalen Netz wurde eine sehr gute Klassifikationsleistung erreicht, mit dem Entscheidungsbaum konnten trotz schwächerer Generalisierungsleistung Regeln generiert werden.

5.4 Drug Design

5.4

Neben den Anwendungen in der Biologie können NN auch in der Chemie eingesetzt werden. Insbesondere im Bereich „Drug Design" können mit NN komplexe Klassifikationsaufgaben gelöst werden. Unter **Drug Design** (dt.: Wirkstoffdesign) versteht man alle Schritte und Techniken, die erforderlich sind, um ein wirksames Medikament zu erstellen. Der Ausgangspunkt ist der Raum aller möglichen chemischen Strukturen, also aller Moleküle M. Dieser Raum sei mit $Chem$ bezeichnet und enthält ca. 10^{100} Elemente. Gesucht ist also ein $M \in Chem$ mit einer bestimmten Eigenschaft, Einfluss auf einen Krankheitsprozess zu nehmen, z.B. einen carcinogenen Stoff zu inhibieren. Tut ein Molekül M dies, wird es als **bioaktiv** bezeichnet. Ein Stoff ist zu $x\%$ bioaktiv, wenn er zu x Prozent in die zentrale Zirkulation gelangt.

Im Gegensatz zu chemischen Techniken des High Throughput Screening (Abk.: HTS) ist das Ziel des **Virtual Screening** (dt.: virtuelles Sieben) die schrittweise Reduzierung von $Chem$ auf eine kleine Menge $Lead$ potenzieller Wirkstoffe (engl.: lead candidates) in silico, d.h. innerhalb des Computers. Es ist $Lead \ll Chem$. Neben allgemeinen Filtern werden Ähnlichkeitssuchen und Optimierungsmethoden angewendet. Aber auch Expertenwissen wird modelliert. Der **Enrichment Factor** gibt an, um wieviel höher der gefundene Anteil an gesuchten Strukturen im Vergleich zur A-priori-Wahrscheinlichkeit ist. Zu optimierende Eigenschaften sind z.B. auch die Absorption, die Distribution, der Metabolismus, die Exkretion (ADME), die Toxizität und die Cancerogenität. Mehr zum Thema Wirkstoffdesign findet man im gleichnamigen Werk [BKK96].

Das Anwenden einer Datenanalysemethode zur Erkennung bioaktiver Moleküle mit den gewünschten Eigenschaften in einer Sammlung (engl.: library) *Lib* mit $Lead \ll Lib \ll Chem$ von Molekülen nennt man **QSAR**-Ansatz (Quantitative Structure Activity Relationship). Eine Auflistung von Libraries ist in [SSS03, S. 38] gegeben.

Zum Filtern kann man Experten-Regeln verwenden, z.B. die Lipinski-Regeln („Rule of 5"). Den Filter passieren nicht die Stoffe mit folgenden Eigenschaften

- Clog $P > 5$ (Index für Lipohilie),
- Molekulares Gewicht > 500,
- Anzahl der hydrogenen Donor-Gruppen > 5,
- Anzahl der hydrogenen Akzeptor-Gruppen > 10.

Dieser Filter ist einfach in der Anwendung, aber nicht spezifisch datenbasiert.

Moleküle können durch Zeichenketten repräsentiert werden, z.B. SMARTS- oder SMILES-Zeichenketten. Mit Programmen können Eigenschaften von Molekülen aus diesen Zeichenketten berechnet oder geschätzt werden. Z.B. kann man in SMILES-Notation einen Benzolring als C%1=CC=CC=C%1 notieren.

Aus der Sicht der Datenanalyse gehen wir davon aus, dass die Eigenschaften (sog. **Deskriptoren**) eines Moleküls in einem Deskriptor(-vektor) enthalten sind. Eigenschaften können sein: Molekulargewicht, Energiemessungen, topologische 2D-Abstände, 3D-Abstände eines 3D-Modells (zwischen Atomtypen), Ladungen, vgl. [SSS03, S. 68-69] (QSAR), [BS00, S. 64-68] (Abstände zwischen Deskriptoren). Im Falle von Drug-spezifischen 3D-Modellen spricht man auch von *Pharmakophor-Deskriptoren.* Einen Überblick über Strukturbasiertes Virtual Screening mittels Docking (Einpassen von Molekülen) gibt [Lyn02].

In Anlehnung an die Abb. aus [BS00, S. 119] skizzieren wir in der Abb. 5.1 wie Deskriptoren bei einer Neuronalen Netz-Analyse verwendet werden.

Wir bemerken, dass der Leser die hier in geringem Umfang verwendeten biologischen und chemischen Grundkenntnisse ausführlich in [CR03], [MM03] und [Str96] findet.

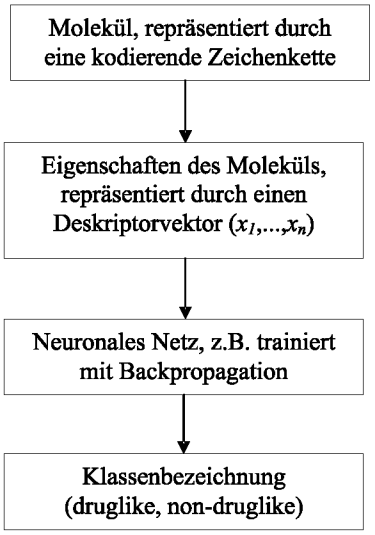

Abbildung 5.1. Prinzip einer Analyse von Deskriptoren mit Neuronalen Netzen

In [MSPGGM+03] wird ein NN verwendet, um 447 Moleküle anhand 62 topologischer Deskriptoren in die Klassen „Drug", „Non-Drug" und „unbestimmt" einzuteilen. Unter den Molekülen finden sich u.a. analgetische, antibakterielle, antidepressive und anti-inflammatorische Wirkstoffe und Non-Drugs. Die topologischen Deskriptoren sind die Anzahlen bestimmter Atome und Gruppen, z.B. von Kohlenstoff und Sauerstoff, die Anzahlen einfacher, doppelter und dreifacher Bindungen, die Anzahlen von Elementen im Molekülgraph mit einer bestimmten Anzahl von Kanten sowie die Abstände von Atomen und Gruppen. Verwendet wurde ein 62-n-1-Netz, $n = 2, 4, 8, 16, 32$ mit Backpropagation-Lernen aus dem SNNS. Es wurden Trainings-, Validierungs- und Testdaten verwendet. Trainiert wurde in 10000 Epochen. Mit zwei versteckten Neuronen wurden 89.58% Moleküle der Trainingsdaten korrekt klassifiziert, aber nur 76.36% der Testdaten.

5.5 Weitere Anwendungen

Weitere Anwendungen von Backpropagation-Netzen im Rahmen der Bio- und Chemieinformatik sind:

- die Analyse von Nukleinsäure-Sequenzen, z.B. durch Klassifikation ihrer Merkmale [KW03].
- die Proteinstrukturvorhersage [BBF+99], [HIH03].
- die Proteinsequenzanalyse [NY03].

- die Vorhersage von E. coli-Promoter-Regionen [HS92]. Promoter-Regionen sind initiale Regionen, an denen RNA-Polymerase oder Proteine binden, die die Transkription steuern.
- die Vorhersage der minimalen Medikamentendosis, um eine Verlängerung der Überlebenszeit von Krebs-infizierten Labormäusen zu erreichen anhand verschiedener Reste im Molekül [ASI90b], [ASI90a].

Einen Überblick über den Einsatz neuronaler Netze in der Molekularen Biologie gibt auch [SW98]. Weitere Anwendungen im Vergleich zu anderen Netzwerktypen findet man auch im Kap. 8 oder in [WM00], [HS92], darunter die folgenden, oft speziell auf die konkreten Lebewesen und Problemstellungen zugeschnittenen:

- die Vorhersage von kodierenden Sequenzen, d.h. von open reading frames (dt.: offene Leseraster) für Prokaryoten und Intron-Exon-Unterscheidung für Eukaryoten,
- die Erkennung von Spleißstellen (engl.: splice sites),
- die Signalpeptiderkennung,
- die Tertiärstrukturerkennung,
- die Klassifikation der Proteinfamilien.

Wir erkennen in den Beispielen die folgenden Gemeinsamkeiten:

- die Identifikation grundsätzlich relevanter Eingaben und Ausgaben, passend zum Anwendungsproblem und
- das Ausprobieren verschiedener Netzwerkarchitekturen.

Die wichtigsten natürlichen Strukturen sind die Sequenz und das Molekül. Um diese Strukturen datenanalytisch auswerten zu können, müssen ihre Eigenschaften kodiert werden. Hierzu kommen eine Fülle von Kodierungsarten in Frage wie die BIN-Kodierung für Sequenzen und die Deskriptorkodierung für Moleküle, die im Rechner als Zeichenketten abgespeichert vorliegen können. Um das Ergebnis einer Analyse bewerten zu können, wurden verschiedene Güteindices eingeführt, die üblicherweise verwendet werden. Eine klassische Anwendung neuronaler Netze ist die Vorhersage der Sekundärstruktur anhand der Sequenz. Eine weitere Anwendung ist die Klassifikation von Sequenzen in spaltbare und nicht spaltbare. Das Gebiet des Drug Design liefert weitere Anwendungsmöglichkeiten neuronaler Netze wie das Virtual Screening oder die Einstufung der Toxizität von Stoffen. Insgesamt ist man einer Klassifikation der Moleküle in „Drug-like“ und „nicht Drug-like“ interessiert. Viele weitere Einsatzmöglichkeiten kommen für überwachte neuronale Netze

in Frage, wie die Vorhersage bestimmter Sequenzabschnitte oder die Klassifikation von Proteinfamilien.

Kapitel 6

RBF-Netze und Varianten

6

6

6 RBF-Netze und Varianten

In diesem Kapitel lernen wir einen anderen Netztyp kennen, die sog. **RBF-Netze**. Dabei steht „RBF" kurz für **Radiale Basisfunktionen**. Der Hauptunterschied zu Backpropagation-Netzen besteht in der Modellierung einer Lernaufgabe. Beim Backpropagation-Lernen hatte die Änderung einer Neuronenaktivität eine *globale* Auswirkung auf die Fehlerfläche. Bei RBF-Netzen haben einzelne Neuronen im Wesentlichen nur eine *lokale* Auswirkung auf den Fehler. Es gibt kein Gradientenlernen wie bei Backpropagation-Netzwerken, so dass auch keine Ableitungen berechnet werden müssen.

Lehrziele:
- Den Einsatz der radialen Basisfunktionen als Aktivierungsfunktion und die Begriffe Zentrum und Radius verstehen,
- die mathematische Grundidee des ursprünglichen RBF-PG-Netzes kennen,
- die Idee der Regularisierung zur Kenntnis nehmen,
- den Unterschied zwischen Interpolation und Approximation kennen und die Idee des generalisierten RBF-Netzes und des RBF-MD-Netzes kennen,
- die Topologie und das Lernverfahren des Probabilistischen Neuronalen Netzes beschreiben können,
- die Idee des RCE-Lernens kennen,
- das Lernverfahren des RBF-DDA-Netzes beschreiben können,
- weitere Varianten von RBF-Netzen kennen wie das Überwachte Wachsende Neuronale Gas oder die Verwendung elliptischer Basisfunktionen,
- einige typische Anwendungsbeispiele außerhalb der Bioinformatik angeben können,
- wissen, dass man die RBF-Netze als Spezialfall der Support-Vektor-Maschinen ansehen kann.

Wir werden zuerst das Original-RBF-Netzwerk nach Poggio und Girosi (1990) kennenlernen, vgl. [PG90b], [PG90a]. Anschließend werden wir die Varianten der RBF-Netzwerke besprechen sowie deren Anwendungen.

6.1 RBF-Netz nach Poggio und Girosi

In der Abb. 6.1 ist die Netztopologie des RBF-Netzes nach Poggio und Girosi (hier kurz: RBF-PG-Netz) dargestellt. Wir haben k Eingaben $x_1, \ldots, x_k$ und *genau so viele* verdeckte Neuronen mit zugehörigen Gewichten $w_1, \ldots, w_k$. Nur eine Ausgabe y_{ij} ist abgebildet. Das Netz kann kanonisch auf beliebig

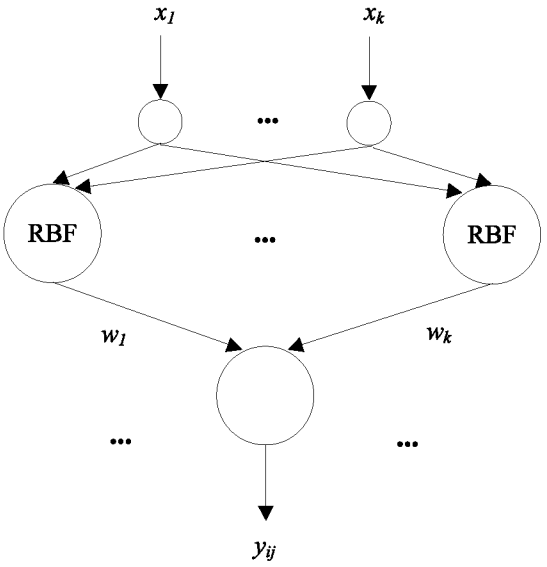

Abbildung 6.1. Netztopolgie eines RBF-PG-Netzes

viele Ausgaben erweitert werden. Die Eingaben sind vollständig mit der verdeckten Schicht verbunden.

Die **radialen Basisfunktionen** seien mit h_i bezeichnet. Sie werden üblicherweise definiert als

$$h_i(x, z) = \exp^{-(\frac{d(x,z)}{\sigma})^2} \quad . \tag{29}$$

In der Formel (29) bezeichnet $d(x, z)$ den Abstand zwischen einem aktuellen Sample x und dem sog. **Zentrum** z einer RBF, $d(x, z) = ||x - z||$. Als Norm wird üblicherweise die euklidische Norm verwendet, die die euklidische Metrik induziert. Bei Gauß-ähnlichen RBF gibt ein Parameter σ den **Radius** an, d.h. die Standardabweichung bei einem bestimmten Funktionswert. In der Formel (29) bedingt ein größeres σ eine größere Standardabweichung.

Die abstrakte mathematische Aufgabe, die durch solch ein Netz gelöst werden kann, ist zunächst einmal die sog. **Interpolation**, d.h. gesucht ist eine Funktion f mit $f(x_i) = y_i$ für alle i (vgl. z.B. [Zel94, S. 226-228]).

Gesucht sind die Gewichte in

$$f(x) = \sum_{i=1}^{k} w_i h_i(||x - x_i||) \ . \tag{30}$$

Aktivierungselement: Wiederholen Sie die Begriffe Norm, Metrik und Gaußsche Normalverteilung.

Aktivierungselement: Skizzieren Sie einige Gauß-Funktionen und deren (summierte) Überlagerung.

Wir erreichen dies durch das Lösen eines linearen Gleichungssystems mit k Gleichungen und k Unbekannten in der Formel (31).

$$\sum_{i=1}^{k} w_i h_i(||x - x_i||) = y_i \ , \ \ j = 1, \ldots, k \tag{31}$$

Seien h_i die Gauß-Funktionen, dann erhält man mit $w = (w_1, \ldots, w_k)$, $y = (y_1, \ldots, y_k)$ und $H =$

$$\begin{pmatrix} h_1(||x_1 - x_1||) \cdots h_1(||x_n - x_1||) \\ \vdots \qquad \ddots \qquad \vdots \\ h_n(||x_n - x_1||) \cdots h_n(||x_n - x_n||) \end{pmatrix}$$

die Gleichung

$$Hw = y \tag{32}$$

Durch (numerische) Inversion ergibt sich $w = H^{-1}y$, vgl. die numerische Literatur [SW92].

6.2 Erste und zweite Variante – Regularisierung und Glättung

6.2

Die erste Variante dieses Vorgehens erfolgt, weil man in der Praxis immer mit *verrauschten* Daten arbeitet. Nach Tikhonov (1977) kann man den Ansatz der **Regularisierung** verwenden, indem man einen nahe bei Null liegenden **Regularisierungsterm** λ einführt und $H + \lambda I$ (I ist die Einheitsmatrix,

engl.: identity matrix) betrachtet. Es ergibt sich die (numerische) Lösung

$$w = (H + \lambda I)^{-1} y \tag{33}$$

Um die Interpolationsfunktion zu **glätten**, kann man für f zusätzliche Polynome p_i verwenden:

$$f(x) = \sum_{i=1}^{k} w_i h_i(||x - x_i||) + \sum_{i=1}^{k} d_i p_i(x) \tag{34}$$

Diesen Ansatz wollen wir hier aber nicht weiter ausführen.

6.3

6.3 Dritte und vierte Variante – Generalisiertes RBF-Netz und RBF-MD-Netz

Wichtiger ist, dass oft in der Praxis eine exakte Interpolation nicht sinnvoll ist, sondern dass man eigentlich eine **Approximation** durchführen möchte. Gesucht ist eine glatte Approximationsfunktion f mit $\sum_{i=1}^{m<k}(f(x_i) - y_i) < \varepsilon$. Hierzu verwendet man nur $m < k$ Basisfunktionen, da k Eingaben k Gewichte bedingen und dies auf große Gleichungssysteme führt.

Wir wählen also die RBF-Neuronenzahl in der versteckten Schicht fest $< k$. Es ergibt sich analog

$$Hw = y \ . \tag{35}$$

Allerdings ist nun für $k > m$ das Gleichungssystem überbestimmt und kann nicht exakt gelöst werden. Deshalb verwendet man die *Pseudoinverse* $H^+ := (H^T H)^{-1} H^T$ und setzt

$$w = H^+ y \ . \tag{36}$$

Dies funktioniert, falls die Zentren eine (echte) Teilmenge der Trainingssamples sind. Eine weitere ähnliche Variante ist das RBF-Netz von Moody und Darken [MD89] (RBF-MD-Netz), das wir nur kurz vorstellen. Die Verarbeitung im RBF-MD-Netz geschieht in folgenden Schritten:

1. Initialisierung der Zentren durch ein Clusterverfahren.
2. Einstellung der Radien proportional zum mittleren Abstand der nächsten RBF-Neuronen.
3. Die Gewichte werden wieder durch Berechnung der Pseudoinversen eingestellt.

Trotz der o.g. Lösungsansätze bestehen aber weiterhin die folgenden Probleme:

- die Wahl der Zentren. Die Zentren können z.B. durch ein Clusterverfahren berechnet werden mit den Zentren als Clustermittelpunkten.
- die Wahl der Radien und anderer Parameter der Basisfunktionen. Die Radien können z.B. als Cluster-Radius angesehen werden.
- die Art der Basisfunktionen, z.B. könnte man statt radialsymmetrischen Funktionen ellipsoide wählen.

Die folgenden RBF-Netz-Varianten arbeiten adaptiv. Lösungsansätze und ihre Eigenschaften werden präsentiert.

6.4 Fünfte Variante – PNN, RCE- und DDA-Netze

Exakte Rechnungen lassen sich oft nur für bestimmte Funktionen durchführen und die Herleitung kann schwierig sein. Deshalb stellt sich die Frage nach *adaptiven* Varianten mit einem stärkeren algorithmischen Charakter. Um daraufhin zu arbeiten, wollen wir im Folgenden kurz zwei Netzwerke und ihre Schwächen vorstellen, das **Probabilistische NN** von Specht [Spe90] (Abk.: PNN) und das RCE-Netz (engl. für: Restricted Coloumb Energy) von Reilly, Cooper und Elbaum [RCE82]. Dabei handelt es sich nicht direkt um RBF-Netze, aber die Ideen dieser beiden Netze wurden zum sog. RBF-DDA-Netz vereint, das im Anschluss als fünfte Variante vorgestellt wird. Ohne auf die Details einzugehen, stellen wir kurz ein paar Fakten über das PNN zusammen.

- Das RBF-DDA-Lernen verwendet die *Bayessche Entscheidungstheorie*, vgl. [BH99, Kap. 2]. Den Satz von Bayes findet man im Anhang A.
- Alle Eingaben müssen als Gewichte in der Musterschicht gespeichert werden (hoher Speicherplatzbedarf).
- Die Eingabeschicht ist vollständig mit der Musterschicht verbunden.
- In der Summationsschicht gibt es genauso viele Neuronen wie Klassen. Diese Neuronen sind mit den entsprechenden Neuronen gleicher Klasse der Musterschicht verbunden.
- Die Berechnung der Gewichte entspricht der Speicherung im Training, das daher schnell ist.

Das **RCE-Netz** ahmt die Abstoßungsreaktionen geladener Teilchen nach.

Aktivierungselement: Wiederholen Sie die Aussagen des *Satzes von Bayes*, und beschreiben Sie die Funktionsweise eines Bayesschen Klassifikators.

Aktivierungselement: Was besagt das Coloumbsche Gesetz?

In der Abb. 6.2 sehen wir ein Ergebnis nach einem RCE-Lernen. Das Netz hat die folgenden Eigenschaften.

- Die Netztopologie ist ähnlich zu der des PNN (allerdings mit OR-Neuronen in der 2. Schicht),
- Aktivierungsfunktion $f(r_j - d(z_j, x))$, z_j Zentren, x Eingabevektor, r_j Radius/Schwellenwert, d Metrik, $f(z) = 1$, falls $z > 0$ und $f(z) = 0$, falls $z = 0$,
- keine Überlappungen der Neuronenaktivitäten zwischen verschiedenen Klassen (keine gute Generalisierung),
- weniger Zentren als Trainingspunkte.

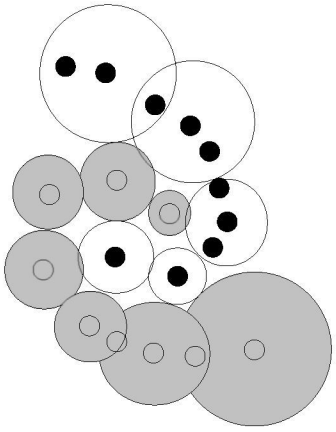

Abbildung 6.2. Ergebnis eines RCE-Lernens auf spiralähnlichen Daten (Skizze)

Das *destruktive* Lernverfahren für alle Trainings-Eingaben x, mit Initialisierung einer Hyperkugel für x mit einem vordefinierten Radius, wird im Folgenden skizziert:

Algorithmus 6.1 (RCE-Lernen) 6.1

1. Gehe alle Neuronen durch und unterscheide die vier Fälle:

a) Die Eingabe liegt in einer Hyperkugel. Man unterscheidet:

a1) Die Eingabe gehört zur gleichen Klasse wie die Hyperkugel: tue nichts.
a2) Die Eingabe gehört nicht zur gleichen Klasse wie die Hyperkugel: Die Radien der Hyperkugeln müssen reduziert werden (**destruktive Operation** = bereits gelernte Muster werden falsch eingeordnet).

b) Die Eingabe liegt nicht in einer Hyperkugel. Man unterscheidet:

b1) Die Eingabe gehört zur gleichen Klasse wie die Hyperkugel: keine Aktion.
b2) Die Eingabe gehört nicht zur gleichen Klasse wie die Hyperkugel: Erniedrige Radius, um Durchschnittsbildung zu vermeiden.

2. Wiederhole 1 bis keine destruktive Operation mehr notwendig ist.

□

Aktivierungselement: Recherchieren Sie genauer die Arbeitsweise des PNN und des RCE-Netzes bei Interesse. Veranschaulichen Sie sich die Algorithmen.

Das **RBF-DDA-Netz** (DDA steht für „Dynamic Decay Adjustment") versucht die Vorteile des PNN und des RCE-Netzes zu kombinieren [BH95], [Ber97], [BH98]. Das PNN verwendet im Gegensatz zum RCE-Netz überlappende Aktivierungsbereiche. Der Vorteil des RCE-Netzes ist die geringere Anzahl an benötigten Neuronen im Vergleich zur Anzahl an Datentupeln. Dies führt zu einer guten Generalisierung unter Verwendung weniger Zentren.

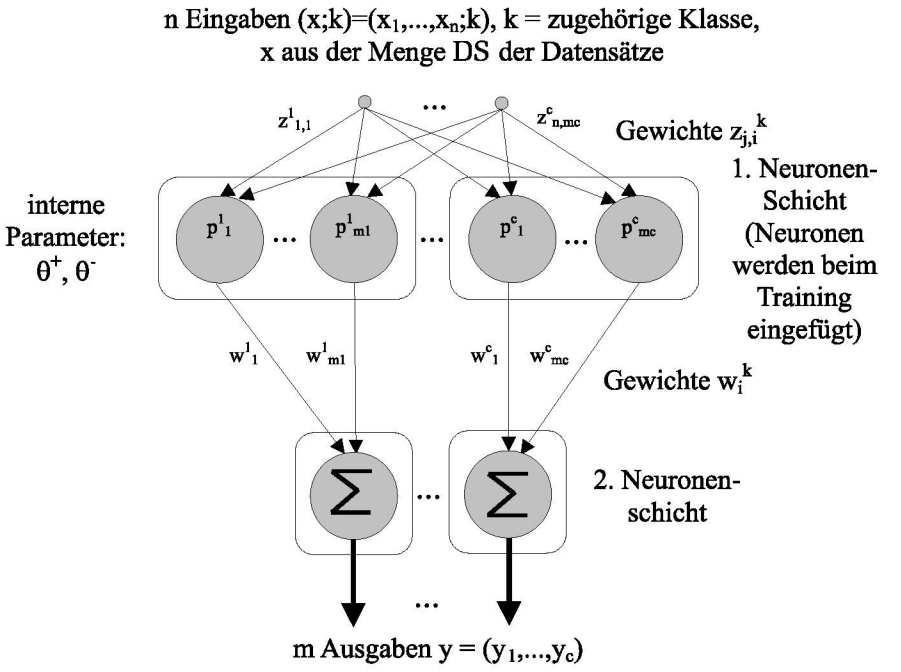

Abbildung 6.3. Architektur des RBF-DDA-Netzes

6.2 Algorithmus 6.2 (RBF-DDA-Netz)

Parameter: R_i^k ist die Aktivierungsfunktion mit Radius σ_i^k des i-ten Prototyps p_i^k der k-ten Klasse, siehe Formel (37) im Anschluss an den Algorithmus. Die Standardwahl der Schwellen θ^+ und θ^- ist $\theta^+ := 0.4$ und $\theta^- := 0.2$. Eine andere Wahl kann das Lernverhalten beeinflussen.

Lernphase einer Epoche:
1. Gewichte zurücksetzen:
for $k = 1$ **to** c
  for $i = 1$ **to** m_k
    $w_i^k := 0$;
  end
end

2. Muster durchgehen:
for $r = 1$ **to** size($\mathcal{DS}$)
  if $R_i^k(x_r) \geq \theta^+$ für ein $i \in 1, \ldots, m_k$,

3. Muster durch p_i^k überdeckt (cover):
   $w_i^k := w_i^k + 1$;

4. Neues Neuron einfügen (commit):
   else
   $m_k := m_k + 1$;
   $w_{m_k}^k := 1.0$;
   $z_{m_k}^k := x$;

5. Radius adaptieren:
   $\sigma_{m_k}^k := \min_{l \neq k, 1 \leq j \leq m_l} \left\{ \sqrt{-\frac{||z_j^l - z_{m_k}^k||^2}{\ln \theta^-}} \right\}$;
   end

6. Konfliktneuronen verkleinern (shrink):
   for $l \neq k$
   for $j = 1$ **to** m_l
   $\sigma_j^l := \min \left\{ \sigma_j^l, \sqrt{-\frac{||x - z_j^l||^2}{\ln \theta^-}} \right\}$;
   end
   end
end

□

Im Schritt 3 verändert [BH95] nur das Gewicht *eines* Neurons, auch wenn die Bedingung auf mehrere Neuronen zutreffen kann. Es besteht aber nicht unbedingt ein Unterschied bezüglich der Performanz des Netzes, falls man das Gewicht all dieser Neuronen erhöht. Es wird noch die Anwendungsphase des Netzes beschrieben.

Ausführungsphase (symmetrische Radien):

$$\text{1. Schicht:} \qquad R_i^k(x) := \exp\Big(- \frac{||x - z_i^k||_2^2}{(\sigma_i^k)^2} \Big), \tag{37}$$

$$\text{2. Schicht:} \quad y_k(x) := y_k(R_i^k(x)) := \sum_{i=1}^{m_k} w_i^k R_i^k(x) \tag{38}$$

und normalisierte Klassenwahrscheinlichkeit:

$$P(k|x) = \frac{y_k}{\theta^- + \sum_{l=1}^{c} y_l} \quad . \tag{39}$$

Interessant ist die Eigenschaft der Konvergenz des RBF-DDA-Lernverfahrens auf den Trainingsdaten, insofern man voraussetzt, dass keine identischen Trainingsvektoren mit unterschiedlicher Klassenzugehörigkeit existieren [Ber97, S. 37-38]. Diese Voraussetzung ist allerdings in der Praxis nicht unbedingt erfüllt.

Ein Abwandlung des RBF-DDA-Lernens wird in [Pae04b] vorgeschlagen. Nach jeder Epoche wird überprüft, ob jeweils mehr als ein Testsample von den Neuronen überdeckt wird. Ist das der Fall, so werden die als temporär gekennzeichneten Neuronen zu dauerhaften Neuronen. Deshalb heißt die Variante auch RBF-DDA-T („T“ = temporär). Ist das nicht der Fall, so werden die temporär eingefügten Neuronen wieder entfernt und auch die dazugehörigen Trainingssamples. Das resultierende RBF-DDA-T-Netzwerk benötigt je nach Datensatz ca. 50% weniger Neuronen bei gleichbleibender Klassifikationsgüte. Weitere Varianten, die wir nur kurz ansprechen werden, sind z.B. das überwacht wachsende neuronale Gas oder Basisfunktionen-Netze mit ellipsoiden Basisfunktionen.

6.5

6.5 Sechste Variante – Überwachtes Wachsendes Neuronales Gas

Das überwacht wachsende neuronale Gas (engl.: Supervised Growing Neural Gas, SGNG) [Fri94] beruht auf dem (unüberwachten) wachsenden neuronalen Gas (GNG), das wir im Abschnitt 9.5 über Kohonen-Netze kennenlernen werden. Im Unterschied werden in der überwachten Variante RBF-Neuronen verwendet und es wird fehlerbasiert gelernt.

Aktivierungselement: Bei Interesse finden Sie in [Fri98] eine hübsche Nebeneinanderstellung der verschiedenen Lernverfahren Kohonen-Lernen, NG, GNG, SGNG sowie der (wachsenden) Zellstrukturen (GCS). Wie gesagt, dazu aber auch mehr im Abschnitt über Kohonen-Karten.

An dieser Stelle sei noch darauf hingewiesen, dass das GNG im Gegensatz zum Neuronalen Gas (NG) keine feste Anzahl von Neuronen verwendet, sondern eine wachsende.

6.6 Siebte Variante – Elliptische Basisfunktionen

6.6

In [Ber97, S. 82-89] werden anstelle von radialsymmetrischen Aktivierungsfunktionen *elliptische* Basisfunktionen verwendet. Die Aktivierung ergibt sich zu

$$f(x) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp -\frac{1}{2}(x-z)^T \Sigma^{-1}(x-z) \ . \tag{40}$$

mit z als Zentrum und Σ als Kovarianzmatrix. Die Aktivierung ist ein Produkt verschiedener eindimensionaler Normalverteilungen. Das hat den Vorteil einer geringfügig kleineren Anzahl von Neuronen, aber den Nachteil eines hohen Berechnungsaufwands im Training.

Eine Verwendung von elliptischen Basisfunktionen macht nur in speziellen Anwendungen Sinn, nämlich wenn die Daten zu den Ellipsoiden „passen“ und Hyperkugeln weniger geeignet sind.

6.7 Anwendungen – SVM

6.7

Die Diversität der vorgestellten Varianten zeigt, dass die Auswahl eines geeigneten Netzes für ein Problem durchaus schwierig sein kann und z.Z. eher experimentell denn exakt möglich ist. Weitere Varianten findet man in [HJ01a], [HJ01b]. Es folgt eine kleine Auswahl von Anwendungen (außerhalb der Bioinformatik) von RBF-Netzen. Anwendungen in der Bioinformatik werden im Kap. 8 dargestellt.

- In [PB99] wird mit RBF-Netzen (elliptischen Basisfunktionen) das Gummiprofil in der Reifenherstellung untersucht.
- In [Bla98, Kap. 5] werden RBF-Netze zur Patientenvorhersage im Vergleich zum medizinischen Score SAPS II verwendet (beide mit Ergebnissen ähnlicher Güte).
- In [SM96], [THV94] und [CB94] werden RBF-Netze zur Systemidentifikation nichtlinearer Systeme verwendet.
- In [JSYY02] werden gewöhnliche Differentialgleichungen mit RBF-Netzen gelöst.
- In [Pae02d] wird eine Methode vorgeschlagen, die basierend auf Projektionen der RBF-Radien, eine Merkmalswahl durchführen kann.
- In [NVS99] werden RBF-Netze für Navigationsaufgaben eingesetzt.
- In [KK02] werden RBF-Netze für kryptographische Aufgaben verwendet.

Zum Abschluss des Kapitels sei angemerkt, dass sich sowohl Backpropagation-Netzwerke als auch RBF-Netzwerke als Spezialfall der sog. **Support-Vektor-Maschinen** erweisen (Abk.: SVM), die an Bedeutung gewonnen haben, da sie sehr gute Generalisierungseigenschaften besitzen. Allerdings spielen sie eine untergeordnete Rolle bei der Regelgenerierung. Es sei z.B. auf [Hay99, Kap. 6] oder [Sch97] verwiesen. An dieser Stelle soll nur die Grundidee wiedergegeben werden.

Die Datenpunkte x_i kann man evtl. nicht gut im zugrunde liegenden Raum in verschiedene Klassen auftrennen. Die Idee ist eine nichtlineare Transformation φ dieser Datenpunkte in einen Transformationsraum zu $\varphi(x)$, so dass die transformierten Punkte besser trennbar sind. Solch eine Transformation kann durch eine RBF gegeben sein. In diesem Zusammenhang bezeichnet man $K(x, x_i) := \varphi^T(x)\varphi(x_i)$ als *Kern*, z.B. ist bei den RBF-Netzen $K(x, x_i) = \exp(-\frac{1}{2\sigma^2}||x - x_i||)$. Die Kerne sind also in der versteckten Schicht zu finden. Die Zentrumsvektoren sind dann die Support-Vektoren. Bei der SVM wird dann ein Optimierungsproblem gelöst, dass optimale Hyperflächen im Transformationsraum (im Sinne von Abständen der Support-Vektoren zu den Hyperflächen) findet.

Anwendungen der SVM in der Bioinformatik sind beispielsweise:
- Sekundärstrukturvorhersage [HS01],
- Bestimmung der Strukturklassen von Proteinen [CLXC02],
- Vorhersage der Bindungsstärke von Peptiden der Länge neun an Klasse-I-MHC-Molekülen im Rahmen der Immunologie [RKSK04],
- Chemieinformatik: Klassifikation Drug/Non-Drug [BFSS03].

In diesem Kapitel haben wir eine Reihe von Ideen präsentiert, die die RBF-Netze betreffen, also von neuronalen Netzen mit radialen Basisfunktionen als Aktivierungsfunktionen. Vorgestellt wurden mathematische und algorithmische Ideen wie die des RBF-PG-Netzes und des RBF-MD-Netzes. Die Netztopologie des PNN und das RCE-Lernen führten zum RBF-DDA-Netz. Es existieren zahlreiche Varianten, die die Adaption der Netzparameter betreffen. Durch die Verwendung der radialen Basisfunktion als Kern wird das RBF-Netz ein Spezialfall der SVM. RBF-Netze wie SVM werden für viele Problemstellungen zur Klassifikation innerhalb und außerhalb der Bioinformatik eingesetzt.

Kapitel 7

7

Regelgenerierung aus RBF-Netzen

7

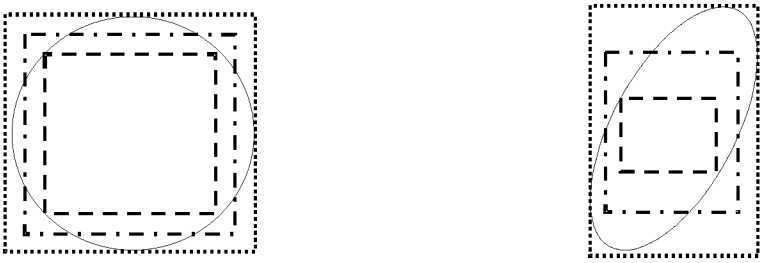

Abbildung 7.1. Regelgenerierung: Quadrat aus einem Kreis; Rechteck aus einer Ellipse

7 Regelgenerierung aus RBF-Netzen

Angenommen, man hat ein fertig trainiertes bzw. berechnetes RBF-Netz vorliegen. Wir fragen uns, ob wir die Struktur des Netzes, d.h. die Zuordnung der Eingabe zur Ausgabe, durch Regeln beschreiben können. Im Einzelnen soll das kurze Kapitel zu folgenden beiden Zielen führen, die später im Rahmen der Fuzzy-Logik erweitert werden.

Lehrziele:
- Die Probleme einer naiven Regelgenerierung aus RBF-Netzen kennen,
- die Idee des RecBF-DDA-Netzes kennen und wissen was eine Intervall-Regel ist.

7.1 Naive Idee der Regelgenerierung

Ein Kreis mit dem Radius r entspricht einer Regel, wenn man den Kreis durch ein Quadrat ersetzt. Analog könnte man eine Ellipse durch ein Rechteck ersetzen, vgl. die Abb. 7.1. Bei solch einer Vorgehensweise treten allerdings Probleme auf.

- Die Zuordnung eines Kreises zu einem Quadrat bzw. einer Ellipse zu einem Rechteck ist nicht eindeutig.
- Kreise können keine rechteckigen Regeln repräsentieren.
- Nicht-achsenparallele Ellipsen lassen zu viel Platz für eine Interpretation.
- Die Regelanzahl ist gleich der Anzahl der Neuronen, was bei einer hohen Anzahl von Neuronen nicht sinnvoll erscheint.
- Die Netze sind zum Lernen der Klassifikations- bzw. Approximationsaufgabe vorgesehen, nicht speziell zur Regelgenerierung (Problem der A-posteriori Regelgenerierung).

Aufgrund dieser Probleme werden wir im nächsten Abschnitt eine Lösung vorstellen, die statt RBF-Funktionen rechteckige Basisfunktionen verwendet.

7.2

7.2 RecBF-DDA-Netz

In [BH95] werden statt Kreise als Basisfigur Rechtecke (Hyperquader in höheren Dimensionen) gewählt. Unendliche Ausdehnungen werden dabei in den Dimensionen zugelassen. Nun benötigt man noch ein entsprechendes Lernverfahren, dass sowohl die Klassifikation als auch die Regelgenerierung unterstützt. Diese Lösung wird beim **RecBF-DDA** (Rec = Abk. für „Rectangular") vorgeschlagen. Strenggenommen haben Rechtecke keine *Radien*, sondern *Ausdehnungen* in die einzelnen Dimensionen; in Analogie zu den RBF-DDA Netzen werden wir aber weiterhin von Radien sprechen.

7.1 **Algorithmus 7.1 (RecBF-DDA-Netz)**

Parameter:

Für alle Dimensionen d benötigt man zusätzlich zu den RBF-DDA-Parametern einen Parameter $\sigma_{d,\min}$, der zu „schmale" Regeln vermeiden hilft, d.h. der σ_i, $i = 1, \ldots, n$, nach unten begrenzt; er kann als Standard (bei normalisierten Daten in $[0, 1]$) auf 0.1 gesetzt werden. - K^∞ sei die Menge derjenigen Radien, die *unendlich* werden dürfen; K^f sei die Menge der endlichen Radien. „f" steht für *finite*. Im Laufe des Verfahrens können als unendlich definierte Radien von K^∞ in die Menge K^f durch die shrink-Operation wechseln.

Eine Erweiterung stellt die Verwendung zweier Radiensätze $\sigma^k_{i,+}$ und $\sigma^k_{i,-}$ mit $\sigma^k_{i,+}$ und $\sigma^k_{i,-}$ pro Prototyp p^k_i anstelle des einen Radiensatzes σ^k_i dar. Man erhält dann *asymmetrische* Rechtecke, die zu einer Verbesserung der Regelgenerierung führen, siehe [BH95].

Lernphase:

Der Lernalgorithmus unterscheidet sich im shrink-Mechanismus. Es wird diejenige Dimension zur Verkleinerung des Radius ausgewählt, die das größte Volumen des verbleibenden Rechtecks erhält. Dabei ist eine Fallunterscheidung zwischen den Radien aus K^∞ und K^f nötig (bevorzugt K^f schrumpfen).

□

Wir beschreiben die Ausführungsphase für symmetrische Radien, wobei d eine von n Dimensionen des Datenraums sei.

1. Schicht:

$$R_i^k(x) := \min_{1 \le d \le n} A(x_d, z_{d,i}^k, \sigma_{d,i}^k),\ k = 1, \ldots, c,\ i = 1, \ldots, m_k, \quad (41)$$

$$A(x_d, z_{d,i}^k, \sigma_{d,i}^k) := \begin{cases} 1 & , |x_d - z_{d,i}^k| \le \sigma_{d,i}^k \\ 0 & , \text{sonst} \end{cases} \quad (42)$$

2. Schicht: gewichtete Summe (38) analog zum RBF-DDA-Netz.

Für die i-te Regel erhalten wir durch die Regelgenerierung aus den Rechtecken:

$$i\text{-te Regel } \textbf{if } \forall\ 1 \le d \le n : x_d \in (z_{d,i}^k - \sigma_{d,i}^k, z_{d,i}^k + \sigma_{d,i}^k) \textbf{ then Klasse } k$$

Diese Form von Regeln gilt für endliche Gebiete; erlaubt man unendliche Radien, so kann man die für die Regel wichtigen Dimensionen selektieren. Eine unwichtige Dimension d ist eine mit der Regel: **if** ... **and** $x_d \in (-\infty, \infty)$ **and** ... **then** Klasse c.

Definition 7.2 Wir bezeichnen eine Regel der Form **7.2**

$$\textbf{if } \forall\ 1 \le d \le n : x_d \in (z_{d,i}^k - \sigma_{d,i}^k, z_{d,i}^k + \sigma_{d,i}^k) \textbf{ then Klasse } k$$

als **Intervall-Regel**. Eine Intervall-Regel besteht also aus Bedingungen an die Variablen in den einzelnen Dimensionen innerhalb eines endlichen, halboffenen oder unendlichen Intervalls zu liegen.

□

Die Verwendung einer naiven Idee zur Regelgenerierung führte zu einer Reihe von Problemen, so dass das RecBF-DDA-Netz mit rechteckigen Basisfunktionen in der Literatur vorgeschlagen wurde. Das Problem einer Zuordnung der Intervall-Regeln zu sprachlichen Regeln, d.h. Verwendung von Termen wie „hoch“ oder „mittel“ bleibt aber bestehen. Später werden wir sehen, dass „intelligentere“ Lösungen durch *Neuro-Fuzzy-Systeme* gegeben sind. Wenn uns die Begriffe der Fuzzy-Theorie geläufig sind, werden wir u.a. sehen wie die RBF-Netze mit der Fuzzy-Theorie kombiniert werden können.

Kapitel 8

RBF-Netze in der Bioinformatik

8

8

8 RBF-Netze in der Bioinformatik

Wir stellen fünf Anwendungen aus den Bereichen Bioinformatik und Chemieinformatik vor. Anschließend wird eine RBF-Netz-Variante vorgestellt, die *Bio-Basisfunktionen* verwendet.

Lehrziele:
- Die Verwendung von neuronalen Netzen zur Klassifikation von Genomsignaturen anhand der Motive kennenlernen,
- die Möglichkeit der Verwendung neuronaler Netze zur Proteinklassifikation kennen,
- weitere Anwendungen neuronaler Netze kennenlernen wie die Klassifikation von Phytoplanktonarten, die Vorhersage des Siede- und Flammpunkts von organischen Stoffen und die Vorhersage der Retention einer Flüssig-Chromatographie,
- die Verwendung von Bio-Basisfunktionen kennen.

8.1 Klassifikation von Genomsignaturen

Betrachten wir die Nukleotide A, C, G, T. Ein **Motiv** ist eine Subsequenz, z.B. ist CAT ein Motiv der Länge 3 der Sequenz GCATAGCA. In [CB02] wählt man eine Fensterlänge von 100, um aus solch einer Sequenz Zeichenketten der Länge 100 zu bilden. Dabei wird das Fenster jeweils um ein Zeichen nach rechts geschoben. Die Motivlänge sei als 3 vorgegeben. Eine *Genomsignatur* oder *Profil* seien alle Motive zusammen mit ihren Häufigkeiten.

Für die Eingabe eines RBF-Netzes wird in [CB02] die Genomsignatur gewählt. Als Ausgabe sind Genomklassen vorgegeben. Bei „(1,0)“ gehört die Signatur zu der Genomklasse *Aeropyrum pernix*, bei „(0,1)“ zu der Genomklasse *Mesorhizobinum loti*. Jeweils 2000 Signaturen von beiden Klassen werden verwendet, jeweils 1000 zum Trainieren. Backpropagation mit 7 versteckten Neuronen lieferte 97.48% korrekte Vorhersagen. Ausprobiert wurden 1 bis 9 versteckte Neuronen. Die Zentren eines RBF-Netzes wurden zufällig mit ausgewählten Trainingsdaten initialisiert und mit k-means adjustiert. Mit 200 Zentren erreichte man 93.99% korrekte Vorhersagen. Ausprobiert wurden 200, 500 und 1000 Neuronen. Gelernt wurde mit einem Update der Zentrumsvektoren.

Ein SOM-Training, vgl. Kap. 9, mit einer 4×4-Karte brachte mit einer Klassifikation durch die Höhe der Aktivierung 96.45% korrekte Klassifikationen. Ausprobiert wurden $n \times n$-Karten, $n = 3, \ldots, 10$. Erstaunlicherweise ist in [CB02] das RBF sogar schlechter als die unüberwacht trainierte SOM. Möglicherweise kann dies an der starren Anzahl der Zentren des RBF-Netzes liegen. Man hat z.B. nicht 100, 200, 300, ... , 900, 1000 Zentren getestet. Wir haben im Abschnitt 6.4 aber bereits ein RBF-Lernen kennengelernt, bei dem dynamisch Neuronen eingefügt (und auch ausgefügt) werden.

8.2 Proteinklassifikation

Bei der Proteinklassifikation nach [WLDH02] handelt es sich um eine ähnliche Anwendung wie die vorhergehende. Üblicherweise geht man davon aus, dass zwei Proteine zur gleichen Superklasse gehören, wenn man eine hohe Homologie bei einem Sequenzalignment erhält.

In [WLDH02] werden 10 Superklassen betrachtet, z.B. Cytochrome c, c6, b, b5, Globin, u.a. Als Eingabe werden die Häufigkeiten der (max. 400) Kombinationen von 2 Aminosäuren betrachtet. Die 20 Aminosäurecodes wurden vorher in 6 Codes umkodiert. Die Eingabe x wird normalisiert durch $x_{\text{norm}} := x/(L - n + 1)$ mit L Länge der Sequenz und n Länge der Kombinationen, hier $n = 2$.

Ein Backpropagation-Netz mit 56-15-10-Architektur und ein RBF-Netz mit 71 Zentren (56 Eingaben, 10 Ausgaben) liefern etwa gleiche Ergebnisse. Trainiert wurde mit ursprünglich 949 Proteinsequenzen, getestet mit weiteren 533. Als Erweiterung wird eine Kombination zweier RBF-Netze zum Training vorgeschlagen.

8.3 Klassifikation von Phytoplanktonarten

In der Arbeit [WBMJ99] steht als Eingabe ein 11-dimensionaler Datenvektor aus jeder Messung an Phytoplankton bereit. Diese Eigenschaften sind zytometrische Eigenschaften wie Diffraktion (Beugung), Lichtstreuung, Floureszenz-Parameter.

34 Arten Phytoplankton werden als Ausgabe betrachtet. Je 400 Messungen wurden pro Art für Training und Test vorgenommen, also insgesamt 400 mal 34 mal 2 Messungen. 68 (2 mal 34) RBF-Neuronen wurden in der versteckten Schicht verwendet. Ausprobiert wurden 1 bis 4 Knoten pro Klasse. Man

erreichte 91.5% korrekte Resultate. Zusätzlich wurde die Relevanz der Eingabedimensionen überprüft. Die 4 Diffraktionsparameter waren nicht so wichtig wie die anderen 7 Parameter. War eine Ausgabe zu einer Eingabe niedriger als eine festgelegte Schwelle, so wurde die Eingabe als eine dem Netz unbekannte Art eingestuft.

8.4 Vorhersage des Siede- und Flammpunkts

8.4

In der Anwendung [TSMH99] wird der Siede- und Flammpunkt von organischen Stoffen vorhergesagt. Es werden 26 Eingaben verwendet, die Häufigkeit 25 funktionaler Gruppen und der molekulare Konnektivitätsindex 1_χ. Z.B. hat $C_7H_4ClF_3$ die funktionalen Gruppen Cl (1), F (3), $C_{\text{aromatisch}}$ (6), NH_2 (0), usw. Zwei Ausgeneuronen sind für den Siede- und Flammpunkt vorgesehen. Je ein Drittel der 400 Stoffe wurde zum Trainieren, Validieren und Testen verwendet. Man erreicht eine Vorhersagegenauigkeit im Rahmen von $\pm 10\%$.

8.5 Vorhersage der Retention einer Flüssig-Chromatographie

8.5

In [XLZ+02] wird die Retention (Rückhalt) einer Flüssig-Chromatographie (engl.: retention of liquid chromatographic) vorhergesagt. Betrachten wir dazu folgende Verbindung: $X-C_6H_4-CH=N-C_6H_4-Y$ mit den zwei Substituenten X und Y. Verschiedene Substituenten liefern 70 Verbindungen; 52 wurden zum Trainieren vorgesehen, 18 zum Testen eines RBF-Netzes.

Anhand quantenchemischer Parameter (Dipolmoment, Energien der höchsten besetzten und niedrigsten unbesetzten Molekülorbitale, Ladung des negativsten Atoms, Summe der absoluten Ladungsrate aller Atome in zwei vorgegebenen funktionellen Gruppen, totale Energie des Moleküls) und einer 4-dimensionalen Codierung der Position der Substituenten soll die Retention vorhergesagt werden.

Es wurde ein RBF-Training verwendet, bei dem für alle Trainingsmuster ein Neuron in der versteckten Schicht vorgesehen ist, ähnlich zum PNN. Die Eingaben wurden Mittelwert- und Varianz-normiert. Die Architektur ist also ein 11-52-1-Netz. Mit einem festen Zentren-Radius von 0.38 erzielte man im Mittel 1.56% Abweichung der Retention vom korrekten Wert.

Zusammenfassend kann man festhalten, dass bei konkreten Anwendungen Parameter (z.B. Radien der Zentren, Anzahl der Neuronen) ausprobiert werden und Details (z.B. Normierung, Lernalgorithmus) angepasst werden müssen. Kommen wir nun zu den Bio-Basisfunktionen.

8.6

8.6 Bio-Basisfunktionen

Im Folgenden wird ein Ansatz wiedergegeben, der anstelle von RBF **Bio-Basisfunktionen** (Abk.: BBF) verwendet [THYD03]. Die Aufgabe ist die Charakterisierung der Aktivität proteolytischer Spaltungsstellen (engl.: cleavage sites) von Sequenzen. Höhere Organismen verwenden Proteasen beispielsweise zur Verdauung oder zur Spaltung von Signalpeptiden. Es wird ein Vorgehen vorgeschlagen, dass ohne Kodierung wie BIN20 auskommt.

In einem RBF-Netz werden die RBF-Neuronen in der versteckten Schicht durch BBF-Neuronen erstezt. Dabei versteht man unter Biobasen Proteinsequenzen aus einer gegebenen Datenmenge. Während in den RBF-Neuronen ein Euklidischer Abstand berechnet wird, greift man bei den BBF-Neuronen auf Abstands-Matrizen zurück, z.B. auf die *Dayhoff'sche Abstandsmatrix* [DSO78]. Eine hohe Ähnlichkeit soll zu einer hohen Aktivität führen. Als Beispiel sei der (biologische) Dayhoff-Abstand genannt, z.B. von M zu A ist der Abstand 0.0, von Z zu E 1.1 und von G zu F -0.6.

Sei D die Anzahl der Stellen der Eingabesequenz, die Einfügesignale akzeptieren. D ist gleichzeitig die Anzahl der Eingabedimensionen des BBF-Netzes. Sei C die Menge der Aminosäuren mit der Mächtigkeit 20. Die Eingaben x_n sind aus dem Raum $C \times \ldots \times C = C^D$ mit D Stellen. Sei s die Ähnlichkeit, b_n die maximal mögliche Ähnlichkeit zu x und α eine Konstante. Dann wird die Aktivierungsfunktion f definiert durch:

$$f(x_n) := \exp\left(\alpha \frac{s(x, x_n) - b_n}{b_n}\right) \tag{43}$$

Berechnet werden die Gewichte durch die Pseudo-Inversen-Technik (Dritte Variante im Kap. 6). Für die Klassenausgabe wird ein Schwellwert für die Aktivierung des Ausgabeneurons verwendet. Berechnet wurden Mittelwerte und Standardabweichungen von 20 Modellen mit verschiedenen Trainingsdaten.

Angewendet wurde die BBF-Methode auf 413 Tetrapeptid-Sequenzen, u.a. mit den Klassen „cleaved“ und „uncleaved“. Die Vorhersage war zu 98.3%

korrekt. Bei einer Anwendung auf 362 HIV-Protease-Sequenzen (114 mit positiven Spaltungsstellen und 248 negativen) erreichte man 93.4% korrekte Vorhersagen.

Der Typ der RBF-Netze lässt sich vielfältig anwenden. Anwendungsbeispiele waren neben anderen die Klassifikation von Genomsignaturen oder von Proteinen. Wie wir gesehen haben, kann es sinnvoll sein, verschiedene Typen neuronaler Netze zu einer Problemstellung auszuprobieren, da nicht alle Netze die gleiche Performanz erreichen müssen. Ein interessanter Aspekt ist die Variante der Bio-Basisfunktionen, bei denen direkt der Abstand der Aminosäuren in die Berechnung der Aktivierung einfließt.

Kapitel 9

Kohonen-Netze und Varianten

9

9 Kohonen-Netze und Varianten

In diesem Kapitel werden die unüberwacht lernenden *Selbstorganisierenden Karten* inklusive einiger Varianten vorgestellt. Zur Visualisierung wird die *U-Matrix-Methode* präsentiert.

Lehrziele:
- Die Bestandteile einer selbstorganisierenden Karte wie beipielsweise die Nachbarschaftsfunktion angeben können,
- das Schema einer SOM skizzieren können,
- einige typische Anwendungen nennen können,
- die Idee der Lernenden Vektorquantisierung kennen und von der SOM unterscheiden können,
- die U-Matrix-Methode zur Visualisierung der SOM kennen,
- die Idee der Wachsenden Zellstrukturen kennen,
- wissen, wie das Neuronale Gas funktioniert,
- die Problematik nicht-stationärer Daten kennen.

9.1 Selbstorganisierende Karten

Selbstorganisierende Karten (engl.: self-organizing maps, Abk.: SOM, auch: *Kohonen-Karten*) sind im Wesentlichen durch Kohonen [Koh82] bekannt geworden. Es existieren zahlreiche Varianten dieses unüberwachten Lernverfahrens, das auf einer Nachbarschafts-erhaltenden Karte basiert, die sich im Datenraum entfaltet [Koh97]. Die Knoten der Karte repräsentieren nach Beendigung (Konvergenz) des Lernverfahrens den Datenraum. Einen Überblick über Kohonen-Netze geben auch [Hay99, Kap. 9], [Bra95, S. 140-169] und [Zel94, Kap. 15]. Ähnlich zur SOM funktioniert die Lernende Vektorquantisierung (Abk.: LVQ), die aber keine Nachbarschaften verwendet, siehe [Zel94, Kap. 14] oder [Koh97, Kap. 6].

Die SOM-Lernregel wird nach [Koh97, S. 86-88] wiedergegeben: Der Eingaberaum $\mathcal{I}$ sei eine Teilmenge von $\mathbb{R}^n$. Jedem *Knoten* (Neuron) i aus einem Ausgaberaum $\mathcal{O}$ sei ein **Referenzvektor** $w_i = (w_{i1}, \ldots, w_{in})$ zugeordnet, siehe die Abb. 9.1. Die Knoten sind in einer Topologie, z.B. einer rechteckigen, angeordnet. $x(t)$ sei der Eingabevektor zum Zeitpunkt t. Sei I der **Gewinner-Knoten**, d.h. $I := \text{index}(\min_i\{||x(t) - w_i(t)||\})$.

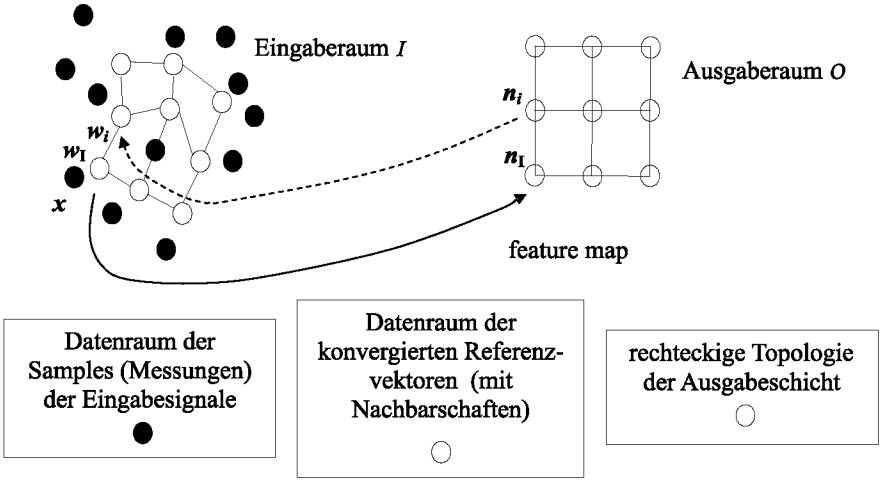

Abbildung 9.1. Das Schema einer SOM

Die **Nachbarschaftsfunktion** $N(I, i; t)$ sei z.B. durch eine Gaußsche Funktion $N(I, i; t) := \eta(t) \exp\left(-\frac{\|n_I - n_i\|}{2\sigma^2(t)}\right)$ gegeben (η Lernrate, n_I und n_i Ortsvektoren aus $\mathcal{O}$ der Knoten I bzw. i, oft aus $\mathbb{R}^2$ gewählt). Die Gewichte $w_i(t)$ zum Zeitpunkt $t = 0$ seien zufällig gewählt. Die Lernregel lautet dann

$$w_i(t+1) := w_i(t) + N(I, i; t)(x(t) - w_i(t)) \ . \tag{44}$$

Einige Varianten und Anwendungen der SOM sind:

- In [Koh97, Kap. 5] wird u.a. ein überwachtes Lernverfahren mit SOM vorgestellt sowie ASSOM (Abk. für: Adaptive Subspace SOM) zur Segmentierung von Bildern.
- Web-SOM [KKL+00] zur Erlernung von Clusterstrukturen in Textdokumenten.
- Wachsende Kohonen-Netze (Growing Grid) werden in [Fri98, Abschnitt 9.3] vorgestellt.
- In [Rit93] wird die *Parametrized Self-Organizing Map* eingeführt. Trotz weniger verwendeter Knoten soll mit Hilfe einer Interpolation eine möglichst stetige Ausgabe erreicht werden.
- [Bau96] beschreibt, wie der sog. *magnification factor* (dt.: Vergrößerungsfaktor) μ kontrolliert werden kann. Damit erreicht man im Falle $\mu = 1$ die Adaption der SOM-Knoten an die Dichte der zugrundeliegenden Daten, allerdings mit deutlich erhöhtem Laufzeitverhalten.
- [HV02] modifiziert die Metrik für einzelne Dimensionen adaptiv, so dass relevante Dimensionen stärker berücksichtigt werden.

- In [Gui98], [GU99] wird ein SOM-Ansatz vorgestellt, der die Struktur multivariater Zeitreihen medizinischer Daten erlernt.
- [VWG00] gibt einen aktuellen Überblick über einige Anwendungen der SOM in der Medizin, insbesondere zur Dimensionsreduktion.
- Anwendungen von SOM für Probleme aus der Bioinformatik (Clusterung von Merkmalen von Aminosäuren) findet man in [WM00, Abschnitt 4.2].

Aktivierungselement: Implementieren Sie die Lernregel der SOM anhand einer (3×3)-Karte im Einheitsquadrat $[0,1] \times [0,1]$. Verteilen Sie zur Initialisierung die Daten und die Gewichtsvektoren zufällig innerhalb eines von Ihnen festzulegenden Bereich des $\mathbb{R}^2$. Visualisieren Sie die Punkte nach jedem Lernschritt.

9.2 Lernende Vektorquantisierung

9.2

Die lernende Vektorquantisierung (engl.: learning vector quantization, Abk.: LVQ) von Kohonen wird der Vollständigkeit halber erwähnt. Das Lernverfahren LVQ funktioniert ähnlich wie das Kohonen-Lernen. Bei der LVQ werden allerdings Klasseninformationen benötigt, die zum Lernen der Gewichte zwischen Eingabe- und LVQ-Schicht verwendet werden; es gibt keine Nachbarschaftsfunktion. Sei g der Index des Gewinnerneurons und x eine Eingabe. Dann werden die Gewichte w_k zum Zeitpunkt $t+1$ definiert als

$$w_k(t+1) := w_k(t) \pm \alpha(t)(x(t) - w_k(t)) \ . \tag{45}$$

Das Vorzeichen „+“ wird verwendet, falls die Klasse der Eingabe mit der des Gewinnerneurons übereinstimmt, ansonsten wird das Vorzeichen „−“ verwendet. Die übrigen Gewichte bleiben unverändert.

9.3 U-Matrix-Methode

9.3

Die *U-Matrix-Methode* [US90] wurde als Visualisierung von Kohonen-Netzen eingeführt. Das „U“ steht dabei für *unified distance*. Größere Abstände der Kohonen-Knoten führen zu größeren Matrixeinträgen, die als Clustergrenzen visualisiert werden können, je nach Höhe in verschiedenen Farben. Zur Einführung der U-Matrix werden die Schreibweisen aus dem letzten Kapitel

beibehalten mit dem Zusatz, dass die Knoten $n_i \in \mathbb{R}^2$ in einem Gitter im Ausgaberaum mit zugehörigen Gewichten w_{ij} angeordnet sind und d eine Metrik (Abstandsmaß) im Eingaberaum bezeichnet.

Die folgende Definition der *U-Matrix* geht von einer ausgelernten zweidimensionalen $(n \times n)$-Kohonen-Karte $\mathcal{K}$ mit quadratischem Gitter aus.

