Undergraduate Topics in Computer Science

Wolfgang Ertel

# Introduction to Artificial Intelligence

UTiCS

Springer

# Bachelor-Themen in Informatik

**Undergraduate Topics in Computer Science (UTiCS) bietet hochwertige Lehrinhalte für Studenten aller Bereiche der Informatik und Informationswissenschaft. Von grundlegendem und theoretischem Material bis hin zu Abschlussthemen und -anwendungen verfolgen UTiCS-Bücher einen frischen, prägnanten und modernen Ansatz und eignen sich ideal für das Selbststudium oder für einen ein- oder zweisemestrigen Kurs. Die Texte sind alle von ausgewiesenen Experten auf ihrem Gebiet verfasst, von einem internationalen Beirat geprüft und enthalten zahlreiche Beispiele und Problemstellungen. Viele enthalten vollständig funktionierende Lösun**

**Für weitere Bände:**
**www.springer.com/series/7592**

**Wolfgang Ertel**

# Einführung in Künstliches Intelligenz

**Übersetzt von Nathanael Black**
**Mit Illustrationen von Florian Mast**

Springer

**Prof. Dr. Wolfgang Ertel**
**FB Elektrotechnik und Informatik**
**Hochschule Ravensburg-Weingarten**
**Fachhochschule Weingarten**
**Deutschland**
**ertel@hs-weingarten.de**



**ISSN 1863-7310**
**ISBN 978-0-85729-298-8** **e-ISBN 978-0-85729-299-5**
**DOI 10.1007/978-0-85729-299-5**
**Springer London Dordrecht Heidelberg New York**

**Katalogisierung der British Library in Publikationsdaten**
**Ein Katalogeintrag für dieses Buch ist in der British Library erhältlich**

**Kontrollnummer der Library of Congress: 2011923216**

**Ursprünglich in deutscher Sprache veröffentlicht von Vieweg+Teubner, 65189 Wiesbaden, Deutschland, als „Ertel, Wolfgang: Grundkurs Künstliche Intelligenz. 2. rev. Auflage". © Vieweg+Teubner | GWV Fachverlage GmbH, Wiesbaden 2009**



**Gedruckt auf säurefreiem Papier**

**Springer ist Teil von Springer Science+Business Media (www.springer.com)**

# Vorwort

Künstliche Intelligenz (KI) hat das klare Ziel, Intelligenz zu verstehen und intelligente Systeme aufzubauen. Allerdings sind die Methoden und Formalismen auf dem Weg zu diesem Ziel nicht fest verankert, was dazu geführt hat, dass KI heute aus einer Vielzahl von Teildisziplinen besteht. Die Schwierigkeit bei einem Einführungskurs in die KI liegt darin, möglichst viele Zweige zu vermitteln, ohne zu viel Tiefe und Präzision zu verlieren.

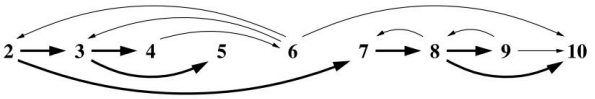
Das Buch von Russell und Norvig [RN10] ist mehr oder weniger die Standardeinführung in die KI. Da dieses Buch jedoch 1.152 Seiten umfasst und für die meisten Studenten zu umfangreich und zu teuer ist, waren die Anforderungen für das Schreiben dieses Buches klar: Es sollte eine leicht zugängliche Einführung in die moderne KI zum Selbststudium oder als Grundlage für ein Viererstudium sein Einstündige Vorlesung mit höchstens 300 Seiten. Das Ergebnis liegt vor

Auf 300 Seiten kann ein so umfangreiches Fachgebiet wie KI nicht vollständig abgedeckt werden. Um zu vermeiden, dass das Buch zu einem Inhaltsverzeichnis wird, habe ich versucht, in die Tiefe zu gehen und konkrete Algorithmen und Anwendungen in jedem der folgenden Zweige vorzustellen: Agenten, Logik, Suche, Argumentation mit Unsicherheit, maschinelles Lernen und neuronale Netze.

Die Bereiche Bildverarbeitung, Fuzzy-Logik und Verarbeitung natürlicher Sprache werden nicht im Detail behandelt. Der für die gesamte Informatik wichtige Bereich der Bildverarbeitung ist eine eigenständige Disziplin mit sehr guten Lehrbüchern, wie z. B. [GW08]. Einen ähnlichen Status hat die Verarbeitung natürlicher Sprache. Bei der Erkennung und Generierung von Text und gesprochener Sprache werden Methoden der Logik, des probabilistischen Denkens und neuronaler Netze angewendet. In diesem Sinne ist dieser Bereich Teil der KI. Andererseits ist die Computerlinguistik ein eigener umfangreicher Zweig der Informatik und hat viele Gemeinsamkeiten mit formalen Sprachen. In diesem Buch werden wir an mehreren Stellen auf solche geeigneten Systeme hinweisen, jedoch keine systematische Einführung geben. Für eine erste Einführung in dieses Gebiet verweisen wir auf die Kapitel. 22 und 23 in [RN10]. Die Fuzzy-Logik bzw. Fuzzy-Set-Theorie hat sich aufgrund ihrer primären Anwendung in der Automatisierungstechnik zu einem Teilgebiet der Regelungstheorie entwickelt und wird in den entsprechende Daher verzichten wir hier auf eine Einführung.

Die Abhängigkeiten zwischen den Kapiteln des Buches sind in der unten gezeigten Grafik grob skizziert. Um es einfach zu halten, Kap. 1 mit der grundsätzlichen Einleitung für alle weiteren Kapitel entfällt. Der dickere Pfeil von 2 nach 3 bedeutet beispielsweise, dass die Aussagenlogik eine Voraussetzung für das Verständnis der Prädikatenlogik ist. Der dünne Pfeil von 9 bis 10 bedeutet, dass neuronale Netze für das Verständnis von Reinforcement Learning hilfreich, aber nicht unbedingt notwendig sind. Dünn nach hinten

Pfeile sollen deutlich machen, dass spätere Kapitel das Verständnis vertiefen können
zu bereits erlernten Themen.




Dieses Buch richtet sich an Studierende der Informatik und anderer technischer Naturwissenschaften
Naturwissenschaften und erfordert größtenteils Mathematikkenntnisse auf High-School-Niveau.
An mehreren Stellen Erkenntnisse aus der linearen Algebra und der mehrdimensionalen Analysis
wird gebraucht. Für ein tieferes Verständnis der Inhalte ist die aktive Bearbeitung der Übungen
unabdingbar. Das bedeutet, dass die Lösungen nur konsultiert werden sollten
nach intensiver Arbeit mit jedem Problem, und nur um die eigenen Lösungen zu überprüfen, getreu
Leonardo da Vincis Motto „Lernen ohne Hingabe schadet dem Gehirn“. Etwas
Schwierigere Probleme sind mit ÿ und besonders schwierige mit ÿ ÿ gekennzeichnet.
Probleme, die Programmierkenntnisse oder spezielle Informatikkenntnisse erfordern, sind
mit ÿ gekennzeichnet .

Auf der Website des Buches unter www.hs-weingarten.de/~ertel/aibook digitale Materialien zu
den Übungen wie Trainingsdaten für Lernalgorithmen, eine Seite mit Verweisen auf im Buch
erwähnte KI-Programme, eine Linkliste zu die behandelten Themen, a
anklickbare Liste der Bibliographie, eine Errata-Liste und Präsentationsfolien für Dozenten
kann gefunden werden. Ich bitte den Leser, Anregungen, Kritik und Tipps dazu zu senden
Fehler direkt an ertel@hs-weingarten.de.

Dieses Buch ist eine aktualisierte Übersetzung meines deutschen Buches „Grundkurs Künstliche
Intelligenz“ erschienen im Vieweg Verlag. Mein besonderer Dank gilt dem Übersetzer
Nathan Black, der in einer hervorragenden transatlantischen Zusammenarbeit zwischen Deutschland und
Kalifornien hat über SVN, Skype und E-Mail diesen Text erstellt. Ich bin Franz Kurfeß dankbar, der
mich Nathan vorgestellt hat; an Matthew Wight für das Korrekturlesen der Übersetzung
Buch und an Simon Rees vom Springer Verlag für seine Geduld.

Ich möchte meiner Frau Evelyn für ihre Unterstützung und Geduld dabei danken
zeitaufwändiges Projekt. Besonderer Dank gilt Wolfgang Bibel und Chris Lobenschuss, die das
deutsche Manuskript sorgfältig korrigiert haben. Ihre Anregungen und Diskussionen führen zu
vielen Verbesserungen und Ergänzungen. Zum Lesen der Korrekturen und
Für weitere wertvolle Dienste möchte ich mich bei Richard Cubek, Celal Döven und Joachim bedanken
Feßler, Nico Hochgeschwender, Paul Kirner, Wilfried Meister, Norbert Perk, Peter Radtke, Markus
Schneider, Manfred Schramm, Uli Stärk, Michel Tokic, Arne
Usadel und alle interessierten Studierenden. Mein Dank geht auch an Florian Mast für die
Unbezahlbare Cartoons und eine sehr effektive Zusammenarbeit.

Ich hoffe, dass dieses Buch Ihnen während Ihres Studiums dabei hilft, meine Faszination mit Ihnen zu teilen
Künstliche Intelligenz.

Ravensburg Wolfgang Ertel
Februar 2011

# Inhalt

# Einführung 1

## 1.1 Was ist künstliche Intelligenz?

Der Begriff *Künstliche Intelligenz* weckt Emotionen. Da ist zum einen unsere Faszination für *Intelligenz,* die uns Menschen scheinbar einen besonderen Platz unter den Lebensformen einräumt. Es stellen sich Fragen wie *„Was ist Intelligenz?“, „Wie kann man Intelligenz messen?“* oder *„Wie funktioniert das Gehirn?“.* All diese Fragen sind sinnvoll, wenn es darum geht, künstliche Intelligenz zu verstehen. Die zentrale Frage für den Ingenieur, insbesondere für den Informatiker, ist jedoch die Frage nach der intelligenten Maschine, die sich wie ein Mensch verhält und intelligentes Verhalten zeigt.

Das Attribut *„künstlich“* könnte ganz andere Assoziationen hervorrufen. Es weckt Ängste vor intelligenten Cyborgs. Es erinnert an Bilder aus Science-Fiction-Romanen. Es stellt sich die Frage, ob wir versuchen sollten, unser höchstes Gut, die Seele, zu verstehen, zu modellieren oder sogar zu rekonstruieren.

Bei solch unterschiedlichen Interpretationen wird es schwierig, den Begriff *künstliche Intelligenz* oder *KI* einfach und eindeutig zu definieren. Dennoch möchte ich versuchen, anhand von Beispielen und historischen Definitionen den Bereich der KI zu charakterisieren. Im Jahr 1955 definierte John McCarthy, einer der Pioniere der KI, als erster den Begriff *künstliche Intelligenz* etwa wie folgt:

> Ziel der KI ist es, Maschinen zu entwickeln, die sich so verhalten, als wären sie intelligent.

Um diese Definition zu testen, könnte sich der Leser das folgende Szenario vorstellen. Etwa fünfzehn kleine Roboterfahrzeuge bewegen sich auf einer geschlossenen, vier mal vier Meter großen Fläche. Man kann verschiedene Verhaltensmuster beobachten. Manche Fahrzeuge bilden kleine Gruppen mit relativ wenig Bewegung. Andere bewegen sich friedlich durch den Raum und weichen jeder Kollision elegant aus. Wieder andere scheinen einem Anführer zu folgen. Auch aggressives Verhalten ist zu beobachten. Ist das, was wir sehen, intelliger

Nach McCarthys Definition können die oben genannten Roboter als intelligent bezeichnet werden. Der Psychologe Valentin Braitenberg hat gezeigt, dass dieses scheinbar komplexe Verhalten durch sehr einfache elektrische Schaltkreise hervorgerufen werden kann [Bra84]. Sogenannte Braitenberg-Fahrzeuge verfügen über zwei Räder, die jeweils von einem unabhängigen Elektromotor angetrieben werden. Die Geschwindigkeit jedes Motors wird durch eine

W. Ertel, *Einführung in die künstliche Intelligenz,*
Bachelor-Themen in Informatik, DOI
10.1007/978-0-85729-299-5_1, 

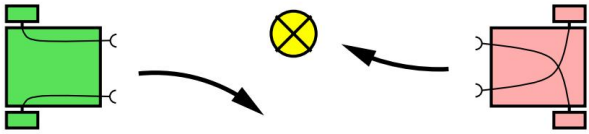


Abb. 1.1 Zwei sehr einfache Braitenberg-Fahrzeuge und ihre Reaktionen auf eine Lichtquelle

vorne am Fahrzeug, wie in Abb. 1.1 dargestellt. Je mehr Licht auf den Sensor trifft, desto schneller läuft der Motor. Fahrzeug 1 im linken Bildteil entfernt sich entsprechend seiner Konfiguration von einer Punktlichtquelle. Fahrzeug 2 hingegen bewegt sich auf die Lichtquelle zu. Durch weitere kleine Modifikationen können andere Verhaltensmuster entstehen, sodass wir mit diesen sehr einfachen Fahrzeugen das oben beschriebene beeindruckende Verhalten realisieren können.

Offensichtlich ist die obige Definition unzureichend, da KI das Ziel hat, schwierige praktische Probleme zu lösen, die für das Braitenberg-Fahrzeug sicherlich zu anspruchsvoll sind. In der Encyclopedia Britannica [Bri91] findet man eine Definition, die wie folgt laute

> KI ist die Fähigkeit digitaler Computer oder computergesteuerter Roboter, Probleme zu lösen, die normalerweise mit den höheren intellektuellen Verarbeitungsfähigkeiten des Menschen verbunden sind

**Doch diese Definition weist auch Schwächen auf.** Es würde beispielsweise zugeben, dass ein Computer mit großem Speicher, der einen langen Text speichern und bei Bedarf abrufen kann, über intelligente Fähigkeiten verfügt, denn das Auswendiglernen langer Texte kann sicherlich als eine ***höhere intellektuelle Verarbeitungsfähigkeit*** des Menschen angesehen werden, ebenso wie beispielsweise die schnelle Multiplikation zweier 20-stelliger Zahlen. Nach dieser Definition ist also jeder Computer ein KI-System. Dieses Dilemma wird durch die folgende Definition von Elaine Rich [Ric83] elegant gelöst:

> Künstliche Intelligenz ist die Untersuchung, wie man Computer dazu bringt, Dinge zu tun, in denen Menschen derzeit besser sind.

Rich, prägnant und prägnant charakterisiert, was KI-Forscher in den letzten 50 Jahren getan haben. Auch im Jahr 2050 wird diese Definition noch aktuell sein.

Aufgaben wie die Ausführung vieler Berechnungen in kurzer Zeit sind die Stärken digitaler Computer. In dieser Hinsicht übertreffen sie den Menschen um ein Vielfaches. In vielen anderen Bereichen ist der Mensch der Maschine jedoch weit überlegen. Wer beispielsweise einen fremden Raum betritt, erkennt die Umgebung innerhalb von Sekundenbruchteilen und kann bei Bedarf ebenso schnell Entscheidungen treffen und Maßnahmen planen. Bisher ist diese Aufgabe für autonome1 Roboter zu anspruchsvoll. Nach Richs Definition ist dies also eine Aufgabe der KI. Tatsächlich ist die Forschung zu autonomen Robotern ein wichtiges, aktuelles Thema in der KI. Der Bau von Schachcomputern hingegen hat an Bedeutung verloren, da diese bereits auf dem Niveau von Großmeistern oder darüber hinaus spielen.

Es wäre jedoch gefährlich, aus Richs Definition zu schließen, dass es bei KI nur um die pragmatische Umsetzung intelligenter Prozesse geht. Intelligente Systeme im Sinne von Richs Definition können nicht ohne eine tiefe Intelligenz aufgebaut werden

1Ein autonomer Roboter arbeitet selbstständig, ohne manuelle Unterstützung, insbesondere ohne Fernbedienung Kontrolle.

Verständnis des menschlichen Denkens und intelligenten Handelns im Allgemeinen, weshalb die Neurowissenschaften (siehe Abschn. 1.1.1) für die KI von großer Bedeutung sind. Dies zeigt auch, dass die anderen zitierten Definitionen wichtige Aspekte der KI widerspiegeln.

Eine besondere Stärke der menschlichen Intelligenz ist die Anpassungsfähigkeit. Wir sind in der Lage, uns durch Lernen an verschiedene Umweltbedingungen anzupassen und unser Verhalten entsprechend zu ändern . Gerade weil unsere Lernfähigkeit der von Computern so weit überlegen ist, ist ***maschinelles Lernen*** nach Richs Definition ein zentrales Teilgebiet der KI.

## 1.1.1 Gehirnforschung und Problemlösung

Durch die Erforschung intelligenter Systeme können wir versuchen zu verstehen, wie das menschliche Gehirn funktioniert, und es dann am Computer modellieren oder simulieren. Viele Ideen und Prinzipien auf dem Gebiet der neuronalen Netze (siehe Kapitel 9) stammen aus der Hirnforschung und dem verwandten Gebiet der Neurowissenschaften.

Ein ganz anderer Ansatz ergibt sich daraus, zielorientiert vorzugehen, von einem Problem auszugehen und zu versuchen, die optimale Lösung zu finden. Wie der Mensch das Problem löst, wird hier als unwichtig behandelt. Die Methode ist bei diesem Ansatz zweitrangig. An erster Stelle steht die optimale intelligente Lösung des Problems. Anstatt eine feste Methode (wie zum Beispiel die Prädikatenlogik) anzuwenden, hat die KI stets das Ziel, intelligente Agenten für möglichst viele verschiedene Aufgaben zu schaffen. Da die Aufgaben sehr unterschiedlich sein können, ist es nicht verwunderlich, dass die derzeit in der KI eingesetzten Methoden oft auch sehr unterschiedlich sind. Ähnlich wie die Medizin, die viele verschiedene, oft lebensrettende Diagnose- und Therapieverfahren umfasst, bietet auch KI eine breite Palette wirksamer Lösungen für unterschiedlichste Anwendungsbereiche.
Betrachten Sie zur geistigen Inspiration Abb. 1.2 auf Seite 4. Genau wie in der Medizin gibt es keine universelle Methode für alle Anwendungsbereiche von KI, sondern eine Vielzahl möglicher Lösungen für die Vielzahl unterschiedlicher großer und kleiner Alltagsprobleme.

*Die Kognitionswissenschaft* widmet sich der Erforschung des menschlichen Denkens auf einem etwas höheren Niveau. Ähnlich wie die Hirnforschung liefert dieser Bereich der praktischen KI viele wichtige Ideen. Andererseits führen Algorithmen und Implementierungen zu weiteren wichtigen Schlussfolgerungen darüber, wie menschliches Denken funktioniert. Somit profitieren diese drei Bereiche von einem fruchtbaren interdisziplinären Austausch. Gegenstand dieses Buches ist jedoch in erster Linie die problemorientierte KI als Teildisziplin der Informatik.

Es gibt viele interessante philosophische Fragen rund um Intelligenz und künstliche Intelligenz. Wir Menschen haben Bewusstsein; das heißt, wir können über uns selbst nachdenken und sogar darüber nachdenken, dass wir in der Lage sind, über uns selbst nachzudenken. Wie entsteht Bewusstsein? Viele Philosophen und Neurologen glauben heute, dass Geist und Bewusstsein mit der Materie, also mit dem Gehirn, verbunden sind. Die Frage, ob Maschinen eines Tages einen Geist oder ein Bewusstsein haben könnten, könnte irgendwann in der Zukunft relevant werden. Beim Geist-Körper-Problem geht es insbesondere darum, ob der Geist an den Körper gebunden ist oder nicht. Wir werden diese Fragen hier nicht diskutieren. Der interessierte Leser kann [Spe98, Spe97] zu Rate ziehen und ist eingeladen, sich im Rahmen des KI-Technologie-Studiums eine persönliche Meinung zu diesen Fragen zu bilden.

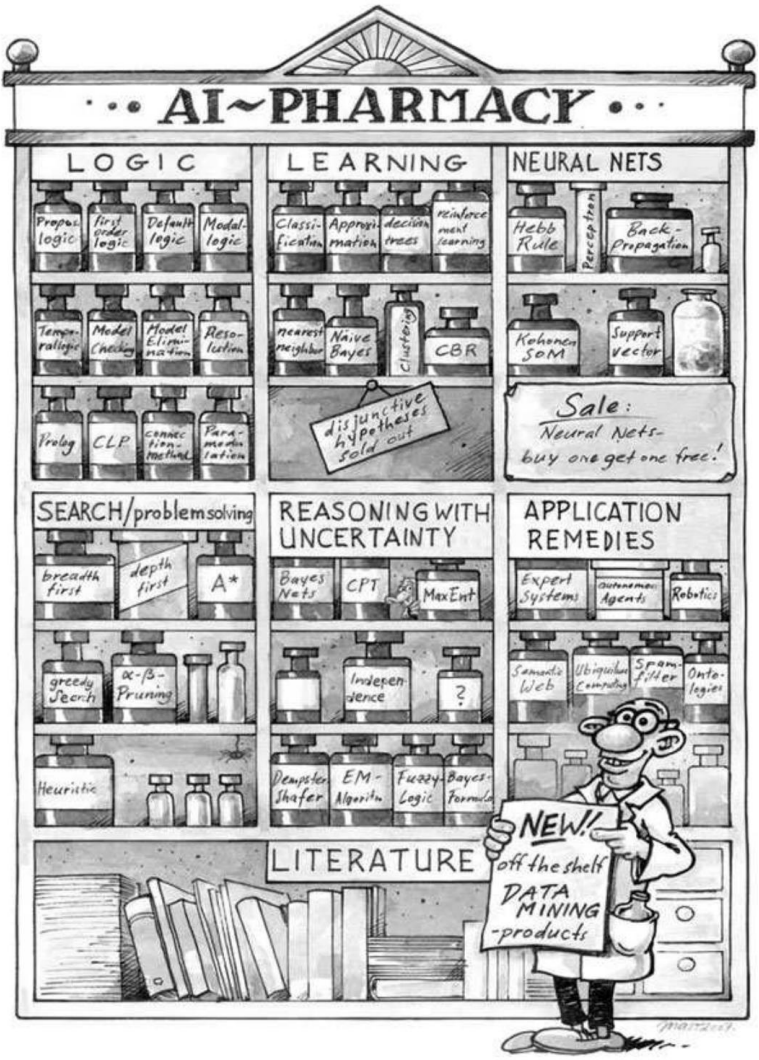



**Abb. 1.2 Eine kleine Auswahl der von KI angebotenen Lösungen**

## 1.1.2 Der Turing-Test und Chatterbots

**Alan Turing machte sich als früher Pionier der KI einen Namen mit seiner Definition einer intelligenten Maschine, bei der die jeweilige Maschine den folgenden Test bestehen muss. Die Testperson Alice sitzt in einem verschlossenen Raum mit zwei Computerterminals. Ein Terminal ist mit einer Maschine verbunden, das andere mit einer nicht böswilligen Person, Bob. Alice kann in beide Terminals Fragen eingeben. Sie erhält die Aufgabe, nach fünf Minuten zu entscheiden, welches Terminal zum Automaten gehört. Die Maschine besteht den Test, wenn sie Alice in mindestens 30 % der Fälle austricksen kann [Tur50].**

Während der Test philosophisch sehr interessant ist, ist er für die praktische KI, die sich mit der Problemlösung befasst, kein sehr relevanter Test. Die Gründe dafür sind ähnlich wie oben im Zusammenhang mit den Braitenberg-Fahrzeugen (siehe Übung 1.3 auf Seite 14).

Der KI-Pionier und Gesellschaftskritiker Joseph Weizenbaum entwickelte ein Programm namens *Eliza,* das die Fragen einer Testperson wie ein menschlicher Psychologe beantworten soll [Wei66]. Tatsächlich konnte er in vielen Fällen Erfolge vorweisen. Angeblich führte seine Sekretärin oft lange Diskussionen mit dem Programm. Heutzutage gibt es im Internet viele sogenannte *Chatterbots,* deren erste Reaktionen zum Teil recht beeindruckend sind. Nach einer gewissen Zeit wird jedoch ihr künstlicher Charakter deutlich. Einige dieser Programme sind tatsächlich lernfähig, während andere über außergewöhnliche Kenntnisse in verschiedenen Fächern verfügen, beispielsweise in Geographie oder Softwareentwicklung. Es gibt bereits kommerzielle Anwendungen für Chatterbots im Online-Kundensupport und möglicherweise weitere im Bereich E-Learning. Es ist denkbar, dass der Lernende und das E-Learning-System über einen Chatterbot kommu Der Leser möchte möglicherweise mehrere Chatterbots vergleichen und ihre Intelligenz in Übung 1.1 auf Seite 14 bewerten.

## 1.2 Die Geschichte der KI

KI greift auf viele wissenschaftliche Errungenschaften der Vergangenheit zurück, die hier nicht erwähnt werden, denn KI als eigenständige Wissenschaft existiert erst seit der Mitte des 20. Jahrhunderts. Tabelle 1.1 auf Seite 10 mit den wichtigsten KI-Meilensteinen und eine grafische Darstellung der Hauptbewegungen der KI in Abb. 1.3 auf Seite 6 ergänzen den folgenden Text.

### 1.2.1 Die ersten Anfänge

In den 1930er Jahren legten Kurt Gödel, Alonso Church und Alan Turing wichtige Grundlagen für die Logik und die theoretische Informatik. Von besonderem Interesse für die KI sind Gödels Theoreme. Der Vollständigkeitssatz besagt, dass die Prädikatenlogik erster Ordnung vollständig ist. Das bedeutet, dass jede wahre Aussage, die in der Prädikatenlogik formuliert werden kann, mit den Regeln einer formalen Analysis beweisbar ist. Auf dieser Grundlage könnten später automatische Theorembeweiser als Imple Mit dem Unvollständigkeitssatz zeigte Gödel, dass es in Logiken höherer Ordnung wahre Aussagen gibt, die nicht beweisbar sind.2 Damit deckte er schmerzhafte Grenzen formaler Systeme auf.

In diesen Zeitraum fällt auch Alan Turings Beweis der Unentscheidbarkeit des Halteproblems. Er zeigte, dass es kein Programm gibt, das entscheiden kann, ob ein bestimmtes beliebiges Programm (und seine entsprechende Eingabe) in einer Endlosschleife ausgefü

2Logiken höherer Ordnung sind Erweiterungen der Prädikatenlogik, in denen nicht nur Variablen, sondern auch Funktionssymbole oder Prädikate als Begriffe in einer Quantifizierung auftreten können. Tatsächlich hat Gödel nur gezeigt, dass jedes System, das auf Prädikatenlogik basiert und die Peano-Arithmetik formulieren kann, unvollständig

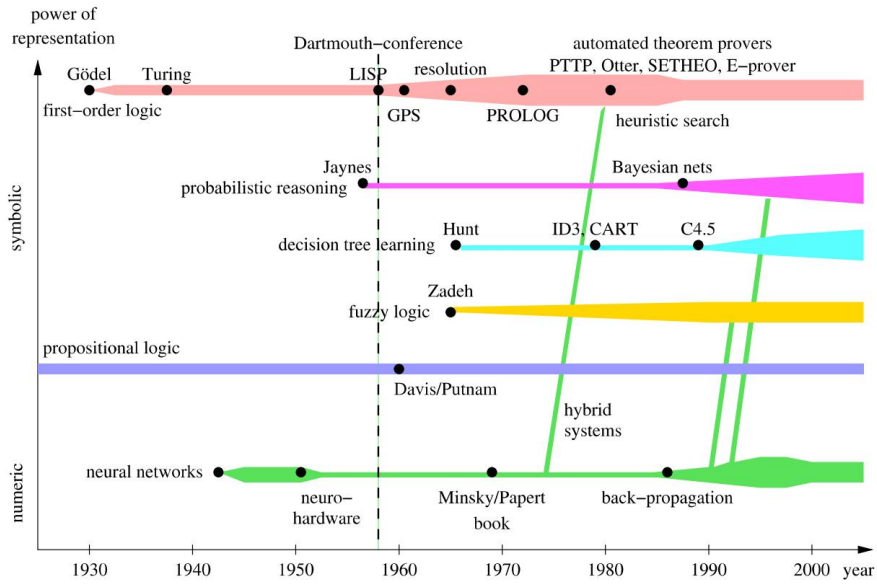



**Abb. 1.3 Geschichte der verschiedenen KI-Bereiche. Die Breite der *Balken* zeigt die Prävalenz der Methodenverwendung an**

**Damit identifizierte Turing auch eine Grenze für intelligente Programme. Daraus folgt beispielsweise, dass es niemals ein universelles**

**Programmverifikationssystem geben wird.3 In den 1940er Jahren entwarfen McCulloch, Pitts und Hebb auf der Grundlage neurowissenschaftlicher Erkenntnisse die ersten mathematischen Modelle neuronaler Netze. Damals fehlte den Computern jedoch die nötige Leistung, um ei**

## 1.2.2 Logik löst (fast) alle Probleme

**KI als praktische Wissenschaft der Gedankenmechanisierung konnte natürlich erst beginnen, als es programmierbare Computer gab. Dies war in den 1950er Jahren der Fall. Newell und Simon führten mit Logic Theorist den ersten automatischen Theorembeweiser ein und zeigten damit auch, dass man mit Computern, die eigentlich nur mit Zahlen arbeiten, auch Symbole verarbeiten kann. Gleichzeitig führte McCarthy mit der Sprache LISP eine Programmiersprache ein, die speziell für die Verarbeitung symbolischer Strukturen geschaffen wurde. Beide Systeme wurden 1956 auf der historischen Dartmouth-Konferenz vorgestellt, die als Geburtstag der KI gilt.**

**In den USA entwickelte sich LISP zum wichtigsten Werkzeug zur Implementierung symbolverarbeitender KI-Systeme. Danach entwickelte sich die als Resolution bekannte logische Folgerungsregel zu einem vollständigen Kalkül der Prädikatenlogik.**

---

**3Diese Aussage gilt für „völlige Korrektheit“, was einen Nachweis der korrekten Ausführung sowie einen Nachweis der Beendigung für jede gültige Eingabe impliziert.**

In den 1970er Jahren wurde die logische Programmiersprache PROLOG eingeführt Europäisches Gegenstück zu LISP. PROLOG bietet den Vorteil, direkt zu ermöglichen Programmierung mit Horn-Klauseln, einer Teilmenge der Prädikatenlogik. Wie LISP, PROLOG verfügt über Datentypen zur komfortablen Verarbeitung von Listen.

Bis weit in die 1980er Jahre herrschte vor allem bei der KI ein Durchbruchsgeist viele Logiker. Grund dafür waren die beeindruckenden Erfolge in Symbolverarbeitung. Mit dem Projekt „Computersysteme der fünften Generation“ in Japan und des ESPRIT-Programms in Europa wurden große Investitionen in den Bau getätigt Intelligente Computer.

Für kleine Probleme automatische Beweiser und andere Symbolverarbeitungssysteme hat manchmal sehr gut funktioniert. Die kombinatorische Explosion des Suchraums definierte jedoch ein sehr enges Fenster für diese Erfolge. Diese Phase der KI wurde in [RN10] als „Schau, Ma, keine Hände!“ beschrieben. Epoche.

Denn der wirtschaftliche Erfolg von KI-Systemen blieb hinter den Erwartungen zurück, die Finanzierung Die Nachfrage nach logikbasierter KI-Forschung in den Vereinigten Staaten ging in den 1980er Jahren dramatisch zurück.

### 1.2.3 Der neue Konnektionismus

In dieser Phase der Ernüchterung treffen Informatiker, Physiker und Kognitive Mit inzwischen ausreichend leistungsfähigen Computern konnten Wissenschaftler zeigen, dass mathematisch modellierte neuronale Netze lernfähig sind Trainingsbeispiele, um Aufgaben auszuführen, die bisher eine aufwendige Programmierung erforderten. Aufgrund der Fehlertoleranz solcher Systeme und ihrer Fähigkeit, Muster zu erkennen, waren vor allem bei der Mustererkennung erhebliche Erfolge möglich. Gesichtserkennung in Fotos und Handschrifterkennung sind zwei Beispielanwendungen. Das System Nettalk konnte Sprache aus Beispieltexten lernen [SR86]. Unter dem Namen *Konnektionismus* wurde eine neue Unterdisziplin der KI geboren.

Der Konnektionismus boomte und die Subventionen flossen. Aber bald ist auch hier die Machbarkeit gegeben Grenzen wurden offensichtlich. Die neuronalen Netze könnten beeindruckende Fähigkeiten erlangen, aber es war meist nicht möglich, das erlernte Konzept in einfache Formeln oder Formeln zu fassen logische Regeln. Versuche, neuronale Netze mit logischen Regeln oder dem Wissen zu verbinden menschlicher Experten stieß auf große Schwierigkeiten. Darüber hinaus keine zufriedenstellende Lösung zur Strukturierung und Modularisierung der Netzwerke gefunden.