9.1 **Definition 9.1 (U-Matrix)**

a) Die U-Matrix [US90], [Ult91, Abschnitt 7.3] wird mit Hilfe folgender Abstände $(i, j \in \mathbb{N}_n)$ definiert:

$$d_x(i,j) := d(w_{ij}, w_{i+1,j}) \ , \tag{46}$$

$$d_y(i,j) := d(w_{ij}, w_{i,j+1}) \ , \tag{47}$$

$$d_{xy}(i,j) := d(w_{ij}, w_{i+1,j+1}) \ , \tag{48}$$

$$d_{yx}(i,j) := d(w_{i,j+1}, w_{i+1,j}) \ , \tag{49}$$

$$d_z(i,j) := \frac{1}{2}(d_{xy}(i,j) + d_{yx}(i,j)) \text{ und} \tag{50}$$

$$d_u(i,j) := 0 \ . \tag{51}$$

b) Die **U-Matrix** $U \in \mathbb{R}^{(2n-1)\times(2n-1)}$ ist gegeben durch:

$$U := \begin{pmatrix} d_z(1,1) \ldots d_y(1,j)\, d_z(1,j) \ldots d_y(1,n)\, d_z(1,n) \\ \vdots \quad \ldots \quad \vdots \quad \vdots \quad \ldots \quad \vdots \quad \vdots \\ d_x(i,1) \ldots d_u(i,j)\, d_x(i,j) \ldots d_u(i,n)\, d_x(i,n) \\ d_z(i,1) \ldots d_y(i,j)\, d_z(i,j) \ldots d_y(i,n)\, d_z(i,n) \\ \vdots \quad \ldots \quad \vdots \quad \vdots \quad \ldots \quad \vdots \quad \vdots \\ d_x(n,1) \ldots d_u(n,j) d_x(n,j) \ldots d_u(n,n) d_x(n,n) \\ d_z(n,1) \ldots d_y(n,j) d_z(n,j) \ldots d_y(n,n) d_z(n,n) \end{pmatrix} \ . \tag{52}$$

Zu beachten ist, dass für eine $(n \times n)$-Karte eine $((2n-1) \times (2n-1))$-Matrix entsteht.

□

Anschließend werden die Eingabedaten durch $\mathcal{K}$ abgebildet und zusammen mit dem Ergebnis der U-Matrix und dem Kohonen-Gitter visuell dargestellt. Dabei werden „Täler“ in der visuellen Darstellung als Cluster angesehen,

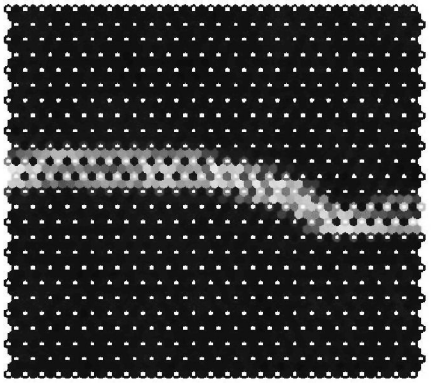

Abbildung 9.2. U-Matrix zweier getrennt liegender dreidimensionaler Normalverteilungen. In der Mitte hell, oben und unten dunkel. Erstellt mit „SOM Toolbox Version 1.0 beta“ (www.cis.hut.fi/projects/somtoolbox/)

die durch „Mauern“ getrennt sind. Dabei treten einige Probleme auf, wie z.B. unterschiedliche relative Höhen oder unterbrochene Mauern, die eine Interpretation der Cluster erschweren (siehe [UH91, Abschnitt 4.3]). Die U-Matrix-Methode kann man auch im Sinne einer *Multidimensionalen Skalierung* (Abk.: MDS) sehen, da multivariate Daten auf eine zweidimensionale Karte abgebildet werden. Zwei Anwendungen sind die Clusterung von Blutmesswerten [Ult91, Abschnitt 4.4] und die Lawinenprognose [Sch95].

Weitere Visualisierungsmethoden von SOM werden in [Ves99] beschrieben und veranschaulicht. In der Abb. 9.2 ist als Beispiel die U-Matrix zweier getrennt liegender dreidimensionaler Normalverteilungen wiedergegeben. Im Sinne einer Clusteranalyse kann man die „vertikale Mauer“ als Clustergrenze interpretieren, so dass in der Abb. 9.2 zwei getrennt liegende Cluster auf der zweidimensionalen Ebene zu sehen sind.

9.4 Wachsende Zellstrukturen

In [Fri92], [Fri98] werden die **Growing Cell Structures** (Abk.: GCS; Wachsende Zellstrukturen) betrachtet. Es handelt sich um eine Erweiterung der Kohonen-Netze dahingehend, dass Knoten und Kanten eingefügt werden können, die Netzstruktur also variabel ist, aber die Netzdimensionalität fest bleibt. Dabei werden Triangulierungen des Datenraumes vorgenommen, die allerdings entarten können. Die GCS sind bezüglich ihrer Netztopologie *flexibler* als Kohonen-Karten durch das Einfügen von Knoten. [CZ00a] präsentiert einen Zusatz für das Einfügen von Knoten *außerhalb* des triangulierten Knotennetzes.

9.5

9.5 (Wachsende) Neuronale Gase

Ähnlich wie ein Kohonen-Netz, nur ohne *feste* Netztopologie, funktioniert das **Neuronale Gas** (Abk.: NG) [Mar92]. Die Knoten breiten sich erst aus, anschließend können Nachbarschaften ausgebildet werden. Als Ergänzung zum NG werden im **Growing Neural Gas** (Abk.: GNG) Knoten eingefügt (und entfernt). Dabei werden die Nachbarschaften während des Lernvorgangs gebildet. Der Lernvorgang von NG und GNG wird in [Fri98, Kap. 8] anschaulich verglichen. Eine überwacht arbeitende Variante des GNG ist das Supervised GNG (Abk.: SGNG, [Fri94], [Fri98, Abschnitt 13.2]), bei dem der Knoten-Kanten-Struktur RBF-Knoten aufgesetzt werden und der Ausgabefehler zum Trainieren des Netzes mit einer Delta-Lernregel benutzt wird. Wir geben den Algorithmus zum Neuronalen Gas (NG) im Detail wieder (angelehnt an [Fri98, S. 66-67]).

9.2

Algorithmus 9.2 (Neuronales Gas)

1a) Initialisiere eine Menge M von Knoten (=Neuronen) z_i mit n Elementen: $M := \{z_1, \ldots, z_n\}$.

1b) Initialisiere die Referenzvektoren aller Knoten zufällig durch einen Zufallswert aus einer (angenäherten) Verteilung V der Daten: $w_i := random(V)$.

1c) Setze den diskreten Zeitparameter $t := 0$.

2) Betrachte eine Eingabe x (d sei z.B. die euklidische Metrik):

3) Sortiere die Menge $N := \{d(x, w_1), \ldots, d(x, w_n)\}$. Seien $i_1, \ldots, i_n$ die Indices der w_i, so dass N aufsteigend sortiert ist.

4) Adaptiere die Referenzvektoren unter Verwendung der Nachbarschaftsfunktion $h(t, k) := \exp(\frac{1-k}{\lambda(t)})$ und $\lambda(t) := \lambda_{\text{start}}(\frac{\lambda_{\text{ende}}}{\lambda_{\text{start}}})^{\frac{t}{t_{\max}}}$ sowie $\alpha(t) := \alpha_{\text{start}}(\frac{\alpha_{\text{ende}}}{\alpha_{\text{start}}})^{\frac{t}{t_{\max}}}$ (mit Konstanten $\alpha_{\text{start}} > \alpha_{\text{ende}}$, $\lambda_{\text{start}} > \lambda_{\text{ende}}$):

$$\Delta w_{i_k} := \alpha(t) h(t, k)(x - w_{i_k}) \ , \quad k = 1, \ldots, n \ . \tag{53}$$

Durch die Parameter sinkt zum Einen die Nachbarschaftsreichweite (h und λ) und zum Anderen die Adaptionsrate (α) weiter bezüglich der Sortierung der eingeordneten Gewichte gemäß Schritt 3.

5) Inkrementiere t.

6) Wenn $t < t_{\max}$ (oder eine max. Anzahl von Epochen noch nicht erreicht ist), gehe zu Schritt 2.

□

9.6 Nicht-stationäre Erweiterungen

Obige Verfahren funktionieren mit *stationären* Daten (*stationäre Zellstrukturen*). Die Idee, bei *nicht-stationären* Datenmengen ein GNG einzusetzen, bei dem das Ausfügen von Knoten möglich ist, wird in [Fri97] (GNG-U, Abk. für: GNG with *utility criterion*) oder auch in [Ham01] aufgegriffen. Dies kann man als *nichtstationäre* Zellstrukturen bezeichnen.

Zu dieser Problematik existieren weitere Ansätze, z.B. die *ART-Netze* (Abk. für: Adaptive Resonanztheorie) [Gro76]. Eine Weiterentwicklung ist eine Verknüpfung mit der Fuzzy-Technologie [CGR91] zum *Fuzzy-ART-Netz* und in [Kas93] zum *Simplified Fuzzy-ART-Netz*. Da sich z.B. Patientenkollektive durch veränderte Lebensbedingungen und Behandlungsmethoden im Laufe der Jahre verändern können, kann der Einsatz von nichtstationären Zellstrukturen beispielsweise in einer diagnostischen Anwendung lohnen, die *dauerhaft* eingesetzt werden soll. Eine Anwendung auf die Daten einer Genexpressionsanalyse einer Fuzzy-ARTMAP-Variante ist in [Azu00] beschrieben. Dort werden „DLBCL"-Patienten (Abk. für: diffuse large B-cell lymphoma) identifiziert. Ein ARTMAP ist ein überwacht arbeitendes Verfahren, bei dem zwei unüberwacht arbeitende ART-Netze verschaltet werden.

Das Lernen einer selbstorganisierenden Karte ist mittlerweile zu einem Standardverfahren zur Datenrepräsentation geworden. Üblicherweise wird eine zweidimensionale Karte mit rechteckiger Topologie verwendet. Die U-Matrix-Methode ist eine Möglichkeit eine gelernte Netzstruktur zu visualisieren. Einige Varianten sind die überwacht lernende Vektorquantisierung, die Wachsenden Zellstrukturen und das Neuronale Gas.

Kapitel 10

10 Regelgenerierung mit Kohonen-Netzen

10

10 Regelgenerierung mit Kohonen-Netzen

Kohonen-Karten in n Dimensionen passen sich unüberwacht einem $m > n$ dimensionalen Datenraum an. Hat man die Dimension der Kohonen-Karte geeignet gewählt und eine geeignete Anzahl von Knoten gefunden, so kann man z.B. mit der U-Matrix-Methode Datencluster visualisieren, insbesondere falls $n = 2$ ist. Ganz natürlich stellt sich die Frage, ob man nicht für die Zugehörigkeit von Daten zu einem Cluster Regeln angeben kann.

Lehrziele:
- Einen Überblick über die Sig*-Methode bekommen,
- eine Signifikanzmatrix berechnen können,
- die vorgestellte Regelgenerierungsmethode in den bisherigen Kontext stellen können.

10.1 Sig*-Methode

10.1

An dieser Stelle wird die **sig*-Methode** nach [Ult91, Abschnitte 7.5, 7.6] wiedergegeben, die solch eine Regelmenge angibt.

Algorithmus 10.1 (Sig*-Methode) 10.1

0.) Datenvorverarbeitung: Transformation der Daten auf eine (0,1)-Standardnormalverteilung (falls das durch eine nichtlineare Transformation möglich ist).

1.) Trainieren einer 2-dimensionalen selbstorganisierenden Karte.

2.) Anwenden der U-Matrix-Methode. Man erhält (evtl.) eine Einteilung in k Cluster $C_1, \ldots, C_k$. Welche Variablen sind nun für die Einteilung notwendig?

3.) Berechnung einer **Signifikanzmatrix** $\mathrm{sig}(C_k)$ des Clusters C_k: $\mathrm{sig}(i,j)$ ist eine (empirische) Kennzahl für die Wichtigkeit des Unterschieds der i-ten Variable des Clusters C_k zur i-ten Variable des Clusters C_j.

Das Zeilenmaximum beschreibt, für welches Cluster eine Variable am markantesten ist. Das Spaltenmaximum beschreibt, für welche Variable ein Cluster am markantesten ist.

Die Kennzahl zur Unterscheidung kann z.B. der Mittelwert oder die Standardabweichung der Daten o.ä. sein.

4.) Berechne für alle Cluster C_k eine **markierte Signifikanzmatrix** $\text{sig}^*(C_k)$ durch Markierung aller Zeilen- und Spaltenmaxima in $\text{sig}(C_k)$.

5.) Wir bezeichnen als $\text{sigl}^*_{C_l}(C_k)$ die **Signifikanzliste** des Clusters C_k bez. des Clusters C_l die gemäß ihrer Größe geordnete k-te Spalte von $\text{sig}^*(C_k)$.

Die Signifikanzliste $\text{sigl}^*C_l(C_k)$ enthält also Variablen $V_1, \ldots, V_n$, die gemäß ihrer Wichtigkeit für eine Unterscheidung des Clusters C_l vom Cluster C_k geordnet und evtl. markiert ist. Aus Symmetriegründen gilt: $\text{sigl}^*_{C_l}(C_k) = \text{sig}^*_{C_k}(C_l)$.

6.) Generierung von Zuordnungsregeln: Aus (einer Teilmenge der) Variablen $V_1, \ldots, V_n$ kann mit Bedingungen $\text{Cond}_1, \ldots, \text{Cond}_n$ an die Wertebereiche der Variablen eine Regel der Form „WENN $\text{Cond}_1(V_1)$ UND ... UND $\text{Cond}_r(V_r)$ DANN Cluster C“ erzeugt werden.

□

Diese Regel wird **Zuordnungsregel** genannt, falls a) nur *relevante Variablen* in der Regel vorkommen und b) nur *entscheidungsrelevante Bedingungen Cond* verwendet werden, d.h. Bedingungen, die auch tatsächlich Cluster unterscheiden. Wie kann man dies erreichen?

Zu a) Man kann die Komponenten mit dem größten Informationsgehalt (vgl. Kap. 20) wählen. Es wird nur auf Minimierung der Komponenten, aber nicht auf diagnostische Sicherheit geachtet. Alle Variablen zuzulassen ist nicht sinnvoll, da überflüssige Variablen die Lesbarkeit der Regeln erschweren. Ultsch schlägt eine (heuristische) Methodik vor, die auf den Signifikanzlisten beruht. Eine Menge R von Variablen ist die Menge der relevanten Variablen genau dann wenn: i) R enthält die ersten s Variablen aus der Signifikanzliste $\text{sigl}^*_{C_l}(C_k)$ (die mit den größten Werten, wobei eine Schwelle festgelegt werden kann, wieviel Prozent der summierten Signifikanz erreicht werden sollen) und ii) R enthält alle markierten Komponenten aus $\text{sigl}^*_{C_l}(C_k)$.

Zu b) Als entscheidungsrelevante Bedingung wird das Enthaltensein eines Variablenwertes im **Normalbereich** $N(V)$ angesehen, wobei man mit W

die vorkommenden Werte der Variablen V bezeichnet und definiert:

$$N(V) := [\text{mean}(W) - 2 \cdot \text{std}(W), \text{mean}(W) + 2 \cdot \text{std}(W)] \quad (54)$$

In $N(V)$ liegen bei einer $(\text{mean}(W), \text{std}(W))$-Normalverteilung 95% aller Daten. Das ist als heuristische Festlegung für die Entscheidungsrelevanz von Bedingungen anzusehen.

Wir erhalten nun die Zuordnungsregeln nun wie folgt: Sei 0 der Nullvektor oder ein ausgezeichnetes Cluster (z.B. „normal"). Für jedes Cluster C_i generiere die relevanten Variablen und entscheidungsrelevanten Bedingungen zum Erhalt von Regeln aus den Signifikanzlisten $\text{sigl}^*{}_0(C_i)$.

10.2 Diskussion

Bevor wir ab dem Kapitel 12 die Fuzzy-Theorie beleuchten, diespeziell zur Extraktion sprachlicher Regeln nützlich ist, fassen wir noch einmal kurz zusammen, was wir speziell über Regelgenerierung (neben all dem anderen Stoff) bisher gelernt haben:

- A-posteriori-Regelextraktion aus FF-Netzen (Kap. 4),
- Regelgenerierung mit RBF-Netzen (Kap. 7) und
- Regelgenerierung unter Verwendung von Kohonen-Netzen und der U-Matrix-Methode in diesem Kapitel.

Diese Ansätze zur Regelgenerierung sind heuristisch, im Fall der A-priori-Regelextraktion aus FF-Netzen aufwändig und im Fall der SOM nur möglich für Verteilungen, die auf Normalverteilungen transformierbar sind. Wir sehen später, dass die Inhalte der Fuzzy-Theorie uns weitere Möglichkeiten der Regelgenerierung eröffnen.

Kapitel 11

11

Kohonen-Netze in der Bioinformatik

11

11 Kohonen-Netze in der Bioinformatik

Kohonen-Netze können für die gleichen Anwendungen verwendet werden, die bereits bei den überwachten Lernverfahren Backpropagation und RBF-Lernen genannt wurden. Da das Kohonen-Lernen unüberwacht ist, kann man allerdings keine Klasseninformation zum Lernen verwenden. Man hat aber auch nicht immer eine Klasseneinteilung zur Verfügung. Ist sie unbekannt, versucht man sie gerade durch eine SOM-Clusterung und -Visualisierung zu finden.

Lehrziele:
- Wissen, wie man mit einer SOM clustern kann, was anhand des Beispiels der Clusterung von Aminosäuren einführend dargestellt wird,
- Anwendungen der SOM im Drug Design kennen wie den Einsatz zum virtuellen Screening,
- Anwendung der SOM zur Clusterung von Genexpressionsdaten verstehen,
- die wesentlichen Ideen weiterer Anwendungen nachvollziehen können.

11.1 Clusterung von Aminosäuren

Als einführende Anwendung geben wir die Clusterung der Aminosäuren nach [WM00, Kap. 4.2] wieder. Anhand der vier Eigenschaften Masse, Hydrophobizität, Oberfläche, Drehneigung wird eine (10×10)-Karte unüberwacht trainiert. Die gefundenen Cluster enthalten die folgenden Aminosäuren, vgl. Abb. 11.1: {W,Y}, {A,G,P,S,T}, {D,E,H,K,N,Q,R} und {C,F,I,L,M,V}. Üblicherweise definiert man die Austauschgruppen (engl.: exchange groups) als {H,R, K}, {D,E,N,Q}, {S,T,P,A,G}, {M,I,L,V}, {F,W,Y}, {C}. Man sieht, dass in der SOM die ersten beiden Austauschgruppen zusammen liegen. F und C liegen anders.

Zur Beschreibung der 20 Aminosäuren wurden in [SW98] 7-dimensionale Eigenschaftsvektoren verwendet, um eine (4×4)-Karte zu trainieren, vgl. Abb. 11.2. Wie in der Abb. 11.1 sieht man, dass die Elemente der Austauschgruppen benachbart sind, wobei hier H, N und Q auf einen Knoten abgebildet wurden.

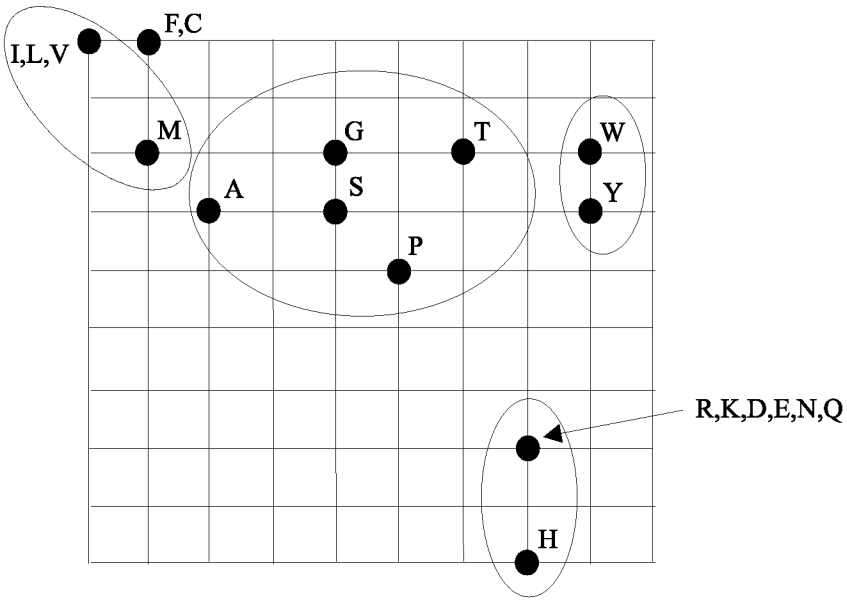

Abbildung 11.1. Aminosäuren auf dem SOM-Gitter und zugehörige Cluster. Nach [WM00, Abb. 4.7]

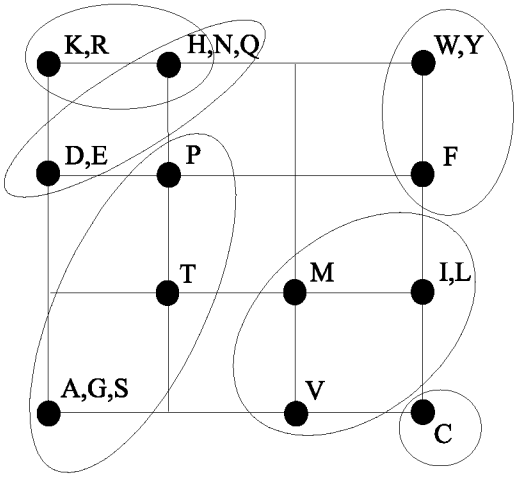

Abbildung 11.2. Aminosäuren auf dem SOM-Gitter und zugehörige Cluster. Nach [SW98]

11.2 Anwendungen im Drug Design

Im Folgenden werden Arbeiten aus dem Gebiet „Drug Design“ beschrieben. Versucht wird Eigenschaften von Stoffen vorherzusagen, um deren Wirkung einschätzen zu können.

Als erste Anwendung präsentieren wir die Clusterung von HIV-Oktapeptiden [YC03]. Um Medikamente gegen AIDS zu entwickeln, ist es u.a. wichtig Inhibitoren für HIV-Protease zu konstruieren, deren Spaltungskapazität man vorhersagen kann. Zu diesem Zweck werden 362 HIV-Oktapeptide zum Training einer (6×6)-SOM verwendet, 114 mit Spaltungsaktivität, 248 ohne. Erstere sind wichtig für das Drug Design. Mit einer BIN20-Kodierung erhält man einen (8 mal 20 gleich) 160-dimensionalen Trainingsvektor pro HIV-Oktapeptid.

Es wird angenommen, dass jeder Cluster ein gemeinsames Motiv für die entsprechenden Oktapeptide enthält. Deshalb wird mit konventionellen Alignment-Methoden ein solches für jedes Cluster gesucht. Die Motive werden als Regeln zur Klassifikation unbekannter Daten verwendet. Beispielsweise ist das Motiv SQNYPXVQ in 21% aller Daten enthalten. 73% dieser Daten sind Spaltungs-aktiv. Die so erhaltene Klassifikation hat Vorteile gegenüber einer mit Entscheidungsbäumen durchgeführten. Insbesondere wenn man Mutationen (unter Berücksichtigung der Dayhoff-Matrix) vornimmt, werden die Daten immer noch gut klassifiziert, d.h. die Klassifikation ist robust gegenüber Mutationen.

Im Folgenden werden verschiedene Anwendungen der Kohonen-Karten im Drug Design vorgestellt.

In [Sch00] werden 5000 Drug- und 5000 Non-Drug-Moleküle in einer Deskriptorvektor-Darstellung als Eingabe einer geschlossenen (10×10)-Karte (Torus) verwendet. Verschiedene Grautöne sind mit dem prozentualen Anteil der Drugs assoziiert, d.h. jedem der 100 Knoten ist ein Grauton zugewiesen, je nach Anteil der Drugs unter denen der Knoten aktiviert ist. Die Anwendung dient der Erhöhung der hit rates im Screening-Prozess. Ein Tool zur Visualisierung und Analyse von Drug-Daten wird in [GDWS03] beschrieben.

Eine Unterteilung von Antidepressiva und anderen Medikamenten wird in [SSS03, Kap. 2] präsentiert.

Auch in [SN03] wird eine SOM für die Optimierung des virtuellen Screenings verwendet, eine (8×8)-Karte. Als Target wird ein menschlicher A_{2A}-Rezeptor betrachtet. Trainiert wurde mit kombinatorisch erzeugten, virtuellen Produkten, und zwar mit P_1-Rezeptor Antagonisten, die zur Klasse der GPCR-gekoppelten Rezeptoren gehören (GPCR ist Abk. für: G-protein coupled receptor).

Eine Reihe von Anwendungen findet man in [SW98]: a) Eine (10×10)-Karte von N-terminalen eukaryotischen Proteinsequenzen wird nach einem Kohonen-Training hauptsächlich in zytoplastisch, sekretorisch und mitochondrial eingeteilt. b) Für die über 33000 Stoffe in der MedChem-Datenbank wird eine (7×7)-Karte für Thrombininhibitoren trainiert. c) Kann man die molekulare Diversität von Molekülen abbilden, so kann man leichter Moleküle mit bestimmten Eigenschaften auffinden. Zu diesem Zweck wurde eine (25×25)-Karte trainiert.

In [MVJB03] werden 568 Stoffe betrachtet, um deren Toxizität zu untersuchen und vorherzusagen (mit einer (25×25)-Karte).

11.3 SOM und U-Matrix-Methode zur Genexpressionsanalyse

In [Kas01] wird eine SOM zur Clusterung von Genexpressionsdaten aus DNA-Mikroarrays eingesetzt. Als Indikator für die Aktivität eines Gens wird das Auftreten von Transport-RNA jedes Gens gemessen. Hier wurde die Expression von 1551 Genen der Hefe Saccharomyces arevisae verwendet. Mittels der U-Matrix-Methode wurde die SOM visualisiert. Anschließend wurden noch ähnliche Daten auf ähnliche Farben projiziert. Vergleiche hierzu auch die Webseiten unter www.cis.hut.fi/sami/wsom01.

11.4 Projektion eines 3D-Protein-Modells mittels SOM

Um die Struktur eines Proteins zu erkennen, wird ein 3D-Modell benutzt, dass man im Raum rotieren kann. Dabei findet man nicht unbedingt eine ideale Ansicht des Proteins. In [SM03] wird eine Abbildung vorgeschlagen, die ein 3D-Modell auf die 2D-Ebene abbildet. Als Eingabe für eine Kohonen-Karte werden 3D-Koordinaten der rückseiten Kohlenstoff-Alpha (CA)-Atome der Aminosäurenresiduen verwendet. Die Karte enthält etwa so viele Knoten wie Residuen im Protein. Die Höhe und die Breite des Gitters wird durch das Verhältnis der ersten beiden Hauptkomponenten bestimmt. 20 verschiedene

Abbildungen wurden trainiert, die beste nach einem Abstandskorrelations-Kriterium ausgewählt. Verschiedene Rotationen wurden verwendet.

11.5 Abbau von PCB

11.5

In [Car02] wird ein Kohonen-Lernen zur Beurteilung des Abbaus von Polychlorinierten Biphenylen (PCB) unter Verwendung der Bakterien Aspergillus Niger präsentiert. Dies dient der Entlastung der Umwelt. Dazu wurden 44 PCB's verwendet, und deren mikrobiologische Abbaurate untersucht. Zehn Dimensionen bezeichnen die Positionen der Chloratome. (n×n)-Karten wurden für $n = 4, \ldots, 12$ verwendet. Die Vorhersage erreicht (nur) eine Genauigkeit von $\pm 25\%$.

11.6 Weitere Anwendungen

11.6

In [ZG93] werden Kohonen-Netze auch für die folgenden Anwendungen trainiert: Herkunftsklassifikation von Olivenölen, Prognose der Bindungsaktivität, Bestimmung des Zusammenhangs zwischen Infrarot-Spektrum und der Struktur (vgl. auch Kap. 5) und Projektion des (3D-)elektrostatischen Potenzials auf eine (2D-)Kohonen-Karte.

Aktivierungselement: Schauen Sie sich bei Interesse die eine oder andere Originalarbeit an.

In diesem Kapitel haben wir einige typische Anwendungen der Kohonen-Karten in der Bio- und Chemieinformatik kennengelernt. Insbesondere Genexpressionsdaten oder kodierte Moleküldaten können mit einer SOM geclustert und visualisiert werden. Regionen aktiver Gene bzw. bioaktiver Moleküle sollen damit visualisiert werden.

Kapitel 12

12 Fuzzy-Mengenlehre

12

12 Fuzzy-Mengenlehre

Wie auch schon bei den NN geben wir zu Beginn des neuen Teils Literatur zur Fuzzy-Theorie (Abk.: FT) an. Anschließend führen wir ausgehend von der Fuzzy-Menge die wichtigsten Begriffe der FT ein, die man zum Arbeiten in diesem Gebiet benötigt.

Lehrziele:
- Eine Übersicht über wichtige Literatur zur Fuzzy-Wissenschaftswelt erhalten,
- genau wissen, wie Fuzzy-Mengen definiert sind,
- die wichtigsten Begriffe aus der Fuzzy-Theorie wie Zugehörigkeitsfunktion und Zugehörigkeitsgrad kennen,
- die wichtigsten Zugehörigkeitsfunktionen wie Dreiecks- und Trapezfunktion verwenden können,
- mit Fuzzy-Mengenoperationen umgehen können und wissen, dass diese auf den t-Normen und t-Conormen basieren,
- Fuzzy-Relationen und Fuzzy-Kompositionen verstehen,
- die grundlegende Idee einer Fuzzy-Datenbank kennen.

12.1 Literaturübersicht

Wir teilen die Literaturübersicht in Konferenzen, Zeitschriften und Lehrbücher auf.

Konferenzen:

- International Conference on Fuzzy Systems and Knowledge Discovery (FSKD),
- IEEE International Conference on Fuzzy Systems (FUZZ-IEEE),
- International Conference of the North American Information Processing Society (NAFIPS),
- International Conference on Computational Intelligence (Fuzzy Days) in Dortmund,
- European Symposium on Intelligent Technologies, Hybrid Systems and their Implementation on Smart Adaptive Systems (EUNITE) von 2001 bis 2004,
- IASTED International Conference on Neural Networks and Computational Intelligence,

- International Conference in Fuzzy Logic and Technology (EUSFLAT),
- International Conference on Artificial Intelligence and Soft Computing (ICAISC).

Zeitschriften:

- Journal of Intelligent and Fuzzy Systems,
- Fuzzy Sets and Systems,
- Fuzzy Optimization and Decision Making,
- The Journal of Fuzzy Mathematics,
- IEEE Transactions on Fuzzy Systems.

Lehrbücher:

- Zimmermann [Zim91],
- Bothe [Bot99],
- Kruse/Gebhardt/Klawonn [KGK93],
- Aliev [ABA00],
- Goos [Goo98],
- Kosko [Kos97],
- Springer-Reihe: Studies in Fuzziness and Soft Computing.

12.2

12.2 Fuzzy-Mengen

Wie bereits im Kapitel 2 angedeutet, behandelt die FT die Modellierung unscharfen Wissens. Das Wissen kann vom Experten in reinen *Fuzzy-Systemen* vorgegeben werden oder aber auch maschinell adaptiert werden, z.B. durch neuronale Netze in *Neuro-Fuzzy-Systemen.*

Zu Beginn geben wir eine Definition der *scharfen* und *unscharfen* Menge sowie einiger wichtiger Begriffe der Fuzzy-Mengenlehre. Natürlich hofft der Autor, den Leser initial für dieses Themengebiet „fuzz-zinieren" zu können.

12.1

Definition 12.1 (Grundbegriffe der Fuzzy-Theorie)
a) Eine Menge $M \subset X$ heißt **scharf** oder **naiv**, falls für alle Elemente $x \in X$ gilt: $x \in M$ oder $x \notin M$.

b) Eine Menge M heißt **unscharf** oder **Fuzzy-Menge**, falls jedem $x \in M$ eine **Zugehörigkeit** (engl.: membership) $\mu_M(x) = \mu(x) \in [0,1]$ zugeordnet ist. Die Funktion, die den x die Zugehörigkeit zuordnet, heißt **Zugehörigkeitsfunktion**. Wir definieren weiter

c) den **Träger** (engl.: support) von M als $\{x \in M | \mu_M(x) > 0\}$,

d) den α**-Schnitt** (engl.: α-cut) von M als $\{x \in M | \mu_M(x) > \alpha\}$ und

e) den **Kern** (engl.: core) von M als $\{x \in M | \mu_M(x) = 1\}$.

□

Auf scharfen Mengen kennen wir folgende Operationen: $\subset, =, \cap, \cup, \bar{}, \setminus$. Es gelten die bekannten Gesetze: Kommutativität und Assoziativität für $\cap, \cup$; Distributivität; Idempotenz von $\cap, \cup$ (z.B. $M \cup M = M$); Identitäten (z.B. $M \cup \emptyset = M$); Transitivität von $\subset$; Involution ($\bar{\bar{M}} = M$), Kontradiktion ($M \cap \bar{M} = \emptyset$) und die de Morganschen Gesetze.

Aktivierungselement: Wiederholen Sie die mengentheoretischen Aussagen (und bei Interesse auch deren Beweise) für die scharfen Mengen.

Man kann die Mengenoperationen unter Verwendung der **charakteristischen Funktion**

$$\chi_M : \ M \to \{0,1\}, \ \chi_M(x) := \begin{cases} 1, & x \in M \\ 0, & x \notin M \end{cases} \tag{55}$$

definieren. Es gelten die Beziehungen:

$$\chi_{M \cap N}(x) = \min\{\chi_M, \chi_N\} \ , \tag{56}$$

$$\chi_{M \cup N}(x) = \max\{\chi_M, \chi_N\} \ , \tag{57}$$

$$\chi_{\bar{M}}(x) = 1 - \chi_M \ , \tag{58}$$

$$M = N \Leftrightarrow \forall x \in M \ : \ \chi_M(x) = \chi_N(x) \ , \tag{59}$$

$$M \subset N \Leftrightarrow \forall x \in M \ : \ \chi_M(x) \leq \chi_N(x) \ . \tag{60}$$

Wir sehen später in den Formeln (56) bis (60) eine Analogie zur Fuzzy-Mengenlehre.

Das Problem naiver (scharfer) Mengen ist die Modellierung des unscharfen, ungenauen, vagen Wissens, z.B. der Aussage „Gehalt IST hoch“. In dieser Aussage kann man nicht genau sagen, welches Gehalt hoch ist und welches nicht. Die exakte Höhe des Gehalts ist nicht feststellbar.
Bei der Schreibweise einer Fuzzy-Menge unterscheiden wir den diskreten Fall

$$M = \{(x_1, \mu(x_1)), \ldots, (x_n, \mu(x_n))\} = \sum_{i=1}^{n} \frac{\mu(x_i)}{x_i} \tag{61}$$

und den stetigen Fall

$$M = \left\{ \int_{x \in M} \frac{\mu_M(x)}{x} \right\} . \tag{62}$$

12.2 **Beispiel 12.2**
Wir geben ein Beispiel für die Gehälter an einer Universität. Wir modellieren die Zugehörigkeit zu einem hohen Gehalt durch die oben genannte diskrete Aufzählungsform: (Professor, 0.7), (Wissenschaftlicher Mitarbeiter, 0.3), (Studentische Hilfskraft, 0.0).

□

Wie „fuzzy“ eine Menge ist kann mit einer Berechnung entschieden werden. Eine Möglichkeit hierzu wird in [Pae03b] präsentiert. Eine diskrete Menge hat dort den Fuzziness-Index 0 und eine kontinuierliche Menge den Index 1. Fuzzy-Mengen haben einen Index $\in]0, 1[$.

Wir definieren nun drei wichtige (eindimensionale) Zugehörigkeitsfunktionen, die in der Abb. 12.1 skizziert sind:

12.3 **Definition 12.3 (Zugehörigkeitsfunktionen)**
a) **Dreiecksfunktion:**

$$\mu(x) := \begin{cases} 1 & , \text{ x=b} \\ \frac{x-a}{b-a} & , \ x \in]a, b[\\ 1 - \frac{x-b}{c-b} & , \ x \in]b, c[\\ 0 & , \text{ sonst} \end{cases} \tag{63}$$

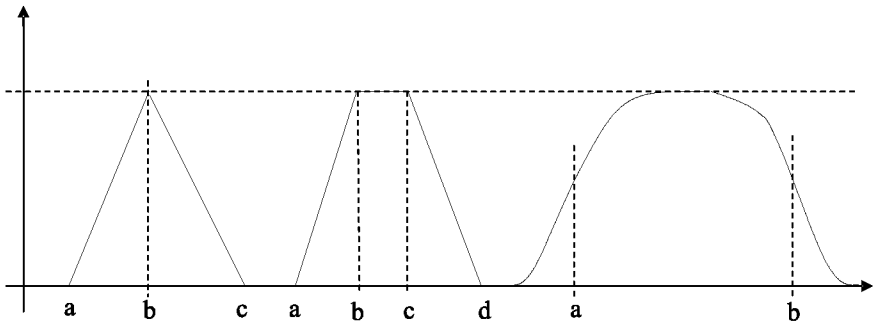

Abbildung 12.1. Dreiecks-, Trapez- und Π-Funktion

b) **Trapezfunktion:**

$$\mu(x) := \begin{cases} 1 & , \ x \in [b,c] \\ \frac{x-a}{b-a} & , \ x \in]a,b[\\ 1 - \frac{x-c}{d-c} & , \ x \in]c,d[\\ 0 & , \ \text{sonst} \end{cases} \tag{64}$$

Die Dreiecksfunktion ist ein Spezialfall der Trapezfunktion.

c) **Π-Funktion:**

$$\mu(x) := \begin{cases} 1 & , \ x \in [a+\delta, b-\delta] \\ f_S(x) & , \ x \in]a-\delta, a+\delta[\\ 1 - f_S(x) & , \ x \in]b-\delta, b+\delta[\\ 0 & , \ \text{sonst} \end{cases} \tag{65}$$

Der Punkt a ist Wendepunkt der Funktion, δ ist der Abflachungsparameter. Die Π-Funktion kann auch asymmetrisch definiert werden, wenn man verschiedene Abflachungsparameter verwendet. Die Funktion f_S ist definiert als:

$$f_S(x) := \begin{cases} 0 & , \ x \leq a-\delta \\ 2(\frac{(x-a+\delta)}{2\delta}) & , \ x \in]a-\delta, a[\\ 1 - 2(\frac{(a-x+\delta)}{2\delta}) & , \ x \in]a, a+\delta[\\ 1 & , \ x \geq a+\delta \end{cases} \tag{66}$$

□

12.3

12.3 Fuzzy-Mengenoperationen

Um die Mengenoperationen auf Fuzzy-Mengen zu übertragen, führen wir die Begriffe *t-Norm* und *t-Conorm* (*s-Norm*) ein (mit $I = [0, 1]$ als Einheitsintervall).

12.4

Definition 12.4

a) Eine Abbildung $\Delta : \ I \times I \to I$ heißt **t-Norm** genau dann, wenn für alle x, y, w, z gilt:

a1) Monotonie: $x \leq y, \ \ w \leq z \Rightarrow x \Delta w \leq y \Delta z$,

a2) Kommutativität: $x \Delta y = y \Delta x$,

a3) Assoziativität: $(x \Delta y) \Delta z = x \Delta (y \Delta z)$,

a4) Identitäten: $x \Delta 0 = 0$ und $x \Delta 1 = x$.

b) Eine Abbildung $\nabla : \ I \times I \to I$ heißt **t-Conorm** genau dann, wenn für alle x, y, w, z gilt: a1), a2), a3) und a4') $x \nabla 0 = x$ und $x \nabla 1 = 1$.

□

Man kann ∇ durch $\triangle$ definieren als: $x \nabla y := 1 - (1 - x) \Delta (1 - y)$. In der nächsten Definition wollen wir Fuzzy-Mengenoperationen so einführen, wie es üblicherweise in der Literatur geschieht.

12.5

Definition 12.5 (Fuzzy-Mengenoperationen)

a) Die **Vereinigung** V von Fuzzy-Mengen M, N kann definiert werden als $V := M \cup N$ mit „max“ als t-Conorm, d.h. $\mu_V(x) := \max\{\mu_M(x), \mu_N(x)\}$.

b) Der **Durchschnitt** D von Fuzzy-Mengen M, N kann definiert werden als $D := M \cap N$ mit „min“ als t-Norm, d.h. $\mu_D(x) := \min\{\mu_M(x), \mu_N(x)\}$.

c) Das **Komplement** einer Fuzzy-Menge M kann definiert werden als $\bar{M}$ mit $\mu_{\bar{M}}(x) := 1 - \mu_M(x)$.

d) Man nennt M und N **disjunkt** genau dann, wenn $\mu_{M \cap N} = 0$.

□

Aktivierungselement: Skizzieren Sie Def. 12.5 a), b), c) unter Verwendung zweier sich überlappender Zugehörigkeitsfunktionen.

Aktivierungselement: Eine andere t-(Co)Norm ist: $\mu_{M\cap N}(x) := \mu_M(x)\mu_N(x)$ bzw. $\mu_{M\cup N}(x) := \mu_M(x) + \mu_N(x) - \mu_M(x)\mu_N(x)$. Überprüfen Sie das.

Es gilt

$$M = N \Leftrightarrow \forall x: \quad \mu_M(x) = \mu_N(x) \ , \tag{67}$$

$$M \subset N \Leftrightarrow \forall x: \quad \mu_M(x) \leq \mu_N(x) \ . \tag{68}$$

Hier wird die Analogie zu den charakteristischen Funktionen der scharfen Mengen deutlich.

12.4 Fuzzy-Relationen

12.4

Nun wollen wir die Begriffe der *Relation* und der *Komposition* innerhalb der naiven Mengenlehre auf die Fuzzy-Mengenlehre übertragen.

Definition 12.6 (Relation und Komposition) 12.6

a) Eine **Relation** R ist definiert als eine Teilmenge des Kreuzprodukts

$$R \subset \times_{i=1}^{n} M_i := \{(x_1, \ldots, x_n) | x_1 \in M_1, \ldots, x_n \in M_n\} \ . \tag{69}$$

b) Die **Komposition** zweier Relationen R, S ist definiert als

$$R \circ S \subset U \times W \text{ mit } (u,w) \in R \circ S :\Leftrightarrow \exists v \in V: \quad (u,v) \in R \wedge (v,w) \in S \ . \tag{70}$$

□

Definition 12.7 (Fuzzy-Relation und Fuzzy-Komposition) 12.7

a) Eine **Fuzzy-Relation** R ist gegeben durch

$$\mu_{M_1 \times \ldots \times M_n}(x_1, \ldots, x_n) := \min\{\mu_{M_1}(x_1), \ldots, \mu_{M_n}(x_n)\} \ . \tag{71}$$

b) Seien $R \subset X \times Y$ und $S \subset Y \times Z$ zwei Fuzzy-Relationen. Die **Fuzzy-Komposition** ist gegeben durch

$$\mu_{R \circ S}(x, z) := \sup_{y \in Y} \min\{\mu_R(x, y), \mu_S(y, z)\} \ . \tag{72}$$

□

In der Formel (72) existiert das Minimum zweier Werte. y kann unendlich viele Werte annehmen, so dass „sup" verwendet werden muss. Wir verwenden die eingeführten Begriffe in den nächsten Abschnitten.

In einer Datenbank, die die Fuzzy-Terminologie unterstützt, könnte eine SQL-Abfrage z.B. modifiziert lauten:

select Gehalt
from Table
where Membership(x) $\geq \mu$

oder

select Gehalt
from Table
where Gehalt = hoch

Eine andere Abfrage könnte lauten:

select Sequenz
from Sequence_table
where Sequenz IST sehr ähnlich

Nachdem man die Definition der Fuzzy-Menge verinnerlicht hat, wird man besser verstehen, dass viele Sachverhalte der realen Welt nicht eindeutig sind. Z.B. ist die Ähnlichkeit zweier Sequenzen nicht eindeutig definiert. Zwei Sequenzen sind nicht entweder ähnlich oder unähnlich, sondern sie können ähnlicher oder unähnlicher zueinander sein. Dies wird im Abschnitt über Anwendungen der FT in der Bioinformatik modelliert werden. Die wichtigsten Begriffe, insbesondere den der Zugehörigkeitsfunktion sollte man gut beherrschen, um sich später einen leichten Zugang zu den Fuzzy-Systemen zu ermöglichen. Abschließend sei noch erwähnt, dass der Vordenker Lotfi A. Zadeh

bereits 1969 über die Anwendung der FT in den Lebenswissenschaften nachgedacht hat [Zad69].

Kapitel 13

13 Das Extensionsprinzip und Fuzzy-Zahlen

13

13 Das Extensionsprinzip und Fuzzy-Zahlen

Wir erklären in diesem kurzen Kapitel zuerst das *Extensionsprinzip*. Der Sinn ist die Übertragung bekannter Theorien, die auf der naiven Mengenlehre beruhen, auf entsprechende Fuzzy-Theorien. Als Beispiel werden wir kurz erläutern, wie man Zahlen zu Fuzzy-Zahlen machen kann.

Lehrziele:
- Den Sinn und Zweck des Extensionsprinzips verstehen, insbesondere die Verwendung der sup- und min-Operatoren.
- die Anwendung des Extensionsprinzips zur Definition der Fuzzy-Zahlen verstehen.

13.1 Das Extensionsprinzip

13.1

Das Extensionsprinzip ist ein zentrales Prinzip in der Fuzzy-Theorie, das festlegt, wie man Fuzzy-Mengen durch eine Abbildung transformiert.

Definition 13.1 (Extensionsprinzip) 13.1

Seien $M_1, \ldots, M_n$ Fuzzy-Mengen (die durch die scharfen Mengen $U_1, \ldots, U_n$ definiert sind). Sei $f : U \to V$ eine Abbildung von U nach V (scharfe Grundmengen). Wir geben an, welche Fuzzy-Menge eine Fuzzy-Menge, die über U definiert ist, als Bild hat. Sei dazu N die entsprechende Fuzzy-Menge über dem Universum V. Für $y \in V$ sei

$$\mu_N(y) = \sup_{(x_1,\ldots,x_n)\in f^{-1}(y)} \min\{\mu_{M_1}(x_1), \ldots, \mu_{M_n}(x_n)\} \ . \tag{73}$$

□

Man sagt, N **erweitere** die Fuzzy-Mengen $M_1, \ldots, M_n$ in V. Bei diskreten Mengen kann „sup“ durch „max“ ersetzt werden.

13.2 Fuzzy-Zahlen

13.2

Ohne hier genau auf die Axiome einzugehen, die für Fuzzy-Zahlen gelten sollen, betrachten wir direkt die Operationen $\sharp \in \{+, -, \cdot, /\}$ für Fuzzy-Zahlen

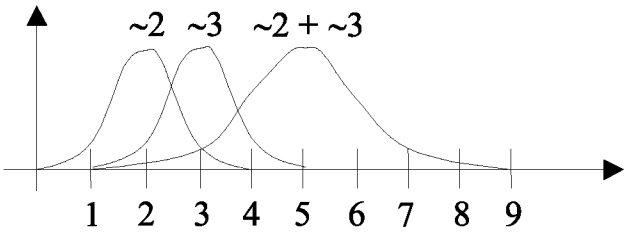

Abbildung 13.1. Addition zweier Fuzzy-Zahlen $\backsim 2$ und $\backsim 3$

$\backsim a$ und $\backsim b$, z.B. $\backsim 2$ und $\backsim 3$. Nach dem Extensionsprinzip gilt für alle $x, y, z \in \mathbb{R}$

$$\mu_{a \sharp b}(z) = \sup_{x \sharp y = z} \min\{\mu_a(x), \mu_b(y)\} \ . \tag{74}$$

Die Fuzzy-Operationen sind nicht leicht auszuführen; man muss ein Optimierungsproblem unter Nebenbedingungen lösen. Man beschränkt sich daher oft auf besonders einfach gebaute Zugehörigkeitsfunktionen.

Wir beschließen den Abschnitt mit der Skizze 13.1, in der zwei Fuzzy-Zahlen addiert werden und weisen auf eine Anwendung im Abschnitt 18.5 hin. Merken sollten Sie sich die Grundidee, wie man das Extensionsprinzip anwenden kann, um unscharfe Konzepte zu modellieren.

Kapitel 14

Fuzzy-Datenanalyse

14

14

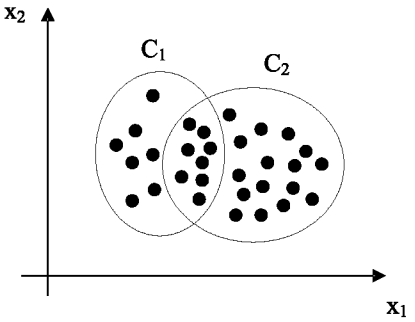

Abbildung 14.1. Zwei Fuzzy-Cluster in einer zweidimensionalen Datenmenge. Die Punkte in der Schnittmenge gehören zu beiden Clustern

14 Fuzzy-Datenanalyse

Grundsätzlich kann man mit Hilfe der Fuzzy-Theorie die Datenanalyse zur *Fuzzy-Datenanalyse* erweitern, indem man anstelle von scharfen Datenmengen unscharfe betrachtet.

Lehrziele:
- Die Idee der Übertragung des k-means-Clusterverfahrens zum Fuzzy-k-means-Clusterverfahren verstehen,
- die Idee des Fuzzy-k-nearest-neighbour-Verfahrens kennenlernen,
- wissen, dass viele Datenanalyseverfahren in einer Fuzzy-Variante existieren.

14.1 Fuzzy-Clusterung

14.1

Verwendet man die naive Mengenlehre als Grundlage, so kann ein Element x entweder einem Cluster C angehören oder nicht. Denkbar wäre aber auch, dass x zu mehreren Clustern gehört, die sich überlappen.

Bei einer **Fuzzy-Clusterung** gehört x nur mit einer bestimmten Zugehörigkeit zu einem oder mehreren Clustern, vgl. die Abb. 14.1. Ein bekanntes Verfahren ist das **Fuzzy-k-means-Verfahren**, das im nächsten Abschnitt erläutert wird. Anschließend geben wir noch einige Hinweise auf andere Datenanalysemethoden, die mit der Fuzzy-Theorie verknüpft wurden, aber für unser Anliegen nicht im Mittelpunkt stehen. Die Fuzzy-Klassifikation und -Regelgenerierung lernen wir in den nächsten Kapiteln kennen.

Beim (scharfen) k-means-Verfahren wird die Zielfunktion

$$z = \sum_{i=1}^{n} \sum_{j=1}^{k} \chi_{C_j}(x_i)(d_2(x_i, c_j))^2 \tag{75}$$

minimiert. Dabei sind die C_j Cluster und c_j die Zentren als geometrische Mittel von C_j. Beim Fuzzy-k-means-Verfahren wird

$$z = \sum_{i=1}^{n} \sum_{j=1}^{k} \mu_{C_j}(x_i)(d_2(x_i, c_j))^2 \tag{76}$$

minimiert. Die C_j sind die Fuzzy-Cluster, die c_j die Fuzzy-Zentren. Die k-ten Koordinaten der Fuzzy-Zentren c_j sind definiert durch

$$c_j^{(k)} := \frac{\sum_{i=1}^{n} \mu_{C_j}^m(x_i) x_i^{(k)}}{\sum_{i=1}^{n} \mu_{C_j}^m(x_i)} \quad . \tag{77}$$

Der Parameter m kontrolliert die Fuzziness. Sie sehen, dass die charakteristische Funktion χ in (75) durch die Zugehörigkeitsfunktion μ in (76) ersetzt wurde. Diesen Übergang haben Sie bereits im Kapitel 12 kennengelernt.

14.2

14.2 Fuzzy-Klassifikation

Neben der k-means-Clusterung haben wir im Kap. 2 auch die Idee der k-nearest-neighbour-Klassifikation kennengelernt. Ein Datentupel x wurde derjenigen Klasse zugeordnet, für die der Abstand zwischen x und dem Mittelwert der k nächsten Datentupeln der Klasse am geringsten ist. Da bei einer Fuzzy-Repräsentation der Daten jeder Datentupel x mit einem Zugehörigkeitsgrad μ_{K_j} zur Klasse K_j gehört, kann der Zugehörigkeitsgrad $\mu_{K_j}(x)$ durch die Zugehörigkeitsgrade der k nächsten Nachbarn $y_1, \ldots, y_k$ der jeweiligen Klasse K_j berechnet werden, siehe (78).

$$\mu_{K_j}(x) := \frac{\sum_{i=1}^{k} \mu_{K_j}(y_i) d(x, y_i)^m}{\sum_{i=1}^{k} d(x, y_i)^m} \tag{78}$$

Der Exponent in (78) gewichtet den Beitrag der Zugehörigkeitsgrade gemäß dem Abstand stärker oder schwächer. Als Anwendung der Fuzzy-k-nearest-neighbour-Klassifikation nennen wir die Vorhersage der Proteinzugänglichkeit in Lösung [SKL05].

14.3 Weitere Verfahren

Neben der Fuzzy-Datenanalyse existieren weitere Ansätze, FT mit Datenanalyseansätze zu verbinden. Wir werden einige dieser Ansätze erwähnen, verweisen aber im Rahmen dieses Buches auf die weiterführende Literatur.

Wir haben im Kapitel 2 die lineare Optimierung kennengelernt. In [RV02] wird beschrieben, wie man die Parameter eines solchen Optimierungsproblems durch Fuzzy-Mengen modellieren kann. In [IKW01] wird das Software-Tool „FuReA“ zur Fuzzy-Regressionsanalyse vorgestellt. Die Fuzzy-Regression wird in [HH03] mit dem Ansatz der Support-Vektor-Maschine verbunden. Einen Überblick über Fuzzy-Methoden in der Statistik gibt [Tah03]. Interessant ist die Idee, Nachbarschaften von Datenpunkten mit erweiterten Histogrammtechniken zu explorieren [BWP05].

Wir beschließen damit diesen kurzen Abschnitt über die Fuzzy-Datenanalyse mit dem Hinweis, dass in den nächsten Kapiteln in die wichtigsten Ziele der Fuzzy-Datenanalyse „Klassifikation“ und „Regelgenerierung“ ausführlich eingeführt wird. Im Kapitel 18 werden Anwendungsbeispiele in der Bioinformatik vorgestellt.

Kapitel 15

15

Fuzzy-Logik

15

15 Fuzzy-Logik

Eventuell werden Sie sich fragen, warum in den vorhergehenden Kapiteln von „Fuzzy-Theorie“ und nicht von „Fuzzy-Logik“ die Rede ist. Das Arbeitsgebiet der Logik behandelt insbesondere Schließmechanismen der Art: A ist wahr UND aus A folgt B DANN B ist wahr. Solche Schlussweisen kennen Sie aus der Aussagen- und Prädikatenlogik.

Bisher haben wir lediglich einige Begriffe und Sachverhalte aus der Fuzzy-Theorie kennengelernt, aber noch keine Schließmechanismen (auch *Inferenzmechanismen* genannt) der Art: A hat Zugehörigkeitsgrad μ_A UND aus A folgt B DANN B hat Zugehörigkeitsgrad μ_B. Diese Art von Inferenzschemata werden in der Fuzzy-Logik behandelt. Deshalb haben wir bislang allgemein von „Fuzzy-Theorie“ gesprochen.

Man kann den Zugehörigkeitsgrad mit dem *Wahrheitswert* einer Aussage identifizieren. Oft nähert der Zugehörigkeitsgrad den Wahrheitswert aber nur an.

Lehrziele:
- Die Mamdami-Inferenz als wichtigste Fuzzy-Inferenz kennen,
- die Definition der linguistischen Variable zur Bezeichnung von Fuzzy-Mengen und das dazugehörige Beispiel verstehen.

15.1 Fuzzy-Inferenz

An dieser Stelle wollen wir zunächst die **Fuzzy-Inferenz** erklären und den Zusammenhang mit Fuzzy-Regeln herstellen, die als Prämisse und/oder Conclusio Fuzzy-Terme besitzen. Wegen $(A \Rightarrow B) \Leftrightarrow (\bar{A} \vee B)$ kann man definieren

$$\mu_{A \Rightarrow B}(x, y) := \max\{1 - \mu_A(x), \mu_B(y)\} \ . \tag{79}$$

In der Praxis fordert man, dass $A \Rightarrow B$ wahr sein soll, wenn $\mu_A(x) \leq \mu_B(y)$ ist. B sollte wahrer sein, je besser A erfüllt ist. Deshalb verwendet man die sog. **Mamdami-Inferenz** (1976)

$$\mu_{A \Rightarrow B}(x, y) := \mu_{A \wedge B}(x, y) = \min\{\mu_A(x), \mu_B(y)\} \ . \tag{80}$$

Mit Hilfe der einfachen Mamdami-Inferenz lässt sich Wissen automatisch gut verarbeiten.

15.2

15.2 Linguistische Variablen

Als Grundlage der Fuzzy-Regeln und -Systeme, die im nächsten Abschnitt verwendet werden, führen wir sog. *linguistische Variablen* nach [Zim91] ein. Die linguistischen Variablen dienen dazu, sprachlich formulierte, vage Sachverhalte von Anwendungen zu modellieren.

15.1 **Definition 15.1 (Linguistische Variable)**
Eine **linguistische Variable** $\mathcal{L}$ kann formal definiert werden als $\mathcal{L} := (x, T(x), U, G, A)$ mit

- x: Name der linguistischen Variablen $\mathcal{L}$,
- $T(x)$: linguistische Terme, d.h. eine Menge von Bezeichnern $\{b_1, \ldots, b_n\}$ von Fuzzy-Mengen $\{B_1, \ldots, B_n\}$,
- U: Grundraum, über dem die Fuzzy-Mengen $\{B_1, \ldots, B_n\}$ definiert sind,
- G: syntaktisches Regelwerk für die Bezeichner $\{b_1, \ldots, b_n\}$, (i.d.R. eine formale Grammatik), um für den Menschen nur sinnvolle Bedeutungen zuzulassen und
- A: semantisches Regelwerk, dass jedem Bezeichner $\{b_1, \ldots, b_n\}$ seine Bedeutung (mittels Zugehörigkeitsfunktionen $\mu_{B_1}, \ldots, \mu_{B_n} : U \rightarrow [0, 1]$) zuweist.

□

15.2 **Beispiel 15.2** Die linguistische Variable $\mathcal{L}$ in diesem einfachen Beispiel sei mit x = *Körpertemperatur* bezeichnet. Die linguistischen Terme sind $T(x) = \{b_1, b_2, b_3\} := \{$erniedrigt, normal, erhöht$\}$. Der Grundraum U ist durch $U := [35, 42]$ gegeben. Exemplarisch wird A(erniedrigt) definiert. A(normal) und A(erhöht) kann man analog definieren. Es sei A(erniedrigt) $:= \{(u, \mu_{erniedrigt}(u)) | u \in U\}$ mit $\mu_{erniedrigt}(u) := 0$, $u \in [36.5, 42]$, $\mu_{erniedrigt}(u) := \frac{u-36}{0.5}$, $u \in]36, 36.5[$, $\mu_{erniedrigt}(u) := 1$, $u \in [35, 36]$. Das syntaktische Regelwerk G ist hier die Grammatik $G = (S, N, T, P)$, wobei S ein Startsymbol ist, $N := \{S\}$ die Menge der Nicht-Terminalsymbole, $T :=$ {erniedrigt,normal, erhöht} die Menge der Terminalsymbole und $P := \{S \mapsto$ erhöht, $S \mapsto$ normal, $S \mapsto$ erniedrigt$\}$ die Menge der Produktionsregeln.

□

Modifikatoren (engl.: modifier) verändern die Bedeutung der Bezeichner der linguistischen Variable. Durch die Anwendung des Modifikators „stark“ erhält man anstelle von „erkältet“ den zusammengesetzten Bezeichner „stark erkältet“. Zur Modellierung der entsprechenden Zugehörigkeitsfunktion μ von „stark erkältet“ kann man z.B. μ potenzieren zu μ^2.

Vage Variablen kann man also umsetzen in linguistische Variablen. Man spricht von einer **Fuzzifizierung**. Das ist der erste Schritt zur Erstellung eines Fuzzy-Systems.

Kapitel 16

Fuzzy-Systeme

16

16

16 Fuzzy-Systeme

Nachdem wir umfangreiche Kenntnisse über NN gesammelt haben und bereits wissen, was eine Fuzzy-Menge ist, erklären wir nun die Komponenten eines *Fuzzy-Systems*. Mit diesem Kapitel nähern wir uns auch der adaptiven Regelgenerierung durch hybride Neuro-Fuzzy-Systeme, die im nächsten Kapitel eingeführt werden.

Lehrziele:
- Den Grundaufbau eines Fuzzy-Systems angeben können,
- wissen, was man unter dem Mamdami- und Sugeno-Regelmodell versteht,
- die Min-Max-Inferenz skizzieren können,
- den Prozess der Defuzzifizierung beschreiben können,
- wissen, dass ein Fuzzy-System nicht unbedingt ein adaptives System sein muss,
- einige Anwendungsfelder für Fuzzy-Systeme nennen können,
- wissen, dass die Fuzzy-Regelung ein wichtiges Gebiet der Fuzzy-Theorie ist.

16.1 Schema eines Fuzzy-Systems

In der Abb. 16.1 sehen Sie das Schema eines Fuzzy-Systems, das z.B. zur Klassifikation eingesetzt werden kann. Wird das Verhalten modellierter Systeme kontrolliert, also geregelt, ist die Schleife in der Abb. 16.1 durch die Rückkopplung über das reale System geschlossen. Ein solches Beispiel ist durch die Temperaturregelung einer Heizung gegeben.

Die Modellierung der linguistischen Variablen stellt bereits den Schritt der sog. *Fuzzifizierung* dar. Ergänzend zur Abb. 16.1 soll erwähnt werden, dass ein scharfer Messwert auf ein sog. **Fuzzy-singleton** abgebildet wird und anschließend unscharf gemacht werden kann. Ein Fuzzy-singleton s ist durch eine vertikale Gerade von 0 nach 1 an der Stelle s auf der x-Achse gegeben. Beispiele für existierende Fuzzy-Systeme in der medizinischen Anwendung sind CADIAG-2 [Adl90] und MEDFRAME/CADIAG-IV [BLK+96] zur Diagnosenerstellung in der Inneren Medizin in den Gebieten Rheumatologie und Krankheiten des Pankreas und der Gallenblase.

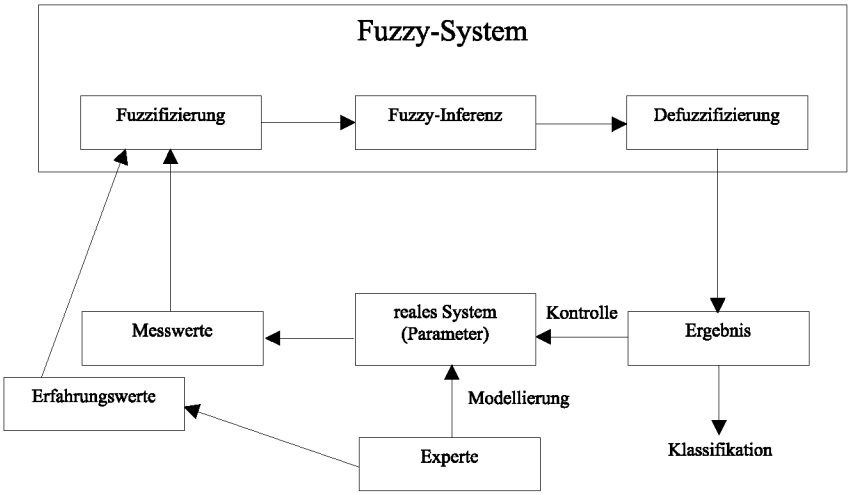

Abbildung 16.1. Schema eines Fuzzy-Systems

16.2

16.2 Regelmodelle

Um eine *Fuzzy-Inferenz* durchführen zu können, fehlt uns noch ein wichtiger Bestandteil, nämlich eine Fuzzy-Regelmenge, die die Wissensbasis repräsentiert.

Im **Mamdami-Regelmodell** haben die Regeln die Form

WENN x_1 IST A_1 UND ... UND x_n IST A_n DANN y IST B,

d.h. alle Eingabevariablen x_i haben Attributausprägungen A_i und erzeugen dadurch eine Ausgabe y mit einer Eigenschaft B. Ein einfaches Beispiel ist „WENN Hund IST groß UND Hund IST schlau DANN Hund bellt wenig".

Im **Sugeno-Regelmodell** (auch *TSK-Modell* genannt nach Takagi, Sugeno, Kang) haben Regeln die Form

WENN x_1 IST A_1 UND ... UND x_n IST A_n DANN $y = f(x_1, \ldots, x_n)$.

Hierzu ist ein Beispiel „WENN Hund IST groß UND Hund IST schlau DANN Bellverhalten $= 1$".

Beim Sugeno-Regelmodell können also Funktionswerte, die von der Eingabe abhängig sind, als Ausgabe modelliert werden.

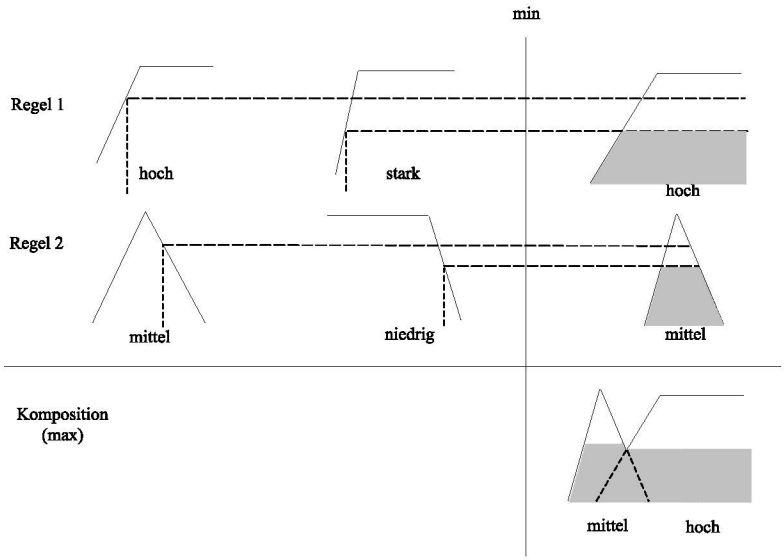

Abbildung 16.2. Schema der Min-Max-Inferenz in einem Fuzzy-System

16.3 Fuzzy-Inferenz

16.3

Um aus einer Regelmenge Schlüsse zu ziehen, also um von Eingaben auf Ausgaben anhand der Regelmenge folgern zu können, benötigen wir einen Fuzzy-Inferenzmechanismus. Wir beschreiben die zu diesem Zweck häufig verwendete **Min-Max-Inferenz**, die man sich anhand eines Beispiels verdeutlichen und gut merken kann. Betrachten wir dazu die zwei Regeln, vgl. Abb. 16.2:

WENN x_1 IST hoch UND x_2 IST stark DANN y IST hoch,

WENN x_1 IST mittel UND x_2 IST schwach DANN y IST mittel.

Aktivierungselement: Formulieren Sie die Min-Max-Inferenz für den Allgemeinfall (mit min- bzw. max-Operatoren).

Wir müssen nun die (graue) Fläche F unter der resultierenden Zugehörigkeitsfunktion in einen scharfen Wert y^* zurückwandeln. Dieser Prozess wird

Defuzzifizierung genannt. Die üblichen Methoden sind:

a) **Schwerpunktmethode.**

$$y^* = \frac{\int y \mu_R(y) dy}{\int \mu_R(y) dy} \tag{81}$$

Im diskreten Fall kann das Summenzeichen anstelle des Integrals verwendet werden. Die Methode ist sinnvoll, da sie exakt den Schwerpunkt der Fläche berechnet.

b) **Gewichteter Mittelwert.**

$$y^* = \frac{\sum_i \bar{y}_i \mu_i(\bar{y}_i)}{\sum_i \mu_i(\bar{y}_i)} \tag{82}$$

mit $\bar{y}_i := \max \mu_i(y)$. Diese Methode ist schneller, wenngleich etwas heuristischer. Sie ist insbesondere üblich bei symmetrischen Zugehörigkeitsfunktionen.

16.4

16.4 Diskussion

Die oben genannte Fuzzy-Systematik ist zwar ein Verfahren des Soft Computing (wenn nicht sogar *das* Verfahren des Soft Computing), aber es ist kein adaptives Verfahren, da die Wissensbasis nicht gelernt wird, sondern vom Experten modelliert werden muss.

Die Hauptanwendung der Mamdami-Inferenz ist der Mamdami-Kontrollregler. Wenn man klassifizieren möchte, kann man als Conclusio die Klasseninformation verwenden. Falls die Klasseninformation nicht unscharf ist, sondern ein Fuzzy-singleton, kann man die Höhe der Werte in der Conclusio separat für jede Klasse betrachten. Die Klasse mit dem höchsten Wert wird dann der Eingabe zugeordnet.

Ein nicht adaptives Fuzzy-System ist das System von Wang, Mendel [WM92], das fest vorgegebene, unveränderliche Dreiecksfunktionen verwendet. Eine adaptive Verbesserung ist das System von Higgins, Goodman [HG93], in dem dynamisch Zugehörigkeitsfunktionen eingefügt werden und zwar an der Stelle des größten Fehlers.

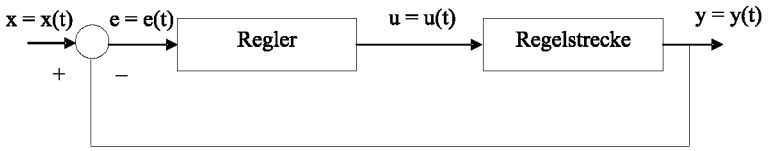

Abbildung 16.3. Schema eines Reglers

Eine größere Flexibilität bezüglich der Adaptivität eines Fuzzy-Systems erreichen wir im nächsten Kapitel durch Kombination von Fuzzy-Systemen mit Neuronalen Netzen. Bevor wir zum Ende dieses Kapitels kommen, nennen wir einige Anwendungsgebiete, in denen Fuzzy-Systeme eingesetzt werden:

- Bio- und Chemieinformatik,
- Medizinische Entscheidungssysteme, Life Science,
- Finanzmanagement, Risikomanagement, Betrugserkennung,
- Telekommunikationsdienste,
- Bildverarbeitung (unscharfe Bilddatensegmentierung),
- Verkehrsleitsysteme, Bahn-Steuerungssysteme,
- Robotik,
- Waschmaschinen (Regelung des Grades der Trübung der Lauge),
- Temperaturrregler, Klimaanlagensteuerung, Industrieprozesse.

Da *Regler* ein wichtiges Anwendungsgebiet der Fuzzy-Systeme sind, möchten wir kurz diesen Themenkreis beleuchten. In der Abb. 16.3 ist das Schema eines einschleifigen **Regelkreises** abgebildet, der aus dem eigentlichen **Regler** (Stellgrößenberechnung) und der **Regelstrecke** (ausführende Prozesseinheit) besteht. Es existieren komplizierte Regelmodelle, die theoretisch schwer zu handhaben sind. Das notwendige Fachwissen zur Erstellung eines *nichtlinearen* Fuzzy-Reglers ist geringer. Nicht unterschätzt werden darf beim Entwurf eines (Fuzzy-)Reglers das Stabilitätsverhalten. Z.B. darf der Regler nicht bei Eingabe von „warm“ nicht immer „erhöhen“ ausgeben, da dann (evtl. zum Schaden der Prozessanlage) die Temperatur immer höher wird. Oder es darf auch nicht zum Oszillieren des Reglers kommen, d.h. ständiges Hin- und Her zwischen „erhöhen“ und „erniedrigen“ (führt zu Verschleißerscheinungen, erhöhter Rohstoffverbrauch). Um die Stabilität des Reglers herzustellen, müssen die Parameter oft von Hand eingestellt werden. Der Entwurf eines Reglers erfordert deshalb eine gute Kenntnis der Literatur (Regelungstechnik) und eine lange Erfahrung, da der Entwickler die Verantwortung für die störungsfreie, stabile Funktion des Reglers übernehmen muss. Diese Aspekte der Fuzzy-Regelung finden Sie beispielsweise in [DHR93] ausgeführt.

Zum Verständnis des nächsten Kapitels sollten Sie sich unbedingt einprägen, was ein Fuzzy-System ist und wie es arbeitet. Dabei spielen insbesondere die Fuzzifizierung aus dem vorhergehenden Kapitel, die Inferenz und die Defuzzifizierung eine große Rolle. Es gibt eine sehr große Anzahl von Anwendungen in den verschiedensten Wissenschaftsdisziplinen, von denen wir im Kapitel 18 einige aus dem Bereich der Bioinformatik vorstellen werden.

Kapitel 17

17 Neuro-Fuzzy-Systeme

17

17 Neuro-Fuzzy-Systeme

Kommen wir nun in diesem Buch über Soft Computing zur Wissensextraktion durch sog. **Neuro-Fuzzy-Systeme**. Darunter wollen wir ein System verstehen, dass auf der Basis von Fuzzy-Regeln unscharfe Ein- und/oder Ausgaben wie ein Fuzzy-System verarbeiten kann und dass diese Wissensbasis anhand der Daten durch einen NN-Mechanismus adaptiert. Da in solchen Systemen NN und Fuzzy-Theorie eng verbunden sind, sprechen wir von einem **hybriden System**. Denkbar wären auch losere Kopplungen, bei denen z.B. das NN nur Parameter des Fuzzy-Systems optimiert.

Lehrziele:
- Das Funktionsprinzip von Fuzzy-Neuronen verstehen,
- das Neuro-Fuzzy-System ANFIS kennenlernen,
- die Idee des FUZZY-ARTMAP verstehen,
- das Prinzip des Neuro-Fuzzy-Systems NEFCLASS verstehen,
- die Idee des geometrischen Lernverfahrens des Fuzzy-RecBF-DDA-Neuro-Fuzzy-Systems wiedergeben können,
- einige Anwendungsgebiete von Neuro-Fuzzy-Systemen nennen können.

17.1 Einführung

17.1

Eine erste Möglichkeit FT mit NN zu verbinden besteht in der Einführung von **Fuzzy-Neuronen**, die miteinander verknüpft Fuzzy-Funktionen approximieren können.

Vereinbarung 17.1 (Fuzzy-Neuronen) 17.1

a) Wir modellieren ein **ODER-Fuzzy-Neuron** durch

$$y = \nabla_{j=1}^{n}(x_j \Delta w_j) \ . \tag{83}$$

b) Ein **UND-Fuzzy-Neuron** definieren wir durch

$$y = \Delta_{j=1}^{n}(x_j \nabla w_j) \ . \tag{84}$$

□

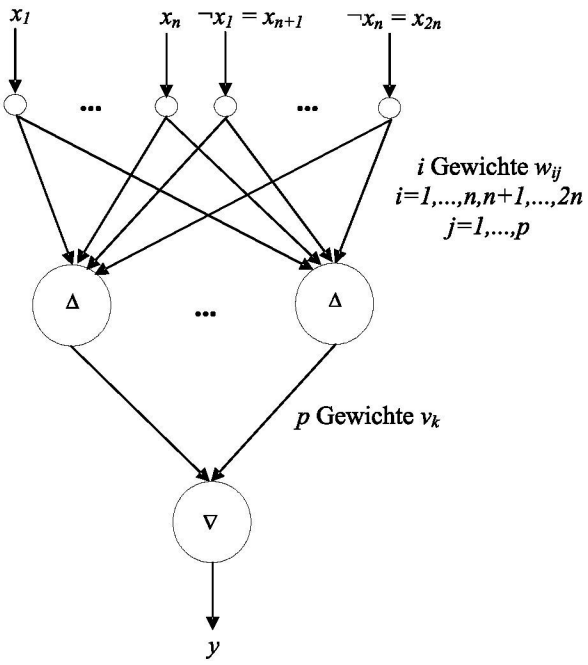

Abbildung 17.1. Der Logik-Prozessor

Aktivierungselement: Skizzieren Sie das ODER- und das UND-Fuzzy-Neuron.

Mit dem **Logik-Prozessor** von Pedrycz können Fuzzy-Funktionen approximiert werden, vgl. die Abb. 17.1.

Aktivierungselement: Gegeben sei ein Logik-Prozessor mit $n = 3$ und $p = 3$ sowie den Gewichten $w_{11} = w_{12} = w_{13} = 0.2, w_{21} = w_{22} = w_{23} = 0.3, w_{31} = w_{32} = w_{33} = 0.4$ und $v_1 = 0.6, v_2 = 0.5, v_3 = 0.4$. Berechnen Sie für $(x_1, x_2, x_3) = (0.2, 0.5, 0.7)$ die Ausgabe y.

Eine andere einfache Möglichkeit NN mit Fuzzy-Theorie zu koppeln besteht in der direkten Verwendung von (normierten) RBF-Neuronen als Zugehörigkeitsfunktionen. In [JS93] wird gezeigt, dass ein RBF-Netz dann die Form eines Mamdani-Reglers haben kann. Allerdings ist das RBF-Lernverfahren

nicht speziell auf Regelgenerierung zugeschnitten, so dass die Performanz spezieller Neuro-Fuzzy-Systeme höher ist.

Da es nicht *das* Neuro-Fuzzy-System gibt, sondern sehr viele Ansätze, die unterschiedliche Vor- und Nachteile haben, fällt die Auswahl für dieses Buch oder für eine Anwendung nicht leicht. Einige Verfahren sind bekannter, einige strukturell einfacher, einige performanter. Ein umfassender Vergleich der Systeme existiert nicht, so dass wir uns darauf beschränken, die bekannten Neuro-Fuzzy-Systeme vorzustellen. Der Autor geht ausdrücklich nicht davon aus, dass es keine anderen, anwendbare Systeme gibt. Es folgt also eine kleine, aber feine Auswahl an Neuro-Fuzzy-Systemen:

1. ANFIS von Jang [Jan93],
2. FUZZY-ARTMAP von Carpenter, Grossberg et al. [CGR91],
3. NEFCLASS von Nauck und Kruse, zusammengefasst in [NKK96],
4. FUZZY-RecBF-DDA von Huber und Berthold [HB95], [BH95].

Die Anwendungsbereiche sind generell die gleichen wie bei Fuzzy-Systemen, wobei Neuro-Fuzzy-Systeme besonders geeignet sind, wenn kein oder kaum Expertenwissen vorliegt, so dass die Parameter anhand der Daten möglichst ohne Vorwissen adaptiert werden müssen.

Wir beschränken uns im Rahmen des Buches auf eine fundamentale Beschreibung der Systeme, der Netztopologie und der Lernalgorithmen sowie auf die Nennung ihrer Vor- und Nachteile. Für diejenigen, die solch ein System implementieren und damit arbeiten möchten oder auch Verbesserungen vornehmen möchten, wird das ergänzende Studium der Originalarbeiten empfohlen.

Aktivierungselement: Studieren Sie bei Interesse die Originalarbeiten (so viele oder wenige wie Sie Zeit und Interesse haben).

Es folgt eine Beschreibung der vier o.g. Neuro-Fuzzy-Systeme, wobei wir die beiden letztgenannten etwas ausführlicher beschreiben werden.

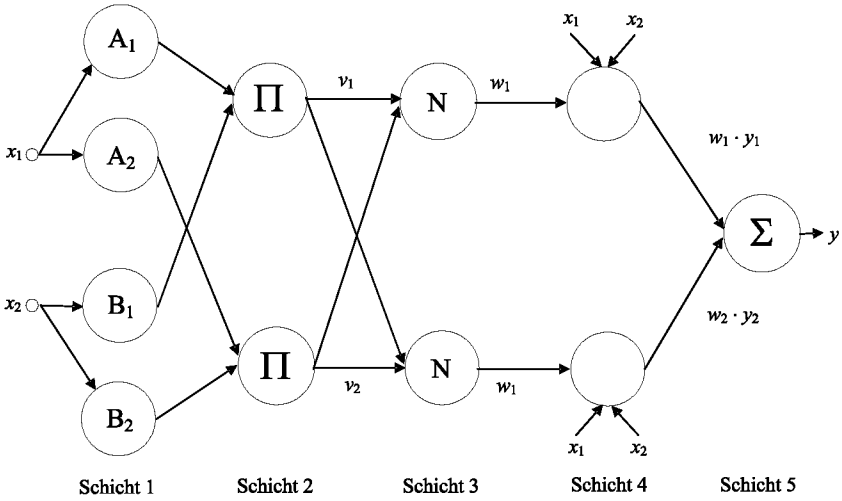

Abbildung 17.2. Netzwerktopologie von ANFIS

17.2

17.2 Das Neuro-Fuzzy-System nach Jang (ANFIS)

Wir listen einige Eigenschaften des ANFIS (engl. Abk. für: Adaptive Neuro-Fuzzy Inference System) auf.

- ANFIS ist ein Sugeno-Fuzzy-System, das adaptiert wird.
- Es verwendet differenzierbare Fehlerfunktionen, ähnlich zu Backprop.
- Es ist in der Fuzzy-Toolbox von Matlab enthalten.

Das Netz besitzt eine feste Netzwerkstruktur, vgl. Abb. 17.2. Wir gehen von dem folgenden Sugeno-Modell aus, das eine Linearkombination der Eingabe ausgibt:

WENN x_1 IST A_1 UND x_2 IST B_1 DANN $y_1 = p_1 x_1 + q_1 x_2 + r_1$,

WENN x_1 IST A_2 UND y_2 IST B_2 DANN $y_2 = p_2 x_1 + q_2 x_2 + r_2$.

Der Inferenzmechanismus lautet in diesem Modell

$$y = \frac{v_1 y_1 + v_2 y_2}{v_1 + v_2} = w_1 + w_2 \ . \tag{85}$$

Es folgt eine Beschreibung der Schichten.

Schicht 1: $O_{1,i} = \mu_{A_i}(x_1)$, und $= O_{1,i+2} = \mu_{B_i}(x_1)$, $i = 1, 2$.

Schicht 2: $O_{2,i} = v_i = \mu_{A_i}(x_1)\Delta\mu_{B_i}(x_2)$, $i = 1, 2$.

Schicht 3: $O_{3,i} = w_i = \frac{v_i}{v_1+v_2}$, $i = 1, 2$ (Gewichtung).

Schicht 4: $O_{4,i} = w_i y_i = w_i(p_i x_1 + q_i x_2 + r_i)$, p_i, q_i, r_i Parameter.

Schicht 5: $O_5 = \sum_i w_i y_i = \frac{\sum_i v_i y_i}{\sum_i v_i}$ (Summation durch ein Ausgabeneuron).

Das Lernen findet ähnlich zu Backpropagation statt und hat daher ähnliche Eigenschaften wie Backpropagation. Ein Anwendungsbeispiel ist ein Steuerungssystem als Landehilfe für Flugzeuge [Ric02].

17.3 Das Neuro-Fuzzy-System nach Carpenter et al. (FUZZY-ARTMAP)

17.3

Im Abschnitt 9.6 haben wir die Adaptive Resonanztheorie und die Idee eines ARTMAP vorgestellt. Eine Erweiterung, das sog. *FUZZY-ARTMAP*, besitzt die folgenden Eigenschaften:

- Plastizität (neue Muster können neue Kategorien erzeugen).
- Stabilität (alte Muster gehen nicht verloren, wenn sie zu neuen Mustern nicht ähnlich sind).
- Das Netz hat eine Art Kurzzeitgedächtnis (Vergleichsschicht) und eine Art Langzeitgedächtnis (Erkennungsschicht).
- Operationen sind gegenüber dem scharfen ART-1 (vgl. z.B. [Zel94, Kap. 22]) in Fuzzy-Notationen übertragen.
- Das Lernverfahren ist nicht speziell auf Regelgenerierung ausgelegt und ist trotz Fuzzy-Verarbeitung nicht der performanteste Regellerner.
- ART-1: Binäre Eingaben, ART-2: Reelle Eingaben, FUZZY-ART: Reelle Eingaben.

Bei einem FUZZY-ART-Netz ist die Vergleichsschicht und die Erkennungsschicht vollständig miteinander verbunden. Es gibt einen Verstärkungsfaktor und einen Reset-Faktor. Es gibt eine reellwertige bottom-up-Matrix zur Ähnlichkeitsberechnung in der Erkennungsphase und eine binäre top-down-Matrix zum Vergleich der Korrektheit der erfolgten Clusterzuordnung.

In der Lernphase eines ART-Netzes werden Prototypen, die Gewichtsvektoren w_j entsprechen, mit dem aktuellen Sample I (input) verglichen. Ist

$$\frac{|I \wedge w_j|}{|I|} \geq p \tag{86}$$

erfüllt, so wird ein Adaptionskriterium angestoßen (p Ähnlichkeitsparameter $\in [0,1]$). In der Fuzzy-Notation wird dieses übertragen zu

$$\frac{|I \Delta w_j|}{|I|} \geq p \ . \tag{87}$$

Die eine FUZZY-ART-Komponente eines FUZZY-ARTMAP lernt eine unüberwachte Clusterung wie ein ART-Netz aus den Daten, die andere Komponente erhält die Klassenzugehörigkeiten. Eine verbindende MAP-Komponente versucht die Unterschiede zwischen den beiden FUZZY-ART-Komponenten anzupassen.

Ein Anwendungsbeispiel ist das medizinische Diagnostik-System [LTL+99].

17.4

17.4 Das Neuro-Fuzzy-System nach Nauck und Kruse (NEFCLASS)

Wir beschreiben das Lernverfahren und die Netztopologie des Systems *NEFCLASS* [NKK96]. NEFCLASS besitzt die folgenden Eigenschaften:

- Es besitzt keine Regelgewichtung und erzeugt somit eindeutig interpretierbare Regeln, vgl. [Nau00].
- Eine ungefähre Vorgabe der Zugehörigkeitsfunktionen ist notwendig und somit auch deren Anzahl, die aber auch adaptiert werden kann.
- Gelernt wird durch ein Backpropagation-ähnliches Verfahren, aber ohne die Differenzierbarkeit zu fordern.
- Jeweils alle Attribute gehen in die Regeln ein; eine zusätzliche Regelaggregation ist als Nachbearbeitung empfehlenswert.
- I.d.R. werden Dreiecke als Zugehörigkeitsfunktionen verwendet; es können auch Trapeze verwendet werden.

Die FF-Netztopologie von NEFCLASS ist in der Abb. 17.3 abgebildet. NEFCLASS basiert auf einem dreischichtigen Fuzzy-Perzeptron. Es hat gemeinsame Gewichte für gleiche Terme in der Prämisse, um die Interpretierbarkeit des Systems zu gewährleisten.

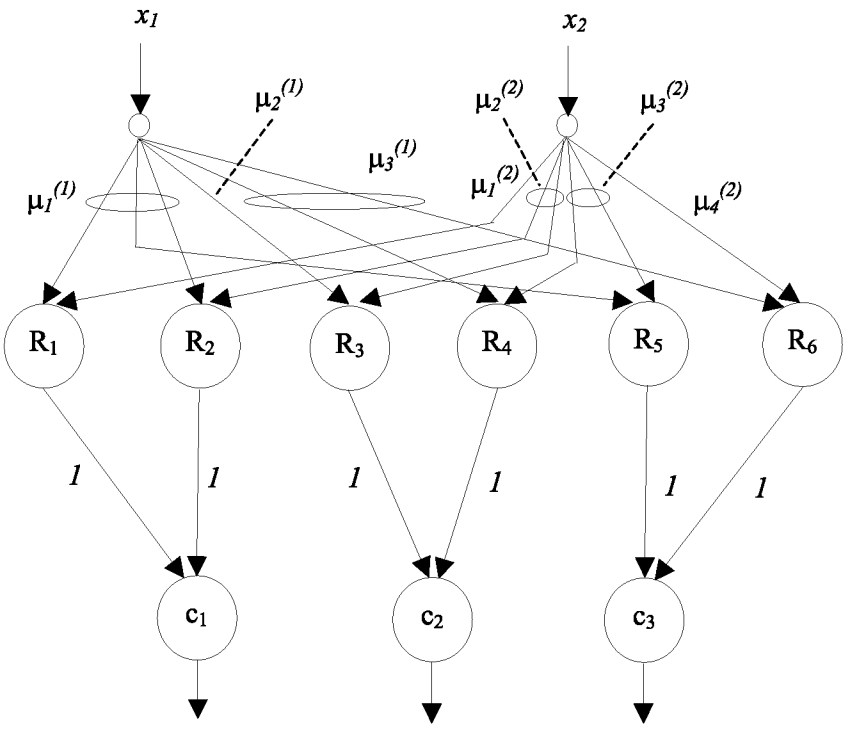

Abbildung 17.3. Netztopologie eines NEFCLASS-Systems mit zwei Eingaben, sechs Regeln und drei Klassen

Die Gewichte der zweiten Schicht, $W(R,c)$ genannt, sind entweder gleich 0 oder gleich 1. Die Aktivierung der Ausgabe ist $c_j = \frac{\sum_{R \text{ Regel}} W(R,c) o_R}{\sum_{R \text{ Regel}} W(R,c)}$ mit o_R als Ausgabe der zweiten Schicht. Die jeweilige Aktivierungsfunktion ist eine Dreiecksfunktion.

17.2 Algorithmus 17.2 (NEFCLASS-Lernen)

I) Regellernalgorithmus:
Gegeben seien n Eingaben $x_1, \ldots, c_n$, k Regelneuronen $R_1, \ldots, R_k$ und m Ausgabeneuronen $c_1, \ldots, c_m$.

1) Wähle nächstes Sample aus und propagiere es.

2) Für jede Eingabeeinheit x_i finde die Zugehörigkeitsfunktion $\mu_{j_i}^{(i)}$ mit

$$\mu_{j_i}^{(i)} = \max_{j \in \{1,\ldots,q_i\}} \mu_j^{(i)}(a_{x_i}) \quad . \tag{88}$$

3) Gibt es noch nicht maximal viele Regeln und gibt es kein Regelneuron mit $W(x_1, R) = \mu_{j_1}^{(1)}, \ldots, \mu_{j_n}^{(n)}$, dann erzeuge ein solches Neuron und verbinde es mit dem Ausgabeneuron c_l, falls $t_l = 1$ ist.

4) Gehe zu Schritt 1), falls noch nicht alle Samples bearbeitet wurden, sonst gehe zu Schritt 5).

5) Bearbeite Regelmenge nach (wird hier nicht näher ausgeführt), z.B. durch Auswahl oder Aggregation.

II) Fuzzy-Mengen-Lernalgorithmus:
1) Wähle nächsten Sample aus und propagiere ihn; bestimme Ausgabevektor.

2) Für jedes Ausgabeneuron c bestimme den Deltawert $\delta_c^{(p)} = \text{sgn}(t_c^{(p)} - o_c^{(p)}) \cdot E_c^{(p)}$, wobei $E_c^{(p)}$ der Fuzzy-Fehler $E_c^{(p)} := 1 - e^{-\beta(t_c^{(p)} - o_c^{(p)})^2}$ ist (t teacher, $\beta > 0$).

3) Für jede Regel R mit $o_R > 0$ bestimme:

3a) den Deltawert $\delta_R^{(p)} = o_R^{(p)}(1 - o_R^{(p)}) \sum_c W(R,c) \delta_c^{(p)}$.

3b) das Eingabeneuron y, so dass $W(y,R)(o_y^{(p)}) = \min_x \{W(x,R)(o_x^{(p)})\}$.

3c) für die Fuzzy-Menge $W(y,R)$ die Deltawerte ihrer Parameter a,b,c unter Verwendung der Lernraten $\sigma_a,\sigma_b,\sigma_c>0$:

$$\delta_b^{(p)}=\sigma_a\delta_R^{(p)}(c-a)\ \mathrm{sgn}(o_y^{(p)}-b)\ , \tag{89}$$

$$\delta_a^{(p)}=-\sigma_b\delta_R^{(p)}(c-a)+\delta_b^{(p)}\ , \tag{90}$$

$$\delta_c^{(p)}=-\sigma_c\delta_R^{(p)}(c-a)+\delta_b^{(p)}\ . \tag{91}$$

Addiere diese Deltawerte zu den Parametern von $W(y,R)$, falls dies nicht gegen gegebene Randbedingungen verstößt.

4) Breche ab, falls Abbruchbedingung (kleiner Fehler) erfüllt ist, ansonsten gehe zu Schritt 1).

□

Zwei Anwendungsbeispiele sind die Modellierung eines inversen Pendels aus [NKK96] und die medizinische Diagnose [NK99].

17.5 Das Neuro-Fuzzy-System nach Huber und Berthold (Fuzzy-RecBF-DDA)

17.5

Wir beschreiben die Architektur und das Lernen des Neuro-Fuzzy-Systems [HB95], [BH95]. Die Idee des Lernens ist es, die Geometrie von trapezoiden Zugehörigkeitsfunktionen zu verändern. Vorab geben wir die Eigenschaften des Systems, hier *Fuzzy-RecBF-DDA* genannt, wieder:

- Heuristische Adaption mit grundsätzlicher Verringerung der Verarbeitungskomplexität zur Vermeidung der kombinatorische Explosion): Angenommen man möchte Regeln für einen d-dimensionalen Datensatz, $d\in\mathbb{N}$, kombinatorisch erzeugen, so entstehen bei einer Regel bereits $2d$ Richtungen, wenn man die $\pm$-Richtungen betrachtet. Jede der $2d$ Möglichkeiten kann in Kombination auftreten, d.h. man erhält unter Verwendung des Binomischen Lehrsatzes

$$\sum_{i=1}^{2d}\binom{2d}{i}=2^{2d}-1=4^d-1 \tag{92}$$

Möglichkeiten, eine Ausdehnung einer d-dimensionalen Regel vorzunehmen. Das führt bereits bei kleinerem d zu einer sehr hohen Zahl an möglichen Kombinationen, siehe die Tab. 17.1. Der Fuzzy-RecBF-DDA-

Tabelle 17.1. Anzahl u der generell möglichen Kombinationen bei einer Regelgenerierung mit Rechtecken. Anzahl v der von Fuzzy-RecBF-DDA überprüften Möglichkeiten

d	$u = 4^d - 1$	$v = 2d$
1	3	2
2	15	4
3	63	6
4	255	8
5	1023	10
10	1048575	20
20	$\approx 1.1 \cdot 10^{12}$	40
50	$\approx 1.3 \cdot 10^{30}$	100

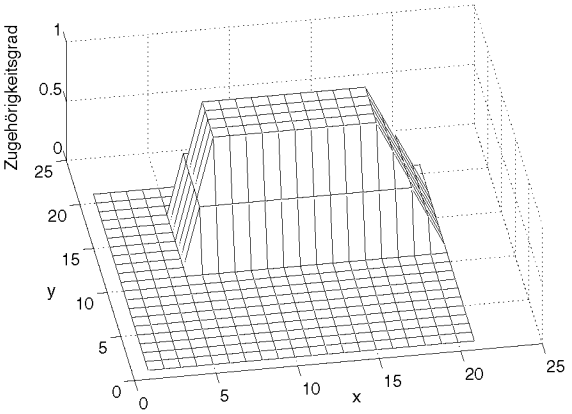

Abbildung 17.4. Eine zweidimensionale Zugehörigkeitsfunktion

Algorithmus überprüft nur $2d$ Ausdehnungen (jede Dimension, jede Richtung). Trotz der Einsparung an überprüften Möglichkeiten generiert der Algorithmus als adaptives System gute, suboptimale Ergebnisse.

- Erkennung redundanter Variablen innerhalb einer Regel, d.h. hat ein Teil einer Regel die Form „... WENN var_j IN $(-\infty, +\infty)$ UND ... DANN Klasse c“, dann ist die Variable j redundant für diese Regel.
- Keine explizite Vorgabe von Fuzzy-Variablen wie z.B. „niedrig“, „mittel“, „hoch“ notwendig, wenngleich möglich. Die Trapezoide adaptieren sich an die Daten, und deren Anzahl ist nicht von vornherein festgelegt.
- Abhängigkeit von der Reihenfolge der Präsentation der Trainingsdaten.
- Sofortige Erzeugung einer neuen Regel bei Ausreißern.

Die Netztopologie ist die gleiche wie beim RBF-DDA-Netz. Als Aktivierungsfunktionen werden statt RBF Hyper-Trapezoide verwendet (Abb. 17.4).

Die Verwendung von Hyper-Trapezoiden (basierend auf Trapez-Funktionen) ist der Verwendung von Hyper-Pyramiden (basierend auf Dreiecksfunktionen) vorzuziehen [Pae02a]. Der Grund hierfür ist, dass ein punktförmiger Core-Bereich („Pyramidenspitze") als Bereich zum Fuzzy-Niveau 1.0 nicht sicher von den Testdaten getroffen wird, so dass die tatsächliche Sicherheit unter dem Niveau 1.0 liegt.

Algorithmus 17.3 (Fuzzy-RecBF-DDA) **17.3**

Parameter: Im Gegensatz zum RecBF-DDA-Netz werden hier pro Regel *zwei* Rechtecke adaptiert, ein inneres (*core*) und ein äußeres (*support*), so dass sich nach dem Lernprozess in kanonischer Weise trapezoide Zugehörigkeitsfunktionen und damit Fuzzy-Regeln erzeugen lassen. Für die Ausdehnungen der Rechtecke werden die Parameter $\lambda^k_{d,i,-}, \lambda^k_{d,i,+}$ (inneres Rechteck) und $\Lambda^k_{d,i,-}, \Lambda^k_{d,i,+}$ (äußeres Rechteck), Dimension $d = 1, \ldots, n$, Anzahl der Regeln pro Klasse $i = 1, \ldots, m_k$ eingeführt.

Die folgenden Schritte beschreiben den Lernvorgang einer Epoche.
1. Gewichte zurücksetzen:
for l = 1 **to** c
  for i = 1 **to** m_l
    $w^l_i := 0$;
    $\lambda_{d,i,\pm} := 0$;
  end
end

2. Muster x durchgehen (x sei aus der Klasse k):
for $r = 1$ **to** $\sharp\mathcal{DS}$ (durchlaufe Datensamples)
  if (p^k_i covers x) **then**

3. Muster durch p^k_i überdeckt (cover):
   $w^k_{i_*} := w^k_{i_*} + 1$; erhöhe nur einen einzigen Zähler i_* aus der Menge derjenigen i, für die p^k_i x überdeckt
   adjustiere die $\lambda^k_{d,i,\pm}$, so dass sie x überdecken;

4. Neues Neuron einfügen (commit):
  else
    $z^k_{m_k+1} := x$;

$\lambda_{d,m_k+1,\pm} := 0$; Core-Bereich ist nur der Zentrumspunkt
$\Lambda^k_{d,m_k+1,\pm} := \infty$; Support-Bereich ist unendlich

5. **for** $l \neq k$, $1 \leq j \leq m_l$
Schrumpfe $p^k_{m_k+1}$ anhand z^l_j, d.h.
shrink($p^k_{m_k+1}$,z^l_j);
end
$m_k := m_k + 1$;
$w^k_{m_k} := 1.0$;
end

6. Konfliktneuronen verkleinern (shrink):
for $l \neq k$, $1 \leq j \leq m_l$
Schrumpfe p^l_j anhand x, d.h.
shrink(p^l_j,x);
end
end

□

Es folgt die Beschreibung des im Algorithmus verwendeten „shrink"-Moduls.

17.4 Algorithmus 17.4 (Shrink-Mechanismus)

Shrink-Algorithmus shrink(p,x) („minimaler Volumenverlust"):
$M := |z_{\tilde{d}} - x_{\tilde{d}}|$ mit $\tilde{d} := \text{index}\Big(\min_{1 \leq d \leq n} \Big\{ \Big| \frac{\Lambda_{d,\pm} - |z_d - x_d|}{\Lambda_{d,\pm}} \Big| \Big\} \Big)$;
(Berechne $\tilde{d}$ mittels +-Richtung in der Dimension d für Λ, falls $z_d - x_d \geq 0$, sonst verwende die −-Richtung, für alle *endlichen* Radien Λ.)

Falls M existiert, setze:
$\Lambda_{d,\min,\pm} := M$;
$\Lambda_{d,\text{bestfinite},\pm} := M$, falls $M \geq \sigma_{d,\min}$;

$\Lambda_{d,\max,\pm} := \max\{|z_d - x_d| \mid 1 \leq d \leq n\}$;
(Berechne $\Lambda_{d,\max,\pm}$ mittels +-Richtung in der Dimension d für Λ, falls $z_d - x_d \geq 0$, sonst verwende die −-Richtung, für alle *unendlichen* Radien Λ.)

Berechne ein neues $\Lambda_{d,\pm}$, d.h. ein *shrink* in einer Richtung für einen Radius:
if $\Lambda_{d,\text{bestfinite},\pm}$ existiert **then**
$\Lambda_{d,\pm} := \Lambda_{d,\text{bestfinite},\pm}$;
if $\Lambda_{d,\max,\pm} \geq \Lambda_{d,\min,\pm}$ **then**

$\Lambda_{d,\pm} := \Lambda_{d,\max,\pm}$;
else
$\Lambda_{d,\pm} := \Lambda_{d,\min,\pm}$;
end
end

□

Im Folgenden sei d eine von n Dimensionen des Datenraums. Die Ausführungsphase ergibt sich im Falle asymmetrischer Radien zu:

1. Schicht:

$$S_i^k(x) := \min_{1 \le d \le n} A(x_d, z_{d,i}^k, \Lambda_{d,i,+}^k) \text{ mit} \tag{93}$$

$$A(x_d, z_{d,i}^k, \Lambda_{d,i,+}^k) := \begin{cases} 1 & , 0 \le x_d - z_{d,i}^k \le \Lambda_{d,i,+}^k \\ 0 & , \text{sonst} \end{cases} \tag{94}$$

und

$$T_i^k(x) := \min_{1 \le d \le n} A(x_d, z_{d,i}^k, \Lambda_{d,i,-}^k) \text{ mit} \tag{95}$$

$$A(x_d, z_{d,i}^k, \Lambda_{d,i,-}^k) := \begin{cases} 1 & , 0 \le z_{d,i}^k - x_d \le \Lambda_{d,i,-}^k \\ 0 & , \text{sonst} \end{cases} \tag{96}$$

werden zusammengefasst zu:

$$R_i^k(x) := \max\{S_i^k(x), T_i^k(x)\} \ . \tag{97}$$

2. Schicht (gewichtete Summe analog zum RBF-DDA-Netz):

$$y_k(x) := y_k(R_i^k(x)) := \sum_{i=1}^{m_k} w_i^k R_i^k(x) \ .$$

In der Ausgabe kann ein Winner-Takes-All-Mechanismus zur Entscheidung verwendet werden, welche Klasse gewinnt. Sind die Aktivierungen gleich, so findet keine Zuordnung zu einer Klasse statt.

In [PB02b], [Pae02f] wird gezeigt, dass verschiedene Trainingsdurchläufe mit unterschiedlichen Trainingsdaten wie erwartet unterschiedliche Regelmengen erzeugen. Vorgeschlagen wird die Berechnung der ähnlichsten Regelmenge als diejenige Regelmenge, die im Mittel am ähnlichsten zu allen anderen erzeugten Regelmengen ist. In der Anwendungsphase kann man diese ähnlichste

Regelmenge verwenden. Ansonsten hätte man keine Möglichkeit, aus n trainierten Netzwerken das geeignetste für die Anwendung auszuwählen.

Einige Anwendungsbeispiele und Varianten sind:

- Modellierung von CPU-Ausfällen [Hub98],
- Merkmalsauswahl auf medizinischen Daten [SB99],
- Regelgenerierung aus Daten von Patienten mit septischen Schock zur Erstellung eines Alarmsystems [Pae01], [Pae03c], [PA02], [Pae02b],
- Integration von metrischen und symbolischen Daten [Ber03], Verwendung verschiedener Normen [GB04],
- Aufbau hierarchischer Regeln aus Daten [GB03],
- Variante des Shrink-Algorithmus, um unendliche Ausdehnungen anders zu berücksichtigen [Pae02b].
- Regelgenerierung aus einer Datenbank mit chemischen Molekülen [PS04], [PFF+04].

Die Kombination von Neuronalen Netzen mit der Fuzzy-Logik führt zu den hybriden Neuro-Fuzzy-Systemen. Eine einfache Idee ist die Verwendung von Fuzzy-Neuronen im neuronalen Netz. Heute existieren sehr viele Modelle, die sich teilweise nur in Details unterscheiden. Wir haben vier Modelle vorgestellt, die sich deutlich voneinander unterscheiden und die zu mehreren Anwendungen geführt haben, um Ihnen einen sinnvollen ersten Einblick in die Neuro-Fuzzy-Systemlandschaft zu gewähren. Es wäre schön, wenn Sie die Grundideen der vier Modelle ANFIS, FUZZY-ARTMAP, NEFCLASS und Fuzzy-RecBF-DDA kennen und die Systeme voneinander abgrenzen können.

Kapitel 18

Fuzzy-Technologie in der Bioinformatik

18

18

18 Fuzzy-Technologie in der Bioinformatik

In diesem Kapitel präsentieren wir Anwendungen, die die Fuzzy-Terminologie verwenden oder in denen Neuro-Fuzzy-Systeme zum Einsatz kommen.

Lehrziele:
- Die Anwendung des ANFIS-Systems in der Sekundärstrukturvorhersage kennenlernen,
- die Anwendung von FUZZY-ART und FUZZY-ARTMAP zur Genexpressionsanalyse zur Kenntnis nehmen,
- Neuro-Fuzzy-Systeme zur Motiv-Extraktion kennenlernen,
- die Modellierung des Phänotyps mit Fuzzy-Zahlen nachvollziehen können,
- die Fuzzifizierung des Polynukleotidraumes und die Abstandsberechnung in diesem Raum kennen,
- weitere Anwendungen nennen können,
- verstehen, wie das Fuzzy-RecBF-DDA-System zum virtuellen Screening eingesetzt werden kann.

18.1 Sekundärstrukturvorhersage

Ausgangspunkt für die Arbeit [HIH03] ist die SCOP-Datenbank (Structural Classifications of Proteins), in der Proteine nach ihren Faltungsklassen abgespeichert sind. Verwendet wird hier nur die Unterscheidung zwischen „Alpha“ (Helices) und „Beta“ (Blätter). Von diesen Proteinen existieren Fouriertransformierte Infrarot (FTIR)-Spektren, mit Daten pro 1cm^{-1}.

Spektren von 7 Alpha-Proteinen und 10 Beta-Proteinen werden zum Lernen verwendet. Die Lernleistung wurde mit leave-one-out überprüft, d.h. je eins der Proteine wurde getestet, während mit den übrigen trainiert wurde.

Verwendet wurde das Neuro-Fuzzy-System ANFIS (vgl. Kap. 17) aus der Fuzzy Logic Toolbox des Programmpakets Matlab. Die Faltungsklasse wird mit der maximalen Amid-I-Band Position im Infrarot-Spektrum bestimmt, d.h. ANFIS wurde mit einer Eingabe und einer Ausgabe verwendet. Die Regeln haben die Form „IF input is A THEN output is c“ mit A als Fuzzy-Menge und c als Konstante. Die Klassifikationsentscheidung wird auf Basis einer Schwellwertfunktion getroffen.

Anschließend werden die Werte des Absorptionsvermögens in Struktur-spezifischen Netzwerken trainiert. Dafür wurden RPROP-Netzwerke verwendet, genauer: 101 Eingaben aus 1600 - 1700cm^{-1} und eine versteckte Schicht mit zwei Knoten.

Mit dieser neuronalen Gesamtarchitektur erreichte man eine Verbesserung gegenüber einer einfachen neuronalen Netz-Architektur. Die Trainingsmenge erscheint allerdings etwas gering, um insgesamt die Lernleistung zu beurteilen. Mit ANFIS wurden die folgenden Regeln gefunden:

1) WENN Amid-I-Band-Maximum ist NAHE BEI 1634cm^{-1} DANN Beta,

2) WENN Amid-I-Band-Maximum ist NAHE BEI 1646cm^{-1} DANN Beta,

3) WENN Amid-I-Band-Maximum ist NAHE BEI 1657cm^{-1} DANN Alpha.

18.2 Clustering von Daten zur Genexpressionsanalyse mit FUZZY-ART

Es werden 45 Gene von *Saccaromyces cerevisae* betrachtet [THHK02]. Die Expressionsraten wurden zu den Zeitpunkten $t = 0, 0.5, 2, 5, 7, 9, 11.5$h gemessen und zwischen 0 und 1 normiert. Es wurden 4, 5, 6 und 7 Cluster ausprobiert. Evaluiert wurden die Cluster nach den Kriterien „früh“, „mittel“, „mittel bis spät“ und „spät“, d.h. ein Cluster wurde nach den mehrheitlich vorkommenden Genen bezeichnet. Fünf Cluster erschienen dabei als beste Wahl. Verglichen wurde die Clusterung mit einer hierarchischen Clusterung, einer k-means-Clusterung und einer SOM-Clusterung. Die Ergebnisse waren in ihrer Höhe vergleichbar gut.

18.3 Genexpressionsanalyse mit FUZZY-ARTMAP

Zur Erkennung von B-Zell-Abnormitäten wird eine Variante des FUZZY-ARTMAP mit Genexpressionsdaten trainiert [Azu00]. Genauer gesagt betrachtet man die sog. diffuse large B-cell lymphoma (DLBCL). Man möchte Krebs-Subtypen durch Klassifikation oder Clusterung entdecken. Expressionslevel von 5 Genen (CD10, BCL-6, TTG-2, IRF-4, BCL-2) werden verwendet, um 63 Samples von B-Zellen zu gewinnen (45 DLBCL, 18 normal). Auch hier wurde leave-one-out zur Bestimmung der Klassifikationsleistung verwendet, die je nach Parameterwahl zwischen 65% und 76% liegt. Lässt man anstelle zweier Klassen zu, dass mehrere Clusterneuronen eingefügt wer-

den, so kann man die Klasse DLBCL weiter unterteilen in „germinal centre (GC) B-like DLBCL“ und „activated B-like DLBCL“.

18.4 Motiv-Extraktion mit Neuro-Fuzzy-Optimierung

Es wird eine statistische Methode angewendet, um Motive (Subsequenzen) aus Familien von Proteinsequenzen zu extrahieren [CH02]. Anschließend wird ein Neuro-Fuzzy-System zur Optimierung der Klassifikationsaufgabe verwendet. Der Sinn dahinter ist die Klassifikation einer unbekannten Sequenz in ihre zugehörige Proteinfamilie.

Wir betrachten starre und flexible Motive, z.B. A-x(5)-B oder x(2,4). Im ersten Fall findet man zwischen zwei Nukleotiden A und B genau 5 Leerstellen, im zweiten Fall findet man nur 2 bis 4 Leerstellen. Beispielsweise deckt das Proteinmotiv ABC-x(1,3)-DEF die Sequenzen ABCHHDEF und ABCAAADEF ab. Möglichst kurze und häufig auftretende Motive sollen gefunden werden.

Die notwendigen Schritte für diese Aufgabe sind Sequenzvorverarbeitung, Motiverzeugung, Motivauswahl und Motivoptimierung.

Sequenzvorverarbeitung: ABC-x(1,3)-DEF kann man als Folge von *Ereignissen* und *Intervallen* schreiben:

A-x(0)-B-x(0)-C-x(1,3)-D-x(0)-E-x(0)-F.

Schreibt man jedes Element als eine Kombination zweier Ereignisse mit zugehörigem Intervall, so erhält man: A-x(0)-B, B-x(0)-C, usw.

Motiverzeugung: Längere Motive erzeugt man durch Aneinanderhängen, z.B. wird aus C-x(2)-C und C-x(3)-F das längere Motiv C-x(6)-F.

Motivauswahl: Aus der Menge der nun vorhandenen Motive kann man diejenigen Auswählen, die besonders gut gewünschte Motive und möglichst wenig unerwünschte Motive abdecken.

Motivoptimierung: Die Eingaben eines zur Optimierung verwendeten Netzwerks sind die Intervalle. Die versteckte Schicht kann z.B. aus RBF-Knoten (=Zugehörigkeitsfunktionen) bestehen. Ein Motiv kann einer Sequenz auf

verschiedene Weise eingepasst werden. Zum Trainieren kann man die Intervalle der besten Einpassung im Sinne eines Abstandes zwischen Motiv und eingepasstem Motiv wählen.

Anwendung: Angewendet wurde das System auf Daten aus der PROSITE-Datenbank:

a) C2H2-Zinkfinger-Protein (418 Sequenzen): Das Motiv C-x(12)-H wurde in 99% aller Sequenzen gefunden. Allerdings ist das Motiv nicht spezifisch genug, d.h. es ist auch in anderen Proteinen enthalten. Deshalb verlängert man das Motiv, so dass es spezifischer wird.

b) EGF (engl. Abk. für: epidermal growth factor) Protein: Durch Optimierung erhält man aus C-x(1)-C-x(5)-G-x(2)-C das Motiv C-x(1,4)-C-x(4,6)-G-x(2,4)-C.

Nach der Beschreibung von Anwendungen von Neuro-Fuzzy-Verfahren werden in den folgenden Abschnitten zwei Möglichkeiten präsentiert, wie man anstelle einer präzisen Modellierung eine Fuzzy-Modellierung verwenden kann.

18.5

18.5 Vererbung von Eigenschaften

Um Vererbungsprozesse zu modellieren, kann man Phänotypen mit reellen Zahlenwerten angeben. Da man oft nicht genau diese Werte angibt, sondern oft auch Bezeichnungen wie „niedrig“, „mittel“ und „hoch“, ist die Idee statt präziser Zahlenwerte Fuzzy-Zahlen wie in Kap. 13 zu verwenden [CRLB03].

Der Phänotyp besteht zu einem Teil aus der Summe der Effekte der Allele der zu analysierenden Orte in Chromosomen. Angenommen man hat eine Eigenschaft, die durch zwei Orte bestimmt wird. Jeder der Orte habe zwei Allele (A,a) und (B,b). Sei e(XY) der Effekt bei Vorhandensein der Allele X und Y. Wir beschreiben durch (a,b,c) eine Dreiecksfunktion für eine Fuzzy-Zahl. a ist der linke untere Punkt des Dreiecks, b der obere mittlere Punkt

und c der rechte untere Punkt. Wir nehmen an, dass gilt:

$$\begin{aligned} e(AA)&=(0,0,0),\\ e(Aa)&=(0,0,0),\\ e(aa)&=(0,1,1.5),\\ e(BB)&=(-1,0,1),\\ e(Bb)&=(-1,0,1),\\ e(bb)&=(1,2,3)\ . \end{aligned}$$

Man erhält z.B. (-1,0,1) für AABB durch Addition der Fuzzy-Zahlen.

Aktivierungselement: Welche Werte erhält man für die anderen Kombinationen AAAA bis bbbb?

Die Eigenschaften könnte man z.B. durch die vier Dreiecksfunktionen low = (-3,-3,-1), mittel = (-3,-1,1), hoch = (-1,-1,3) und sehr hoch = (1,3,3) modellieren. Danach kann man ein Fuzzy-System zur Weiterverarbeitung der Daten anwenden.

18.6 Fuzzifizierung von Polynukleotiden

18.6

In [TN03] werden Polynukleotide fuzzifiziert. Jede DNA-Sequenz besteht aus Elementen des Alphabets {T, C, A, G}. T steht für Thymin, C für Cytosin, A für Adenin und G für Guanin. Ein Triplet kodiert eine Aminosäure. Für die RNA stehen die Elemente U (Uracil), C, A, G zur Verfügung.

Kosko hat 1992 vorgeschlagen, die Menge aller Fuzzy-Mengen als Hyperwürfel zu modellieren. Ist $X = \{x_1, \ldots, x_n\}$ eine Menge. Dann hatten wir eine Fuzzy-Menge als Abbildung $\mu : X \to [0,1]$ definiert. $\mu(x)$ ist der Zugehörigkeitsgrad. Jede mögliche Zugehörigkeitsfunktion definiert einen Punkt

$$(\mu(x_1), \ldots, \mu(x_n)) \tag{98}$$

im Hyperwürfel durch eine bijektive Zuordnung. Binäre („crispe“) Abbildungen haben ihre Punkte auf den Ecken des Hyperwürfels.

Wir können (im Falle der RNA) U als 1000 kodieren, C als 0100, A als 0010 und G als 0001. Der genetische Code ist (3 mal 4 gleich) 12-dimensional.

Jedes der 64 Codons liegt auf einer der 2 hoch 12 gleich 4096 Ecken des 12-dimensionalen Hyperwürfels, z.B. ist der Code für Histidin (CAU) gleich (0,1,0,0;0,0,1,0;1,0,0,0).

Legen wir Zugehörigkeitsgrade ungleich Null zugrunde, so ist ein Beispiel (0.3,0.4,0.1,0.2;0,0,1,0;1,0,0,0). Das heißt, wir haben _AU sicher. Der erste Buchstabe ist unsicher, z.B. erhalten wir UCG (Serin) nur mit einer Zugehörigkeit von 0.3. Da bei der Sequenzanalyse Ungenauigkeiten auftreten können, ist eine solche Modellierung sinnvoll.

Kommen wir nun zu der Bestimmung von Abständen der Polynukleotide: Es seien zwei Punkte p und q im 12-dimensionalen Raum gegeben. Wir können einen Abstand durch

$$d(p,q) := \frac{\sum_{i=1}^{12} |p_i - q_i|}{\sum_{i=1} 12 \max\{p_i, q_i\}} \tag{99}$$

definieren.

Aktivierungselement: Berechnen Sie die Abstände von Histidin, Serin und Glutamin zueinander.

Verwenden wir unseren Fuzzy-Vektor, so ergibt sich ein Abstand von _CG und CCG zu d(_CG,CCG) $= 1/3$.

Betrachten wir die Genomsequenz von *Mycobacterium tuberculosis* H37Rv. Sie hat 4411529 Basenpaare und enthält ca. 4000 Gene. Man kann die Häufiigkeit von T, C, A, G an den drei Basisseiten des Codons zählen, z.B. hat man 216051 mal T an der ersten Basisstelle. Das ist als Wahrscheinlichkeit 0.1632. Insgesamt erhält man einen Punkt im 12-dimensionalen Hyperwürfel

$$(0.163, 0.309, 0.172, 0.356; 0.204, 0.315, 0.176, 0.306; 0.165, 0.346, 0.159, 0.330). \tag{100}$$

Die 4639221 Basenpaare von Escherichia coli K-12 ergeben den Vektor

$$(0.161, 0.242, 0.260, 0.337; 0.312, 0.229, 0.285, 0.175; 0.262, 0.257, 0.183, 0.298). \tag{101}$$

Wir erhalten als Abstand nach (99) d(*M.tubercolosis*,*E.coli*)= 0.248.

18.7 Weitere Anwendungen

18.7

Im Folgenden nennen wir noch Anwendungen, in denen Moleküle eine größere Rolle spielen, d.h. Anwendungen aus der Molekularen Biologie und der Chemieinformatik.

- [EYA+01]: Anwendung eines FUZZY-ARTMAP zur Vorhersage des Siedepunkts oder der kritischen Temperaturen bzw. den Drücken mit einem Fehler unter 1% für Deskriptordaten.
- [EKB02a], [EKB02b]: Vergleich molekularer Oberflächen anhand ihrer fuzzifizierten 3D-Eigenschaften, z.B. wird die Elektrostatik als high negative, negative, neutral, positive und high positive modelliert.
- [LV02]: Fuzzifizierung der Moleküloberflächen, da die Position einzelner Atome aufgrund der thermischen Energie unscharf ist.
- [NBG+02]: Anhand von Deskriptoren wird die Toxizität von 568 Molekülen bestimmt. Nach einer Dimensionsreduktion, u.a. mit PCA, werden 17 Dimensionen betrachtet.

18.8 Virtuelles Screening mit Neuro-Fuzzy-Systemen

18.8

Zum Ende wird eine Anwendung vorgestellt, in der es um virtuelles Screening [BS00] unter Anwendung des Fuzzy-RecBF-DDA-Systems aus dem Kap. 17 geht. In Kürze lässt sich die Anwendung in der Molekularen Bioinformatik wie folgt beschreiben. Ein aktuelles Forschungsgebiet ist das Wirkstoffdesign unter Verwendung von Computern und intelligenten Algorithmen, der sog. *in silico*-Ansatz. Solch einen Ansatz verwenden wir zur Unterscheidung aktiver und inaktiver Liganden für verschiedene Drug Targets. Diese Art von retrospektivem Screening wird erreicht durch die Kodierung der Daten mit Deskriptoren und einer anschließenden Neuro-Fuzzy-Klassifikation. Zusätzlich zur Klassifikation wird auch eine Merkmalswahl und eine Regelgenerierung durchgeführt. Mit Hilfe dieses Ansatzes können wir Regeln angeben, die Regionen mit deutlich mehr aktiven Liganden beschreiben, verglichen mit der A-priori-Wahrscheinlichkeit. Mit diesem Ansatz haben wir höhere Enrichment-Faktoren (Abk.: ef) erzielen können als mit Ansätzen, die auf Ähnlichkeitsberechnungen [FFR+03] beruhen.

Zum Trainieren und Testen des Systems verwenden wir eine Datenbank, die insgesamt ca. 5000 aktive und inaktive Moleküle enthält [SS03]. Ein Drug Target ist z.B. die Matrix-Metalloproteinase (Abk.: MMP). Für alle Moleküle wird eine 150-dimensionale Deskriptorvektor-Darstellung berechnet und

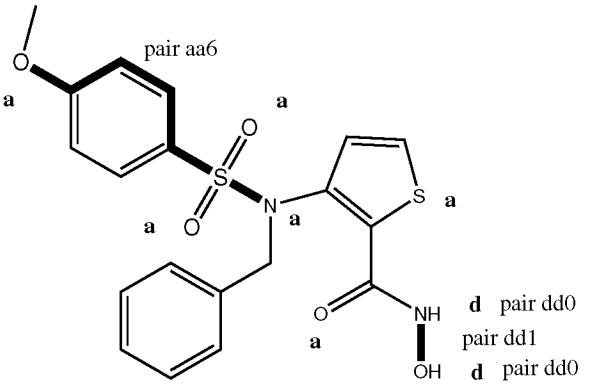

Abbildung 18.1. Beispiel eines MMP-Inhibitors mit den Bedingungen der Regel bezüglich den Typen a und d

zwar durch Zählen von ganzzahligen Bindungsabständen zwischen Atomtypen (Wasserstoffbrückendonoren, -akzeptoren, positiv und negativ geladene Gruppen und lipophile Atome/Gruppen) [SNGS99]. Da acht Zähler Null waren, blieben 142 Dimensionen für die Analyse übrig.

Mit einer Trainingsmenge, bestehend aus der Hälfte der Vektoren, wurde ein Fuzzy-RecBF-DDA-System trainiert. Zur Merkmalswahl wurde die Idee verwendet, dass in diesem System Variablen unwichtig in einer Regel sind, wenn das Bedingungsintervall in der Regel von $-\infty$ bis $+\infty$ geht. Wir haben 21 der 142 Dimensionen für ein erneutes Training verwendet und erhielten im Mittel einen ROC-Flächeninhalt von 0.83.

Als Ergebnis erhalten wir z.B. die folgende Regel R mit freq(R)= 1.1%, conf(R)= 80.8%, und ef = 49.

R: **if** dd0 > 0.00 **and** var dd1 > 0.00 **and** aa6 > 0.00 **then** Klasse „aktiv".

Wir sehen, dass die Regel die Struktur des Molekülgraphen in chemisch sinnvoller Weise widerspiegelt, so dass der Ansatz für ein virtuelles Screening geeignet erscheint, nicht zuletzt deshalb, weil die Intervallregel-Bedingungen schnell überprüfbar ist. Aus 142 Dimensionen konnte eine 3-dimensionale, einfache Beschreibung generiert werden, die auch als Grundlage für eine Expertendiskussion diente. Die ef können durch Optimierung mit evolutionären Strategien, die wir im Kapitel 27 des Buches einführen, noch verbessert werden [Pae04a].

Nachdem wir in den vorherigen Kapiteln eine Reihe von Fuzzy-Techniken kennengelernt haben wie Fuzzy-Logik, Fuzzy- und Neuro-Fuzzy-Systeme, haben wir in diesem Kapitel einen Blick auf die Anwendungsmöglichkeiten geworfen. Präsentiert wurden die Sekundärstrukturvorhersage von Proteinen aus der SCOP-Datenbank, die Clusterung von Genexpressionsdaten mittels Genen von *Saccaromyces cerevisae*, die Analyse von Genexpressionsdaten mit FUZZY-ARTMAP zur Erkennung von B-Zell-Abnormitäten und die Motiv-Extraktion von Proteinen aus der PROSITE-Datenbank mittels Neuro-Fuzzy-Systemen. Fuzzy-Modellierungen wurden verwendet, um unscharfe Phänotypen zu formulieren und um den Polynukleotidraum zu fuzzifizieren. Auch wurde eine Anwendung zum virtuellen Screening vorgestellt. Insgesamt sind alle diese Anwendungen unterschiedlich und zeigen das Potenzial der Fuzzy-Technologie in der Bioinformatik. Zum Abschluss dieses Kapitels soll noch erwähnt werden, dass man Fuzzy-Regler im Bereich der Regelung von Bioreaktoren einsetzen kann.

Kapitel 19

Maschinelles Lernen im Rahmen der KI und der Logik

19

19

19 Maschinelles Lernen im Rahmen der KI und der Logik

Im folgenden Abschnitt werden ausgewählte Literaturhinweise zum Maschinellen Lernen (Abk.: ML) und zur Künstlichen Intelligenz (Abk.: KI) gegeben. Da diese Arbeitsgebiete schon lange existieren, aber immer noch aktuell sind, gibt es bereits viel Literatur und es kommt ständig viel hinzu. Zuvor geben wir die Lehrziele diese Kapitels an.

Lehrziele:
- Einen Literaturüberblick über Literaturquellen im Bereich des Maschinellen Lernens gewinnen,
- wissen, dass die gebräuchlichsten Logiken die Aussagen- und die Prädikatenlogik sind,
- die Inferenzmechanismen Deduktion, Induktion und Abduktion unterscheiden können,
- die wichtigsten Ideen der Programmiersprache PROLOG wie das Resolutionsprinzip kennen,
- den Begriff des Expertensystems einordnen können,
- als Beispiel für einen Ansatz des Induktiven Logischen Programmierens die Vorhersage der Faltungsklassen kennenlernen.

19.1 Literaturübersicht

Wir beschränken uns auf Literatur, die auch Data Mining, Intelligente Datenanalyse und Mustererkennung beinhaltet, verzichten aber auf Hinweise zur Robotik. Wir geben wieder Konferenzen, Zeitschriften und Lehrbücher an.

Konferenzen:

- International Conference on Machine Learning (ICML),
- European Conference on Machine Learning (ECML),
- International Conference on Machine Learning and Applications (ICMLA),
- International Joint Conference on Artificial Intelligence (IJCAI),
- AAAI National Conference on Artificial Intelligence,
- European Conference on Artificial Intelligence (ECAI),
- International Conference on Tools with Artificial Intelligence (ICTAI),
- German Conference on Artificial Intelligence (KI),

- International Conference on Multiagent Systems (ICMAS),
- International Congress on Intelligent Systems and Applications (ISA),
- International Conference on Data Mining (ICDM),
- ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD),
- SIAM International Conference on Data Mining,
- International Conference on Machine Learning and Data Mining (MLDM),
- International Conference on Intelligent Data Engineering and Automated Learning (IDEAL),
- Indian International Conference on Artificial Intelligence (IICAI),
- Springer-Reihe: Lecture Notes in AI.