### 1.2.4 Argumentation unter Unsicherheit

KI als praktische, zielgerichtete Wissenschaft suchte nach einem Ausweg aus dieser Krise. Eins wollte die Fähigkeit der Logik, Wissen explizit darzustellen, mit der Stärke neuronaler Netze im Umgang mit Unsicherheit vereinen. Es wurden mehrere Alternativen vorgeschlagen.

Die vielversprechendste *probabilistische Argumentation* arbeitet mit bedingten Wahrscheinlichkeiten für Aussagenkalkülformeln. Seitdem wurden viele Diagnose- und Expertensysteme für Probleme des alltäglichen Denkens mithilfe von *Bayes'schen Netzen entwickelt.* Der Erfolg Bayes'scher Netzwerke beruht auf ihrer intuitiven Verständlichkeit

saubere Semantik der bedingten Wahrscheinlichkeit und aus der jahrhundertealten, mathematisch fundierten Wahrscheinlichkeitstheorie.

Die Schwächen der Logik, die nur mit zwei Wahrheitswerten arbeiten kann, können sein gelöst durch *Fuzzy-Logik,* die pragmatisch unendlich viele Werte zwischen Null und Eins einführt. Auch wenn seine theoretischen Grundlagen bis heute nicht völlig gesichert sind, Insbesondere in der Regelungstechnik wird es erfolgreich eingesetzt.

Ein ganz anderer Weg führte zur erfolgreichen Synthese von Logik und neuronalen Netzen unter dem Namen *Hybridsysteme.* Hierzu wurden beispielsweise neuronale Netze eingesetzt Lernen Sie Heuristiken zur Reduzierung des riesigen kombinatorischen Suchraums bei der Beweisfindung [SE90].

Methoden des Entscheidungsbaumlernens aus Daten funktionieren auch mit Wahrscheinlichkeiten. Systeme wie CART, ID3 und C4.5 können schnell und automatisch sehr genaue Daten erstellen Entscheidungsbäume, die aussagenlogische Konzepte darstellen und dann verwendet werden können als Expertensysteme. Heute gehören sie zu den beliebtesten Techniken des maschinellen Lernens (Abschn. 8.4).

Seit etwa 1990 hat sich *Data Mining* als Teildisziplin der KI in diesem Bereich entwickelt der statistischen Datenanalyse zur Extraktion von Wissen aus großen Datenbanken. Daten Mining bringt keine neuen Techniken für die KI mit sich, sondern führt vielmehr dazu, dass große Datenbanken genutzt werden müssen, um explizites Wissen zu gewinnen. Eine Anwendung mit großem Markt Potenzial liegt in der Steuerung von Werbekampagnen großer Unternehmen auf Basis der Analyse von vielen Millionen Käufen ihrer Kunden. Typischerweise werden maschinelle Lerntechniken wie z als Entscheidungsbaumlernen kommen hier ins Spiel.

### 1.2.5 Verteilte, autonome und lernende Agenten

Verteilte künstliche Intelligenz (DAI) ist seit etwa 20 Jahren ein aktiver Forschungsbereich 1985. Eines seiner Ziele ist der Einsatz paralleler Computer zur Steigerung der Effizienz von Problemlösern. Es stellte sich jedoch heraus, dass dies auf den hohen Rechenaufwand zurückzuführen war Angesichts der Komplexität der meisten Probleme ist der Einsatz „intelligenter“ Systeme vorteilhafter als die Parallelisierung selbst.

Ein ganz anderer konzeptioneller Ansatz ergibt sich aus der Entwicklung autonomer Softwareagenten und Roboter, die wie menschliche Teams kooperieren sollen.
Wie bei den oben genannten Braitenberg-Fahrzeugen gibt es viele Fälle, in denen Ein einzelner Agent ist selbst mit unbegrenzten Ressourcen nicht in der Lage, ein Problem zu lösen. Erst die Zusammenarbeit vieler Akteure führt zum intelligenten Verhalten bzw. zu die Lösung eines Problems. Eine Ameisenkolonie oder eine Termitenkolonie kann sich errichten Gebäude von sehr hoher architektonischer Komplexität, obwohl es keine einzige Ameise gibt versteht, wie das Ganze zusammenpasst. Dies ähnelt der Situation von Bereitstellung von Brot für eine Großstadt wie New York [RN10]. Es gibt keine zentrale Planungsstelle für Brot, sondern Hunderte von Bäckern, die ihre jeweiligen Bedürfnisse kennen Bereiche der Stadt und backen Sie dort die entsprechende Menge Brot.

Der aktive Kompetenzerwerb durch Roboter ist ein spannendes Gebiet der aktuellen Forschung. Dort sind heute beispielsweise Roboter, die selbständig laufen lernen oder verschiedene Aufgaben ausführen motorische Fähigkeiten rund um den Fußball (Kap. 10). Kooperatives Lernen mehrerer Roboter zu Gemeinsam Probleme lösen steckt noch in den Kinderschuhen.

### 1.2.6 KI wird erwachsen

Die oben genannten Systeme, die KI heute anbietet, sind kein Universalrezept, sondern eine Werkstatt mit einer überschaubaren Anzahl an Werkzeugen für ganz unterschiedliche Aufgaben. Die meisten dieser Tools sind gut entwickelt und stehen als fertige Softwarebibliotheken zur Verfügung, oft mit komfortablen Benutzeroberflächen. Die Auswahl des richtigen Tools und dessen sinnvoller Einsatz im Einzelfall bleibt dem KI-Entwickler bzw. Knowledge Engineer überlassen. Wie jedes andere Handwerk erfordert auch dieses eine solide Ausbildung, die dieses Buch fördern

Wie kaum eine andere Wissenschaft ist KI interdisziplinär, denn sie stützt sich auf interessante Erkenntnisse aus so unterschiedlichen Bereichen wie Logik, Operations Research, Statistik, Regelungstechnik, Bildverarbeitung, Linguistik, Philosophie, Psychologie und Neurobiologie. Hinzu kommt das Themengebiet der jeweiligen Bewerbung.
Ein KI-Projekt erfolgreich zu entwickeln ist daher nicht immer so einfach, aber fast immer äußerst spannend.

## 1.3 Agenten

Obwohl der Begriff *„intelligente Agenten“* für die KI nicht neu ist, hat er erst in den letzten Jahren unter anderem durch [RN10] an Bedeutung gewonnen. *Agent* bezeichnet ganz allgemein ein System, das Informationen verarbeitet und aus einer Eingabe eine Ausgabe erzeugt. Diese Wirkstoffe können auf viele verschiedene Arten klassifiziert werden.

In der klassischen Informatik werden vor allem ***Softwareagenten*** eingesetzt (Abb. 1.4 auf Seite 11). In diesem Fall besteht der Agent aus einem Programm, das aus Benutzereingaben ein Ergebnis berechnet.

In der Robotik hingegen werden ***Hardware-Agenten (auch*** Roboter genannt) eingesetzt, die zusätzlich über Sensoren und Aktoren verfügen (Abb. 1.5 auf Seite 11).
Mit den Sensoren kann der Agent seine Umgebung wahrnehmen. Mit den Aktoren führt es Aktionen aus und verändert seine Umgebung.

Hinsichtlich der Intelligenz des Agenten unterscheidet man zwischen ***Reflexagenten,*** die nur auf Eingaben reagieren, und ***Agenten mit Gedächtnis,*** die auch die Vergangenheit in ihre Entscheidungen einbeziehen können. Ein fahrender Roboter beispielsweise, der über seine Sensoren seine genaue Position (und die Zeit) kennt, hat als Reflexagent keine Möglichkeit, seine Geschwindigkeit zu bestimmen. Wenn es jedoch die Position in kurzen, diskreten Zeitschritten speichert, kann es so leicht seine Durchschnittsgeschwindigkeit im vorherigen Zeitintervall berech

Wenn ein Reflexagent von einem deterministischen Programm gesteuert wird, stellt er eine Funktion der Menge aller Eingaben zur Menge aller Ausgaben dar. Ein Agent mit Gedächtnis hingegen ist im Allgemeinen keine Funktion. Warum? (Siehe Übung 1.5 auf Seite 14.)
Reflexagenten sind in Fällen ausreichend, in denen das zu lösende Problem einen Markov-Entscheidungsprozess beinhaltet. Hierbei handelt es sich um einen Prozess, bei dem lediglich der aktuelle Zustand benötigt wird, um die optimale nächste Aktion zu bestimmen (siehe Kap. 10).

Ein mobiler Roboter, der sich in einem Gebäude von Raum 112 zu Raum 179 bewegen soll, führt andere Aktionen aus als ein Roboter, der sich zu Raum 105 bewegen soll. Mit anderen Worten, die Aktionen hängen vom Ziel ab. Solche Agenten werden als *zielorientiert bezeichnet.*

Tabelle 1.1 Meilensteine in der Entwicklung der KI von Gödel bis heute 1931 Der Österreicher Kurt Gödel zeigt, dass in ***der Prädikatenlogik*** **erster Ordnung alle wahren Aussagen ableitbar sind [Göd31a]. In der Logik höherer Ordnung hingegen gibt es wahre Aussagen, die nicht beweisbar sind [Göd31b]. (In [Göd31b] zeigte Gödel, dass die mit den Axiomen der Arithmetik erweiterte Prädikatenlogik unvollständig ist.)**

**1937 Alan Turing weist mit dem Halteproblem auf die Grenzen intelligenter Maschinen hin [Tur37].**

**1943 McCulloch und Pitts modellieren *neuronale Netze* und stellen die Verbindung zu propositionalen Netzen her Logik.**

**1950 Alan Turing definiert maschinelle Intelligenz mit dem *Turing-Test* und schreibt über lernende Maschinen und genetische Algorithmen [Tur50].**

**1951 Marvin Minsky entwickelt eine neuronale Netzwerkmaschine. Mit 3000 Vakuumröhren er simuliert 40 Neuronen.**

**1955 Arthur Samuel (IBM) entwickelt ein Lernschachprogramm, das besser spielt als sein Entwickler [Sam59].**

**1956 McCarthy organisiert eine Konferenz im Dartmouth College. Hier der Name *Artificial Intelligenz* wurde erstmals eingeführt.**

**Newell und Simon von der Carnegie Mellon University (CMU) präsentieren *Logic Theorist,* das erste symbolverarbeitende Computerprogramm [NSS83].**

**1958 McCarthy erfindet am MIT (Massachusetts Institute of Technology) die Hochsprache *LISP.* Er schreibt Programme, die in der Lage sind, sich selbst zu modifizieren.**

**1959 Gelernter (IBM) entwickelt den Geometry Theorem Prover.**

**1961 Der General Problem Solver (GPS) von Newell und Simon imitiert menschliches Denken [NS61].**

**1963 McCarthy gründet das AI Lab an der Stanford University.**

**1965 Robinson erfindet den *Resolutionskalkül* für die Prädikatenlogik [Rob65] (Abschn. 3.5).**

**1966 Weizenbaums Programm Eliza führt Dialoge mit Menschen in natürlicher Sprache [Wei66] (Abschn. 1.1.2).**

**1969 zeigen Minsky und Papert in ihrem Buch *Perceptrons* , dass das Perzeptron ein sehr einfaches ist neuronales Netz, kann nur lineare Funktionen darstellen [MP69] (Abschn. 1.1.2).**

**1972 Der französische Wissenschaftler Alain Colmerauer erfindet die logische Programmiersprache *PROLOG* (Kap. 5).**

**Der britische Arzt de Dombal entwickelt ein *Expertensystem* zur Diagnose akuter Bauchschmerzen [dDLS+72]. Dies bleibt in der damaligen Mainstream-KI-Community unbemerkt (Abschn. 7.3).**

**1976 Shortliffe und Buchanan entwickeln MYCIN, ein Expertensystem zur Diagnose von Infektionskrankheiten, das in der Lage ist, mit Unsicherheit umzugehen (Kap. 7).**

**1981 Japan startet mit großem Aufwand das „Projekt der fünften Generation“ mit dem Ziel Aufbau einer leistungsstarken PROLOG-Maschine.**

**1982 R1, das Expertensystem zur Konfiguration von Computern, spart Digital Equipment Corporation 40 Millionen Dollar pro Jahr [McD82].**

**1986 Renaissance neuronaler Netze unter anderem durch Rumelhart, Hinton und Sejnowski [RM86]. Das System Nettalk lernt, Texte laut vorzulesen [SR86] (Kap. 9).**

**1990 Pearl [Pea88], Cheeseman [Che85], Whittaker, Spiegelhalter bringen Wahrscheinlichkeitstheorie mit *Bayes'schen Netzwerken* in die KI (Abschn. 7.4). Multiagentensysteme erfreuen sich zunehmender Beliebtheit.**

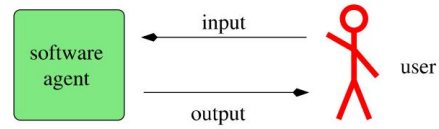
**1992 Tesauros TD-Gammon-Programm demonstriert die Vorteile der Verstärkung Lernen.**

**1993 Weltweite *RoboCup*- Initiative zum Bau fußballspielender autonomer Roboter [Roba].**

Tabelle 1.1 (Fortsetzung)

| |
|---|
| 1995 Ausgehend von der statistischen Lerntheorie entwickelt Vapnik Support-Vektor-Maschinen, die heute eine große Bedeutung haben. |
| 1997 IBMs Schachcomputer Deep Blue besiegt den Schachweltmeister Gary Kasparov. |
| Erster internationaler RoboCup-Wettbewerb in Japan. |
| 2003 Die Roboter im RoboCup zeigen eindrucksvoll, wozu KI und Robotik fähig sind erreichen. |
| 2006 Servicerobotik wird zu einem wichtigen KI-Forschungsgebiet. |
| 2010 Autonome Roboter beginnen, ihre Richtlinien zu erlernen. |
| 2011 IBMs Programm „Watson“ zum Verständnis natürlicher Sprache und zur Beantwortung von Fragen besiegt zwei menschliche Champions in der US-Fernsehquizshow „Jeopardy!“. (Abschn. 1.4). |

Abb. 1.4 Ein Softwareagent mit Benutzerinteraktion



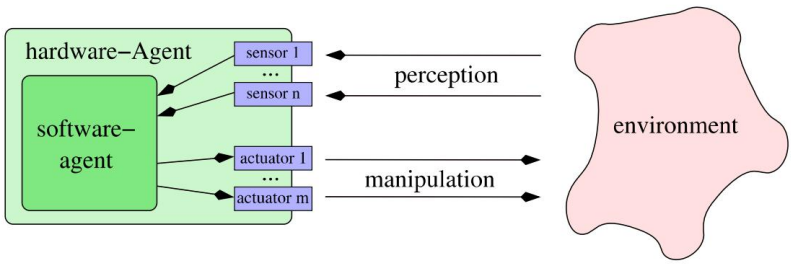




Abb. 1.5 Ein Hardware-Agent

*Beispiel 1.1* Ein Spamfilter ist ein Agent, der eingehende E-Mails in gewünschte oder unerwünschte (Spam-)Kategorien einordnet und alle unerwünschten E-Mails löscht. Sein Ziel als zielorientierter Agent besteht darin, alle E-Mails in die richtige Kategorie einzuordnen. Bei dieser gar nicht so einfachen Aufgabe können dem Agenten gelegentlich Fehler unterlaufen. Da das Ziel darin besteht, alle E-Mails korrekt zu klassifizieren, wird versucht, so wenig Fehler wie möglich zu machen. Allerdings ist das nicht immer das, was der Benutzer im Sinn hat. V Von 1.000 E-Mails macht Agent 1 nur 12 Fehler. Agent 2 hingegen macht bei denselben 1.000 E-Mails 38 Fehler. Ist es also schlechter als Agent 1? Die Fehler beider Agenten werden in der folgenden Tabelle, der sogenannten „Verwirrungsmatrix“, detaillierter dargestellt:

**Agent 1:**

| | | richtige Klasse | |
|---|---|---|---|
| | | gesuchter | Spam |
| Der Spamfilter entscheidet | gesucht | 189 | 1 |
| | Spam | 11 | 799 |

**Agent 2:**

| | | richtige Klasse | |
|---|---|---|---|
| | | gesuchter | Spam |
| Der Spamfilter entscheidet | gesucht | 200 | 38 |
| | Spam | 0 | 762 |

Tatsächlich macht Agent 1 weniger Fehler als Agent 2, aber diese wenigen Fehler sind schwerwiegend, da der Benutzer 11 potenziell wichtige E-Mails verliert. Da es sich hierbei um zwei Arten von Fehlern unterschiedlicher Schwere handelt, sollte jeder Fehler mit dem entsprechenden Kostenfaktor gewichtet werden (siehe Abschn. 7.3.5 und Aufgabe 1.7 auf Seite 14).

Die Summe aller gewichteten Fehler ergibt die durch Fehlentscheidungen verursachten Gesamtkosten. Das Ziel eines *kostenbasierten Agenten* besteht darin, die Kosten von Fehlentscheidungen langfristig, also im Durchschnitt, zu minimieren. In Abschn. Abschn. 7.3 lernen wir das Diagnosesystem LEXMED als Beispiel für einen kostenbasierten Agenten kennen.

Analog dazu besteht das Ziel eines nutzenbasierten Agenten darin, den Nutzen richtiger Entscheidungen langfristig, also im Durchschnitt, zu maximieren. Die Summe aller Entscheidungen gewichtet mit ihren jeweiligen Nutzenfaktoren ergibt den Gesamtnutzen.

Von besonderem Interesse in der KI sind *lernende Agenten,* die in der Lage sind, sich anhand von Trainingsbeispielen oder durch positives oder negatives Feedback so zu verändern, dass der durchschnittliche Nutzen ihrer Handlungen mit der Zeit wächst (siehe Kap. 8).

Wie in Abschn. 1.2.5 kommen zunehmend *verteilte Agenten* zum Einsatz, deren Intelligenz nicht in einem Agenten lokalisiert ist, sondern erst durch die Zusammenarbeit vieler Agenten sichtbar wird.

Das Design eines Agenten orientiert sich neben seinem Ziel stark an seiner *Umgebung* bzw. an seinem Bild der Umgebung, das stark von seiner Sensorik abhängt. Die Umgebung ist *beobachtbar ,* wenn der Agent stets den vollständigen Zustand der Welt kennt. Ansonsten ist die Umgebung nur *teilweise beobachtbar.*
Wenn eine Aktion immer zum gleichen Ergebnis führt, dann ist die Umgebung *deterministisch.* Ansonsten ist es *nichtdeterministisch.* In einer *diskreten Umgebung* treten nur endlich viele Zustände und Aktionen auf, wohingegen eine *kontinuierliche Umgebung* unendlich viele Zustände oder Aktionen aufweist.

## 1.4 Wissensbasierte Systeme

Ein Agent ist ein Programm, das eine Abbildung von Wahrnehmungen auf Handlungen durchführt. Für einfache Agenten ist diese Sichtweise des Problems ausreichend. Bei komplexen Anwendungen, bei denen der Agent auf eine große Menge an Informationen zurückgreifen muss und eine schwierige Aufgabe erfüllen soll, kann die Programmierung des Agenten sehr kostspielig und unklar sein, wie vorzugehen ist. Hier bietet KI einen klaren Weg, der die Arbeit erheblich vereinfachen wird.

Zunächst trennen wir *das Wissen* vom System oder Programm, das das Wissen beispielsweise nutzt, um Schlussfolgerungen zu ziehen, Fragen zu beantworten oder einen Plan zu entwickeln. Dieses System wird als *Inferenzmechanismus bezeichnet.* Das Wissen wird in einer *Wissensdatenbank (KB) gespeichert.* Der Erwerb von Wissen in der Wissensbasis wird als *Knowl Edge Engineering* bezeichnet und basiert auf verschiedenen Wissensquellen wie menschlichen Experten, dem Wissensingenieur und Datenbanken. Aktiv lernende Systeme können Wissen auch durch aktive Erkundung der Welt erwerben (siehe Kap. 10). In Abb. 1.6 auf Seite 13 wird die allgemeine Architektur wissensbasierter Systeme dargestellt.

Der Übergang zu einer Trennung von Wissen und Schlussfolgerung hat mehrere entscheidende Vorteile. Durch die Trennung von Wissen und Inferenz können Inferenzsysteme weitgehend anwendungsunabhängig implementiert werden. Es ist zum Beispiel viel

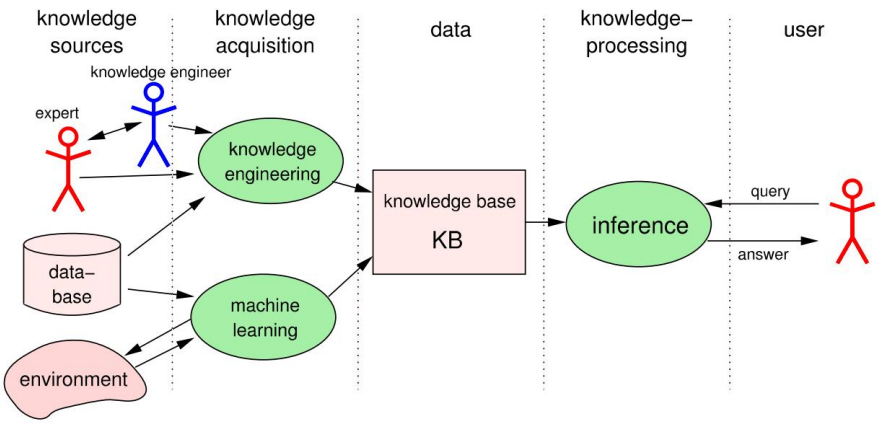



**Abb. 1.6 Struktur eines klassischen Wissensverarbeitungssystems**

**Es ist einfacher, die Wissensbasis eines medizinischen Expertensystems zu ersetzen, als ein zu programmieren völlig neues System.**

**Durch die Entkopplung der Wissensbasis von der Schlussfolgerung kann Wissen deklarativ gespeichert werden. In der Wissensdatenbank gibt es nur eine Beschreibung davon Wissen, das unabhängig vom verwendeten Inferenzsystem ist. Ohne das klare Trennung, Wissen und Verarbeitung von Inferenzschritten würden miteinander verwoben, und jede Änderung des Wissens wäre sehr kostspielig.**

**Als praktische Schnittstelle zwischen Mensch und Maschine bietet sich die formale Sprache an zur Darstellung von Wissen in der Wissensdatenbank. In den folgenden Kapiteln werden wir eine ganze Reihe solcher Sprachen kennenlernen. Zuerst in Kap. 2 und 3**
**Es gibt *Aussagenlogik* und *Prädikatenlogik erster Ordnung* (PL1). Aber auch andere Malismen wie probabilistische Logik, Fuzzy-Logik oder Entscheidungsbäume werden vorgestellt.**
**Wir beginnen mit der Aussagenrechnung und den zugehörigen Folgerungssystemen. Aufbauend auf Daraufhin stellen wir die Prädikatenlogik vor, eine mächtige Sprache, die für Maschinen zugänglich und in der KI sehr wichtig ist.**

**Als Beispiel für ein groß angelegtes wissensbasiertes System möchten wir auf Folgendes verweisen Software-Agent „Watson“. Entwickelt bei IBM zusammen mit mehreren Universitäten, Watson ist ein Frage-Antwort-Programm, das mit natürlichen Hinweisen gefüttert werden kann Sprache. Es basiert auf einer Wissensbasis, die vier Terabyte Festplattenspeicher umfasst, darunter auch den Volltext von Wikipedia [FNA+09]. Watson wurde innerhalb entwickelt Das DeepQA-Projekt von IBM, das in [Dee11] wie folgt charakterisiert wird:**

> **Das DeepQA-Projekt bei IBM stellt eine große Herausforderung in der Informatik dar, die darauf abzielt veranschaulichen, wie die umfassende und zunehmende Zugänglichkeit von Inhalten in natürlicher Sprache und die Integration und Weiterentwicklung der Verarbeitung natürlicher Sprache, des Informationsabrufs, des maschinellen Lernens, der Wissensrepräsentation und des Denkens sowie massiv paralleler Berechnungen die automatische Frage-Antwort-Technologie im offenen Bereich vorantreiben können Pun es konkurriert klar und konsequent mit der besten menschlichen Leistung.**

**In der US-Fernsehquizshow „Jeopardy!“ besiegte Watson im Februar 2011 die beiden menschlichen Champions Brad Rutter und Ken Jennings in einem Zwei-Spiele-Spiel.**
**Kombiniertes Punktespiel und gewann den Preis von einer Million Dollar. Einer von Watsons Kollegen**

Besondere Stärken waren die sehr schnelle Reaktion auf die Fragen, was dazu führte, dass Watson den Summer (mittels eines Magnetventils) oft schneller als seine menschlichen Konkurrenten betätigte und dann die erste Antwort auf die Frage geben konnte.

Die hohe Leistung und die kurzen Reaktionszeiten von Watson sind auf eine Implementierung auf 90 IBM Power 750-Servern zurückzuführen, die jeweils 32 Prozessoren enthalten, was zu 2880 parallelen Prozessoren führt.

## 1.5 Übungen

Übung 1.1 Testen Sie einige der im Internet verfügbaren Chatterbots. Beginnen Sie zum Beispiel mit www.hs-weingarten.de/~ertel/aibook in der Linksammlung unter Turingtest/Chatterbots, oder unter www.simonlaven.com oder www.alicebot.org. Schreiben Sie eine Ausgangsfrage auf und messen Sie die Zeit, die für jedes der verschiedenen Programme benötigt wird, bis Sie sicher wissen, dass es sich nicht um einen Menschen handelt.

ÿ ÿ Übung 1.2 Unter www.pandorabots.com finden Sie einen Server, auf dem Sie ganz einfach einen Chatterbot mit der Auszeichnungssprache AIML erstellen können. Entwickeln Sie je nach Interesse einen einfachen oder komplexen Chatterbot oder ändern Sie einen bestehenden.

Aufgabe 1.3 Begründen Sie die Ungeeignetheit des Turing-Tests als Definition von *„künstlicher Intelligenz“* in der praktischen KI.

ÿ Übung 1.4 Viele bekannte Inferenzprozesse, Lernprozesse usw. sind NP-vollständig oder sogar unentscheidbar. Was bedeutet das für KI?

Aufgabe 1.5 (a)

Warum ist ein deterministischer Agent mit Gedächtnis im mathematischen Sinne keine Funktion von der Menge aller Eingaben zur Menge aller Ausgaben? (b) Wie kann

man den Agenten mit dem Gedächtnis verändern oder ihn so modellieren, dass er wird? Äquivalent zu einer Funktion, verliert aber nicht ihr Gedächtnis?

Übung 1.6 Es gebe einen Agenten mit Gedächtnis, der sich innerhalb einer Ebene bewegen kann. Von seinen Sensoren erhält es in regelmäßigen Takten t seine genaue Position (x, y) in kartesischen Koordinaten. (a) Geben Sie

eine Formel an, mit der der Agent seine Geschwindigkeit aus der Strömung berechnen kann Zeit t und die vorherige Messung von t ÿ t.

(b) Wie muss der Agent verändert werden, damit er auch seine Beschleunigung berechnen kann? Geben Sie auch hier eine Formel an.

ÿ Übung 1.7

(a) Bestimmen Sie für beide Agenten in Beispiel 1.1 auf Seite 11 die durch die Fehler verursachten Kosten und vergleichen Sie die Ergebnisse. Gehen Sie hier davon aus, dass das manuelle Löschen einer Spam-E-Mail einen Cent kostet und das Wiederherstellen einer gelöschten E-Mail oder der Verlust einer E-Mail einen Dollar kostet.

(b) Bestimmen Sie für beide Agenten den Gewinn, der durch korrekte Klassifizierungen entsteht, und vergleichen Sie die Ergebnisse. Gehen Sie davon aus, dass für jede gewünschte erkannte E-Mail ein Gewinn von einem Dollar anfällt und für jede korrekt gelöschte Spam-E-Mail ein Gewinn von einem Cent.

# Aussagelogik

# 2

In der Aussagenlogik werden, wie der Name schon sagt, Aussagen durch logische Operatoren verknüpft. Die Aussage *„die Straße ist nass“* ist eine Aussage, ebenso wie *„es regnet“.* Diese beiden Sätze können zu dem neuen Satz verbunden werden

*Wenn es regnet, ist die Straße nass.*

Formeller geschrieben

*es regnet* ÿ *die Straße ist nass.*

Diese Notation hat den Vorteil, dass die Elementarsätze in unveränderter Form wieder auftauchen. Damit wir präzise mit Aussagenlogik arbeiten können, beginnen wir mit einer Definition der Menge aller Aussagenlogikformeln.

## 2.1 Syntax

**Definition 2.1** Sei *Op* = {¬,ÿ,ÿ,ÿ, ÿ, (, )} die Menge der logischen Operatoren und ÿ eine Menge der Symbole. Die Mengen *Op,* ÿ und {t,f } sind paarweise disjunkt. ÿ heißt *Signatur* und seine Elemente sind die *Satzvariablen.* Die Menge der aussagenlogischen Formeln ist nun rekursiv definiert: • t und f sind (atomare) Formeln. • Alle Satzvariablen, also alle Elemente aus ÿ, sind (atomare) Formeln las.

• Wenn A und B Formeln sind, dann sind ¬A, (A), A ÿ B, A ÿ B, A ÿ B, A ÿ B ebenfalls Formeln.

Diese elegante rekursive Definition der Menge aller Formeln ermöglicht uns die Generierung unendlich viele Formeln. Zum Beispiel gegeben ÿ = {A,B,C},

A ÿ B, A ÿ B ÿ C, A ÿ A ÿ A, C ÿ B ÿ A, (¬A ÿ B) ÿ (¬C ÿ A)

sind Formeln. (((A)) ÿ B) ist ebenfalls eine syntaktisch korrekte Formel.

W. Ertel, *Einführung in die künstliche Intelligenz,*
Bachelor-Themen in Informatik, DOI
10.1007/978-0-85729-299-5_2, 

**Definition 2.2 Wir lesen die Symbole und Operatoren folgendermaßen:**

**t: „wahr“**
**f: „falsch“**
**¬A: „nicht A“ (Negation)**
**A ÿ B: „A und B“ (Verbindung)**
**A ÿ B: „A oder B“ (Disjunktion)**
**A ÿ B: „Wenn A, dann B“ (Implikation (auch *materielle Implikation genannt))***
**A ÿ B: „A genau dann, wenn B“ (Äquivalenz)**

**Die so definierten Formeln sind bisher rein syntaktische Konstruktionen ohne Bedeutung. Uns fehlt noch die Semantik.**

## 2.2 Semantik

**In der Aussagenlogik gibt es zwei Wahrheitswerte: t für „wahr“ und f für „falsch“. Wir Beginnen Sie mit einem Beispiel und fragen Sie sich, ob die Formel A ÿ B wahr ist. Der Die Antwort lautet: Es kommt darauf an, ob die Variablen A und B wahr sind. Wenn zum Beispiel A steht für *„Heute regnet es“* und B für *„Heute ist es kalt“* und beides ist wahr, dann ist Aÿ B wahr. Wenn B jedoch *„Heute ist es heiß“* darstellt (und das ist falsch), dann A ÿ B ist falsch.**

**Wir müssen dem Propo offensichtlich Wahrheitswerte zuweisen, die den Zustand der Welt widerspiegeln Positionsvariablen. Deshalb definieren wir**

**Definition 2.3 Eine Abbildung I : ÿ ÿ {w, f }, die einen Wahrheitswert zuordnet Jede Aussagevariable wird *Interpretation genannt.***

**Da jede Aussagevariable zwei Wahrheitswerte annehmen kann, hat jede aussagenlogische Formel mit n verschiedenen Variablen 2n verschiedene Interpretationen. Wir Definieren Sie die Wahrheitswerte für die Grundoperationen, indem Sie alle möglichen Interpretationen in einer *Wahrheitstabelle* darstellen (siehe Tabelle 2.1 auf Seite 17).**

**Die leere Formel gilt für alle Interpretationen. Um die Wahrheit herauszufinden Für komplexe Formeln müssen wir auch die Reihenfolge der Operationen für logische Werte definieren Betreiber. Wenn Ausdrücke in Klammern gesetzt sind, wird der Term in den Klammern ausgewertet**

**Tabelle 2.1 Definition der logischen Operatoren durch Wahrheitstabelle**

| AB | (A) | ¬A | A ÿ B | A ÿ B | A ÿ B | A ÿ B |
|---|---|---|---|---|---|---|
| ttt | | ft | | T | T | T |
| t ft | | ff | | T | F | F |
| ft | | ft | F | T | T | F |
| fff | | t | F | F | T | T |

Erste. Bei Formeln ohne Klammern sind die Prioritäten wie folgt geordnet, beginnend mit der stärksten Bindung: ¬,ÿ,ÿ,ÿ, ÿ.