Zeitschriften:

- AI Communications,
- Annals of Mathematics and Artificial Intelligence,
- Applied Artificial Intelligence,
- Applied Intelligence,
- Applied Soft Computing,
- Artificial Intelligence,
- Artificial Intelligence in Engineering,
- Computational Intelligence,
- Data Mining and Knowledge Discovery,
- Engineering Applications of Artificial Intelligence,
- Expert Systems: The International Journal of Knowledge Engineering and Neural Networks,
- Expert Systems with Applications,
- IEEE Intelligent Systems & their Applications,
- IEEE Transactions on Knowledge and Data Engineering,
- IEEE Transactions on Pattern Analysis and Machine Intelligence,
- IEEE Transactions on Systems, Man, and Cybernetics, Part A/B/C,
- Intelligent Data Analysis,
- International Journal of Approximate Reasoning,
- International Journal of Computational Intelligence and Applications,
- International Journal of Intelligent Systems,
- Journal of Artificial Intelligence Research,
- Journal of Experimental & Theoretical Artificial Intelligence,
- Knowledge-Based Systems,
- Pattern Analysis & Applications,
- Data & Knowledge Engineering,
- Decision Support Systems,

- Knowledge and Information Systems,
- Cybernetics and Systems,
- Journal of Automated Reasoning,
- Journal of Classification,
- Pattern Recognition,
- Pattern Recognition Letters,
- Artificial Intelligence in Medicine,
- Journal of Machine Learning Research,
- Machine Learning,
- Soft Computing,
- KI – Zeitschrift Künstliche Intelligenz.

Lehrbücher (Auswahl):

- Mitchell [Mit97] (ML),
- Keller [Kel00] (ML, KI),
- Han/Kamber [HK00] (ML, Data Mining),
- Beierle/Kern-Isberner [BKI00] (ML, KI),
- Helbig [Hel91] (KI).

19.2 Logik

19.2

Unter ML im Rahmen der KI und der Logik in diesem Abschnitt wollen wir vor allem Verfahren verstehen, die auf Logiksystemen basieren.

Das Arbeitsgebiet der KI versucht, Systeme zu entwickeln, die eine Art von intelligentem Verhalten haben, d.h. die sich an unbekannte oder ähnliche Situationen anpassen können, z.B. das Lenken eines Fahrzeugs, Schach spielen oder das Treppensteigen o.ä. Die Methoden des Soft Computing, die in den anderen Kapiteln präsentiert werden, haben zum Fortschritt der KI in der Datenanalyse wesentlich beigetragen. Im Vergleich zu Logik-basierten Systemen (*symbolische* KI) werden die Methoden, die insbesondere die Verwendung NN beinhalten, auch als *subsymbolische* KI bezeichnet, da das Wissen oft nicht explizit gespeichert ist. Wir haben aber bereits mit den Neuro-Fuzzy-Systemen hybride Verfahren kennengelernt.

Als Programmiersprachen werden z.B. LISP (funktionale Programmierung) und PROLOG (Logik-orientierte Programmierung) eingesetzt. Auf die Sprache PROLOG gehen wir später noch kurz ein, da die Bedeutung von PRO-

LOG für die Regelgenerierung höher ist als die von LISP. Wir gehen im Folgenden generell nur auf Sachverhalte ein, die im Zusammenhang mit der Regelgenerierung stehen, nicht aber z.B. auf Sprach- oder Bildverarbeitung oder auf allgemeinere Problemlösealgorithmen.

Die bekannteste Logik ist die **Aussagenlogik**, in der Aussagen wie „Die Lampe ist an." als wahr oder falsch bewertet werden. Das **Prädikatenkalkül** ergänzt die Aussagenlogik um Existenz- und Allquantoren, z.B. bedeutet „$\exists x : \ A(x)$", da mindestens ein Parameter x existiert, so dass die Aussage $A(x)$ gilt.

Folgende *Schließmechanismen* (oder *Inferenzmechanismen*) kann man klassischerweise unterscheiden:

19.1 **Definition 19.1 (Inferenzmechanismen)**
a) **Deduktion.** „WENN aus A folgt B und A gilt DANN B gilt".

Beispiel: WENN Alle Professoren sind reich, Dr. X ist Professor DANN Dr. X ist reich.

Diese Art der Schlussweise ist die Schlussweise für Beweistechniken in der Mathematik schlechthin. Man schließt vom Allgemeinen auf das Spezielle.

b) **Induktion.** „WENN A gilt und B gilt DANN aus A folgt B".

Beispiel: WENN Dr. X ist Professor, Dr. X ist reich, DANN Alle Professoren sind reich.

Man schließt aus (sich wiederholenden) Beobachtungen auf die Hauptprämisse, d.h. auf eine Regel. Man schließt vom Speziellen auf das Allgemeine (Umkehrung der Deduktion), was natürlich im Ergebnis falsch sein kann.

In der Disziplin des **Induktiven logischen Programmierens** (Abk.: ILP) versucht man aus allgemeinen Sachverhalten (unter Benutzung von PROLOG) auf Regeln zu schließen. Problematisch ist bei dieser Schlussweise der Umgang mit unscharfen Beobachtungen.

c) **Abduktion.** „WENN aus A folgt B und B gilt DANN A gilt".

Beispiel: WENN Alle Professoren sind reich, Dr. X ist reich DANN Dr. X ist Professor.

□

Die *Fuzzy-Logik* haben Sie bereits im Kapitel 15 kennengelernt. Die Fuzzy-Inferenz wird auch als *approximatives Schließen* bezeichnet.

19.3 PROLOG

19.3

Die Grundbausteine von PROLOG sind die **Horn-Clausen**, die üblicherweise in der Form „$C : -A_1, \ldots, A_k$“ notiert werden, d.h. aus der UND-Verknüpfung der Literale A_i folgt C.

Definition 19.2 (Resolutionsprinzip) 19.2

a) Eine **Resolvente** zweier Clausen C_1, C_2 mit den Literalen A in C_1 und $\bar{A}$ in C_2 ist die Clause $C = C_1 \cup C_2$.
Die Ableitung der leeren Clause durch Rückwärtsverkettung zum Zweck der Widerlegung einer Clausenmenge wird als **Resolutionsprinzip** bezeichnet.

b) Eine **Substitution** s in Clausen mit Prädikaten $P(x)$ ist das Ersetzen von x durch y, kurz: $s = x/y$.
Eine Substitution s heißt **Unifikator** von $\{A_1, \ldots, A_n\}$, wenn gilt $A_1\tau = \ldots = A_n\tau$. Der Unifikator τ heißt **allgemeinster Unifikator** (engl.: most general unifier, mgu), wenn für jeden weiteren Unifikator σ eine Substitution θ existiert mit $\sigma = \tau \circ \theta$.
Kann man bei einer Anfrage (z.B. bruder(tom,tim)?) die leere Clause als mgu erhalten, so ist die Anfrage negativ zu beantworten.

□

Aktivierungselement: Unifizieren Sie $\{Q(x, g(z, y), f(a)), Q(g(a, b), x, f(z))\}$ durch Auffinden des mgu.

19.4 Expertensysteme

Klassische **Expertensysteme** sind folgendermaßen aufgebaut: Wissensbasis (Fakten und Regeln), Schließmechanismus, Ein- und Ausgabemodul. Der Schließmechanismus ist klassisch ein symbolischer Regelgenerator, der in moderneren Architekturen durch Fuzzy-Mechanismen, statistische Berechnungen und/oder neuronale Netze ersetzt bzw. ergänzt wird. Das klassische Expertensystem gibt *bekannte* Informationen an den Anwender. Eine Einführung in klassische Expertensysteme gibt [Hel91, Kap. 7]. Dort werden als Beispiel für medizinische Expertensysteme MYCIN (Ermittlung eines Erregers einer bakteriellen Infektion) und VM (Überwachung der mechanischen Atmungsunterstützung) vorgestellt sowie die Expertensysteme MOLGEN (Planung molekulargenetischer Experimente) und DENDRAL (Ermittlung chemischer Strukturformeln durch Interpretation ihrer Massenspektrogramme).

In der Praxis kommt es vor, dass ein Regelsystem nicht statisch ist, sondern dynamischen Veränderungen unterworfen ist. Solche Regelsysteme müssen revidiert werden; sie sind insofern nicht monoton wie statische Systeme, die erweitert werden können, aber nicht revidiert werden müssen. In *nichtmonotonen* Regelsystemen muss es möglich sein, Regeln zurückzunehmen oder zu verändern bzw. zu entscheiden, welche Regel im Zweifelsfall angewendet werden soll. Dies geschieht durch ein sog. *Truth-Maintenance-System.* Eine formale Behandlung des Themas findet man in [BKI00].

Auf analoge und weitere unscharfe Schließmechanismen gehen wir im Kapitel 24 ein.

19.5 Vorhersage von Faltungsklassen

Im Arbeitsgebiet der strukturellen Bioinformatik ist ein ILP-Ansatz angesiedelt, der 45 Faltungsklassen anhand von Regeln vorhersagt [CMS03] (mit 97%-iger Korrektheit). Eine Rossmann-Faltung wird beschrieben durch die beiden „ODER“ verknüpften Regeln:

```
fold(A,'NAD(P)-binding Rossmann-fold domains') :-
number_helices(3=<(A=<4)), helix(A,B,h,b),
contains(B,g,nterm), contains(B,g,inter).

OR
```

```
fold(A,'NAD(P)-binding Rossmann-fold domains') :-
sheet(A,B,para), helix(A,C,h,g), helix(A,D,h,i),
helix_angle(C,D,para), sheet_top_6(B,3,2,1,4,5,6).
```

Ausgeschrieben bedeutet das: Eine Rossmann-Faltung wird beschrieben durch eine Anzahl von 3 bis 4 Helices, und der α-Helix B als zweites Kernelement in der Sequenz, und B enthält ein Glycin sowohl in der Mitte als auch in n-terminalen Regionen; oder durch ein paralleles Blatt B, das aus 6 Strängen mit der Topologie 321456 besteht, und durch α-Helices C und D als siebtes bzw. neuntes Kernelement in der Sequenz, und C und D berühren sich und sind parallel. Ein Beispiel für diesen Faltungstyp kann man unter http://wehih.wehi.edu.au/scop/data/scop.b.d.c.A.html (Alcohol dehydrogenase-like, C-terminal domain [51736]) einsehen.

Das Maschinelle Lernen zusammengefasst mit dem Gebiet der Künstlichen Intelligenz ist ein unüberschaubar großes Gebiet, was Sie bereits an der Auswahl der Literaturquellen erkennen können. In diesem Kapitel haben wir Ihnen einige grundlegende Sachverhalte und Ideen präsentiert. Die klassische Logik ist der Ursprung der formalen Wissensverarbeitung. In diesem Rahmen wurden die ersten Expertensysteme konstruiert. Immer noch werden die Programmiersprachen PROLOG oder auch LISP dazu verwendet, wenngleich mittlerweile andere System-Modellierungen wie Fuzzy-Modellierungen einen hohen Verbreitungsgrad haben. Ein Beispiel für Induktive Logische Programmierung mittels PROLOG ist die Vorhersage von Faltungsklassen.

Konzentrieren wollen wir uns im Rahmen des Maschinellen Lernens zur Regelerzeugung im Folgenden auf Entscheidungsbaumverfahren und auf Assoziationsregeln sowie auf einige weitere Methoden.

Kapitel 20

Entscheidungsbäume

20

20

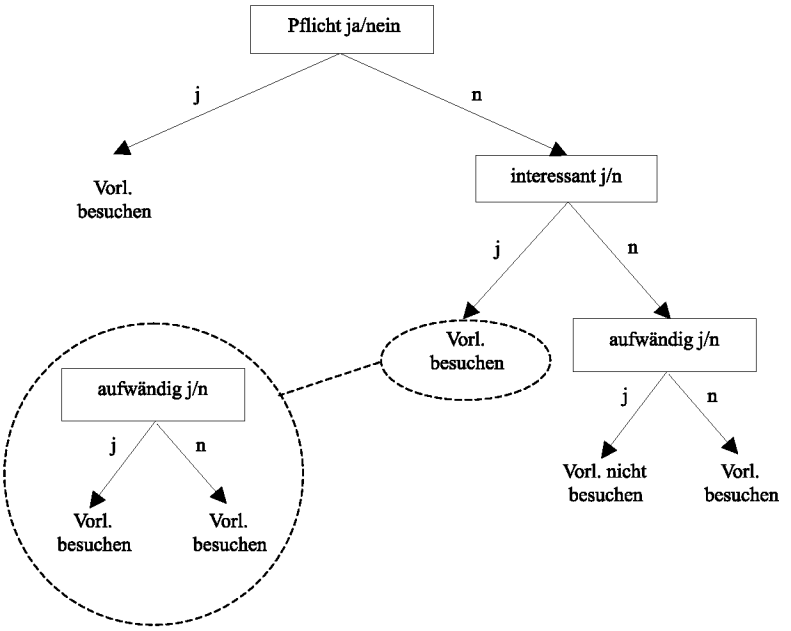

Abbildung 20.1. Ein Entscheidungsbaum

20 Entscheidungsbäume

Ein **Entscheidungsbaum** ist nichts anderes als eine hierarchische Baumstruktur, bei der in einem Knoten ein Verzweigungskriterium für ein Attribut enthalten ist. Im Prinzip ist also ein Entscheidungsbaum ein Suchbaum, dessen Pfade als Regeln interpretiert werden. Deshalb ist es sinnvoll, die „wichtigsten“ Attribute weiter oben im Baum anzugeben und die „unwichtigeren“ weiter unten im Baum. Das ist auch vor allem deshalb wichtig, da man evtl. die unwichtigeren Attribute und deren Verzweigungsknoten ganz weglassen (prunen) kann. Schauen wir uns einen solchen Baum in der Abbildung 20.1 an. Selbstverständlich kann der Baum auch in einigen Knoten mehr als zwei Kinder haben.

Der Entscheidungsbaum wurde mit den (binären) Attributen {Pflicht, interessant, aufwändig, früh} berechnet. Dabei hat sich herausgestellt, dass man bestimmte Äste wegprunen konnte, da in diesen feineren Verzweigungen (fast) immer die gleiche Entscheidung getroffen wurde. Es hat sich herausgestellt, dass es keine Rolle spielt, ob die Vorlesung früh stattfindet oder nicht, so dass das Attribut gänzlich unwichtig ist. Eine Regel lautet also beispielsweise:

WENN Vorlesung IST nicht Pflicht UND Vorlesung IST interessant DANN Vorlesung besuchen.

Wie kann man einen solchen Baum (für diskrete Variablen) berechnen? Wir beschränken uns auf eine an [CT91] und [BH99, Kap. 6] angelehnte Darstellung im Rahmen der Informationstheorie.

Lehrziele:
- Die wichtigsten Begriffe aus der Informationstheorie wie Entropie und Informationsgewinn verstehen,
- wissen, wie man einen Entscheidungsbaum berechnen und darstellen kann,
- ein erstes Beispiel für einen Entscheidungsbaum kennenlernen,
- einige Schwächen nennen können.

20.1

20.1 Informationstheorie

Wir stellen als Erstes einige Begriffe aus der Informationstheorie bereit.

20.1

Definition 20.1 (Informationstheorie) Sei X eine Zufallsvariable und $X = x_k$ eine Realisierung davon.

a) Die **Information** I des Ereignisses x_k mit Wahrscheinlichkeit P_k wird definiert durch

$$\mathrm{I}(x_k) := -\log \mathrm{P}_k \ . \tag{102}$$

b) Die **Entropie** $\mathrm{H}(X)$ der (reellen, diskreten) Zufallsvariablen X wird als Maß für die „Unordnung“ der Zufallsvariablen X definiert durch

$$\mathrm{H}(X) := \mathrm{E}(\mathrm{I}(x_k))_{x_k} = -\sum_{X=x_k} \mathrm{P}_k \log \mathrm{P}_k \ . \tag{103}$$

c) Seien X, Y zwei (reelle, diskrete) Zufallsvariablen und $X = x_k$ bzw. $Y = y_k$ Realisierungen der Zufallsvariablen X bzw. Y. Die **Transinformation** (engl.: mutual information) wird definiert durch

$$\mathrm{I}(X;Y) := \mathrm{H}(X) - \mathrm{H}(X|Y) = \sum_{X=x_k} \sum_{Y=y_k} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) \ .$$

Dabei bezeichnet $\mathrm{H}(X|Y) = -\sum_{X=x_k} \mathrm{P}_k \sum_{Y=y_j} \mathrm{P}_{k|j} \log \mathrm{P}_{k|j}$ die **bedingte Entropie** (engl.: conditional entropy) mit $P_{k|j} := P(x_k|y_j)$.

d) Der **Informationsgewinn** (engl.: information gain) ist definiert als

$$G(X;Y) := H(X) - H(X|Y) \ . \tag{104}$$

□

20.2 Entscheidungsbaum-Lernen

20.2

Jetzt ist das Berechnen eines Entscheidungsbaums nicht mehr schwierig. Man setze X als Klasseninformation und Y als Attribut. Berechne für jedes Attribut den Informationsgewinn und wähle dasjenige Attribut mit dem höchsten Informationsgewinn als Knoten im Baum aus. Führe für die entstehenden Teilbäume mit den restlichen Attributen diesen Schritt durch, solange noch Attribute vorhanden sind. Als Nachverarbeitung können Teilbäume mit zu niedrigem Informationsgewinn geprunt werden.

Das ist auch schon die Idee des ID3-Verfahrens (Abk. für: Interactive Dichotomizer 3). Ein weiteres Verfahren (C4.5) kann verrauschte Daten besser verarbeiten durch die Wahl eines anderen Splitting-Kriteriums im Baumknoten.

Beispiel 20.2 20.2

Gegeben seien die folgenden Daten der Tabelle 20.1. „As1“ und „As2“ bezeichnen den Anteil zweier Aminosäuren in der Sequenz. Wir berechnen den Informationsgewinn der beiden Variablen (die Attributausprägungen kürzen wir mit dem ersten Buchstaben ab), d.h. gesucht ist

$G(\text{Klasse} \mid \text{As1}) = H(\text{Klasse}) - H(\text{Klasse} \mid \text{As1})$.

Es ist $H(\text{Klasse}) = -P(\text{g}) \log_2 P(\text{g}) - P(\text{k}) \log_2 P(\text{k}) = -5/10 \log_2(5/10)$ $-5/10 \log_2(5/10) = -0.5 \cdot (-1) - 0.5 \cdot (-1) = 0.5 + 0.5 = 1$ [bit].

Es ist $H(\text{Klasse} \mid \text{As1}) = -5/10 \cdot 1/5 \log_2(1/5) - 5/10 \cdot 4/5 \log_2(4/5) - 5/10 \cdot$ $4/5 \log_2(4/5) - 5/10 \cdot 1/5 \log_2(1/5) = 0.28$ [bit] und $H(\text{Klasse} \mid \text{As2}) = 0.67$ [bit].

Tabelle 20.1. Sequenz-Daten

Sequenz Nr.	As1	As2	Klasse
1	hoch	niedrig	krank
2	hoch	niedrig	krank
3	niedrig	niedrig	gesund
4	hoch	hoch	krank
5	hoch	hoch	gesund
6	hoch	niedrig	krank
7	niedrig	niedrig	gesund
8	niedrig	hoch	krank
9	niedrig	niedrig	gesund
10	niedrig	niedrig	gesund

Wir erhalten $G(\text{Klasse} \mid \text{As1}) = H(\text{Klasse}) - H(\text{Klasse} \mid \text{As1}) = 1 - 0.28 = 0.72$ [bit] und $G(\text{Klasse} \mid \text{As2}) = 1 - 0.67 = 0.33$ [bit].

Deshalb wählen wir in einem Entscheidungsbaum das Attribut „As1“ für den ersten Entscheidungsknoten aus.

□

Aktivierungselement: Machen Sie sich das Beispiel klar und rechnen Sie das Beispiel für die Teilbäume weiter.

Aktivierungselement: Bewerten Sie Ihnen bekannte Vorlesungen anhand der in der Abb. 20.1 genannten Attribute und entscheiden Sie, ob Sie diese Vorlesungen besuchen würden. Berechnen Sie dazu einen Entscheidungsbaum.

20.3 **Beispiel 20.3** In [BS00, S. 28-30] wird die Anwendung eines Entscheidungsbaums für die Unterscheidung der Klassen „drug“ (aus CMC = Comprehensive Medicinal Chemistry Database) und „non-drug“ (aus ACD = Available Chemistry Directory) wiedergegeben. Die Eingaben waren 7-dimensional (sieben Deskriptoren). Zum Trainieren und Testen wurden jeweils 3500 Stoffe verwendet. 80% der Stoffe aus CMC und ca. 70% der Stoffe aus ACD wurden richtig erkannt. Zwei Pfade aus diesem Entscheidungsbaum sind:

WENN Anzahl der Donor-Gruppen < 3 UND Anzahl der Akzeptor-Gruppen ≤ 3 DANN Klasse „drug“,

WENN Anzahl der Donor-Gruppen ≥ 3 UND Anzahl der Akzeptor-Gruppen > 3 UND Anzahl der Akzeptor-Gruppen > 8 DANN Klasse „non-drug“.

Die Verzweigtheit der Regeln ist als Nachteil anzusehen.

□

Beispiel 20.4 Für die Aktivierung von T-Zellen sind bestimmte Motive verantwortlich. In [SKSK99] wird unter Verwendung eines Entscheidungsbaums untersucht, welche Motive von Peptiden der Längen acht bis zehn an ein Klasse-I-MHC-Molekül binden (MHC ist Abk. für: Major Histocompatibility Complex). Verwendet wurde die Datenbank MHCPEP. 174 stark bindende Peptide bezüglich HLA-A*0201 wurden für die Analyse ausgesucht. Die notwendige Nomenklatur findet man unter http://www.gene.ucl.ac.uk/nomenclature/ (HUGO Gene Nomenclature Committee). „HLA-A“ bedeutet „Major Histocompatibility Complex, Class I, A“. In der „Nucleotide“-Datenbank des NCBI findet man u.a. den zugehörigen Organismus, hier *Homo sapiens*. Ein Ergebnis war, dass 76.8% der negativen Peptide das Motiv xx der Länge 2, $x \in \{\text{G, R, S, W, Y}\}$ oder das Motiv yzz der Länge 3, $y \in \{\text{C, D, T, V}\}$, $z \in \{\text{A, K, M}\}$, enthalten. **20.4**

□

Auch für *stetige* Attribute müssen sinnvolle diskrete Entscheidungsintervalle gefunden werden. Während man bei den symbolischen Attributen einfache Berechnungen vornehmen konnte, werden bei stetigen Attributen auch adaptive Algorithmen eingesetzt, die heuristisch suboptimale Entscheidungsintervalle finden können. Numerische Daten können aber auch als Vorverarbeitung geclustert werden und somit diskretisiert werden. Folgende Probleme treten beim Regellernen mit Entscheidungsbäumen auf:

a) Es liegt keine echte Multidimensionalität der Datenauswertung vor, sondern eine hierarchische Auswertung. Dadurch entstehen „parallele“ Regeln in den verschiedenen Ästen, deren Gesamtverständnis (Verständnis der gesamten Regelmenge) schwierig ist.

b) Pruning erhöht zwar die Verständlichkeit der Regeln, kann aber insbesondere bei nicht-achsenparallelen Klassengrenzen zum prunen vieler vorhandener, wenig häufigen Regeln führen, so dass dann die Klassifikationsleistung gering wird.

c) Entscheidungsbaum-Verfahren sind sehr anfällig gegenüber der Reihenfolge der Präsentation der Trainingsdaten, d.h. zwei Bäume können sich sehr stark unterscheiden.

Entscheidungsbaumlernen ist aber ein schnelles Verfahren, das eine gewisse Verbreitung gefunden hat. Zur Berechnung benötigt man statistische Informationen über die Daten, die z.B. durch die Informationstheorie gegeben werden. Attribute mit einem höheren Informationsgewinn stehen beim ID3-Entscheidungsbaum höher im Baum. Als Beispiel haben wir bereits eine Klassifikation von Molekülen kennengelernt, werden aber weitere im Kapitel 25 vorstellen.

Kapitel 21

Assoziationsregeln

21

21

21 Assoziationsregeln

Zwei prinzipiell mögliche Ansätze zum Auffinden von häufig vorkommenden Mustern und deren Abhängigkeiten in einem vorhandenen Datensatz sind zum einen die Generierung von *Assoziationsregeln* und zum anderen die Bildung von *Generalisierungsregeln*. Diese Verfahren werden hauptsächlich für symbolische Variablen eingesetzt, sind aber nicht notwendig darauf beschränkt. Wir stellen im folgenden Abschnitt den grundsätzlichen Unterschied dieser Paradigmen vor. Anschließend werden wir die Assoziationsregelgenerierung präsentieren und im nächsten Kapitel dann die Generierung von Generalisierungsregeln.

Lehrziele:
- Das Prinzip der Generalisierung und Spezialisierung von Attributmengen verstehen,
- die wichtigen Gütekennzahlen Häufigkeit und Konfidenz für Itemsets und Assoziationsregeln definieren können,
- einige Anwendungen von Assoziationsregeln kennen,
- den A-priori-Algorithmus verstehen und anwenden können,
- wissen, dass der A-priori-Algorithmus in vielerlei Hinsicht verändert oder erweitert werden kann,
- wissen, dass es Alternativen zum A-priori-Verfahren gibt.

21.1 Assoziation und Generalisierung

Wir präsentieren im Folgenden eine anwendungsorientierte Darstellung des einfachen und bekannten Algorithmus *A-priori* [AS94], der in die beiden Schritte *Itemseterzeugung* und *Identifikation häufiger Itemsets* aufgeteilt ist. Die Grundlage der Verfahren bildet in dieser Arbeit eine Ansammlung von Mengen (engl.: *Itemsets*), die jeweils elementare Attributausprägungen (engl.: *Items*) als Elemente besitzen. Eine Ansammlung ist eine Menge, in der aber jedes Element mehrfach vorkommen darf. In einer Datenbanktabelle mit Attributen schließen sich verschiedene exklusive Attributausprägungen desselben Attributs aus. Bei einer Implementierung sollte dies berücksichtigt werden, um nicht etwa verbotene Kombinationen zu überprüfen.

In der Abb. 21.1 ist das Prinzip der (1) Assoziationsregeln und der (2) Generalisierungsregeln dargestellt. Die beiden Regelparadigmen werden im nächsten Abschnitt in einheitlicher Notation präsentiert.

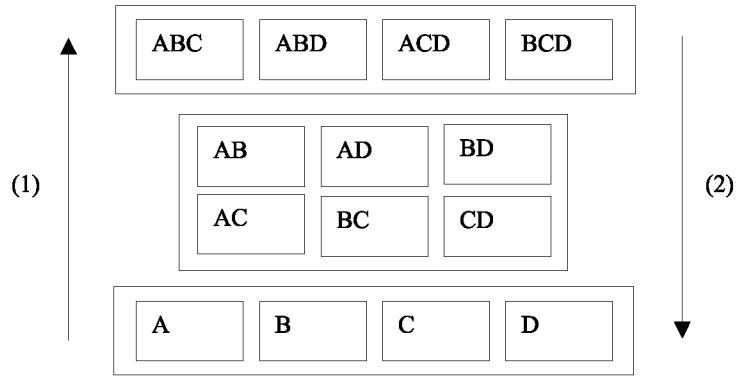

Abbildung 21.1. Prinzip der (1) Assoziationsregeln, (2) Generalisierungsregeln. Zu sehen sind die Mengen von Itemsets mit den Items A,B,C,D

21.2

21.2 Grundbegriffe und Anwendungen

Wir starten mit den wichtigsten Definitionen, die für die Assoziationsregelgenerierung und deren Gütebewertung benötigt werden. Die Definition und der englische Sprachgebrauch in der Def. 21.1 beziehen sich auf die Ideen in [AS94].

21.1

Definition 21.1 (Assoziationsregel)
Es seien elementare Attributausprägungen vorgegeben, sog. **Items**. I, J und K seien Mengen von Items, sog. **Itemsets** (oder auch **Muster**), die in der Menge $\mathcal{IS}$ zusammengefasst werden. „$\sharp$" bezeichne im Folgenden die Anzahl der Elemente einer Menge.

a) Die **Häufigkeit** (engl.: frequency) $freq(I)$ des Itemsets $I \in \mathcal{IS}$ ist das Verhältnis der Anzahl derjenigen Itemsets $K \in \mathcal{IS}$, die das Itemset I enthalten, zur Gesamtzahl der Itemsets, d.h.

$$freq(I) := \frac{\sharp\{K \in \mathcal{IS} | I \subset K\}}{\sharp\mathcal{IS}} = \frac{\sharp\mathcal{IS}(I)}{\sharp\mathcal{IS}} \quad . \tag{105}$$

Ist $\min_{freq} \in [0,1]$ eine untere Schranke für die Häufigkeit, so heißt I **häufig** (engl.: frequent), wenn $freq(I) \geq \min_{freq}$ gilt.

b) Der Ausdruck $I \Rightarrow J$ heißt **Assoziationsregel** (kurz: Regel) bezüglich der Menge der Itemsets.

c) Die **Häufigkeit** $freq(I \Rightarrow J)$ der Regel $I \Rightarrow J$ ist definiert durch $freq(I \cup J)$. Aufgrund der Kommutativität des Vereinigungsoperators gilt $freq(I \Rightarrow J) = freq(J \Rightarrow I)$. Ist $\min_{freq} \in [0,1]$ eine untere Schranke für die Häufigkeit, so heißt die Regel $I \Rightarrow J$ **häufig**, wenn $freq(I \Rightarrow J) \geq \min_{freq}$ gilt. Mit $Freq$ werde die Auflistung der häufigen Regeln bezeichnet.

d) Die **Konfidenz** (engl.: confidence) einer Regel $I \Rightarrow J$ wird definiert als die Anzahl derjenigen Itemsets $K \in \mathcal{IS}$, die die Items von I und J enthalten, geteilt durch die Anzahl derjenigen Itemsets K, die nur I enthalten:

$$conf(I \Rightarrow J) := \frac{\sharp\{K \in \mathcal{IS} | (I \cup J) \subset K\}}{\sharp\{K \in \mathcal{IS} | I \subset K\}} = \frac{\frac{\sharp\mathcal{IS}(I \cup J)}{\sharp\mathcal{IS}}}{\frac{\sharp\mathcal{IS}(I)}{\sharp\mathcal{IS}}} = \frac{freq(I \cup J)}{freq(I)} \quad . \tag{106}$$

Ist $\min_{conf} \in [0,1]$ eine untere Schranke für die Konfidenz, so heißt $I \Rightarrow J$ **konfident** (engl.: confident), wenn $conf(I \Rightarrow J) \geq \min_{conf}$ gilt. Mit $Conf$ werde die Auflistung der konfidenten Regeln bezeichnet.

e) Zielsetzung: In Abhängigkeit der Anwendung wähle man die Schranken $\min_{freq}$ und $\min_{conf}$. Gesucht sind alle Assoziationsregeln $I \Rightarrow J$, so dass $I, J \neq \emptyset$, $I \cap J = \emptyset$, $freq(I \Rightarrow J) \geq \min_{freq}$ und $conf(I \Rightarrow J) \geq \min_{conf}$ gilt. Je nach Anwendung können weitere Gütemaße berücksichtigt werden.

□

Die Assoziationsregelmechanismen erzeugen die klassischen „Warenkorbregeln" aus dem Bereich des kommerziellen Data Mining (vgl. [Bol96]): „Wenn Windeln gekauft, dann auch Bier gekauft." Ein Ladenbesitzer kann bei einer hohen Konfidenz dieser Regel die Waren Windeln und Bier nebeneinander stellen, was ihm u.U. einen Mehrumsatz einbringen kann und dem Kunden Laufarbeit spart. Ob diese Regel *tatsächlich* einen Nutzen bringt, ist nicht sicher, da das Optimierungsproblem „Positioniere alle Waren optimal, so dass ein maximaler Umsatz erzielt wird!" nicht durch das Umstellen einzelner Waren gelöst wird, d.h. der Ladenbesitzer muss die durch die Assoziationsregeln generierte *Hypothese* überprüfen. Weitere Anwendungen von Assoziationsregeln sind:

- Entdeckung von Alarmursachen in Telekommunikationsnetzen [HKM+96],
- Beschreibung von Herzkrankheiten [O+01],

- Wettervorhersage [Höp01] und
- Text-Mining in medizinischen Texten [BP01].

21.3

21.3 A-priori-Algorithmus

Nachdem wir nun wissen, was Assoziationsregeln sind und wo man sie beispielsweise anwenden kann, klären wir als nächstes wie man sie erzeugt. Der A-priori-Algorithmus [AS94] kann anhand seines einfachen Grundprinzips zur Regelgenerierung gut erklärt werden.

21.2

Algorithmus 21.2 (A-priori-Algorithmus)
0) Initialisierung: Sei M_1 die Menge aller Items, die die Minimumbedingung der Häufigkeit erfüllen und $k = 1$.

1) Inkrementiere k. Bilde die Menge M_k aller Itemsets der Länge k aus den Elementen der Menge M_{k-1}, d.h. erzeuge die Kandidaten.

2) Entferne aus der Menge M_k alle Itemsets, die die Minimumbedingung der Häufigkeit nicht erfüllen und gehe wieder zu Schritt 1, bis die maximale Länge l der Itemsets erreicht ist (=maximale Attributanzahl). In diesem Fall fahre fort mit Schritt 3.

3) Gib die Menge $M = \bigcup_{k=1,\ldots,l} M_k$ als Menge der häufigen Itemsets aus.

4) Nachbearbeitung: Berechne die Konfidenz der häufigen Itemsets und prune die nicht konfidenten Itemsets.

□

Es ist gesichert, dass alle häufigen Itemsets gefunden werden, obwohl im Schritt 2) Kandidaten-Itemsets entfernt werden, weil gilt: Erfüllen die Itemsets I, J mit $I \cap J = \emptyset$ *nicht* die Bedingung $freq(I) \geq \min_{freq}$, $freq(J) \geq \min_{freq}$, so erfüllt auch das Itemset $I \cup J$ *nicht* die Bedingung $freq(I \cup J) \geq \min_{freq}$.

21.3

Beispiel 21.3 Wir betrachten erneut Tab. 20.1. Die Häufigkeit der Attributausprägung „niedrig“ n_1 des Attributs As1 ist $freq(As1 = n_1) = 5/10$, die

von „hoch“ h_1 ist $freq(As1 = h_1) = 5/10$. Die Häufigkeit der Attributausprägung „niedrig“ n_2 des Attributs As2 ist $freq(As2 = n_2) = 7/10$, die von „hoch“ h_2 ist $freq(As2 = h_2) = 3/10$. Hätten wir $\min_{freq}$ auf 0.35 gesetzt, so würden wir $As2 = h_2$ nicht mehr betrachten, so dass wir nur die Kombinationen $As1 = n_1, As2 = n_2$ und $As1 = h_1, As2 = n_2$ betrachten müssen als Itemsets der Länge 2. Es ist $freq(As1 = n_1, As2 = n_2) = 4/10$ und $freq(As1 = h_1, As2 = n_2) = 3/10$. Die zweite Regel erfüllt nicht die Minimumbedingung der Häufigkeit, so dass nur das Itemset $I := (As1 = n_1, As2 = n_2)$ häufig ist. Wie konfident ist I? Es ist $conf_{\text{krank}}(I) = 0$ und $conf_{\text{gesund}}(I) = 1$. Wir hätten eine sichere Regel für „gesund“, aber keine Regel für „krank“.

□

Aktivierungselement: Die Häufigkeit ist eine monotone Eigenschaft. Deshalb ist das Pruning in den Kandidatenmengen möglich. Die Konfidenz ist nicht monoton. Konstruieren Sie ein Beispiel.

Aktivierungselement: Im Internet finden Sie einige Implementierungen des A-priori-Algorithmus. Probieren Sie den Algorithmus einmal auf einem Datensatz Ihrer Wahl aus.

21.4 Erweiterungen

Wichtige Erweiterungen der Assoziationsregelgenerierung sind:

- Verwendung der Regeln zur Klassifikation [LHM98], [YH03],
- Verwendung metrischer Daten [SA96],
- Integration von Einschränkungen (engl.: constraints) des Suchraums in die Regelgenerierung [SVA97],
- Analyse von Zeitreihendaten [AS95],
- Berechnung von Interessantheitsmaßen für Regeln [HH99],
- Sampling der Daten, um die Geschwindigkeit der Regelerzeugung zu erhöhen [Toi96, Kap. 5],
- Parallelisierung der Regelerzeugung [Zak98] und
- Visualisierung der Regeln [WWT99].

Zwei aktuelle Anwendungen aus der Bioinformatik findet der Leser im Kap. 25.

Möchte man die Assoziationsregeln zur Klassifikation wie in [LHM98] einsetzen, so beinhaltet das Itemset J die Klassenzugehörigkeiten $\{K_1, \ldots, K_m\}$, aber keine allgemeinen Attribute.

Die Kandidaten sind Itemsets, die gebildet werden, die aber nicht notwendig die Gütebedingungen erfüllen und somit eine Überprüfung dergleichen stattfinden muss. Verbesserungen betreffen vor allem die Erzeugung der Kandidaten oder besser gesagt die Vermeidung der Erzeugung der Kandidaten und die Verwendung effizienter Datenstrukturen innerhalb von Datenbanken.

Ein alternativer Algorithmus zur Erzeugung häufiger Muster ist ein auf sog. **Frequent-Pattern-Bäumen** (kurz: FP-Baum) [HPY00] beruhender, der ohne Kandidatenerzeugung auskommt. Im Prinzip ist der FP-Baum wie ein Präfixbaum mit Zählern aufgebaut, so dass nicht häufige Itemsets geprunt werden können. Für die Erzeugung aller häufigen Itemsets müssen rekursiv Sub-FP-Bäume aufgebaut werden. In [PB01] wird beispielsweise ein FP-Baum-Algorithmus zur Erstellung von Regeln mit medizinischen Daten verwendet.

Aktivierungselement: Betrachten Sie die Attributausprägungen der beiden Variablen in der Tabelle 20.1. Ordnen Sie die Attributausprägungen nach ihrer Häufigkeit und anschließend die Datentupel so, dass diejenigen mit häufigeren Attributen an vorderer Stelle stehen. Sind bei einem Attribut mehrere Ausprägungen gleich häufig, so wird nach dem anderen Attribut geordnet. Der erste Knoten in einem von Ihnen zu erstellenden Präfix-Baum sei nun ein Dummy-Knoten, der nur zur Verzweigung dient. Fügen Sie die geordenten Datentupel so ein, dass in jedem Knoten nur eine Attributausprägung steht und inkrementieren Sie bei Null beginnend einen Zähler für jede Attributausprägung. Beschreiben Sie den Ergebnisbaum im Vergleich zum Entscheidungsbaum.

Daten in einer relationalen Datenbank werden üblicherweise durch Attribute charakterisiert. Die Häufigkeit und die Konfidenz verschiedener Mengen von Attributausprägungen (Itemsets) liefern Hinweise auf typische Gemeinsamkeiten der Daten. Die Zusammenhänge zwischen Itemsets können durch

Assoziationsregeln beschrieben werden. Zur Generierung solcher Regeln werden häufig Varianten des A-priori-Verfahrens verwendet. Hierzu haben wir Ihnen Beispiele und Anwendungsgebiete vorgestellt. Es existieren aber auch andere Ansätze zur Generierung Attribut-orientierter Regeln.

Kapitel 22

Ausblick auf Zeitreihen

22

22

22 Ausblick auf Zeitreihen

Bislang waren unsere Problemstellungen statisch, d.h. nicht von der Zeit abhängig. Viele Phänomene in der Datenanalyse sind aber zeitabhängig, z.B. Molekülbewegungen oder Signalpfade. Wir geben im nächsten Abschnitt einen kurzen Überblick über die Zeitreihenproblematik und besprechen anschließend eine Anwendung in der Bioinformatik.

Lehrziele:
- Problemstellungen kennen, die eine Zeitreihenanalyse notwendig machen,
- die Fenster-Technik verstehen und beispielsweise im Rahmen von Assoziationsregeln und neuronalen Netzen anwenden können,
- wissen, dass es statistische Verfahren zur Zeitreihenanalyse gibt,
- das Beispiel einer Rekonstruktion einer Zeitreihenanalyse für Microarray-Daten kennenlernen.

22.1 Grundideen

Zeitreihen treten in natürlicher Weise als Daten auf, die eine temporäre Information tragen. Modellieren kann man die Zeitreihendaten durch einen zusätzlichen **Zeitstempel** [TJS00]. Wir geben einige Beispiele für Daten, die zeitabhängig sind:

- Signaldaten, z.B. Sprache, Video (als Sequenz von Einzelbildern), EKG (Herzsignal),
- allgemeine Daten, z.B. Wetterdaten (zur „Wettervorhersage"), Börsendaten („Aktienkurse"),
- Patientendaten, z.B. über einem Krankheitsverlauf,
- Molekülbewegungen, z.B. bei Docking-Abläufen,
- Sequenzen, die über eine längere Zeit hinweg mutieren („Phylogenie"),
- Abläufe in regulatorischen Pfaden und
- Objekte im Universum.

Zwei wichtige Probleme sind die *Modellierung* einer Zeitreihe und die *Vorhersage* von zukünftigen Werten der Zeitreihendaten bei vorgegebenen Messungen. Grundsätzlich möchte man aus der Vergangenheit etwas über die Zukunft lernen.

In der statistischen Literatur gibt es zahlreiche Modellvorschläge, z.B. die *Autoregressive Moving Average (kurz: ARMA)-Modelle*, deren Parameter anhand der Daten berechnet werden müssen, siehe beispielsweise [SS95], [Ton90].

Wir können an dieser Stelle nur zwei allgemeine, einfache Methoden zur Auswertung von Zeitreihen beschreiben, die auf der **Fenster-Technik** (engl.: windowing) beruht: Wenn man regelmäßig gemessene Werte hat, z.B. die jede volle Stunde gemessen werden, kann man eine *Fenster-Größe* m festlegen, z.B. $m = 6$ [h] und dieses Fenster immer um eine Zeiteinheit verschieben, beispielsweise um 1h. Man erhält dann m-dimensionale Datentupel der Form $(v_1, \ldots, v_m), (v_2, \ldots, v_{m+1}), (v_3, \ldots, v_{m+2})$ usw.

Man kann nun mit diesen Tupeln Assoziationsregeln in der vorgestellten Weise erzeugen. Man erhält dann Regeln der Form [AS95]:

WENN $A_{j_1} = x_{j_1}$ (zum Zeitpunkt j_1) UND ... UND $A_{j_k} = x_{j_k}$ (zum Zeitpunkt j_k) DANN Conclusio

mit $\{j_1, \ldots, j_k\} \subset \{1, \ldots, m\}$. Diese Methode entdeckt Muster in Zeitreihen, die statisch und wiederkehrend sind und die immer auf dasselbe Ereignis (Conclusio) hindeuten, z.B. auf einen Netzausfall in Telekommunikationsnetzen [HKM+96].

Setzt man für einen Fenster-Tupel die Klasseninformation auf den jeweils nächsten Wert in der Zeitreihe, so kann man eine Zeitreihenprognose mit einem NN oder einem Neuro-Fuzzy-System trainieren, vgl. Abb. 22.1. Regeln können dann in der gewohnten Weise extrahiert werden. Eine Erweiterung der U-Matrix-Methode zur Regelgenerierung mit Zeitreihendaten findet man in [GU99].

Erwähnt werden soll auch noch der Vergleich zweier Zeitreihen. Vermutet man einfache lineare Zusammenhänge, so kann man *Kreuzkorrelationen* ausrechnen. Man kann auch versuchen, mit Hilfe von *Ähnlichkeitsmaßen* die Distanz zweier Zeitreihenabschnitte zueinander zu vergleichen. Schwierig wird das Ganze dadurch, dass a) die Zeitreihen *verrauscht* sein können bzw. *Artefakte* vorhanden sein können und b) die Zeitreihen evtl. *skaliert* werden müssen, weil z.B. alle Muster in der zweiten Zeitreihe eine andere Länge haben. Viel schwieriger ist die Analyse von Zeitreihen, bei denen in unregelmäßigen Abständen Daten gemessen werden, da man dann keine Fenster zur Verfügung hat.

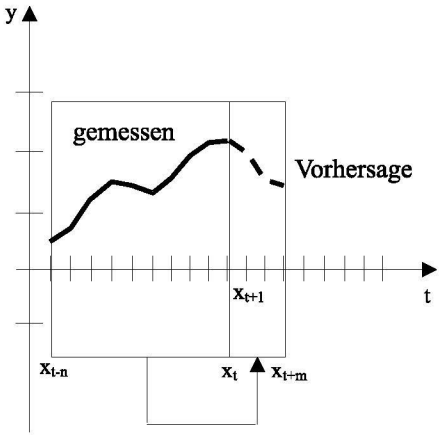

Abbildung 22.1. Schema einer Zeitreihenprognose

Unsere Absicht dieses Kapitels beschränkt sich auf die Darstellung der Idee einer Zeitreihenanalyse. Für umfangreichere Studien zu spezielleren Themen wird auf die Literatur verwiesen. Wir wollen aber im Folgenden noch auf ein Beispiel zum Thema Zeitreihen in der Bioinformatik eingehen.

22.2 Rekonstruktion einer Zeitreihe aus Microarray-Daten

22.2

Einen Ansatz des ML zur Rekonstruktion einer Zeitreihe aus *ungeordnet* vorliegenden Micoarray-Daten wird in [MLK03] beschrieben. Ausgegangen wird von verrauschten Messungen ohne Zeitinformation, die zudem nicht häufig erfolgt sind. Wären die Daten nicht verrauscht und häufig gemessen, so könnte man z.B. einen Polygonzug durch die Punkte legen. Bei verrauschten Daten ist eine Permutation σ gesucht, die die ursprünglichen n Messungen $x(1), x(2), \ldots, x(n)$ in eine (möglichst richtige) Zeitreihe $x(\sigma(1)), x(\sigma(2)), \ldots, x(\sigma(n))$ transformiert.

Algorithmus 22.1 (Rekonstruktion einer Zeitreihe) 22.1

1. Finde den minimalen Spannbaum T im gewichteten Graphen, der aus den Messungen und den mit einer Unähnlichkeit versehenen Kanten besteht.

2. Bestimme den diametrischen Pfad P, d.h. einen durchgehenden Pfad ohne Verzweigungen.

3. Falls P fast alle Knoten besucht, so ist P der Pfad der geordneten Zeitreihe. Die Verzweigungen werden nicht berücksichtigt.

4. Lässt P viele Knoten aus oder gibt es lange Verzweigungsketten, so schreibt man die Messungen der Teilpfade rekursiv in einen PQ-Baum. Die Äquivalenzklassen des PQ-Baumes sind gültige Permutationen der Zeitreihenmessungen.

□

Die Vorhersage von Zeitabläufen kann tatsächliche Messungen ersparen, die gerade bei Zeitreihen wegen der notwendigen häufigen Messungen teuer wären. Selbstverständlich muss man die Zeitreihe vorhersagen, wenn der Varlauf in der realen Zukunft interessiert, wie z.B. bei der Prognose von Krankheitsverläufen, da man keine zukünftigen Messungen zum gegenwärtigen Zeitpunkt vornehmen kann. Die Fenster-Technik ist eine weit verbreite Möglichkeit aus vorangegangenen Messungen einen zukünftigen Verlauf zu simulieren. Innerhalb von Sequenzen könnte man mit der Fenster-Technik Bereiche vorhersagen, die von besonderem Interesse sind. Microarray-Daten kann man ebenfalls in ihrer zeitlichen Entwicklung betrachten. An dieser Stelle können wir leider die Zeitreihenanalyse, für die auch viele statistische Ansätze existieren, nicht weiter vertiefen.

Kapitel 23

Generalisierungsregeln

23

23

23 Generalisierungsregeln

Hat man Datensätze vorliegen, die durch ihre Attribute und Klassenzugehörigkeit beschrieben werden, so kann man umgekehrt zur Vorgehensweise bei der Erzeugung von Assoziationsregeln fragen, ob man diese Datensätze *generalisieren* kann. Diese Fragestellung ist schon älter, und sie ist sehr naheliegend. Im nächsten Abschnitt stellen wir das Versionsraumlernen und dessen Eigenschaften vor. Dann gehen wir auf Varianten ein, bevor wir im Abschnitt 23.4 ausführlicher einen Ansatz zur heuristischen Erzeugung von Generalisierungsregeln vorstellen.

Lehrziele:
- Verstehen, was mit einem Konzept gemeint ist,
- definieren können, was der Versionsraum ist,
- die grundlegende Idee des Versionsraum-Lernverfahrens verstehen,
- die Ideen des AQ- und CN2-Verfahrens zur Kenntnis nehmen,
- die Idee der hierarchischen Generalisierung mit Generalisierungsbäumen kennen,
- wissen, dass man Generalisierungsregeln mittels Durchschnittsbildung heuristisch erzeugen kann,
- den Begriff des Abschluss eines Itemsets kennen,
- die Probleme nennen können, die bei der Generalisierung mit den verschiedenen Verfahren auftreten.

23.1 Versionsraumlernen

23.1

Ein Lernverfahren, das die Generalisierung und die Spezialisierung beinhaltet, ist das *Versionsraumlernen* (kurz: VR-Lernen) [Mit77]. Es handelt sich beim VR-Lernen nicht um ein Verfahren des Soft Computing, da es nicht mit vagen oder fehlerhaften Daten umgehen kann. Wir wollen es nur in gebotener Kürze einführen, um die historischen Wurzeln modernerer Verfahren kennenzulernen. Viele der eingeführten Begriffe werden auch heute noch verwendet.

Definition 23.1 (Versionsraum) 23.1

a) Ein **Konzept** kann man formal einführen als Abbildung κ einer Grundmenge M nach $\{0, 1\}$. Ein Sample (Beispiel, Instanz) x gehört zum Konzept, falls $\kappa(x) = 1$ ist, sonst nicht. Man nennt die Menge aller positiven Instanzen auch **Extension**.

b) Ein Konzept heißt **vollständig**, falls alle positiven Samples abgedeckt werden. Es heißt **korrekt**, falls kein negatives Beispiel abgedeckt wird. Es heißt **konsistent**, falls es vollständig und korrekt ist.

Es wird nun beim Konzeptlernen die induktive Lernaufgabe gestellt, ein konsistentes Konzept für Samples zu finden.

c) Ein Konzept κ_1 heißt **allgemeiner** als ein Konzept κ_2, falls für alle Samples x gilt: $\kappa_2(x) = 1 \Rightarrow \kappa_1(x) = 1$, d.h. das erste Konzept deckt mehr Beispiele ab. Das zweite Konzept ist dann **spezieller** als das erste.

d) Der **Versionsraum** enthält alle korrekten und vollständigen (also konsistenten) Konzepte.

□

Die speziellsten Konzepte sind immer die Beispiele selbst. Das allgemeinste Konzept ist dasjenige, das alle Beispiele abdeckt. Als Schreibweise führen wir dafür einen Stern ein, der als Platzhalter für ein Attribut steht, also z.B. bei 4 Attributen: ****.

Klar ist, dass das Durchsuchen aller Kombinationen von Attributen bzw. deren Platzhaltern aufgrund der kombinatorischen Explosion nicht effizient ist. Hätte man nur drei Attribute mit je zwei Ausprägungen (große und kleine Buchstaben), so müsste man bereits *abc*, *abC*, *aBc*, *Abc*, *aBC*, *AbC*, *ABc*, *ABC*, *ab**, *aB**, *Ab**, *AB**, *a*c*, *a*C*, *A*c*, *A*C*, **bc*, **Bc*, **bC*, **BC*,***c*, ***C*, **b**, **B**, *a***, *A***, *** auf Konsistenz überprüfen.

Das Versionsraum-Lernverfahren (oder auch: **Kandidaten-Eliminations-Methode**) ([Mit97, S. 32], [BKI00, S. 126] oder [Kel00, S. 311]) arbeitet inkrementell bezüglich präsentierter Beispiele, die positiv oder negativ sein können.

23.2 **Algorithmus 23.2 (Kandidaten-Eliminations-Methode)**

a) schrittweise *Generalisierung*, angefangen von den speziellen Beispielen: Jedesmal wenn ein positives Beispiel noch nicht abgedeckt wird, muss soweit generalisiert werden, dass das Beispiel abgedeckt wird.

b) schrittweise *Spezialisierung*, angefangen von dem allgemeinsten Konzept: Jedesmal wenn ein negatives Beispiel irrtümlich abgedeckt wird, muss soweit spezialisiert werden, dass dieses negative Beispiel nicht mit abgedeckt wird.

Nach iterativer Ausführung der Schritte a) und b) stimmen die erhaltenen Mengen aus a) und b) überein.

□

Leider können identische Beispiele, die z.B. zu 90% zu Klasse 1 gehören und zu 10% zu Klasse 2, nicht mit dem Versionsraumlernen verarbeitet werden, obwohl eine Regel der Art „WENN Beispielattribute = ... DANN Klasse 1 mit 90%-iger Korrektheit" durchaus Sinn macht. Verfahren, die ungenaue, verrauschte Daten nicht verarbeiten können, sind keine guten Beispiele für Soft Computing, sondern eigentlich das Gegenteil davon, da beim Soft Computing ja oft gerade ungenaue Daten ausgewertet werden sollen.

Aktivierungselement: Machen Sie sich die Kandidaten-Eliminations-Methode anhand eines Beispiels klar.

Wir wollen noch kurz darauf hinweisen, dass eine Generalisierung von Attributen nicht unbedingt bedeuten muss, dass alle Attributausprägungen zu einem * generalisiert werden müssen, z.B. kann man „Katja", „Claudia" zu „Frau" verallgemeinern und „Peter", „Hans" zu „Mann" anstelle von * für alle Menschen.

23.2 AQ- und CN2-Verfahren

23.2

Der **AQ-Algorithmus** von Michalski arbeitet ähnlich wie das Versionsraumlernen, allerdings sequentiell: Es wird versucht, eine Teilmenge von positiven Beispielen (z.B. aus einer Klasse) abzudecken, diese also zu verallgemeinern. Das Gleiche geschieht mit der Restmenge. Nach und nach werden so die positiven Beispiele abgedeckt.

Diese Teilmenge wird **Stern** genannt; genauer: Ein Stern $G(x, N)$ zu einem positiven Beispiel x bezüglich der Menge der gesamten negativen Beispielmenge N ist der Versionsraum, der durch das Versionsraumlernen entsteht.

Der Algorithmus hat damit dieselben Nachteile wie das VR-Lernen. Da ein Stern sehr groß werden kann, wird nachbearbeitet, d.h. es werden die „besten" Verallgemeinerungen ausgewählt, die z.B. nur wenige Attribute besitzen. Es gibt eine Reihe von unterschiedlichen Versionen bis zur Version AQ20. Wir wollen das aber hier nicht vertiefen.

Eine Verbindung von AQ-Algorithmen und Entscheidungsbäumen bietet das CN2-Verfahren [CN89]. Neu ist hier, dass bei der Erzeugung eines Sterns nur die im informationstheoretischen Sinn besten Prämissen im Stern bleiben. Dadurch wird das Verfahren, im Gegensatz zum reinen AQ-Verfahren, zu einem Verfahren des Soft Computing. Wer dies vertiefen möchte, möge in [Mit97] nachschlagen.

23.3 Hierarchische Generalisierung

Bei der hierarchischen Generalisierung [BLH99] werden ausgehend von den Trainingsbeispielen (unterste = 0. Schicht im entstehenden Baum) immer zwei Trainingsbeispiele um je ein Attribut generalisiert. Z.B. wird aus den Itemsets mit je drei Attributausprägungen $a_1b_3c_4$ und $a_1b_2c_4$ in der 0. Schicht das Itemset $a_1 * c_4$ in der 1. Schicht, wobei $*$ für $\{b_3, b_2\}$ steht. Je zwei Itemsets aus niedrigeren Schichten können bezüglich einem Attribut generalisiert werden. Das Ergebnis ist ein **Generalisierungsbaum**. Jeder Knoten im Baum enthält eine feste Regel für eine fest gewählte Klasse. Je weiter oben der Knoten im Baum ist, desto allgemeiner ist die Regel. Der Algorithmus wurde zur Erkennung von Kreditkartenbetrug angewendet, ist also ein Anomalie-Entdeckungsalgorithmus, der Auffälligkeiten innerhalb einer Klasse (z.B. Betrug) erkennen kann.

23.4 Heuristische Generalisierungsregeln zur Klassifikation

Die Idee der heuristischen Generalisierung [Pae02c], [PB02a], [Pae02e] ist die Generalisierung von Itemsets um mehr als ein Attribut, wobei ungenaue Daten oder fehlende Daten verarbeitet werden können. Die Generalisierung geschieht mittels *Durchschnittsbildung*. Die entstehenden generalisierten Regeln können zur Klassifikation von Daten mehrerer Klassen verwendet werden, indem ein winner-takes-all-Verfahren auf der Basis der *gewichteten Konfidenz* angewendet wird. Die Problematik der kombinatorischen Explosion bei der Durchschnittsbildung wird durch Heuristiken vermindert. Das Verfahren hat gegenüber anderen genannten Verfahren Vorteile, insofern „echtes" Wissen

in den erzeugten Klassifikationsregeln enthalten ist, die mehrdimensionale Struktur der Daten ausgewertet wird, beliebig viele Klassen vorhanden sein dürfen, und ungenaue Daten verarbeitet werden können.

Betrachten wir ein einführendes Beispiel mit zwei Mengen, die Items enthalten, $I_1 = ABC$ (Abk. für $A \wedge B \wedge C$) und $I_2 = BCD$, z.B. $A =$ „grün“, $B =$ „groß“, $C =$ „4 Türen“, und $D =$ „schnell“. I_1 und I_2 beschreiben die Eigenschaften zweier Entitäten, z.B. zweier Autos. Wir suchen nun nach Gemeinsamkeiten der Autos. Beide Autos haben 4 Türen. Nur das erste Auto ist groß und das zweite schnell. Mengentheoretisch ausgedrückt haben wir den Durchschnitt von I_1 und I_2, $I_1 \cap I_2 = ABC \cap BCD = BC$, gebildet. Mengendurchschnitte bilden einen natürlichen Zugang zur Regelgenerierung. Wir verlangen nicht, dass alle Mengen von Items einer Klasse angehören. Ein fehlendes Item stört nicht bei der Durchschnittsbildung. Die Durchschnittsbildung erlaubt uns, um mehrere Attribute gleichzeitig zu reduzieren. Der Durchschnittsoperator kann als eine Art Rekombinationsoperator angesehen werden (vgl. Kap. 27).

Beispiel 23.3 Wir betrachten die acht Itemsets $ABCD$, $AEFG$, $BEFG$, $CEFG$, $DEFG$, BE, CF und DG. In [BTP$^+$00] wird der **Abschluss** (engl.: closure) $\mathrm{cl}(I)$ eines Items bez. des Itemsets I definiert als der Durchschnitt aller Itemsets, die I enthalten. Der Abschluss von A ist $\mathrm{cl}(A) = ABCD \cap AEFG = A$. Ist $\mathrm{cl}(I) = I$, dann nennen wir das Itemset I **abgeschlossen** (engl.: closed). In unserem Beispiel ist A abgeschlossen. Im Algorithmus [BTP$^+$00] werden alle Abschlüsse häufiger Itemsets berechnet, und zwar der Reihe nach für die Itemsets der Länge $1, 2, 3, \ldots$ Deshalb generiert dieser Algorithmus genau so viele Kandidaten wie A-priori. **23.3**

Wenn wir von einer Durchschnittsbildung ausgehen und die Durchschnitte direkt aus den Samples berechnen, erhalten wir direkt eine Wahrscheinlichkeit von $6/(7 + 6 + 5 + 4 + 3 + 2 + 1) = 6/28 \approx 0.21$ für das Erzeugen des abgeschlossenen, interessanteren Itemset EFG. Die Wahrscheinlichkeit für das Erzeugen des abgeschlossenen, weniger interessanten Itemset A beträgt nur $1/28$. Das Beispiel zeigt, dass es plausibel sein kann, anstelle von A-priori-Kandidaten heuristisch Generalisierungen zu erzeugen.

□

Im Folgenden sei $\mathcal{IS}$ eine endliche Menge von Itemsets und betrachten Samples als Itemsets. Identische Itemsets werden nur einmal unter Angabe ihrer Häufigkeit bezüglich jeder Klasse gespeichert. Wir bezeichnen einen Durchschnitt K zweier Mengen I, J als **nicht-trivial**, wenn $K \neq I$, $K \neq J$ und $K \neq \emptyset$ gilt. Wir verwenden die Häufigkeit und Konfidenz zur Bewertung der Regeln $I \Rightarrow c$, $I \in \mathcal{IS}$, mit c Klassenbezeichnung, wie in [AS94]. Wir schreiben $freq(I \Rightarrow c)$ bzw. $conf(I \Rightarrow c)$. Wir geben den grundlegenden Algorithmus „GenDurchschnitt" zum Auffinden von Generalisierungsregeln an, den wir später noch verändern werden.

23.4 Algorithmus 23.4 (GenDurchschnitt)

Eingabeparameter: Menge der Itemsets $\mathcal{IS}$, maxlevel, γ_i (Minimum-Schwellen für die Bewertungen)

Ausgabeparameter: $\mathcal{IS}^F$ (generalisierte Regeln; beinhaltet die initiale Menge $\mathcal{IS}$), level, startlevel.

1. Initialisierung
$\mathcal{IS}_{\text{new}} := \mathcal{IS}$;
level := 1;
startlevel(level) := 1;
endofalg := false;

2. Generiere Level
while endofalg = false **do**
  startlevel(level+1) := $\sharp(\mathcal{IS}_{\text{new}})+1$;
  oldIS := $\sharp\mathcal{IS}_{\text{new}}$;
  gehe aktuelles Level durch
  for i = startlevel(level) **to** startlevel(level+1)−2
    gehe alle Itemsets durch *ohne* Itemsets der
    vorhergehenden Level zu berücksichtigen;
    for $j = i + 1$ **to** startlevel(level+1)−1
      Inter := $\mathcal{IS}_{\text{new}}(i) \cap \mathcal{IS}_{\text{new}}(j)$;
      if Inter ist ein nicht-trivialer Durchschnitt
        and (Inter $\notin \mathcal{IS}_{\text{new}}$) **then**
        $\mathcal{IS}_{\text{new}}(\sharp\mathcal{IS}_{\text{new}} + 1)$:= Inter;
      end if, **end** for j, **end** for i

3. Teste Terminierung
  if ($\sharp\mathcal{IS}_{\text{new}}$ = oldIS) **or** (level ≥ maxlevel) **then**

endofalg := true;
$\mathcal{IS}^F := \mathcal{IS}_{\text{new}}$;
else level := level +1; **end**
end while

4. Filtere alle Regeln, die höhere Bewertungen haben als γ_i.

□

Der Nachteil dieses Vorgehens ist die kombinatorische Explosion bei der Durchschnittsbildung. Ein Prunen wie bei Assoziationsregeln ist nicht möglich, da die Häufigkeit nicht monoton bezüglich der Generalisierung ist. Deshalb werden wir im Folgenden eine Heuristik einführen. Wir setzen einen *äußeren Generalisierungsindex* ein, um zu bestimmen, ob ein weiteres Generalisierungslevel notwendig wird. Wir verwenden einen *inneren Generalisierungsindex*, um festzustellen, ob innerhalb eines Levels die Durchschnittsbildung abgebrochen werden kann. Zur Berechnung des inneren Generalisierungsindex verwenden wir einen gleitenden Mittelwert, der aus der Anzahl der neu generierten Itemsets pro Itemset erstellt wird. Wird er zu klein, fahren wir mit dem nächsten Level fort. Wir kombinieren dies mit einer maximalen Anzahl max_{new} von Itemsets, die aus einem Itemset erzeugt werden dürfen.

Definition 23.5 (Generalisierungsindices) **23.5**

a) Wir nennen den o.g. gleitenden Mittelwert M, den **inneren Generalisierungsindex** $G^{(in)} := M$.

b) Sei m die maximale Anzahl neu generierter Itemsets in einem Level und e die Anzahl der tatsächlich generierten neuen Itemsets, die nicht nicht-trivial sind und nicht bereits vorhanden sind. Dann definieren wir den **äußeren Generalisierungsindex** als

$$G^{(out)} := \frac{e}{m} \quad . \tag{107}$$

□

Fällt $G^{(out)} \in [0, 1]$ unter einer vordefinierte Schwelle, dann terminiert der Algorithmus. Liegt der Index nahe bei 1, so fährt der Algorithmus mit einem weiteren Level fort.

Wir wollen die Regelgenerierung durch Durchschnittsbildung mit den Assoziationsregeln vergleichen. A-priori generiert sehr viele, auch überflüssige Regeln. Wenn wir die Regeln ABC, BCD der Klasse c betrachten, generiert GenDurchschnitt nur die eine zusätzliche Regel BC. A-priori geht von den Items A, B, C und D der Länge 1 aus und bildet Kombinationen der Länge 2. Die Kombination AD ist weder in ABC noch in BCD enthalten. Das Itemset BC ist häufig, aber auch die Items B und C. Wir benötigen eigentlich nicht alle häufigen Itemsets zur Klassifikation. Am folgenden Beispiel werden wir sehen, dass Assoziationsregeln zu einem unerwünschten Effekt führen, den wir als **Kontextverwischung** bezeichnen.

23.6 **Beispiel 23.6 (Kontextverwischung)**
Bezeichne B das Item „hoher Blutdruck“ und C das Item „niedriger pH-Wert“. Das entscheidende Wissen ist die *Kombination* BC beider Items. Würden wir nur B oder C als Klassifikationsregeln verwenden, so würde man glauben, dass ein hoher Blutdruck oder ein niedriger pH-Wert allein bereits entscheidend wären. Dies könnte sein, muss aber nicht! Würden wir nur B als Generalisierungsregel einem Arzt präsentieren, so könnte er zu einem falschen Schluss gelangen. Er könnte denken, dass die Verabreicherung eines Blutdruck-senkenden Mittels zum Vorteil des Patienten wäre, obwohl evtl. dieses Medikament den pH-Wert weiter absenken könnte zum Nachteil des Patienten. In diesem Sinn ist eine längere Generalisierungsregel geeigneter für den Arzt.

□

Im folgenden Beispiel gehen wir darauf ein, wie geeignet die erzeugten Regelmengen für die Klassifikation *unbekannter* Daten sind, d.h. wir beschreiben die *Robustheit* der Klassifikatoren.

23.7 **Beispiel 23.7 (Robustheit)**
Wir betrachten erneut die Itemsets ABC und BCD der Klasse c. GenDurchschnitt generiert BC; ein A-priori-Verfahren würde die Regeln B und C bevorzugen. Das unbekannte, zu klassifizierende Itemset sei nun AB. Es ist $BC \not\subset AB$, i.e. AB würde als „nicht klassifizierbar“ eingestuft werden. In der Tat könnte AB einer anderen Klasse als BC angehören, so dass diese Einteilung sinnvoll ist. Aber es ist $B \subset AB$, d.h. AB würde als Klasse d zugehörig klassifiziert werden, obwohl AB nicht zur Klasse d gehören muss. Würden wir

C anstelle von B nehmen, was sich als scheinbar äquivalente Wahl anbieten würde, so hätten wir $C \not\subset AB$. Also verhält sich B tatsächlich anders als C, wenn wir *unbekannte* Itemsets erwarten, was z.B. in der Medizin der Fall ist, da neue Patienten ein individuell anderes Verhalten zeigen können.

□

Kürzere A-priori-Regeln sind also nicht so robust wegen des Effekts der Kontextverwischung. Also können im Life Science-Bereich längere Generalisierungsregeln eine geeignete Alternative zu Assoziationsregeln sein.

Wir haben den Ansatz der Generalisierungsregeln auf eine Datenbank mit 362 Patienten mit septischem Schock angewendet. Jeder Patient entspricht einem Itemset, das sich aus den Aufnahmedaten und täglichen Messungen zusammensetzt. Insgesamt standen 96 Items zur Verfügung, die wir mit Hilfe der folgenden *Wichtigkeit* auf 27 Items reduziert haben.

Definition 23.8 (Wichtigkeit) **23.8**

$$imp_c(A) := \sum_{i=1, A \in I_i}^{r_l} freq(I_i \Rightarrow c)\ \ c\text{-}wconf(I_i \Rightarrow c) \frac{1}{|I_i|} \qquad (108)$$

□

Dabei bezeichnet $c\text{-}wconf$ die gewichtete Konfidenz, die die Häufigkeiten der verschiedenen Klassen berücksichtigt.

Definition 23.9 (Gewichtete Konfidenz) **23.9**

Die **gewichtete Konfidenz** $wconf$ für die Klasse c kann durch die Konfidenz und den Klassenwahrscheinlichkeiten definiert werden oder einfacher durch die c-Häufigkeiten unter Berücksichtigung aller Klassenhäufigkeiten:

$$c\text{-}wconf(I \Rightarrow c) := \frac{c\text{-}freq(I \Rightarrow c)}{\sum_{d=1}^{m} d\text{-}freq(I \Rightarrow d)} \qquad (109)$$

□

Ohne auf die genauen Klassifikationsergebnisse einzugehen, geben wir eine Beispielregel an:

If nicht traumatisch **and** Anzahl der Organversagen > 2 **and** beatmet **and** Hämofiltration **and** keine Dialyse **and** Katecholamine verabreicht **then** verstorben ($wconf = 89.73\%$, $freq = 6.63\%$).

Insgesamt haben wir mit den Assoziations- und Generalisierungsregeln zwei unterschiedliche Möglichkeiten zur Regelgenerierung und Klassifikation kennengelernt. Obwohl die Idee der Generalisierung schon älter ist, sind die auf diesem Paradigma beruhenden Verfahren mittlerweile etwas weniger verbreitet. Die kombinatorische Explosion der möglichen Itemsets kann im Vergleich zu Assoziationsregeln nicht sehr effizient durch Heuristiken begegnet werden. Allerdings besteht an dieser Stelle noch Spielraum, Verfahren zu verändern oder zu kombinieren. Die Verwendung intelligenter Heuristiken kann dazu beitragen. Im folgenden Kapitel werden wir noch weitere Möglichkeiten des ML kennenlernen.

Kapitel 24

Weitere Verfahren des Maschinellen Lernens

24

24

24 Weitere Verfahren des Maschinellen Lernens

Wir präsentieren in diesem Kapitel zwei weitere bekanntere Ansätze aus dem ML, das *Case Based Reasoning* und die *Rough Set-Theorie*. Die Vorgehensweisen bieten ein Anwendungspotenzial im Rahmen der Bioinformatik.

Lehrziele:
- Verstehen, was sich hinter dem analogen Schließen verbirgt, insbesondere auch im Vergleich mit dem deduktiven, induktiven und abduktiven Schließen aus den vorherigen Kapiteln,
- die Idee des Case Based Reasoning wiedergeben können,
- verstehen, dass die Modellierung der Ähnlichkeit eine große Bedeutung im Case Based Reasoning hat und Kriterien für ein sinnvolles Ähnlichkeitsmaß angeben können,
- ein Beispiel zum Case Based Reasoning kennenlernen, dass auf einer Proteindatenbank basiert,
- die Idee der Rough-Set-Theorie kennen,
- die wichtigsten Begriffe der Rough-Set-Theorie, wie die untere und die obere Annäherung und den Ungenauigkeitsgrad einer Menge, definieren und berechnen können.

24.1 Analoges Schließen – Case Based Reasoning

24.1

Das **analoge Schließen** ist eine Inferenzmethode, bei der aufgrund von analogen Fakten Schlüsse gezogen werden: „WENN aus A folgt B DANN aus ungefähr$A$ folgt B“. Die Idee des analogen Schließens liegt dem **Case Based Reasoning** (Abk.: CBR, dt.: Fallbasiertes Schließen) [Kol93], [PDY01] zugrunde. Anwendungen sind z.B. Verarbeitung von Fällen in der Medizin (Krankheitsverläufe von Patienten), von juristischen Fällen oder von Sequenzdaten. Im CBR hat man es oft mit umfangreichen Fällen zu tun, die sehr individuell sind, d.h. die schwer generalisierbar sind. Deshalb definieren wir als Fallbasiertes System ein System, das eine Teilmenge der präsentierten Samples *unverändert* abspeichert. Die Verarbeitung wird durch den folgenden Algorithmus angegeben.

Algorithmus 24.1 (CBR) 24.1

1) Suche *ähnliche* Fälle in der Wissensbasis W zu einem präsentierten Fall F.

2) Überprüfe die Menge der angegebenen Fälle auf Verwendbarkeit für den neuen Fall F.

3) Integriere den neuen Fall (je nach Ergebnis der Überprüfung) in die Wissensbasis W, z.B. muss ausgewertet werden, ob F in W integriert werden sollte oder ob ältere Fälle aus W entfernt werden müssen.

□

Ein wichtiger Bestandteil eines CBR-Systems ist die Modellierung der **Ähnlichkeit**, die von der Realisierung der Fälle im System abhängt. Ähnlichkeit kann manchmal sehr einfach modelliert werden (z.B. die Ähnlichkeit zweier reeller Zahlen durch deren Differenz), aber nicht immer. Die Ähnlichkeit von komplexen Datenobjekten kann schwer zu bestimmen sein, z.B. die Ähnlichkeit zweier kranker Patienten, Ähnlichkeit des Verlaufs zweier Aktienkurse, Ähnlichkeit der Kriminalität zweier Verbrecher oder Ähnlichkeit der Funktion zweier Sequenzen. Die Modellierung komplexer Ähnlichkeiten kann i.d.R. nur heuristisch geschehen. Selbst im Fall zweier Zahlen könnte man eine Ähnlichkeit auch durch die Anzahl gemeinsamer Teiler definieren: Dann wären 4 und 8 ähnlicher zueinander als 4 und 5. Es kommt immer auf die Fragestellung an, die mit einem System bearbeitet werden muss.

Sinnvolle Festlegungen an eine Kennzahl für Ähnlichkeiten (zur besseren Interpretierbarkeit) können aber sein:

- Beschränktheit,
- Normiertheit zwischen [0,1], 0 = unähnlich, 1 = ähnlich,
- Linearität, d.h. lineare Bewertung zwischen [0,1], also 0.5 = halb ähnlich/halb unähnlich und nicht etwa 0.1 = schon fast vollständig ähnlich.

Eine Anwendung des CBR-Ansatzes im Rahmen der Bioinformatik ist z.B. [CW03]. Es wurden kodierende Sequenzabschnitte gespeichert, um für neue Sequenzen die kodierenden Regionen aufzufinden.
Manchmal kann es sinnvoll sein, für Teilaspekte verschiedene Ähnlichkeitskennzahlen $s_1, \ldots, s_l$ festzulegen und dann *gewichtet*, je nach Bedeutung, zu kombinieren. Wir geben ein einfaches Beispiel, um zu lernen, wie man Ähnlichkeiten kombinieren kann.

Beispiel 24.2 Wir wollen uns ein (einfaches) CBR-System für den Hauskauf erstellen. Fälle wären in diesem Fall Personen/Familien, die Häuser gekauft haben. Die Personen/Familien modellieren wir durch die Anzahl der Erwachsenen E und der Kinder K, durch das verfügbare Gesamteinkommen G und durch das ersparte Geld S. Die Häuser modellieren wir durch die Quadratmeter Wohnfläche W. Den Keller und den Garten berücksichtigen wir hier der Einfachhheit halber nicht. **24.2**

1) Ähnliche Personen/Familien: Wir definieren s_1 durch

$$s_1(E_1, K_1; E_2, K_2) := \frac{|\min\{E_1 + K_1, E_2 + K_2\}|}{E_1 + K_1 + E_2 + K_2}, \tag{110}$$

und behandeln damit Kinder wie Erwachsene.

2) Ähnliches Einkommen: Diese Ähnlichkeit modellieren wir binär, da wir vermuten, dass Personen mit deutlich unterschiedlichem Vermögen völlig andere Wünsche haben. Es sei also

$$s_2(G_1, S_1; G_2, S_2) := \begin{cases} 1 & , \text{ falls } G_1 \in [0.8G_2, 1.2G_2] \text{ und } S_1 \in [0.8S_1, 1.2S_2] \\ 0 & , \text{ sonst} \end{cases} \tag{111}$$

3) Kombination: Gehen wir davon aus, dass uns die Personen/Familienähnlichkeit wichtiger erscheint als das Vermögen, so können wir definieren

$$s_3 := 0.6s_1 + 0.4s_2 \ . \tag{112}$$

4) Hausähnlichkeit:

$$s_H(W_1, W_2) := \frac{\min\{W_1, W_2\}}{W_1 + W_2} \tag{113}$$

Bestehe z.B. die Familie F eines Professors aus zwei Erwachsenen und zwei Kindern mit einem Jahreseinkommen von 300 Sterntalern und 2000 Sterntalern Gespartes und habe der Single-Student K ein Einkommen von 30 Sterntalern und 5 Sterntalern Gespartes. Dann ergibt sich eine Ähnlichkeit von

$$s_H(F, K) = 0.6\frac{1}{4} + 0.4 \cdot 0 = 0.6 \cdot 0.25 = 0.15 \tag{114}$$

Wir können also nach unserem Modell davon ausgehen, dass unser Professor und der Student wenig Gemeinsamkeiten beim Hauskauf haben.

□

24.3 **Beispiel 24.3** Bei einer DNA-Sequenzanalyse können Fehler auftreten: mehrere falsche Nukleotide zuviel oder zuwenig können vorliegen (sog. *frameshift errors*). Die Fehlerrate wird i.Allg. unter 1% liegen, aber sie beeinträchtigt dennoch die richtige Analyse der Sequenz. Ein falsches Nukleotid kann durch partielle Alignments entdeckt werden. Solche Alignments, z.B. mit BLAST erstellt, entdecken aber keine verschobenen Leseraster. Deshalb werden in [Sha90] partielle Alignments zu einer bestehenden Proteindatenbank, also Fall-basiert, mit überlappenden Fenstern durchgeführt. Inkonsistente partielle Alignments können entdeckt werden und bleiben unberücksichtigt. Die Details und die Anwendung auf *E. coli*-Genen findet man in [Sha90].

24.2

24.2 Rough Set-Theorie

In der Rough Set-Theorie (grob übersetzt: Theorie von Mengenungenauigkeiten) [Paw82] werden Regeln durch (ungenaue) Annäherung an Beispieldaten generiert. Gehen wir von einem Universum U von Mengenobjekten aus, die einer Klasse $\{0,1\}$ angehören und sei I eine Relation, die sog. *Unterscheidbarkeitsrelation* (engl.: indiscernibility relation). In diesem Sinne sei $I(x)$ die Menge aller Elemente, die von x ununterscheidbar sind.

24.4 **Definition 24.4** Sei $X \subset U$.

a) Wir definieren die **I-untere** und die **I-obere Annäherung** von X als

$$I_*(X)=\{x \in U | I(x) \subset X\}, \tag{115}$$

$$I^*(X)=\{x \in U | I(x) \cap X \neq \emptyset\} \ . \tag{116}$$

b) Der **Rand** (engl.: boundary) von X ist definiert als $R_I := I^*(X) - I_*(X)$. Randelemente können z.B. nicht genau klassifiziert werden.

c) Gilt $R_I = \emptyset$, dann heißt X hier **genaue** Menge (engl.: crisp), ansonsten **ungenau** (engl.: rough).

d) Die Ungenauigkeit kann gemessen werden durch $\alpha_I(X) = \frac{|I_*(X)|}{|I^*(X)|} \in [0,1]$.

e) Wir definieren den **Ungenauigkeitsgrad** (engl.: degree of rough membership) als $\mu_X^I(x) := \frac{|X \cap I(x)|}{|I(x)|}$. Es gilt dann: $I_*(X) = \{x \in U | \mu_X^I(x) = 1\}$,

Tabelle 24.1. Medizinische Daten. K=Kopfschmerz, M=Muskelschmerz, T=Temperatur

Patienten-Nr.	K	M	T	Kl.= Grippe
1	n	y	hoch	j
2	y	n	hoch	j
3	y	y	sehr hoch	j
4	n	y	normal	n
5	y	n	hoch	n
6	n	y	sehr hoch	j

$I^*(X) = \{x \in U | \mu_X^I(x) > 0\}$ und $R_I(X) = \{x \in U | 0 < \mu_X^I(x) < 1\}$. Insofern könnte man $I_*(X)$ als Kern und $I^*(X)$ als Träger bezeichnen.

f) Unter einem **Redukt** (engl.: reduct) einer Menge von Ununterscheidbarkeitsrelationen verstehen wir eine Untermenge dieser Relationen ohne Veränderung der Relationsaussage, m.a.W. haben wir Attribute, die eine Menge von Datensätzen einer bestimmten Klasse beschreiben. So ist ein Redukt eine Untermenge von Attributen, die diese Datensätze genauso gut beschreiben.

□

Beispiel 24.5 (aus [Paw82], j = ja, n = nein) **24.5**

In der Tabelle 24.1 sind die Patienten 2,3,5 ununterscheidbar bezüglich des Attributs K, da sie die j-Ausprägung haben. Ununterscheidbare Patientenmengen bezüglich der Attribute K und M sind $\{1, 4, 6\}$, $\{2, 5\}$ und $\{3\}$.

Die obere Annäherung von Patienten mit Grippe ist $I^*(Kl. = j) = \{1, 2, 3, 5, 6\}$, die untere $I_*(Kl. = j) = \{1, 3, 6\}$ (2 ist nicht unterscheidbar von 5); der Rand ist $R_I(Kl. = n) = \{2, 5\}$.

Der Grad der Ungenauigkeit für das Konzept „Grippe“ ist damit $\alpha_I(Kl. = j) = 3/5$. Es gilt z.B. $\mu^I_{Grippe} = \frac{|\{1,2,3,6\}|}{|\{1\}|} = 1$.

□

Aktivierungselement: Berechnen Sie die obere und untere Annäherung, den Rand sowie den Grad der Ungenauigkeit für das Konzept „Kl.=n“ (keine Grippe). Berechnen Sie μ^I_{Grippe} für die Patienten $2, 3, 4, 5$, und 6.

Durch Berechnung der Redukte der ganzen Menge sehen wir, dass man entweder M oder H entfernen kann, ohne die Beschreibung zu verändern. Die (um ein Attribut reduzierte) Tabelle kann in Regeln transformiert werden. Durch einige weitere Schritte kann die Regelmenge u.U. weiter vereinfacht werden und als Regelmenge zur Klassifikation verwendet werden. Die Erstellung eines medizinisches Expertensystems als Anwendung findet man in [Tsu98]. Genexpressionsdaten werden in [MKN+02] ausgewertet.

Das Case-Based Reasoning ist immer dann ein sinnvoller Ansatz im Rahmen des ML, wenn man wenig Gemeinsamkeiten in den Daten vermutet, so dass man auf die orignalen Daten, die Fälle, zurückgreifen muss. Dabei wird man versuchen, auf möglichst ähnliche Fälle zurückzugreifen. Die Rough-Set-Theorie ist ein anderer Ansatz, der versucht, Ungenauigkeiten bei der Datenmodellierung zu berücksichtigen.

Wie wir gesehen haben, wurden bei den verschiedenen Lernverfahren aus den Gebieten NN, FT und ML Parameter von Hand eingestellt und Heuristiken verwendet. Gemeinsam ist vielen Regellernverfahren, die auf vagen, heuristischen Prinzipien beruhen, dass sie durch ES verbessert werden könnten. Deshalb wollen wir ab dem übernächsten Kapitel des Buches die ES behandeln. Vorher wollen wir aber im nächsten Kapitel noch auf Anwendungen des ML in der Bioinformatik eingehen.

Kapitel 25

Maschinelles Lernen in der Bioinformatik

25

25

25 Maschinelles Lernen in der Bioinformatik

In den letzten Kapiteln haben wir bereits einige Beispiele für Anwendungen des ML in der Bioinformatik genannt. Im Folgenden wollen wir uns weiteren Anwendungen widmen, zunächst Anwendungen von Entscheidungsbäumen, dann von Assoziationsregeln. Am Ende des Kapitels werden wir auch wieder auf Anwendungen in der Chemieinformatik eingehen.

Lehrziele:
- Das Beispiel der Anwendung von Entscheidungsbäumen zur Einteilung von Single Nucleotide Polymorphisms verstehen,
- ein Beispiel von Entscheidungsbaum-Lernen zur Kombination von Sekundärstrukturelementen kennenlernen,
- wissen, wie Entscheidungsregeln für Spoligotyping-Daten verwendet werden können,
- Beispiele kennenlernen, in denen Assoziationsregeln verwendet werden, wie bei Genexpressionsdaten von Hefe und bei Protein-Protein-Interaktionen,
- die Idee hinter der Genontologie kennen,
- wissen, wie man ML zum Auffinden relevanter Molekülstrukturen verwenden kann.

25.1 Single Nucleotide Polymorphisms

Ändert man in einer Sequenz ein einzelnes Nukleotid, so spricht man von einem SNP (Abk. für: single nucleotide polymorphism). Ein SNP kann z.B. durch eine Insertion oder Deletion verursacht sein. Man ist an den Effekten dieser Änderung interessiert, um die Auswirkungen auf eine Medikamententherapie zu erforschen. Nicht alle Änderungen wirken sich auf die Genfunktion oder auf das kodierte Protein aus. Interessant ist eine Vorhersage, ob die Genfunktion verändert wird. Neben der Vorhersage sind auch Regeln interessant, die die Entscheidung beschreiben. In [KW03] werden Support Vector Machines zur Klassifikation und Entscheidungsbäume zur Klassifikation und Regelgenerierung für diese Aufgabenstellung verglichen.

Zwei spezielle Datensätze wurden verwendet: *lac repressor*-Daten (mit 3303 Mutationen) und T4 *lysozyme*-Daten (mit 1990 Mutationen). Vier Klassen wurden vorgegeben (kein Effekt, geringer Effekt, starker Effekt, vollständi-

ges Ausbleiben). Als Attribute wurden u.a. verwendet: Residuenidentitäten der originalen und mutierten Residuen, physikochemische Klassen der Residuen (hydrophob, polar, geladen, glycoid), Sequenzerhaltungs-Score an der mutierten Position, Molekularmassenveränderung nach Mutation, Hydrophobizitätsunterschiede, Sekundärstruktur und Wasserlöslichkeit.

Die Klassifikationsleistung wurde mit 10-facher Kreuzvalidierung bewertet, also immer 9 Teile Training und dann ein Teil Test mit insgesamt 10 Versuchen. Die Klassifikationsleistung von SVM ist etwas besser, aber beim Entscheidungsbaumlernen kann man leicht Regeln angeben.

25.2

25.2 Kombination von Sekundärstrukturvorhersagen

In [SML99] sind Sequenzen mit bekannter Struktur (Spirale, Strang, Helix) und deren Vorhersagen vorgegeben, die mit sechs verschiedenen Methoden (DSC, NNSSP, PBLOCK, PHD, PREDATOR, ZPRED. JPRED) erzielt wurden. Eine Regel aus einem geprunten Baum lautet z.B.

WENN NNSSP IST Helix UND PHD IST Spirale UND PREDATOR IST Strang UND ZPRED IST Helix DANN Spirale

mit 18 Samples und 10.9 falsch klassifizierten Samples, geschätzt aus dem ungeprunten Baum. Durch Pruning kann man in vielen Fällen eine kleine Verbesserung der Einzelergebnisse erzielen.

Aktivierungselement: Angenommen, man hat zwei Vorhersageverfahren V und W. Mit den Ergebnissen (berechnete Klassenzugehörigkeit und Konfidenz der Regel mit der höchsten Konfidenz für diese Entscheidung) der beiden Verfahren trainiert man einen Entscheidungsbaum E. Warum muss das Ergebnis nicht höher sein als die Einzelergebnisse, die mit V und W erzielt wurden? Warum ist die Kombination E zweier Experten nicht unbedingt besser in Hinblick auf das Klassifikationsergebnis als die einzelnen Experten V und W?

25.3 Entscheidungsregeln von Spoligotyping-Daten 25.3

Tuberkulose ist eine schwere Infektionskrankheit. Deshalb sollen Tuberkulose-Epidemien verhindert werden. Für eine Analyse der Krankheit kann man den Direct Repeat (DR) Ort von *Mycobacterium tubercolosis* betrachten, wo 36 (fast) identische Basenpaare angeordnet sind. Zwischen diesen Basenpaaren sind Spacer verschiedener kurzer Länge vorhanden. Die Spoligotyping-Technik ist eine PCR-basierte, reverse Cross-Blot Hybridisierungs-Technik, wobei „PCR" die Abk. für „Polymerase Chain Reaction" ist. Sie kann verwendet werden, um die Lage der Spacer zu bestimmen. Spoligotyping-Signaturen wurden in [SMRS02] Regionen zugeteilt, z.B. Lateinamerika und Mittelmeer (Abk.: LAM). Die Zahl bezeichnet die Position des Spacers, „v" vorhanden, „n" nicht vorhanden. Ein Beispiel für eine gefundene Regel mit 100% korrekten Klassifikationen von 4.9% der LAM ist:

WENN „34" IST n UND „36" IST n UND „31" IST v UND „22" IST n UND „42" IST v UND „9" IST n UND „8" IST n DANN Klasse LAM.

Wir kommen nun zu zwei Anwendungen der Assoziationsregeln, in denen Genexpressionsdaten von Hefe und Protein-Protein-Interaktionen ausgewertet werden.

25.4 Regelerzeugung von Genexpressionsdaten von Hefe 25.4

In [CH03] werden 300 Genexpressions-Datensätze von Hefe analysiert. Als ein Item wird ein Transkript oder ein Protein angesehen, dass in einer relativen Fülle vorhanden ist. Die Zahlenwerte werden diskretisiert, z.B. in „highly expressed" oder „highly repressed" sowie „weder noch". Betrachtet wurden 6316 Transkripte, die zu 300 Mutationen oder chemischen Behandlungen von Hefe gehören. Eine Regel lautet z.B. (alle Gene „highly expressed"):

$$\{\text{YHM1}\} \Rightarrow \{\text{ARG1,ARG4,ARO3,CTF13,HIS5,LYS1,RIB5, ...} \quad (117)$$

$$\text{..., SNO1,SNZ1,YHR029C,YOL118C}\} \quad (118)$$

mit 11% Häufigkeit und 81% Konfidenz. Dabei bedeutet z.B. YHM1: „High copy suppressor of ABF2ts is defect, putative mitochondrial carrier protein"; ARG1: „Arginosuccinate synthetase (key enzyme in arginine biosynthesis)".

25.5 Protein-Protein-Interaktionen

In [OKSI02] betrachtet man die Hefe *Saccharomyces cerevisae*. Verwendet wurden 1841 Protein-Paare aus YPD (Yeast Proteome Database), 1064 Proteinpaare aus MIPS (Munich Information Center for Protein Sequences), 1482 Proteinpaare aus einer Datenbank von Ito et al. [ICO+01] und 957 Proteinpaare aus Uetz et al. [UGC+00].

Attributeigenschaften eines Proteins können aus Genomdatenbanken entnommen werden, hier 275 aus YPD, wie z.B. die biochemische Funktion, 154 aus einer EC-Nummerierung (enzyme classification), 491 aus SWISS-PROT, z.B. strukturelle Eigenschaften, 698 aus PROSITE, 20 für die Bias-Darstellung von Aminosäuren (wird gesetzt bei gehäuften lokalem Auftreten), 3589 Segment-Cluster, 14 aus Aminosäurenmuster. Insgesamt hat man 4307 interagierende Proteinpaare (linkes, rechtes Protein) und 5241 Attribute zusammengetragen. Eine Regel, die man erhalten hat, lautet:

WENN linkes Protein: Localization IST nuclear nucleolus
UND Keyword IST nuclease
DANN rechtes Protein: Localization IST nuclear nucleolus

mit 14 Proteinpaaren und 100% Konfidenz. Da man aber Tausende Regeln hat, muss man die Regeln nach ihrer Relevanz ordnen, z.B. kann man Regeln mit mehr positiven Interaktionen höher bewerten. Für weitere Bewertungskriterien siehe [OKSI02].

25.6 Die Gen-Ontologie

Eine Aufgabe der Bioinformatik ist das systematische Zusammenstellen, Speichern und Bereitstellen von Daten. Die Darstellung der Daten kann systematisch unter Berücksichtigung ihrer Abhängigkeiten in Ontologien geschehen. Eine solche Ontologie ist „the Gene Ontology“ (dt.: Gen-Ontologie) (Abk.: GO), www.geneontology.org/, in der Gene mit ihren Genprodukten abgespeichert sind. Drei Ontologien existieren momentan, eine zur Darstellung der molekularen Funktionen, eine zur Darstellung der biologischen Prozesse und eine zur Darstellung der zellulären Komponenten.

Die Beschreibung der biologischen Prozesse der GO wird in [HLK03] verwendet, um aus Zeitreihen von Microarray-Daten Hypothesen für neue Gene mit bislang unbekanntem Prozess zu generieren. Um zu verstehen, wie diese Zuweisung durchgeführt werden kann, muss man wissen, dass die GO als

direkter, azyklischer Graph abgespeichert ist. Die Knoten repräsentieren die Proteinfunktionsbeschreibungen. Eine Kante zwischen zwei Knoten v_j und v_i existiert, wenn die Beschreibungen von v_j eine Teilmenge von Beschreibungen von v_i sind. Jedes Gen wird mit einer Menge von GO-Termen annotiert. Als Entscheidungstechnik zur Einordnung unbekannter Gene wird die Rough-Set-Theorie aus dem Abschnitt 24.2 verwendet. Es werden heuristische Approximationen an die Redukte berechnet, die als Klassifikationsregeln verwendet werden. Anhand eines Experiments zum Zellzyklus in menschlichen Fibroplasten konnte gezeigt werden, dass mit diesem überwachten Ansatz sinnvolle Hypothesen generiert werden können.

25.7 Unterscheidung zwischen Drugs und Non-Drugs

25.7

Die Trainingsdaten für die Arbeit [WG00] stammen aus dem WDI (World Drug Index) und dem ACD (Available Chemical Directory). Nach einer allgemeinen Vorverarbeitung bleiben 38416 Moleküle und 169331 Moleküle aus ACD übrig. Die Daten wurden wie folgt aufgeteilt: 10000 Moleküle zum Trainieren, 20000 zum Testen (Modellvergleich) und der Rest zum Validieren (Beurteilung des besten Modells). Der Deskriptorvektor von Ghose und Crippen wird verwendet, der die Anzahl von 120 Atomtypen zählt und ein 121-ter Deskriptor für die restlichen Typen. In der folgenden Regel bedeutet „C:“, dass ein C-Atom in der nach dem Doppelpunkt folgenden Verbindung enthalten ist. Ein „−−“ steht für eine aromatische Verbindung. R repräsentiert eine beliebige funktionale Gruppe.

WENN C: $=CR_2 > 0$ UND C: R−−CH−−R ≤ 2 DANN Non-Drug

Die Regel ist gültig für 3497 Moleküle, wobei davon fälschlicherweise 597 als Drug eingestuft werden.

25.8 Auffinden relevanter Molekülstrukturen

25.8

In [BB02], [HBB03] werden Moleküle als Graph modelliert und zwar die Bindungstypen als Kanten und die Atomtypen/Ringe als Ecken. Beginnend mit einem Atom als Wurzel können immer größere Fragmente auf den Ebenen eines Baumes eingetragen werden. Dieser kann z.B. mit A-priori durchsucht werden. Man erhält die häufigsten Substrukturen. Man kann Zähler für verschiedene Klassen einführen. HIV-Daten wurden nach ihrer Aktivität klassifiziert. Auch wenn durch Einführung eines Attributs pro Ring der Suchraum deutlich verkleinert werden kann, erzeugt ein Baumverfahren viele Regeln.

Offen bei diesem Ansatz bleibt die Frage, wie man mit strukturell ähnlichen Molekülen umgeht, deren Aktivität sich aber stark unterscheidet. Weiß man nicht, welches Fragment man als Baumwurzel nehmen sollte, dann kann man mit verschiedenen Fragmenten aus dem Molekül einen Teilbaum aufbauen.

Im Maschinellen Lernen werden die verschiedenartigsten individuellen Probleme aus dem Bereich der Bioinformatik verarbeitet wie die Einteilung von Single Nucleotide Polymorphisms, die Regelgenerierung für Protein-Protein-Interaktionen oder das Auffinden relevanter Molekülstrukturen. Wir haben insbesondere die Anwendung von Entscheidungsbäumen und von Assoziationsregeln hervorgehoben. Algorithmen, die auf den klassischen Datenstrukturen Graph und Baum basieren, spielen dabei eine große Rolle.

Kapitel 26

26 Überblick Optimierung, Genetik, Evolution

26

26 Überblick Optimierung, Genetik, Evolution

Bevor wir *Evolutionäre Strategien* (kurz: ES) zur optimierten Problemlösung einsetzen, möchten wir an das Kap. 2 anknüpfen und auf die biologischen Grundlagen dieser Algorithmen eingehen. In den Folgekapiteln führen wir ausführlich die ES ein und erläutern insbesondere auch deren Einsatz für Anwendungen der Bioinformatik und in regelbasierten Systemen.

Lehrziele:
- Einen kurzen Überblick bekommen, welche Probleme und biologischen Fakten zur Idee der Evolutionären Strategie geführt haben,
- wiederholen, welchen Aufbau die biologische DNA besitzt,
- die biologischen Begriffe des Überkeuzens, der Mutation, der Population, des Genotyps und des Phänotyps verstehen.

26.1 Einführung

Um zu verstehen, wie die Idee der ES geboren wurde, stellen Sie sich vor, Sie hätten eine Aufgabe zu lösen (z.B. im Rahmen Ihrer Diplomarbeit). Die Aufgabe bestünde darin, eine optimale Form für Flugzeugturbinen zu finden in Bezug auf das Durchströmungsverhalten der Luft. Sie sollen also die Leistung der Turbine optimieren. Was würden Sie tun? Sie würden zuerst ein Modell der Form der Turbine aufstellen, das durch Parameter gegeben wäre (z.B. bestimmte Längen, Krümmungen). Dann würden Sie ein bekanntes Optimierungsverfahren wählen, das allgemein verwendet wird, also z.B. einen Simplex-Algorithmus. Leider werden Sie feststellen, dass die Lösung nicht befriedigend ist. Was nun? – Sie haben eine gute bis geniale Idee! Schwefel hat in seiner Diplomarbeit 1965 gerade solch ein Problem durch Optimierung mittels einer eigens konstruierten ES gelöst.

Wir stellen im nächsten Abschnitt die grundlegenden Sachverhalten aus der Biologie zusammen, die für diese Art der Problemlösung eine Rolle spielen, vgl. auch [SHF94] und [DS02]. Unser primäres Ziel ist, genau so viel des biologischen Vorbilds für einen Algorithmus zu übernehmen wie es notwendig und sinnvoll erscheint. Ein Algorithmus, der keinem biologischen Vorbild folgt und dennoch sehr effizient ist, ist einem weniger effizienten biologisch motivierten Algorithmus vorzuziehen. Wir werden aber sehen, dass das biologische Vorbild sehr wohl interessante und effiziente Vorgehensweisen motiviert.

26.2

26.2 Grundlagen

Darwins Werk (1859) „On the Origin of Species by Means of Natural Selection" erscheint. Beschrieben wird die Evolution des Menschen aus primitiven Arten durch natürliche Selektion. Das Prinzip „survival of the fittest" besagt, dass nur die fittesten Individuen überleben. Die Fitness eines Individuums kann von vielen einzelnen Faktoren abhängen wie z.B. Stärke im Kampf oder Anpassungsfähigkeit. Mendel entdeckt 1865 („Untersuchungen über Pflanzenhybride") die Vererbungsgesetze, z.B. das Rekombinationsgesetz.

Man kann eine *Zelle* als den Grundbaustein des Lebens ansehen, wobei die Zelle selbst aus chemischen Bestandteilen aufgebaut ist. Fast alle Zellen enthalten einen *Zellkern* (nucleus) (Bakterien sind eine Ausnahme). Der Zellkern enthält die Träger der Erbsubstanz, die sog. *Chromosomen*. Die Erbsubstanz sind die *Gene*. Die Chromosomen bestehen aus *Nukleinsäuren* und *Proteinen*. Diese Struktur wurde von Watson, Crick 1953 entdeckt. Die wichtigste Nukleinsäure ist die *Desoxyribonukleinsäure* (DNS, engl.: **DNA**, A=acid, dt.: Säure), die als (verdrehter) Doppelstrang auftritt. Die Bausteine heißen *Nukleotide*. Die Nukleotide kodieren vier verschiedene Basen: Adenin (A), Guanin (G), Thymin (T) und Cytosin (C); A und T sowie C und G sind komplementär, d.h. liegen sich im Doppelstrang gegenüber. Je drei Nukleotide (*Basentriplett*) kodieren die *Aminosäuren*. Eine Folge von Basentripletts steht für eine Aminosäuresequenz (*Primärstruktur* von Proteinen). Mehrere räumlich angeordnete *Sekundärstrukturen* (das sind gefaltete Primärstrukturen) bilden Proteinmoleküle (*Tertiärstruktur*), die z.B. den Stoffwechsel durch Hormone steuern oder als Antikörper an der Immunabwehr beteiligt sind.

Wir gehen nicht auf die in mehreren Phasen ablaufende Zellteilung ein, stellen aber fest, dass durch *Überkreuzen* (crossing-over) Erbgut aus zwei DNA-Strängen neu verteilt wird. Ein *Allel* ist ein Gen auf einem DNA-Strang. Die Proteinsynthese findet in den *Ribosomen* (durch Transkription) statt. Die *mRNA* (messenger Ribonukleinsäure) liest die Informationen eines offenen DNA-Strangs, die dann von der *tRNA* (transport RNA) in die Ribosomen übertragen wird. Die RNA verwendet als Baustein statt Cytosin Uracil (U). Der *Promoter* ist eine Startsequenz auf einem Gen (Endsequenz = *Terminator*). *Mutationen* sind spontan auftretende, strukturelle Veränderungen der Chromosomen, z.B. durch Röntgenstrahlen oder UV-Licht. Man erwartet beim Menschen etwa eine Mutation auf 10^4 bis 10^5 Genen.

Die biologischen Fakten betreffen die Evolution *eines* Menschen. Aber man kann auch nach der Evolution einer *Population* von Menschen fragen. Diese wird gesteuert durch die Vermehrung und kann sich z.B. bei Nahrungsmittelknappheit anders entwickeln als bei einer ausreichenden Versorgung. Dabei betrachtet man den *Gen-Pool* einer *Generation* und analysiert die Veränderungen im Gen-Pool der folgenden Generationen. Als *Genotyp* bezeichnet man die genetische Ausstattung, als *Phänotyp* dessen Ausprägung (z.B. Augenfarbe).

Damit wissen wir erstmal genug, um die ES einführen zu können. Wir wollen noch einmal festhalten, dass die Simplex-Methode der Optimierung eine *lineare* Methode ist und das Verfahren wie Backpropagation zwar nichtlineare Funktionen approximieren kann, aber bei sehr komplexen, evtl. sogar nichtdifferenzierbaren Fehlerflächen wegen vieler lokaler Minima nur schwer oder gar nicht sinnvoll arbeitet. Wir gehen davon aus, dass unser Problem komplex ist und wir eine Optimierung mit ES durchführen wollen. Das können sowohl diskrete, kombinatorische als auch kontinuierliche Problemstellungen sein. Es wird sich herausstellen, dass man die eingeführten biologischen Begriffe analog im Rahmen der ES verwenden kann.

Kapitel 27

Evolutionäre Strategien

27

27

27 Evolutionäre Strategien

Wir geben zuerst Literaturhinweise zu ES. Im weiteren Verlauf lernen wir nach und nach die Algorithmen kennen, die zur ES führen.

Lehrziele:
- Kennenlernen der wichtigsten Literaturquellen zu Evolutionären Strategien,
- verstehen, wie zufällige Zustandsveränderungen im Metropolis-Algorithmus verwendet werden,
- die Idee angeben können, die zum Simulated Annealing führt,
- die Verwendung der Begriffe Population, Fitnessabbildung, Mutation, Rekombination und Selektion beherrschen,
- den grundlegenden Ablauf eines evolutionären Algorithmus angeben können,
- wissen, wie man einfache Mutationen, Rekombinationen und Selektionen berechnen kann,
- die verwendeten Evolutionsschemata notieren können,
- einige typische Anwendungen aufzählen können.

27.1 Literaturübersicht

Wir geben wieder Konferenzen, Zeitschriften und Lehrbücher an.

Konferenzen:

- IEEE International Conference on Evolutionary Computation (CEC),
- Genetic and Evolutionary Computation Conference (GECCO),
- International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA),
- Asia-Pacific Conference on Simulated Evolution and Learning (SEAL).

Zeitschriften:

- Evolutionary Computation,
- IEEE Transactions on Evolutionary Computation,
- Journal of Heuristics.

Lehrbücher und -kapitel:

- Goldberg [Gol89],
- Weicker [Wei02],
- Holland [Hol92],
- Tettamanzini/Tomassini [TT01],
- Pohlheim [Poh00],
- Schöneburg [SHF94],
- M. Mitchell [Mit96],
- Jacob [BH99, Kap. 9].

27.2

27.2 Metropolis-Algorithmus und Simulated Annealing

Um deterministische Algorithmen flexibler zu machen, z.B. um lokale Minima zu überwinden, ist eine Idee, sich den *Zufall* zu Nutze zu machen. Im **Metropolis-Algorithmus** betrachtet man das zu optimierende System als ein Elementarteilchensystem, das zu einem Zeitpunkt t und einer bestimmten Temperatur T einen Zustand z (engl.: state) annimmt. Boltzmann erkannte, dass die Wahrscheinlichkeit, mit der ein bestimmter Zustand z bei einer bestimmten Temperatur T angenommen wird, von der Energie $E(z)$ abhängt. Die *Boltzmann-Verteilung* ist durch die Dichtefunktion $p_T(z) = c \cdot \exp(\frac{-E(z)}{kT})$, c zusätzlicher (hier vernachlässigter) Faktor, k Boltzmann-Konstante, gegeben. Bei der Optimierung würde man davon ausgehen, dass das System nur Zustände mit niedrigerer Energie annehmen kann, um ein Gleichgewicht zu erreichen. Die Idee des Metropolis-Algorithmus ist es auch einige wenige (durch vom Zufall abhängige Störungen) Zustände mit höherer Energie zuzulassen. Der Metropolis-Algorithmus ist so eigentlich kein biologisch motivierter Algorithmus, sondern ein physikalisch motivierter.

27.1

Algorithmus 27.1 (Metropolis-Algorithmus)

1. Solange noch nicht eine maximale Anzahl an Durchläufen bearbeitet wurde, gehe zu Schritt 2, sonst beende den Algorithmus.

2. Berechne einen neuen Systemzustand z_{neu} durch zufällige Veränderung des alten Zustands z. (Mutationsschritt)

3. Berechne die Differenz $\Delta E := E(z_{\text{neu}}) - E(z)$. (Evaluationsschritt)

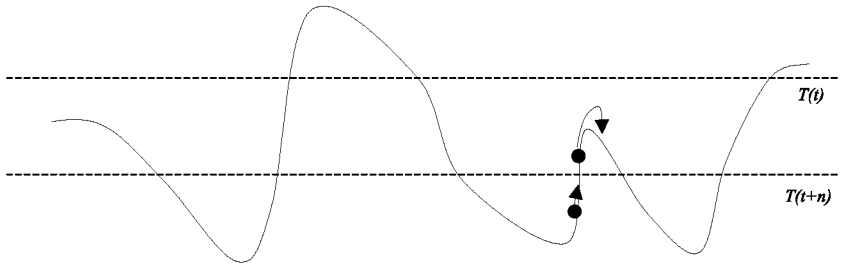

Abbildung 27.1. Simuliertes Abkühlen zur Minimumsuche mit zufälligem Aufstieg

4. Gilt $\Delta E < 0$ (neuer Zustand mit niedrigerer Energie wird übernommen) oder gleichverteilter Zufallswert aus $[0, 1] < \exp(\frac{-\Delta E}{T})$ (Ablehnung von z_{neu} mit der Wahrscheinlichkeit $1 - \exp(\frac{\Delta E}{kT})$), dann übernehme den neuen Systemzustand $z := z_{\text{neu}}$ und fahre mit Schritt 1 fort. (Selektionsschritt)

□

Das System kühlt nach mehreren Metropolis-Schritten immer weiter ab und gelangt so in einen Gleichgewichtszustand. Allerdings wird das System vom Zufall am Anfang der Abkühlung gleich stark gestört wie am Ende der Abkühlung. Um das System wirklich abkühlen zu lassen, lässt das **Simulated Annealing** am Ende des Abkühlens nicht mehr so große Störungen zu, indem die Temperatur herabgesetzt wird und damit die Wahrscheinlichkeit für Störungen gesenkt wird. Der Metropolis-Algorithmus wird einfach erweitert zum Algorithmus 27.2.

Algorithmus 27.2 (Simulated Annealing, vgl. Abb. 27.1) **27.2**

1. Initialisierung.

2. Solange die Temperatur T noch nicht die Endtemperatur erreicht hat, führe den Metropolis-Algorithmus mit aktuellem T durch.

3. Abkühlungsschritt: Setze $T := \alpha \cdot T$, $\alpha \in]0, 1[$ und gehe zu Schritt 2.

□

27.3

27.3 Evolutionäre Strategien und Anwendungen

Nun sind wir soweit, dass wir die Grundbegriffe und das algorithmische Schema der ES einführen können. Wir führen zuerst die notwendigen Begriffe ein.

27.3

Definition 27.3

a) Sei S der Grund- oder Suchraum, in dem eine optimale Lösung gesucht wird.

b) Die Struktur eines Individuums sei durch sein **Genom** g_i gegeben. g_i kann als reeller Merkmalsvektor realisiert werden.

c) Eine **Population** zum Zeitpunkt t sei durch die Ansammlung $P(t) = \{g_1(t), \ldots, g_r(t)\}$ gegeben.

d) Zur Bewertung eines Individuums g_i sei $f : S \to \mathbb{R}$ eine **Fitness-Abbildung**, die g_i die Fitness $f(g_i)$ zuweist. Die Auswertung der Fitness bezeichnet man als **Evaluation**.

e) Als **Mutation** bezeichnen wir eine Abbildung $f_{\text{mut}} : P^s \to P^s$, die eine Teilpopulation von s Individuen (zufällig) modifiziert.

f) Unter einer **Rekombination** verstehen wir eine Abbildung $f_{\text{rek}} : P^s \to P^u$, die s Individuen zu u Individuen kombiniert. Die Individuen für die Rekombinationen werden zufällig ausgewählt.

g) Die **Selektion** $f_{\text{sel}} : P^{s_1} \to P^{s_2}$ wählt aus den (variierten) Individuen einen Teil aus. Dabei können bei den variierten Kinderindividuen auch nicht variierte Elternindividuen mit übernommen werden.

□

27.4

Algorithmus 27.4 (Evolutionärer Algorithmus)

1. Initialisiere eine Population $P(0)$;

2. Stoppkriterium erfüllt (hohe Fitness erreicht oder maximale Anzahl von Generationen erreicht), dann breche ab und gebe $P(t)$ zurück, ansonsten fahre mit Schritt 3 fort (inkrementiere t);

3. Rekombination: $P(t) := f_{\text{rek}}(P(t))$;

4. Mutation: $P(t) := f_{\text{mut}}(P(t))$;

5. Fitness-Berechnung: Bilde $f(P(t))$;

6. Selektion: $P(t+1) := f_{\text{sel}}(P(t))$;

□

Die Parameter selbst, z.B. **Mutationsrate**, **Selektionsrate**, **Rekombinationsrate** können selbst mit variiert und selektiert werden, um z.B. eine falsche Wahl von Steuerparametern vorzubeugen. Die o.g. Raten legen fest, wie viele Mutationen, Selektionen und Rekombinationen auftreten dürfen.

Erinnern wir uns daran, dass die Dichtefunktion einer Normalverteilung mit Mittelwert μ und Varianz σ gegeben ist durch

$$\mathrm{N}(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp(-\frac{(x-\mu)^2}{2\sigma}) \ . \tag{119}$$

Wir können dann eine *Mutation* eines Individuums (oder des vektorwertigen Genotyps) beschreiben als zufällige, normalverteilte Abweichung des Originalindividuums, also als

$$f_{\text{mut}}(g_i) := g_i + \mathrm{N}(0, \sigma) \ . \tag{120}$$

Andere Mutationen, insbesondere für Strategieparameter, sind denkbar, z.B. könnte man die Variationsraten für die Strategieparameter abkühlen gemäß des Simulated Annealing.

Die Aufgabe einer Mutation ist es, Genotypen zu erzeugen, die im gesamten Suchraum liegen können (Ausschöpfung des Suchraums).

Eine gebräuchliche Möglichkeit der (kontinuierlichen) *Rekombination* ist die Mittelwertbildung

$$f_{\text{rek}}(x_1, \ldots, x_m) := \frac{1}{m} \sum_{i=1}^{m} x_i \ . \tag{121}$$

Eine andere Möglichkeit ist die zufällige Zuordnung (diskrete Auswahl) einzelner Objekte (für die Wahrscheinlichkeiten gilt $\sum_{i=1}^{m} p_i = 1$):

$$f_{\text{rek}}(x_1, \ldots, x_m) := \begin{cases} x_1 & \text{mit } p(x_1) \\ \ldots & \ldots \\ x_m & \text{mit } p(x_m) \end{cases} \tag{122}$$

Die Aufgabe der Rekombinationen ist es, eine schnellere Konvergenz der ES zu erreichen. Man kann auf einer Subpopulation Rekombinationen anwenden (lokale Rekombination) oder aber auf der ganzen Population (globale Rekombination).

Aktivierungselement: Implementieren Sie Mutations- und Rekombinationsoperatoren in einer gängigen Programmiersprache.

Wir müssen nun noch Mutationen, Rekombinationen und Selektionen in **Evolutionsschemata** zusammenführen und tun dies in der Notation von Rechenberg [Rec73], [Rec94]:

a) $(1+\lambda)$-Strategie: λ mutierte Kindergenotypen werden *zusammen* (deshalb „+“) mit dem einzigen (deshalb „1“) Elterngenotypen in einen Selektionstopf übernommen. Nun wird ein Individuum für die nächste Generation ausgewählt, z.B. zufällig oder die fittesten („survival of the fittest“). Der Elterngenotyp kann nur durch einen fitteren Kindergenotyp ersetzt werden.

b) $(1, \lambda)$-Strategie: analog zur $(1+\lambda)$-Strategie, aber es werden *ausschließlich* die Kindergenotypen in den Selektionspool übernommen, ohne den Elterngenotypen (deshalb: ,).

c) Allgemeiner: $(\mu/\rho + \lambda)$ bzw. $(\mu/\rho, \lambda)$-Strategie: μ Elterngenotypen erzeugen λ Kindergenotypen durch Mutation und anschließender Rekombination von je ρ Elterngenotypen für den Selektionspool. Die Elterngenotypen werden je nach „+“ oder „ , “ in den Selektionspool übernommen. Überleben können die μ fittesten Individuen im Selektionspool.

Weitere Festlegungen können Strategieparameter betreffen. Man kann auch (z.B. auf Parallelrechnern) k Populationen betrachten, die verschiedene Operationen und Parameter verwenden. Man kann auch bestimmte Festlegungen

für eine bestimmte Anzahl von Generationen treffen. Wir listen einige Anwendungen von ES (Evolutionäre und genetische Algorithmen), teilweise mit Literaturhinweisen, auf:

- Funktionsoptimierung,
- Prozessoptimierung (z.B. Klimaregelung),
- Stundenplanoptimierung,
- Warteschlangenoptimierung,
- Optimierung eines medizinischen Scores [Pae03a],
- optimierte Teststrategien im Software Engineering [BPS03],
- Optimierung von Telekommunikationsnetzen,
- Travelling Salesperson Problem,
- Optimieren von sich dynamisch verändernden Parameterumgebungen,
- Genetisches Programmieren, vgl. Kap. 29,
- Optimieren der Struktur von NN,
- Einsatz einer SOM zum Ordnen der Neuronen der versteckten Schicht eines FF-NN mit anschließender 2D-Rekombination der Gewichtsmatrix [KM03].
- Sequenzalignment (Bioinformatik), vgl. auch Kap. 31,
- Optimierung von Regeln, vgl. Kap. 32,
- Verwendung einer DNA-basierten Codon-Kodierung innerhalb eines genetischen Algorithmus [GE03b].

ES werden aber in allen Bereichen eingesetzt: Betriebswirtschaft, Versicherungswesen, Technik, Medizin, Bioinformatik, Chemie, Life Science, usw. Das liegt einfach daran, dass die Optimierung ein sehr universelles Werkzeug ist und überall möglichst optimale Lösungen sehr gefragt sind.

Wir werden im Kap. 31 auf Anwendungen der ES in der Bioinformatik eingehen, nachdem wir in diesem Kapitel die Entwicklung vom Metropolis-Algorithmus über das Simulated Annealing hin zu den ES kennengelernt haben. Die Verwendung der bedeutenden Prinzipien der Mutation, Rekombination und Selektion müssen Sie beherrschen, um die folgenden Kapitel gut zu verstehen.

Kapitel 28

Genetische Algorithmen

28

28

28 Genetische Algorithmen

Ursprünglich gab es zwei Schulen, die Schule der *evolutionären Algorithmen*, begründet durch Schwefel und Rechenberg und die Schule der *genetischen Algorithmen* (Abk.: GA), die von Holland eingeführt wurden. Der Unterschied liegt darin, dass bei genetischen Algorithmen *diskrete* Kodierungen eines Problems verwendet werden und Operatoren verwendet werden, die stärker *genetisch* motiviert sind, d.h. stärker durch die Genetik beeinflusst sind. Allerdings unterscheiden sich beide Ansätze im Grundsatz nicht, so dass die o.g. Einführung in evolutionäre Algorithmen auch als Einführung in genetische Algorithmen angesehen werden kann, z.B. können die Evolutionsschemata übernommen werden; deshalb sprachen wir auch bislang allgemein von ES. Z.B. kann die in Kap. 27 eingeführte Mutation im Rahmen der GA als Punktmutation eines (haploiden) Genstrangs angesehen werden. Die Rekombination wird als Operation für diploide Genstränge angesehen.

Lehrziele:
- Kennenlernen einiger spezieller Operationen für genetische Algorithmen wie das Crossover,
- Wichtige Selektionskriterien nennen können,
- die Idee der Classifier-Systeme angeben können und dessen Bestandteile angeben können.

28.1 Mutation, Rekombination und Selektion

An dieser Stelle sollen die spezielleren Operationen eingeführt werden, die vor allem in GA verwendet werden. Anschließend gehen wir auf verschiedene Arten der Selektion ein.

Der Rekombinationsoperator **Crossover** (dt.: Überkreuzung) verbindet Sequenzen zweier oder mehrerer Genotypen. Im Spezialfall zweier Elterngenotypen und dem Austausch zweier Sequenzen spricht man von einem **Ein-Punkt-Crossover**, vgl. die Abb. 28.1.

Die **Inversion** (oder Umkehrung) ist definiert als:

$$g_{\text{inv}}(g_1, \ldots, g_{i_1}, \ldots, g_{i_2}, \ldots, g_n) := (g_1, \ldots, g_{i_2}, g_{i_2-1}, \ldots, g_{i_1+1}, g_{i_1}, \ldots, g_n) \tag{123}$$

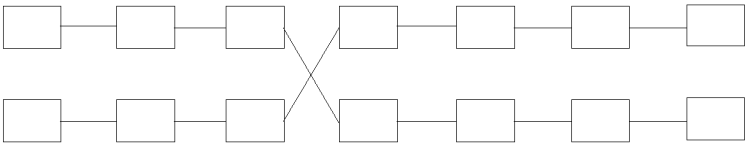

Abbildung 28.1. Ein-Punkt-Crossover

Dieser Operator kann bei diskreten Optimierungsproblemen, z.B. TSP, eingesetzt werden, z.B. auch allgemeiner mit einer Permutation π:

$$g_{\text{inv}}(g_1, \ldots, g_{i_1}, \ldots, g_{i_2}, \ldots, g_n) := (g_1, \ldots, \pi(g_{i_1}, \ldots, g_{i_2}), \ldots, g_n) \quad (124)$$

Die **Löschung** (engl.: deletion) wird definiert als:

$$g_{\text{del}}(g_1, \ldots, g_{i_1}, \ldots, g_{i_2}, \ldots, g_n) := (g_1, \ldots, g_{i_1-1}, g_{i_2+1}, \ldots, g_n) \quad (125)$$

Wir führen die **Verdopplung** (engl.: duplication) ein als:

$$g_{\text{dup}}(g_1, \ldots, g_{i_1}, \ldots, g_{i_2}, \ldots, g_n) := (g_1, \ldots, g_{i_1}, \ldots, g_{i_2}, g_{i_1}, \ldots, g_{i_2}, \ldots, g_n) \quad (126)$$

Die Verdopplung kann als Instrument der Mutationsratensteuerung eingesetzt werden. Die Löschung und die Verdopplung können bei Genen eingesetzt werden, bei denen Werte fehlen oder die eine variable Länge haben. Dies kann z.B. bei Alignierungsproblemen unterschiedlich langer, evtl. fehlerbehafteter Sequenzen sinnvoll sein.

Am Ende dieses Abschnitts werden noch Selektionskriterien für GA eingeführt, da nicht unbedingt nur die fittesten Individuen überleben sollen. Bei der **Fitness-proportionalen Selektion** (auch: Roulette-Wheel Selection) ist die Wahrscheinlichkeit des Überlebens eines Individuums proportional zu seiner Fitness. Problematisch kann hierbei sein, dass einige Individuen mit sehr hoher Fitness kaum noch das Überleben anderer Individuen zulassen. Man kann anstelle des Absolutbetrags der Fitness die Reihenfolge der Fitness betrachten (**Rang-basierte Selektion**). Man kann dann festlegen, dass das Individuum auf Platz Eins die höchste Wahrscheinlichkeit des Überlebens hat und das Individuum auf dem letzten Platz die geringste. Überleben wie bei den evolutionären Algorithmen die fittesten (ohne Wahrscheinlichkeitszuordnung), dann haben wir wieder die survival-of-the-fittest-Strategie, auch **elitäre Selektion** gennant. Um eine höhere Diversität der Individuen zu erreichen, können auch einige Individuen zufällig überleben, die eine geringe Fitness haben. Bei der **Nachbarschafts-Selektion** werden nur Individuen in der Nachbarschaft, die durch eine vorgegebene Topologie festgelegt ist, be-

rücksichtigt. Bei der **Wettbewerbs-Selektion** (engl.: Tournament Selection) wird immer das jeweils fitteste Individuen aus einer Anzahl von mehreren Individuen selektiert.

28.2 Classifier-Systeme

28.2

Als Anwendung der genetischen Algorithmen erwähnen wir die *Classifier Systeme*. Die Classifier Systeme wurden in den 70er-Jahren entwickelt, siehe z.B. [Hol92]. Wir werden uns im Folgenden an die Darstellung in [Gol89, Kap. 6] anlehnen. Wir gehen davon aus, dass wir Regeln der Art

„WENN Bedingung DANN Aktion"; kurz: „Bedingung/Aktion"

suchen und Datensätze mit binären Attributausprägungen vorliegen haben, also z.B. bei fünf Attributen $[0,1,1,0,1]/[1,1,0,1,1]$. Die Bedingungen in den Regeln sollen möglichst gut generalisiert werden. Die binär kodierte Aktion wird ausgeführt, wenn die Bedingung der Regel zutrifft. Wir wollen an dieser Stelle zwei Fälle unterscheiden:

1. Die ausgeführte Aktion kann wieder als Bedingung aufgefasst werden, z.B. in Control-Anwendungen. Im o.g. Beispiel hieße dies, dass die Aktion $[1,1,0,1,1]$ (z.B. Bewegung nach rechts hinten) wieder als Bedingung aufgefasst werden kann (also: WENN Bewegung nach rechts hinten DANN ...).
2. Die Aktion kann nicht als Bedingung aufgefasst werden, z.B. bei einem reinen Klassifikationsproblem. Wir schreiben in diesem Fall die Klassenzugehörigkeit als Aktion: $[0,1,1,0,1]$/Klasse.

Die von uns gewählten Schreibweisen lassen sofort unser Anliegen erkennen: Wir packen alle Datensätze (in serieller Abfolge) in eine Ansammlung und betrachten diese Ansammlung als unsere Regelmenge, die wir noch durch Einführen von Platzhaltern „$*$" generalisieren wollen. Um neue Regeln in das System einzuführen (bzw. um alte zu löschen), verwenden wir einen GA. Diese Vorgehensweise ist sehr direkt. Was uns im Wesentlichen noch fehlt, ist die Fitness-Bewertung. Die Berechnung der Fitness beschreibt sozusagen die Ausführungsphase des Systems. Die Fitness, die in einem Classifier-System *Stärke* (engl.: strength) genannt wird, wird im Anschluss an ihre Berechnung zu Lernzwecken verändert.

Der Lernalgorithmus des Classifier-Systems nennt sich **Credit Assignment-Algorithmus** (dt.: Wertzuweisung), siehe [Hol92] für Details. Wie bereits erwähnt, können GA-Schritte kanonisch eingeführt werden, die die Regelmenge evolvieren. Dies kann nach der Abarbeitung der Schritte des Credit Assignment-Algorithmus geschehen. Evolviert werden dann die Regeln als Individuuen. Individuen können rekombiniert werden; Nullen, Einsen und Sterne können mutiert werden.

Mittlerweile gibt es viele Varianten, z.B. die Extended Classifier Systems (XCS) [Wil95], die versuchen vollständige Abdeckungen bei geringer Überschneidung und minimaler Regelanzahl zu finden.

Die GA sind eine Variante der ES, insofern sie stärker an die Genetik angelehnte diskrete Operatoren verwenden. Insbesondere kombiniert das Crossover zwei Individuen, indem jeweils ein Teil des einen Individuums mit dem des anderen verbunden wird. Sowohl für ES als auch für GA stehen verschiedene Selektionskriterien zur Auswahl. Die Rang-basierte Selektion spielt dabei die größte Rolle. Regeln können beispielsweise auf binären Attributen basieren, so dass diese mittels GA optimiert werden können. In einem Classifier-System geschieht dies sukzessive unter Verwendung von Platzhaltern, die sowohl Nullen als auch Einsen gemeinsam repräsentieren können. Solch ein Classifier-System kann sich auch dynamisch an sich verändernde Daten anpassen.

Kapitel 29

Genetisches Programmieren

29

29

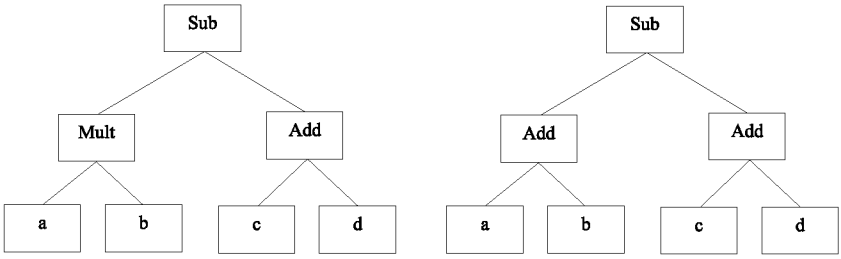

Abbildung 29.1. Ein Syntaxbaum und ein mutierter Syntaxbaum

29 Genetisches Programmieren

Wir schildern die Idee des Genetischen Programmierens und geben einige einführende Anwendungsbeispiele.

Lehrziele:
- Die Idee des Genetischen Programmierens verstehen,
- wissen, dass beim Genetischen Programmieren symbolische Datenstrukturen wie Syntax-Bäume oder Terme verwendet werden.

29.1 Idee des Genetischen Programmierens

Unter **Genetischem Programmieren** versteht man die Anwendung der GA auf Datenstrukturen wie sie in Programmiersprachen vorkommen, also z.B. auf Baumstrukturen [Koz92], [Koz94], [KBIAK99], [KKS+03]. Ziel ist, dass sich Programme selbst optimieren oder dass der Programmcode gar von selbst entsteht. Wie bei allen Anwendungen von GA gibt es zwei Hauptprobleme, die zu lösen sind:

a) die Wahl der richtigen Kodierung der Individuen und

b) die Definition problemadäquater Operationen.