Klare Unterscheidung zwischen der Äquivalenz von Formeln und der syntaktischen Äquivalenz Alence, wir definieren

**Definition 2.4** Zwei Formeln F und G heißen semantisch äquivalent, wenn sie nehmen für alle Interpretationen den gleichen Wahrheitswert an. Wir schreiben F ÿ G.

Semantische Äquivalenz dient vor allem dazu, die Metasprache nutzen zu können ist, natürliche Sprache, um über die Objektsprache zu sprechen, nämlich Logik. Die Aussage „A ÿ B“ vermittelt, dass die beiden Formeln A und B semantisch äquivalent sind. Der Die Aussage „A ÿ B“ hingegen ist ein syntaktisches Objekt der formalen Sprache der Aussagenlogik.

Je nachdem, in wie vielen Interpretationen eine Formel wahr ist, können wir einteilen Formeln in die folgenden Klassen:

**Definition 2.5** Eine Formel heißt

- *Erfüllbar,* wenn es für mindestens eine Interpretation wahr ist.
- *Logisch gültig* oder einfach *gültig ,* wenn es für alle Interpretationen wahr ist. Zwar werden Mulas auch *Tautologien genannt.*
- *Unerfüllbar,* wenn es für keine Interpretation wahr ist.

Jede Interpretation, die einer Formel genügt, wird als *Modell* der Formel bezeichnet.

Offensichtlich ist die Negation jeder allgemeingültigen Formel unerfüllbar. Die Nega tion einer erfüllbaren, aber nicht allgemein gültigen Formel F ist erfüllbar.

Wir sind nun in der Lage, Wahrheitstabellen für komplexe Formeln zu erstellen, um deren Wahrheitsgehalt zu ermitteln Wahrheitswerte. Wir setzen dies mithilfe von Formeläquivalenzen sofort in die Tat um die in der Praxis wichtig sind.

**Satz 2.1** *Die Operationen* ÿ,ÿ *sind kommutativ und assoziativ, und die Folgende Äquivalenzen gelten grundsätzlich:*

| | |
|---|---|
| ¬A ÿ B ÿ A ÿ B | *(Implikation)* |
| A ÿ B ÿ ¬B ÿ ¬A | *(Kontraposition)* |
| (A ÿ B) ÿ (B ÿ A) ÿ (A ÿ B) ¬(A ÿ B) ÿ ¬A ÿ ¬B | *(Gleichwertigkeit)* |
| ¬(A ÿ B) ÿ ¬A ÿ ¬B | *(De Morgans Gesetz)* |

A ÿ (B ÿ C) ÿ (A ÿ B) ÿ (A ÿ C) *(Distributivgesetz)*
A ÿ (B ÿ C) ÿ (A ÿ B) ÿ (A ÿ C)

| | |
|---|---|
| A ÿ ¬A ÿ w | *(Tautologie)* |
| A ÿ ¬A ÿ f | *(Widerspruch)* |
| A ÿ f ÿ A | |
| A ÿ w ÿ w | |
| A ÿ f ÿ f | |
| A ÿ w ÿ A | |

***Beweis*** Um die erste Äquivalenz zu zeigen, berechnen wir die Wahrheitstabelle für ¬A ÿ B und A ÿ B und stellen Sie sicher, dass die Wahrheitswerte für beide Formeln für alle Interpretationen gleich sind. Die Formeln sind also äquivalent und somit auch alle Werte der letzte Spalte sind „t“s.

| AB | ¬A | ¬A ÿ BA | ÿ B | (¬A ÿ B) | ÿ (A ÿ B) |
|---|---|---|---|---|---|
| tt | f | t | t | t | t |
| tf | f | f | t | f | t |
| ft | t | t | f | t | t |
| ff | t | t | t | t | t |

Die Beweise für die anderen Äquivalenzen sind ähnlich und werden als Übungen empfohlen für den Leser (Übung 2.2 auf Seite 29).

## 2.3 Beweissysteme

Bei der KI geht es uns darum, vorhandenes Wissen zu übernehmen und daraus Neues abzuleiten Wissen oder die Beantwortung von Fragen. In der Aussagenlogik bedeutet das, das zu zeigen eine *Wissensbasis* KB – also eine (möglicherweise umfangreiche) Aussagenlogikformel – Es folgt eine Formel Q1 . Daher definieren wir zunächst den Begriff „Folge“.

1Hier steht Q für Abfrage.

**Definition 2.6 Eine Formel *KB beinhaltet* eine Formel Q (oder Q *folgt* aus *KB),* wenn jedes Modell von *KB* auch ein Modell von Q ist. Wir schreiben *KB* | Q.**

**Mit anderen Worten: In jeder Interpretation, in der *KB* wahr ist, ist auch Q wahr. Kurz gesagt: Wann immer *KB* wahr ist, ist auch Q wahr. Da für den Begriff der Folgerung Interpretationen von Variablen herangezogen werden, handelt es sich um einen semantischen Begriff.**

**Jede Formel, die nicht gültig ist, wählt sozusagen eine Teilmenge der Menge aller Interpretationen als ihr Modell. Tautologien wie A ÿ ¬A beispielsweise schränken die Zahl der zufriedenstellenden Interpretationen nicht ein, da ihr Satz leer ist. Die leere Formel gilt daher in allen Interpretationen. Für jede Tautologie T gilt ÿ | T Intuitiv bedeutet dies, dass Tautologien immer wahr sind, ohne Einschränkung der Interpretationen durch eine Formel. Kurz gesagt schreiben wir | T . Nun zeigen wir einen wichtigen Zusammenhang zwischen dem semantischen Konzept der Folgerung und der syntaktischen Implikation.**

**Satz 2.2 (Deduktionstheorem)**

**A | B *genau dann, wenn* | A ÿ B.**

***Beweis* Beachten Sie die Wahrheitstabelle zur Implikation:**

| ABA ÿ B |
|---|
| ttt |
| t ff ft t fft |
| |
| |

**Eine willkürliche Implikation A ÿ B ist offensichtlich immer wahr, außer bei der Interpretation A ÿ t,B ÿ f . Nehmen Sie an, dass A | B hält. Das bedeutet, dass für jede Interpretation, die A wahr macht, auch B wahr ist. Die kritische zweite Zeile der Wahrheitstabelle gilt in diesem Fall nicht einmal. Daher ist A ÿ B wahr, was bedeutet, dass A ÿ B eine Tautologie ist. Damit ist eine Richtung der Aussage aufgezeigt.**

**Nehmen Sie nun an, dass A ÿ B gilt. Somit ist die kritische zweite Zeile der Wahrheitstabelle auch ausgesperrt. Jedes Modell von A ist dann auch ein Modell von B. Dann ist A | B hält.**

**Wenn wir zeigen wollen, dass *KB* Q mit sich bringt, können wir auch mit der Wahrheitstabellenmethode zeigen, dass *KB* ÿ Q eine Tautologie ist. Damit haben wir unser erstes *Beweissystem* für Aussagenlogik, das leicht zu automatisieren ist. Der Nachteil dieser Methode ist die im schlimmsten Fall sehr lange Rechenzeit. Konkret muss im schlimmsten Fall mit n Aussagevariablen für alle 2n Interpretationen der Variablen die Formel *KB* ÿ Q ausgewertet werden. Die Rechenzeit wächst daher exponentiell**

mit der Anzahl der Variablen. Daher ist dieses Verfahren zumindest im schlimmsten Fall für große Variablenzahlen unbrauchbar.

Wenn eine Formel *KB* eine Formel Q mit sich bringt, dann ist nach dem Deduktionssatz *KB* ÿ Q eine Tautologie. Daher ist die Negation ¬*(KB* ÿ Q) unerfüllbar. Wir haben

¬*(KB* ÿ Q) ÿ ¬(¬KB ÿ Q) ÿ *KB* ÿ ¬Q.

Daher ist auch *KB* ÿ ¬Q erfüllbar. Wir formulieren diese einfache, aber wichtige Konsequenz des Deduktionssatzes als Satz.

**Satz 2.3 (Beweis durch Widerspruch)** *KB* | Q *genau dann, wenn KB* ÿ ¬Q *unerfüllbar ist.*

Um zu zeigen, dass die Abfrage Q aus der Wissensbasis *KB folgt,* können wir auch die negierte Abfrage ¬Q zur Wissensbasis hinzufügen und einen Widerspruch ableiten. Aufgrund der Äquivalenz A ÿ ¬A ÿ f aus Satz 2.1 auf Seite 18 wissen wir, dass ein Widerspruch unerfüllbar ist. Damit ist Q bewiesen. Dieses in der Mathematik häufig verwendete Verfahren wird auch in verschiedenen automatischen Beweiskalkülen wie der Auflösungsrechnung und bei der Abarbeitung von PROLOG-Programmen eingesetzt.

Eine Möglichkeit, zu vermeiden, dass alle Interpretationen mit der Wahrheitstabellenmethode überprüft werden müssen, ist die syntaktische Manipulation der Formeln *KB* und Q durch Anwendung von Inferenzregeln mit dem Ziel, sie stark zu vereinfachen, sodass wir am Ende sofort erkennen können, dass *KB* | F. Wir nennen diesen syntaktischen Prozess *Ableitung* und schreiben *KB* ÿ Q. Solche syntaktischen Beweissysteme werden *Kalküle genannt.* Um sicherzustellen, dass ein Kalkül keine Fehler erzeugt, definieren wir zwei grundlegende Eigenschaften von Ka

**Definition 2.7** Ein Kalkül heißt *Klang ,* wenn jeder abgeleitete Satz semantisch folgt. Das heißt, wenn für die Formeln *KB* und Q das gilt

wenn *KB* ÿ Q, dann *KB* | Q.

Ein Kalkül heißt *vollständig ,* wenn alle semantischen Konsequenzen abgeleitet werden können. Das heißt, für die Formeln *KB* und Q gilt:

wenn *KB* | Q dann *KB* ÿ Q.

Die Solidität eines Kalküls stellt sicher, dass alle abgeleiteten Formeln tatsächlich semantische Konsequenzen der Wissensbasis sind. Das Kalkül erzeugt keine „falschen Konsequenzen“. Die Vollständigkeit eines Kalküls hingegen stellt sicher, dass der Kalkül nichts übersieht. Ein vollständiger Kalkül findet immer dann einen Beweis, wenn die zu beweisende Formel aus der Wissensbasis folgt. Wenn eine Rechnung korrekt ist

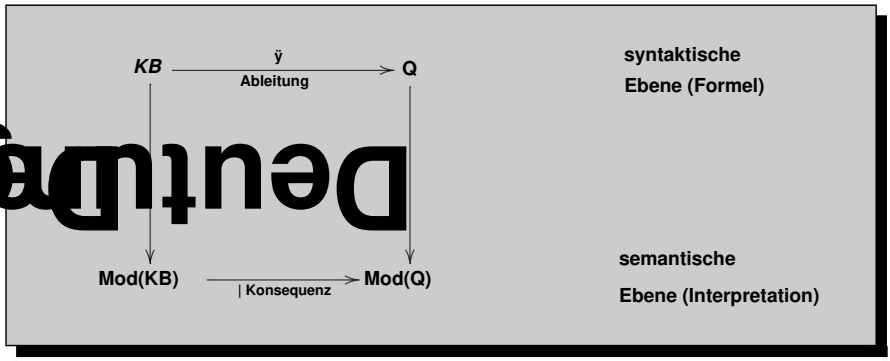



**Abb. 2.1 Syntaktische Ableitung und semantische Folgerung. Mod(X) stellt die Menge der Modelle einer Formel X dar**

**und vollständig, dann sind syntaktische Ableitung und semantische Folgerung zwei äquivalente Beziehungen (siehe Abb. 2.1).**

**Um automatische Proofsysteme möglichst einfach zu halten, werden diese üblicherweise hergestellt Formeln in konjunktiver Normalform bearbeiten.**

Definition 2.8 Eine Formel liegt **genau dann in der *konjunktiven Normalform (KNF)*** vor, wenn sie aus einer ***Konjunktion*** besteht

**K1 ÿ K2 ÿ···ÿ Km**

**von Klauseln. Eine Klausel Ki besteht aus einer *Disjunktion***

**(Li1 ÿ Li2 ÿ···ÿ Lini )**

**von Literalen. Schließlich ist ein *Literal* eine Variable (positives Literal) oder eine negierte Variable (negatives Literal).**

**Die Formel (A ÿ B ÿ ¬C) ÿ (A ÿ B) ÿ (¬B ÿ ¬C) liegt in konjunktiver Normalform vor. Die konjunktive Normalform stellt keine Einschränkung für die Formelmenge dar, weil:**

**Satz 2.4 *Jede aussagenlogische Formel lässt sich in eine äquivalente konjunktive Normalform umwandeln.***

***Beispiel 2.1*** **Wir bringen A ÿ B ÿ C ÿ D in die konjunktive Normalform, indem wir die Äquivalenzen aus Satz 2.1 auf Seite 18 verwenden:**

**A ÿ B ÿ C ÿ D ÿ ¬(A**
**ÿ B) ÿ (C ÿ D) (Implikation) ÿ (¬A ÿ ¬B) ÿ (C ÿ D) (de Morgan) ÿ (¬A ÿ (C ÿ D )) ÿ**
**(¬B ÿ (C ÿ D)) (Distributivgesetz) ÿ ((¬A ÿ C) ÿ (¬A ÿ D)) ÿ ((¬B ÿ C) ÿ (¬B ÿ D))**
**( Verteilungsgesetz) ÿ (¬A ÿ C) ÿ (¬A ÿ D) ÿ (¬B ÿ C) ÿ (¬B ÿ D) (Assoziativgesetz)**

**Jetzt fehlt uns nur noch ein Kalkül zum syntaktischen Beweis aussagenlogischer Formeln. Wir beginnen mit dem *Modus Ponens,* einer einfachen, intuitiven Schlussregel, die aus der Gültigkeit von A und A ÿ B die Ableitung von B ermöglicht. Wir schreiben dies formal als**

$$\frac{A,\ A \ddot{y} B}{B}.$$

**Diese Notation bedeutet, dass wir die Formel(n) unterhalb der Zeile aus den durch Kommas getrennten Formeln oberhalb der Zeile ableiten können. Modus ponens ist in der Regel für sich genommen zwar gesund, aber nicht vollständig. Wenn wir zusätzliche Regeln hinzufügen, können wir eine vollständige Rechnung erstellen, auf die wir hier jedoch nicht eingehen möchten.** Stattdessen werden wir die ***Auflösungsregel*** **untersuchen**

$$\frac{A \ddot{y} B,\ \neg B \ddot{y} C}{A \ddot{y} C} \tag{2.1}$$

**als Alternative. Der abgeleitete Satz heißt *resolvent.* Durch eine einfache Transformation erhalten wir die äquivalente Form**

$$\frac{A \ddot{y} B,\ B \ddot{y} C}{A \ddot{y} C}.$$

**Wenn wir A auf f setzen, sehen wir, dass die Auflösungsregel eine Verallgemeinerung des Modus Ponens ist. Die Auflösungsregel ist gleichermaßen anwendbar, wenn C fehlt oder wenn A und C fehlen. Im letzteren Fall lässt sich der Leersatz aus dem Widerspruch B ÿ ¬B ableiten (Übung 2.7 auf Seite 30).**

## 2.4 Auflösung

**Wir verallgemeinern nun die Auflösungsregel noch einmal, indem wir Klauseln mit einer beliebigen Anzahl von Literalen zulassen. Bei den Literalen A1,...,Am, B, C1,...,Cn lautet die *allgemeine Auflösungsregel***

$$\frac{(A1 \ddot{y} \cdots \ddot{y} Am \ddot{y} B),\ (\neg B \ddot{y} C1 \ddot{y} \cdots \ddot{y} Cn)}{(A1 \ddot{y} \cdots \ddot{y} Am \ddot{y} C1 \ddot{y} \cdots \ddot{y} Cn)}. \tag{2.2}$$

Wir nennen die Literale B und ¬B komplementär. Die Auflösungsregel löscht ein Paar komplementärer Literale aus den beiden Klauseln und kombiniert die restlichen Literale zu einer neuen Klausel.

Um zu beweisen, dass aus einer Wissensbasis *KB* eine Abfrage Q folgt, führen wir einen Widerspruchsbeweis durch. Nach Satz 2.3 auf Seite 20 müssen wir zeigen, dass aus *KB* ÿ ¬Q ein Widerspruch abgeleitet werden kann . In Formeln in konjunktiver Normalform tritt ein Widerspruch in Form zweier Sätze (A) und (¬A) auf, die auf den Leersatz als deren Auflösung führen. Der folgende Satz stellt uns sicher, dass dieser Prozess wirklich wie gewünscht funktioniert.

Damit die Rechnung vollständig ist, benötigen wir eine kleine Ergänzung, wie das folgende Beispiel zeigt. Als Wissensbasis sei die Formel (A ÿ A) gegeben. Um mit der Auflösungsregel zu zeigen, dass wir von dort (A ÿ A) ableiten können, müssen wir zeigen, dass die leere Klausel aus (A ÿ A) ÿ (¬A ÿ ¬A) abgeleitet werden kann. Mit der Auflösungsregel allein ist dies unmöglich. Mit *der Faktorisierung,* die das Löschen von Kopien von Literalen aus Klauseln ermöglicht, wird dieses Problem beseitigt. Im Beispiel führt eine doppelte Anwendung der Faktorisierung zu (A) ÿ (¬A) und ein Auflösungsschritt zur leeren Klausel.

Satz 2.5 *Der Auflösungskalkül für den Beweis der Unerfüllbarkeit von Mulas in konjunktiver Normalform ist solide und vollständig.*

Denn es ist die Aufgabe des Auflösungskalküls, daraus einen Widerspruch abzuleiten *KB* ÿ ¬Q, es ist sehr wichtig, dass die Wissensbasis *KB konsistent* ist :

Definition 2.9 Eine Formel *KB* heißt *konsistent ,* wenn sich daraus kein Widerspruch ableiten lässt, also eine Formel der Form ÿ ÿ ¬ÿ.

Ansonsten lässt sich aus *KB* alles ableiten (siehe Übung 2.8 auf Seite 30). Das gilt nicht nur für die Auflösung, sondern auch für viele andere Kalküle.

Unter den Berechnungen für den automatisierten Abzug spielt die Auflösung eine herausragende Rolle. Daher möchten wir noch etwas intensiver damit zusammenarbeiten. Im Gegensatz zu anderen Kalkülen hat die Auflösung nur zwei Inferenzregeln und funktioniert mit Formeln in konjunktiver Normalform. Dies vereinfacht die Umsetzung. Ein weiterer Vorteil im Vergleich zu vielen Berechnungen liegt in der Reduzierung der Möglichkeiten zur Anwendung von Inferenzregeln in jedem Schritt des Beweises, wodurch der Suchraum reduziert und die Rechenzeit verkürzt wird.

Als Beispiel beginnen wir mit einem einfachen Logikrätsel, das die wichtigen Schritte ermöglicht eines vorzulegenden Auflösungsnachweises.

*Beispiel 2.2* Logikrätsel Nummer 7 mit dem Titel „ Eine *bezaubernde englische Familie“* aus dem deutschen Buch [Ber89] lautet (übersetzt ins Englische):

Obwohl ich sieben Jahre lang mit großem Erfolg Englisch gelernt habe, muss ich zugeben, dass ich völlig perplex bin, wenn ich Engländer Englisch sprechen höre. Kürzlich nahm ich, von edlen Gefühlen bewegt, drei Anhalter mit, einen Vater, eine Mutter und eine Tochter, von denen ich schnell merkte, dass sie Engländer waren und nur Englisch sprachen. Bei jedem der folgenden Sätze schwankte ich zwischen zwei möglichen Interpretationen. Sie sagten mir Folgendes (die zweite mögliche Bedeutung steht in Klammern): Der Vater: „Wir fahren nach Spanien (wir kommen aus Newcastle).“
Die Mutter: „Wir fahren nicht nach Spanien und kommen aus Newcastle (wir haben in Paris Halt gemacht und fahren nicht nach Spanien).“ Die Tochter: „Wir kommen nicht aus Newcastle (wir haben in Paris Halt gemacht).“
Was ist mit dieser bezaubernden englischen Familie?

Um ein solches Problem zu lösen, gehen wir in drei Schritten vor: Formalisierung, Transformation in Normalform und Beweis. In vielen Fällen ist die Formalisierung der mit Abstand schwierigste Schritt, da leicht Fehler gemacht oder kleine Details vergessen werden können. (Daher ist die praktische Übung sehr wichtig. Siehe Übungen 2.9–2.11.)

Hier verwenden wir die Variablen S für *„Wir fahren nach Spanien“,* N für *„Wir kommen aus“. Newcastle“* und P für *„Wir haben in Paris angehalten“* und gelten als Formalisierung der drei Sätze Vater, Mutter und Tochter

$$(S \ddot{y} N) \ddot{y} [(\neg S \ddot{y} N) \ddot{y} (P \ddot{y} \neg S)] \ddot{y} (\neg N \ddot{y} P).$$

Durch Ausklammern von ¬S in der mittleren Unterformel wird die Formel in einem Schritt in CNF übernommen. Die Nummerierung der Klauseln mit tiefgestellten Indizes ergibt

$$KB \ddot{y} (S \ddot{y} N)_1 \ddot{y} (\neg S)_2 \ddot{y} (P \ddot{y} N)_3 \ddot{y} (\neg N \ddot{y} P)_4.$$

Jetzt beginnen wir mit dem Auflösungsbeweis, zunächst noch ohne Abfrage Q. Ein Ausdruck der Form „Res(m, n): Klauselk “ bedeutet, dass Klausel durch Auflösung von Klausel m mit Klausel n erhalten wird und die Nummer k hat.

Res(1, 2): $(N)_5$

Res(3, 4): $(P)_6$

Res(1, 4): $(S \ddot{y} P)_7$

Wir hätten Klausel P auch aus Res(4, 5) oder Res(2, 7) ableiten können. Jeder weitere Lösungsschritt würde zur Ableitung bereits vorhandener Klauseln führen. Da es die Ableitung des Leersatzes nicht zulässt, wurde somit gezeigt, dass die Wissensbasis nicht widersprüchlich ist. Bisher haben wir N und P abgeleitet. Um zu zeigen, dass ¬S gilt, fügen wir die Klausel $(S)_8$ als negierte Abfrage zur Menge der Klauseln hinzu. Mit dem Auflösungsschritt

Res(2, 8): $()_9$

Der Beweis ist vollständig. Somit gilt ¬S ÿ N ÿ P. Die „charmante englische Familie“ stammt offenbar aus Newcastle, hat in Paris Halt gemacht, reist aber nicht nach Spanien.

*Beispiel 2.3* Logikrätsel Nummer 28 aus [Ber89] mit dem Titel „ Der *Hochsprung“* lautet

Drei Mädchen üben Hochsprung für ihre Abschlussprüfung im Sportunterricht. Die Messlatte ist auf 1,20 Meter festgelegt. „Ich wette“, sagt das erste Mädchen zum zweiten, „dass ich es schaffen werde, wenn und nur wenn du es nicht tust.“ Wenn das zweite Mädchen dasselbe zum dritten sagte, das wiederum dasselbe zum ersten sagte, wäre es dann möglich, dass alle drei ihre Wetten gewinnen?

**Wir zeigen durch den Beweis durch Beschluss, dass nicht alle drei ihre Wetten gewinnen können.**
***Formalisierung:***

| | |
|---|---|
| Der Sprung des ersten Mädchens gelingt: A, | Wette des ersten Mädchens: (A ÿ ¬B), |
| der Sprung des zweiten Mädchens gelingt: B, | Wette des zweiten Mädchens: (B ÿ ¬C), |
| der Sprung des dritten Mädchens gelingt: C. | Wette des dritten Mädchens: (C ÿ ¬A). |

**Behauptung: Die drei können nicht alle ihre Wetten gewinnen:**

$$Q \ddot{y} \neg((A \ddot{y} \neg B) \ddot{y} (B \ddot{y} \neg C) \ddot{y} (C \ddot{y} \neg A))$$

**Es muss nun durch Resolution gezeigt werden, dass ¬Q unerfüllbar ist.**
***Umwandlung in CNF:*** **Erste Mädchenwette:**

$$(A \ddot{y} \neg B) \ddot{y} (A \ddot{y} \neg B) \ddot{y} (\neg B \ddot{y} A) \ddot{y} (\neg A \ddot{y} \neg B) \ddot{y} (A \ddot{y} B)$$

**Die Wetten der anderen beiden Mädchen durchlaufen analoge Transformationen und wir erhalten die negierte Behauptung**

$$\neg Q \ddot{y} (\neg A \ddot{y} \neg B)1 \ddot{y} (A \ddot{y} B)2 \ddot{y} (\neg B \ddot{y} \neg C)3 \ddot{y} (B \ddot{y} C)4 \ddot{y} (\neg C \ddot{y} \neg A)5 \ddot{y} (C \ddot{y} A)6.$$

**Von dort leiten wir mittels Resolution den Leersatz ab:**

$$\text{Res}(1, 6): (C \ddot{y} \neg B)7$$
$$\text{Res}(4, 7): (C)8$$
$$\text{Res}(2, 5): (B \ddot{y} \neg C)9$$
$$\text{Res}(3, 9): (\neg C)10$$
$$\text{Res}(8, 10): ()$$

**Somit ist die Behauptung bewiesen.**

## 2.5 Hornklauseln

**Ein Satz in konjunktiver Normalform enthält positive und negative Literale und kann in der Form dargestellt werden**

$$(\neg A1 \ddot{y} \cdots \ddot{y} \neg Am \ddot{y} B1 \ddot{y} \cdots \ddot{y} Bn)$$

mit den Variablen A1,...,Am und B1,...,Bn. Dieser Satz kann in zwei einfachen Schritten in die entsprechende Form umgewandelt werden

$$A1 \ddot{y} \cdots \ddot{y} Am \ddot{y} B1 \ddot{y} \cdots \ddot{y} Bn.$$

Diese Implikation enthält die Prämisse, eine Konjunktion von Variablen, und die Schlussfolgerung, eine Disjunktion von Variablen. Zum Beispiel: *„Wenn das Wetter schön ist und Schnee liegt, gehe ich Skifahren oder arbeite.“* ist ein Satz dieser Form. Der Empfänger dieser Nachricht weiß mit Sicherheit, dass der Absender nicht schwimmen geht.
Eine deutlich klarere Aussage wäre : *„Wenn das Wetter schön ist und Schnee liegt, gehe ich Skifahren.“* Der Empfänger weiß es jetzt definitiv. Daher nennen wir Klauseln mit höchstens einem positiven Literal definite Klauseln. Diese Klauseln haben den Vorteil, dass sie nur eine Schlussfolgerung zulassen und dadurch deutlich einfacher zu interpretieren sind. Viele Beziehungen können durch Klauseln dieser Art beschrieben werden. Wir definieren daher

**Definition 2.10** Klauseln mit höchstens einem positiven Literal der Form

$$(\neg A1 \ddot{y} \cdots \ddot{y} \neg Am \ddot{y} B) \text{ oder } (\neg A1 \ddot{y} \cdots \ddot{y} \neg Am) \text{ oder } B$$

oder gleichwertig)

$$A1 \ddot{y} \cdots \ddot{y} Am \ddot{y} B \text{ oder } A1 \ddot{y} \cdots \ddot{y} Am \ddot{y} f \text{ oder } B.$$

werden ***Hornsätze*** (nach ihrem Erfinder) genannt . Eine Klausel mit einem einzigen positiven Literal ist eine ***Tatsache.*** In Sätzen mit negativem und einem positiven Literal wird das positive Literal als ***Kopf bezeichnet.***

Um die Darstellung von Horn-Klauseln besser zu verstehen, kann der Leser sie aus den Definitionen der Äquivalenzen ableiten, die wir derzeit verwendet haben (Übung 2.12 auf Seite 30).

Hornsätze sind nicht nur im täglichen Leben, sondern auch im formalen Denken einfacher zu handhaben, wie wir im folgenden Beispiel sehen können. Die Wissensbasis soll aus den folgenden Klauseln bestehen (das „ÿ“, das die Klauseln bindet, wird hier und im folgenden Text weggelassen):

***(schönes_Wetter)1***

***(Schneefall)2***

***(Schneefall ÿ Schnee)3***

***(schönes Wetter ÿ Schnee ÿ Skifahren)4***

Wenn wir nun wissen wollen, ob *das Skifahren* hält, lässt sich dies leicht ableiten. Als Inferenzregel genügt hier ein leicht verallgemeinerter Modus Ponens:

$$\frac{A1 \ddot{y} \cdots \ddot{y} Am, A1 \ddot{y} \cdots \ddot{y} Am \ddot{y} B}{B}.$$

Der Beweis für *„Skifahren“* hat die folgende Form (MP(i1,...,ik) stellt die Anwendung des Modus Ponens auf die Klauseln i1 bis ik dar :

**MP(2, 3):** ***(Schnee)5*** **MP(1,**

**5, 4):** ***(Skifahren)6.***

Mit Modus Ponens erhalten wir einen vollständigen Kalkül für Formeln, die aus aussagenlogischen Hornsätzen bestehen. Bei großen Wissensbeständen kann der Modus Ponens jedoch viele unnötige Formeln ableiten, wenn man mit den falschen Sätzen beginnt. Daher ist es in vielen Fällen besser, eine Berechnung zu verwenden, die mit der Abfrage beginnt und rückwärts arbeitet, bis die Fakten vorliegen. Solche Systeme werden als ***Rückwärtsverkettung bezeichnet,*** im Gegensatz zu ***Vorwärtsverkettungssystemen ,*** die mit Fakten beginnen und schließlich die Abfrage ableiten, wie im obigen Beispiel mit dem Modus Ponens.

Für die Rückwärtsverkettung von Horn-Klauseln wird ***die SLD-Auflösung*** verwendet. SLD steht für ***„Selection Rule Driven Linear Resolution for Definite Clauses“.*** Im obigen Beispiel erweitert um die negierte Abfrage ***(Skifahren*** **ÿ f )**

***(schönes_Wetter)1***

***(Schneefall)2***

***(Schneefall*** **ÿ** ***Schnee)3***

***(schönes Wetter*** **ÿ** ***Schnee*** **ÿ** ***Skifahren)4***

***(Skifahren*** **ÿ f )5**

Wir führen die SLD-Lösung beginnend mit den Lösungsschritten durch, die sich aus dieser Klausel ergeben

**Res(5, 4):** ***(schönes_Wetter*** **ÿ** ***Schnee*** **ÿ f )6**

**Res(6, 1):** ***(Schnee*** **ÿ f )7**

**Res(7, 3):** ***(Schneefall*** **ÿ f )8**

**Res(8, 2): ()**

und leiten Sie einen Widerspruch mit dem leeren Satz ab. Hier können wir leicht die ***„lineare Auflösung“ erkennen,*** was bedeutet, dass die weitere Verarbeitung immer für die aktuell abgeleitete Klausel erfolgt. Dies führt zu einer starken Reduzierung des Suchraums. Darüber hinaus werden die Literale der aktuellen Klausel immer in einer festen Reihenfolge (z. B. von rechts nach links) verarbeitet ***(„Selection Rule Driven“).*** Die Literale der aktuellen Klausel werden aufgerufen

*Unterziel.* Die Literale der negierten Abfrage sind die *Ziele.* Die Inferenzregel für einen Schritt lautet

$$\frac{A_1 \ddot{y} \cdots \ddot{y} A_m \ddot{y} B_1, B_1 \ddot{y} B_2 \ddot{y} \cdots \ddot{y} B_n \ddot{y} f}{A_1 \ddot{y} \cdots \ddot{y} A_m \ddot{y} B_2 \ddot{y} \cdots \ddot{y} B_n \ddot{y} f}.$$

Vor der Anwendung der Inferenzregel müssen B1, B2,...,Bn – die aktuellen Teilziele – bewiesen werden. Nach der Anwendung wird B1 durch das neue Teilziel A1 ÿ···ÿ Am ersetzt. Um zu zeigen, dass B1 wahr ist, müssen wir nun zeigen, dass A1 ÿ···ÿ Am wahr sind. Dieser Vorgang wird so lange fortgesetzt, bis die Liste der Unterziele der aktuellen Klauseln (der sogenannte *Zielstapel)* leer ist. Damit wurde ein Widerspruch festgestellt. Wenn es für ein Unterziel ¬Bi keinen Satz mit dem Komplementärliteral Bi als Satzkopf gibt, endet der Beweis und es kann kein Widerspruch gefunden werden. Die Anfrage ist somit unbeweis

Die SLD-Auflösung spielt in der Praxis eine wichtige Rolle, da Programme in der logischen Programmiersprache PROLOG aus prädikatenlogischen Horn-Klauseln bestehen und ihre Verarbeitung mittels SLD-Auflösung erfolgt (siehe Übung 2.13 auf Seite 30, bzw. Kap. 5).