Im Falle eines (Syntax-)baums kann man z.B. die Baumknoten als Gene kodieren: $g = (Sub, Mult, Add, a, b, b, d)$, vgl. die Abb. 29.1.

Operationen können z.B. sein: Punkt-Mutation (= Verändern eines Baumknotens), Permutation (= Vertauschen von Operationen), Abschneiden (=Abschneiden von Knoten mit einer bestimmten Tiefe), Löschung (=Löschen

von Teilbäumen), Erweiterung (= Anhängen von Teilbäumen), Verdopplung (=Anhängen von Teilbäumen, die im Baum vorkommen), Teilbaum-Mutation (= Verändern von Baumknoten in einem Teilbaum), Teilbaum-Crossover (= Überkreuzen von Teilbäumen). Neu eingeführt sind hier also vor allem die Teilbaumoperationen. Um nicht in jeder Generation alle Operationen durchführen zu müssen, kann man z.B. zufällig aus einem Operatorpool in jeder Generation eine Menge von Operationen auswählen, die zur Anwendung kommen.

29.2

29.2 Anwendungen

Die Idee der Evolution in Syntax-Bäumen kann auf die Evolution von Entscheidungsbäumen übertragen werden. Als Fitness kann man z.B. das Produkt aus Sensitivität und Spezifität (zur Bewertung des Klassifikationsergebnisses) wählen.

Betrachtet man die Bedingung einer Regel der Form „WENN Var 1 IST Attribut 1 UND ... UND Var n IST Attribut n“, so kann man mit einem GA die Attributwerte variieren und Bedingungen an Variablen einfügen oder weglassen. Die Operatoren der Regel sind dabei festgelegt. Verändert man auch die Operatoren, z.B. ein UND in ein ODER oder ein IST in ein $\geq$, dann kann man die Vorgehensweise dem GP zuordnen, da die Syntax verändert wird.

GP kann verwendet werden, um die komplexe Struktur von Netzwerken zu erlernen, wie sie z.B. im Schaltungsentwurf vorkommen. In [KML+01] wird aus zeitabhängigen Daten von Substanzen ein Netzwerk für ablaufende chemische Reaktionen erlernt.

Mit Hilfe des Genetischen Programmierens sucht [MM00] eine Funktion (Koeffizienten, Operatoren) zur analytischen Lösung der eindimensionalen Schrödinger-Differentialgleichung. Allgemein können im Rahmen des GP die variablen Bestandteile eines Ausdrucks Zahlen, Potenzen, Rechenoperationen, mathematische oder symbolische Operatoren sein.

In diesem kurzen Kapitel wollten wir Ihnen zeigen, dass es eine weitere wichtige Variante der ES gibt, das Genetische Programmieren. Während die evolutionären Algorithmen eher numerische Daten und die genetischen Algorithmen eher binäre Daten verarbeiten, beschäftigt sich das Genetische Program-

mieren vor allem mit symbolischen Daten wie Baumstrukturen, Termen oder Gleichungen.

Kapitel 30

30 Formale Aspekte evolutionärer Strategien

30

30 Formale Aspekte evolutionärer Strategien

In den vorangegangenen Kapiteln haben wir eine Reihe von Verfahren kennengelernt, von denen wir behauptet haben, sie funktionieren und seien nützlich für Anwendungsprobleme. Wir haben auch des öfteren mathematische Schreibweisen oder Sachverhalte verwendet (z.B. das Summenzeichen, die Ableitung o.ä.), aber wir haben so gut wie keine theoretischen Hintergründe kennengelernt. Z.B. wissen wir, dass man mit bestimmten Netzen stetige Funktionen approximieren kann, aber wir wissen nicht, *warum*. Wir wissen, dass ES zur Optimierung eingesetzt werden können, wissen aber nicht, *warum* ES konvergieren und eine Lösung liefern. Auch wissen wir in vielen Fällen kaum etwas darüber, wie man bestimmte Parameter von Verfahren optimal einstellen kann, um auf bestimmten Datensätzen die besten Ergebnisse zu erhalten. Oft wird behauptet, dass man experimentell arbeiten kann, und experimentell gefundene Lösungen ausreichen. Aber was ist, wenn man den Fehler nicht theoretisch abschätzen kann, Fehler aber gefährlich sein können? Was ist, wenn ein angeblich, vermutlich funktionierendes Verfahren doch in bestimmten Sonderfällen keine gute Lösung liefert?

Das Fazit aus den angestellten Überlegungen ist, dass zwar oft experimentell gefundene Anwendungslösungen ausreichen, aber gerade die Forschung die Aufgabe hat, Grundprinzipien (theoretisch) zu erklären und allgemein gültige Aussagen zu finden, um so die praktischen Lösungen abzusichern. Dabei geht es aber nicht nur um uninteressante Sonderfälle, sondern gerade um die interessanten, allgemeinen Fälle. Gerade die auf heuristischen Überlegungen basierenden Verfahren (wie z.B. Verfahren des Soft Computing) bedürfen auch theoretischer Ergebnisse. Oft ist es so, dass auch die theoretische Arbeit mühselig und langwierig ist und man oft trotz großer Anstrengungen kein befriedigendes Ergebnis findet. Eine konkrete Anwendung nutzt einem Anwender, eine interessante Theorie aber einer ganzen Anwendergemeinschaft. Zu eher praktisch orientierten Themen wie Soft Computing gehört daher auch die Erforschung des theoretischen Hintergrunds, idealerweise zusammen mit den praktischen Erfordernissen.

Wie Sie sehen, ist in diesem Buch der Theorieteil nicht sehr lang. Wir wollen uns aber doch wenigstens darum bemühen, das *Schema-Theorem* und die *Building-Block-Hypothese* im Rahmen der GA vorzustellen, die eigentlich erst den Einsatz der GA rechtfertigen. An dieser Stelle sei vorab noch erwähnt, dass es gerade zu ES eine Reihe theoretischer Arbeiten gibt, die sich mit dem

Entwurf und der Auswahl der Kodierung, der ES-Operationen oder der ES-Strategien beschäftigen und die die folgenden Begriffsbildungen verwendet.

Lehrziele:
- Wissen, was ein Schema ist und die damit verbundenen Begriffe verstehen,
- das Schema-Theorem zusammen mit der Building block-Hypothese formulieren können,
- die Probleme verstehen, die mit dem Schema-Theorem zusammenhängen,
- wissen, dass es weitere Varianten der evolutionären Optimierung gibt.

30.1

30.1 Theoretische Grundbegriffe

Um das Schema-Theorem vorzustellen, benötigen wir einige Definitionen, die zusammen mit kleinen Beispielen vorgestellt werden.

30.1

Definition 30.1 Sei Γ eine (binär kodierte) Population von Individuuen. Der Kode habe die Länge l, d.h. $\Gamma := \{[x_1, \ldots, x_l] | x_i \in \{0,1\}, i = 1, \ldots, l\}$.

a) Ein **Schema** S ist eine Teilmenge von Γ, die durch $\{0, 1, *\}$ definiert wird. Der Stern „$*$“ steht als Platzhalter für ein beliebiges Element 0 oder 1.

Bsp.: $[0,1,1,0,*] = \{[0,1,1,0,0],[0,1,1,0,1]\}$. Eine Schema kann man als Generalisierung interpretieren.

b) Die **Ordnung** $o(S)$ eines Schemas S ist gleich der Anzahl der Nullen oder Einsen.

Bsp.: $o([0,*,1,0,*]) = 3$.

c) Die **(definierende) Länge** $\delta(S)$ eines Schemas S ist gleich der Länge zwischen der ersten und letzten Null oder Eins.

Bsp.: $\delta([0,*,1,0,*]) = 4$.

d) Es sei $f : \Gamma \to \mathbb{R}_+^0$ eine Fitness-Abbildung. Angenommen, wir wählen zufällig (gemäß einer Gleichverteilung) ein Individuum eines Schemas, dann können wir die **absolute Fitness** definieren als Erwartungswert über alle im

Schema S enthaltenen Individuen γ.

$$f(S) := \frac{1}{2^{l-o(S)}} \sum_{\gamma \in S} f(\gamma) \tag{127}$$

Aktivierungselement: Begründen Sie, dass die Anzahl der im Schema S enthaltenen Individuen γ gleich $2^{l-o(S)}$ ist.

Bsp.: Es sei $S = [0, 1, 1, 0, *]$ ein Schema mit $o(S) = 4$, $\delta(S) = 4$ und (totaler) Länge $l = 5$. Es seien die Fitness-Werte $f(\gamma_1 = [0, 1, 1, 0, 1]) = 5.37$ und $f(\gamma_2 = [0, 1, 1, 0, 0]) = 4.21$ gegeben. Dann ist $f(S) = \frac{1}{2^{5-4}}(f(\gamma_1) + f(\gamma_2)) = 4.79$.

e) Die **relative Fitness** sei gegeben als der Erwartungswert über die Fitness-Werte der Individuen, die zufällig aus einer Population x stammen und zum Schema S gehören. Die Abbildung $q_x : \Gamma \to [0, 1]$ weist durch $q_x(\gamma)$ bez. x den Anteil der Individuen zu, die den Genotyp γ besitzen.

$$f_x(S) := \frac{1}{q_x(S)} \sum_{\gamma \in S} q_x(\gamma) f(\gamma) \tag{128}$$

Bsp.: Sei γ_2 aus dem Beispiel zu d) nun zweimal vorhanden, dann ergibt sich für $f_x(S)$ der Wert $\frac{1}{3}(5.37 + 2 \cdot 4.21) \approx 4.59$.

□

30.2 Das Schema-Theorem

30.2

Wir formulieren das **Schema-Theorem**.

Satz 30.2.1 Sei $(X_t)_{t=0,1,\ldots}$ die Folge von Populationen, die ein GA erzeugt; sei $c := \frac{f_{X_t}(S) - f(X_t)}{f(X_t)}$, $t = 0, 1, \ldots$ konstant. Weiter sei p_{cross} die Crossover-Rate und p_{mut} die Mutationsrate. Dann gilt

30.2.1

$$E(q_{X_t}(S)|X_0) \geq q_{X_0}(S)(1 + c)^t \left(1 - p_{\text{cross}} \frac{\delta(S)}{l - 1} - o(S) p_{\text{mut}}\right) , \tag{129}$$

d.h. Schemata mit kurzer definierender Länge, niedriger Ordnung und überdurchschnittlicher Fitness werden mit exponentiellem Wachstum bezüglich den Folgegenerationen generiert.

□

Direkt im Anschluss können wir die sog. **Building block-Hypothese** formulieren.

30.2 **Folgerung 30.2** Ein GA erzeugt (sub-) optimale Lösungen durch Erzeugen von Schemata kurzer definierender Länger, niedriger Ordnung und überdurchschnittlicher Fitness. Diese Schemata werden als sog. **building blocks** bezeichnet.

□

Der Nutzen liegt auf der Hand: Zur Lösung eines Problems müssen wir ein Problem geschickt kodieren, um die building blocks zu erzeugen, z.B. sollte man kürzeren Regeln eine höhere Fitness als längeren zuweisen.

Das Schema-Theorem ist aber in folgender Hinsicht problematisch:

a) Man müsste für *bestimmte* Operationen und ES-Strategien Varianten des Schemata-Theorems beweisen.

b) Die Konvergenz ist eine schöne Eigenschaft, aber in praktischen Aufgabenstellungen wird nur eine *endliche* Anzahl von Generationen erzeugt. Wie gut eine Lösung dann ist, wird hier nicht klar.

Den nicht langen Beweis finden Sie in [TT01] oder [Gol89]. Er wird iterativ geführt unter Verwendung von $E(q_{X_t}(S)|X_{t-1})$.

Aktivierungselement: Studieren Sie den Beweis bei Interesse. Mehr zu diesem Themenkreis finden Sie auch in [Wei02].

30.3

30.3 Weiterführende Fragestellungen

Weitere Ansätze zur Verbesserung der Optimierungseigenschaften existieren, z.B. die sog. *Nischenbildung*. Hier werden in einem Pool von Populationen bestimmte Populationen separat evolviert, um ihnen eine ungestörte Evolution zu ermöglichen.
Andere Problemstellungen betreffen die *Mehrkriterienoptimierung*, bei der mehr als eine Zielfunktion ausgewertet werden muss, z.B. bei der Stundenplanoptimierung. Ein Ziel dabei wäre, die Freistunden zu reduzieren, ein anderes, den Wünschen der Lehrkräfte zu genügen.

In [PM02] werden *coevolutionäre Strategien* mit evolutionären verglichen. In einer coevolutionären Strategie werden mehrere Populationen evolviert, die Informationen austauschen dürfen.

Die neuere Arbeit [CPG03] stellt die Frage: Sollte man zur Lösung eines Problems besser eine evolutionäre Optimierung mit einer großen Population durchführen oder sollte man besser mehrere evolutionäre Optimierungen mit kleineren Populationen durchführen? Als Faustregel kann man sich merken, dass es besser ist eine Optimierung mit einer großen Population durchzuführen, insbesondere dann, wenn die Ordnung der building blocks hoch ist.

Mit diesem Kapitel wollen wir erreichen, dass Sie daran denken, dass neben den Anwendungen auch die Theorie eine wichtige Rolle im Soft Computing spielt, wenngleich sich dieses Buch eher der Modellierung und den Anwendungen in der Bioinformatik widmet. Gerade die Herleitung von Eigenschaften erlaubt die Anwendung der Algorithmen. Zur Formulierung des Schema-Theorems haben wir den wichtigen Begriff des Schemas eingeführt. Unter Berücksichtigung der Building block-Hypothese sollte eine geeignete Kodierung innerhalb einer Anwendung modelliert werden. Interessant kann auch die Berücksichtigung neuerer Modellierungsoptionen, wie die der Mehrkriterien-Optimierung, sein.

Kapitel 31

Evolutionäre Strategien in der Bioinformatik

31

31

31 Evolutionäre Strategien in der Bioinformatik

Nachdem wir in den letzten Kapiteln die grundlegenden Algorithmen und auch die wichtigsten theoretischen Begriffe kennengelernt haben, stellen wir nun eine Auswahl der zahlreichen Anwendungsmöglichkeiten von ES in der Bioinformatik dar. Auch berücksichtigen wir erneut die Chemieinformatik.

Bei der Anwendung einer ES sind insbesondere die Kodierung der Individuen, die Festlegung der Mutations- und Rekombinationsoperatoren sowie die Auswahl einer geeigneten Fitness-Abbildung wichtig. Deshalb gehen wir in den meisten Anwendungen systematisch auf diese Punkte ein.

Lehrziele:
- Verstehen, wie die Varianten der ES zur Krebsvorhersage anhand von Microarray-Daten eingesetzt werden können,
- Wissen, wie man ein multiples Sequenzalignment mit GA entwerfen kann,
- Nachvollziehen können, wie aus kurzen Sequenzabschnitten längere Sequenzen durch Optimierung rekonstruiert werden können,
- wissen, wie man ES auf Moleküldaten anwenden kann,
- verstehen, dass ES im Molekularem Docking von Interesse ist,
- weitere Anwendungen aufzählen können.

31.1 Krebsvorhersage mit Simulated Annealing

Die Aufgabe besteht in der Klassifikation von Microarray-Daten zur Krebsvorhersage und in der Identifikation der relevanten Gene [Deu03].

Kodierung: Microarray-Daten bestehen aus einer Menge G von Genen. Ein Krebstyp ist als Klasse zugeordnet. Verschiedene Untermengen G_i von G dienen als Startpopulation.

Mutation: Hinzufügen eines Gens, Weglassen eines Gens in G_i.

Rekombination: Wird nicht verwendet.

Fitness-Kriterium: leave-one-out-Performanz unter Verwendung eines nearest-neighbour-Klassifikators, d.h. für jeden korrekt vorhergesagten Wert wird der Wert Eins addiert plus einen Belohnungsterm für kleine Interklassenabstände.

Reproduziert wird mit einer Wahrscheinlichkeit, die proportional zu einem Gewicht w ist, das als Exponential der Differenz des Scores zwischen G_i und mutierter Genmenge brechnet wird. Der Prozess wird mit Simulated Annealing abgekühlt.

Anwendungsversuche:

a) SRBCT-Daten (Abk. für: small round blue cell tumors) von Kindern. 2308 Gene sind vorhanden. Mit einer Auswahl derjenigen Gene, die am besten alleine klassifizieren, wird gestartet. Die Daten werden in 4 Klassen unterteilt. Durchschnittlich 13 Gene sind in einem evolvierten Prädiktor vorhanden.

b) Leukämie-Daten mit den 2 Klassen lymphoplastische Leukämie (ALL) und akute myeloide Leukämie (AML). 7129 Gene werden betrachtet (38 Trainings- und 34 Testpunkte).

c) Analyse von Diffuse large B-cell lymphoma (DLBCL) anhand 6817 Genen.

d) 214 Tumor-Samples, 16063 Gene, Einteilung in 14 Klassen.

31.2 Krebsvorhersage mit Genetischen Algorithmen

Wiederum besteht die Aufgabe in der Klassifikation von Microarray-Daten und in der Identifikation der relevanten Gene, hier wiederum zur Klassifikation von Krebstypen [OT03].

Kodierung: Untermenge von Genen.

Mutation: u.a. Punktmutationen.

Rekombination: Crossover-Operationen.

Fitness-Kriterium: basierend auf Cross-Validierung und Testdatenfehler (eines Maximum-Likelihood-Klassifikators), u.a. Roulette-Wheel-Selektion.

Anwendungsversuche:

a) Datensatz „NC160“ mit 9 Tumorklassen (Brust, Zentrales Nervensystem, Dickdarm, Leukämie, Melanom, Lunge, Eierstock, Niere, Geschlechtsorgane), 9703 Genen und 64 Expressionsprofilen.

b) Datensatz „GCM“ mit 14 Klassen, 16063 Genen und 198 Expressionsprofilen.

Neben der Genexpressionsanalyse ist das Multiple Alignment ein interessantes Anwendungsgebiet evolutionärer Strategien.

31.3 Multiples Alignment mit Genetischen Algorithmen

Ein multiples Alignment kann verwendet werden zum Auffinden ähnlicher funktioneller Abschnitte oder zur Erklärung evolutionärer Prozesse. Dynamisches Programmieren findet ein optimales Alignment, ist aber komplex in der Berechnung für viele Sequenzen. Die Berücksichtigung von Lücken (engl.: gaps) ist dann auch problematisch. Eine alternative Möglichkeit ein möglichst optimales multiples Alignment heuristisch zu finden ist die Verwendung genetischer Algorithmen [SF03].

Kodierung: Neben A, C, T, G werden sog. Concensus-Codes verwendet, die eine Kurzform für eine alternative Auswahl darstellen, z.B. „Y“ für „C oder T“. Eine Sequenz wird aufgesplittet in vier binäre Sequenzen, je eine für jedes Nukleotid mit einer „1“ für das Vorhandensein eines solchen.

Mutation: Punktmutation.

Rekombination: Für alle 32Bit-Abschnitte zweier binärer Sequenzen wird ein Crossover durchgeführt.

Fitness-Kriterium: gewichtete Summe der $n(n-1)/2$ Paarähnlichkeiten von n Sequenzen, die mit einer Score-Matrix gebildet werden. Die Score-Berechnung kann aufgeteilt werden in Übereinstimmungen, Nicht-Übereinstimmungen, Lücken. Das Alignment wird vom Ende zum Anfang aus den evolvierten Sequenzen konstruiert.

Anwendungsversuche: 20 kurze DNA-Sequenzen von 60 bis 300 Basenpaaren mit ca. 50% Übereinstimmung. Mit 64 Individuen wurde gestartet. Die Mutationsrate betrug weniger als 1%. Die fittesten Individuen überlebten. Je mehr Basenpaare verwendet wurden, desto weniger qualitative Alignments wurden gefunden, wenngleich für weniger lange Sequenzen vergleichbare Ergebnisse zu Standardmethoden gefunden wurden.

31.4 Rekonstruktion von Sequenzen

Nehmen wir an, wir hätten beim Sequenzieren ein *Spektrum* von Nukleotiden der Länge 3 erhalten: {ACT, CTC, TCT, CTG, TGG}. Als Überlappungen sind Abschnitte der Länge 2 zugelassen. Dann erhielten wir daraus die Sequenz ACTCTGG. Allerdings können sich falsche Dreierkombinationen als Fehler einschleichen. Deshalb benötigt man einen Algorithmus zur Optimierung der Rekonstruktion [FC03, Kap. 3].

Kodierung: Vektor von Permutationen der Elemente des Spektrums.

Mutation: Keine.

Rekombination: Greedy Crossover; das erste Nukleotid eines Kind-Individuums wird zufällig gewählt, die weiteren Nukleotide werden der Reihe nach so gewählt, dass eine maximale Überlappung gegeben ist.

Fitness-Kriterium: Länge der längsten zusammenhängenden Subsequenz einer vorgegebenen maximalen Länge.

Anwendungsversuche: GenBank-Sequenzen werden betrachtet, und zwar Sequenzen der Länge 109, ..., 509 und Spektren der Länge 100, ..., 500. Die Übereinstimmung sinkt von fast 100% auf 82% bei zunehmender Sequenzlänge.

Wir kommen nun zu Anwendungen aus dem Bereich Drug Design.

31.5 Optimierung im Drug Design Prozess

Wir betrachten erneut den chemischen Raum $Chem$ aller Moleküle und erklären wie man bei einer Optimierung vorgehen kann [SSS03], [BS00]. Vom Standpunkt der Optimierung aus kann man folgendes Zielkriterium formulieren: Finde das optimale Molekül $M \in Chem$ unter den Nebenbedingungen $N_1, \ldots, N_n$, $n \in \mathbb{N}$. Eine wichtige Nebenbedingungen, die erfüllt sein muss, ist die Synthesemöglichkeit bzw. die praktische Existenz eines Moleküls. In diesem Fall kann man auch von *gültigen* Molekülen sprechen. Weitere Nebenbedingungen sind z.B. das Enthaltensein bestimmter funktioneller Gruppen oder ein maximales Molekülgewicht, geringe Toxizität, usw. Durch die praktischen Anforderungen an ein spezifisches Drug, sind eine Fülle von Nebenbedingungen die Regel.

Es kann erwünscht sein, nicht nur das Optimum zu finden, sondern auch Moleküle, die die Optimalitätskriterien beinahe erfüllen, um Alternativen für das weitere Drug Design zu haben. Wir wollen solche Moleküle als *Drug-like* bezeichnen, und die zugehörige Menge als $Lead$. Wir suchen also (wie in Kap. 5) einen Algorithmus, der $Lead \subset Chem$ bestimmt. Anschließend muss im Labor getestet werden ob ein $M \in Lead$ tatsächlich ein geeignetes Drug ist, worauf wir aber nicht eingehen.

Wichtig ist in diesem Zusammenhang auch die Merkmalswahl, d.h. die Auswahl der Merkmale, die ein $M \in Lead$ von $M \in Chem - Lead$ unterscheiden. Man kann davon ausgehen, dass eine spezifische Menge $Lead$ dünn (engl.: sparse) in $Chem$ verteilt ist, aber nicht unbedingt, dass sich $Lead$ einfach von $Chem - Lead$ unterscheiden lässt. Die Elemente aus $Chem$ liegen transformiert in einem Merkmalsraum $Merk$ vor, z.B. als Werte von *Deskriptoren*, so dass dann dort die Merkmalswahl durchgeführt werden kann.

Aktivierungselement: Welche Probleme kann die Abbildung $Chem \rightarrow Merk$ in Hinblick auf eine Optimierung mit ES verursachen?

Wie könnte man nun die oben beschriebene Optimierungsaufgabe im Virtuellen Screening bzw. De-Novo-Design lösen? Die Antwort erscheint zunächst leicht: durch eine ES. Dazu muss eine Kodierung festgelegt werden, geeignete Mutations- und Rekombinationsoperatoren gefunden werden und eine Fitness-Funktion angegeben werden.

Kodierung: denkbar ist (je nach Aufgabenstellung) eine Aminosäuresequenz, eine SMILES-Zeichenkette oder ein Fragment-basierter Ansatz.

Mutation: Austausch einer Aminosäure, Änderung eines SMILES-Zeichens (unter Beachtung der Syntax), Änderung eines Fragments (unter Beachtung chemischer Expertenheuristiken).

Rekombination: Sequenz-Crossover, „Molekül-Crossover“.

Fitness-Funktion: Sequenzähnlichkeit, Ähnlichkeiten der Deskriptoren basierend auf SMILES-Zeichenketten bzw. Fragmenten.

Die Rekombination von Molekülen bereitet Probleme. Konnte man z.B. zwei binäre Vektoren oder eine Sequenz leicht rekombinieren, so fällt uns die Rekombination von Molekülen schwer. Es ist aber gerade die Rekombination, die die (schnelle) Konvergenz einer ES sichert.

31.6 Evolutionäre Strategie im Molekularen Docking

Betrachtet wird das Problem, einen Liganden optimal in ein Zielprotein einzupassen (Molekulares Docking) [Yan03]. Wir geben wieder Kodierung, Mutations- und Rekombinationsoperatoren sowie die verwendete Fitness-Funktion an.

Kodierung der Individuen: In einem Vektor werden die 3D-Position des Liganden, die Rotationswinkel, Drehwinkel von rotierbaren Bindungen und die Parameter von z.B. Gauß-Funktionen zur Mutation gespeichert, letzteres zur integrierten Evolvierung der ES-Parameter.

Mutation: z.B. Gauß-basierte Änderung der Position, Winkel und Änderung der Schrittweiten.

Rekombination: gewichteter Mittelwert zweier Vektoren.

Fitness: Energiefunktion, die sich aus verschiedenen Bestandteilen wie inter- und intramolekularer Energie zusammensetzt. Je niedriger die Energie, je höher die Fitness des Moleküls.

31.7 Weitere Anwendungen

Wir weisen in diesem Abschnitt noch auf weitere Arbeiten hin: Die Arbeit [HKKN00] handelt vom Auffinden neuer bioaktiver Inhibitoren des Dopamin-Transporters (DAT), da Kokain die Freisetzung von Dopamin behindert. Es werden 2D-QSAR-Methoden verwendet. Zur Variablenselektion wird ein genetischer Algorithmus eingesetzt.

In [RBM+98] wird eine gekoppelte Fragestellung untersucht: Man betrachte die HIV-Protease. Diese mutiert häufig, so dass ein spezifischer Inhibitor nicht mehr geeignet bindet. Die grundlegende Strategie der Optimierung des Inhibitors funktioniert hier durch die Mutation der HIV-Protease nicht. Der Ansatz verwendet nun eine *coevolutionäre* Strategie, bei der versucht wird, durch zwei entgegengesetzte Optimierungsziele einen optimalen Inhibitor zu

finden. Man optimiert nun die HIV-Protease, um eine maximale Aktivität bei der Inhibition zu erhalten (worst case: resistenter HIV-Virus) und optimiert gleichzeitig damit den Inhibitor, der die Aktivität der Protease minimiert.

Weitere Anwendungen sind:

- [CLM00]: Vorhersage von gemeinsamen RNA-Sekundärstrukturen mit einem genetischen Algorithmus.
- [Kel98]: Oligonukleotid-Analyse zur funktionellen Genomanalyse, auch bei niedriger Sequenz-Homologie geeignet, unter Verwendung eines genetischen Algorithmus.
- [RWO+03]: Analyse von Microarray-Daten mit GP. In einem Baum entsprechen die Pfade symbolischen Funktionen zur Berechnung des Expressionslevels; die inneren Knoten enthalten die vier Grundrechenoperationen, die Blätter die Variablen und Koeffizienten.
- [LKM03]: Optimierung der Reihenfolge von Genexpressionsdaten zur 2D-Visualisierung durch Formulierung als Traveling-Salesperson-Problem.
- [HB03]: Erkennung der Promoter-Regionen als Motive von DNA mit einem GP-Automaten, d.h. ein Automatenmodell wird evolviert, z.B. durch Hinzufügen/Weglassen von Zuständen.
- [KLM03] und [HOST03]: Evolution von phylogenetischen Bäumen.
- [NHH98]: Multiples Alignment mit genetischen Algorithmen.
- [FC03, Kap. 6]: U.a. Optimierung (Energieminimierung) der Proteinfaltung in einem 2D-Gitter.
- [FC03, Kap. 12]: Optimierung metabolischer Pfade in (Petri-)Netzwerken mit ES.
- [Bra03]: Modellierung der Parameter von Differentialgleichungssystemen in biochemischen Pfaden durch adaptives Lernen und durch ES.

In diesem Kapitel haben wir eine Reihe unterschiedlicher Anwendungen der ES vorgestellt. Namentlich handelte es sich um die Analyse von Microarray-Daten, um die Optimierung und Rekonstruktion von Sequenzen und um Anwendungen im Drug Design. Viele weitere Anwendungen sind bereits durchgeführt worden, wie die Vorhersage der RNA-Struktur oder die Evolution phylogenetischer Bäume. Aus diesen zahlreichen Resultaten ergibt sich auch eine zentrale Bedeutung der ES innerhalb der Bioinformatik.

Kapitel 32

32 Evolutionäre Strategien in Fuzzy-Systemen

32

32 Evolutionäre Strategien in Fuzzy-Systemen

Wir haben bereits drei Möglichkeiten zur Evolution regelerzeugender Systeme kennengelernt:

- Evolution von (Entscheidungs-)bäumen,
- Classifier-Systeme und
- indirekt durch Evolution der Gewichte neuronaler Netze mit anschließender Regelgenerierung (das haben wir nicht explizit diskutiert).

Wir wollen in diesem letzten Abschnitt zu ES deshalb ergänzend auf die Möglichkeit der Integration von Evolution und Fuzzy-Systemen eingehen.

Lehrziele:
- Wissen, dass man ein Regelsystem mit dem Michigan-Ansatz optimieren kann,
- Wissen, dass man eine Menge von Regelsystemen mit dem Pittsburgh-Ansatz optimieren kann,
- die Idee der Optimierung von Zugehörigkeitsfunktionen verstehen,
- die Mutations- und Rekombinationsoperatoren zur Optimierung von Regeln, Regelsystemen und Zugehörigkeitsfunktionen kennen.

32.1 Ansätze zur Regeloptimierung

Seit den 90er Jahren wird versucht, Fuzzy-Systeme mit Hilfe von ES zu verbessern (Regel- und Klassifikationsperformanz). Wir wollen drei grundlegende Ansätze vorstellen.

Beim **Michigan-Ansatz** gehen wir davon aus, dass wir *ein* Mamdami-Fuzzy-System vorliegen haben, das wir z.B. per Hand erstellt haben. Wir vermuten, dass wir die Performanz des Systems durch ES steigern können. Im Michigan-Ansatz wird jede Regel R_i einer Regelmenge $\mathcal{R}$ als Individuum kodiert, z.B. wird aus

WENN $A_{i1} = a_{i1}$ UND ... UND $A_{in} = a_{in}$ DANN Klasseninformation$_i$

das Individuum $[a_{i1}, \ldots, a_{in}]$, wobei $a_{ij} = *$ zugelassen ist.

Beim **Pittsburgh-Ansatz** haben wir $m > 1$ Mamdami-Fuzzy-Systeme vorliegen und versuchen durch ES ein besseres System aus den m Systemen zu erhalten. Beim Pittsburgh-Ansatz kodieren wir ein Regelsystem $\mathcal{R}_i$ mit n_i Regeln als Individuum $[R_1, \ldots, R_{n_i}]$.

Während die ersten beiden Ansätze darauf abzielen, Regeln bzw. Regelsysteme zu evaluieren, kann man ES auch verwenden, um die Zugehörigkeitsfunktionen zu evolvieren.

Der Einfachheit halber verwenden wir hier zur Erklärung als Zugehörigkeitsfunktionen immer symmetrische Dreiecksfunktionen, die durch die zwei Parameter m (Mitte des Dreiecks) und d (Abstand der Seitenecken von der Mitte). Dann kodieren wir k Zugehörigkeitsfunktionen als

$$[(m_1, d_1), \ldots, (m_k, d_k)] \ . \tag{130}$$

32.2 Mutations- und Rekombinationsoperatoren

Diese Ansätze aus dem vorhergehenden Abschnitt kann man modifizieren oder kombinieren, je nach Problemstellung. Wir werden nun noch einfache Mutations- und Rekombinationsoperatoren vorstellen.

1.) (Michigan-Ansatz) Mutation: Bei einer Punktmutation wird ein $A_{ij} = a_{ij}$ ersetzt durch $A_{ij} = a'_{ij}$, wobei a'_{ij} aus dem Wertevorrat der Fuzzy-Variablen (evtl. inkl. $*$) stammt, z.B. könnte aus [HOCH, MITEL, MITTEL] [MITTEL, MITTEL, MITTEL] werden (Punktmutation der ersten der drei Variablenausprägungen).

Rekombination: Bei einem Crossover würden z.B. aus den Kodierungen zweier Regeln (mit je zwei Variablen) [MITTEL, BITTER], [HOCH, MILD] als Ergebnisregeln [MITTEL, MILD], [HOCH, BITTER] entstehen (Austausch der zweiten Variablenausprägungen).

2.) (Pittsburgh-Ansatz) Mutation: z.B. Weglassen einer nicht so fitten Regel oder Hinzufügen einer zufällig erzeugten Regel.

Rekombination: z.B. Austausch von Regeln aus den Regelsystemen $\mathcal{R}_i$ und $\mathcal{R}_j$.

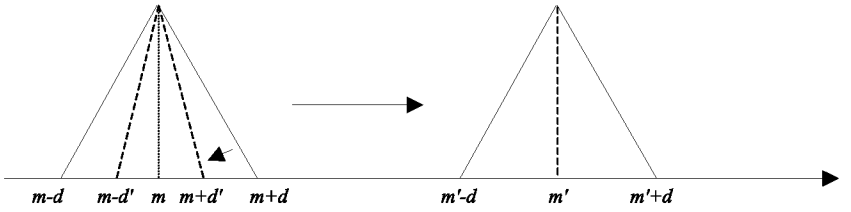

Abbildung 32.1. Mutationen von Zugehörigkeitsfunktionen

3.) (Zugehörigkeitsfunktionen) Mutation: Der Parametertupel (m_i, d_i) wird verändert zu $(m_i + N(0, s_m), d_i)$ (Verschiebung) oder $(m_i, d_i + N(0, s_d))$ (Verbreiterung bzw. Verengung), vgl. die Abb. 32.1. Es kann auch die Anzahl der Zugehörigkeitsfunktionen verändert werden. Die Mutationsrate für Anzahlveränderungen sollte kleiner als die von Punktmutationen sein.

Rekombination: Aus zwei Tupeln (m_i, d_i) und (m_j, d_j) wird $(\frac{m_i+m_j}{2}, \frac{d_i+d_j}{2})$ oder $(\frac{m_i+m_j}{2}, d_i)$ bzw. eine andere Kombination.

Die Fitness einer Regel kann z.B. durch deren Konfidenz, Häufigkeit und Länge definiert werden. Die Fitness einer Regelmenge kann durch die Klassifikationsperformanz, der (mittleren) Regelperformanz und der Anzahl der Regeln berechnet werden.

Wie wir bereits mehrfach gesehen haben, können mittels Regeln erklärende Systeme adaptiert werden. Die Erklärungsleistung solcher Systeme muss aber nach einer Adaption nicht optimal sein. Auch ein durch einen Anwendungsexperten modelliertes System kann Schwächen aufweisen. Deshalb wurden drei typische Ansätze vorgestellt, mit denen man Regeln, Regelmengen oder Zugehörigkeitsfunktionen optimieren kann, um bessere Systeme zu erhalten.

Kapitel 33

33 Naturanaloge Algorithmen – Überblick

33

33 Naturanaloge Algorithmen – Überblick

Stellt man die Frage: „Was ist Bioinformatik?“, so wird man sicher zuerst an die Anwendung von Methoden der Informatik (Datenbanken und Algorithmen) in der Biologie, Biochemie oder Chemie denken. Andererseits kann man auch die Anwendung biologischer Prinzipien in der Informatik in Betracht ziehen, also eine Art „Infologie“ betreiben. Wie wir sehen werden, gibt es viele interessante Aspekte im Rahmen solch einer Infologie, und was besonders interessant ist und wie wir später sehen werden, diese Infologie hat ihrerseits wieder Anwendungen im Rahmen der Bioinformatik. Nichts desto trotz bleiben wir aber im Folgenden beim Begriff „Naturanaloge Algorithmen“ oder kurz NAA.

Lehrziele:

- Einen ersten Überblick zur Literatur im Bereich der Naturanalogen Algorithmen bekommen,
- die Idee des DNA-Computing kennen und das Experiment von Adleman beschreiben können,
- die Idee des Membrane-Computing kennen,
- verstehen, wie die Prinzipien des natürlichen Immunsystems auf Künstliche Immunsysteme übertragen werden,
- die postive bzw. negative Selektion verstehen,
- das Prinzip der klonalen Selektion wiedergeben können,
- einige Einsatzgebiete von Künstlichen Immunsystemen angeben können,
- das Gebiet des Künstlichen Lebens einordnen können,
- wissen wie eine Künstliche Chemie funktioniert,
- einen zellulären Automaten definieren können,
- die Idee von Langton's Ameise verstehen,
- über einige mit dem Künstlichen Leben zusammenhängende Aspekte nachdenken können,
- den Begriff des Künstlichen Lebens von den Begriffen Systembiologie und Organic Computing trennen können,
- Beispiele für natürliches emergentes Verhalten angeben können,
- den Partikelschwarmalgorithmus formulieren können und den Unterschied zu evolutionären Algorithmen erklären können.

33.1

33.1 Literaturübersicht

Wir stellen im Folgenden fünf unterschiedliche, wichtige Richtungen der NAA vor, die wir in den nächsten Abschnitten und im nächsten Kapitel weiter verfolgen. Die ersten beiden Paradigmen orientieren sich stärker an dem Entwurf von allgemeinen Biocomputern, während die anderen drei Paradigmen stärker die Datenanalyse betonen und deshalb im Rahmen des Buches etwas ausführlicher präsentiert werden. Zu allen Themen existiert vertiefende Spezialliteratur.

1. DNA-Computing [Adl94], [Pis98],
2. Membrane-Computing [Pau00], [Pau02],
3. Künstliche Immunsysteme [CT02], [Cas03], [Das98],
4. Künstliches Leben (engl.: Artificial Life) [Lan95] und
5. Schwarmalgorithmen [BDT99], [KE01] (z.B. Ameisen- , Bienen-, oder Vogelschwärme).

Neben dieser Bücherauswahl nennen wir wieder eine Auswahl an speziellen Zeitschriften, Konferenzen und Webseiten.

Konferenzen:

- International Conference on Artificial Immune Systems (ICARIS; erste Konferenz in 2002),
- International Conference on Ant Algorithms (ANTS),
- European Conference on Artificial Life (ECAL),
- International Conference on Artificial Life (ALIFE),
- International Meeting on DNA-Based Computers,
- Workshop on Membrane Computing (WMC).

Zeitschriften:

- Artificial Life,
- Biosystems,
- Natural Computing,
- Theoretical Computer Science.

Webseiten:

- www.alife.org,
- http://iridia.ulb.ac.be/ mdorigo/ACO/ACO.html,

- http://ls11-www.cs.uni-dortmund.de,
- www.bionik.tu-berlin.de,
- www.organic-computing.org,
- http://psystems.disco.unimib.it.

33.2 DNA-Computing

33.2

Beim **DNA-Computing** geht es um das Rechnen auf molekularer Ebene mit DNA. Warum kann das sinnvoll sein? Eine biochemische Reaktion kann deutlich schneller ablaufen als die elektrische oder optische Signalübertragung, so dass das Rechnen deutlich schneller sein kann. Allerdings muss hierzu die Rechenaufgabe erst biologisch übersetzt und hinterher wieder zurück übersetzt werden, was einen sehr großen Aufwand bedeutet.

Das DNA-Computing ist eher als Randgebiet des praktisch orientierten Soft Computing einzuordnen, und steht dem technisch orientierten Bio-Computing näher (Stichwort „Bio-Chips"). Wir geben an dieser Stelle nur die Ideen eines Experiments von Adleman [Adl94], [Pis98] wieder.

In diesem Experiment geht es darum zu entscheiden, ob ein Pfad in einem Graphen die Hamilton-Eigenschaft hat oder nicht, d.h. es wird ein Pfad gesucht, der von einer Ecke zu einer anderen verläuft und alle anderen Ecken genau einmal besucht. Es ist ein NP-hartes Problem. Solche Probleme sollen in Zukunft durch Bio-Computing effizienter lösbar sein. [Adl94] beschreibt einen polynomiellen, nichtdeterministischen Algorithmus auf molekularer Ebene zur Lösung des Problems. Dazu sei ein Graph G mit einer Eckenmenge V (vertices) gegeben:

Algorithmus 33.1 (Algorithmus von Adleman) 33.1

1. Generiere einen Zufallspfad in G.

2. Filter Eins: Behalte alle Pfade, die in der richtigen Startecke beginnen und in der richtigen Zielecke enden.

3. Filter Zwei: Behalte nur diejenigen Pfade, die Ecken $n = \sharp V$ besuchen.

4. Filter Drei: Behalte nur diejenigen Pfade, die jede Ecke mindestens einmal besuchen.

5. Ist die gefilterte Menge ungleich der leeren, so existiert ein Hamilton-Pfad; ist sie leer, dann existiert kein solcher Pfad.

□

Der Trick liegt in der Kodierung des Problems in DNA-Strängen, von denen Kopien und Kombinationen im Labor erzeugt werden. Dort können 2^n Kopien in n Schritte erzeugt werden. Theoretisch könnte man das Hamiltonsche Pfadproblem für ca. 70 Ecken lösen. Für mehr Ecken würde man mehr Moleküle als auf unserer Welt vorhanden sind benötigen. Anwendungen eines Molekularcomputers können die Speicherung sein, das Entschlüsseln von kodierten Daten oder das Sequenzieren von DNA-Daten.

Eine vorgeschlagene Sprache, DNA-Pascal, arbeitet auf dem Alphabet {A, C, T, G}. Eine Operation „Right Append" könnte wie folgt aussehen (mit T als DNA-Strang und a aus dem Alphabet):

$$T_{\text{neu}} := T_{\text{alt}}a; \tag{131}$$

Ein Programmtext spiegelt dann die molekularen Operationen wider, die zur Lösung eines Problems benötigt werden.

33.3 Membrane-Computing

Beim Membrane-Computing geht es um die Modellierung eines Rechenmodells, dass auf biologischen Membranen basiert. Die Einheit einer Zelle wird durch eine Membran von der Umgebung abgegrenzt. Diese wird durch eine Lipid-Doppelschicht gebildet, also aus zwei Schichten von Lipiden, die im Innern hydrophob und nach außen hydrophil sind. Membranen sind aber nicht undurchlässig, sondern sie können spezifische Moleküle passieren lassen. Das Flüssigmosaikmodell für Membranen geht davon aus, dass neben den Lipiden Membranproteine angeordnet sind, die spezifisch für Membranfunktionen verantwortlich sind wie beispielsweise das Steuern von Pumpen und Kanälen.

Eine Membran im Sinne der Informatik ist im einfachsten Fall eine Abstraktion der oben beschriebenen biologischen Membran. Für die Abstraktion benötigt man Ansammlungen (auch Multi-Mengen genannt) von *Objekten*, d.h.

Elemente in Objekten dürfen mehrfach vorkommen, und eine Menge von *evolutionären Regeln*. Die Objekte werden in der Membran platziert und dürfen sich verändern. Diesem Modell ist daher eine parallele Verarbeitung inhärent. Mehrere Membranen können durchnummeriert und ineinander verschachtelt werden und so eine Zelle mit mehreren Regionen bilden. Ein solches System vom Membranen, Objekten und Regeln heißt kurz *P-System* [Pau02].

Definition 33.2 Ein **P-System** wird formal beschrieben als **33.2**

$$P = (O, M, w_1, \ldots, w_m, R_1, \ldots, R_m, i_{\text{out}}) \tag{132}$$

mit

a) einer Menge von Objekten O, die durch alphabetische Symbole beschrieben werden kann,

b) einer Membranstruktur M, die aus m einzelnen Membranen besteht, die jeweils eine Region einschließen,

c) m Ansammlungen w_i, $i = 1, \ldots, m$, von Objekten, die den einzelnen Regionen zugeordnet sind. Diese Ansammlungen können durch Zeichenketten beschrieben werden.

d) m endlichen Mengen R_i, $i = 1, \ldots, m$, von evolutionären Regeln für die Objekte aus O, die wiederum den einzelnen Regionen zugeordnet sind. Die Regeln haben die Form $u \rightarrow v$, $u \in O$ und $v \in O \times \tilde{O}$, wobei $\tilde{O}$ die Menge $\tilde{O} = \{\text{here}, \text{out}\} \cup \{in_j | j = 1, \ldots, m\}$ bezeichnet. Sie gibt die Zielrichtung der Regel an, wobei „here“ der Übersichtlichkeit halber weggelassen werden kann.

e) einem Index i_{out} für eine einzelne Membran, die als *Output-Membran* angesehen wird.

□

Die Regionen $(w_1^t, \ldots, w_m^t)$ im Zeitpunkt t können (parallel) in einen Folgezustand $(w_1^{t+1}, \ldots, w_m^{t+1})$ überführt werden.

Das folgende Beispiel ist für $k = 2$ aus [Pau00] adaptiert. Das P-System (132) ist gegeben durch

- $O = \{a, c, \tilde{c}, d\}$,
- $M = [_1 \ [_2 \ ]_2 \ [_3 \ ]_3 \ ]_1$,

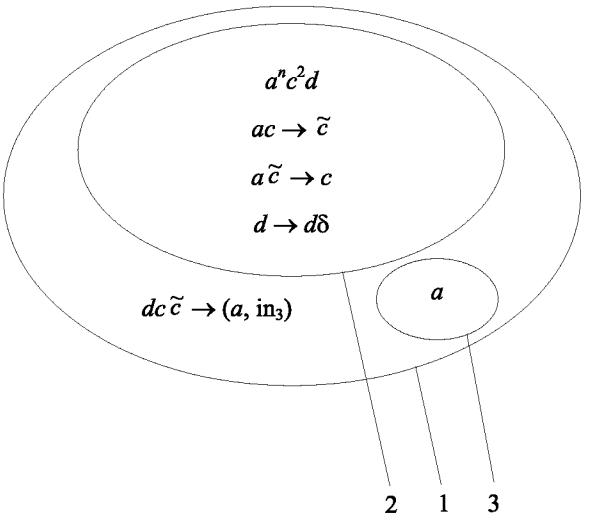

Abbildung 33.1. Ein P-System zur Entscheidung, ob n eine gerade Zahl ist

- drei Regionen w_1, w_2, w_3 mit $w_1 = \varepsilon$ (ε leeres Wort), $w_2 = a^n c^2 d$ und $w_3 = a$,
- drei Regelmengen $R_1 = \{dc\tilde{c} \to (a, \text{in}_3)\}$, $R_2 = \{ac \to \tilde{c}, a\tilde{c} \to c, d \to d\delta\}$ (die ersten beiden Regeln werden der dritten in der Bearbeitung vorgezogen) und $R_3 = \emptyset$,
- dem Index für die Output-Membran $i_{\text{out}=3}$.

Das P-System ist in der Abb. 33.1 veranschaulicht. In der Membran 2 wird 2 von n subtrahiert. Die Regel in der ersten Membran kommt nur zum Einsatz, falls n kein Vielfaches von 2 ist, also n ungerade ist. Sie wird maximal einmal angewendet. Die Output-Membran (Membran 3) enthält in diesem Fall zwei Elemente. Ist n gerade, enthält sie nur ein Element nach Beendigung der Berechnung.

Viele Erweiterungen der P-Systeme sind denkbar, die beispielsweise die Permeabilität der Membran betreffen. Man könnte eine zusätzliche Regel einführen, die gleichzeitig beide Durchlassrichtungen betrifft: $(a, \text{out}; b, \text{in})$. Auch kann man Fuzzy-Logik in P-Systemen verwenden, um die auftretenden Unschärfe der Objekte in Regeln zu modellieren [CMMR04]. Aus unscharf modellierten Objekten resultieren dann wiederum unscharf modellierte Objekte, formuliert als Fuzzy-Multimengen.

33.4 Künstliche Immunsysteme

Elemente von verteilten Algorithmen und ES finden sich in den sog. **Künstlichen Immunsystemen** wieder, die das natürliche Immunsystem zum Vorbild haben [CT02], [Cas03]. Die verteilten Bausteine eines natürlichen Immunsystems sind verschiedene Zellen und Moleküle. Verhindert werden soll eine Körperinfektion durch (i.d.R. fremde) Antigene. B- und T-Zellen erkennen Antigene und können bei Aktivierung Antigene binden bzw. chemische Stoffe aktivieren, die ein Antigen binden. Einige dieser Zellen, die sog. Gedächtniszellen, speichern die Struktur der erkannten Antigene. Dies ist aber zunächst nur eine vereinfachte Darstellung. Viele chemische Bausteine übernehmen Teilfunktionen. Die Aufgabe der verteilten Bausteine ist die Optimierung des Immunsystems gegenüber den fremdartigen Antigenen.

Eine erste Errungenschaft ist die Entdeckung des Prinzips der Impfung durch Jenner (1796). Den ersten Medizin-Nobelpreis erhielt von Behring (1901) für seine Arbeiten über Antikörper. Ohne hier auf die weitere Geschichte der Immunolgie einzugehen sei angemerkt, dass Tonegawa (1987) den Nobelpreis für seine Arbeiten im Gebiet der molekularen Immulogie erhielt.

Zwei grundsätzlichen Prinzipien der Immunabwehr sind gegeben durch

- das *klonale Selektionsprinzip* (engl.: clonal selection principle) mit den verschiedenen Unterprinzipien, z.B. positive oder negative Selektion, klonale Expansion, Anpassung der Immunantwort, Autoimmunität, Unterscheidung zwischen körpereigenen und -fremdem Stoffen.
- die *Immun-Netzwerk-Theorie* (engl.: immune network theory), entstanden nach Jerne (1974) in dem die Immunsystemreaktionen als Interaktionen in einem Netzwerk modelliert werden.

Wir können an dieser Stelle nicht auf die komplexe Theorie der biologischen Immunologie eingehen, sondern werden uns direkt den Künstlichen Immunsystemen (Abk.: KImm) zuwenden, die in vereinfachter Weise die Funktionsweise des natürlichen Immunsystems nachahmen. Dabei berücksichtigen wir vor allem die Übertragung des klonalen Selektionsprinzips auf KImm. Anwendungsgebiete von KImm sind daher diejenigen, in denen Muster effizient erkannt und unterschieden werden sollen:

- Mustererkennung,
- Anomalie-Erkennung,

- Agentensysteme,
- allgemeine Aspekte der Datenanalyse (z.B. Optimierung).

An dieser Stelle erklären wir nun, wie ein KImm aufgebaut werden kann. Ein KImm operiert abstrakt im Raum binärer Strings, in dem Abstände, z.B. die Hamming-Distanz, zur Erkennung (durch Selektionsalgorithmen) eingesetzt werden. Aber auch der reelle Zahlenraum oder symbolische Räume mit den entsprechenden Abständen können verwendet werden. Anstelle von Abstand spricht man auch oft von Verwandtschaft, Ähnlichkeit oder Affinität (engl.: affinity). Wir erklären im Folgenden die positive (negative) Selektion als Erkennung durch Affinitätsberechnungen auf Mengen von Zeichenketten:

33.3 **Algorithmus 33.3 (Algorithmus Positive/Negative Selektion)**
0. Das Selbst sei repräsentiert durch eine Menge S von Strings;

1. Erzeuge zufällig eine initiale, potenzielle Menge P von Strings (T-Zellen);

2. Vergleiche alle Elemente p von P mit allen Elementen s von S;

3. positive Selektion: Ist die Affinität von p zu einem s höher als ein Schwellwert (Selbsterkennung), dann wird p in die Menge V der verfügbaren T-Zellen im System aufgenommen, ansonsten gelöscht.

3.' negative Selektion: Ist die Affinität von p zu einem s höher als ein Schwellwert (Fremderkennung), dann wird p gelöscht, ansonsten wird p in die Menge V der verfügbaren T-Zellen im System aufgenommen.

□

Der *klonale Selektionsalgorithmus* steuert das KImm zur abstrakten Problemlösung.

33.4 **Algorithmus 33.4 (Klonale Selektion)**
0. Erzeuge zufällig eine initiale Menge von Individuen P;

1. Für alle antigenen Muster y wiederhole die folgenden Schritte bis ein geeignetes Stoppkriterium erfüllt ist:

2. Berechne die Affinität von y zu allen $x \in P$;

3. Klonale Selektion und Expansion: Sortiere die x nach ihrer Affinität aufsteigend und wähle diejenigen mit höchster Affinität aus. Kloniere die ausgewählten x proportional zu ihrer Affinitätsrate;

4. Mutiere die x in einer Höhe, die antiproportional zu der Affinität ist. Die Mutationen werden zu P hinzugenommen. Wähle die besten $x \in P$ zur Speicherung zur weiteren Antigenerkennung aus;

5. Ersetze einige wenige $p \in P$ durch neue Individuen.

□

Bei der klonalen Selektion wird wie bei ES die Mutation verwendet. Eine Hybridisierung mit ES ist vorstellbar. Die Individuen können verteilt repräsentiert werden. Wir listen einige Anwendungen aus [CT02] und einige weitere auf.

- Erkennung von Infektionskrankheiten,
- Viruserkennung im Computer,
- Erkennung ungewöhnlicher Bildbestandteile,
- Lösen des TSP-Problems,
- Erkennung von Software- oder Hardware-Fehlern,
- Modellierung von Ökosystemen als KImm,
- Ein Spamfilter, der auf Immunität basiert [OW03],
- RBF-Netzlernen durch das KImm-Paradigma [CZ02].

An dieser Stelle präsentieren wir die Inhalte der Arbeit [AI03], in der KImm auf das Problem der Analyse von Genexpressionsdaten angewendet wird.

Beispiel 33.5 KImm zur Genexpressionsanalyse [AI03]: Es werden u.a. Daten von 72 Patienten betrachtet. 7109 Gene werden in die Analyse eingebunden. Alle diese Patienten leiden an einer von zwei Leukämiearten. Gezeigt wird, dass ein neuronales Netz, eine SVM und das KImm fehlerlos klassifizieren. Allerdings fehlen hier Versuchswiederholungen mit verschiedenen Einteilungen in Trainings- und Testdaten. Eine Klasse wird als Selbst angesehen. Mittels negativer Selektion und klonaler Selektion wird eine Erkennung erreicht. Die gespeicherten „Schwellwert-Regeln“ für Leukämie erinnern an Regressi- **33.5**

onspolynome: $1.21896X_{3675} + 1.5858X_{4474} + 1.46134X_{1540} - 1.19885X_{2105} + 1.84803X_{757} + 1.82983X_{4038}$.

□

33.5

33.5 Künstliches Leben

Was sind die Prinzipien des Lebens? Kann man Leben künstlich modellieren? – Das Arbeitsgebiet **Künstliches Leben** befasst sich mit dem Entwurf von Bausteinen und Algorithmen, die Leben nachahmen. Man fordert in diesem Zusammenhang oft die Reproduzierbarkeit der Bausteine und die Lernfähigkeit der künstlichen Systeme.

Aktivierungselement: Welches der folgenden Beispiele sind Beispiele für lernende und/oder reproduzierbare Systeme? Ordnen Sie die „Lebensformen" gemäß ihrer Intelligenz. Begründen Sie Ihre Entscheidung.
- Roboterhund,
- Roboterhand,
- Computerviren,
- KImm,
- (echter) Delphin,
- Balu, der (Stoff-)Bär.

Das Gemeinsame dieser Systeme ist, dass sie Teilaspekte (echten) Lebens *simulieren*, allerdings nicht mit dem Anspruch echtes Leben vollständig zu erklären, sondern eher mit dem Anspruch Realweltprobleme zu lösen oder aber auch nur Spaß an den entdeckten Prinzipien zu haben. In diesem Gebiet kann man durchaus auch eine künstlerische Komponente entdecken. Viele Systeme faszinieren durch ihre Formen, Bewegungen, Farben o.ä. Allerdings wird man intuitiv diesen künstlichen Systemen noch nicht zubilligen, dass sie *wirklich* leben, obwohl sie ein etwas mehr oder weniger selbstständiges Verhalten zeigen.

Neben der Entwicklung künstlichen Lebens ist ein interessanter Ansatz selbstverständlich auch die Erforschung des z.Z. real existierenden Lebens, d.h. das Verstehen komplexer lebender Systeme (z.B. Gruppenverhalten). Früher war

dieses Arbeitsgebiet auch stärker unter dem Namen „Kybernetik“ bekannt. Wir stellen im Folgenden einige Originalarbeiten aus dem Gebiet des Künstlichen Lebens vor.

Ein sehr nahe an dem Gebiet der Erforschung des Ursprungs des Lebens von molekularer Seite aus ist das Gebiet der *Künstlichen Chemie* [DZB01]. Eine **Künstliche Chemie** ist ein von Menschenhand (im Computer simuliertes) reales chemisches System. Man kann sich eine Künstliche Chemie $(M, \mathcal{R}, \mathcal{A})$ aus folgenden Bestandteilen zusammengesetzt vorstellen:

- $M = \{m_1, \ldots, m_n\}$ ist eine Menge von Molekülen m_i (*Reaktor* oder *Suppe*), die explizit z.B. durch Symbole oder Zahlen (manchmal in einem 2D oder 3D zellulären Automaten, vgl. Def. 33.8) dargestellt werden. Eine implizite Darstellung durch eine Grammatik ist ebenfalls denkbar.
- $\mathcal{R}$ ist eine Menge von Regeln, die Reaktionsmechanismen beschreiben in der Form: $m_1 + \ldots, m_q \longrightarrow m'_1 + \ldots + m'_r$. Matrixoperationen können ebenfalls als Modellierung von Reaktionsgleichungen verwendet werden.
- $\mathcal{A}$ ist ein Algorithmus, der den Ablauf der Regeln im Reaktor steuert. Denkbar sind u.a. stochastische Algorithmen oder eine Modellierung durch Differentialgleichungen.

Beispiel 33.6 Sei $M := \{A, B\}$, **33.6**
R_1: $A + A \longrightarrow A + A + B$ (zwei A ergeben zwei A und ein B),
R_2: $A + B \longrightarrow A + B + B$,
R_3: $B + A \longrightarrow B + A + B$,
R_4: $B + B \longrightarrow B + B + A$ mit

Algorithmus $\mathcal{A}$: Bis zu einem Stoppkriterium wiederhole den folgenden Schritt (Die Reaktionspartner werden z.B. in einem Array fester Länger zufällig angeordnet): Array$(t + 1)$:= Reaktion(rand(Array(t)), rand(Array(t)));

□

Wir möchten an dieser Stelle nicht auf die Fülle der existierenden (ähnlichen) Modelle eingehen, da das Grundprinzip hier bereits deutlich klar wird. Verfeinerte 3D-Modelle können zur Simulation der Molekularen Dynamik eingesetzt werden.

Wir beschreiben an dieser Stelle das etwas umfangreichere System [Hut02] (Squirm3: http://www.sq3.org.uk).

33.7 **Beispiel 33.7** Vorgegeben sind in diesem Reaktor Atome mit einem Typ aus $\{a, b, c, d, e, f\}$ und einem Zustand aus $\{0, 1, 2, 3, 4, \dots\}$.

Die Regeln sind gegeben durch:
R_1: $e8 + e0 \longrightarrow e4e3$ ($e8$ und $e0$ ergeben ein verbundenes Molekül $e4e3$),
R_2: $x4 + y1 \longrightarrow x2y5$,
R_3: $x5 + x0 \longrightarrow x7x6$,
R_4: $x3 + y6 \longrightarrow x2y3$,
R_5: $x7 + y3 \longrightarrow x4y3$,
R_6: $f4f3 \longrightarrow f8 + f8$ (Aufspaltung),
R_7: $x2y8 \longrightarrow x9y1$ (Umwandlung),
R_8: $x9y9 \longrightarrow x8 + y8$ (Aufspaltung).

□

In einem Experiment wurde ein Startmolekül $e8a1b1c1f1$ in einer (20×20)-Matrix („Welt“) mit 75 separaten Atomen gegeben. Nach einigen tausend Schritten fanden sich replizierte Moleküle in der Suppe.

Weitere Ergänzungen sind denkbar, wie z.B. die Einführung einer Nachbarschaft, in der Reaktionen ablaufen dürfen oder die Einführung einer zusätzlichen Operation, die die Veränderung eines Atoms durch kosmische Strahlung (Evolution) simuliert. Zusätzliche Atome können von Zeit zu Zeit zur Suppe hinzugegeben werden.

Aktivierungselement: Sehen Sie sich das System aus dem Bsp. 33.7 auf der o.g. Webseite an.

In [Tou03] wird das *Wachstum von Pflanzen* nachgeahmt unter Verwendung von 3D-Mutationsoperatoren wie lokalen Rotationen oder Verzweigungen. Als Fitness wird ein Flächeninhalt aus der Vogelsicht verwendet. Je größer die grüne Fläche von oben ist, desto fitter ist die Pflanze. Die Simulation von Leben wird auch in [PK00] betrieben. In diesem Fall werden wachsende Algen der Art Chlorella Kessleri simuliert. Jede Zelle hat eine interne Energie

zur Verfügung, die durch (simulierte) Außenbedingungen verändert werden kann, z.B. durch toxische Stoffe oder durch die Lichtverhältnisse.

Als modelltheoretische Grundlage für **Langton's Ameisen** dient uns ein *zellulärer Automat*. Zelluläre Automaten (engl.: cellular automata) fanden bereits in den 40er Jahren durch Ulam und von Neumann ihren Einsatz.

Definition 33.8 (Zellulärer Automat, 2D) 33.8

Unter einem **zellulären Automaten** (Z, f) verstehen wir ein Matrixschema $Z^{(t)} = (z^{(t)})_{i,j}$ mit $i = n_1, \ldots, n_2$, $j = m_1, \ldots, m_2$ mit $m_1 \leq m_2, n_1 \leq n_2 \in \mathbb{Z}$ (endlicher zellulärer Automat), oder aber das Schema enthält unendlich viele Einträge (unendlicher zellulärer Automat). Mit $^{(t)}$ wird der Zeitpunkt t bezeichnet. Die z-Einträge im Matrixschema gehören einem Wertebereich an, z.B. binäre Werte $\{0, 1\}$. Zum Zeitpunkt $t = 1$ wird der zelluläre Automat initialisiert. Die Transferfunktion f transformiert $Z^{(t)}$ in $Z^{(t+1)}$.

□

Beispiel 33.9 Sei (Z, f) ein endlicher zellulärer Automat mit $Z^{(1)} := \begin{pmatrix} 01 \\ 10 \end{pmatrix}$ und $f(Z^{(t)}) := \begin{pmatrix} (z_{11}+1) \mod 2, (z_{12}+1) \mod 2 \\ (z_{21}+1) \mod 2, (z_{22}+1) \mod 2 \end{pmatrix}$. Dann erhalten wir $Z^{(2)} = \begin{pmatrix} 10 \\ 01 \end{pmatrix}$, $Z^{(3)} = \begin{pmatrix} 01 \\ 10 \end{pmatrix}$, usw. 33.9

□

Eine Erweiterung für den 3D-Fall besteht einfach in der Verwendung einer zusätzlichen dritten Dimension von Zellen.