## 2.6 Berechenbarkeit und Komplexität

Die Wahrheitstabellenmethode stellt als einfachstes semantisches Beweissystem für die Aussagenlogik einen Algorithmus dar, der jedes Modell einer beliebigen Formel in endlicher Zeit bestimmen kann. Somit sind die Mengen unerfüllbarer, erfüllbarer und gültiger Formeln entscheidbar. Die Rechenzeit der Wahrheitstabellenmethode zur Erfüllbarkeit wächst im ungünstigsten Fall exponentiell mit der Anzahl n der Variablen, Nr. 2 da die Wahrheitstabelle Zeilen hat. Eine Optimierung, die Methode der *semantischen Bäume,* vermeidet die Betrachtung von Variablen, die nicht in Klauseln vorkommen, und spart so in vielen Fällen Rechenzeit, im schlimmsten Fall ist sie aber ebenfalls exponentiell.

Bei der Auflösung wächst im ungünstigsten Fall die Anzahl der abgeleiteten Sätze exponentiell mit der Anzahl der Anfangssätze. Um zwischen den beiden Verfahren zu entscheiden, können wir daher die Faustregel verwenden, dass bei vielen Klauseln mit wenigen Variablen die Wahrheitstabellenmethode vorzuziehen ist und bei wenigen Klauseln mit vielen Variablen die Auflösung wahrscheinlich schneller abgeschlossen wird .

Es bleibt die Frage: Kann der Beweis in der Aussagenlogik schneller gehen? Gibt es bessere Algorithmen? Die Antwort: wahrscheinlich nicht. Schließlich hat S. Cook, der Begründer der Komplexitätstheorie, gezeigt, dass das 3-SAT-Problem NP-vollständig ist. 3-SAT ist die Menge aller CNF-Formeln, deren Klauseln genau drei Literale haben. Somit ist klar, dass es für 3-SAT wahrscheinlich (modulo des P/NP-Problems) keinen polynomialen Algorithmus und daher wahrscheinlich auch keinen allgemeinen gibt. Für Horn-Klauseln gibt es jedoch einen Algorithmus, bei dem die Rechenzeit zum Testen der Erfüllbarkeit nur linear mit zunehmender Anzahl an Literalen in der Formel wächst.

## 2.7 Anwendungen und Einschränkungen

Theorembeweiser für Aussagenlogik gehören zum alltäglichen Werkzeugset des Entwicklers in der digitalen Technologie. Zu diesen Aufgaben gehören beispielsweise die Verifizierung digitaler Schaltkreise und die Generierung von Testmustern zum Testen von Mikroprozessoren in der Fertigung. Als Datenstruktur zur Verarbeitung aussagenlogischer Formeln werden auch spezielle Beweissysteme eingesetzt, die mit binären Entscheid

In der KI wird Aussagenlogik in einfachen Anwendungen eingesetzt. Einfache Expertensysteme können beispielsweise durchaus mit Aussagenlogik arbeiten. Allerdings müssen die Variablen alle diskret sein und nur wenige Werte aufweisen, und es dürfen keine Kreuzbeziehungen zwischen den Variablen bestehen. Komplexe logische Zusammenhänge lassen sich mit der Prädikatenlogik wesentlich eleganter ausdrücken.

Die probabilistische Logik ist eine sehr interessante und aktuelle Kombination aus Aussagenlogik und probabilistischer Berechnung, die die Modellierung unsicheren Wissens ermöglicht. Es wird in Kap. ausführlich behandelt. 7. In diesem Kapitel wird auch die Fuzzy-Logik besprochen, die unendlich viele Wahrheitswerte zulässt.

## 2.8 Übungen

ÿ Übung 2.1 Geben Sie eine Backus-Naur-Formengrammatik für die Syntax von Aussagen an Logik.

Aufgabe 2.2 Zeigen Sie, dass die folgenden Formeln Tautologien sind: (a) ¬(A ÿ B) ÿ ¬A ÿ ¬B (b) A ÿ B ÿ ¬B ÿ ¬A (c) ((A ÿ B) ÿ (B ÿ A)) ÿ (A ÿ B) (d) (A ÿ B) ÿ (¬B ÿ C) ÿ (A ÿ C)

Aufgabe 2.3 Wandeln Sie die folgenden Formeln in konjunktive Normalform um: (a) A ÿ B (b) A ÿ B ÿ A ÿ B (c) A ÿ (A ÿ B) ÿ B

Aufgabe 2.4 Überprüfen Sie die folgenden Aussagen auf Erfüllbarkeit bzw. Gültigkeit. (a) (play_lottery ÿ six_right) ÿ Gewinner (b) (play_lottery ÿ six_right ÿ (six_right ÿ win)) ÿ win (c) ¬(¬gas_in_tank ÿ (gas_in_tank ÿ ¬car_starts) ÿ ¬car_starts)

ÿ ÿ Übung 2.5 Programmieren Sie mit der Programmiersprache Ihrer Wahl einen Theorembeweis für Aussagenlogik unter Verwendung der Wahrheitstabellenmethode für Formeln in konjunktiver Normalform. Um eine kostspielige Syntaxprüfung der Formeln zu vermeiden, können Sie Klauseln als Listen oder Mengen von Literalen und die Formeln als Listen oder Mengen von Klauseln darstellen. Das Programm soll anzeigen, ob die Formel unerfüllbar, erfüllbar oder wahr ist, und die Anzahl unterschiedlicher Interpretationen und Modell

**Aufgabe 2.6**

**(a) Zeigen Sie, dass Modus Ponens eine gültige Inferenzregel ist, indem Sie zeigen, dass A ÿ (A ÿ B) | B. (b) Zeigen Sie, dass die Resolutionsregel (2.1) eine gültige Inferenzregel ist.**

**ÿ Aufgabe 2.7 Zeigen Sie durch Anwendung der Auflösungsregel, dass im Konjunktivnormal Form ist die leere Klausel gleichbedeutend mit der falschen Aussage.**

**ÿ Übung 2.8 Zeigen Sie, dass man mit Resolution jede beliebige Klausel daraus „ableiten“ kann eine Wissensbasis, die einen Widerspruch enthält.**

**Aufgabe 2.9 Formalisieren Sie die folgenden logischen Funktionen mit den logischen Operatoren und zeigen Sie, dass Ihre Formel gültig ist. Präsentieren Sie das Ergebnis in CNF. (a) Die XOR-Operation (exklusiv oder) zwischen zwei Variablen. (b) Die Aussage, *dass mindestens zwei der drei Variablen* A,B,C *wahr sind.***

**ÿ Übung 2.10 Lösen Sie den folgenden Fall mit Hilfe eines Auflösungsbeweises: „Wenn der Täter einen Komplizen hatte, dann kam er mit einem Auto.“ Der Täter hatte keinen Komplizen und keinen Schlüssel, oder er hatte den Schlüssel und einen Komplizen. Der Verbrecher hatte den Schlüssel. Ist der Verbrecher mit einem Auto gekommen oder nicht?“**

**Aufgabe 2.11 Zeigen Sie durch Auflösung, dass die Formel aus (a) Aufgabe 2.2(d) eine Tautologie ist. (b) Aufgabe 2.4(c) ist nicht erfüllbar.**

**Aufgabe 2.12 Beweisen Sie die folgenden Äquivalenzen, die für die Arbeit mit Hornsätzen wichtig sind: (a) (¬A1 ÿ···ÿ¬Am ÿ B) ÿ A1 ÿ···ÿ Am ÿ B (b) (¬A1 ÿ· ··ÿ¬Am) ÿ A1 ÿ···ÿ Am ÿ f (c) A ÿ w ÿ A**

**Aufgabe 2.13 Zeigen Sie durch SLD-Auflösung, dass die folgende Horn-Klauselmenge unerfüllbar ist.**

| | | |
|---|---|---|
| **(A)1** | **(D)4** | **(A ÿ D ÿ G)7** |
| **(B)2** | **(E)5** | **(C ÿ F ÿ E ÿ H )8** |
| **(C)3** | **(A ÿ B ÿ C ÿ F )6** | **(H ÿ f )9** |

**ÿ Übung 2.14 In Abschn. 2.6 heißt es: „Damit ist klar, dass es wahrscheinlich (modulo des P/NP-Problems) keinen polynomialen Algorithmus für 3-SAT gibt und daher wahrscheinlich auch keinen allgemeinen.“ Begründen Sie das „wahrscheinlich“ in diesem Satz.**

# Prädikatenlogik erster Ordnung

# 3

Viele praktisch relevante Probleme lassen sich nicht oder nur sehr umständlich in der Sprache der Aussagenlogik formulieren, wie wir am folgenden Beispiel leicht erkennen können. Die Aussage

*„Roboter 7 befindet sich an der xy-Position (35, 79)“*

kann tatsächlich direkt als aussagenlogische Variable verwendet werden

*„Robot_7_is_situated_at_xy_position_(35, 79)“*

für das Denken mit Aussagenlogik, aber das Denken mit dieser Art von Aussage ist sehr unbequem. Angenommen, 100 dieser Roboter können irgendwo auf einem Raster von 100×100 Punkten anhalten. Um jede Position jedes Roboters zu beschreiben, bräuchten wir $100 \cdot 100 \cdot 100 = 1.000.000 = 10^6$ verschiedene Variablen. Die Definition von Beziehungen zwischen Objekten (hier Robotern) wird wirklich schwierig. Die Beziehung

*„Roboter* A *steht rechts von Roboter* B.“

ist semantisch nichts anderes als eine Menge von Paaren. Von den 10.000 möglichen x-Koordinatenpaaren gibt es $(99 \cdot 98)/2 = 4851$ geordnete Paare. Zusammen mit allen 10.000 Kombinationen möglicher y-Werte für beide Roboter ergibt sich $(100 \cdot 99) = 9900$ für Mulas des Typs

*Robot_7_is_to_the_right_of_robot_12* ÿ
*Robot_7_is_situated_at_xy_position_(35, 79)* ÿ
Robot_12_is_situated_at_xy_position_(10, *93)* ÿ ···

Definieren Sie diese Beziehungen, jeweils mit $(104)^2 \cdot 0{,}485 = 0{,}485 \cdot 10^8$ Alternativen auf der rechten Seite. In der Prädikatenlogik erster Ordnung können wir hierfür ein Prädikat Position(Zahl, *xPosition, yPosition) definieren.* Die obige Relation muss nicht mehr als eine große Anzahl von Paaren aufgezählt werden, sondern wird abstrakt mit einer Formregel beschrieben

ÿu ÿv *is_further_right(u,* v) ÿ ÿxu
ÿyu ÿxv ÿyv *position(u,* xu, yu) ÿ *position(v,* xv, yv) ÿ xu > xv,

Wobei ÿu als „für jedes u“ und ÿv als „es existiert v“ gelesen wird.

W. Ertel, *Einführung in die künstliche Intelligenz,*
Bachelor-Themen in Informatik, DOI
10.1007/978-0-85729-299-5_3, 

In diesem Kapitel definieren wir die Syntax und Semantik der *Prädikatenlogik erster Ordnung (PL1),* zeigen, dass viele Anwendungen mit dieser Sprache modelliert werden können und zeigen, dass es für diese Sprache einen vollständigen und fundierten Kalkül gibt.

## 3.1 Syntax

Zunächst festigen wir die syntaktische Struktur von Begriffen.

**Definition 3.1** Es sei V eine Menge von Variablen, K eine Menge von Konstanten und F eine Menge von Funktionssymbolen. Die Mengen VK und F sind paarweise disjunkt. Wir definieren die Menge der *Terme* rekursiv: • Alle Variablen und Konstanten sind (atomare) Terme. • Wenn t1,...,tn Terme sind und f ein n-stelliges Funktionssymbol, dann ist f (t1,...,tn) ist auch ein Begriff.

Einige Beispiele für Begriffe sind f (sin(ln(3)), exp(x)) und g(g(g(x))). In der Lage sein stellen logische Beziehungen zwischen Begriffen her, wir bilden Formeln aus Begriffen.

**Definition 3.2** Sei P eine Menge von Prädikatsymbolen. *Formeln der Prädikatenlogik* werden wie folgt aufgebaut: • Wenn t1,...,tn Terme sind und p ein n-stelliges Prädikatssymbol, dann ist p(t1,...,tn) eine (atomare) Formel. • Wenn A und B Formeln sind, dann sind ¬A, (A), A ÿ B, A ÿ B, A ÿ B, A ÿ B ebenfalls Formeln.
• Wenn x eine Variable und A eine Formel ist, dann sind ÿx A und ÿx A ebenfalls Formeln. ÿ ist der universelle Quantor und ÿ der Existenzquantor.
• p(t1,...,tn) und ¬p(t1,...,tn) heißen Literale. • Formeln, in denen jede Variable im Gültigkeitsbereich eines Quantors liegt, werden *Sätze erster Ordnung* oder *geschlossene Formeln genannt.* Variablen, die nicht im Gültigkeitsbereich eines Quantors liegen, werden *freie Variablen genannt.* • Die Definitionen 2.8 (CNF) und 2.10 (Horn-Klauseln) gelten analog für Formeln prädikatslogischer Literale.

In Tabelle 3.1 auf Seite 33 sind mehrere Beispiele für PL1-Formeln aufgeführt mit ihren intuitiven Interpretationen.

Tabelle 3.1 Beispiele für Formeln in der Prädikatenlogik erster Ordnung. Bitte beachten Sie, dass es sich hier um eine *Mutter* handelt Funktionssymbol

| Formel | Beschreibung |
|---|---|
| ÿx *Frosch(x)* ÿ *grün(x)* ÿx | Alle Frösche sind grün |
| *Frosch(x)* ÿ *braun(x)* ÿ groß(x) ÿx | Alle braunen Frösche sind groß |
| *mag(x, Kuchen)* | Jeder mag Kuchen |
| ¬ÿx *Likes(x, Kuchen)* | Nicht jeder mag Kuchen |
| ¬ÿx *Likes(x, Kuchen)* | Niemand mag Kuchen |
| ÿx ÿy *mag(y,* x) ÿx | Es gibt etwas, das jedem gefällt |
| ÿy *mag(x,* y) ÿx ÿy | Es gibt jemanden, der alles mag |
| *mag(y,* x) ÿx ÿy | Alles wird von jemandem geliebt |
| *mag(x,* y) ÿx | Jeder mag etwas |
| *Kunde(x)* ÿ *mag(bob ,x)* ÿx *Kunde(x)* | Bob mag jeden Kunden |
| ÿ *Likes(x, Bob)* ÿx *Bäcker(x)* ÿ ÿy | Es gibt einen Kunden, den Bob mag |
| *Kunde(y)* ÿ mag(x, y) | Es gibt einen Bäcker, der alle seine mag Kunden |
| ÿx *älter(Mutter(x),* x) ÿx | Jede Mutter ist älter als ihr Kind |
| älter(Mutter(Mutter(x)), x) | Jede Großmutter ist älter als sie Kind der Tochter |
| ÿx ÿy ÿz rel(x, y) ÿ rel(y, z) ÿ rel(x, z) | *rel* ist eine transitive Beziehung |

## 3.2 Semantik

In der Aussagenlogik wird jeder Variablen durch eine Interpretation direkt ein Wahrheitswert zugeordnet. In der Prädikatenlogik wird die Bedeutung von Formeln rekursiv über die definiert Aufbau der Formel, indem wir zunächst Konstanten, Variablen und Funktion zuweisen Symbole zu Objekten in der realen Welt.

**Definition 3.3** Eine *Interpretation* I ist definiert als

- Eine Abbildung von der Menge der Konstanten und Variablen K ÿ V auf eine Menge W von Namen von Objekten in der Welt.
- Eine Zuordnung von der Menge der Funktionssymbole zur Menge der Funktionen im Welt. Jedem n-stelligen Funktionssymbol ist eine n-stellige Funktion zugeordnet.
- Eine Abbildung von der Menge der Prädikatsymbole auf die Menge der Beziehungen im Welt. Jedem n-stelligen Prädikatsymbol ist eine n-stellige Relation zugeordnet.

*Beispiel 3.1* Es seien c1, c2, c3 Konstanten, *„plus“* ein zweistelliges Funktionssymbol und *„gr“* ein zweistelliges Prädikatsymbol. Die Wahrheit der Formel

F ÿ *gr(plus(c1,* c3), c2)

hängt von der Interpretation I ab. Wir wählen zunächst die folgende naheliegende Interpretation der Konstanten, der Funktion und der Prädikate in den natürlichen Zahlen:

$$I1: c1 \ddot{y} 1, c2 \ddot{y} 2, c3 \ddot{y} 3, plus \ddot{y} +, gr \ddot{y}>$$

Somit wird die Formel abgebildet

$$1 + 3 > 2, \text{ oder nach Auswertung } 4 > 2.$$

Die Größer-als-Beziehung auf der Menge {1, 2, 3, 4} ist die Menge aller Zahlenpaare (x, y) mit x>y, d. h. die Menge G = {(4, 3), (4, 2), (4, 1), (3, 2), (3, 1), (2, 1)}. Da (4, 2) ÿ G, ist die Formel F unter der Interpretation I1 wahr. Wenn wir jedoch die Interpretation wählen

$$I2: c1 \ddot{y} 2, c2 \ddot{y} 3, c3 \ddot{y} 1, plus \ddot{y} \ddot{y}, gr \ddot{y}>,$$

wir erhalten

$$2 \ddot{y} 1 > 3 \text{ oder } 1 > 3.$$

Das Paar (1, 3) ist kein Mitglied von G. Die Formel F ist unter der Interpretation I2 falsch. Offensichtlich hängt die Wahrheit einer Formel in PL1 von der Interpretation ab. Nach dieser Vorschau definieren wir nun die Wahrheit.

**Definition 3.4 •**

Eine atomare Formel p(t1,...,tn) ist *wahr* (oder gültig) unter der Interpretation I , wenn nach Interpretation und Bewertung aller Terme t1,...,tn und Interpretation des Prädikats p bis die n-Stellen-Relation r, das gilt

$$(I(t1), \ldots, I(tn)) \ddot{y} r.$$

- Die Wahrheit quantifiziererloser Formeln folgt aus der Wahrheit atomarer Formeln – wie in der Aussagenrechnung – durch die Semantik der in Tabelle 2.1 auf Seite 17 definierten logischen Operatoren. • Eine Formel

ÿx F ist unter der Interpretation I genau dann wahr, wenn sie gilt ist wahr bei einer willkürlichen Änderung der Interpretation für die Variable x (und nur für x)

- Eine Formel ÿx F ist unter der Interpretation I genau dann wahr, wenn es für x eine Interpretation gibt, die die Formel wahr macht.

Die Definitionen der semantischen Äquivalenz von Formeln für die Konzepte erfüllbar, wahr, unerfüllbar und Modell sowie semantische Folgerung (Definitionen 2.4, 2.5, 2.6) werden unverändert aus der Aussagenlogik in die Prädikatenlogik übernommen.

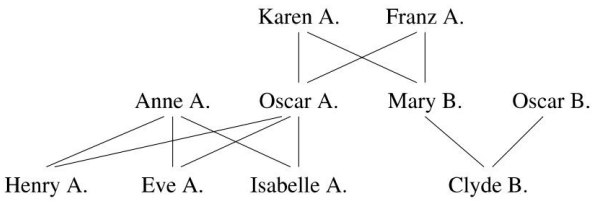



**Abb. 3.1 Ein Stammbaum. Die Kanten, die von Clyde B. aufwärts zu Mary B. und Oscar B. reichen, stellen das Element (Clyde B., Mary B., Oscar B.) als Kindbeziehung dar**

**Satz 3.1** ***Die Sätze*** **2.2** ***(Abzugssatz) und*** **2.3** ***(Widerspruchsbeweis) gelten analog für*** **PL1.**

***Beispiel 3.2*** Der in Abb. 3.1 **dargestellte Stammbaum stellt die Beziehung grafisch (auf der semantischen Ebene) dar**

**Kind = {(Oscar A., Karen A., Frank A.), (Mary B., Karen A., Frank A.), (Henry A., Anne A., Oscar A.), (Eve A., Anne A., Oscar A.), (Isabelle A., Anne A., Oscar A.), (Clyde B., Mary B., Oscar B.)}**

**Das Tripel (Oscar A., Karen A., Frank A.) steht beispielsweise für die Aussage** ***„Oscar A. ist ein Kind von Karen A. und Frank*** **A.“. Aus den Namen lesen wir die Ein-Stellen-Relation ab**

**Weiblich = {Karen A., Anne A., Mary B., Eve A., Isabelle A.}**

**der Frauen. Wir wollen nun Formeln für familiäre Beziehungen aufstellen. Zuerst definieren wir ein dreistelliges Prädikat** ***child(x,*** **y, z) mit der Semantik**

***I(Kind(x,*** **y, z)) = w ÿ (I(x),I(y),I(z)) ÿ Kind.**

**Unter der Interpretation** ***I(oscar)*** **= Oscar A.,** ***I(eve)*** **= Eve A.,** ***I(anne)*** **= Anne A. gilt auch, dass child(eve,** ***anne, oscar).*** **Damit child(eve,** ***oscar, anne)*** **wahr ist, benötigen wir, with**

**ÿx ÿy ÿz** ***Kind(x,*** **y, z) ÿ** ***Kind(x,*** **z, y),**

**Symmetrie des Prädikatkindes** ***in*** **den letzten beiden Argumenten. Für weitere Definitionen verweisen wir auf Übung 3.1 auf Seite 54 und definieren das Prädikat** ***Deszendent*** **rekursiv als**

**ÿx ÿy** ***Nachkomme(x,*** **y) ÿ ÿz** ***Kind(x,*** **y, z) ÿ**
**(ÿu ÿv** ***Kind(x,*** **u, v) ÿ** ***Nachkomme(u,*** **y)).**

**Jetzt bauen wir eine kleine Wissensbasis mit Regeln und Fakten auf. Lassen**

*KB* ÿ weiblich (Karen) ÿ weiblich (Anne) ÿ weiblich (Mary)

ÿ *weiblich (Eve)* ÿ *weiblich (Isabelle)*

ÿ *Kind (Oscar, Karen, Franz)* ÿ Kind (Mary, *Karen, Franz)*

ÿ Kind (Eve, *Anne, Oscar)* ÿ *Kind (Henry, Anne, Oscar)*

ÿ *Kind (Isabelle, Anne, Oscar)* ÿ *Kind (Clyde, Mary, Oscar)*

ÿ (ÿx ÿy ÿz *Kind(x,* y, z) ÿ *Kind(x,* z, y))

ÿ (ÿx ÿy *Nachkomme(x,* y) ÿ ÿz *Kind(x,* y, z)

ÿ (ÿu ÿv *Kind(x,* u, v) ÿ *Nachkomme(u,* y))).

Wir können nun beispielsweise fragen, ob die Propositionen child(eve, *oscar, anne)* oder Descendent(eve,franz) ableitbar sind. Dazu benötigen wir einen Kalkül.

## 3.2.1 Gleichheit

Um Begriffe vergleichen zu können, ist Gleichheit eine sehr wichtige Relation in der Prädikatenlogik. Die Gleichheit von Termen in der Mathematik ist eine Äquivalenzrelation, also reflexiv, symmetrisch und transitiv. Wenn wir Gleichheit in Formeln verwenden wollen, müssen wir entweder diese drei Attribute als Axiome in unsere Wissensbasis aufnehmen oder wir müssen Gleichheit in die Infinitesimalrechnung integrieren. Wir gehen den einfachen Weg und definieren ein Prädikat „=“, das abweichend von Definition 3.2 auf Seite 32 in der in der Mathematik üblichen Infix-Notation geschrieben wird. (Eine Gleichung x = y könnte natürlich auch in der Form eq(x, y) geschrieben werden.) Somit haben die Gleichheitsaxiome die Form

$$\begin{aligned} &\ddot{y}x\, x = x\, \ddot{y}x && \text{(Reflexivität)} \\ &\ddot{y}y\, x = y\, \ddot{y}\, y = x && \text{(Symmetrie)} \\ &\ddot{y}x\, \ddot{y}y\, \ddot{y}z\, x = y\, \ddot{y}\, y = z\, \ddot{y}\, x = z && \text{(Transitivität).} \end{aligned} \tag{3.1}$$

Um die Einzigartigkeit der Funktionen zu gewährleisten, benötigen wir zusätzlich

$$\ddot{y}x\, \ddot{y}y\, x = y\, \ddot{y}\, f(x) = f(y) \text{ (Substitutionsaxiom)} \tag{3.2}$$

für jedes Funktionssymbol. Analog benötigen wir für alle Prädikatsymbole

$$\ddot{y}x\, \ddot{y}y\, x = y\, \ddot{y}\, p(x)\, \ddot{y}\, p(y) \text{ (Substitutionsaxiom).} \tag{3.3}$$

Auf ähnliche Weise formulieren wir auch andere mathematische Beziehungen, etwa die „<“-Relation (Übung 3.4 auf Seite 55).

Oft muss eine Variable durch einen Term ersetzt werden. Um dies richtig durchzuführen und Um es einfach zu beschreiben, geben wir die folgende Definition.

**Definition 3.5 Wir schreiben** ÿ[x/t] **für die Formel, die sich ergibt, wenn wir jedes freie Vorkommen der Variablen** x **in** ÿ **durch den Term t ersetzen.** Dabei lassen wir im Term t **keine Variablen zu , die in ÿ quantifiziert sind . In diesen Fällen müssen Variablen umbenannt werden, um dies sicherzustellen.**

***Beispiel 3.3*** **Wenn in der Formel** ÿxx **= y die freie Variable** y durch den Term x **+ 1 ersetzt wird ,** ist das Ergebnis ÿxx **=** x **+ 1. Bei korrekter Substitution erhalten wir die Formel** ÿxx **=** y **+ 1, die hat eine ganz andere Semantik.**

## 3.3 Quantoren und Normalformen

**Nach Definition 3.4 auf Seite 34** ist **die Formel** ÿxp(x) **genau dann wahr, wenn sie für alle Interpretationen der Variablen x wahr ist. Anstelle des Quantors könnte man** p(a1) **ÿ···ÿ** p(an) **für alle Konstanten a1 ···** an **in K schreiben. Für** ÿxp(x) könnte man p(a1) **ÿ ··· ÿ schreiben** Pfanne). **Daraus folgt mit dem Gesetz von de Morgan das**

ÿx ÿ **ÿ ¬ÿx¬ÿ.**

**Durch diese Äquivalenz sind universelle und existentielle Quantoren wechselseitig ersetzbar.**

***Beispiel 3.4*** **Die Aussage *„Jeder möchte geliebt werden“* ist äquivalent zur Aussage *„Niemand möchte nicht geliebt werden“.***

**Quantoren sind ein wichtiger Bestandteil der Ausdruckskraft der Prädikatenlogik. Für die automatische Inferenz in der KI sind sie jedoch störend, da sie die Struktur von Formeln komplexer machen und die Anzahl anwendbarer Inferenzregeln in jedem Schritt eines Beweises erhöhen. Unser nächstes Ziel ist es daher, für jede prädikatenlogische Formel eine äquivalente Formel in einer standardisierten Normalform mit möglichst wenigen Quantoren zu finden. Als ersten Schritt bringen wir universelle Quantoren an den Anfang der Formel und definieren sie so**

**Definition 3.6 Eine prädikatenlogische Formel** ÿ **liegt in *Pränex-Normalform* vor, wenn gilt:**

• ÿ **=** Q1x1 ··· Qnxn **ÿ.** • ÿ **ist eine quantorenlose Formel.** • Qi **ÿ {ÿ, ÿ} für** i **=** 1,...,n.

Vorsicht ist geboten, wenn eine quantifizierte Variable außerhalb ihres Quan-Bereichs erscheint tifier, wie zum Beispiel x in

$$\ddot{y}xp(x) \ddot{y} \ddot{y}xq(x).$$

Hier muss eine der beiden Variablen umbenannt werden, und zwar in

$$\ddot{y}xp(x) \ddot{y} \ddot{y}yq(y)$$

Der Quantor kann leicht in den Vordergrund gebracht werden und wir erhalten als Ausgabe die äquivalente Formel

$$\ddot{y}x \ddot{y}yp(x) \ddot{y} q(y).$$

Wenn wir jedoch den Quantor korrekt in den Vordergrund stellen möchten

$$(\ddot{y}xp(x)) \ddot{y} \ddot{y}yq(y) \tag{3.4}$$

Wir schreiben die Formel zunächst in der entsprechenden Form

$$\neg(\ddot{y}xp(x)) \ddot{y} \ddot{y}yq(y).$$

Der erste universelle Quantor wird nun zu

$$(\ddot{y}x\neg p(x)) \ddot{y} \ddot{y}yq(y)$$

und nun können die beiden Quantoren endlich nach vorne gezogen werden

$$\ddot{y}x \ddot{y}y\neg p(x) \ddot{y} q(y),$$

was gleichbedeutend mit ist

$$\ddot{y}x \ddot{y}yp(x) \ddot{y} q(y).$$

Wir sehen dann, dass wir in (3.4) nicht einfach beide Quantoren in den Vordergrund ziehen können. Vielmehr müssen wir zunächst die Implikationen eliminieren, damit es keine Verneinungen zu den Quantifizierern gibt. Generell gilt, dass wir Quantoren nur dann herausziehen dürfen, wenn Negationen nur direkt auf atomaren Unterformeln existieren.