Kann eine einfache Verhaltensregel (einfache Transferfunktion) ein komplexes Verhalten (komplexes Matrixmuster) bedingen? Wir betrachten einen binären, unendlichen 2D zellulären Automaten, bei dem initial alle Werte im Matrixschema Paare (x, y, a) sind. Alle x, y sind Null. Alle a sind Null bis auf $a(0, 0) := 1$. Letzteres bedeutet, dass auf diesem Feld eine Ameise (ein Agent) befindet, der die Richtung „oben" gespeichert hat. Sei $a = 2$ „links", $a = 3$ „unten" und $a = 4$ „rechts". Die Transferfunktion sei durch die folgende Anweisungsfolge gegeben [Lan95]:

1. Ist $a > 0$, dann invertiere Bit des Eintrags. (Ist 0 weiß und 1 schwarz, dann invertiere die Farbe.)
2. Bewege die Ameise einen Eintrag weiter in Richtung a.
3. Ist das neue Feld weiß, so erhöhe a um 1 (ist $a = 4$, setze $a = 1$). (Linksdrehung)
4. Ist das neue Feld schwarz, so erniedrige a um 1 (ist $a = 1$, setze $a = 4$). (Rechtsdrehung)

In diesem Sinne bahnt sich die Ameise einen Weg durch das Matrixschema und erzeugt ein Muster. Die Bestimmung des Musters, dessen Dauerhaftigkeit und dessen Symmetrien ist ein komplexes Problem.

Aktivierungselement: a) Skizzieren Sie die ersten 10 Zeitschritte. b) Was passiert, wenn der Automat mit zwei Ameisen gestartet wird? Sind die Ergebnisse von der Startposition der beiden Ameisen abhängig?

Das Schema eines zellulären Automaten kann auch dazu verwendet werden, um sich reproduzierende Strukturen zu modellieren, wie z.B. *Langton's Loop*. Das sind Automaten, die Symmetrien innerhalb ihres Musters ausbilden. Die symmetrischen Teile des Musters werden dann getrennt, und die getrennten Teile des Musters entwickeln sich separat weiter. Dieses und Weiteres über *Selbstreplikation* findet der Leser in dem historischen Überblick [Sip98].

Bevor wir zum Abschluss unseres kleinen Abschnitts über Künstliches Leben kommen, geben wir aus einer Liste offener Probleme und Ziele innerhalb dieses Gebiets [BMP+00] die wichtigsten Fragestellungen wieder:

I) Wie entsteht Leben aus dem Unlebendigen?

- Erzeugung eines molekularen Organismen-Prototyps *in vitro*,
- Erreichen des Übergangs zum Leben innerhalb der Künstlichen Chemie *in silico*,
- Bestimmung des Überlebenspotenzials neuartiger lebender Organismen,
- Simulation eines (einzelligen) Organismus während seines ganzen Lebens,
- Erklärung des Entstehungsprozesses von Regeln und Symbole aus der (physikalischen) Dynamik lebender Organismen.

II) Welche Vorzüge bieten lebende Systeme? Welches sind ihre Grenzen?

- Bestimmung der unabdingbaren Invarianten in der Evolution des Lebens,
- Bestimmung der evolutionären Mechanismen, die einen Übergang von spezifischen zu allgemeinen Verhaltensweisen ausmachen,
- Bestimmung der Vorhersagbarkeit der evolutionären Konsequenzen bei Veränderung von Organismen und Ökosystemen,
- Modellierung einer Kommunikations- und Informationstheorie für evolvierende Systeme.

III) Welchen Bezug hat Leben zum Bewusstsein, zu Maschinen und zur Kultur?

- Demonstration der Emergenz von Intelligenz und Bewusstsein in Künstlichen Lebensformen,
- Etablierung von ethischen Prinzipien für Künstliches Leben.

Diese Liste beinhaltet eine Fülle von praktischen, theoretischen, aber auch philosophischen Fragestellungen. Anhand dieser drei übergeordneten Punkte kann man erahnen, dass die Informatik, die Chemie und die Biologie neben anderen Disziplinen eine herausragende Rolle im Bereich des Künstlichen Lebens spielen.

Wir gehen an dieser Stelle abschließend auf zwei aktuelle Arbeitsgebiete ein, bei der das „Künstliche" eine Rolle spielt, die *Systembiologie* und das *Organic Computing*. Die Systembiologie versucht komplexe, dynamische Abläufe im Körper zu modellieren, z.B. die Funktionsweise einer Zelle (siehe das E-Cell Projekt unter http://www.e-cell.org/). Gesucht werden in der Systembiologie die „Schaltpläne" der Natur. Das künstliche Leben ist in diesem Fall eine Simulation der realen Lebensbausteine und deren Interaktionen. Die Disziplin des Organic Computing versucht ebenfalls der zunehmenden Komplexität Herr zu werden, und zwar der Komplexität heutiger Soft- und Hardware-Systeme. Ziel des Organic Computing ist durch die Einbeziehung der *Selbstorganisation* Computer- oder Softwaresysteme zu entwerfen, die sich selbstständig weiter entwickeln und kontrollieren. Solch ein System kann man als eine Art künstliches Leben auffassen.

Im Folgenden interessiert uns wieder die Anwendung der biologischen Prizipien auf algorithmisch zu lösende Probleme, z.B. auf Optimierungsprobleme. Es geht im Folgenden nicht um eine Simulation (also nicht um eine vollständige Nachahmung, um Künstliches Leben zu erschaffen), sondern um die Über-

nahme gerade derjenigen Prinzipien, die für die Lösung einer Problemstellung relevant sind.

33.6 Schwarmalgorithmen

Lebewesen handeln nicht nur individuell, sondern auch in Abhängigkeit anderer (gleichartiger) Lebewesen. Bestimmte Arten treten dabei als **Schwarm** auf, d.h. das Überleben einzelner Individuen im Schwarm wird durch das Auftreten als gemeinsamer Schwarm gesichert oder optimiert. Beispiele für Schwärme sind u.a.:

- Ameisen (werden im nächsten Kapitel behandelt),
- Bienen, Wespen,
- Termiten,
- Vogelschwärme und
- Fischschwärme.

Das gemeinsame Auftreten kann z.B. der gemeinsamen Futtersuche, der Aufzucht von Nachkommen, dem Nestbau, der Verteidung oder der (unbewussten) Optimierung physikalischer Parameter dienen. Der Schwarm zeigt ein beobachtbares, komplexes, systematisches Verhalten, das von dem einzelnen Individuum nicht bewusst beabsichtigt ist. Man bezeichnet das Verhalten als **emergentes** Verhalten.

Zum Lösen von algorithmischer Aufgabenstellungen haben sich seit Mitte der 90er Jahre insbesondere die Ameisenkolonialgorithmen oder die Partikelschwarmalgorithmen (engl.: particle swarm algorithms, Abk. PSA) [EK95] etabliert. Die PSA kann man als verteilte, stochastische Suchalgorithmen interpretieren. Deshalb betrachten wir im Anschluss einige (wenige) PSA-Anwendungen und im nächsten Kapitel die Ameisen sowie die Ameisenkolonialgorithmen.

Wir betrachten ein Feedforward-Netz mit Eingabeneuronen, einer versteckten Schicht und Ausgabeneuronen und beschreiben den Einsatz der PSA zur Optimierung von Neuronalen Netzen [AkM02]. Ein Partikelschwarm besteht aus individuellen Partikeln, die sich in ihrer Position und in ihrer Geschwindigkeit unterscheiden. Die Position X und die Geschwindigkeit V werden während des Lernens in der folgenden Art und Weise verändert (Indices für Individuen

und Dimension werden weggelassen, w Gewicht, c_1, c_2 kleine Zufallszahlen):

$$X(t+1) := X(t) + V(t+1) \text{ mit} \tag{133}$$

$$V(t+1) := wV(t) + c_1(G(t) - X(t)) + c_2(L(t) - X(t)) \ . \tag{134}$$

$G(t)$ ist die Position des aktuell besten Partikels in der Menge aller Partikel (globale Optimierung). $L(t)$ ist die beste Position des individuellen Partikels (lokale Optimierung). An dieser Stelle soll uns die Variante aus [AkM02] nicht weiter interessieren, bei der immer nur die besten Partikel in einem Zeitschritt überleben, so dass $L(t)$ nicht verwendet wird, aber explizit auch schlechtere Lösungen zugelassen werden können.

Experimente wurden durchgeführt für den IRIS-Datensatz (4-3-3-Netzwerk), New-Thyroid-Datensatz (5-3-3-Netzwerk) und Glass-Datensatz (9-7-7-Netzwerk). Die Resultate wurden durch den MSE und durch die Fehlklassifikationsrate verglichen. Mit der vorgestellten Variante wurden bessere Ergebnisse erzielt als mit Backpropagation. Allerdings reichen die Experimente nicht aus, um die Verbesserung signifikant zu belegen, da zu wenige Datensätze untersucht wurden und auf Versuchswiederholungen verzichtet wurde.

Es müssen nicht alle Partikel im Partikelschwarm als Nachbarn angesehen werden. In der Arbeit [MCRN02] werden Experimente mit verschiedenen Nachbarschaftsmodellen anhand von 20 Partikeln durchgeführt. Die „extremen" Topologien sind der vollständig verbundene Graph und der Ring (jeder Partikel hat zwei Nachbarn). Die verwendeten Topologien sind: 3D-Pyramide, Graph mit Knoten der Ordnung 4, Graph mit 4 Cliquen a 5 Partikel. Verwendete Datensätze sind u.a. Diabetes und verschiedene Zeitreihen. Die beste Topologie war die Pyramiden-Topologie. Die Kritik an dieser Arbeit ist die Gleiche wie an der vorher genannten Arbeit.

Weitere Anwendungen sind:

- Grobe Vorjustierung von Clusterzentren im Rahmen des Dokumentenclustering durch einen PSA. Anschließend werden die initialen Clusterzentren durch einen k-means-Algorithmus verfeinert [BYSZ02].
- Das Verhalten fliegender 3D-Objekte im Raum um eine Energiequelle (Futter) wird untersucht [SKPF03].
- PSA zur optimalen Merkmalswahl als Eingabe eines neuronalen Netzes für eine QSAR-Anwendung [AC02].

In diesem Kapitel wurden eine ganze Reihe verschiedener Ansätze aus dem Bereich der Naturanalogen Algorithmen vorgestellt. Der prominenteste Vertreter des Natural Computing (sinngemäß dt.: durch die Natur inspiriertes Rechnen) ist das DNA-Computing. Das Experiment von Adleman zeigt, dass es prinzipiell möglich ist, mit DNA zu rechnen. Viele theoretische Modelle wurden bislang entwickelt, deren effektive Umsetzung in die Praxis noch aussteht. Eines dieser Modelle ist auch das des P-Systems im Membrane-Computing, bei dem Berechnungen in den verschiedenen Membranen parallel durchgeführt werden können. Ein P-System kann auf vielerlei Arten variiert werden. Eine Möglichkeit ist auch die Verwendung von Fuzzy-Logik in Membran-Systemen.

Die Künstlichen Immunsysteme nutzen das Prinzip des natürlichen Immunsystems, Fremdes zu erkennen und sich dagegen zu wehren. Eine klassische Anwendung ist das Erkennen und Abwehren von Computerviren im Rechner. Das klonale Selektionsprinzip ist dabei im Rahmen der KImm das algorithmische Pendant zum evolutionären Algorithmus im Rahmen der Optimierung. Solch ein KImm wurde beispielsweise zur Analyse von Genexpressionsdaten verwendet.

Im Gebiet des Künstlichen Lebens geht es vor allem um die Simulation des echten Lebens, zumindestens von typischen Aspekten davon. Die Modell einer Künstlichen Chemie kann eingesetzt werden, um chemische Reaktionsabläufe zu simulieren. Zelluläre Automaten sind Werkzeuge, um die (parallele) Simulation von Teilchen-artigen Abläufen zu simulieren. Ein einfaches Beispiel solch eines Ablaufs ist der durch Langton's Ameise gegebene. Im Bereich des Künstlichen Lebens wurden viele Ansätze entwickelt; dennoch ist man noch davon entfernt, echtes Leben in seiner Gesamtheit simulieren zu können. Im Rahmen der Systembiologie hat man sich die Aufgabe gesteckt, eine ganze Zelle zu simulieren.

Eine weitere interessante Möglichkeit, biologische und informatorische Prinzipien zu verbinden, sind die Schwarmalgorithmen. Hier haben sich bislang die Modellierungen des Partikelschwarms auf der Basis der Analogie zu Vogelschwärmen und die Ameisenkoloniealgorithmen hervorgetan.

Kapitel 34

34 Ameisen und Ameisenkoloniealgorithmen

34

34 Ameisen und Ameisenkolonialgorithmen

Unter den Schwarmalgorithmen, die sich durch eine **verteilte Architektur** auszeichnen, sind gerade die **Ameisenkolonialgorithmen** (engl.: ant colony optimization algorithms, Abk.: ACO) zur Optimierung eingesetzt worden. Das Paradigma der Ameisenkolonialgorithmen wurde von Dorigo 1991 zum erstenmal beschrieben [DMC91]. Dies wollen wir im nächsten Abschnitt beleuchten. Die Ameisenkolonialgorithmen stellen insofern eine erweiterte Alternative zu ES dar. Sie sind den Partikelschwarmalgorithmen zuzuordnen, da einzelne Ameisen eine lokale (zufällige) Suche vollziehen und die Ameisenkolonie durch die Pheromonspuren eine globale Suche vollzieht.

In diesem Kapitel wollen wir aber zuerst auf natürliche Ameisenkolonien eingehen und uns erst dann den künstlichen Ameisenkolonien inklusive den dazugehörigen Algorithmen widmen. Bei den natürlichen Ameisen interessiert uns vor allem die (natürliche) Optimierung ihrer Lebensbedingungen. Dabei ist insbesondere interessant, wie die Wegoptimierung bei der Nahrungssuche bewerkstelligt wird. Diese Analogie ist es vor allem, die dann bei den Ameisenkolonialgorithmen die entscheidende Rolle spielt.

Lehrziele:
- einen kleinen Überblick über das gemeinsame Leben der natürlichen Ameisen gewinnen,
- die Idee der Ameisenkolonialgorithmen angeben können,
- erklären können, wie das „Ant System“ auf das TSP-Problem angewendet werden kann,
- Erweiterungen des Ant Systems kennen,
- Anwendungsmöglichkeiten der Ameisenkolonialgorithmen angeben können, wie die Optimierung der Vorhersage der Proteinfaltung in HP-Modellen.

34.1 Natürliche Ameisen

Ameisen [Kir01], [HW94] sind *eusoziale* Insekten. Sie können einzeln nicht (lange) überleben. Sie leben in sog. *Ameisenstaaten.* Eine Reihe von Aufgaben zeigt das Miteinander und arbeitsteilige Füreinander der Ameisen:

- Futtersuche,
- Erzeugung von Nachkommen (durch Geschlechtstiere) und Nachwuchspflege (durch sterile Arbeiterameisen),
- Nestverteidigung und
- Nestbau.

Verschiedene Ameisen können auf einzelne Aufgaben spezialisiert sein. Allerdings scheint es nicht so zu sein, dass jede einzelne Ameise weiß, was das Ergebnis der Handlungen aller Ameisen ergeben soll, sondern jede einzelne Ameise erfüllt ihr Programm (*emergentes Verhalten*). Den größten Teil der Ameisen in einem Ameisenstaat stellen die Arbeiterameisen („Arbeiterkaste"). Eine oder mehrere Königinnen können Eier legen. Diese entwickeln sich zu Larven, dann zu Puppen, schließlich zu Ameisen. Eine Jungkönigin kann im Falle ihres (nicht sehr wahrscheinlichen) Überlebens einen neuen Staat gründen. Die Größe eines Staates kann in der Größenordnung von 100 Ameisen bis über 1 Million reichen. Verschiedene Nester können Verbände bilden. Wanderameisen bauen keine Nester. Es gibt eine Vielfalt an verschiedenen Ameisen, was Form und Größe betrifft.

Interessant im Zusammenhang mit den Ameisenkoloniealgorithmen ist das Vorhandensein von Drüsen im Ameisenkörper, die Verdauungssekrete, Futtersäfte, Botenstoffe u.a. produzieren. Die Botenstoffe (**Pheromone**) dienen dem Signalisieren von Botschaften an andere Ameisen. Die Duftstoffe können durch die Antennen wahrgenommen werden. Wege können so markiert werden, dass andere Ameisen diesen Weg erkennen können („Ameisenstraße"). Dies kann zum Nahrungstransport interessant sein. Die Ameisen finden den kürzesten Weg zum Futter bzw. zum Nest zurück. Pheromone können aber auch der Gefahrenwarnung dienen. Beispielsweise können verschiedene Ameisenvölker miteinander kämpfen. Die Analogie, an der wir nun anknüpfen, ist die verteilte Arbeitsorganisation und die Kommunikation durch Pheromone.

34.2

34.2 Ameisenkoloniealgorithmen

Bei der Modellierung eines neuen Optimierungsparadigmas achteten die Erfinder auf die Wiederverwendbarkeit für allgemeine Optimierungsprobleme und auf die Möglichkeit die Algorithmen zu parallelisieren (Populationsansatz), siehe [DMC96], [BDT99].

Der Ansatz besteht aus der Verwendung lokaler Agenten, den künstlichen Ameisen, die miteinander über die (künstlichen) Pheromonspuren kommuni-

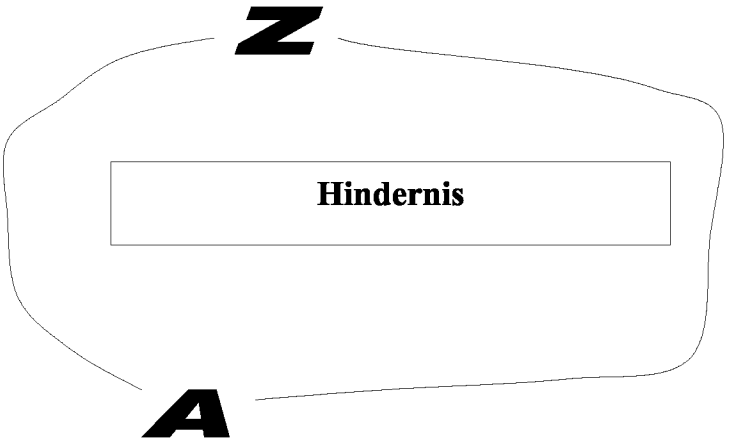

Abbildung 34.1. Eine Ameisenkolonie optimiert die Weglänge. A= Ameisennest, Z= Ziel (Futterquelle). Nach [DMC96]

zieren können. Der noch folgende Algorithmus kann als autokatalytischer Prozess angesehen werden, d.h. der Optimierungsprozess unterstützt sich selbst durch positives Feedback (positive reinforcement).

In der Abb. 34.1 starten viele Ameisen auf der Suche nach Futter. Vom Ameisennest aus hinterlässt jede Ameise eine Pheromonspur, die nach einer Zeit verschwindet. Die Stärke der Pheromonspur bestimmt die Wahrscheinlichkeit mit der sich eine Ameise auf dieser Spur bewegt. Ohne Hindernis würden alle Ameisen geradewegs zur Futterquelle laufen. Liegt ein Hindernis auf dem geraden Weg, so werden zunächst beide Wege herum erkundet. Nach einer gewissen Zeit ist die Pheromonspur auf dem kürzeren Weg stärker, da dort immer mehr Ameisen langlaufen. Die Ameisenkolonie hat damit den Weg vom Ameisennest zur Futterquelle optimiert. Wir stellen die zugrunde liegende, künstliche Ameisenkolonie als Population vor. Anschließend präsentieren wir den Algorithmus nach [DMC96], der dort noch in seiner einfachen Form „Ant System“ genannt wird.

Die Ameisenkolonie: Gegeben seien n Städte mit ihren Abständen d_{ij} zueinander. Das TSP-Problem kann nun als Graph dargestellt werden. Eine Tour durch genau alle Städte minimaler Länge ist gesucht.

Wir geben vorab die Bedeutung der im Algorithmus 34.1 verwendeten Parameter an:

- $\alpha \geq 0$: relative Wichtigkeit der Pheromonspur,
- $\beta \geq 0$: relative Wichtigkeit der Sichtweite,
- $\rho \in [0, 1[$: Dauerhaftigkeit der Spur; $1 - \rho$: Verflüchtigung,
- Q: Konstante proportional zur Spurintensität, die durch die Ameisen erzeugt wird,
- $b_i(t)$: Anzahl der Ameisen in der i-ten Stadt zum Zeitpunkt t,
- $\tau_{ij}(t)$: Pheromonstärke der Kante (i, j) zwischen zwei Städten.

Die iterative Optimierung in *Ameisenzyklen* (engl.: ant cycles) des Ameisenkoloniealgorithmus kann mit einem Kriterium, z.B. der maximalen Anzahl an Zyklen abgebrochen werden.

34.1 **Algorithmus 34.1 (Ameisenkoloniealgorithmus „Ant System“:)**
1. Initialisierung:

Setze $t := 0$;

Wähle für jede Kante eine zufällige Pheromonstärke τ_{ij};
Setze das Pheromon-Update $\Delta\tau_{ij} := 0$;
Platziere die m Ameisen auf die n Knoten;

Erstelle eine leere Tabuliste T_k für jede Ameise;

2. Tabuliste:

Für jede Ameise k füge die initiale Stadt in die Tabuliste T_k ein;

3. Wahrscheinlichkeitsberechnung:

Wiederhole die folgenden Schritte für alle Ameisen $k = 1, \ldots, m$, bis die Tabulisten voll sind (also $(n-1)$ Wiederholungen):

Suche eine Nachfolgerstadt j aus mit der Wahrscheinlichkeit $p_{ij}^k(t)$ mit

$$p_{ij}^k(t) := \frac{(\tau_{ij}(t))^\alpha \cdot (\eta_{ij})^\beta}{\sum_{k \in \text{erlaubt}_k} (\tau_{ik}(t))^\alpha \cdot (\eta_{ik}(t))^\beta} \quad , \tag{135}$$

falls j nicht in der Tabuliste T_k ist; ansonsten setze $p_{ij}^k(t)$ gleich mit der Sichtweite (engl.: visibility) $\eta_{ij} := 1/d_{ij}$;

Bewege die k-te Ameise zur Stadt j;
Füge die Stadt j in die Tabuliste T_k ein;

4. Pheromon-Update:

Für alle Ameisen $k = 1, \ldots, m$:

Berechne die Länge L_k der Tour der k-ten Ameise;
Speichere die kürzeste Tour ab;

Für jede Kante (i, j) und jede Ameise $k = 1, \ldots, m$ (Q Konstante) setze:

$$\Delta\tau_{ij}^k := \frac{Q}{L_k} \ , \tag{136}$$

falls die Kante (i, j) in der Tour T_k enthalten ist; sonst setze gleich 0.

5. Pheromonberechnung:

Für jede Kante (i, j) berechne (nach einem Ameisenzyklus)
$\tau_{ij}(t + n) := \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$;
$t := t + n$;

Erhöhe Schritt für Abbruchkriterium;
Setze für jede Kante (i, j) den Wert $\Delta\tau_{ij} := 0$;

6. Abbruch:

Wenn Abbruchkriterium nicht erfüllt, dann leere Tabulisten T_k
und gehe wieder zu Schritt 2; sonst gebe kürzesten Weg aus;

□

Für komplexere Probleme mit höherer Laufzeit werden die folgenden Varianten vorgeschlagen [BDT99, Abschnitt 2.3.2], [DS04]:

- Einschränkung der Explorationsregel (transition rule) hin zu den besseren Lösungen (eine Art simuliertes Abkühlen).
- Nicht mehr alle Ameisen dürfen nach Beendigung eines Ameisenzyklus Pheromone updaten, sondern nur diejenige Ameise, die den bislang ins-

gesamt besten Weg gefunden hat (zielgerichtete Suche), und nur die zugehörigen Kanten werden adaptiert.

- Verwendung einer lokalen Update-Regel, beim Übergang einer Ameise i nach j:

$$\tau_{ij} := (1 - \rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0 \tag{137}$$

mit $\tau_0 := (nL_{nn})^{-1}$, n Anzahl der Städte, L_{nn} ist die Länge einer Tour, die durch eine nearest-neighbour-Heuristik gefunden wurde. Die Anwendung der lokalen Update-Regel macht den beschrittenen Weg der Ameise nachfolgenden Ameisen unattraktiver. Dies erhöht die Diversität der begangenen Pfade im Suchraum.

- Verwendung einer Kandidatenliste: Um von einer Stadt aus nicht den Übergang zu allen Städten zu ermöglichen, bringt man Vorwissen ein, eine Liste von Städten, die von einer Stadt aus bereist werden darf. Dies reduziert den Suchraum hin zu lokal sinnvollen Lösungsmöglichkeiten.

34.3 Anwendungen

Im Rahmen dieses Kurses sollen zwei Anwendungen der Ameisenkoloniealgorithmen behandelt werden, zum einen die *Regelgenerierung* und zum anderen die *Vorhersage der Proteinfaltung.*

Um einen Ameisenkoloniealgorithmus anzuwenden, ist es wichtig eine geeignete Repräsentation des Problems zu finden. Betrachten wir „WENN Bedingung DANN Folgerung"-Regeln, die aus einem Datensatz gelernt werden sollen [PLF01]. Der zu optimierende Weg wird als Regel modelliert, die Stück für Stück um Attribute ergänzt wird, beginnend mit einer leeren Regel. Nach Terminierung des Algorithmus wird die beste Regel ausgegeben. Die Regellänge wird begrenzt durch ihre Häufigkeit $freq$ der Musterabdeckung. Gemäß der Performanz der Regel (Anzahl der Attribute und diskrete Werte der Attribute, Sensitivität, Spezifität) werden die Pheromone pro Ameise für die beteiligten Attribute eines Pfades geändert. Ein Abbruchkriterium kann der Informationsgehalt aller Regeln sein. Alle diese Festlegungen und Parameter können verschiedenartig verwendet werden.

Es wird eine Anwendung eines Ameisenkoloniealgorithmus zur Vorhersage der Proteinstruktur anhand der Aminosäuresequenz beschrieben [SAHH02]. Hierfür werden 2D hydrophobic-polar-Modelle verwendet (2D HP). Alle Aminosäuren können als hydrophob oder polar klassifiziert werden. Dadurch kann

die Sequenz abstrahiert werden zu einer Sequenz bestehend nur aus H's und P's. Die Faltung wird ohne Überkreuzung auf ein 2D-Gitter abgebildet, so dass die H-H Kontakte nichtbenachbarter H's gezählt werden können. Eine Konformation c mit n dieser Kontakte habe dann die freie Energie $E(c) := -n$. Das Auffinden einer Faltung mit minimaler Energie im Modell ist NP-hart.

Aktivierungselement: Versuchen Sie die Sequenz

PPHPPHHPPPPHHPPPPHHPPPPHH

so im 2D-Gitter anzuordnen, dass die freie Energie minimal ist.

Die Optimierung, die auch mit einer ES angegangen werden könnte, lässt sich mit einem Ameisenkolonialgorithmus wie folgt bewerkstelligen: Der Pfad lässt die Richtungen „geradeaus", „links", „rechts" zu. Das Pheromon-Update kann man proportional zu der freien Energie der gefundenen Faltung für jede Faltung (=Weg) durchführen. Für längere Sequenzen ≥ 36 scheint der Algorithmus aufgrund des komplexen Suchraums immer weniger performant zu werden. Weitere Anwendungen sind:

- Verkehrsflussoptimierung [HPJ02]: Ameisen finden eine optimale Verteilung von Fahrzeugen (Ameise = Fahrzeug) in einem vorgegebenen Straßensystem.
- Optimales Routing in (Telekommunikatioins-) Netzwerken.
- Job Scheduling-Optimierung.
- Clusterung von Web Server Log Files zur Untersuchung des Benutzerverhaltens [LMV03]. Die Idee der Clusterung mit Ameisenkolonialgorithmen ist die Analogie zum Nestbau der Ameisen: Ist die Ameise fremd, d.h. unähnlich, so baut sie ein neues Nest (Cluster); ist sie ähnlich, so darf sie in das Nest (dann gehört sie zum Cluster). Allerdings können Ameisen auch wieder aus dem Nest verstoßen werden, wenn ähnlichere Ameisen das Nest betreten (Cluster-Refinement).
- Optimierung von Funktionswerten [MKP$^+$00]. Verschiedene Funktionen werden optimiert, mit und ohne Nebenbedingungen, z.B. die Rosenbrock-Funktion (138), vgl. die Abb. 34.2,
- Stundenplanoptimierung [SKS02]: Das Problem der Stundenplanoptimierung wird als Graph (engl.: construction graph) modelliert. Ein Pfad im

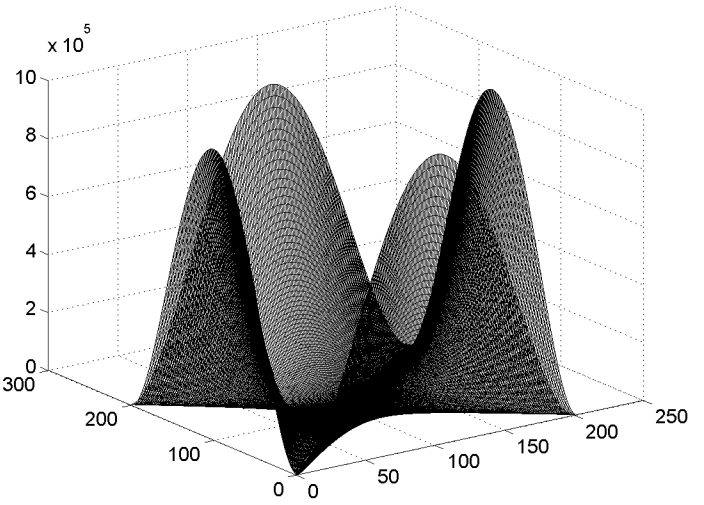

Abbildung 34.2. Rosenbrock-Funktion im Intervall $[-10, 10] \times [-10, 10]$

Graphen entspricht dann einer Lösung. Die Pheromone geben an, auf welchen Knoten, also zu welchen Zeiten, bevorzugt eine Veranstaltung gelegt wird. Dabei kann man berücksichtigen, dass Veranstaltungen gleichzeitig oder gerade nicht gleichzeitig stattfinden sollen.

- Modellierung der baumartigen Pfade, die bei Überfällen kriegerischer Ameisen entstehen [Sol00].
- Erzeugung eines Regressionsbaums unter Verwendung von Ameisenkoloniealgorithmen für eine QSAR-Anwendung [IA01].
- Multiples Sequenzalignment mit Ameisenkoloniealgorithmen [MJ03],
- Vorhersage der Bindungsstärke von Peptiden an Klasse-2-MHC-Moleküle durch Ameisenoptimierung eines MSA [KSD05].

Die Formel für die *Rosenbrock-Funktion* lautet

$$f(x, y) := 100 \cdot (x^2 + y^2)^2 + (1 + x)^2 \quad . \tag{138}$$

Die Ameisen bilden eine Gemeinschaft, in der sie Aufgaben teilen und so ihre Überlebenschancen optimieren. Die Analogie zum Auffinden des optimalen Wegs vom Ameisennest zum Futter führte zur Idee der Ameisenkoloniealgorithmen. Diese Algorithmen existieren mittlerweile in einer Vielzahl von verschiedenen Ausprägungen; das erste dieser Systeme war das „Ant System". Eine Reihe erfolgreicher Anwendungen liegt bereits vor, wie beispielsweise

die Optimierung von Telekommunikationsnetzen oder die Optimierung von HP-Modellen zur Proteinfaltung.

Kapitel 35

Ausblick

35

35

35 Ausblick

Der Autor hat mit den vorliegenden Buchkapiteln versucht, die wichtigsten Themen des Soft Computing abzudecken. Die Kapitel haben die Themen Datenanalyse, Neuronale Netze, Fuzzy-Theorie, Maschinelles Lernen, evolutionäre Strategien und naturanaloge Algorithmen behandelt, alles unter Berücksichtigung der Anwendungen innerhalb der Bioinformatik, aber auch der Chemieinformatik und der Medizin („Life Science"). Auch wurden eine Reihe von hybriden Systemen, die zwei oder mehr einzelne Paradigmen verbinden, behandelt.

Soft Computing ist bereits heute eine weit verbreitete Schlüsseltechnologie, die hilft vielerlei Anwendungsprobleme zu lösen, so auch in Zukunft verstärkt Probleme aus der Bioinformatik. Die Verbindung von Modellen, Anwendungen und Theorie macht auch gerade den Reiz des Soft Computing aus. Die weltweite Bedeutung des Soft Computing wird anhand der beiden im Jahr 2002 ausgetragenen Multikonferenzen klar:

- Multikonferenz auf Hawaii, USA: IEEE Int. Conf. on Fuzzy Systems, Int. Joint Conf. on Neural Networks, Congress on Evolutionary Computation (insgesamt 1100 Beiträge auf ca. 6600 Seiten). In 2006 wird die Multikonferenz in Vancouver, Kanada, stattfinden.
- Multikonferenz in Singapur: Int. Conf. on Fuzzy Systems and Knowledge Discovery, Int. Conf. on Neural Information Processing, Asia-Pacific Conf. on Simulated Evolution and Learning (insgesamt ca. 800 Beiträge auf ca. 4000 Seiten).

Auch die aktuellen Konferenzen über genetische und evolutionäre Algorithmen wie GECCO im Jahr 2005 in Washington, D.C., USA mit ca. 300 Beiträgen auf 2200 Seiten und IEEE CEC im Jahr 2005 in Edinburgh, GB mit ca. 400 Beiträgen auf 2800 Seiten zeigen die weltweite Bedeutung des Soft Computing, insbesondere der evolutionären und naturanalogen Methoden und deren Anwendungen, die für die Zukunft der Informatik und der Bioinformatik eine herausragende Bedeutung haben. Zusammenfassend kann man behaupten, dass alle vorgestellten Themenkomplexe in der Zukunft weiterhin dauerhaft eine bedeutende Rolle spielen werden. Lassen Sie sich also fuzz-zinieren, naturanalogisieren, evolutionär optimieren, maschinell belehren und neuronal vernetzen.

Aktivierungselement: Was halten Sie von dem Buch? Was fanden sie besonders interessant? Haben Sie Anregungen, Kritik, Verbesserungsvorschläge?

Anhang

A

Wahrscheinlichkeitsrechnung – Übersicht

A

A Wahrscheinlichkeitsrechnung – Übersicht

Wir wiederholen die grundlegenden Begriffe der Wahrscheinlichkeitsrechnung wie Ereignis, Wahrscheinlichkeit, Zufallsvariable, Erwartungswert, Mittelwert, Varianz, Standardabweichung, Korrelation, Kovarianz, Verteilung, Normalverteilung, bedingte Wahrscheinlichkeit und stochastische Unabhängigkeit. Als vertiefende Literatur sei z.B. [Bos91] und [Bos92] empfohlen.

Definition A.1 (Grundbegriffe der Wahrscheinlichkeitsrechnung) **A.1**

a) Der **Ereignisraum** Ω ist der Raum aller Elementarereignisse, z.B. die Würfelaugen $\Omega = \{1, 2, 3, 4, 5, 6\}$. Ein **Ereignis** ist ein Element aus Ω, d.h. ein Ausgang eines Zufallsexperiments.

b) Die **Wahrscheinlichkeit** P eines Ereignisses ist definiert als

$$\frac{\text{Häufigkeit eines Ereignisses}}{\text{Gesamthäufigkeit aller Ereignisse}},$$

z.B. $P(1) = P(\{1\}) = \frac{1}{6}$.

c) Eine **Zufallsvariable** X ist eine Abbildung $X : \Omega \to \mathbb{R}_0^+$ mit den Wahrscheinlichkeiten $P(\{\omega \in \Omega | X(\omega) = x\})$ (diskreter Fall) und $P(\{\omega \in \Omega | X(\omega) \in (a, b]\})$ (kontinuierlicher Fall).

d) Eine **diskrete Verteilung** ist gegeben durch $(x_i, P(X = x_i))$. Ein Beispiel einer **stetigen Verteilung** ist die bekannte **Normalverteilung** $\mathcal{N}(0, 1)$ mit Mittelwert 0 und Varianz 1 sowie der **Dichtefunktion** („Gaußsche Glockenkurve“) $\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$. Die Dichtefunktion hat Wendepunkte bei $-1, 1$ und ein Maximum an der Stelle 0.

□

Definition A.2 (Grundbegriffe der Statistik) **A.2**

a) Der **Mittelwert** oder **Erwartungswert** ist definiert als $\mu = \mathrm{E}(X) := \sum_{i=1}^{n} x_i P(X = x_i)$.

b) Die **Varianz** ist definiert als $\sum_{i=1}^{n}(x_i - \mathrm{E}(X))^2 P(X = x_i)$. Die **Streuung** oder **Standardabweichung** definieren wir als $\sqrt{\mathrm{V}(X)}$.

c) Allgemein wird die **Dichte** einer $\mathcal{N}(\mu, \sigma)$-Normalverteilung modelliert durch

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \ .$$

d) Es folgen die Definitionen der Kovarianz, der Korrelation, der bedingten Wahrscheinlichkeit $P(A|B)$ und von stochastisch unabhängigen Ereignissen A, B:

$$\mathrm{KOV}(X, Y) := \mathrm{E}((X - \mathrm{E}(X))(Y - \mathrm{E}(Y))),$$

$$\mathrm{KOR}(X, Y) := \frac{\mathrm{KOV}(X, Y)}{\mathrm{V}(X)\mathrm{V}(Y)} \in [-1, 1] \ .$$

e) Die bedingte Wahrscheinlichkeit zweier Ereignisse ist gegeben durch

$$P(A|B) := \frac{P(AB)}{P(B)} \ .$$

Dann sind A, B stochastisch unabhängige Ereignisse genau dann wenn gilt: $P(A|B) = P(A)$. Die Bedingung ist äquivalent zu $P(AB) = P(A)P(B)$.

□

A.0.1 **Satz A.0.1 (Satz von Bayes)**

A sei ein Ereignis. Die B_i seien disjunkte Ereignisse, die den Grundraum Ω überdecken, also für die $\bigcup_i B_i = \Omega$ gilt. Dann gilt:

$$P(B_i|A) = \frac{P(A|B_i)P(B_i)}{\sum_j P(A|B_j)P(B_j)} \ .$$

□

Anhang

B

Einführung in Matlab

B

B Einführung in Matlab

Matlab ist eine Sammlung von Software-Toolboxen mit vordefinierten Funktionalitäten und einer eigenen Entwicklungsumgebung. Die Standard-Toolbox „Matlab“ stellt die grundlegenden Daten- und Kontrollstrukturen bereit. Sie ist die Basis, die für alle anderen Toolboxen benötigt wird. Im Folgenden werden wir nach und nach die wichtigsten Komponenten der Matlab-Arbeitsumgebung kennenlernen. Die nachfolgenden kleinen Abschnitte umfassen Lektionen, die man studienbegleitend bearbeiten kann. Dadurch wird man in die Lage versetzt, selbstständig Programme anzuwenden und zu entwickeln. Auch kann das erworbene Wissen in weiterführenden Praktika angewendet werden und natürlich auch als Grundlage für eine Tätigkeit in der Forschung oder in der Industrie.

Unter www.mathworks.de kann die vollständige Dokumentation im HTML- oder im PDF-Format eingesehen werden. Die Ausführungen beziehen sich im Wesentlichen auf die Matlab-Version 6.5 Release 13, sind aber nicht spezifisch auf diese Version zugeschnitten, sondern allgemein gehalten. Die ersten beiden Handbücher, die man zu Hilfe nehmen sollte, sind „Getting Started“ und „Using Matlab“. Letzteres enthält bereits über 1000 Seiten.

B.1 Das Toolbox-Konzept

Matlab ist in Toolboxen untergliedert, die separat zu erwerben sind. Die *Matlab*-Toolbox ist dabei die Grundlage aller weiteren Toolboxen. Sie stellt die Arbeitsumgebung, einen Editor, eine Programmiersprache, und einen Editor für Graphische Benutzeroberflächen (engl.: graphical user interface, Abk.: GUI) zur Verfügung. Alle weiteren Toolboxen erweitern das Basispaket. Jede Toolbox stellt andere Voraussetzungen an bereits vorhandene Toolboxen.

Besonders geeignet zur Unterstützung von Datenanalysen ist die *Statistics*-Toolbox zur Beschreibung und Analyse von Datensätzen, z.B. durch eine PCA. In der *Neural Networks*-Toolbox sind gängige Netzwerktypen implementiert, z.B. Backpropagation, RBF- und Kohonen-Netze. In der *Optimization*-Toolbox sind Verfahren zur Lösung von Optimierungsproblemen enthalten, z.B. das Simplex-Verfahren. Es gibt auch eine *Fuzzy*-Toolbox, mit deren Hilfe man ein Fuzzy-System erstellen kann, das sog. ANFIS-System. In der *Bioinformatics*-Toolbox sind Module zur Datenformatierung von Bioinformatik-Daten enthalten sowie z.B. Programme zur Sequenzanalyse.

Insgesamt kann man sagen, dass Matlab als Entwicklungswerkzeug für Soft Computing-Verfahren (in der Bioinformatik) geeignet ist. Weitere Toolboxen sind z.B. die Database-Toolbox oder die Web Server-Toolbox. Dazu ist anzumerken, dass der Umfang von Matlab in den letzten Jahren ständig gewachsen ist. Die genannten und weitere Toolboxen werden im Folgenden beschrieben.

Wir gehen an dieser Stelle nicht auf die Installation einer Einzelplatz- bzw. einer Serverlizenz ein. Der Installationsprozess installiert automatisch alle durch einen Lizenzschlüssel freigeschalteten Toolboxen, z.B. in ein Verzeichnis c:\matlab6R13. Matlab läuft unter den Betriebssystemen Windows und Unix. Der Programmcode kann bis auf wenige Ausnahmen unter beiden Betriebssystemen ausgeführt werden. Pfadangaben müssen durch das Betriebssystem bedingt angepasst werden.

Aktivierungselement: Probieren Sie so viel wie möglich selbst aus.

Matlab ist hauptsächlich prozedural orientiert, was dem Entwickler entgegenkommt, der hauptsächlich in der Entwicklung und Forschung Funktionen programmiert, die sich häufiger ändern. Allerdings gibt es auch objektorientierte Elemente wie z.B. ein Neuronales Netz-Objekt.

B.2 B.2 Starten von Matlab

Entweder startet man Matlab vom Programmmenü aus (Windows) oder vom Kommandozeileninterpreter mit dem Befehl *matlab* (Linux). Damit hat man die Arbeitsumgebung gestartet. Für jemanden, der bereits mit einer Programmiersprachenumgebung gearbeitet hat, ist die Arbeitsumgebung zu einem guten Teil selbsterklärend. Mit *Control C* kann ein laufendes Programm abgebrochen werden.

B.3 B.3 Die Hilfe

Die Hilfe kann mit *help* aufgerufen werden. Auch kann sie als Menüpunkt gestartet werden. Gibt man nach *help* den gesuchten Befehl ein, z.B. *help sqrt*, so erhält man die Hilfe zu dem entsprechenden Befehl, hier zur Quadratwurzel. In Matlab gibt es eine ganze Reihe solcher Funktionen, die mit einem entsprechendem Befehl aufgerufen werden können. Kennt man einen Befehl

nicht genau, so kann man mit *lookfor* <*String*> den angegebenen String in der ersten Zeile jedes Programms suchen.

B.4 Der Editor

B.4

Den Editor kann man mit *edit* starten. Schlüsselwörter sind im Text hervorgehoben. Im Menü kann man Befehle zum Editieren finden oder die Variablen eines Programms (Workspace) ansehen. Auch gibt es Möglichkeiten, seinen Quellcode zu debuggen, z.B. durch Setzen von Breakpoints.

B.5 Die Arbeitsumgebung – Pfade

B.5

Alle von Matlab verwendeten Funktionen stehen in Pfaden. Diese Pfade sind automatisch gesetzt. Schreibt man selbst Programme, so macht es Sinn, einen Pfad auf sein Heimatverzeichnis zu setzen und auf entsprechende Unterpfade, z.B. mit

path(path,'c:\mypath\myprograms'); (Windows) bzw.

path(path,'c:/home/mypath/myprograms'); (Unix).

Gibt man ein Semikolon hinter dem Befehl an, so wird die Ausgabe einer Funktion nicht angezeigt. Lässt man es weg, so wird es angezeigt. Probieren Sie es mit *sqrt(12)* bzw. *sqrt(12);* aus. Man beachte, möglichst keine Funktionen mit gleichen Namen in verschiedenen Pfaden anzulegen, da ansonsten die erste gefundene Funktion verwendet wird.

Die Funktion *sqrt* können Sie nicht mit dem Editor editieren, da *sqrt* eine built-in Funktion ist, die compiliert vorliegt und deren Quellcode dem Benutzer nicht offen gelegt wird.

B.6 Funktionen und Skripte

B.6

Jede Funktion hat die folgende Form:

```
function a = Myfunction(b)
% Kommentar

<Programmtext>
```

```
% Ende
function a = Myfunction(b)
% mehrere Eingabe- und
Ausgabeparameter
% ...
```

Ein Skript hat die Form:

```
% Myscript
% Kommentar

<Programmtext>

% Ende
```

Wir empfehlen die Verwendung von Funktionen, da Funktionen lokale Variablenbereiche haben, während Skripte auf die globalen Variablen zurückgreifen. Variablen müssen nicht deklariert werden; es empfiehlt sich dennoch im Kommentar am Funktionsanfang verwendete Variablen aufzulisten, da der Kommentarkopf beim Aufruf von *help* angezeigt wird. Die Erweiterung des Dateinamens ist „*.m*“.

B.7 Datenstrukturen

Variablen werden bei der ersten Zuweisung automatisch deklariert, z.B.

```
a = 5;
b = 2.735;
c = 'String';
```

Konstanten sind Variablen, die nicht verändert werden. Man achte darauf, Datentypen nicht zu vermischen. Variablen sollten mit einem Default-Wert initialisiert werden. Der Speicherplatz kann explizit mit *clear* <*variable*> freigegeben werden. Neben den einfachen Datentypen, sind für uns insbesondere der Array und der Cell-Array interessant.

```
M1 = [];
M2 = [1 2 3 4];
M3 = [1, 2, 3, 4];
```

```
M4 = [1; 2; 3; 4];

M5 = [1 2 3 4; 5 6 7 8; 9 10 11 12]

M6 = M5';
M6(1,1) = 17; % setzt den ersten Eintrag auf 17
```

$M1$ ist ein leerer Array, $M2, M3$ die gleichen Zeilenarrays, $M4$ ein Spaltenarray und $M5$ eine Matrix. $M6$ ist $M5$ transponiert.

Aktivierungselement: Was ist der Unterschied zwischen *M2 * M2'* und *M2 .* M2*?

Der Cell-Array ist eine Struktur, die verschiedene Datentypen aufnehmen kann.

```
M1 = {};

M2{1}.a = 10;
M2{1}.b = [1 2 3];
M2{2}.a = 16;
M2{2}.b = [2 3 4];

M3 = M2'; % transponieren
```

$M1$ ist ein leerer Cell-Array. $M2$ ist ein Cell-Array mit zwei Einträgen und jeweils zwei Feldern.

B.8 Kontrollstrukturen

Die üblichen Kontrollfunktionen sind vorhanden. Eine Repeat-Until-Schleife muss als While-Schleife formuliert werden.

```
if ((a == 5) & (b = 7)) | ((b > 9) & ~(a == 8))
  f = 8;
end % if
```

```
switch a
  case 1
    f = 5;
  case 2
    f = 6;
  otherwise
    f = 8;
end % switch

while (a < 100)
  a = a + 1;
end % while

a=[];
for i = 1 : 10
  a = [a i];
end % for
```

B.9 Sonstige Funktionen / Demos

Wir empfehlen die Handbücher zu konsultieren, insbesondere um den Umfang an mathematischen Funktionen kennenzulernen. Einige Beispiele sind im Folgenden genannt. Matlab bietet z.B. Funktionen zur Linearen Algebra (Matrix- und Vektorrechnung, Eigenwerte), zur Numerik (Gleichungssysteme, Interpolation), zur statistischen Analyse (Korrelation, Regression) und zum Lösen von Differentialgleichungen (Anfangswertprobleme gewöhnlicher Differentialgleichungen). Darauf können wir an dieser Stelle nicht eingehen.

Mit dem Befehl

```
demos
```

erhalten Sie eine Fülle an Demonstrationen des Leistungsumfangs von Matlab.

```
a = sum([1 2 3 5 7]); % Summe = 18
b = find(a == 5); % gibt Index an
c = norm(a); % Euklidische Norm

A = [2 1; 3 4];
```

```
eig(A); % Eigenwerte

roots([1 3]); % Nullstellen von 1x^2+3

size(A) % Dimensionen der Matrix

mean(a); % Mittelwert
median(a); % Median
std(a); % Standardabweichung
min(a); % Minimum
max(a); % Maximum
```

Aktivierungselement: Probieren Sie den Befehl *convhull* aus.

Mit den Pfeiltasten können die jeweils letzten Befehle angezeigt werden.

B.10 Daten laden und abspeichern

B.10

Mit den Befehlen *load* und *save* können Daten direkt geladen und abgespeichert werden. Die Syntax und die Optionen schauen Sie sich am besten mit *help load* bzw. *help save* an. Mit den Funktionen wie *fscanf* und *fprintf* können Dateneinlese- und auslesemodule selbst geschrieben werden. Auch kann Text ausgegeben werden.

B.11 Datensammlungen

B.11

Zu einem Datensatz zur Klassifikation sollte man angeben: Anzahl der Klassen, Anzahl der Samples und die Anzahl der Dimensionen. Bei einem Datensatz zur Clusterung entfällt die Angabe der Klassen. Eine Methode sollte unabhängig von diesen Parametern funktionieren. Bei einer multiplen Sequenzanalyse hätte man die Parameter: Länge der Sequenzen, Anzahl der Sequenzen. Eine ganze Reihe weiterer Angaben sind denkbar.

Folgende (Benchmark-)Datensammlungen sind beispielsweise im Internet erhältlich (ohne Anspruch auf Vollständigkeit):

1. SWISS-Prot (protein-knowledgebase), Sequenzdaten:
http://www.expasy.org/sprot/

Verwendet man den Taxonomy Browser NEWT, so erhält man für Eukaryota / Fungi_Metazoa Group / Fungi / Microsporidae 13 SWISS-Prot entries. Die Zeile

KITH_ENCCU / O96720 / TK / Thymidine kinase (EC 2.7.1.21) / Encephalitozoon cuniculi / 212

liefert im FASTA-Format:

```
>sp|O96720|KITH_ENCCU Thymidine kinase (EC 2.7.1.21) -
             Encephalitozoon cuniculi. (noch erste Zeile)
MTRGTLNFVTSPMNAGKTANMLLRARHAATLGRRVLLAKPLSDTRHESSVIRSRCGIEMK
CDLCAGPEFSFTKDVLYGDVDILLVDEAQFLSSRQIDELREVADVHGIPVWCYGLLTDFK
KNLFEGSKVLVELCDKMIELDIVCYFCKADGRFHLKYANGKAVVEGPSIDISIPGDGKFV
AVCHMCWTEKTSTSEEVQDPRVLCAKVIPVDR
```

2. Verschiedene Datensätze, auch aus der Bioinformatik, findet man in:

http://www.kdnuggets.com/, speziell
http://www.kdnuggets.com/datasets/index.html,

3. Benchmark-Daten von Drosophila melanogaster DNA-Sequenzen:

http://www.fruitfly.org/sequence/drosophila-datasets.html

4. Strukturvorhersage:

http://genome.imim.es/main/databases.html

5. Drug Design (Datensätze zum Buch [ZG93]):

http://www2.chemie.uni-erlangen.de/publications/ANN-book/datasets/

Neben strukturierten Daten kann es interessant sein, Methoden anhand von Zufallsdaten auszuprobieren, um den Unterschied der Ergebnisse festzustellen oder den worst case zu erproben. Matlab stellt dazu einige Funktionen zur Verfügung:

```
rand('state',sum(100*clock)) % Initialisierung
                             % des Zufallsgenerators
a = rand(2,2) % 2x2-Matrix von gleichverteilten Zufallszahlen
b = randn(100,1) % 100-dim. Array mit normalverteilten
                 % Zufallszahlen
c = randperm(12) % 12-dim. Array (1 bis 12 zufällig permutiert)
```

B.12 Entwurf von Experimentierskripten

Im Folgenden wird ein einfacher und effektiver Aufbau einer Experimentierumgebung angegeben. Der Aufbau ist hierarchisch. Das Modul, das bei einer Analyse gestartet wird, nennen wir *Experiment*. Es kann z.B. mit einem selbsterklärenden Namen versehen werden, *Experiment_001_DatensatzXY*.

```
function Experiment
% Experimentierumgebung

% Pfade setzen, alternativ: alles in ein Verzeichnis bearbeiten
inputpath = ...;
outputpath = ...;
programpath = ...;

path = path(path,programpath);

% Daten laden
cd(inputpath);
data1 = load(...);
data2 = load(...);

% Parameterliste, alternativ dazu: Parameterdatei laden
param1 = ...;
param2 = ...;

% Initialisierungen
nrversuche = ...;
```

```
a = ...;
b = ...;

% Funktionen zur Datenvorverarbeitung aufrufen
data1clean = DataClean(data1, param1, a, ...);

% Benchmarkmodul aufrufen / Versuchswiederholungen
for i = 1 : nrversuche
  [result_i, ...] = BenchmarkNeuralNetwork(data1clean, param2,
  ...)
end % for

% Benchmarkergebnisse einzelner Versuche (statistisch) auswerten
%   und anzeigen
mean(...);
plot(...)

% Einstellungen und Ergebnisse abspeichern
save results results
save param param

% End Experimentierumgebung
function [...] = BenchmarkNeuralNetwork(...)
% Benchmarkmodul

...

% Analyse
[weights, ...] = NeuralNetwork(...);

% Performanzkriterien berechnen
sensitivity = ...;

% end BenchmarkNeuralNetwork
```

Der Vorteil einer solchen autarken Experimentierumgebung ist die Möglichkeit sie als Batch-Routine zu starten, z.B. auf einem Rechnercluster. Eine Variante wäre es, die Parameter, Pfade und Daten via einer GUI einzustellen bzw. zu laden. Dazu müsste man alle Parameter an eine Schnittstelle geben:

```
function Experiment(inputpath,...,data1,...,param1,...)
% Experimentierumgebung
...
```

Möchte man dies alles mit einer Benutzeroberfläche während der Verarbeitung steuern oder Ergebnisse anzeigen, so müssen GUI und Programm interagieren können. Matlab arbeitet hier Event-getrieben, z.B. ist ein Mausklick ein Event, bei dem man angeben muss, was darauf hin passieren soll. Man bedenke, dass die Entwicklung einer Programmoberfläche aufwändig ist und erst dann sinnvoll ist, wenn eine Spezifikation der Funktionalität feststeht.

B.13 Die Statistics-Toolbox

Zur Unterstützung der Forschung kann man klassische (wenngleich nicht unbedingt einfache) Methoden der Statistik verwenden. Die Statistics-Toolbox bietet dazu eine Reihe von Möglichkeiten an.

Merke: In der Datenanalyse können kleine Fehler große Auswirkungen haben. Deshalb sollte man sich um eine sorgfältige Arbeitsweise und um ein Verständnis der verwendeten Funktionen bemühen! Oder kurz gesagt: Sorgfalt geht vor Schnelligkeit! Oft wird beides verlangt.

Die Toolbox bietet u.a. Module zu folgenden Themen an: Verteilungen, deskriptive Statistik, Regression, Hypothesentests, Multivariate Statistik (u.a. PCA, Clusterung) und Hidden Markov-Modelle. Wenden Sie eine Funktion erst an, wenn Sie genau wissen, was diese tut. Wir geben einige einfache Beispiele und verweisen ansonsten auf das umfassende Handbuch:

```
x = [-5:0.05:5];
y = normpdf(x,0,1); % Dichte der N(0,1)-Normalverteilung
plot(x,y)

hist([1 2 2 3 3 3 4 4 4 4 5 5 5 6 6 7]) % Histogramm

a = [1 2 3 4 5 6 7 8 9 10];
b = [10 9 8 7 6 5.5 4.2 2.9 2 1];
corrcoef(a,b) % Korrelation
```

Aktivierungselement: Berechnen Sie *mean([1 2 3 NaN 5 6])*, *mean([1 2 3 4 5 6])* und *nanmean(1 2 3 NaN 5 6)*. NaN steht für „Not A Number". Welchen Sinn hat das?

Aktivierungselement: Berechnen Sie *qqplot(x,y)* mit den eben genannten x, y. Welche Aussage ergibt der Plot?

B.14

B.14 Die Neural Network-Toolbox

Die Toolbox eignet sich, um Neuronale Netze zu trainieren und zu testen. Neben den unten beschriebenen Netzen gibt es u.a. noch Perzeptronen und rekurrente Hopfield-Netze. Es können auch objektorientierte Bausteine zu eigenen Netzen zusammengefügt werden. Möchte man die Methoden unabhängig von der speziellen Toolbox (Stichwort: Portierbarkeit der Programme) implementieren, so kann es sinnvoll sein, nicht auf die speziellen Befehle der Toolbox zurückzugreifen.

Mit dem Befehl *nnd* startet man eine interaktive Demonstration.

Beispiel: Wir modellieren ein Netz mit drei Eingaben $x \in [0,1]$ und drei Gewichten $w_{1,i} = i$ $(i = 1, \ldots, 3)$ als Wx mit der Identität als Ausgabe und Bias $b = 0$. Wir berechnen die Ausgabe der Vektoren $(1,2,3)^T$ und $(4,5,6)^T$ (Ergebnis: $14 = 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3$ bzw. 32).

```
net = newlin([0,1;0,1;0,1],1);
net.IW{1,1} = [1 2 3];
net.b{1} = 0;

v =[1 4;2 5;3 6];
out = sim(net,v);
```

Mit *nntool* kann man interaktiv ein Netzwerk konstruieren. Z.B. können verschiedene Aktivierungsfunktionen eingestellt werden.

Das iterative Training übernimmt die Funktion *adapt*, der das Netz, die Trainingsdaten und die Klasseninformationen übergeben werden.

Ein Perzeptron kann mit *newp* erzeugt werden. Hier kann man die spezielle Lernfunktion *learnp* verwenden. *sim* und *learnp* ergeben *train*.

Mit der Funktion *newff* kann man ein Feedforward-Netz erzeugen. Das o.g. Prinzip des Netzwerk-Objekts wird beibehalten. In der Dokumentation zur Neural Network-Toolbox finden Sie eine Reihe von Schaubildern, die die Netzwerkarchitekturen verdeutlichen. Mit *sim* kann eine Ausgabe berechnet werden. Verschiedene Gradientenverfahren stehen zur Auswahl (z.B. *traingd*). Die Levenberg-Marquardt-Methode (*trainml*) ist ein schnelles Verfahren, bei dem die zweiten Ableitungen (Hesse-Matrix) durch die ersten Ableitungen angenähert wird. Verschiedene Vorverarbeitungsmöglichkeiten existieren, z.B. eine PCA. Insgesamt kann man die Backpropagation-Netzwerke gut für eine Klassifikationsanalyse verwenden. Durch *hintonw* wird ein Hinton-Diagramm für die Gewichte erzeugt.

Die Funktion *newrbe* erzeugt ein RBF-Netz, das auf den Trainingsdaten exakt ist (kein Trainingsfehler). *newrb* fügt nur so viele Neuronen wie nötig ein. *newgrnn* erzeugt ein generalisiertes Regressionsnetzwerk. Ein PNN kann mit *newpnn* erzeugt werden.

Zum Erlernen eines Kohonen-Netzwerks muss die Netztopologie festgelegt werden. Ein (8×8)-Gitter wird erzeugt durch

```
nodes = gridtop(8,8);
plotsom(nodes)
```

newsom erzeugt eine neue selbstorganisierende Karte. Mit *learnk* wird diese trainiert.

B.15 Graphical User Interface

B.15

Unter Matlab kann man mit *guide* ein Tool zur Erstellung einer GUI aufrufen. Buttons, Eingaben, Anzeigen, Menüauswahlen, u.a., sind möglich. Ein solches Objekt hat bestimmte Eigenschaften (properties), z.B. einen Namen oder eine Farbe. Mit dem Befehl *guidata* kann man Handles zu Objekten abspeichern. Soll bei einen Knopfdruck eine Funktion ausgeführt werden, so trägt man den Funktionsaufruf als *Callback* ein. In weiteren Verlauf kann ein Objekt über seinen Namen angesprochen, seine Eigenschaften abgefragt (*get*-Befehl) und verändert werden (*set*-Befehl).

B.16 Die Fuzzy-Toolbox

Mit dem Befehl *fuzzy* starten Sie eine GUI, mit der Sie Fuzzy-Systeme erstellen können. Beispielsweise gibt es einen Editor für Zugehörigkeitsfunktionen (*mfedit*) und einen Viewer für Regeln (*ruleview*). Das Neuro-Fuzzy-System kann auch von der Kommandozeile aus aufgerufen werden (*anfis*).

Desweiteren kann ein Fuzzy-k-means angewendet werden. Z.B. kann man mit dem Befehl *fcm(data,3)* nach drei Clustern in dem Datensatz *data* suchen.

B.17 Die Optimization-Toolbox

In dieser Toolbox gibt es Funktionen zur linearen und nichtlinearen Minimierung von Funktionen mit und ohne Nebenbedingungen und zum Lösen nichtlinearer Gleichungen. Die meisten Funktionen setzen ein Grundverständnis in numerischer Mathematik voraus.

Gegeben sei beispielsweise das folgende Problem:
Zielfunktion: $f(x) = 5x_1 + 15x_2 = Min\,!$.
Nebenbedingungen: 1) $10x_1 + 4x_2 = 20$,
2)$8x_1 + 12x_2 \leq 30$,
3)$x_1 \geq 0$,
4)$x_2 \geq 0$. Wir geben die Berechnung an. Dabei seien die Matlab-Variablen gegeben durch $\min f^T \cdot x$, $Aeq \cdot x = beq$, $A \cdot x \leq x$ und $x \geq 0$ (lower bounds).

```
f = [5 15];
A = [10 4];
b = [20];
Aeq = [8 12];
beq = [30];
lb = [0 0];

[x, fval, exitflag, output] = linprog(f, A, b, Aeq, beq, lb, ...,
[], [], optimset('Display','iter'));
```

In unserem kleinen Experiment erhielten wir als Lösung $x^T = (1.3636, 1.5909)$ nach 6 Iterationen.

Aktivierungselement: Schreiben Sie ein Programm, das mittels einer evolutionären Strategie die Lösung zum gleichen Problem optimieren kann.

B.18 Die Genetic Algorithm- and Direct Search-Toolbox B.18

Wie wir im Laufe des Kurses gelernt haben, können schwierige Optimierungsprobleme besser durch eine stochastische Suche oder durch genetische Algorithmen gelöst werden (falls z.B. keine Gradienteninformationen vorhanden sind). Die Genetic Algorithm and Direct Search-Toolbox trägt dem Rechnung und erweitert die Optimization-Toolbox. Mit dem Befehl *gatool* ruft man eine GUI auf, in der man die Bestandteile eines genetischen Algorithmus konfigurieren kann, also die Fitness-Funktion, die Operatoren und deren Parameter. Die Optimierung kann auch von der Befehlszeile aus gestartet werden mit *ga*. Die Fitness-Funktion kann angezeigt werden. *psearchtool* öffnet eine GUI zur stochastischen Suche, die auch von der Befehlszeile aus mit *patternsearch* durchgeführt werden kann.

B.19 Die Bioinformatics-Toolbox und SimBiology B.19

Die Bioinformatics-Toolbox enthält spezielle Analyse- und Darstellungsfunktionen für Daten aus dem Bereich der Bioinformatik.