*Beispiel 3.5* Wie in der Analysis allgemein bekannt ist, ist die Konvergenz einer Reihe $(a_n)_{n\ddot{y}\mathbb{N}}$ zu einem Grenzwert a definiert durch

$$\ddot{y}\ddot{y} > 0 \ddot{y}n_0 \ddot{y} \mathbb{N} \ddot{y}n>n_0|a_n \ddot{y} a| < \ddot{y}.$$

Mit der Funktion abs(x) für |x|, a(n) für $a_n$, *minus(x,* y) für x ÿ y und den Prädikaten el(x, y) für x ÿ y, gr(x, y) für x>y lautet die Formel

**ÿÿ (gr(ÿ, 0) ÿ ÿn0 (el(n0,N) ÿ ÿn (gr(n, n0) ÿ gr(ÿ, *abs(minus(a(n),* a)))))) . (3.5)**

**Dies ist eindeutig nicht in der Prenex-Normalform der Fall. Da die Variablen der inneren Quantoren ÿn0 und ÿn nicht links von ihren jeweiligen Quantoren vorkommen, müssen keine Variablen umbenannt werden. Als nächstes beseitigen wir die Implikationen und erhalten**

**ÿÿ (¬gr(ÿ, 0)ÿ ÿn0 (¬el(n0,N)ÿ ÿn (¬gr(n, n0)ÿ gr(ÿ, *abs(minus(a(n),* a))) ))).**

**Da jede Negation vor einer Atomformel steht, bringen wir die Quantoren nach vorne, entfernen die überflüssigen Klammern und mit**

**ÿÿ ÿn0 ÿn (¬gr(ÿ, 0) ÿ ¬el(n0,N) ÿ ¬gr(n, n0) ÿ gr(ÿ, *abs(minus(a(n),* a))))**

**es wird ein quantifizierter Satz in konjunktiver Normalform.**

**Die transformierte Formel entspricht der Ausgabeformel. Die Tatsache, dass dies Transformation ist immer möglich, garantiert durch**

**Satz 3.2** ***Jede prädikatenlogische Formel kann in eine äquivalente Formel in Pränex-Normalform umgewandelt werden.***

**Darüber hinaus können wir alle existenziellen Quantoren eliminieren.** Allerdings **entspricht die aus der sogenannten *Skolemisierung*** resultierende Formel nicht mehr der Ausgabe für Mula. **Seine Erfüllbarkeit bleibt jedoch unverändert.** In vielen Fällen, insbesondere wenn man die Unerfüllbarkeit von ***KB*** ÿ ¬Q **zeigen möchte , ist dies ausreichend. Die folgende Formel in Pränex-Normalform wird nun skolemisiert:**

**ÿx1 ÿx2 ÿy1 ÿx3 ÿy2 p(f (x1), x2, y1) ÿ q(y1, x3, y2).**

**Da die Variable y1 offenbar von x1 und x2 abhängt,** wird **jedes Vorkommen von y1 durch eine Skolem-Funktion g(x1, x2) ersetzt. Wichtig ist, dass g ein neues Funktionssymbol ist, das noch nicht in der Formel vorkommt. Wir erhalten**

**ÿx1 ÿx2 ÿx3 ÿy2 p(f (x1), x2, g(x1, x2)) ÿ q(g(x1, x2), x3, y2)**

**und ersetze y2 analog durch h(x1, x2, x3), was zu führt**

**ÿx1 ÿx2 ÿx3 p(f (x1), x2, g(x1, x2)) ÿ q(g(x1, x2), x3, h(x1, x2, x3)).**

**Da nun alle Variablen universell quantifiziert sind, können die universellen Quantoren weggelassen werden, was zu Folgendem führt:**

**p(f (x1), x2, g(x1, x2)) ÿ q(g(x1, x2), x3, h(x1, x2, x3)).**

**NORMALFORMTRANSFORMATION(Formel): 1.**
***Transformation in die Pränex-Normalform:***
**Transformation in die konjunktive Normalform (Satz 2.1):**
**Eliminierung von Äquivalenzen.**
**Beseitigung von Implikationen.**
**Wiederholte Anwendung des de Morganschen Gesetzes und des Verteilungsgesetzes**
**Umbenennen von Variablen bei Bedarf.**
**Universelle Quantoren herausrechnen.**
**2. *Skolemisierung:***
**Ersatz existenziell quantifizierter Variablen durch neue Skolemfunktionen.**
**Löschung resultierender Universalquantoren.**

**Abb. 3.2 Transformation prädikatenlogischer Formeln in Normalform**

**Nun können wir den Existenzquantor (und damit auch den Universalquantor) in (3.5) auf Seite 39** eliminieren, indem wir die Skolem-Funktion **n0(ÿ) einführen . Die skolemisierte Pränex- und Konjunktivnormalform von (3.5) auf Seite 39 lautet also**

**¬gr(ÿ, 0) ÿ ¬el(n0(ÿ),N) ÿ ¬gr(n, n0(ÿ)) ÿ gr(ÿ, *abs(minus(a(n),* a))).**

**Durch Weglassen der Variablen n0 kann die Skolem-Funktion den Namen n0 erhalten .**

**Bei der Skolemisierung einer Formel in der Pränex-Normalform werden alle Existenzquantoren von außen nach innen eliminiert, wobei eine Formel der Form ÿx1 ...ÿxn** ÿy ÿ **durch** ÿx1 ...ÿxn ÿ[y/f (x1,...,xn)], **wobei** f **möglicherweise nicht in ÿ erscheint . Liegt ein Existenzquantor weit außerhalb, etwa in** ÿyp(y), **dann** muss y **durch eine Konstante (also durch ein Nullstellen-Funktionssymbol) ersetzt werden.**

**Das Verfahren zur Transformation einer Formel in konjunktive Normalform ist in dem in Abb. 3.2 dargestellten Pseudocode zusammengefasst. Die Skolemisierung hat eine polynomielle Laufzeit in der Anzahl der Literale. Bei der Transformation in die Normalform kann die Anzahl der Literale in der Normalform exponentiell ansteigen, was zu einer exponentiellen Rechenzeit und einem exponentiellen Speicherverbrauch führen kann. Der Grund dafür ist die wiederholte Anwendung des Distributivgesetzes. Das eigentliche Problem, das sich aus einer Vielzahl von Klauseln ergibt, ist die kombinatorische Explosion des Suchraums für einen nachfolgenden Auflösungsbeweis. Allerdings gibt es einen optimierten Transformationsalgorithmus, der nur polynomiell viele Literale erzeugt [Ede91].**

## 3.4 Beweiskalküle

**Für das Denken in der Prädikatenlogik wurden verschiedene Kalküle des natürlichen Denkens wie die Gentzen-Kalküle oder die Folgenrechnung entwickelt. Wie der Name schon sagt, sind diese Kalküle für die Anwendung durch Menschen gedacht, da die Inferenzregeln umfas**

Tabelle 3.2 Einfacher Beweis mit Modus Ponens und Quantoreneliminierung

| | | |
|---|---|---|
| *WB:* | 1 | Kind (Eve, *Anne, Oscar)* |
| *WB:* | 2 | ÿx ÿy ÿz *Kind(x,* y, z) ÿ *Kind(x,* z, y) |
| ÿE(2) : x/eve,y/anne, *z/oscar* | 3 | kind(eve, *anne, oscar)* ÿ kind(eve, *oscar, anne)* |
| MP(1, 3) | 4 | Kind (Eve, *Oscar, Anne)* |

oder weniger intuitiv und die Kalküle funktionieren auf beliebigen PL1-Formeln. Im nächsten Abschnitt Wir werden uns hauptsächlich auf die Auflösungsrechnung konzentrieren, die in der Praxis die ist wichtigste effiziente, automatisierbare Berechnung für Formeln im Konjunktivnormal bilden. Hier werden wir anhand von Beispiel 3.2 auf Seite 35 einen sehr kleinen „natürlichen“ Beweis liefern. Wir verwenden die Inferenzregel

$$\frac{A,\ A \text{ ÿ } B}{B} \quad \text{(modus ponens, MP) und} \quad \frac{\text{ÿx } A}{A[x/t]} \quad \text{(ÿ-Eliminierung,ÿE).}$$

Der Modus Ponens ist bereits aus der Aussagenlogik bekannt. Beim Eliminieren Bei universellen Quantoren muss man bedenken, dass die quantifizierte Variable x sein muss ersetzt durch einen Grundterm t, also einen Term, der keine Variablen enthält. Der Beweis von child(eve, *oscar, anne)* aus einer entsprechend reduzierten Wissensbasis wird vorgestellt in Tabelle 3.2.

Die beiden Formeln der reduzierten Wissensbasis sind in den Zeilen 1 und 2 aufgeführt In Zeile 3 werden die Universalquantoren aus Zeile 2 eliminiert, und in Zeile 4 lautet die Behauptung abgeleitet mit modus ponens.

Der Kalkül bestehend aus den beiden angegebenen Inferenzregeln ist nicht vollständig. Es kann jedoch durch Hinzufügung weiterer Schlussfolgerungen zu einem vollständigen Verfahren erweitert werden Regeln. Diese nicht triviale Tatsache ist für Mathematik und KI von grundlegender Bedeutung. Der österreichische Logiker Kurt Gödel bewies 1931, dass [Göd31a]

Satz 3.3 (Gödels Vollständigkeitssatz) *Prädikatenlogik erster Ordnung ist vollständig. Das heißt, es gibt einen Kalkül, mit dem jeder Satz, der a ist Konsequenz einer Wissensbasis KB nachgewiesen werden kann. Wenn KB* | ÿ*, dann gilt dass KB* ÿ ÿ.

Jeder wahre Satz in der Prädikatenlogik erster Ordnung ist daher beweisbar. Aber ist Gilt das umgekehrt auch? Ist alles, was wir syntaktisch ableiten können, tatsächlich wahr? Der Die Antwort lautet „Ja“:

Satz 3.4 (Korrektheit) *Es gibt Kalküle, mit denen nur wahre Aussagen bewiesen werden können. Das heißt, wenn KB* ÿ ÿ *gilt, dann gilt KB* | ÿ.

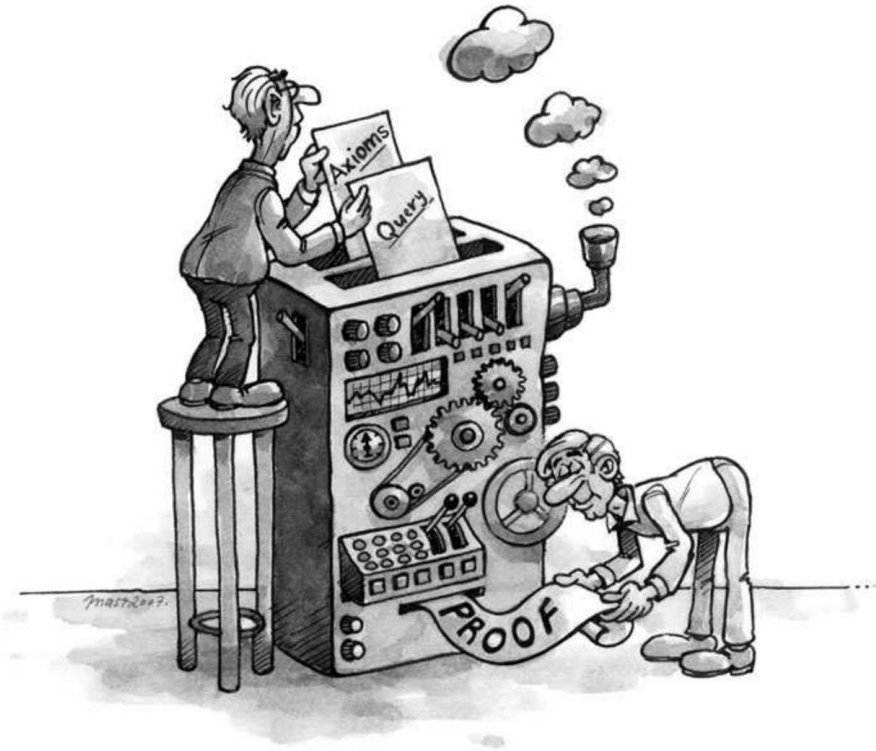



**Abb. 3.3 Die universelle Logikmaschine**

**Tatsächlich sind fast alle bekannten Kalküle korrekt. Schließlich macht es wenig Sinn zu arbeiten mit falschen Beweismethoden. Beweisbarkeit und semantische Konsequenz sind also äquivalente Konzepte, sofern korrekte und vollständige Analysis verwendet wird. Damit Prädikatenlogik erster Ordnung wird zu einem leistungsstarken Werkzeug für Mathematik und KI. Der Die oben genannten Kalküle des natürlichen Abzugs sind für eine Automatisierung eher ungeeignet. Nur Auflösungsrechnung, die 1965 eingeführt wurde und im Wesentlichen funktioniert Nur eine einfache Inferenzregel ermöglichte die Konstruktion leistungsstarker automatisierter Orem-Beweiser, die später als Inferenzmaschinen für Expertensysteme eingesetzt wurden.**

## 3.5 Auflösung

**Tatsächlich löste die korrekte und vollständige Auflösungsrechnung in den 1970er Jahren eine logische Euphorie aus. Viele Wissenschaftler glaubten, dass man fast jede Aufgabe formulieren könne der Wissensdarstellung und Argumentation in PL1 und lösen Sie es dann mit einem automatisierten Beweiser. Prädikatenlogik, eine kraftvolle, ausdrucksstarke Sprache, zusammen mit a Die vollständige Beweisrechnung schien die universelle intelligente Maschine zur Darstellung von Wissen und zur Lösung vieler schwieriger Probleme zu sein (Abb. 3.3).**

Wenn man eine Reihe von Axiomen (d. h. eine Wissensdatenbank) und eine Abfrage als Eingabe in eine solche Logikmaschine einspeist, sucht die Maschine nach einem Beweis und gibt ihn – denn einer existiert und wird gefunden – als Ausgabe zurück. Auf der Grundlage von Gödels Vollständigkeitssatz und der Arbeit von Herbrand wurde viel in die Mechanisierung der Logik investiert. Die Vision einer Maschine, die mit einer willkürlichen, nicht widersprüchlichen PL1-Wissensbasis jede echte Frage beweisen könnte, war sehr verlockend. Dementsprechend werden bisher viele Beweiskalküle für PL1 entwickelt und in Form von Theo-Rem-Beweisenden realisiert. Als Beispiel beschreiben wir hier den historisch wichtigen und weit verbreiteten Auflösungskalkül und zeigen seine Fähigkeiten. Der Grund für die Auswahl der Auflösung als Beispiel für eine Beweisrechnung in diesem Buch ist, wie bereits erwähnt, ihre historische und didaktische Bedeutung. Heutzutage ist die Auflösung nur eine von vielen Berechnungen, die in Hochleist

Wir beginnen damit, den Beweis in Tabelle 3.2 auf Seite 41 mit der Wissensbasis von Beispiel 3.2 zu einem Auflösungsbeweis zusammenzustellen . Zunächst werden die Formeln in die konjunktive Normalform und die negierte Abfrage umgewandelt

$$\neg Q \ddot{y} \neg kind(eve, oscar, anne)$$

wird zur Wissensdatenbank hinzugefügt, die Folgendes bietet

$$KB \ddot{y} \neg Q \ddot{y} (Kind(Eve, Anne, Oscar))1 \ddot{y}$$
$$(\neg Kind(x, y, z) \ddot{y} Kind(x, z, y))2 \ddot{y}$$
$$(\neg kind(eve, oscar, anne))3.$$

Der Beweis könnte dann etwa so aussehen

$$(2)\ x/eve, y/anne, z/oscar : (\neg child(eve, anne, oscar) \ddot{y}$$
$$child(eve, oscar, anne))4$$
$$Res(3, 4) : (\neg child(eve, anne, oscar))5$$
$$Res(1, 5) : ()6,$$

wobei im ersten Schritt die Variablen x,y,z durch Konstanten ersetzt werden. Anschließend folgen zwei Auflösungsschritte unter Anwendung der allgemeinen Auflösungsregel aus (2.2), die unverändert aus der Aussagenlogik übernommen wurde.

Etwas komplexer sind die Sachverhalte im folgenden Beispiel. Wir gehen davon aus, dass *jeder seine eigene Mutter kennt* und fragen, ob *Henry* jemanden kennt.
Mit dem Funktionssymbol *„Mutter“* und dem Prädikat *„weiß“* müssen wir einen Widerspruch ableiten

$$(weiß(x,Mutter(x)))1 \ddot{y} (\neg weiß(Henry, y))2.$$

Durch die Ersetzung $x/Henry, y/Mother(Henry)$ erhalten wir das widersprüchliche Satzpaar

$$(weiß(Henry,Mutter(Henry)))1 \ddot{y} (\neg weiß(Henry,Mutter(Henry)))2.$$

Dieser Ersetzungsschritt wird als *Vereinheitlichung bezeichnet.* Die beiden Literale ergänzen sich, was bedeutet, dass sie bis auf ihre Zeichen gleich sind. Die leere Klausel ist jetzt mit einem Auflösungsschritt ableitbar, durch den gezeigt wurde, dass Henry dies tut jemanden kennen (seine Mutter). Wir definieren

**Definition 3.7** Zwei Literale heißen *unifizierbar,* wenn eine Substitution ÿ vorliegt für alle Variablen, wodurch die Literale gleich sind. Ein solches ÿ wird *Unifikator genannt.* Ein Unifier wird als der allgemeinste Unifier (MGU) bezeichnet, wenn alle anderen Unifier dies können daraus durch Substitution von Variablen gewonnen.

*Beispiel 3.6* Wir wollen die Literale p(f (g(x)), y, z) und p(u, u, f (u)) vereinheitlichen. Mehrere Vereiniger sind

| | |
|---|---|
| ÿ1 : y/f (g(x)), z/f (f (g(x))), ÿ2 : x/h(v), y/f | u/f (g(x)), |
| (g(h(v))), z/f (f (g(h(v)))), ÿ3 : x/h(h(v)), y/f (g(h(h(v)))), z/f (f (g(h(h | u/f (g(h(v))) |
| (v))))), u/f (g(h(h(v)))) | |
| ÿ4 : x/h(a), y/f (g(h(a))), z/f (f (g(h(a)))), ÿ5 : x/a, y/f ( g ( a)), z/f (f | u/f (g(h(a))) |
| (g(a))), | u/f (g(a)) |

wobei ÿ1 der allgemeinste Unifikator ist. Die anderen Vereiniger ergeben sich aus ÿ1 durch Substitutionen x/h(v), x/h(h(v)), x/h(a), x/a.

An diesem Beispiel sehen wir, dass bei der Vereinheitlichung von Literalen die Prädikatssymbole wie Funktionssymbole behandelt werden können. Das heißt, das Literal wird wie ein Begriff Implementierungen von Vereinheitlichungsalgorithmen verarbeiten die Argumente von Funktionen sequentiell. Begriffe werden über die Begriffsstruktur rekursiv vereinheitlicht. Die einfachsten Vereinheitlichungsalgorithmen sind in den meisten Fällen sehr schnell. Im schlimmsten Fall kann die Rechenzeit jedoch exponentiell mit der Größe der Terme wachsen. Denn für automa beweist, dass die überwiegende Zahl der Einigungsversuche scheitern oder sehr einfach sind In den meisten Fällen hat die Worst-Case-Komplexität keine dramatischen Auswirkungen. Die schnellste Vereinigung Algorithmen haben selbst im schlimmsten Fall eine nahezu lineare Komplexität [Bib82].

Wir können nun die allgemeine Auflösungsregel für die Prädikatenlogik angeben:

**Definition 3.8** Die Auflösungsregel für zwei Klauseln in konjunktiver Normalform liest

$$\frac{(A1 \ddot{y} \cdots \ddot{y} Am \ddot{y} B), (\neg B' \ddot{y} C1 \ddot{y} \cdots \ddot{y} Cn) \ddot{y} (B) = \ddot{y} (B\ddot{y})}{(\ddot{y} (A1) \ddot{y} \cdots \ddot{y} \ddot{y} (Am) \ddot{y} \ddot{y} (C1) \ddot{y} \cdots \ddot{y} \ddot{y} (Cn))}, \quad (3.6)$$

wobei ÿ die MGU von B und B ist '.

**Satz 3.5** ***Die Auflösungsregel ist korrekt. Das heißt, die Auflösung ist eine semantische Konsequenz der beiden Elternsätze.***

**Der Vollständigkeit halber benötigen wir jedoch noch eine kleine Ergänzung, wie in der gezeigt folgendes Beispiel.**

***Beispiel 3.7*** **Das berühmte Russell-Paradoxon lautet:** ***„Es gibt einen Friseur, der jeden rasiert, der sich nicht selbst rasiert.“*** **Diese Aussage ist widersprüchlich, also unbefriedigend. Das wollen wir mit Entschlossenheit zeigen. Das in PL1 formalisierte Paradoxon lautet:**

$$\ddot{y}x\ \textit{Rasuren(Friseur,x)}\ \ddot{y}\ \neg\textit{Rasuren(x,}\ x)$$

**und Transformation in Klauselform ergibt (siehe Übung 3.6 auf Seite 55)**

$$(\neg\textit{shaves(barbier,x)}\ \ddot{y}\ \neg\textit{shaves(x,}\ x))1\ \ddot{y}\ (\textit{shaves(barbier,x)}\ \ddot{y}\ \textit{shaves(x,}\ x))2. \tag{3.7}$$

**Aus diesen beiden Sätzen können wir mehrere Tautologien ableiten, aber keinen Widerspruch. Somit ist die Lösung nicht vollständig. Wir brauchen noch eine weitere Inferenzregel.**

**Definition 3.9** ***Die Faktorisierung*** **einer Klausel erfolgt durch**

$$\frac{(A1\ \ddot{y}\ A2\ \ddot{y}\cdots\ddot{y}\ An)\ \ddot{y}\ (A1) = \ddot{y}\ (A2)\ (\ddot{y}\ (A2)\ \ddot{y}\cdots\ddot{y}}{\ddot{y}\ (An))},$$

**wobei ÿ die MGU von A1 und A2 ist.**

Nun lässt sich aus **(3.7)** **ein Widerspruch ableiten**

**Fak(1, ÿ : x/barber) : (¬shaves(barber,** ***barber))3***
**Fak(2, ÿ : x/Barber) : (Rasuren(Barber,** ***Barbier))4***
**Res(3, 4) : ()5**

**und wir behaupten:**

**Satz 3.6** ***Die Auflösungsregel*** **(3.6)** ist ***zusammen mit der Faktorisierungsregel*** **(3.9)** ***vollständig widerlegt. Das heißt, durch Anwendung von Faktorisierungs- und Auflösungsschritten kann der leere Satz aus jeder unerfüllbaren Formel in konjunktiver Normalform abgeleitet werden.***

### 3.5.1 Lösungsstrategien

Während die Vollständigkeit der Auflösung für den Benutzer wichtig ist, kann die Suche nach einem Beweis in der Praxis sehr frustrierend sein. Der Grund dafür ist der immense kombinatorische Suchraum. Auch wenn es zu Beginn nur sehr wenige Klauselpaare in *KB* ÿ ¬Q gibt, generiert der Beweiser mit jedem Auflösungsschritt eine neue Klausel, wodurch sich die Anzahl der möglichen Auflösungsschritte in der nächsten Iteration erhöht. Daher wird seit langem versucht, den Suchraum durch spezielle Strategien möglichst ohne Verlust der Vollständigkeit zu reduzieren. Die wichtigsten Strategien sind die folgenden.

*Bei der Unit-Resolution* werden Auflösungsschritte priorisiert, bei denen eine der beiden Klauseln nur aus einem Literal besteht, einer sogenannten Unit-Klausel. Diese Strategie wahrt die Vollständigkeit und führt in vielen Fällen, aber nicht immer, zu einer Reduzierung des Suchraums. Es handelt sich also um einen heuristischen Prozess (siehe Abschn. 6.3).

Durch Anwendung der Set-of-Support-Strategie erhält man eine garantierte Reduzierung des Suchraums . Hier wird eine Teilmenge von KBÿ¬Q als Supportmenge (SOS) definiert. Jeder Lösungsschritt muss eine Klausel aus dem SOS beinhalten, und die Lösung wird zum SOS hinzugefügt. Diese Strategie ist unvollständig. Sie wird vollständig, wenn sichergestellt ist, dass die Klauselmenge ohne das SOS erfüllbar ist (siehe Übung 3.7 auf Seite 55). Die negierte Abfrage ¬Q wird häufig als anfängliches SOS verwendet.

Bei *der Eingabeauflösung* muss in jedem Auflösungsschritt eine Klausel aus der Eingabemenge *KB* ÿ ¬Q beteiligt sein. Diese Strategie reduziert auch den Suchraum, allerdings auf Kosten der Vollständigkeit.

Mit der *reinen Literalregel* können alle Klauseln gelöscht werden, die Literale enthalten, zu denen es in anderen Klauseln keine komplementären Literale gibt. Diese Regel reduziert den Suchraum und ist vollständig und wird daher von praktisch allen Auflösungsprüfern verwendet.

Wenn die Literale einer Klausel K1 eine Teilmenge der Literale der Klausel K2 darstellen, kann K2 gelöscht werden. Zum Beispiel die Klausel

*(regnet (heute)* ÿ *street_wet (heute))*

ist überflüssig, wenn *street_wet(today)* bereits gültig ist. Dieser wichtige Reduktionsschritt wird *Subsumtion genannt.* Auch die Subsumtion ist abgeschlossen.

### 3.5.2 Gleichheit

Gleichheit ist eine besonders unangenehme Ursache für das explosionsartige Wachstum des Suchraums. Wenn wir (3.1) auf Seite 36 und die in (3.2) auf Seite 36 formulierten Gleichheitsaxiome zur Wissensbasis hinzufügen, dann kann beispielsweise die Symmetrieklausel ¬x = y ÿ y = x mit jeder positiven oder negierten Gleichung vereinheitlicht werden . Dies führt zur Ableitung neuer Klauseln und Gleichungen, auf die erneut Gleichheitsaxiome angewendet werden können, und so weiter. Die Transitivitäts- und Substitutionsaxiome haben ähnliche Konsequenzen. Aus diesem Grund wurden spezielle Gleichheitsschlussregeln entwickelt, die ohne explizite Gleichheitsaxiome auskommen und insbesondere den Suchraum reduzieren. *Demodulation,*

ermöglicht beispielsweise die Ersetzung von t1 durch einen Term t2 , wenn die Gleichung t1 = t2 existiert. Ein Gleichung t1 = t2 wird durch Vereinheitlichung auf einen Term t wie folgt angewendet:

$$\frac{t1 = t2, \quad (\ldots t \ldots), \ddot{y}(t1) = \ddot{y}(t)}{(\ldots \ddot{y}(t2)\ldots)}.$$

Etwas allgemeiner ist *die Paramodulation,* die mit bedingten Gleichungen arbeitet [Bib82, Lov78].

Die Gleichung t1 = t2 erlaubt die Substitution des Termes t1 durch t2 sowie die Ersetzung von t2 durch t1. Es ist in der Regel sinnlos, eine bereits erfolgte Ersetzung rückgängig zu machen durchgeführt worden. Im Gegenteil werden Gleichungen häufig zur Vereinfachung von Begriffen verwendet. Sie werden daher häufig nur in eine Richtung verwendet. Gleichungen, die nur in verwendet werden eine Richtung nennt man gerichtete Gleichungen. Eine effiziente Verarbeitung gerichteter Gleichungen wird durch sogenannte *Term-Rewriting-Systeme erreicht.* Für Formeln mit vielen Gleichungen gibt es spezielle Gleichheitsbeweiser.

## 3.6 Automatisierte Theorembeweiser

Implementierungen von Beweisrechnungen auf Computern werden als Theorembeweiser bezeichnet. Entlang mit spezialisierten Prüfern für Teilmengen von PL1 oder spezielle Anwendungen gibt es heute eine ganze Reihe automatisierter Beweiser für die vollständige Prädikatenlogik und Logik höherer Ordnung, von denen hier nur einige besprochen werden sollen. Das Wichtigste im Überblick Systeme finden Sie in [McC].

Einer der ältesten Auflösungsprüfer wurde im Oratorium des Argonne National Lab in Chicago entwickelt. Basierend auf frühen Entwicklungen ab 1963, Otter [Kal01], wurde 1984 gegründet. Otter wurde vor allem in Spezialgebieten erfolgreich eingesetzt der Mathematik, wie man auf seiner Homepage erfahren kann:

> „Derzeit ist die Hauptanwendung von Otter die Forschung in abstrakter Algebra und formaler Logik. Otter und seine Vorgänger wurden zur Beantwortung vieler offener Fragen in den Bereichen verwendet endliche Halbgruppen, ternäre Boolesche Algebra, Logikkalküle, kombinatorische Logik, Gruppentheorie, Gittertheorie und algebraische Geometrie."

Einige Jahre später entwickelte die Technische Universität München den Hochleistungsprüfer SETHEO [LSBB92] auf Basis der schnellen PROLOG-Technologie. Mit Mit dem Ziel, eine noch höhere Leistung zu erreichen, wurde unter dem Namen PARTHEO eine Implementierung für Parallelrechner entwickelt. Es stellte sich heraus, dass es sich nicht lohnte, bei Theoremprüfern spezielle Hardware einzusetzen, wie dies auch in anderen Bereichen der Fall ist von KI, weil diese Computer sehr schnell von schnelleren Prozessoren überholt werden und intelligentere Algorithmen. München ist auch der Geburtsort von E [Sch02], einem preisgekrönten modernen Gleichungsbeweis, den wir im nächsten Kapitel kennenlernen werden Beispiel. Auf der Homepage von E ist folgende kompakte, ironische Charakterisierung zu lesen:

deren zweiter Teil übrigens für alle existierenden automatisierten Prüfer gilt
Heute.

> „E ist ein reiner Gleichungssatzbeweiser für die Klausellogik. Das heißt, es ist ein Programm, das Sie können eine mathematische Spezifikation (in Klausellogik mit Gleichheit) und eine Hypothese hineinstopfen, die dann ewig läuft und alle Ressourcen Ihrer Maschine verbraucht. Sehr gelegentlich findet es einen Beweis für die Hypothese und sagt es Ihnen ;-).“

Beweise für wahre Aussagen zu finden ist offenbar so schwierig, dass die Suche nur äußerst selten
oder – wenn überhaupt – erst nach sehr langer Zeit gelingt. Wir werden darauf eingehen
dazu ausführlicher in Kap. 4. Hier sollte jedoch nicht nur erwähnt werden
Computern, aber auch die meisten Menschen haben Schwierigkeiten, strenge formale Beweise zu finden.

Obwohl es offensichtlich ist, dass Computer allein in vielen Fällen nicht in der Lage sind, einen
Beweis zu finden, ist es die nächstbeste Lösung, Systeme zu bauen, die halbautomatisch arbeiten
und ermöglichen eine enge Zusammenarbeit mit dem Benutzer. Dadurch kann der Mensch seine eigenen besser anwenden
Kenntnisse über spezielle Anwendungsbereiche und schränken möglicherweise die Suche nach Beweisen ein.
Einer der erfolgreichsten interaktiven Beweiser für Prädikatenlogik höherer Ordnung ist Is abelle
[NPW02], ein gemeinsames Produkt der Universität Cambridge und der University of Cambridge
Technik, München.

Wer auf der Suche nach einem leistungsstarken Prüfer ist, sollte einen Blick auf die aktuellen
Ergebnisse des CASC (CADE ATP System Competition) [SS06] werfen.1 Hier werden wir fündig
der Gewinner von 2001 bis 2006 in den Kategorien PL1 und Clause Normalform war
Manchesters Prüfer Vampire, der mit einer Auflösungsvariante und einem Special arbeitet
Ansatz zur Gleichberechtigung. Das System Waldmeister des Max-Planck-Instituts in Saarbrücken ist
seit Jahren führend in der Gleichstellungsprüfung.

Die vielen Spitzenplätze deutscher Systeme am CASC zeigen, dass die deutsche Forschung
Gruppen im Bereich des automatisierten Theorembeweises spielen heute eine führende Rolle
sowie in der Vergangenheit.

## 3.7 Mathematische Beispiele

Wir wollen nun die Anwendung eines automatisierten Beweisers mit dem oben genannten Beweiser E
[Sch02] demonstrieren. E ist ein spezialisierter Gleichheitsprüfer, der stark schrumpft
den Suchraum durch eine optimierte Gleichbehandlung.

Wir wollen beweisen, dass links- und rechtsneutrale Elemente in einer Halbgruppe gleich sind.
Zunächst formalisieren wir den Anspruch Schritt für Schritt.

1CADE ist die jährliche „Conference on Automated Deduction“ [CAD] und ATP steht für „Automated Theorem Prover“.

**Definition 3.10 Eine Struktur (M,·), die aus einer Menge M mit einer zweistelligen inneren Operation „·“ besteht, heißt Halbgruppe, wenn das Gesetz der Assoziativität gilt**

$$\forall x \forall y \forall z\ (x \cdot y) \cdot z = x \cdot (y \cdot z)$$

**hält. Ein Element e ÿ M heißt linksneutral (rechtsneutral), wenn ÿxe · x = x (ÿxx · e = x).**

**Das muss noch gezeigt werden**

**Satz 3.7** ***Wenn eine Halbgruppe ein linksneutrales Element*** **el** ***und ein rechtsneutrales Element*** **er** hat , ***dann*** ist **el = er .**

**Zuerst beweisen wir den Satz semiformal durch intuitives mathematisches Denken. Offenbar gilt für alle x ÿ M, dass**

$$\mathbf{el} \cdot x = x \tag{3.8}$$

**Und**

$$x \cdot \mathbf{er} = x. \tag{3.9}$$

**Wenn wir in (3.8)** x = **er** und **in (3.9) x = el** setzen, **erhalten wir die beiden Gleichungen el · er = er und el · er = el . Die Verbindung dieser beiden Gleichungen ergibt**

$$\mathbf{el} = \mathbf{el} \cdot \mathbf{er} = \mathbf{er},$$

**was wir beweisen wollen. Im letzten Schritt haben wir übrigens die Tatsache ausgenutzt, dass Gleichheit symmetrisch und transitiv ist.**

**Bevor wir den automatisierten Beweis anwenden, führen wir den Auflösungsnachweis manuell durch. Zunächst formalisieren wir die negierte Abfrage und die Wissensbasis *KB,* bestehend aus den Axiomen als Klauseln in konjunktiver Normalform:**

| | |
|---|---|
| **(¬el = er)1** | **negierte Abfrage** |
| **(m(m(x, y), z) = m(x, m(y, z)))2 ( m(el,x) = x)3 (m(x, er ) = x)4** | |
| **Gleichheitsaxiome:** | |
| **(x = x)5** | **(Reflexivität)** |
| **(¬x = y ÿ y = x)6 (¬x = y ÿ ¬y = z ÿ x = z)7 (¬x = y ÿ m(x, z) = m (y, z))8 (¬x = y ÿ m(z, x) = m(z, y))9** | **(Symmetrie) (Transitivität) Substitution in m Substitution in m,** |

wobei die Multiplikation durch das zweistellige Funktionssymbol m dargestellt wird. Die Gleichheitsaxiome wurden analog zu (3.1) auf Seite 36 und (3.2) auf Seite 36 formuliert. Ein einfacher Auflösungsbeweis hat die Form

```
Res(3, 6, x6/m(el, x3), y6/x3): Res(7,          (x = m(el, x))10
10, x7/x10, y7/m(el, x10)): (¬m(el,x) = z ÿ x = z)11
Res(4, 11, x4/el, x11/er, z11/el): (er = el)12
Res(1, 12, ÿ): ().
```

Hier bedeutet beispielsweise Res(3, 6, x6/m(el, x3), y6/x3) das in der Auflösung von Klausel 3 mit Klausel 6, die Variable x aus Klausel 6 wird durch m(el, x3) mit ersetzt Variable x aus Satz 3. Analog wird y aus Satz 6 durch x aus ersetzt Klausel 3.

Nun wollen wir den Beweiser E auf das Problem anwenden. Die Klauseln werden transformiert in die Klausel Normalformsprache LOP durch die Zuordnung

(¬A1 ÿ···ÿ¬Am ÿ B1 ÿ···ÿ Bn) ÿ B1;...;Bn < ÿA1,...,Am.

Die Syntax von LOP stellt eine Erweiterung der PROLOG-Syntax (siehe Abschn. 5) dar Nicht-Horn-Klauseln. Somit erhalten wir als Eingabedatei für E

```
<- el = äh.                                  % Abfrage
m(m(X,Y),Z) = m(X,m(Y,Z)) . m(el,X) = X      % Assoziativität von m
m(X,er) = X              .                   % linksneutrales Element von m
                         .                   % rechtsneutrales Element von m
```

wobei Gleichheit durch das Prädikatssymbol gl modelliert wird. Ein Anruf beim Prüfer liefert

```
unixprompt> eproof halbgr1a.lop
# Problemstatus ermittelt, Beweisobjekt erstellt
# Beweise für den Problemstatus beginnen
        0 : [--equal(el, er)] : initial
        1 : [++equal(m(el,X1), X1)] : initial
        2: [++equal(m(X1,er), X1)]: initial
        3 : [++equal(el, er)] : pm(2,1)
        4: [--equal(el, el)]: rw(0,3)
        5: []: cn(4)
        6: []: 5: {Beweis}
# Der Beweis für den Problemstatus endet
```

Positive Literale werden durch ++ und negative Literale durch -- gekennzeichnet. In den Zeilen 0 bis 4, Mit Initial markiert, werden die Klauseln aus den Eingabedaten noch einmal aufgelistet. pm(a,b) steht für einen Auflösungsschritt zwischen Klausel a und Klausel b. Wir sehen darin den Beweis

Der von E gefundene Beweis ist dem manuell erstellten Beweis sehr ähnlich. Weil wir ausdrücklich Modellieren Sie die Gleichheit durch das Prädikat gl, kommen die besonderen Stärken von E nicht zum Tragen ins Spiel. Jetzt lassen wir die Gleichheitsaxiome weg und erhalten

```
<- el = äh.                                   % Abfrage
m(m(X,Y),Z) = m(X,m(Y,Z)) . m(el,X) = X       % Assoziativit\"{a}t v. m
m(X,er) = X               .                   % linksneutrales El. v. m
                          .                   % rechtsneutrales El. v. m
```

als Eingabe für den Prüfer.

Der Beweis wird auch kompakter. Wir sehen in der folgenden Ausgabe des beweisen, dass der Beweis im Wesentlichen aus einem einzigen Inferenzschritt auf die beiden relevanten Klauseln 1 und 2 besteht.

```
unixprompt> eproof halbgr1a.lop
# Problemstatus ermittelt, Beweisobjekt erstellt
# Beweise für den Problemstatus beginnen
        0 : [--equal(el, er)] : initial
        1 : [++equal(m(el,X1), X1)] : initial
        2: [++equal(m(X1,er), X1)]: initial
        3 : [++equal(el, er)] : pm(2,1)
        4: [--equal(el, el)]: rw(0,3)
        5 : : cn(4)
        6:      []     : 5 : {Beweis}
# Der Beweis für den Problemstatus endet
```

Der Leser könnte nun einen genaueren Blick auf die Fähigkeiten von E werfen (Übung 3.9 weiter) . Seite 55).

## 3.8 Anwendungen

In der Mathematik werden automatisierte Theorembeweiser für bestimmte Spezialaufgaben eingesetzt. Beispielsweise wurde der wichtige Vierfarbensatz der Graphentheorie erstmals bewiesen 1976 mit Hilfe eines Spezialprüfers. Automatisierte Prüfer spielen jedoch immer noch eine untergeordnete Rolle Rolle in der Mathematik.

Andererseits war zu Beginn der KI die Prädikatenlogik von großer Bedeutung für die Entwicklung von Expertensystemen in der Praxis. Aufgrund seiner Probleme Aufgrund der Modellierungsunsicherheit (siehe Abschn. 4.4) werden heute am häufigsten Expertensysteme entwickelt Verwendung anderer Formalismen.

Heutzutage spielt die Logik bei Verifikationsaufgaben eine immer wichtigere Rolle. Automatisch Programmverifikation ist derzeit ein wichtiges Forschungsgebiet zwischen KI und Software-Engineering. Heutzutage übernehmen immer komplexere Softwaresysteme Aufgaben

von immer größerer Verantwortungs- und Sicherheitsrelevanz. Hier ist ein Nachweis bestimmter Sicherheitseigenschaften eines Programms wünschenswert. Ein solcher Beweis kann nicht durch das Testen eines fertigen Programms erbracht werden, da es im Allgemeinen unmöglich ist, ein Programm auf alle möglichen Eingaben anzuwenden. Dies ist daher ein idealer Bereich für allgemeine oder sogar spezielle Inferenzsysteme. Unter anderem sind heute kryptografische Protokolle im Einsatz, deren Sicherheitseigenschaften automatisch überprüft wurden [FS97, Sch01]. Eine weitere Herausforderung für den Einsatz automatisierter Prüfer ist die Synthese von Software und Hardware. Zu diesem Zweck sollten Prüfer beispielsweise den Softwareentwickler bei der Generierung von Programmen aus Spezifikationen unterstützen.

Auch die Wiederverwendung von Software ist für viele Programmierer heutzutage von großer Bedeutung. Der Programmierer sucht nach einem Programm, das Eingabedaten mit bestimmten Eigenschaften entgegennimmt und ein Ergebnis mit gewünschten Eigenschaften berechnet. Ein Sortieralgorithmus akzeptiert Eingabedaten mit Einträgen eines bestimmten Datentyps und erstellt daraus eine Permutation dieser Einträge mit der Eigenschaft, dass jedes Element kleiner oder gleich dem Der Programmierer formuliert zunächst eine Spezifikation der Abfrage in PL1, bestehend aus zwei Teilen. Der erste Teil *PREQ* umfasst die Voraussetzungen, die erfüllt sein müssen, bevor das gewünschte Programm angewendet wird. Der zweite Teil *POSTQ* enthält die Nachbedingungen, die nach der Anwendung des gewünschten Programms gelten müssen.

Im nächsten Schritt muss eine Softwaredatenbank nach Modulen durchsucht werden, die diese Anforderungen erfüllen. Um dies formal zu überprüfen, muss die Datenbank für jedes Modul M eine formale Beschreibung der Vorbedingungen *PREM* und Nachbedingungen *POSTM* speichern. Eine Annahme über die Fähigkeiten der Module ist, dass sich die Vorbedingungen des Moduls aus den Vorbedingungen der Abfrage ergeben. Das muss es halten

$$PREQ \Rightarrow PREM .$$

Alle Bedingungen, die als Voraussetzung für die Anwendung des Moduls M erforderlich sind, müssen als Vorbedingungen in der Abfrage erscheinen. Wenn beispielsweise ein Modul in der Datenbank nur Listen mit Ganzzahlen akzeptiert, müssen Listen mit Ganzzahlen als Eingabe auch als Vorbedingungen in der Abfrage erscheinen. Eine zusätzliche Anforderung in der Abfrage, dass beispielsweise nur gerade Zahlen vorkommen, stellt kein Problem dar.

Darüber hinaus muss für die Nachbedingungen gelten, dass

$$POSTM \Rightarrow POSTQ.$$

Das heißt, nach Anwendung des Moduls müssen alle Attribute erfüllt sein, die die Abfrage erfordert. Wir zeigen nun die Anwendung eines Theorembeweisers auf diese Aufgabe an einem Beispiel aus [Sch01].

*Beispiel 3.8* VDM-SL, die Vienna Development Method Specification Language, wird häufig als Sprache zur Spezifikation von Vor- und Nachbedingungen verwendet. Gehen Sie davon aus, dass in der Softwaredatenbank die Beschreibung eines Moduls ROTATE verfügbar ist, das das erste Listenelement an das Ende der Liste verschiebt. Wir suchen ein Modul SHUFFLE, das eine beliebige Permutation der Liste erstellt. Die beiden Spezifikationen lauten

**ROTATE(l : *Liste)* lÿ : *Liste***
**vorab *wahr***
**Post**
**(l = [] ÿ l ' = [])ÿ**
**(l = [] ÿ l ' = (Schwanz l)^[Kopf l])**

**SHUFFLE(x : *Liste)* xÿ : *Liste***
**vorab *wahr***
**post ÿi : *Artikel*·**
**(ÿx1, x2 : *Liste* · x = x1^[i]^x2 ÿ**
**ÿy1, y2 : *Liste* · x ' = y1^[i]^y2)**

**Hier steht ~ für die Verkettung von Listen, und „·“ trennt Quantoren mit ihre Variablen vom Rest der Formel. Die Funktionen „Kopf l“ und „Schwanz l“ Wählen Sie jeweils das erste Element und den Rest aus der Liste aus. Die Spezifikation von SHUFFLE gibt an, dass jedes Listenelement i, das vor dem in der Liste (x) war Die Anwendung von SHUFFLE muss im Ergebnis (x) enthalten sein nach der Bewerbung und umgekehrt umgekehrt. Es muss nun gezeigt werden, dass die Formel *(PREQ* ÿ *PREM )* ÿ *(POSTM* ÿ *POSTQ)* ist eine Folge der Wissensdatenbank, die eine Beschreibung der enthält Datentyp *Liste.* Die Beweisaufgabe ergibt sich aus den beiden VDM-SL-Spezifikationen**

$$\ddot{y}l,l\ddot{y},x,x\ddot{y} : \textit{Liste} \cdot (l = x \ddot{y} l \quad \overset{\ddot{y}\,\ddot{y} = x}{} \quad \ddot{y}\,(w \ddot{y} w))\ddot{y}$$

$$(l = x \ddot{y} l \quad ' = x \quad ' \ddot{y}\,((l = [] \ddot{y} l \quad ' = [])\ddot{y}\,(l = [] \ddot{y} l \quad ' = (tl\ l)\hat{}[hd\ l])$$

$$\ddot{y}\,\ddot{y}i : \textit{Element} \cdot (\ddot{y}x1, x2 : \textit{Liste} \cdot x = x1\hat{}[i]\hat{}x2 \ddot{y}\,\ddot{y}y1, y2 : \textit{Liste} \cdot x \quad ' = y1\hat{}[i]\hat{}y2))),$$

**was dann mit dem Beweiser SETHEO bewiesen werden kann.**

**In den kommenden Jahren dürfte das *Semantic Web* eine wichtige Anwendung von PL1 darstellen. Die Inhalte des World Wide Web sollen interpretierbar werden nicht nur für Menschen, sondern auch für Maschinen. Zu diesem Zweck werden Websites eingerichtet eine Beschreibung ihrer Semantik in einer formalen Beschreibungssprache. Die Suche nach Informationen im Web werden dadurch deutlich effektiver als heute, wobei im Wesentlichen nur Textbausteine durchsuchbar sind.**

**Als Beschreibungssprachen werden entscheidbare Teilmengen der Prädikatenlogik verwendet. Die Entwicklung effizienter Kalküle für das Denken ist sehr wichtig und eng damit verbunden zu den Beschreibungssprachen. Eine Abfrage für eine zukünftige semantisch arbeitende Suchmaschine könnte (informell) lauten: Wo in der Schweiz nächsten Sonntag in Höhen unter Wird es auf 2000 Metern gutes Wetter und bestens präparierte Skipisten geben? Um eine solche Frage zu beantworten, ist ein Kalkül erforderlich, der in der Lage ist, sehr schnell zu arbeiten auf großen Mengen von Fakten und Regeln. Hier sind komplexe verschachtelte Funktionsterme weniger wichtig.**

**Als grundlegendes Beschreibungsgerüst wurde das World Wide Web Consortium entwickelt die Sprache RDF (Resource Description Framework). Aufbauend auf RDF ermöglicht die deutlich leistungsfähigere Sprache OWL (Web Ontology Language) die Beschreibung von Beziehungen zwischen Objekten und Objektklassen, ähnlich wie PL1 [SET09]. *Ontologien* sind Beschreibungen von Beziehungen zwischen möglichen Objekten.**

Eine Schwierigkeit beim Erstellen einer Beschreibung der unzähligen Websites ist der Arbeitsaufwand und auch die Überprüfung der Korrektheit der semantischen Beschreibungen. Hier können maschinelle Lernsysteme zur automatischen Generierung von Beschreibungen sehr hilfreich sein. Eine interessante Verwendung der „automatischen“ Generierung von Semantik im Web wurde von Luis von Ahn von der Carnegie Mellon University [vA06] vorgestellt. Er entwickelte Computerspiele, bei denen die über das Netzwerk verteilten Spieler gemeinsam Bilder mit Schlüsselwörtern beschreiben sollen. So wird den Bildern auf spielerische Art und Weise kostenlos eine Semantik zugewiesen. Der Leser kann die Spiele testen und einer Rede des Erfinders in Übung 3.10 auf Seite 55 lauschen.

## 3.9 Zusammenfassung

Wir haben die wichtigsten Grundlagen, Begriffe und Verfahren der Prädikatenlogik bereitgestellt und gezeigt, dass selbst eine der schwierigsten intellektuellen Aufgaben, nämlich der Beweis mathematischer Theoreme, automatisiert werden kann. Automatisierte Beweiser können nicht nur in der Mathematik, sondern insbesondere bei Verifikationsaufgaben in der Informatik eingesetzt werden. Für das alltägliche Denken ist die Prädikatenlogik jedoch in den meisten Fällen ungeeignet. Im nächsten und den folgenden Kapiteln zeigen wir seine Schwachstellen und einige interessante moderne Alternativen. Darüber hinaus zeigen wir in Kap. 5, dass man mit Logik und ihren prozeduralen Erweiterungen elegant programmieren ka

Wer sich für Logik erster Ordnung, Auflösung und andere Kalküle für automatisierte Beweiser interessiert, findet in [New00, Fit96, Bib82, Lov78, CL73] gute Anleitungen für Fortgeschrittene. Verweise auf Internetressourcen finden Sie auf der Website dieses Buches.

## 3.10 Übungen

Aufgabe 3.1 Gegeben seien das dreistellige Prädikat „Kind“ und das einstellige Prädikat „fe male“ aus Beispiel 3.2 auf Seite 35 . Definieren Sie: (a) Ein einstelliges Prädikat „männlich“.
(b) Ein zweistelliges Prädikat „Vater“ und „Mutter“. (c) Ein zweistelliges Prädikat „Geschwister“.
(d) Ein Prädikat „Eltern(x, y, z)“, das genau dann wahr ist, wenn x der Vater und y ist ist die Mutter von z.
(e) Ein Prädikat „Onkel(x, y)“, das genau dann wahr ist, wenn x der Onkel von y ist (verwenden Sie die bereits definierten Prädikate).
(f) Ein zweistelliges Prädikat „Vorfahre“ mit der Bedeutung: *Vorfahren sind Eltern, Großeltern usw. beliebig vieler Generationen.*

Aufgabe 3.2 Formulieren Sie die folgenden Aussagen in der Prädikatenlogik:
(a) Jeder Mensch hat einen Vater und eine Mutter. (b) Manche Menschen haben Kinder. (c) Alle Vögel fliegen.

(d) Es gibt ein Tier, das (einige) getreidefressende Tiere frisst. (e) Jedes Tier frisst Pflanzen oder pflanzenfressende Tiere, die viel kleiner sind als selbst.

Übung 3.3 Passen Sie Übung 3.1 auf Seite 54 an , indem Sie einstellige Funktionssymbole und Gleichheit anstelle von „Vater“ und „Mutter“ verwenden.

Aufgabe 3.4 Geben Sie prädikatenlogische Axiome für die zweistellige Relation „<“ als Gesamtordnung an. Für eine Gesamtordnung müssen wir Folgendes haben: (1) Zwei beliebige Elemente sind vergleichbar. (2) Es ist symmetrisch. (3) Es ist transitiv.

Aufgabe 3.5 Vereinheitlichen Sie (wenn möglich) die folgenden Begriffe und geben Sie die MGU und die resultierenden Begriffe an. (a) p(x, f (y)), p(f (z), u) (b) p(x, f (x)), p(y, y) (c) x = 4 ÿ 7 · x, cos y = z (d) x < 2 · x, 3 < 6 (e) q(f (x, y, z), f (g(w, w), g(x, x), g( y, y))), q(u, u)

Aufgabe 3.6 (a) Transformieren Sie Russells Paradoxon aus Beispiel 3.7 auf Seite 45 in CNF. (b) Zeigen Sie, dass die leere Klausel nicht durch Auflösung ohne Faktorisierung aus (3.7) auf Seite 45 abgeleitet werden kann . Versuchen Sie, dies intuitiv zu verstehen.

Übung 3.7 (a) Warum ist die Auflösung mit der Menge der Unterstützungsstrategie unvollständig? (b) Begründen Sie (ohne zu beweisen), warum die Unterstützungsstrategie vollständig wird, wenn *(KB* ÿ ¬Q)\SOS ist erfüllbar. (c) Warum ist die Lösung mit der reinen Literalregel vollständig?

ÿ Aufgabe 3.8 Formulieren und beweisen Sie mit Auflösung, dass in einer Halbgruppe mit mindestens zwei verschiedenen Elementen a,b, einem linksneutralen Element e und einem linksnullelement n diese beiden Elemente unterschiedlich sein müssen, das heißt, dass n = e. Verwenden Sie die Demodulation, die das Ersetzen von „Gleiches durch Gleiches“ ermöglicht.

Aufgabe 3.9 Besorgen Sie sich den Theorembeweiser E [Sch02] oder einen anderen Beweiser und beweisen Sie die folgenden Aussagen. Vergleichen Sie diese Beweise mit denen im Text. (a) Die Behauptung aus Beispiel 2.3 auf Seite 24. (b) Russells Paradoxon aus Beispiel 3.7 auf Seite 45. (c) Die Behauptung aus Aufgabe 3.8.

Übung 3.10 Testen Sie die Spiele www.espgame.org und www.peekaboom.org, die dabei helfen, Bildern im Web Semantik zuzuordnen. Hören Sie sich den Vortrag über Human Computing von Luis von Ahn an unter: http://video.google.de/videoplay?docid= -8246463980976635143.

# Grenzen der Logik

# 4

## 4.1 Das Suchraumproblem

Wie bereits an mehreren Stellen erwähnt, gibt es bei der Suche nach einem Beweis fast immer bei jedem Schritt viele (je nach Kalkül potenziell unendlich viele) Möglichkeiten zur Anwendung von Folgerungsregeln. Das Ergebnis ist das bereits erwähnte explosionsartige Wachstum des Suchraums (Abb. 4.1 auf Seite 58). Im schlimmsten Fall müssen alle diese Möglichkeiten ausprobiert werden, um den Beweis zu finden, was in der Regel nicht in angemessener Zeit möglich ist.

Wenn wir automatisierte Beweiser oder Inferenzsysteme mit Mathematikern oder menschlichen Experten vergleichen, die Erfahrung in speziellen Bereichen haben, machen wir interessante Beobachtungen. Einerseits können erfahrene Mathematiker Theoreme beweisen, die für automatisierte Beweiser weit außerhalb der Reichweite liegen. Andererseits führen automatisierte Prüfer Zehntausende von Schlussfolgerungen pro Sekunde durch. Im Gegensatz dazu führt ein Mensch vielleicht eine Schlussfolgerung pro Sekunde durch. Obwohl menschliche Experten auf der Objektebene (also bei der Durchführung von Schlussfolgerungen) viel langsamer sind, lösen sie scheinbar schwierige Probleme viel schneller.

Dafür gibt es mehrere Gründe. Wir Menschen verwenden intuitive Kalküle, die auf einer höheren Ebene funktionieren und oft viele der einfachen Schlussfolgerungen eines automatisierten Beweisers in einem Schritt ausführen. Darüber hinaus verwenden wir Lemmata, also abgeleitete wahre Formeln, die wir bereits kennen und die wir daher nicht jedes Mal neu beweisen müssen. Mittlerweile gibt es auch maschinelle Prüfer, die mit solchen Methoden arbeiten. Aber selbst sie können noch nicht mit menschlichen Experten konkurrieren.

Ein weiterer, viel wichtigerer Vorteil von uns Menschen ist die Intuition, ohne die wir keine schwierigen Probleme lösen könnten [PS09]. Der Versuch, die Intuition zu formalisieren, verursacht Probleme. Erfahrungen in angewandten KI-Projekten zeigen, dass in komplexen Bereichen wie der Medizin (siehe Abschn. 7.3) oder der Mathematik die meisten Experten nicht in der Lage sind, dieses intuitive Metawissen verbal zu formulieren, geschweige denn zu formalisieren. Deshalb können wir dieses Wissen nicht programmieren oder in Form von *Heuristiken in Kalküle integrieren.* Heuristiken sind Methoden, die den Weg zum Ziel in vielen Fällen erheblich vereinfachen oder verkürzen, in manchen Fällen (meist selten) aber auch erheblich verlänge

W. Ertel, *Einführung in die künstliche Intelligenz,* Bachelor-Themen in Informatik,
DOI 10.1007/978-0-85729-299-5_4, 


**Abb. 4.1 Mögliche Folgen der Explosion eines Suchraums**

zum Ziel. Die heuristische Suche ist nicht nur für die Logik, sondern allgemein für die Problemlösung in der KI wichtig und wird daher in Kap. ausführlich behandelt. 6.

Ein interessanter Ansatz, der seit etwa 1990 verfolgt wird, ist die Anwendung maschineller Lerntechniken auf das Erlernen von Heuristiken zur Steuerung der Suche nach Inferenzsystemen, die wir nun kurz skizzieren. Ein Auflösungsprüfer hat bei der Suche nach einem Beweis Hunderte oder mehr Möglichkeiten für Auflösungsschritte bei jedem Schritt, aber nur wenige führen zum Ziel. Ideal wäre es, wenn der Beweiser ein Orakel fragen könnte, welche zwei Klauseln er im nächsten Schritt verwenden soll, um den Beweis schnell zu finden. Es gibt Versuche, solche Proof-Directing-Module aufzubauen, die für den nächsten Schritt die verschiedenen Alternativen bewerten und dann die Alternative mit der besten Bewertung auswählen. Im Falle einer Auflösung könnte die Bewertung der verfügbaren Klauseln durch eine Funktion berechnet werden, die einen Wert basierend auf der Anzahl positiver Literale, der Komplexität der Begriffe usw. für jedes Paar auflösbarer Klauseln berechnet.

Wie kann diese Funktion implementiert werden? Da dieses Wissen „intuitiv“ ist, ist der Programmierer damit nicht vertraut. Stattdessen versucht man, die Natur zu kopieren und nutzt maschinelle Lernalgorithmen, um aus erfolgreichen Beweisen zu lernen [ESS89, SE90]. Die Attribute aller Klauselpaare, die an erfolgreichen Auflösungsschritten beteiligt sind, werden als positiv gespeichert, und die Attribute aller nicht erfolgreichen Auflösungen werden als negativ gespeichert. Anschließend wird unter Verwendung dieser Trainingsdaten und eines maschinellen Lernsystems ein Programm generiert, das Satzpaare heuristisch bewerten kann (siehe Abschn. 9.5).

Ein anderer, erfolgreicherer Ansatz zur Verbesserung des mathematischen Denkens ist gefolgt von interaktiven Systemen, die unter der Kontrolle des Benutzers arbeiten. Hier einer könnte Computeralgebraprogramme wie Mathematica, Maple oder Maxima nennen, das automatisch schwierige symbolische mathematische Manipulationen durchführen kann. Die Suche nach dem Beweis bleibt jedoch völlig dem Menschen überlassen. Deutlich mehr Unterstützung beim Beweis bietet die oben erwähnte interaktive Beweiserin Isabelle [NPW02]. suchen. Derzeit gibt es mehrere Projekte, darunter Omega [SB04] und MKM,1 für die Entwicklung von Systemen zur Unterstützung von Mathematikern bei Beweisen.

Zusammenfassend kann man sagen, dass aufgrund der Suchraumproblematik automatisiert wird Beweiser können heute nur relativ einfache Theoreme in speziellen Bereichen mit beweisen wenige Axiome.

## 4.2 Entscheidbarkeit und Unvollständigkeit

Die Prädikatenlogik erster Ordnung bietet ein leistungsfähiges Werkzeug zur Darstellung von Wissen und Argumentation. Wir wissen, dass es korrekte und vollständige Kalküle und Theo-Rem-Beweiser gibt. Beim Beweis eines Theorems, also einer wahren Aussage, ist ein solcher Beweiser sehr hilfreich hilfreich, da man der Vollständigkeit halber nach endlicher Zeit weiß, dass die Aussage ist wirklich wahr. Was ist, wenn die Aussage nicht wahr ist? Der Vollständigkeitssatz (Theorem 3.3 auf Seite 41) beantwortet diese Frage nicht.2 Konkret gibt es keinen Prozess die jede Formel aus PL1 in endlicher Zeit beweisen oder widerlegen kann, denn sie gilt

**Satz 4.1** ***Die Menge der gültigen Formeln in der Prädikatenlogik erster Ordnung ist semi-entscheidbar.***

Dieser Satz impliziert, dass es Programme (Theorembeweiser) gibt, die bei gegebener a Wenn Sie eine wahre (gültige) Formel als Eingabe verwenden, bestimmen Sie deren Wahrheit in endlicher Zeit. Wenn die Formel nicht stimmt gültig, es kann jedoch vorkommen, dass der Prüfer nie anhält. (Mit dieser Frage kann sich der Leser in Übung 4.1 auf Seite 65 auseinandersetzen. ) Aussagenlogik ist entscheidbar denn die Wahrheitstabellenmethode liefert alle Modelle einer Formel in endlicher Zeit. Tatsächlich ist die Prädikatenlogik mit Quantoren und verschachtelten Funktionssymbolen eine Sprache etwas zu mächtig, um entscheidbar zu sein.

Andererseits ist die Prädikatenlogik für viele Zwecke nicht leistungsfähig genug. Oft möchte man Aussagen über Mengen von Prädikaten oder Funktionen machen. Das funktioniert in PL1 nicht, da es nur Quantoren für Variablen kennt, nicht jedoch für Prädikate oder Funktionen.

Kurt Gödel zeigte kurz nach seinem Vollständigkeitssatz für PL1, dass die Vollständigkeit verloren geht, wenn wir PL1 auch nur minimal erweitern, um eine Logik höherer Ordnung zu konstruieren. Eine Logik erster Ordnung kann nur über Variablen quantifizieren. Eine Logik zweiter Ordnung kann dies auch

1www.mathweb.org/mathweb/demo.html.

2Gerade dieser Fall ist in der Praxis besonders wichtig, denn wenn ich bereits weiß, dass eine Aussage wahr ist, Ich brauche keinen Prüfer mehr.

über Formeln erster Ordnung quantifizieren, und eine Logik dritter Ordnung kann über Formeln zweiter Ordnung quantifizieren. Selbst wenn man nur das Induktionsaxiom für die natürlichen Zahlen hinzufügt, ist die Logik unvollständig. Die Aussage *„Wenn ein Prädikat* p(n) *für n gilt , dann gilt auch* p(n + 1) " oder

$$\ddot{y}pp(n) \ddot{y} p(n + 1)$$

ist ein Satz zweiter Ordnung, da er über ein Prädikat quantifiziert. Gödel bewies den folgenden Satz:

**Satz 4.2 (Gödels Unvollständigkeitssatz)** ***Jedes Axiomensystem für die natürlichen Zahlen mit Addition und Multiplikation (Arithmetik) ist unvollständig. Das heißt, es gibt wahre Aussagen in der Arithmetik, die nicht beweisbar sind.***

Gödels Beweis arbeitet mit der sogenannten Gödelisierung, bei der jede arithmetische Formel als Zahl kodiert wird. Es erhält eine eindeutige Gödel-Zahl. Zur Formulierung des Satzes wird nun die Gödelisierung verwendet

F = „Ich bin nicht beweisbar."

in der Sprache der Arithmetik. Diese Formel ist aus folgendem Grund wahr. Angenommen, F ist falsch. Dann können wir F beweisen und damit zeigen, dass F nicht beweisbar ist. Das ist ein Widerspruch. Somit ist F wahr und daher nicht beweisbar.

Der tiefere Hintergrund dieses Theorems besteht darin, dass mathematische Theorien (Axiomsysteme) und ganz allgemein Sprachen unvollständig werden, wenn die Sprache zu mächtig wird. Ein ähnliches Beispiel ist die Mengenlehre. Diese Sprache ist so mächtig, dass man damit Paradoxien formulieren kann. Dabei handelt es sich um Aussagen, die in sich widersprüchlich sind, wie etwa die Aussage, die wir bereits aus Beispiel 3.7 auf Seite 45 kennen, über die Friseure, die alle diejenigen rasieren, die sich nicht selbst rasieren (siehe Übung 4.2 auf Seite 65).3 Das Dilemma besteht darin , dass mit Sprachen die aussagekräftig genug sind, um Mathematik und interessante Anwendungen zu beschreiben, schmuggeln wir Widersprüche und Unvollständigkeiten durch die Hintertür. Dies bedeutet jedoch nicht, dass Logiken höherer Ordnung für formale Methoden völlig ungeeignet sind. Es gibt sicherlich formale Systeme sowie Beweiser für Logik

## 4.3 Der fliegende Pinguin

Anhand eines einfachen Beispiels demonstrieren wir ein grundlegendes Problem der Logik und mögliche Lösungsansätze. Angesichts der Aussagen

3Viele weitere logische Paradoxien finden sich in [Wie].

**Abb. 4.2 Der fliegende Pinguin Tweety**

**1. Tweety ist ein Pinguin**
**2. Pinguine sind Vögel**
**3. Vögel können**
**fliegen Formalisiert in PL1 ergibt die Wissensdatenbank *KB :***

$$\textbf{Pinguin(tweety)}$$
$$\textbf{\textit{Pinguin(x)}} \ddot{y} \textbf{\textit{Vogel(x)}}$$
$$\textbf{\textit{Vogel(x)}} \ddot{y} \textbf{Fliege(x)}$$

**Daraus lässt sich (z. B. mit Auflösung) *„fly(tweety)“* ableiten (Abb. 4.2).4 Offensichtlich ist die Formalisierung der Flugattribute von Pinguinen unzureichend. Wir versuchen es mit der Zusatzaussage , dass *Pinguine nicht fliegen können***

$$\textbf{\textit{Pinguin(x)}} \ddot{y} \textbf{\textit{¬fly(x)}}$$

**Daraus** lässt sich **¬*fly(tweety)* ableiten. Aber *Fly(tweety)* ist immer noch wahr. Die Wissensbasis ist daher inkonsistent. Hier bemerken wir ein wichtiges Merkmal der Logik, nämlich die Monotonie. Obwohl wir ausdrücklich darauf hinweisen, dass Pinguine nicht fliegen können, lässt sich die Gegenseite dennoch ableiten.**

**Definition 4.1 Eine Logik heißt monoton, wenn für eine beliebige Wissensbasis *KB*** und eine beliebige Formel ÿ die Menge der aus ***KB* ableitbaren Formeln eine Teilmenge der aus *KB*** ableitbaren Formeln **ÿ ÿ ist.**

**4Die formale Ausführung dieses und des folgenden einfachen Beweises kann dem Leser überlassen werden (Übung 4.3 auf Seite 65).**

**Wird ein Formelsatz erweitert, so können nach der Erweiterung weiterhin alle bisher ableitbaren Aussagen bewiesen werden, ggf. können auch weitere Aussagen bewiesen werden. Die Menge der beweisbaren Aussagen wächst also monoton, wenn die Menge der Formeln erweitert wird. Für unser Beispiel bedeutet das, dass die Erweiterung der Wissensbasis nie zum Ziel führen wird. Wir modifizieren daher *KB ,* indem wir die offensichtlich falsche Aussage *„(alle) Vögel können fliegen“* durch die genauere Aussage *„(alle) Vögel außer Pinguinen können fliegen“* ersetzen und erhalten als *KB2* die folgenden Klauseln:**

**Pinguin(tweety)**
***Pinguin(x)* ÿ *Vogel(x)***
***Vogel(x)* ÿ *¬Pinguin(x)* ÿ Fliege(x)**
***Pinguin(x)* ÿ *¬Fliege(x)***

**Nun ist die Welt offenbar wieder in Ordnung.** Wir können ***¬fly(tweety)*** **ableiten , aber nicht *fly(tweety),* denn dafür bräuchten wir *¬penguin(x),* was allerdings nicht ableitbar ist. Solange es auf dieser Welt nur Pinguine gibt, herrscht Frieden. Jeder normale Vogel macht jedoch sofort Probleme. Wir möchten den Raben Abraxas (aus dem deutschen Buch „Die kleine Hexe“) hinzufügen und erhalten**

***Rabe(abraxas)***
***Rabe(x)* ÿ *Vogel(x)***
**Pinguin(tweety)**
***Pinguin(x)* ÿ *Vogel(x)***
***Vogel(x)* ÿ *¬Pinguin(x)* ÿ Fliege(x)**
***Pinguin(x)* ÿ *¬Fliege(x )***

**Über die Flugeigenschaften von Abraxas können wir nichts sagen, da wir vergessen haben zu formulieren, dass Raben keine Pinguine sind. Daher erweitern wir *KB3* auf KB4:**

***Rabe(abraxas)***
***Rabe(x)* ÿ *Vogel(x)***
***Rabe(x)* ÿ *¬Pinguin(x)***
**Pinguin(tweety)**
***Pinguin(x)* ÿ *Vogel(x)***
***Vogel(x)* ÿ *¬Pinguin(x)* ÿ Fliege(x )**
***Pinguin(x)* ÿ *¬fly(x)***

**Dass Raben keine Pinguine sind, was für den Menschen selbstverständlich ist, muss hier explizit hinzugefügt werden. Für den Aufbau einer Wissensbasis mit allen rund 9.800 Vogelarten weltweit muss daher für jede Vogelart (außer Pinguine) angegeben werden, dass sie nicht zu den Pinguinen gehört. Für alle anderen Ausnahmen wie den Strauß müssen wir analog vorgehen.**

**Für jedes Objekt in der Wissensdatenbank werden zusätzlich zu seinen Attributen alle Attribute, die es nicht hat, müssen aufgelistet werden.**

Um dieses Problem zu lösen, wurden verschiedene Formen nichtmonotoner Logik entwickelt, die es ermöglichen, Wissen (Formeln) aus der Wissensbasis zu entfernen.
Unter dem Namen *Default-Logik* wurden Logiken entwickelt, die es ermöglichen, Objekten Attribute zuzuweisen, die gültig sind, solange keine anderen Regeln verfügbar sind. Im Tweety-Beispiel wäre die Regel „ Vögel *können fliegen“* eine solche *Standardregel.* Trotz großer Bemühungen haben sich diese Logiken aufgrund semantischer und praktischer Probleme derzeit nicht durchgesetzt.