Verschiedene Sequenzdatenbanken sind direkt abrufbar, z.B. GenBank mit *getgenbank*, Protein Sequence Database PIR-PSD (*getpir*). Für Sequenz- und Genexpressionsdaten gibt es Datentypen und Funktionen, z.B. für das FASTA-Format *fastaread* und *fastawrite*.

Weist man einer Variablen eine Sequenz zu, *sequenz* = *acgcat...*;, so werden Informationen via *whos sequenz* angezeigt. *ntdensity(sequenz)* zeigt die Anteile der Nukleotide über die Länge der Sequenz an. *basecount(sequenz)* zählt die Nukleotide; entsprechend gibt es die Befehle *dimercount* und *codoncount*. *showorfs(sequenz)* bestimmt open reading frames. Bei der Eingabe von *aminolookup('letter',nt2aa('ATC'))* erhält man „Ile isoleucine". Es gibt eine Reihe von Visualisierungsmöglichkeiten für Aminosäuren durch *aacount*. Die Atomarten können gezählt werden (*atomiccomp*) und das Molekulargewicht bestimmt werden (*molweight*).

Ein (globaler) Alignment-Plot zweier Sequenzen kann mit *seqdotplot* angezeigt werden. Mit *nwalign(sequenz1, sequenz2)* wird der Needleman-Wunsch-Algorithmus aufgerufen. Mit *swalign(sequenz1, sequenz2)* erhält man ein (lokales) Alignment.

Genexpressionsdatenobjekte haben Felder, z.B. *.Names* für die Gennamen und *.ColumnNames* für die Markernamen. Daten können mit speziellen Funktionen geplottet werden. Cluster- und PCA-Funktionen sind vorhanden, da die Statistics-Toolbox vorher installiert sein muss. *dayhoff* liefert eine Dayhoff-Scoring-Matrix. Verschiedene Demos können aufgerufen werden, z.B. *aligndemo* (Sequenzanalyse) oder *hmmprofdemo* (Gebrauch von Hidden Markov Modellen).

Mit der SimBiology-Ergänzung können biochemische Pfade modelliert und simuliert werden. Mit dem Befehl *sbiodesktop* wird eine Benutzeroberfläche geöffnet.

B.20 Der Compiler

Zum Abschluss gehen wir auf den Compiler ein, der uns hilft (langsamere) Matlab .m-files in .c-files und schließlich in ausführbare (schnellere) Dateien zu übersetzen, die man ohne Matlab starten kann (unter Windows beispielsweise DLL-Dateien).

Mit dem Befehl *mcc -x filename* wird ein .m-file in ein .c-file umgewandelt und ein .mex-file erzeugt, das das .c-file einbettet. Aus file.m entstehen file.c, file.h, file.m, file_mex.c und file.dll (mex-file für Windows). Damit wird der Quellcode verborgen.

Mit dem Befehl *mcc -m filename* wird ein ohne Matlab ausführbares .m-file in ein .exe-file umgewandelt. Aus file.m entstehen file.c, file.h, file.m, file_main.c und file.exe. Der Matlab-Compiler kann einige „exotische“ und vom Betriebssystem abhängige Befehle nicht übersetzen.

Mit dem Befehl *mex-setup* können weitere auf dem Computer installierte C-Compiler eingestellt werden. Für diese Prozedur wird auf das Handbuch verwiesen. Ein sog. options-file muss dafür konfiguriert werden. Man achte auf das Setzen entsprechender Library-Pfade.

Der Befehl *mcc -O all* (großes O) schaltet alle Compiler-Optimierungen ein, *mcc -O list* listet die Optimierungsoptionen auf.

B.21 Die Database-Toolbox

B.21

Für die direkte Benutzung von Datenbanken unter Matlab können Daten (via ODBC, JDBC) eingebunden werden. Es gibt ein visuelles Tool (*querybuilder*) zur Generierung von Abfragen. Man kann auch via dem Befehl *database* einen Identifier *id* für eine Datenbank erhalten und dann via *curs = exec(id,'select ... from ... where ...');* direkt SQL-Befehle ausführen. *curs* ist ein Objekt; mit *curs.Data* erhält man Zugriff auf die Daten, die auch in der Datenbank updated werden können.

B.22 Die Web Server-Toolbox

B.22

Hier werden Möglichkeiten geschaffen Eingaben und Ausgaben via HTML im Internet zu ermöglichen. Dazu wird HTML-Code in ein Matlab-Template eingebettet. *matlabserver* läuft im Hintergrund. Dieser muss in der Datei *matlabserver.conf* konfiguriert werden.

Ein professionelles Arbeiten mit den Matlab-Toolboxen setzt das Studium der Handbücher voraus. Wir hoffen Ihnen mit dieser kleinen Übersicht diesen Einstieg zu erleichtern.

Zum Abschluss sei darauf hingewiesen, dass das Arbeiten mit nicht lizenzierter Software verboten ist. Die Haftung für Schäden, die durch unbeabsichtigte Fehler in diesem Skript entstehen, wird ausgeschlossen.

Literaturverzeichnis

[ABA00] R. Aliev, K.W. Bonfig und F. Aliew. *Soft Computing: Eine grundlegende Einführung.* Verlag Technik, 2000.

[AC02] D.K. Agrafiotis und W. Cedeno. Feature Selection for Structure-Activity Correlation Using Binary Particle Swarms. *J. Med. Chem.*, 45:1098–1107, 2002.

[Adl90] K.-P. Adlassnig. Update on CADIAG-2: A Fuzzy Medical Expert System for General Internal Medicine. In: W.H. Janko, M. Roubens und H.-J. Zimmermann (Hrsg.), Progress in Fuzzy Sets and Systems. Kluwer Academic Publishers. Seiten 1–6, 1990.

[Adl94] L.M. Adleman. Molecular Computation of Solutions to Combinatorial Problems. *Science*, 266(1021-1024):1021–1024, 1994.

[AI03] S. Ando und H. Iba. Artificial Immune System for Classification of Gene Expression Data. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 1926–1937, 2003.

[AkM02] B. Al-kazemi und C.K. Mohan. Training Feedforward Neural Networks Using Multi-Phase Particle Swarm Optimization. *Proc. of the 9^{th} Int. Conf. on Neural Information Processing (ICONIP), Singapur*, Seiten 2615–2619, 2002.

[AS94] R. Agrawal und R. Skrikant. Fast Algorithms for Mining Association Rules. *Proc. of the 20^{th} Int. Conf. on Very Large Databases (VLDB), Santiago de Chile, Chile*, Seiten 487–499, 1994.

[AS95] R. Agrawal und R. Skrikant. Mining Sequential Patterns. *Proc. of the 11^{th} Int. Conf. on Data Engineering (ICDE), Taipei, Taiwan*, Seiten 3–14, 1995.

[ASI90a] T. Aoyama, Y. Suzuki und H. Ichikawa. Neural Networks Applied to Quantitative Structure-Activity Relationship (QSAR) Analysis. *J. Med. Chem.*, 33:2583–2590, 1990.

[ASI90b] T. Aoyama, Y. Suzuki und H. Ichikawa. Neural Networks Applied to Quantitative Structure-Activity Relationships. *J. Med. Chem.*, 33:905–908, 1990.

[Azu00] F. Azuaje. Making Genome Expression Data Meaningful: Prediction and Discovery of Classes of Cancer Through a Connectionist Learning Approach. *IEEE Int. Conf. on Bioinformatics and Biomedical Egineering*, Seiten 208–213, 2000.

[Bac96] R. Bacher. *Clusteranalyse.* Oldenbourg-Verlag, 2. Aufl., 1996.

[Bau96] H.-U. Bauer. Controling the Magnification Factor of Self-Organizing Feature Maps. *Neural Computation*, 8:757–771, 1996.

[BB02] C. Borgelt und M.R. Berthold. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. *Proc. of the 2^{nd} IEEE Int. Conf. on Data Mining (ICDM), Maebashi City, Japan*, Seiten 51–58, 2002.

[BBF+99] P. Baldi, S. Brunak, P. Frasconi, G. Soda und G. Pollastri. Exploiting the Past and the Future in Protein Secondary Structure Prediction. *Bioinformatics*, 15(11):937–946, 1999.

[BDT99] E. Bonabeau, M. Dorigo und G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, 1999.

[Ber97] M.R. Berthold. *Konstruktives Training von Probabilistischen Neuronalen Netzen für die Musterklassifikation.* Infix-Verlag, 1997. Zugl. Dissertation Univ. Karlsruhe.

[Ber03] M.R. Berthold. Mixed Fuzzy Rule Formation. *Int. J. of Approx. Reasoning*, 32:67–84, 2003.

[BFSS03] E. Byvatov, U. Fechner, J. Sadowski und G. Schneider. Comparison of Support Vector Machine and Artificial Neural Network Systems for Drug/Nondrug Classification. *J. Chem. Inf. Comput. Sci.*, 43(6):1882–1889, 2003.

[BH95] M.R. Berthold und K.-P. Huber. From Radial to Rectangular Basis Functions: A New Approach for Rule Learning from Large Datasets. Internal Report 15-95, Univ. Karlsruhe, 1995.

[BH98] M.R. Berthold und K.-P. Huber. Missing Values and Learning of Fuzzy Rules. *Int. Journ. of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):171–178, 1998.

[BH99] M.R. Berthold und D.J. Hand (Hrsg.). *Intelligent Data Analysis: An Introduction.* Springer-Verlag, 1999.

[Bis95] C.M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, 1995.

[Bis98] C.M. Bishop. *Pulsed Neural Networks.* MIT Press, 1998.

[BKI00] C. Beierle und G. Kern-Isberner. *Methoden wissensbasierter Systeme.* Vieweg-Verlag, 2000.

[BKK96] H.-J. Boehm, G. Klebe und H. Kubinyi. *Wirkstoffdesign.* Spektrum Verlag, 1996.

[Bla98] E. Blanzieri. *Learning Algorithms for Radial Basis Function Networks: Synthesis, Experiments and Cognitive Modelling.* Dissertation, Univ. and Polytechnic of Turin, Italien, 1998.

[BLH99] R. Brause, T. Langsdorf und M. Hepp. Neural Data Mining for Credit Card Fraud Detection. *Proc. of the 11th IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI), Chicago, IL, USA*, Seiten 103–106, 1999. Ausführliche Version erschienen als Interner Bericht 7/99, FB Informatik, Univ. Frankfurt am Main.

[BLK+96] K. Boegl, H. Leitich, G. Kolousek, T. Rothenfluh und K.-P Adlassnig. Clinical Data Interpretation in MEDFRAME/CADIAG-4 Using Fuzzy Sets. *Biomedical Engineering-Applications, Basis & Communications*, 8:488–495, 1996.

[BMP+00] M.A. Bedau, J.S. McCaskill, N.H. Packard u.a. Open Problems in Artificial Life. *Artificial Life*, 6:363–376, 2000.

[Bol96] T. Bollinger. Assoziationsregeln - Analyse eines Data Mining Verfahrens. *Informatik-Spektrum*, 19:257–261, 1996.

[Bos91] K. Bosch. *Elementare Einführung in die Wahrscheinlichkeitsrechnung.* Vieweg-Verlag, 5. durchges. Aufl., 1991.

[Bos92] K. Bosch. *Elementare Einführung in die angewandte Statistik.* Vieweg-Verlag, 4. durchges. Aufl., 1992.

[Bot99] H.-H. Bothe. *Neuro-Fuzzy-Methoden. Einführung in Theorie und Anwendungen.* Springer-Verlag, 1999.

[BP01] C. Blake und W. Pratt. A Semantic Approach to Selecting Features from Text. *Proc. of the 1st Int. Conf. on Data Mining (ICDM), San Jose, CA, USA*, Seiten 59–66, 2001.

[BPS03] A. Baresel, H. Pohlheim und S. Sadeghipour. Structural and Functional Sequence Test of Dynamic and State-Based Software with Evolutionary Algorithms. *Proc. of the 5th Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 2428–2441, 2003.

[Bra95] R. Brause. *Neuronale Netze.* Teubner-Verlag, 2. überarb. und erw. Aufl., 1995.

[Bra03] R. Brause. Adaptive Modeling of Biochemical Pathways. *IEEE 15th Int. Conf on Tools with Art. Intell. (ICTAI), Washington, DC, USA*, Seiten 62–68, 2003.

[BS00] H.-J. Böhm und G. Schneider (Hrsg.). *Virtual Screening for Bioactive Molecules.* Wiley-VCH Verlag, 2000.

[BTP+00] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme und L. Lakhal. Mining Minimal Non-Redundant Association Rules Using Frequent Closed Itemsets. *Proc. of the 1st Int. Conf. on Computational Logic (CL); subconference: 6th Conf. on Database Systems (DOOD), London, UK*, Seiten 972–986, 2000.

[BWP05] M.R. Berthold, B. Wiswedel und D.E. Patterson. Interactive Exploration of Fuzzy Clusters Using Neighborgrams. *Fuzzy Sets and Systems*, 149(1):21–37, 2005.

[BYSZ02] W. Bin, Z. Yi, L. Shaohui und S. Zhongzhi. CSIM: A Document Clustering Algorithm Based on Swarm Intelligence. *Proc. of the Congress on Evolutionary Computation (CEC), Honolulu, HA, USA*, Seiten 477–482, 2002.

[Car02] H.M. Cartwright. Investigation of Structure - Biodegradability Relationships in Polychlorinated Biphenyls Using Self-Organising Maps. *Neural Computing and Applications*, 11:30–36, 2002.

[Cas03] L.N. de Castro. Artificial Immune Systems as Novel Soft Computing Paradigm. *Soft Computing*, 7(8):526–544, 2003.

[CB94] A. K. Chan und G. A. Béau. Nonlinear System Identification Using RBF Networks with a Moving Center Lattice. *ICANN '94*, 2, 1994.

[CB02] L. Chen und L. Boggess. Neural Networks for Genome Signature Analysis. *Proc. of the 9th Int. Conf. on Neural Information Processing (ICONIP), Singapur*, Seiten 1554–1558, 2002.

[CGR91] G.A. Carpenter, S. Grossberg und D.B. Rosen. Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System. *Neural Networks*, 4(6):759–771, 1991.

[CH02] B.C.H. Chang und S.K. Halgamuge. Protein Motif Extraction with Neuro-Fuzzy Optimization. *Bioinformatics*, 18(8):1084–1090, 2002.

[CH03] C. Creighton und S. Hanash. Mining Gene Expresion Databases for Association Rules. *Bioinformatics*, 19(1):79–86, 2003.

[CLM00] J.-H. Chen, S.-Y. Le und J.V. Maizel. Prediction of Common Secondary Structures of RNAs: A Genetic Algorithm Approach. *Nucleic Acids Research*, 28(4):991–999, 2000.

[CLXC02] Y.-D. Cai, X.-J. Liu, X.-B. Xu und K.-C. Chou. Prediction of Protein Structural Classes by Support Vector Machines. *Computers and Chemistry*, 26(3):293–296, 2002.

[CMMR04] J. Casasnovas, J. Miro, M. Moya und F. Rossello. An Approach to Membrane Computing Under Inexactitude. *Int. J. of Foundations of Computer Science*, 15(6):841–864, 2004.

[CMS03] A.P. Cootes, S.H. Muggleton und M.J.E. Sternberg. Title: The Automatic Discovery of Structural Principles Describing Protein Fold Space. *Journal of Molecular Biology*, 330(4):839–850, 2003.

[CN89] P. Clark und R. Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3:261–284, 1989.

[CPG03] E. Cantu-Paz und D.E. Goldberg. Are Multiple Runs of Genetic Algorithms Better than One. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 801–812, 2003.

[CR03] N.A. Campbell und J.B. Reece. *Biologie*. Spektrum Verlag, 6. Aufl., 2003.

[CRLB03] C. Carleos, F. Rodriguez, H. Lamelas und J.A. Baro. Simulating Complex Traits Influenced by Genes with Fuzzy-Valued Effects in Predigreed Populations. *Bioinformatics*, 19(1):144–148, 2003.

[CS03] H.M. Cartwright und L.M. Sztandera (Hrsg.). *Soft Computing Approaches in Chemistry*. Springer-Verlag, 2003. Studies in Fuzziness and Soft Computing, Band 120.

[CT91] T.M. Cover und J.A. Thomas. *Elements of Information Theory*. Wiley & Sons, 1991.

[CT02] L.N. de Castro und J. Timmis. *Artificial Immune Systems: A New Computational Intelligence Paradigm*. Springer-Verlag, 2002.

[CW03] E. Costello und D.C. Wilson. A Case-Based Approach to Gene Finding. *Proc of the 5^{th} Int. Conf. on Case-Based Reasoning, Trondheim, Norwegen*, 2003.

[CZ00a] G. Cheng und A. Zell. Externally Growing Cell Structures for Pattern Classification. *Proc. of the 2^{nd} ICSC Symp. on Neural Computation (NC), Berlin*, Seiten 233–239, 2000.

[CZ00b] I. Cloete und J.M. Zurada (Hrsg.). *Knowledge-Based Neurocomputing.* MIT Press, 2000.

[CZ02] L.N. de Castro und F.J. Von Zuben. Automatic Determination of Radial Basis Functions: An Immunity-Based Approach. *Int. Journal of Neural Systems,* 11(6):523–535, 2002.

[Das98] D. Dasgupta (Hrsg.). *Artificial Immune Systems and Their Applications.* Springer-Verlag, 1998.

[Deu03] J.M. Deutsch. Evolutionary Algorithms for Finding Optimal Gene Sets in Microarray Prediction. *Bioinformatics,* 19(1):45–52, 2003.

[DHR93] D. Driankov, H. Hellendoorn und M. Reinfrank. *An Introduction to Fuzzy-Control.* Springer-Verlag, 1993.

[DMC91] M. Dorigo, V. Maniezzo und A. Colorni. Ant System: An Autocatalytic Optimizing Process. Technical Report No. 91-016, Politecnico di Milano, Italien, 1991.

[DMC96] M. Dorigo, V. Maniezzo und A. Colorni. Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B,* 26(1):1–13, 1996.

[DS02] M. Dugas und K. Schmidt. *Medizinische Informatik und Bioinformatik.* Springer-Verlag, 2002.

[DS04] M. Dorigo und T. Stützle. *Ant Colony Optimization.* Bradford Book, 2004.

[DSO78] M.O. Dayhoff, R.M. Schwartz und B.C. Orcutt. A Model of Evolutionary Change in Proteins. In: Atlas of Protein Sequence and Structure National Biomedical Research Foundation, Band 5, Anhang 3, M.O. Dayhoff (Hrsg.), 345-352, 1978.

[DZB01] P. Dittrich, J. Ziegler und W. Banzhaf. Artificial Chemistries - A Review. *Artificial Life,* 7:225–275, 2001.

[EK95] R.C. Eberhart und J. Kennedy. A New Optimizer Using Particle Swarm Theory. *Proc. of the 6th Int. Symp. on Micro Machine and Human Science, Nagoya, Japan,* Seiten 39–43, 1995.

[EKB02a] T.E. Exner, M. Keil und J. Brickmann. Pattern Recognition Strategies for Molecular Surfaces. I. Pattern Generation Using Fuzzy Set Theory. *J. Comput. Chem.,* 23:1176–1187, 2002.

[EKB02b] T.E. Exner, M. Keil und J. Brickmann. Pattern Recognition Strategies for Molecular Surfaces. II. Surface Complementarity. *J. Comput. Chem.,* 23:1188–1197, 2002.

[EYA+01] G. Espinosa, D. Yaffe, A. Arenas, Y. Chen und F. Giralt. A Fuzzy ARTMAP-Based Quantitative Structure-Property Relationship (QSPR) for Predicting Physical Properties of Organic Compounds. *Ind. Eng. Chem. Res.,* 40:2757–2766, 2001.

[FC03] G.B. Fogel und D.W. Corne (Hrsg.). *Evolutionary Computation in Bioinformatics.* Morgan Kaufmann Publishers, 2003.

[FFR+03] U. Fechner, L. Franke, L. Renner, P. Schneider und G. Schneider. Comparison of Correlation Vector Methods for Ligand-based Similarity Searching. *Journal of Computer Aided Molecular Design*, 17:687–698, 2003.

[Fri92] B. Fritzke. *Wachsende Zellstrukturen - ein selbstorganisierendes neuronales Netzwerk.* Dissertation, Univ. Erlangen-Nürnberg, 1992.

[Fri94] B. Fritzke. Fast Learning with Incremental RBF Networks. *Neural Processing Letters*, 1(1):2–5, 1994.

[Fri97] B. Fritzke. A Self-organizing Network That Can Follow Non-Stationary Distributions. *Proc. of the 7th Int. Conf. on Artificial Neural Networks (ICANN), Lausanne, Schweiz*, Seiten 613–618, 1997.

[Fri98] B. Fritzke. *Vektorbasierte Neuronale Netze.* Shaker-Verlag, 1998. Zugl. Habilitation Univ. Dresden.

[GB03] T. Gabriel und M.R. Berthold. Constructing Hierarchical Rule Systems. *Proc. of Advances in Intelligent Data Analysis V (IDA), Berlin*, Seiten 76–87, 2003. LNCS Vol. 2810.

[GB04] T. Gabriel und M.R. Berthold. Influence of Fuzzy Norms and Other Heuristics on „Mixed Fuzzy Rule Formation". *Int. J. of Approximate Reasoning*, 35:195–202, 2004.

[GDWS03] A. Givehchi, A. Dietrich, P. Wrede und G. Schneider. ChemSpaceShuttle: A Tool for Data Mining in Drug Discovery by Classification, Projection, and 3D Visualization. *QSAR Comb. Sci.*, 22:549–559, 2003.

[GE03a] J. Gasteiger und T. Engel (Hrsg.). *Cheminformatics: A Textbook.* Wiley-VCH, 2003.

[GE03b] J. Gilbert und M. Eppstein. A Case for Codons in Evolutionary Algorithms. *Proc. of the 5th Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 967–978, 2003.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, 1989.

[Goo98] G. Goos. *Vorlesungen über Informatik Band 4: Paralleles Rechnen und nicht-analytische Lösungsverfahren.* Springer-Verlag, 1998.

[Gro76] S. Grossberg. Adaptive Pattern Classification and Universal Recoding: I. Parallel Development and Coding of Neural Feature Detectors. *Biol. Cybern.*, 23:121–134, 1976.

[GU99] G. Guimarães und A. Ultsch. A Method for Temporal Knowledge Conversion. *Proc. of the 3rd Symp. on Intelligent Data Analysis (IDA), Amsterdam, Niederlande*, Seiten 369–380, 1999.

[Gui98] G. Guimarães. *Eine Methode zur Entdeckung von komplexen Mustern in Zeitreihen mit Neuronalen Netzen und deren Überführung in eine symbolische Wissensrepräsentation.* Dissertation, Univ. Marburg, 1998.

[Ham01] F. Hamker. Lifelong Learning Cell Structures – Continously Learning Without Catastrophic Inference. *Neural Networks*, 14:551–573, 2001.

[Han01] A. Hansen. *Bioinformatik.* Birkhäuser, 2001.

[Hay99] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 2. Aufl., 1999.

[HB95] K.-P. Huber und M.R. Berthold. Building Precise Classifiers With Automatic Rule Extraction. *Proc. of the IEEE Int. Conf. on Neural Networks (ICNN), Perth, Australien*, 3:1263–1268, 1995.

[HB03] D. Howard und K. Benson. Evolutionary Computation Method for Promoter Site Prediction in DNA. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 1690–1701, 2003.

[HBB03] H. Hofer, C. Borgelt und M. Berthold. Large Scale Mining of Molecular Fragments with Wildcards. *Proc. of Advances in Intelligent Data Analysis V (IDA), Berlin*, Seiten 380–389, 2003. LNCS Bd. 2810, Springer-Verlag.

[Hel91] H. Helbig. *Künstliche Intelligenz und automatische Wissensverarbeitung.* Verlag Technik Berlin, 1991. Mittlerweile erschienen: 2. stark bearb. Aufl. 1996.

[HG93] C.M. Higgins und M. Goodman. Learning Fuzzy Rule-Based Neural Networks for Control. *Advances in Neural Information Processing Systems (NIPS), Denver, CO, USA*, 5:350–357, 1993.

[HH99] R.J. Hilderman und H.J. Hamilton. Heuristic Measures of Interestingness. *Proc. of the 3^{rd} European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD), Prag, Tschechische Republik*, Seiten 232–241, 1999.

[HH03] D.H. Hong und C. Hwang. Support Vector Fuzzy Regression Machines. *Fuzzy Sets and Systems*, 2:271–281, 2003.

[HIH03] J.A. Hering, P.R. Innocent und P.I. Haris. Neuro-Fuzzy Structural Classification of Proteins for Improved Protein Secondary Structure Prediction. *Proteomics*, 3:1464–1475, 2003.

[HJ01a] R.J. Howlett und L.C. Jain (Hrsg.). *Radial Basis Function Networks 1: Recent Developments in Theory and Applications.* Springer-Verlag, 2001.

[HJ01b] R.J. Howlett und L.C. Jain (Hrsg.). *Radial Basis Function Networks 2: New Advances in Design.* Springer-Verlag, 2001.

[HK00] J. Han und M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufman Publishers, 2000.

[HKKN00] B.T. Hoffmann, T. Kopaijtic, J.L. Katz und A.H. Newman. 2D QSAR Modeling and Preliminary Database Searching for Dopamine Transporter Inhibitors Using Genetic Algorithm Variable Selection of Molconn Z Descriptors. *J. Med. Chem.*, 43:4151–4159, 2000.

[HKM+96] K. Hätönen, M. Klemettinen, H. Mannila, P. Ronkainen und H. Toivonen. Knowledge Discovery from Telecommunication Network Alarm Databases. *12^{th} International Conference on Data Engineering (ICDE), New Orleans, LO, USA*, Seiten 115–122, 1996.

[HLK03] T.R. Hvidsten, A. Laegreid und J. Komorowski. Learning Rule-Based Models of Biological Process From Gene Expression Time Profiles Using Gene Ontology. *Bioinformatics*, 19(9):1116–1123, 2003.

[Hol92] J.H. Holland. *Adaption in Natural and Artificial Systems.* MIT Press, 1992.

[Höp01] F. Höppner. Discovery of Temporal Patterns – Learning Rules About The Qualitative Behaviour of Time Series. *Proc. of the 5^{th} European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD), Freiburg*, Seiten 192–203, 2001.

[HOST03] D. Hang, C. Ofria, T.M. Schmidt und E. Torng. The Effect of Natural Selection on Phylogeny Reconstruction Algorithms. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 13–24, 2003.

[HPJ02] R. Hoar, J. Penner und C. Jacob. Evolutionary Swarm Traffic: If Ant Roads Had Traffic Lights. *Proc. of the Congress on Evolutionary Computation (CEC), Honolulu, HA, USA*, Seiten 1910–1915, 2002.

[HPY00] J. Han, J. Pei und Y. Yin. Mining Frequent Patterns Without Candidate Generation. *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, Dallas, TX, USA*, Seiten 1–12, 2000.

[HS92] J.D. Hirst und M.J. Sternberg. Prediction of Structural and Functional Features of Protein and Nucleic Acid Sequences by Artificial Neural Networks. *Biochemistry*, 31:7211–7218, 1992.

[HS01] S. Hua und Z. Sun. A Novel Method of Protein Secondary Structure Prediction with High Segment Overlap Measure: Support Vector Machine Approach. *Journal of Molecular Biology*, 302:397–407, 2001.

[HS02] R. Hofestädt und R. Schnee. *Studien- und Forschungsführer Bioinformatik.* Spektrum, 2002.

[Hub98] K.-P. Huber. *Datenbasierte Metamodellierung mit automatisch erzeugten Fuzzy-Regeln.* VDI-Verlag, 1998. Fortschritt-Berichte / VDI: Reihe 10, Nr. 551. Zugl. Diss., Univ. Karlsruhe.

[Hut02] T.J. Hutton. Evolvable Self-Replicating Molecules in an Artificial Chemistry. *Artificial Life*, 8:341–356, 2002.

[HV02] B. Hammer und T. Villmann. Generalized Relevance Learning Vector Quantization. *Neural Networks*, 15:1059–1068, 2002.

[HW94] B. Hölldobler und E.O. Wilson (Hrsg.). *Ameisen: Die Entdeckung einer faszinierenden Welt.* Birkhäuser Verlag, 1994.

[IA01] S. Izrailev und D. Agrafiotis. A Novel Method for Building Regression Tree Models for QSAR Based on Artificial Ant Colony Systems. *J. Chem. Inf. Comput. Sci.*, 41:176–180, 2001.

[ICO+01] T. Ito, T. Chiba, R. Ozawa, M. Yoshida, M. Hattori und Y. Sakaki. A Comprehensive Two-hybrid Analysis to Explore the Yeast Protein Interactome. *Proc. Natl Acad. Sci. USA*, 98:4569–4574, 2001.

[IKW01] B. Izyumov, E. Kalinina und M. Wagenknecht. Software Tools for Regression Analysis of Fuzzy Data. *Proc. of the 9th Zittau Fuzzy Colloquium, Zittau*, Seiten 221–229, 2001.

[Jan93] J.-S.R. Jang. ANFIS: Adaptive-Network-Based Fuzzy Inference System. *IEEE Trans. on Systems, Man and Cybernetics*, 23:665–685, 1993.

[JS93] J.-S.R. Jang und C.-T. Sun. Functional Equivalence Between Radial Basis Function Networks and Fuzzy Inference Systems. *IEEE Transactions on Neural Networks*, 4(1):156–159, 1993.

[JSYY02] L. Jianyu, L. Siwei, Q. Yingjian und H. Yaping. Numerical Solution of Differential Equations by Radial Basis Function Neural Networks. *Proc. of the Int. Joint Conf. on Neural Networks (IJCNN), Honolulu, HA, USA*, Seiten 773–777, 2002.

[Kas93] T. Kasuba. Simplified Fuzzy ARTMAP. *AI Expert*, 8(11):18–25, 1993.

[Kas01] S. Kaski. SOM-based Exploratory Analysis of Gene Expression Data. *Proc. of Advances in Self-Organising Maps (WSOM 2001), Lincoln, UK*, Seiten 124–131, 2001.

[KBIAK99] J.R. Koza, F.H. Bennett III, D. Andre und M.A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, 1999.

[KE01] J. Kennedy und R.C. Eberhart. *Swarm Intelligence.* Morgan Kaufmann Publishers, 2001.

[Kel98] A. Kel. A Genetic Algorithm for Designing Gene Family-Specific Oligonucleotide Sets Used for Hybridization. *Bioinformatics*, 14(3):259–270, 1998.

[Kel00] H.B. Keller. *Maschinelle Intelligenz.* Vieweg-Verlag, 2000.

[KGK93] R. Kruse, J. Gerhardt und F. Klawonn. *Fuzzy-Systeme.* Teubner-Verlag, 1993.

[Kir01] W. Kirchner (Hrsg.). *Die Ameisen.* Verlag C.H. Beck, 2001.

[KK02] W. Kinz und I. Kanter. Neural Cryptography. *Proc. of the 9^{th} Int. Conf. on Neural Information Processing (ICONIP), Singapur*, Seiten 1351–1354, 2002.

[KKL+00] T. Kohonen, S. Kaski, K. Lagus, J. Salojärvi, V. Paatero und A. Saarela. Self Organization of a Massive Document Collection. *IEEE Transactions on Neural Networks*, 11(3):574–585, 2000.

[KKS+03] J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu und G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, 2003.

[KLM03] Y.-H. Kim, S.-K. Lee und B.-R. Moon. Optimizing the Order of Taxon Addition in Phylogenetic Tree Construction Using Genetic Algorithm. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 2215–2226, 2003.

[KM03] J.-H. Kim und B.-R. Moon. New Usage of SOM for Genetic Algorithms. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 1100–1111, 2003.

[KML+01] J.R. Koza, W. Mydlowec, G. Lanza, J. Yu und M.A. Keane. Reverse Engineering of Metabolic Pathways from Observed Data using Genetic Programming. *Proc. of the Pacific Symp. on Biocomputing, Mauna Lani, HA, USA*, Seiten 434–445, 2001.

[Koh82] T. Kohonen. Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43:59–69, 1982.

[Koh97] T. Kohonen. *Self-Organizing Maps.* Springer-Verlag, 2. Aufl., 1997. Mittlerweile erschienen: 3. Aufl. 2001.

[Kol93] J. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann Publishers, 1993.

[Kos97] B. Kosko. *Fuzzy Engineering.* Prentice Hall, 1997.

[Koz92] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[Koz94] J.R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[KSD05] O. Karpenko, J. Shi und Y. Dai. Prediction of MHC Class II Binders Using the Ant Colony Search Strategy. *Artificial Intelligence in Medicine*, 2005. Special Issue on Computational Intelligence Techniques in Bioinformatics. Im Druck.

[KW03] V.G. Krishnan und D.R. Westhead. A Comparative Study of Machine-Learning Methods to Predict the Effects of Single Nucleotide Polymorphisms on Protein Function. *Bioinformatics*, 19(17):2199–2209, 2003.

[Lan95] C.G. Langton (Hrsg.). *Artificial Life: An Overview.* MIT Press, 1995.

[Les02] A.M. Lesk. *Bioinformatik: Eine Einführung.* Spektrum-Verlag, 2002.

[LHM98] B. Liu, W. Hsu und Y. Ma. Integrating Classification and Association Rule Mining. *Proc. of the 4^{th} Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, Seiten 80–86, 1998.

[LKM03] S.-K. Lee, Y.-H. Kim und B.-R. Moon. Finding the Optimal Gene order in Displaying Microarray Data. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 2215–2226, 2003.

[LMV03] N. Labroche, N. Monmarche und G. Venturini. AntClust: Ant Clustering and Web Usage Mining. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 25–36, 2003.

[LTL+99] C.P. Lim, H.H. Toh, T.F. Lee, R.F. Harrison und R.L. Kennedy. Application of an Adaptive Neural Network to Medical Decision Support. *Int. Journ. of The Computer, The Internet and Management*, 7(1):9–20, 1999.

[LV02] C.H. Lee und A. Varshney. Representing Thermal Vibrations and Uncertainty in Molecular Surfaces. *SPIE Conf. on Visualization and Data Analysis, San Jose, CA, USA*, 2002.

[Lyn02] P.D. Lyne. Structure-Based Virtual Screening: An Overview. *Drug Discovery Today*, 7(20):1047–1055, 2002.

[Maa97] W. Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural Networks*, 10(9):1659–1671, 1997.

[Mar92] T. Martinetz. *Selbstorganisierende neuronale Netzwerkmodelle zur Bewegungssteuerung.* Dissertation, TU München, 1992.

[MCRN02] R. Mendes, P. Cortez, M. Rocha und J. Neves. Particle Swarms for Feedforward Neural Network Training. *Proc. of the Int. Joint Conf. on Neural Networks (IJCNN), Honolulu, HA, USA*, Seiten 1895–1899, 2002.

[MD89] J.E. Moody und C. Darken. Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation*, 1:281–294, 1989.

[Mit77] T.M. Mitchell. Version Spaces: A Candidate Elimination Approach to Rule Learning. *Proc. of the 5th Int. Joint Conf. on AI, Cambridge, MA, USA*, Seiten 305–310, 1977.

[Mit96] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1996.

[Mit97] T.M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[MJ03] J. Moss und C.G. Johnson. An Ant Colony Algorithm for Multiple Sequence Alignment in Bioinformatics. *Proc. of the 6^{th} Int. Conf. on Artificial Neural Networks and Genetic Algorithms (ICANNGA), Roanne, Frankreich*, Seiten 182–186, 2003.

[MKN+02] H. Midelfart, J. Komorowski, K. Norsett, F. Yadetie, A.K. Sandvik und A. Laegreid. Learning Rough Set Classifiers from Gene Expressions and Clinical Data. *Fundamenta Informaticae*, 53(2):155–183, 2002.

[MKP+00] M. Mathur, S.B. Karale, S. Priye, V.K. Jayaraman und B.D. Kulkarni. Ant Colony Approach to Continuous Function Optimization. *Ind. Eng. Chem. Res.*, 39:3814–3822, 2000.

[MLK03] P.M. Magwene, P. Lizardi und J. Kim. Reconstructing the Temporal Ordering of Biological Samples Using Microarray Data. *Bioinformatics*, 19(7):842–850, 2003.

[MM00] D.E. Makarov und H. Metiu. Using Genetic Programming to Solve the Schrödinger Equation. *J. Phys. Chem.*, 104:8540–8545, 2000.

[MM03] C.E. Mortimer und U. Müller. *Chemie. Das Basiswissen der Chemie.* Thieme Verlag, 6. Aufl., 2003.

[MSPGGM+03] M. Murcia-Soler, F. Perez-Gimenez, F.J. Garcia-March, M.T. Salabert-Salvador, W. Dfaz-Villanueva und M.J. Castro-Bleda. Drugs and Nondrugs: An Effective Discrimination with Topological Methods and Artificial Neural Networks. *J. Chem. Inf. Comput. Sci.*, Seiten 1688–1702, 2003.

[MVJB03] P. Mazzatorte, M. Vracko, A. Jezierska und E. Benfenati. Modeling Toxicity by Using Supervised Kohonen Neural Networks. *J. Chem. Inf. Comput. Sci.*, 43:485–492, 2003.

[MW02] R. Merkl und S. Waack. *Bioinformatik interaktiv.* Wiley-VCH, 2002.

[Nau00] D. Nauck. Adaptive Rule Weights in Neuro-Fuzzy Systems. *Neural Computing and Applications*, 9:60–70, 2000.

[NBG+02] C.-D. Neagu, E. Benfenati, G. Gini, P. Mazzatorta und A. Roncaglioni. Neuro-fuzzy Knowledge Representation For Toxicity Prediction of Organic Compouds. *Proc. of the 15th Europ. Conf. on Artificial Intelligence (ECAI), Lyon, Frankreich*, Seiten 498–502, 2002.

[NHH98] C. Notredame, L. Holm und D.G. Higgins. COFFEE: An Objective Function for Multiple Sequence Alignments. *Bioinformatics*, 14(5):407–422, 1998.

[NK99] D. Nauck und R. Kruse. Obtaining Interpretable Fuzzy Classification Rules from Medical Data. *Artificial Intelligence in Medicine*, 16(2):149–169, 1999.

[NKK96] D. Nauck, F. Klawonn und R. Kruse. *Neuronale Netze und Fuzzy-Systeme.* Vieweg-Verlag, 2. überarb. und erw. Aufl., 1996. Mittlerweile erschienen als 3. Aufl.: D. Nauck, C. Borgelt, F. Klawonn, R. Kruse: Neuro-Fuzzy-Systeme.

[NVS99] V. Neagoe, M. Valcu und B. Sabac. A Neural Approach for Detection of Road Direction in Autonomous Navigation. *Proc. of the Int. Conf., 6th Fuzzy Days, Dortmund*, Seiten 324–333, 1999.

[NWRY02] A. Narayanan, X. Wu und Z. Rong Yang. Mining Viral Protease Data to Extract Cleavage Knowledge. *Bioinformatics*, 18, Suppl. 1:S5–S13, 2002.

[NY03] N. Nagarajan und G. Yona. A Multi-Expert System for the Automatic Detection of Protein Domains from Sequence Information. *Proc. of the 7th Annual Int. Conf. on Research in Computational Molecular Biology, Berlin*, Seiten 224–234, 2003.

[O+01] C. Ordonez u.a. Mining Constrained Association Rules to Predict Heart Disease. *Proc. of the 1st Int. Conf. on Data Mining (ICDM), San Jose, CA, USA*, Seiten 433–440, 2001.

[OKSI02] T. Oyama, K. Kitano, K. Satou und T. Ito. Extraction of Knowledge on Protein-Protein Interaction by Association Rule Discovery. *Bioinformatics*, 18(5):705–714, 2002.

[OT03] C.H. Ooi und P. Tan. Genetic Algorithms Applied to Multi-Class Prediction for the Analysis of Gene Expression Data. *Bioinformatics*, 19(1):37–44, 2003.

[OW03] T. Oda und T. White. Developing an Immunity to Spam. *Proc. of the 5th Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 231–242, 2003.

[PA02] J. Paetz und B. Arlt. A Neuro-Fuzzy Based Alarm System for Septic Shock Patients with a Comparison to Medical Scores. *Proc. of the 3rd Int. Symp. on Medical Data Analysis (ISMDA), Rom, Italien*, Seiten 42–52, 2002.

[Pae01] J. Paetz. Metric Rule Generation with Septic Shock Patient Data. *Proc. of the 1st IEEE Int. Conf. on Data Mining (ICDM), San Jose, CA, USA*, Seiten 637–638, 2001.

[Pae02a] J. Paetz. A Note on Core Regions of Membership Functions. *Proc. of the 2nd Europ. Symp. on Intelligent Technologies, Hybrid Systems and their Implementation on Smart Adaptive Systems (EUNITE), Albufeira, Portugal*, Seiten 167–173, 2002. Verlag Mainz, Wissenschaftsverlag, Aachen.

[Pae02b] J. Paetz. *Adaptive Regelgenerierung und ihre Verwendung zur Diagnose des septischen Schocks.* Dissertation, J.W. Goethe-Universität Frankfurt am Main, 2002.

[Pae02c] J. Paetz. Durchschnittsbasierte Generalisierungsregeln Teil I: Grundlagen. Frankfurter Informatik-Berichte Nr. 1/02, Institut für Informatik, Fachbereich Biologie und Informatik, 2002.

[Pae02d] J. Paetz. Feature Selection for RBF Networks. *Proc. of the 9th Int. Conf. on Neural Information Processing (ICONIP), Singapur*, Seiten 986–990, 2002.

[Pae02e] J. Paetz. Intersection Based Generalization Rules for the Analysis of Symbolic Septic Shock Patient Data. *Proc. of the 2nd IEEE Int. Conf. on Data Mining (ICDM), Maebashi City, Japan*, Seiten 673–676, 2002.

[Pae02f] J. Paetz. Selecting the Representative Neuro-Fuzzy Model. *Proc. of the 1st Int. Conf. on Fuzzy Systems and Knowledge Discovery (FSKD), Singapur*, Seiten 737–741, 2002.

[Pae03a] J. Paetz. Evolving Score Neural Networks. *Proc. of the 1st Indian Int. Conf. on Artificial Intelligence (IICAI)*, Seiten 57–65, 2003.

[Pae03b] J. Paetz. Fuzzy Sets are Fuzzy-Continuous. *Proc. of the 22nd Int. Conf. of the North American Fuzzy Information Processing Soc. (NAFIPS), Chicago, IL, USA*, Seiten 244–247, 2003.

[Pae03c] J. Paetz. Knowledge Based Approach to Septic Shock Patient Data Using a Neural Network with Trapezoidal Activation Functions. *Artificial Intelligence in Medicine*, 28(2):207–230, 2003. Special Issue: Knowledge-Based Neurocomputing in Medicine.

[Pae04a] J. Paetz. Evolutionary Optimization of Interval Rules for Drug Design. *Proc. of the 1st IEEE Int. Symp. on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), La Jolla, CA, USA*, Seiten 238–243, 2004.

[Pae04b] J. Paetz. Reducing the Number of Neurons in Radial Basis Function Networks with Dynamic Decay Adjustment. *Neurocomputing*, 62:79–91, 2004.

[Pau00] G. Paun. Computing with Membranes. *Journal of Computer and System Sciences*, 61:108–143, 2000.

[Pau02] G. Paun. *Membrane Computing: An Introduction.* Springer-Verlag, 2002.

[Paw82] Z. Pawlak. Rough Sets. *Int. Journ. of Computer and Information Sciences*, 11:341–356, 1982.

[PB99] U. Pietruschka und R. Brause. Using Growing RBF-Nets in Rubber Industry Process Control. *Neural Computing and Applications*, 8(2):95–105, 1999.

[PB01] J. Paetz und R. Brause. A Frequent Patterns Tree Approach for Rule Generation with Categorical Septic Shock Patient Data. *Proc. of the 2nd Int. Symp. on Medical Data Analysis (ISMDA), Madrid, Spanien*, Seiten 207–212, 2001.

[PB02a] J. Paetz und R. Brause. Durchschnittsbasierte Generalisierungsregeln Teil II: Analyse von Daten septischer Schock-Patienten. Frankfurter Informatik-Berichte Nr. 2/02, Institut für Informatik, Fachbereich Biologie und Informatik, 2002.

[PB02b] J. Paetz und R. Brause. Rule Generation and Model Selection Used for Medical Diagnosis. *Journal of Intelligent and Fuzzy Systems*, 12(1):69–78, 2002. Special Issue: Challenges for Future Intelligent Systems in Biomedicine.

[PDY01] S.K. Pal, T.S. Dillon und D.S. Yeung (Hrsg.). *Soft Computing in Case Based Reasoning.* Springer-Verlag, 2001.

[PFF+04] J. Paetz, U. Fechner, L. Franke, S. Renner, P. Schneider und G. Schneider. Pharmacophore Feature Selection with a Neuro-Fuzzy System. *Proc. of the 4th Europ. Symp. on Intelligent Technologies, Hybrid Systems and their Implementation on Smart Adaptive Systems (EUNITE), Aachen*, Seiten 179–184, 2004.

[PG90a] T. Poggio und F. Girosi. A Theory of Networks for Approximation and Learning, 1990. AI Memo 1140, Massachusets Institute of Technology und CBIP Paper 31, Whitaker College.

[PG90b] T. Poggio und F. Girosi. Networks for Approximation and Learning. *Proc. of the IEEE*, 78(9):1481–1497, 1990.

[Pis98] Nadia Pisanti. DNA Computing: A Survey. *Bulletin of the EATCS 64*, 64:188–216, 1998.

[PK00] M. Polak und J. Kadukova. Artificial Life Simulation of Living Algae Cells, 2000. In: Sincak, P., Vascak, J. (Hrsg.) Quo Vadis Computational Intelligence?

[PLF01] R.S. Parpinelli, H.S. Lopes und A.A. Freitas. An Ant Colony Based System for Data Mining: Applications to Medical Data. *Proc. of the 3rd Genetic and Evolutionary Computation Conference (GECCO), San Francisco, CA, USA*, Seiten 791–798, 2001.

[PM02] L. Pagie und M. Mitchell. A Comparison of Evolutionary and Coevolutionary Search. *International Journal of Computational Intelligence and Applications*, 2(1):53–69, 2002.

[Poh00] H. Pohlheim. *Evolutionäre Algorithmen: Verfahren, Operatoren und Hinweise für die Praxis.* Springer-Verlag, 2000.

[PS04] J. Paetz und G. Schneider. Virtual Screening using Local Neuro-Fuzzy Rules. *Proc. of the 13th IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE), Budapest, Ungarn*, Seiten 861–866, 2004.

[QS88] N. Qian und T.J. Sejnowski. Predicting the Secondary Structure of Globular Proteins Using Neural Network Models. *J. Mol. Biol.*, 202:865–884, 1988.

[RBM+98] C.D. Rosin, R.K. Belew, G.M. Morris, A.J. Olson und D.S. Goodsell. Computational Coevolution of Antiviral Drug Resistance. *Artificial Life*, 4:41–59, 1998.

[RCE82] D.L. Reilly, L.N. Cooper und C. Elbaum. A Neural Model for Category Learning. *Biological Cybernetics*, 45:35–41, 1982.

[Rec73] I. Rechenberg. *Evolutionsstrategie.* F. Frommann Verlag, 1973. Mit einem Nachwort von Manfred Eigen.

[Rec94] I. Rechenberg. *Evolutionsstrategie '94.* F. Frommann Verlag, 1994.

[Ric02] R. Richards. Application of Multiple Artificial Intelligence Techniques for an Aircraft Carrier Landing Decision Support Tool. *Proc. of the 1st Int. Conf. on Fuzzy Systems and Knowledge Discovery (FSKD)*, Seiten 7–11, 2002.

[Rit93] H. Ritter. Parametrized Self-Organizing Maps. *3rd Int. Conf. on Artificial Neural Networks (ICANN), Amsterdam, Niederlande*, Seiten 568–575, 1993.

[RKSK04] H. Riedesel, B. Kolbeck, O. Schmetzer und E.-W. Knapp. Peptide Binding at Class I Major Histocompatibility Complex Scored with Linear Functions and Support Vector Machines. *Genome Informatics*, 15(1):198–212, 2004.

[Roj93] R. Rojas. *Theorie der neuronalen Netze: Eine systematische Einführung.* Springer-Verlag, 1993.

[RS93] B. Rost und C. Sander. Prediction of Protein Secondary Structure at Better than 70% Accuracy. *J. Mol. Biol.*, 232:584–599, 1993.

[RV02] J. Ramik und M. Vlach. Fuzzy Mathematical Programming: A Unified Approach Based On Fuzzy Relations. *Fuzzy Optimization and Decision Making*, 1:335–346, 2002.

[RWO+03] D.M. Reif, B.C. White, N. Olsen, T. Aune und J.H. Moore. Complex Function Sets Improve Symbolic Discriminant Analysis of Microarray Data. *Proc. of the 5th Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 2277–2287, 2003.

[SA96] R. Srikant und R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. *Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal , Kanada*, Seiten 1–12, 1996.

[SAHH02] A. Shmygelska, R. Aguirre-Herandez und H.H. Hoos. An Ant Colony Optimization Algorithm for the 2D HP Protein Folding Problem. *Proc. of the 3rd Int. Workshop of Ant Algorithms (ANTS), Brüssel, Belgien*, Seiten 40–52, 2002.

[SB99] R. Silipo und M.R. Berthold. Discriminative Power of Input Features in a Fuzzy Model. *Proc. of the 3rd Int. Symp. on Intelligent Data Analysis (IDA)*, Seiten 87–98, 1999.

[Sch95] M. Schweizer. *Wissensanalyse und -erhebung mit Kohonen-Netzen am praktischen Beispiel der Lawinenprognose.* Dissertation, Univ. Zürich, 1995.

[Sch97] B. Schölkopf. *Support Vector Learning.* Dissertation, TU Berlin, 1997.

[Sch00] G. Schneider. Neural Networks are Useful Tools for Drug Design. *Neural Networks*, 13:15–16, 2000.

[Sei05] R. Seising. *Die Fuzzifizierung der Systeme: Die Entstehung der Fuzzy Set Theorie und ihrer ersten Anwendungen: Ihre Entwicklung bis in die 70er Jahre des 20. Jahrhunderts.* Franz Steiner Verlag, Stuttgart, Boethius Band 54, 2005.

[SF03] C. Shyu und J.A. Foster. Evolving Consensus Sequence for Multiple Sequence Alignment with a Genetic Algorithm. *Proc. of the 5th Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 2313–2324, 2003.

[Sha90] J.W. Shavlik. Case-Based Reasoning with Noisy Case Boundaries: An Application in Molecular Biology. Technical Report CS-TR-1990-988, Univ. of Wisconsin, Madison, WI, USA, 1990.

[SHF94] E. Schoeneburg, F. Heinzmann und S. Feddersen. *Genetische Algorithmen und Evolutionsstrategien: Eine Einführung in Theorie und Praxis der simulierten Evolution.* Addison Wesley, 1994.

[Sip98] M. Sipper. Fifty Years of Research on Self-Replication: An Overview. *Artificial Life*, 4:237–257, 1998.

[SKL05] J. Sim, S.-Y. Kim und J. Lee. Prediction of Protein Solvent Accessibility Using Fuzzy k-Nearest Neighbor Method. *Bioinformatics*, 21(12):2844–2849, 2005.

[SKPF03] L. Spector, J. Klein, C. Perry und M. Feinstein. Emergence of Collective Behavior in Evolving Populations of Flying Agents. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 61–73, 2003.

[SKS02] K. Socha, J. Knowles und M. Samples. A Max-Min Ant System for the University Course Timetabling Problem. *Proc. of the 3rd Int. Workshop of Ant Algorithms (ANTS), Brüssel, Belgien*, Seiten 1–13, 2002.

[SKSK99] C.J. Savoie, N. Kamikawaji, T. Sasazuki und S. Kuhara. Use of BONSAI Decision Trees for the Identification of Potential MHC Class I Peptide Epitope Motifs. *Pacific Symp. on Biocomputing*, Seiten 182–189, 1999.

[SM96] I. Scott und B. Mulgrew. Orthonormal Function Neural Network for Nonlinear System Modeling. *Proc. of the Int. Conf. on Neural Networks (ICNN), Washington, DC, USA*, 4:1847–1852, 1996.

[SM03] O. Sverud und R.M. MacCallum. Towards Optimal Views of Proteins. *Bioinformatics*, 19(7):882–888, 2003.

[SML99] J. Selbig, T. Mevissen und T. Lengauer. Decision Tree-based Formation of Concensus Protein Secondary Structure Prediction. *Bioinformatics*, 15(12):1039–1046, 1999.

[SMR03] P.M. Selzer, R.J. Marhöfer und A. Rohwer. *Angewandte Bioinformatik.* Springer-Verlag, 2003.

[SMRS02] M. Sebban, I. Mokrousov, N. Rastogi und C. Sola. A Data-Mining Approach to Spacer Oligonucleotide Typing of Mycobacterium Tubercolosis. *Bioinformatics*, 18(2):235–243, 2002.

[SN03] G. Schneider und M. Nettekoven. Ligand-Based Combinatorial Design of Selective Purineric Receptor (A_{2A}) Antagonists Using Self-Organizing Maps. *J. Comb. Chem.*, 5:233–237, 2003.

[SNGS99] G. Schneider, W. Neidhart, T. Giller und G. Schmid. Scaffold Hopping by Topological Pharmacophore Search: A Contribution to Virtual Screening. *Angewandte Chemie, International Edition*, 38(19):2894–2895, 1999.

[Sol00] R.V. Sole. Pattern Formation and Optimization in Army Raids. *Artificial Life*, 6:219–226, 2000.

[Spe90] D.F. Specht. Probabilistic Neural Networks. *Neural Networks*, 3:109–118, 1990.

[SS95] R. Schlittgen und B.H.J. Streitberg. *Zeitreihenanalyse*. R. Oldenbourg-Verlag, 6. unwes. veränd. Aufl., 1995.

[SS03] P. Schneider und G. Schneider. Collection of Bioactive Reference Compounds for Focused Library Design. *QSAR Comb. Sci.*, 22:713–718, 2003.

[SSS03] G. Schneider und S. Sung-Sau. *Adaptive Systems in Drug Design*. Eurekah.com / Landes Bioscience, 2003.

[Ste03] G. Steger. *Bioinformatik: Methoden zur Vorhersage von RNA- und Proteinstrukturen*. Birkhäuser Verlag, 2003.

[Sto95] M. Stockhausen. *Mathematik für Chemiker*. Steinkopff-Verlag, 1995.

[Str96] L. Stryer. *Biochemie*. Spektrum Verlag, 4. Aufl., 1996. Mittlerweile erschienen als: J.M. Berg, J.L. Tymoczko, L. Stryer: Biochemie, 5. Aufl., 2003.

[SVA97] R. Srikant, Q. Vu und R. Agrawal. Mining Association Rules with Item Constraints. *Proc. of the 3^{rd} Int. Conf. on Knowledge Discovery in Databases and Data Mining (KDD), Newport Beach, CA, USA*, 1997.

[SW92] R. Schaback und H. Werner. *Numerische Mathematik*. Springer-Verlag, 4. vollst. überarb. Aufl., 1992.

[SW98] G. Schneider und P. Wrede. Artificial Neural Networks for Computer-based Molecular Design. *Biophysics & Molecular Biology*, 70:175–222, 1998.

[Tah03] S.M. Taheri. Trends in Fuzzy Statistics. *Austrian Journal of Statistics*, 32:239–257, 2003.

[THHK02] S. Tomida, T. Hanai, H. Honda und T. Kobayashi. Analysis of Expression Profile Using Fuzzy Adaptive Resonance Theory. *Bioinformatics*, 18(8):1073–1083, 2002.

[THV94] S. Tan, J. Hao und J. Vandewalle. Identification of Nonlinear Systems by RBF Neural Networks. *Proc. of the 4^{th} Int. Conf. on Artificial Neural Networks (ICANN), Amsterdam, Niederlande*, Seiten 1203–1206, 1994.

[THYD03] R. Thomson, T.C. Hodgman, Z.R. Yang und A.K. Doyle. Characterizing Proteolytic Cleavage Site Activity Using Bio-Basis Function Neural Networks. *Bioinformatics*, 19(14):1741–1747, 2003.

[TJS00] K. Torp, C.S. Jensen und R.T. Snodgrass. Effective Timestamping in Databases. *The VLDB Journal*, 8(3):267–288, 2000.

[TN03] A. Torres und J.J. Nieto. The Fuzzy Polynucleotide Space: Basic Properties. *Bioinformatics*, 19(5):587–592, 2003.

[Toi96] H. Toivonen. *Discovery of Frequent Patterns in Large Data Collections.* Dissertation, Univ. of Helsinki, 1996. Zugl. Series of Publications A, Report A-1996-5, Dept. of Computer Science, Univ. of Helsinki, Finnland.

[Ton90] H. Tong. *Non-Linear Time Series: A Dynamical System Approach.* Clarendon Press, 1990.

[Tou03] M. Toussaint. Demonstrating the Evolution of Complex Genetic Representations: An Evolution of Artificial Plants. *Proc. of the 5^{th} Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 86–97, 2003.

[Tri00] E.N. Trifonov. Earliest Pages of Bioinformatics. *Bioinformatics*, 16(1):5–9, 2000.

[TSMH99] J. Tetteh, T. Suzuki, E. Metcalfe und S. Howells. Quantitative Structure-Property Relationships for the Estimation of Boiling Point and Flash Point Using a Radial Basis Function Neural Network. *J. Chem. Inf. Comput. Sci.*, 39:491–507, 1999.

[Tsu98] S. Tsumoto. Automated Extraction of Medical Expert System Rules From Clinical Databases Based on Rough Set Theory. *Information Sciences*, 67-84:67–84, 1998.

[Tsu00] H. Tsukimoto. Extracting Rules from Trained Neural Networks. *IEEE Transactions on Neural Networks*, 11(2):377–389, 2000.

[TT01] A. Tettamanzini und M. Tomassini. *Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems.* Springer-Verlag, 2001.

[UGC+00] P. Uetz, L. Giot, G. Cagney, T.A. Mansfield, R.S. Judson, J.R. Knight und D. Lockshon. A Comprehensive Analysis of Protein-Protein Interactions in *Saccaromyces cerevisiae. Nature*, 403:623–627, 2000.

[UH91] A. Ultsch und K.-U. Höffgen. Automatische Wissensakquisition für Fuzzy-Expertensysteme aus selbstorganisierenden neuronalen Netzen. Forschungsbericht Nr.: 404, Univ. Dortmund, 1991.

[Ult91] A. Ultsch. Konnektionistische Modelle und ihre Integration mit wissensbasierten Systemen. Forschungsbericht Nr.: 396, Univ. Dortmund, 1991. Zugl. Habilitationsschrift.

[US90] A. Ultsch und H.P. Siemon. Kohonen's Self-Organizing Feature Maps for Exploratory Data Analysis. *Proc. of the Int. Conf. on Neural Networks (ICNN), Paris, Frankreich*, Seiten 305–308, 1990.

[Ves99] J. Vesanto. SOM-Based Data Visualization Methods. *Intelligent Data Analysis*, 3(2):111–126, 1999.

[VWG00] T. Villmann, H. Wieland und M. Geyer. Data Mining and Knowledge Discovery in Medical Applications Using Self-Organizing Maps. *1^{st} Int. Symp. on Medical Data Analysis (ISMDA), Frankfurt am Main*, Seiten 138–151, 2000.

[WBMJ99] M.F. Wilkins, L. Boddy, C.W. Morris und R.R. Jonker. Identification of Photoplankton from Flow Cytometry Data by Using Radial Basis Function Neural Networks. *Applied and Environmental Microbiology*, 65(10):4404–4410, 1999.

[WCG+97] H. van de Waterbeemd, R.E. Carter, G. Grassy, H. Kubinyi, Y.C. Martin, M.S. Tute und P. Willet. Glossary of Terms Used in Computational Drug Design. Albany Molecular Research, Inc., Technical Report, Band 2, Nr. 17, 1997.

[Wei02] K. Weicker. *Evolutionäre Algorithmen.* Teubner, 2002.

[WG00] M. Wagener und J. van Geerestein. Potential Drugs and Nondrugs: Prediction and Identification of Important Structural Features. *J. Chem. Inf. Comput. Sci.*, 40:280–292, 2000.

[Wil95] S.W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[WLDH02] D. Wang, N.K. Lee, T.S. Dillon und N.J. Hoogenraad. Protein Sequences Classification Using Radial Basis Function (RBF) Neural Networks. *Proc. of the 9th Int. Conf. on Neural Information Processing (ICONIP), Singapur*, Seiten 764–768, 2002.

[WM92] L. Wang und J. Mendel. Generating Fuzzy Rules by Learning from Examples. *IEEE Transactions on Systems, Man, Cybernetics*, 22(6):1414–1427, 1992.

[WM00] C.H. Wu und J.W. McLarty. *Neural Networks and Genome Informatics.* Elsevier, 2000.

[WWT99] P.C. Wong, P. Whitney und J. Thomas. Visualizing Association Rules for Text Mining. *IEEE Symp. on Information Visualization (InfoVis), San Francisco, CA, USA*, 1999.

[XLZ+02] Y.H. Xiang, M.C. Liu, X.Y. Zhang, R.S. Zhang und Z.D. Hu. Quantitative Prediction of Liquid Chromatography Retention of N-Benzylideneanilines Based on Quantum Chemical Parameters and Radial Basis Function Neural Network. *J. Chem. Inf. Comput. Sci.*, 42:592–597, 2002.

[Yan03] J.-M. Yang. An Evolutionary Approach for Molecular Docking. *Proc. of the 5th Genetic and Evolutionary Computation Conference (GECCO), Chicago, IL, USA*, Seiten 2372–2383, 2003.

[YC03] Z.R. Yang und K.-C. Chou. Mining Biological Data Using Self-Organizing Map. *J. Chem. Inf. Comput. Sci.*, 43:1748–1753, 2003.

[YH03] X. Yin und J. Han. Classification based on Predictive Association Rules. *Proc. of the 2003 SIAM Int. Conf. on Data Mining (SDM), San Francisco, CA, USA*, 2003.

[Zad65] L.A. Zadeh. Fuzzy Sets. *Inf. Control*, 8:338–353, 1965.

[Zad69] L.A. Zadeh. Biological Application of the Theory of Fuzzy Sets and Systems. In: L.D. Proctor (Hrsg.), The Proc. of an Int. Symp. on Biocybernetics of the Central Nervous System, Little, Brown and Comp., 1969.

[Zad73] L.A. Zadeh. Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Trans. Syst., Man, Cybern.*, SMC-3(1):28–44, 1973.

[Zad81] L.A. Zadeh. Possibility Theory and Soft Data Analysis. In: L. Cobb, R.M. Thrall (Hrsg.), Mathematical Frontiers of Social and Policy Sciences, Westview Press, 1981.

[Zad96] L.A. Zadeh. Fuzzy Logic = Computing with Words. *IEEE Transactions on Fuzzy Systems*, 2:103–111, 1996.

[Zad00] L.A. Zadeh. Outline of Computational Theory of Perceptions Based on Computing with Words. In: Soft Computing and Intelligent Systems, N.K. Sinha, M.M. Gupta, L.A. Zadeh (Hrsg.), Academic Press, 2000.

[Zak98] M.J. Zaki. *Scalable Data Mining for Rules.* Dissertation, Univ. of Rochester, New York, USA, 1998.

[Zel94] A. Zell. *Simulation neuronaler Netze.* Addison-Wesley, 1994. 1. unveränd. Nachdruck 1996.

[ZG93] J. Zupan und J. Gasteiger. *Neural Networks for Chemists: An Introduction.* VCH Verlagsgesellschaft, 1993. auch als 2. Aufl. 1999 erschienen.

[Zim91] H.J. Zimmermann. *Fuzzy Set Theory and its Applications.* Kluwer Academic Publishers, 2. Aufl., 1991.

Sachverzeichnis