Monotonie kann besonders bei komplexen Planungsproblemen, bei denen sich die Welt verändern kann, unangenehm sein. Wenn zum Beispiel ein blaues Haus rot gestrichen wird, dann ist es danach rot. Eine Wissensdatenbank wie z

$$Farbe(Haus, Blau)$$
$$Farbe(Haus,Rot)$$
$$Farbe(x, y) \ddot{y} Farbe(x, y)$$

lässt den Schluss zu, dass das Haus nach dem Anstrich rot und blau ist. Das hier bei der Planung auftretende Problem wird als *Rahmenproblem bezeichnet.* Eine Lösung hierfür ist die in Abschn. 2.1 vorgestellte Situationsrechnung. 5.6.

Ein interessanter Ansatz zur Modellierung von Problemen wie dem Tweety-Beispiel ist die Wahrscheinlichkeitstheorie. Die Aussage „Alle Vögel können fliegen“ ist falsch. Eine Aussage wie etwa „Fast alle Vögel können fliegen“ ist richtig. Diese Aussage wird genauer, wenn wir eine Wahrscheinlichkeit für „Vögel können fliegen“ angeben. Dies führt zur *probabilistischen Logik,* die heute einen wichtigen Teilbereich der KI und ein wichtiges Werkzeug zur Modellierung von Unsicherheit darstellt (siehe Kap. 7).

## 4.4 Modellierungsunsicherheit

Zweiwertige Logik kann und sollte nur Umstände modellieren, in denen es wahre, falsche und keine anderen Wahrheitswerte gibt. Für viele Aufgaben des alltäglichen Denkens ist die zweiwertige Logik daher nicht aussagekräftig genug. Die Regel

$$Vogel(x) \ddot{y} Fliege(x)$$

gilt für fast alle Vögel, für einige ist sie jedoch falsch. Wie bereits erwähnt, ermöglicht die Arbeit mit Wahrscheinlichkeiten eine genaue Formulierung der Unsicherheit. Die Aussage *„99 % aller Vögel können fliegen“* kann durch den Ausdruck formalisiert werden

$$P(Vogel(x) \ddot{y} Fliege(x)) = 0{,}99.$$

Im Kap. 7 werden wir sehen, dass es hier besser ist, mit bedingten Wahrscheinlichkeiten wie zu arbeiten

$$P(Fliege|Vogel) = 0{,}99.$$

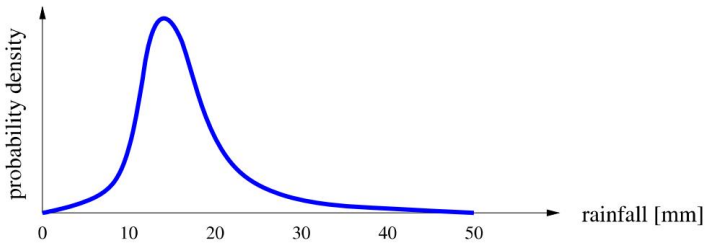



**Abb. 4.3 Wahrscheinlichkeitsdichte des kontinuierlichen variablen Niederschlags**

Mit Hilfe von *Bayes'schen Netzwerken* können komplexe Anwendungen mit vielen Variablen realisiert werden auch modelliert werden.

Für die Aussage „Das Wetter ist schön“ bedarf es eines anderen Modells . Hier ist es oft Es macht keinen Sinn, in Begriffen von wahr und falsch zu sprechen. Die Variable *Weather_is_nice* sollte nicht binär modelliert werden, sondern kontinuierlich mit Werten, zum Beispiel in das Intervall [0, 1]. Weather_is_nice = 0.7 bedeutet dann *„Das Wetter ist ziemlich schön“. Fuzzy-Logik,* die ebenfalls in Kap. vorgestellt wird. 7, wurde für diese Art von entwickelt kontinuierliche (Fuzzy-)Variable.

Auch die Wahrscheinlichkeitstheorie bietet die Möglichkeit, Aussagen über die Wahrscheinlichkeit stetiger Variablen zu treffen. Eine Aussage im Wetterbericht wie *„Es gibt „Eine hohe Wahrscheinlichkeit, dass es etwas Regen geben wird“* könnte beispielsweise genau als Wahrscheinlichkeitsdichte der Form formuliert werden

$$P\,(\textit{Niederschlag} = X) = Y$$

und grafisch etwa wie in Abb. 4.3 dargestellt.

Diese sehr allgemeine und sogar visualisierbare Darstellung beider Arten von Unsicherheit, die wir diskutiert haben, ermöglicht es zusammen mit der induktiven Statistik und der Theorie der Bayes'schen Netzwerke im Prinzip, beliebige probabilistische Antworten zu geben Abfragen.

Wahrscheinlichkeitstheorie und Fuzzy-Logik sind nicht direkt mit Prädikaten vergleichbar Logik, weil sie keine Variablen oder Quantoren zulassen. Sie können daher als angesehen werden Erweiterungen der Aussagenlogik, wie in der folgenden Tabelle gezeigt.

| Formalismus | Anzahl Wahrheitswerte | Wahrscheinlichkeiten ausdrückbar |
|---|---|---|
| Aussagelogik | 2 | – |
| Fuzzy-Logik | ÿ | – |
| Diskrete probabilistische Logik | N | Ja |
| *Kontinuierliche probabilistische Logik* | ÿ | *Ja* |

## 4.5 Übungen

**ÿ Übung 4.1**

**(a) Mit dem folgenden (falschen) Argument könnte man behaupten, dass PL1 entscheidbar ist:**
***Wir nehmen einen vollständigen Beweiskalkül für PL1. Damit können wir in endlicher Zeit einen Beweis für jede wahre Formel finden. Für jede andere Formel ÿ gehe ich wie folgt vor: Ich wende den Kalkül auf ¬ÿ an und zeige, dass ¬ÿ wahr ist. Somit ist ÿ falsch. Somit kann ich jede Formel in PL1 beweisen oder widerlegen.*** **Finden Sie den Fehler im Argument und ändern Sie ihn, damit er**

**richtig wird. (b) Konstruieren Sie einen Entscheidungsprozess für die Menge der wahren und unerfüllbaren Forme PL1.**

**Übung 4.2 (a)**

**Gegeben sei die Aussage *„Es gibt einen Friseur, der jeden rasiert, der sich nicht selbst rasiert.“***
**Überlegen Sie, ob sich dieser Friseur selbst rasiert. (b) Sei M = {x|x /ÿ**
**x}. Beschreiben Sie diese Menge und überlegen Sie, ob M sich selbst enthält.**

**Übung 4.3 Verwenden Sie einen automatisierten Theorembeweiser (z. B. E [Sch02]) und wenden Sie ihn auf alle fünf verschiedenen Axiomatisierungen des Tweety-Beispiels aus Abschn. 4.3 an. 4.3. Validieren Sie die Aussagen des Beispiels.**

# Logikprogrammierung mit PROLOG

# 5

Im Vergleich zu klassischen Programmiersprachen wie C oder Pascal schafft es Logic
Es ist möglich, Beziehungen elegant, kompakt und deklarativ auszudrücken. Automatisiert
Theorembeweiser sind sogar in der Lage zu entscheiden, ob eine Wissensbasis logisch ist
bringt eine Abfrage mit sich. Beweisrechnung und Wissen werden in der Wissensdatenbank gespeichert
streng getrennt. Als Eingabe kann eine im Abschnitt Normalform beschriebene Formel verwendet werden
Daten für jeden Theorembeweiser, unabhängig von der verwendeten Beweisrechnung. Das ist großartig
Wert für das Denken und die Darstellung von Wissen.

Will man Algorithmen implementieren, die zwangsläufig über prozedurale Komponenten
verfügen, reicht eine rein deklarative Beschreibung oft nicht aus. Robert Kowalski, einer
Einer der Pioniere der Logikprogrammierung hat dies mit der Formel verdeutlicht

*Algorithmus = Logik + Kontrolle.*

Diese Idee wurde in der Sprache PROLOG verwirklicht. PROLOG wird verwendet in
viele Projekte, vor allem in den Bereichen KI und Computerlinguistik. Wir geben jetzt eine
Kurze Einführung in diese Sprache, Vorstellung der wichtigsten Konzepte, Darstellung der Sprache
Stärken und vergleichen Sie es mit anderen Programmiersprachen und Theoremprüfern.
Wer einen vollständigen Programmierkurs sucht, wird auf Lehrbücher wie z
[Bra11, CM94] und die Handbücher [Wie04, Dia04].

Die Syntax der Sprache PROLOG erlaubt nur Horn-Klauseln. Logische Notation
und die Syntax von PROLOG sind in der folgenden Tabelle gegenübergestellt:

| PL1 / Klausel Normalform | PROLOG | Beschreibung |
|---|---|---|
| (¬A1 ÿ ··· ÿ ¬Am ÿ B) | B :- A1, ... , Am. | Regel |
| (A1 ÿ ··· ÿ Am) ÿ B | B:- A1, ..., Am. | Regel |
| A | A. | Tatsache |
| (¬A1 ÿ ··· ÿ ¬Am) | ?- A1, ... , Am. | Abfrage |
| ¬(A1 ÿ ··· ÿ Bin) | ?- A1, ... , Am. | Abfrage |

W. Ertel, *Einführung in die Künstliche Intelligenz,*
Bachelor-Themen in Informatik,
DOI 10.1007/978-0-85729-299-5_5, 

```
1 Kind (Oscar, Karen, Frank). 2 Kinder
(Mary, Karen, Frank). 3 Kinder (Eve,
Anne, Oscar). 4 Kinder (Henry,
Anne, Oscar). 5 Kind (Isolde, Anne,
Oscar). 6 Kinder (Clyde, Mary, Oscarb).
7 8 Kind(X,Z,Y) :- Kind(X,Y,Z). 9


10 Nachkomme(X,Y) :- Kind(X,Y,Z). 11
Nachkomme(X,Y) :- Kind(X,U,V), Nachkomme(U,Y).
```

**Abb. 5.1 PROLOG-Programm mit Familienbeziehungen**

**Hier** sind **A1,...,Am,A,B Literale. Die Literale werden wie in PL1 aus Prädikatssymbolen mit Termen als Argumenten konstruiert. Wie wir in der obigen Tabelle sehen können, gibt es in PROLOG keine Negationen im streng logischen Sinne, da das Vorzeichen eines Literals durch seine Position im Satz bestimmt wird.**

## 5.1 PROLOG-Systeme und Implementierungen

**Eine Übersicht über aktuelle PROLOG-Systeme finden Sie in der Linksammlung auf der Homepage dieses Buches. Dem Leser empfehlen wir die sehr leistungsfähigen und frei verfügbaren (unter GNU Public-Lizenzen) Systeme GNU-PROLOG [Dia04] und SWI-PROLOG. Für die folgenden Beispiele wurde SWI-PROLOG [Wie04] verwendet.**

**Die meisten modernen PROLOG-Systeme arbeiten mit einem Interpreter, der auf der *Warren Abstract Machine (WAM) basiert.* Der PROLOG-Quellcode wird in sogenannten WAM-Code kompiliert, der dann vom WAM interpretiert wird. Die schnellsten Implementierungen eines WAM verwalten bis zu 10 Millionen logische Schlussfolgerungen pro Sekunde (LIPS) auf einem 1-GH**

## 5.2 Einfache Beispiele

**Wir beginnen mit den Familienbeziehungen aus Beispiel 3.2 auf Seite 35. Die kleine Wissensbasis *KB* ist – ohne die Fakten für das Prädikat weiblich – als PROLOG-Programm mit dem Namen rel.pl in Abb. 5.1 codiert.**

**Mit dem Befehl kann das Programm im PROLOG-Interpreter geladen und übersetzt werden**

```
?- [rel].
```

Eine erste Abfrage gibt den Dialog zurück

?- Kind (Eve, Oscar, Anne).

Ja

mit der richtigen Antwort Ja. Wie kommt es zu dieser Antwort? Für die Abfrage „?- child(eve,oscar,anne).“ Es gibt sechs Fakten und eine Regel mit demselben Prädikat im Satzkopf. Nun wird versucht, die Abfrage und jedes der komplementären Literale in den Eingabedaten in der Reihenfolge ihres Auftretens zu vereinheitlichen. Wenn eine der Alternativen fehlschlägt, wird zum letzten Verzweigungspunkt zurückverfolgt und die nächste Alternative getestet. Da die Vereinheitlichung bei jedem Fakt fehlschlägt, wird die Abfrage mit der rekursiven Regel in Zeile 8 vereinheitlicht. Nun versucht das System, das Unterziel child(eve,anne,oscar)
Die Abfrage

?- Nachkomme(X,Y).

X = Oscar
Y = Karen

Ja

wird mit der ersten gefundenen Lösung beantwortet, so wie sie ist

?- Nachkomme(clyde,Y).

Y = Maria

Ja

Die Abfrage

?- Nachkomme(clyde,karen).

wird allerdings nicht beantwortet. Der Grund dafür ist die Klausel in Zeile 8, die die Symmetrie des untergeordneten Prädikats angibt. Diese Klausel ruft sich rekursiv auf, ohne dass eine Terminierung möglich ist. Dieses Problem kann mit dem folgenden neuen Programm gelöst werden (Fakten wurden hier weggelassen).

```
1 Nachkomme(X,Y) :- Kind(X,Y,Z). 2 Nachkomme(X,Y) :-
Kind(X,Z,Y). 3 Nachkomme(X,Y) :- Kind(X,U,V),
Nachkomme(U,Y).
```

**Aber jetzt die Frage**

**?- Kind (Eve, Oscar, Anne).**

**wird nicht mehr richtig beantwortet, da die Symmetrie von child in den letzten beiden Variablen nicht mehr gegeben ist. Für beide Probleme findet sich im Programm eine Lösung**

```
1 child_fact(oscar,karen,franz). 2
child_fact(mary,karen,franz). 3
child_fact(eva,anne,oscar). 4
child_fact(henry,anne,oscar). 5
child_fact(isolde,anne,oscar). 6
child_fact(clyde,mary,oscarb). 7 8 child(X,Z,Y) :-

child_fact(X,Y,Z). 9 child(X,Z,Y) :- child_fact(X,Z,Y).
10 11 Nachkomme(X,Y) :- Kind(X,Y,Z). 12

Nachkomme(X,Y) :- Kind(X,U,V), Nachkomme(U,Y).
```

**Durch die Einführung des neuen Prädikats child_fact für die Fakten ist das Prädikat child nicht mehr rekursiv. Allerdings ist das Programm nicht mehr so elegant und einfach wie die – logisch korrekte – erste Variante in Abb. 5.1 auf Seite 68, die zur Endlosschleife führt. Der PROLOG-Programmierer muss, wie auch in anderen Sprachen, auf die Verarbeitung achten und Endlosschleifen vermeiden. PROLOG ist nur eine Programmiersprache und kein Theorembeweiser.**

**Wir müssen hier zwischen *deklarativer* und *prozeduraler Semantik* von PROLOG-Programmen unterscheiden. Die deklarative Semantik ist durch die logische Interpretation der Hornsätze gegeben. Die prozedurale Semantik hingegen wird durch die Ausführung des PROLOG-Programms definiert, die wir nun genauer betrachten wollen. Die Ausführung des Programms aus Abb. 5.1 auf Seite 68 mit der Abfrage child(eve,oscar,anne) ist in Abb. 5.2 auf Seite 71 als Suchbaum** dargestellt.1 **Die Ausführung beginnt oben links mit der Abfrage. Jede Kante stellt einen möglichen SLD-Auflösungsschritt mit einem komplementären unifizierbaren Literal dar. Während der Suchbaum durch die rekursive Regel unendlich tief wird, wird die PROLOG-Ausführung beendet, da die Fakten vor der Regel in den Eingabedaten auftreten.**

**Bei der Abfrage „descendent(clyde,karen)“ hingegen wird die PROLOG-Ausführung nicht beendet. Wir können dies deutlich an dem in Abb. 5.3 auf Seite 71** dargestellten ***Und-Oder-Baum*** erkennen . **In dieser Darstellung führen die Zweige, dargestellt durch,**  **, vom Kopf eines Satzes zu den Unterzielen. Da alle Teilziele einer Klausel gelöst werden müssen, handelt es sich hierbei um *und-Zweige.* Alle anderen Zweige sind *oder-Zweige,* von denen mindestens einer mit seinen übergeordneten Knoten vereinbar sein muss. Die beiden skizzierte**

**1Die Konstanten wurden aus Platzgründen abgekürzt.**

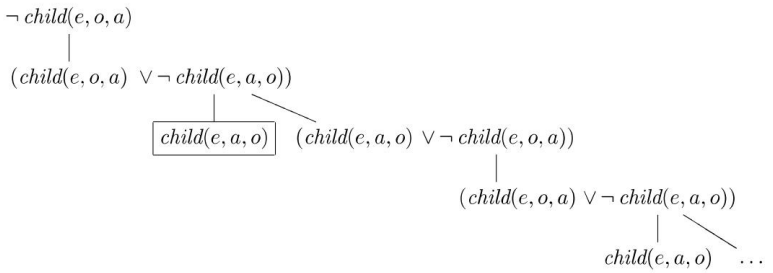



**Abb. 5.2 PROLOG-Suchbaum für child(eve,oscar,anne)**

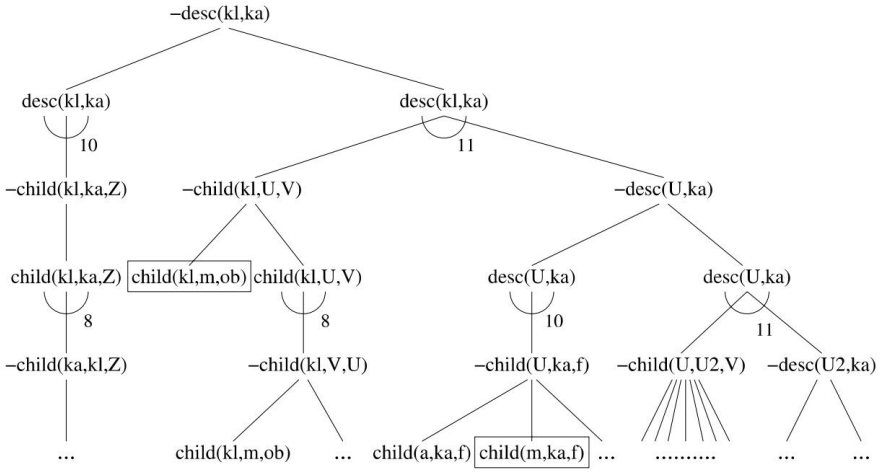



**Abb. 5.3 Und-Oder-Baum für desc(clyde,karen)**

**Lösung der Anfrage. Allerdings terminiert der PROLOG-Interpreter hier nicht, denn er arbeitet mit einer Tiefensuche mit Backtracking (siehe Abschn. 6.2.2) und wählt somit zunächst den unendlich tiefen Pfad ganz links.**

## 5.3 Ausführungskontrolle und Verfahrenselemente

**Wie wir am Beispiel einer Familienbeziehung gesehen haben, ist es wichtig, die Ausführung von PROLOG zu kontrollieren. Insbesondere die Vermeidung unnötiger Rückschritte kann zu erheblichen Effizienzsteigerungen führen. Ein Mittel hierzu ist der *Schnitt.* Indem wir ein Ausrufezeichen in eine Klausel einfügen, können wir verhindern, dass wir an diesem Punkt zurückgehen. Im folgenden Programm berechnet das Prädikat max(X,Y,Max) das Maximum der beiden Zahlen X und Y.**

```
1 max(X,Y,X) :- X >= Y. 2 max(X,Y,Y) :-
X < Y.
```

Wenn der erste Fall (erster Satz) zutrifft, wird der zweite nicht erreicht. Trifft der erste Fall hingegen nicht zu, dann ist die Bedingung im zweiten Fall wahr, was bedeutet, dass sie nicht überprüft werden muss. Zum Beispiel in der Abfrage

```
?- max(3,2,Z), Z > 10.
```

Backtracking wird eingesetzt, da Z=3 ist und die zweite Klausel auf max getestet wird, was zum Scheitern verurteilt ist. Daher ist ein Zurückverfolgen dieser Stelle nicht erforderlich. Dies können wir mit einem Schnitt optimieren:

```
1 max(X,Y,X) :- X >= Y, !. 2 max(X,Y,Y).
```

Daher wird die zweite Klausel nur dann aufgerufen, wenn sie wirklich notwendig ist, also wenn die erste Klausel fehlschlägt. Allerdings macht diese Optimierung das Programm schwerer verständlich.

Eine weitere Möglichkeit zur Ausführungskontrolle ist das eingebaute Prädikat fail, das niemals wahr ist. Im Beispiel der Familienbeziehung können wir mit der Abfrage ganz einfach alle Kinder und deren Eltern ausdrucken

```
?- child\_fact(X,Y,Z), write(X), write(' ist ein Kind von '), write(Y), write(' und '), write(Z),
write('.' ), nl, scheitern.
```

Die entsprechende Ausgabe ist

```
Oscar ist ein Kind von Karen und Frank. Mary ist ein Kind
von Karen und Frank. Eve ist ein Kind von Anne und
Oscar.
...
NEIN.
```

wobei das Prädikat nl einen Zeilenumbruch in der Ausgabe verursacht. Was wäre am Ende die Ausgabe ohne die Verwendung des Fail-Prädikats?

Mit derselben Wissensbasis wird die Abfrage „?- child_fact(ulla,X,Y)“ durchgeführt. würde zu der Antwort Nein führen, da es keine Fakten über Ulla gibt. Diese Antwort ist logisch nicht korrekt. Insbesondere lässt sich nicht beweisen, dass es kein Objekt mit dem Namen ulla gibt. Hier würde der Beweiser E korrekt antworten: „Kein Beweis gefunden.“ Wenn PROLOG also mit Nein antwortet, bedeutet dies nur, dass die Abfrage Q dies nicht kar

bewiesen werden. Dafür muss ¬Q jedoch nicht unbedingt bewiesen werden. Dieses Verhalten wird als *Negation als Fehler bezeichnet.*

Die Beschränkung auf Horn-Klauseln stellt in den meisten Fällen kein großes Problem dar. Es ist jedoch wichtig für die Verfahrensausführung mittels SLD-Auflösung (Abschn. 2.5). Durch das einzeln ermittelte positive Literal pro Klausel haben die SLD-Auflösung und damit die Ausführung von PROLOG-Programmen einen eindeutigen Einstiegspunkt in die Klaus Nur so ist eine reproduzierbare Ausführung von Logikprogrammen und damit eine wohldefinierte prozedurale Semantik möglich.

Tatsächlich gibt es durchaus Problemstellungen, die nicht durch Horn-Klauseln beschrieben werden können. Ein Beispiel ist Russells Paradoxon aus Beispiel 3.7 auf Seite 45, das die Nicht-Horn-Klausel *(shaves(barber,X)* ÿ *shaves(X,* X)) enthält.

## 5.4 Listen

Als Hochsprache verfügt PROLOG wie die Sprache LISP über den praktischen generischen Listendatentyp. Eine Liste mit den Elementen A, 2, 2, B, 3, 4, 5 hat die Form

```
[A,2,2,B,3,4,5]
```

Das Konstrukt [Head|Tail] trennt das erste Element (Head) vom Rest (Tail) der Liste. Mit der Wissensdatenbank

```
Liste([A,2,2,B,3,4,5]).
```

PROLOG zeigt den Dialog an

```
?- list([H|T]).

H=A
T = [2, 2, B, 3, 4, 5]

Ja
```

Durch die Verwendung verschachtelter Listen können wir beliebige Baumstrukturen erstellen. Zum Beispiel die beiden Bäume


kann durch die Listen [b,c] bzw. [a,b,c] und die beiden Bäume dargestellt werden

durch die Listen [[e,f,g],[h],d] bzw. [a,[b,e,f,g],[c,h],d].
In den Bäumen, in denen die inneren Knoten Symbole enthalten, ist das Symbol der Kopf des Liste und die untergeordneten Knoten sind das Ende.

Ein schönes, elegantes Beispiel für die Listenverarbeitung ist die Definition des Prädikats append(X,Y,Z) zum Anhängen der Liste Y an die Liste X. Das Ergebnis wird in Z gespeichert. Die entsprechendes PROLOG-Programm liest

```
1 anhängen([],L,L).
2 append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).
```

Dies ist eine deklarative (rekursive) logische Beschreibung der Tatsache, dass L3 daraus resultiert Anhängen von L2 an L1. Gleichzeitig erledigt dieses Programm aber auch die Arbeit wenn es aufgerufen wird. Der Anruf

```
?- append([a,b,c],[d,1,2],Z).
```

gibt die Substitution Z = [a, b, c, d, 1, 2] zurück, genau wie der Aufruf

```
?- anhängen(X,[1,2,3],[4,5,6,1,2,3]).
```

ergibt die Substitution X = [4, 5, 6]. Hier stellen wir fest, dass append kein a ist Zwei-Stellen-Funktion, sondern eine Drei-Stellen-Beziehung. Eigentlich können wir auch das eingeben „Ausgabeparameter“ Z und fragen Sie, ob er erstellt werden kann.

Auch das Umkehren der Reihenfolge der Elemente einer Liste lässt sich elegant beschreiben und si durch das rekursive Prädikat gleichzeitig programmiert

```
1 nrev([],[]).
2 nrev([H|T],R) :- nrev(T,RT), append(RT,[H],R).
```

Dies reduziert die Umkehrung einer Liste auf die Umkehrung einer Liste, die aus einem Element besteht kleiner. Tatsächlich ist dieses Prädikat aufgrund des Append-Aufrufs sehr ineffizient.
Dieses Programm ist als *Naive Reverse* bekannt und wird oft als PROLOG-Benchmark verwendet (siehe Übung 5.6 auf Seite 81). Die Dinge laufen besser, wenn man mit einem Provisorium fortfährt Speicher, der als Akkumulator bezeichnet wird, wie folgt:

| Aufführen | Akkumulator |
|---|---|
| [A B C D] [] | |
| [b,c,d] | [A] |
| [CD] | [b,a] |
| [D] | [c,b,a] |
| [] | [d,c,b,a] |

**Das entsprechende PROLOG-Programm liest**

```
1 accrev([],A,A). 2
accrev([H|T],A,R) :- accrev(T,[H|A],R).
```

## 5.5 Selbstmodifizierende Programme

**PROLOG-Programme werden nicht vollständig kompiliert, sondern vom WAM interpretiert. Daher ist es möglich, Programme zur Laufzeit zu ändern. Ein Programm kann sich sogar selbst ändern. Mit Befehlen wie „Assert“ und „Retract“ können Fakten und Regeln zur Wissensbasis hinzugefügt oder daraus entnommen werden.**

**Eine einfache Anwendung der Variante Asserta ist das Hinzufügen abgeleiteter Fakten am Anfang der Wissensbasis mit dem Ziel, eine wiederholte, möglicherweise zeitaufwendige Ableitung zu vermeiden (siehe Übung 5.8 auf Seite 81). Wenn wir in unserem Familienbeziehungsbeispiel die beiden Regeln für den Prädikatsnachkommen durch ersetzen**

```
1 :- dynamischer Nachkomme/2. 2
Nachkomme(X,Y) :- Kind(X,Y,Z), Asserta(Nachkomme(X,Y)). 3 Nachkommen(X,Y) :- Kind(X,U,V),
Nachkommen(U,Y), 4 Asserta(Nachkommen(X,Y)).
```

**dann werden alle abgeleiteten Fakten für dieses Prädikat in der Wissensbasis gespeichert und somit in Zukunft nicht erneut abgeleitet. Die Abfrage**

**?- Nachkomme (Clyde, Karen).**

**führt zur Addition der beiden Tatsachen**

**Nachkomme (Clyde, Karen). Nachkomme (Mary, Karen).**

**Durch die Manipulation von Regeln mit „Assert“ und „Retract“ können sogar Programme geschrieben werden, die sich selbst vollständig ändern.** Diese Idee wurde unter dem Begriff ***genetische Programmierung*** **bekannt . Es ermöglicht den Aufbau beliebig flexibler Lernprogramme. In der Praxis zeigt sich jedoch, dass eine Änderung des Codes durch Ausprobieren aufgrund der Vielzahl sinnloser Änderungsmöglichkeiten selten zu einer Leistungssteigerung führt. Die systematische Änderung von Regeln macht die Programmierung hingegen so viel komplexer, dass solche Programme, die ihren eigenen Code umfassend modifizieren, bisher keinen Erfolg hatten. Im Kap. 8 werden wir zeigen, wie maschinelles Lernen recht erfolgreich war. Allerdings werden hier nur sehr begrenzte Änderungen am Programmcode vorgenommen.**

```
1 start :- action(state(left,left,left,left),
2                 state(right,right,right,right)).
3
4 Aktionen (Start, Ziel): –
                plan(Start,Ziel,[Start],Pfad),
5 6 nl,write('Lösung:'),nl,
7 write_path(Pfad).
8 % write_path(Path), fehlgeschlagen. 9          % aller Lösungen ausgegeben

10 Plan (Start, Ziel, Besucht, Pfad): –
11 go(Start,Weiter),
12 sicher(Weiter),
13 \+ member(Next,Visited), % not(member(...))
14          plan(Next,Goal,[Next|Visited],Path).
15 Plan(Ziel,Ziel,Pfad,Pfad).
16
17 go(state(X,X,Z,K),state(Y,Y,Z,K)):-across(X,Y). % Bauer, Wolf
18 go(state(X,W,X,K),state(Y,W,Y,K)):-across(X,Y). % Bauer, Ziege
19 go(state(X,W,Z,X),state(Y,W,Z,Y)):-across(X,Y). % Bauer, Kohl
20 go(state(X,W,Z,K),state(Y,W,Z,K)):-across(X,Y). % Bauer
21
22 quer (links, rechts).
23 quer (rechts, links).
24
25 sicher(Zustand(B,W,Z,K)):- über(W,Z), über(Z,K).
26 sicher(Zustand(B,B,B,K)).
27 sicher(Zustand(B,W,B,B)).
```

**Abb. 5.4 PROLOG-Programm für das Landwirt-Wolf-Ziege-Kohl-Problem**

## 5.6 Ein Planungsbeispiel

***Beispiel 5.1*** **Das folgende Rätsel dient als Problemstellung für ein typisches PRO LOG-Programm.**

> **Ein Bauer möchte einen Kohl, eine Ziege und einen Wolf über einen Fluss bringen, aber sein Boot ist so klein dass er sie nur einzeln überqueren kann. Der Bauer dachte darüber nach und sagte es dann selbst: „Wenn ich zuerst den Wolf auf die andere Seite bringe, dann frisst die Ziege den Kohl. Wenn ich Transportiere zuerst den Kohl, dann wird die Ziege vom Wolf gefressen. Was soll ich machen?"**

**Dies ist eine Planungsaufgabe, die wir mit etwas Überlegung schnell lösen können. Der** Das in Abb. 5.4 dargestellte PROLOG-Programm **wird nicht ganz so schnell erstellt.**

**Das Programm arbeitet mit Begriffen der Form state(Farmer,Wolf,Goat, Kohl), die den aktuellen Zustand der Welt beschreiben. Die vier Variablen mit mögliche Werte left, right geben den Standort der Objekte an. Der zentrale rekursive Prädikatplan erstellt zunächst einen Nachfolgezustand und testet anschließend mit go dessen Sicherhe mit sicher und wiederholt dies rekursiv, bis der Start- und Zielzustand gleich sind (in Programmzeile 15). Die bereits besuchten Staaten werden im gespeichert drittes Argument des Plans. Mit dem eingebauten Prädikat-Member wird geprüft, ob die state Next wurde bereits besucht. Wenn ja, wird es nicht versucht. Die Definition der Hier fehlt das Prädikat write_path für die Aufgabe, den gefundenen Plan auszugeben. Es wird als Übung für den Leser empfohlen (Übung 5.2 auf Seite 80). Für erste Pro-**

**Gramm testet das Literal write_path(Path), das durch write(Path) ersetzt werden kann. Für die Abfrage „?- start.“ Wir bekommen die Antwort**

**Lösung:**
**Bauer und Ziege von links nach rechts**
**Bauer von rechts nach links**
**Bauer und Wolf von links nach rechts**
**Bauer und Ziege von rechts nach links**
**Bauer und Kohl von links nach rechts**
**Bauer von rechts nach links**
**Bauer und Ziege von links nach rechts**

**Ja**

**Zum besseren Verständnis beschreiben wir die Definition von Plan in der Logik:**

**ÿz *plan(z,* z) ÿ ÿs ÿz ÿn [go(s, n) ÿ *sicher(n)* ÿ *plan(n,* z) ÿ *plan(s,* z)]**

**Diese Definition fällt deutlich prägnanter aus als in PROLOG. Dafür gibt es zwei Gründe. Zum einen ist das Ergebnis des entdeckten Plans für die Logik unwichtig. Darüber hinaus ist es nicht wirklich notwendig zu prüfen, ob der nächste Staat bereits besucht wurde, wenn unnötige Fahrten den Landwirt nicht stören. Wenn jedoch \+ member(...) im PROLOG-Programm weggelassen wird, liegt eine Endlosschleife vor und PROLOG findet möglicherweise keinen Zeitplan, selbst wenn einer vorhanden ist. Ursache hierfür ist die rückwärtsverkettende Suchstrategie von PROLOG, die nach dem Prinzip der Tiefensuche (Abschn. 6.2.2) immer auf Unterziele einzeln ohne Einschränkung der Rekursionstiefe arbeitet und daher unvollständig ist. Dies würde einem Theorembeweiser mit einem vollständi**

**Wie bei allen Planungsaufgaben ändert sich der Zustand der Welt, wenn von Schritt zu Schritt Aktionen ausgeführt werden. Dies legt nahe, den Zustand als Variable an alle Prädikate zu senden, die vom Zustand der Welt abhängen, beispielsweise im Prä Die Zustandsübergänge erfolgen im Prädikat go. Dieser Ansatz wird als *Situationskalkül* bezeichnet [RN10]. Eine interessante Erweiterung zum Erlernen von Aktionssequenzen in teilweise beobachtbaren, nichtdeterministischen Welten werden wir i**

## 5.7 Constraint-Logik-Programmierung

**Die Programmierung von Scheduling-Systemen, bei denen viele (zum Teil komplexe) logische und numerische Bedingungen erfüllt sein müssen, kann mit herkömmlichen Programmiersprachen sehr aufwändig und schwierig sein. Genau hier könnte Logik nützlich sein. Man schreibt einfach alle logischen Bedingungen in PL1 und gibt dann eine Abfrage ein. Normalerweise scheitert dieser Ansatz kläglich. Der Grund ist das in Abschn. 2.1 diskutierte Pinguinproblem. 4.3. Die Tatsache, dass Penguin(tweety) dafür sorgt, das**

WAHR. Es schließt jedoch nicht aus, dass raven(tweety) auch wahr ist. Dies mit zusätzlichen Axiomen auszuschließen ist sehr unpraktisch (Abschn. 4.3).

*Constraint Logic Programming (CLP),* das die explizite Formulierung von Einschränkungen für Variablen ermöglicht, bietet einen eleganten und sehr effizienten Mechanismus zur Lösung dieses Problems. Der Interpreter überwacht ständig die Ausführung des Programms auf Einhaltung aller Einschränkungen. Der Programmierer wird vollständig von der Kontrolle der Einschränkungen entlastet, was in vielen Fällen die Programmierung erheblich vereinfachen kann. Dies kommt im folgenden Zitat von Eugene C. Freuder aus [Fre97] zum Ausdruck:

> Die Constraint-Programmierung stellt einen der bislang größten Ansätze der Informatik zum Heiligen Gral des Programmierens dar: Der Benutzer gibt das Problem vor, der Computer löst es.

Ohne auf die Theorie des Constraint Satisfaction Problem (CSP) einzugehen, wenden wir den CLP-Mechanismus von GNU-PROLOG auf das folgende Beispiel an.

*Beispiel 5.2* Die Sekretärin des Albert-Einstein-Gymnasiums muss einen Plan für die Zuteilung von Räumen für Abschlussprüfungen ausarbeiten. Ihm liegen folgende Informationen vor: Die vier Lehrer Mayer, Hoover, Miller und Smith geben in den aufsteigend nummerierten Räumen 1, 2, 3 und 4 Tests für die Fächer Deutsch, Englisch, Mathematik und Physik. Jeder Lehrer gibt einen Test für genau einen ab Motiv in genau einem Raum. Darüber hinaus weiß er Folgendes über die Lehrer und ihre Fächer.

1. Herr Mayer testet nie in Raum 4.
2. Herr Miller testet immer Deutsch.
3. Herr Smith und Herr Miller führen keine Tests in benachbarten Räumen durch.
4. Frau Hoover testet Mathematik.
5. Physik wird immer im Raum Nummer 4 geprüft.
6. Deutsch und Englisch werden im Raum 1 nicht geprüft.

Wer führt in welchem Raum einen Test durch?

Ein GNU-PROLOG-Programm zur Lösung dieses Problems ist in Abb. 5.5 auf Seite 79 dargestellt. Dieses Programm arbeitet mit den Variablen Mayer, Hoover, Miller, Smith sowie German, English, Math, Physics, die jeweils eine ganze Zahl annehmen können Geben Sie als Raumnummer einen Wert von 1 bis 4 ein (Programmzeilen 2 und 5). Eine Bindung Mayer = 1 und Deutsch = 1 bedeutet, dass Herr Mayer den Deutschtest in Raum 1 durchführt. Die Zeilen 3 und 6 sorgen dafür, dass die einzelnen vier Variablen unterschiedliche Werte annehmen. Zeile 8 sorgt dafür, dass im Lösungsfall allen Variablen ein konkreter Wert zugewiesen wird. Diese Zeile ist hier nicht zwingend erforderlich. Bei mehreren Lösungen würden jedoch nur Intervalle ausgegeben. In den Zeilen 10 bis 16 werden die Randbedingungen angegeben, in den restlichen Zeilen werden die Raumnummern für alle Lehrer und alle Fächer in einem einfachen Format ausgegeben.

Das Programm wird mit „['raumplan.pl'].“ in GNU-PROLOG geladen, und mit „Start“. Wir erhalten die Ausgabe

```
[3,1,2,4]
[2,3,1,4]
```

```
 1 Start:-
 2 fd_domain([Mayer, Hoover, Miller, Smith],1,4),
 3 fd_all_different([Mayer, Miller, Hoover, Smith]),
 4
 5 fd_domain([Deutsch, Englisch, Mathematik, Physik],1,4),
 6 fd_all_different([Deutsch, Englisch, Mathematik, Physik]),
 7
 8 fd_labeling([Mayer, Hoover, Miller, Smith]),
 9
10 Mayer #\=4, 11 Miller                    % Mayer nicht in Raum 4
#= Deutsch, 12 dist(Miller,Smith)          % Miller testet Deutsch
#>= 2, 13 Hoover #= Mathematik, 14 Physik   % Abstand Miller/Smith >= 2
#= 4, 15 Deutsch #\=1, 16                  % Hoover testet Mathematik
Englisch #\=1 , 17 nl,                     % Physik in Raum 4
                                           % Deutsch nicht in Zimmer 1
                                           % Englisch nicht in Zimmer 1

18 write([Mayer, Hoover, Miller, Smith]), nl,
19 write([Deutsch, Englisch, Mathematik, Physik]), nl.
```

**Abb. 5.5 CLP-Programm für das Raumplanungsproblem**

Etwas praktischer dargestellt haben wir folgende Raumaufteilung:

| Zimmernr. | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Lehrer | Hoover | Miller | Mayer | Smith |
| Thema | Mathematik | Deutsch | Englische | Physik |

GNU-PROLOG verfügt, wie die meisten anderen CLP-Sprachen, über eine sogenannte *endliche Domäne Constraint-Löser,* mit dem Variablen ein endlicher Bereich ganzer Zahlen zugeordnet werden kann. Dabei muss es sich nicht zwangsläufig um ein Intervall wie im Beispiel handeln. Wir können auch eine Liste eingeben von Werten. Als Übung wird der Benutzer aufgefordert, in Übung 5.9 auf Seite 81 eine zu erstellen CLP-Programm, zum Beispiel mit GNU-PROLOG, für ein nicht ganz so einfaches Logikrätsel. Dieses angeblich von Einstein geschaffene Rätsel kann mit einem CLP ganz einfach gelöst werden System. Wenn wir dagegen versuchen würden, PROLOG ohne Einschränkungen zu verwenden, könnten wir es tun leicht die Zähne ausknirschen. Wer mit PROLOG bzw. eine elegante Lösung findet ein Prüfer, bitte lassen Sie es den Weg zum Autor finden.

## 5.8 Zusammenfassung

Vereinheitlichung, Listen, deklarative Programmierung und die relationale Sicht von Prozeduren, bei denen ein Argument eines Prädikats sowohl als Eingabe als auch als Ausgabe fungieren kann, ermöglichen die Entwicklung kurzer, eleganter Programme für viele Probleme. Viele Programme wären deutlich länger und daher schwieriger zu verstehen, wenn es in einer prozeduralen Form geschrieben wird Sprache. Darüber hinaus sparen diese Sprachfunktionen dem Programmierer Zeit. Daher ist PROLOG auch ein interessantes Werkzeug für Rapid Prototyping, insbesondere für KI

Anwendungen. Die CLP-Erweiterung von PROLOG ist nicht nur bei Logikrätseln hilfreich, sondern auch bei vielen Optimierungs- und Planungsaufgaben.

Seit seiner Erfindung im Jahr 1972 hat sich PROLOG in Europa neben prozeduralen Sprachen zu einer der führenden KI-Programmiersprachen Europas entwickelt. In den USA hingegen dominiert die einheimische Sprache LISP den KI-Markt.

PROLOG ist kein Theorembeweiser. Dies ist beabsichtigt, denn ein Programmierer muss die Verarbeitung einfach und flexibel steuern können und kommt mit einem Theorembeweis nicht weit. Andererseits ist PROLOG allein für den Beweis mathematischer Theoreme nicht sehr hilfreich. Allerdings gibt es durchaus interessante Theoretiker, die in PROLOG programmiert sind.

Als weiterführende Literatur werden [Bra11] und [CM94] sowie die empfohlen Handbücher [Wie04, Dia04] und zum Thema CLP [Bar98].

## 5.9 Übungen

**Aufgabe 5.1** Versuchen Sie, den Satz aus Abschn. zu beweisen. 3.7 über die Gleichheit links- und rechtsneutraler Elemente von Halbgruppen mit PROLOG. Welche Probleme treten auf? Was ist die Ursache dafür?

**Aufgabe 5.2**

(a) Schreiben Sie ein Prädikat write_move(+State1, +State2), das für jede Bootsüberquerung einen Satz wie „Bauer und Wolf kreuzen von links nach rechts“ ausgibt. State1 und State2 sind Begriffe der Form state(Farmer, Wolf, Goat, Cabbage). (b)

Schreiben Sie ein rekursives Prädikat write_path(+Path), das das Prädikat write_move(+State1, +State2) aufruft und alle Aktionen des Farmers ausgibt.

**Aufgabe 5.3**

(a) Auf den ersten Blick ist die Variable Path im Prädikatplan des PROLOG-Programms aus Beispiel 5.1 auf Seite 76 unnötig, da sie offenbar nirgendwo geändert wird. Wofür wird es benötigt?

(b) Wenn wir im Beispiel am Ende der Aktion ein Fail hinzufügen, werden alle Lösungen als Ausgabe ausgegeben. Warum wird jetzt jede Lösung zweimal gedruckt? Wie können Sie das verhindern?

**Aufgabe 5.4**

(a) Zeigen Sie durch Testen, dass der Theorembeweiser E (im Gegensatz zu PROLOG) unter Berücksichtigung der Wissensbasis aus Abb. 5.1 auf Seite 68 die Abfrage „?- Deszendent(clyde, karen)“ beantwortet. korrekt. Warum das? (b)

Vergleichen Sie die Antworten von PROLOG und E für die Abfrage „?- Deszendent(X, Y).“.

**Aufgabe 5.5 Schreiben Sie ein möglichst kurzes PROLOG-Programm, das 1024 Einsen ausgibt.**

**ÿ Übung 5.6 Untersuchen Sie das Laufzeitverhalten des naiven Umkehrprädikats.**

**(a) Führen Sie PROLOG mit der Trace-Option aus und beobachten Sie die rekursiven Aufrufe von nrev, append und accrev.**

**(b) Berechnen Sie die asymptotische Zeitkomplexität von append(L1,L2,L3), also die Abhängigkeit der Laufzeit von der Länge der Liste für große Listen. Nehmen Sie an, dass der Zugriff auf den Kopf einer beliebigen Liste eine konstante Zeit benötigt. (c) Berechnen Sie die Zeitkomplexität von nrev(L,R). (d) Berechnen Sie die zeitliche Komplexität von accrev(L,R). (e) Bestimmen Sie experimentell die Zeitkomplexität der Prädikate nrev, append und accrev, indem Sie beispielsweise Zeitmessungen durchführen (Zeit(+Ziel) liefert Rückschlüsse und CPU-Zeit).**

**Übung 5.7 Verwenden Sie Funktionssymbole anstelle von Listen, um die in Abschn. 5.7 angegebenen Bäume darzustellen. 5.4 auf Seite 73.**

**ÿ Übung 5.8 Die Fibonacci-Folge ist rekursiv definiert durch fib(0) = 1, fib(1) = 1 und fib(n) = *fib(n* ÿ 1) + *fib(n* ÿ 2).** (a) Definieren Sie ein rekursives PROLOG-Prädikat fib(N,R), das ***fib(N )*** **und berechnet gibt es in R zurück.**

**(b) Bestimmen Sie die Laufzeitkomplexität des Prädikats fib theoretisch und anhand von Messung.**

**(c) Ändern Sie Ihr Programm durch die Verwendung von Asserta so, dass unnötige Schlussfolgerungen nicht mehr ausgeführt werden. (d) Bestimmen Sie die Laufzeitkomplexität des geänderten Prädikats theoretisch und durch Messung (beachten Sie, dass dies davon abhängt, ob fib zuvor aufgerufen wurde).**

**(e) Warum ist Fib mit Asserta auch schneller, wenn es zum ersten Mal gestartet wird? nach dem Start von PROLOG?**

**ÿ Übung 5.9 Das folgende typische Logikrätsel wurde angeblich von Albert Einstein geschrieben. Darüber hinaus behauptete er angeblich, dass nur 2 % der Weltbevölkerung in der Lage seien, das Problem zu lösen. Folgende Aussagen werden gemacht. • Es gibt fünf Häuser, jedes in einer anderen Farbe gestrichen. • Jedes Haus wird von einer Person mit einer anderen Nationalität bewohnt. • Jeder Bewohner bevorzugt ein bestimmtes Getränk, raucht eine bestimmte Zigarettenmarke und hat ein bestimmtes Haustier. • Keiner der fünf Menschen trinkt das Gleiche, raucht das Gleiche oder hat das Gleiche dasselbe Haustier.**

**• Hinweise: – Der Brite wohnt im roten Haus.**

**– Der Schwede hat einen Hund.**

**– Der Däne trinkt gern Tee.**

**– Das grüne Haus befindet sich links vom weißen Haus.**

– Der Besitzer des Gewächshauses trinkt Kaffee.
– Die Person, die Pall Mall raucht, hat einen Vogel.
– Der Mann, der im mittleren Haus wohnt, trinkt Milch.
– Der Besitzer des gelben Hauses raucht Dunhill.
– Der Norweger wohnt im ersten Haus.
– Der Marlboro-Raucher wohnt neben demjenigen, der eine Katze hat.
– Der Mann mit dem Pferd wohnt neben dem, der Dunhill raucht.
– Der Winfield-Raucher trinkt gerne Bier.
– Der Norweger wohnt neben dem blauen Haus.
– Der Deutsche raucht Rothmanns.
– Der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt.

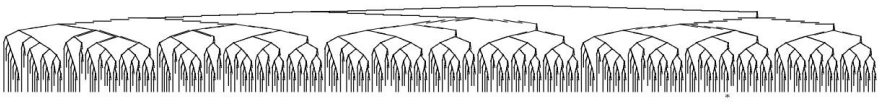
Frage: Wem gehört der Fisch? (a) Lösen Sie das Rätsel zunächst manuell. (b) Schreiben Sie ein CLP-Programm (zum Beispiel mit GNU-PROLOG), um das Rätsel zu lösen. Orientieren Sie sich am Raumplanungsproblem in Abb. 5.5 auf Seite 79.

# Suche, Spiele und Problemlösung

# 6

## 6.1 Einführung

Die Suche nach einer Lösung in einem extrem großen Suchbaum stellt für fast alle Inferenzsysteme ein Problem dar. Vom Ausgangszustand aus gibt es viele Möglichkeiten für den ersten Inferenzschritt. Für jede dieser Möglichkeiten gibt es im nächsten Schritt wiederum viele Möglichkeiten und so weiter. Selbst im Beweis einer sehr einfachen Formel aus [Ert93] mit drei Horn-Klauseln mit jeweils höchstens drei Literalen hat der Suchbaum für die SLD-Auflösung die folgende Form:



Der Baum wurde in einer Tiefe von 14 abgeschnitten und hat eine Lösung im mit ÿ markierten Blattknoten. Eine Darstellung ist nur aufgrund des geringen Verzweigungsfaktors von höchstens zwei und eines Cutoffs bei Tiefe 14 überhaupt möglich. Bei realistischen Problemen können Verzweigungsfaktor und Tiefe der ersten Lösung deutlich größer werden.

Angenommen, der Verzweigungsfaktor ist eine Konstante von 30 und die erste Lösung liegt in der Tiefe 50. Der Suchbaum hat 3050 ÿ 7,2 × 1073 Blattknoten. Die Anzahl der Inferenzschritte ist jedoch noch größer, da nicht nur jeder Blattknoten, sondern auch jeder innere Knoten des Baums einem Inferenzschritt entspricht. Daher müssen wir die Knoten über alle Ebenen addieren und so die Gesamtanzahl der Knoten des Suchbaums erhalten

$$\sum_{d=0}^{50} 30^d = \frac{1 \ddot{y} 3051}{1 \ddot{y} 30} = 7{,}4 \times 1073 \quad ,$$

was die Knotenanzahl nicht wesentlich ändert. Offensichtlich befinden sich fast alle Knoten dieses Suchbaums auf der letzten Ebene. Wie wir sehen werden, ist dies im Allgemeinen der Fall. Nun aber zurück zum Suchbaum mit den 7,4 × 1073 Knoten. Nehmen wir an, wir hätten 10.000 Computer, von denen jeder eine Milliarde Schlussfolgerungen pro Sekunde durchführen kann, und das könnten wi

W. Ertel, *Einführung in die künstliche Intelligenz,* Bachelor-Themen in Informatik,
DOI 10.1007/978-0-85729-299-5_6, © Springer-Verlag London Limited 2011

Verteilen Sie die Arbeit kostenlos auf alle Computer. Die Gesamtrechenzeit für alle 7,4 × 1073-Schlussfolgerungen wäre ungefähr gleich

$$\frac{7{,}4 \times 1073 \text{ Schlussfolgerungen}}{\times 109 \text{ Schlussfolgerungen/Sek}} = 7{,}4 \times 1060 \text{ Sek.} \approx 2{,}3 \times 1053 \text{ Jahre, } 10000$$

Das ist etwa 1043 Mal so viel Zeit wie das Alter unseres Universums. Durch diese einfache Gedankenübung können wir schnell erkennen, dass es keine realistische Chance gibt, einen solchen Suchraum mit den uns zur Verfügung stehenden Mitteln dieser Welt vollständig zu durchsuchen. Darüber hinaus waren die Annahmen zur Größe des Suchraums völlig realistisch. Beim Schach beispielsweise gibt es in einer typischen Situation über 30 mögliche Züge, und eine Partie mit 50 Halbrunden ist relativ kurz.

Wie kann es dann sein, dass es gute Schachspieler gibt – und heutzutage auch gute Schachcomputer? Wie kann es sein, dass Mathematiker Beweise für Theoreme finden, bei denen der Suchraum noch viel größer ist? Offensichtlich nutzen wir Menschen intelligente Strategien, die den Suchraum drastisch verkleinern. Der erfahrene Schachspieler wird ebenso wie der erfahrene Mathematiker durch bloße Beobachtung der Situation viele Handlungen sofort als sinnlos ausschließen. Aufgrund seiner Erfahrung ist er in der Lage, verschiedene Maßnahmen hinsichtlich ihres Nutzens für das Erreichen des Ziels zu bewerten. Oft geht man nach Gefühl vor. Fragt man einen Mathematiker, wie er einen Beweis gefunden hat, antwortet er möglicherweise, dass ihm die Intuition im Traum gekommen sei. In schwierigen Fällen stellen viele Ärzte die Diagnose rein nach dem Gefühl, basierend auf allen bekannten Symptomen. Gerade in schwierigen Situationen fehlt oft eine formale Lösungstheorie, die eine optimale Lösung garantiert. Bei alltäglichen Problemen, wie etwa der Suche nach einer entlaufenen Katze in Abb. 6.1 auf Seite 85, spielt die Intuition eine große Rolle. Mit dieser Art *heuristischer Suchmethode* beschäftigen wir uns in Abschn. 6.3 und beschreiben zusätzlich Prozesse, mit denen Com

Zunächst müssen wir jedoch verstehen, wie die *uninformierte Suche,* also das blinde Ausprobieren aller Möglichkeiten, funktioniert. Wir beginnen mit einigen Beispielen.

*Beispiel 6.1* Mit dem 8-Puzzle, einem klassischen Beispiel für Suchalgorithmen [Nil98, RN10], lassen sich die verschiedenen Algorithmen sehr anschaulich veranschaulichen. Quadrate mit den Zahlen 1 bis 8 werden in einer 3 × 3-Matrix wie in Abb. 6.2 auf Seite 86 verteilt . Ziel ist es, eine bestimmte Reihenfolge der Quadrate zu erreichen, zum Beispiel in aufsteigender Reihenfolge nach Zeilen, wie in Abb. dargestellt .6.2 auf Seite 86. In jedem Schritt kann ein Quadrat nach links, rechts, oben oder unten in den leeren Raum verschoben werden . Der leere Raum bewegt sich also in die entsprechende Gegenrichtung. Für die Analyse des Suchraums ist es sinnvoll, immer die möglichen Bewegungen des leeren Feldes zu betrachten.

Der Suchbaum für einen Startzustand ist in Abb. 6.3 auf Seite 86 dargestellt . Wir können sehen, dass der Verzweigungsfaktor zwischen zwei, drei und vier wechselt. Gemittelt über jeweils zwei Ebenen erhalten wir einen durchschnittlichen Verzweigungsfaktor1 von $\approx 2{,}83$. Wir sehen, dass sich jeder Zustand zwei Ebenen tiefer mehrmals wiederholt, da bei einer einfachen uninformierten Suche jede Aktion im nächsten Schritt rückgängig gemacht werden kann.

---

1Der durchschnittliche Verzweigungsfaktor eines Baumes ist der Verzweigungsfaktor, den ein Baum mit konstantem Verzweigungsfaktor, gleicher Tiefe und gleicher Anzahl an Blattknoten hätte.

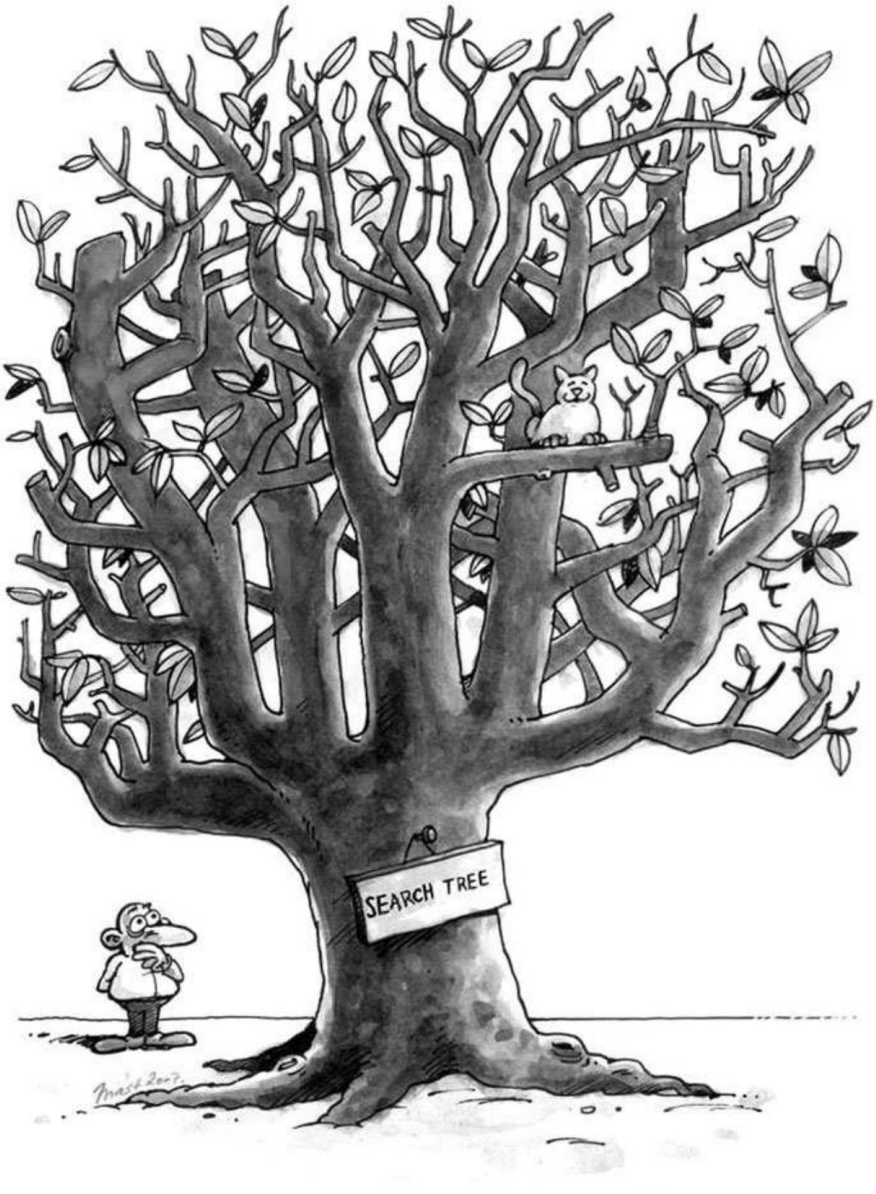



**Abb. 6.1 Ein stark beschnittener Suchbaum – oder: „Wo ist meine Katze?“**

**Wenn wir Zyklen der Länge 2 nicht zulassen, erhalten wir für denselben Startzustand die Suchbaum dargestellt in Abb. 6.4 auf Seite 86. Der durchschnittliche Verzweigungsfaktor wird um etwa 1 reduziert und beträgt 1,8,2**

**2Bei einem 8-Puzzle hängt der durchschnittliche Verzweigungsfaktor vom Ausgangszustand ab (siehe Aufgabe 6.2) . Seite 110).**

**Abb. 6.2 Mögliche Ausgangs- und Zielzustände des 8er-Puzzles**

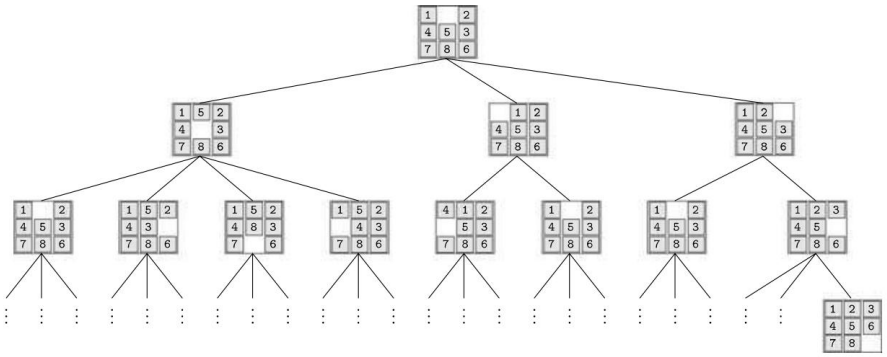


**Abb. 6.3 Suchbaum für das 8-Puzzle. *Unten rechts* ist ein Zielzustand in Tiefe 3 dargestellt. Um Platz zu sparen, wurden die anderen Knoten auf dieser Ebene weggelassen**

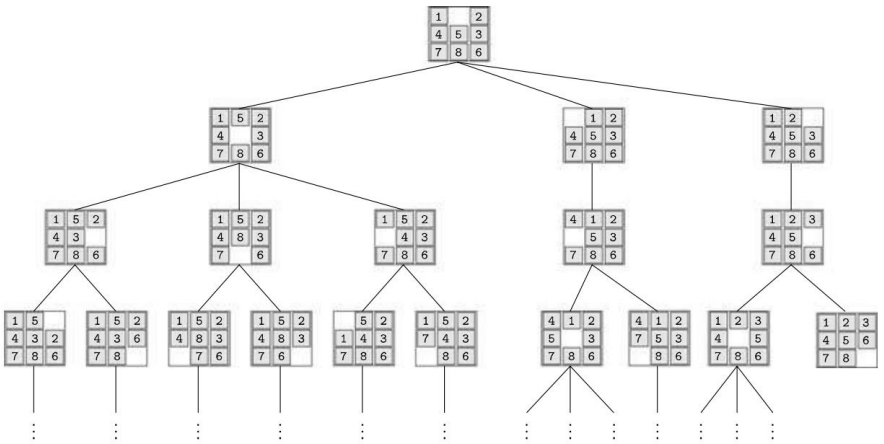


**Abb. 6.4 Suchbaum für ein 8-Puzzle ohne Zyklen der Länge 2**

**Bevor wir mit der Beschreibung der *Suchalgorithmen beginnen,* sind einige neue Begriffe erforderlich. Wir haben es hier mit diskreten Suchproblemen zu tun. Im Zustand s ist eine *Aktion* a1 = a1(s). einem neuen Zustand s . Also s Eine andere Aktion kann zu Zustand s führen , in führt zu andere Wörter: s = a2(s). Rekursive Anwendung aller möglichen Aktionen auf alle Zustände, Beginnend mit dem Startzustand ergibt sich der *Suchbaum.***

**Definition 6.1** Ein Suchproblem wird durch die folgenden Werte definiert : *Zustand:* Beschreibung des Zustands der Welt, in der der Suchagent findet selbst.

*Startzustand:* Der Anfangszustand, in dem der Suchagent gestartet wird.

*Zielzustand:* Erreicht der Agent einen Zielzustand, beendet er sich und gibt eine Lösung aus (falls gewünscht).

*Aktionen:* Alle Agenten haben Aktionen zugelassen.

*Lösung:* Der Pfad im Suchbaum vom Startzustand zum Zielzustand.

*Kostenfunktion:* Weist jeder Aktion einen Kostenwert zu. Notwendig, um eine kostenoptimale Lösung zu finden.

*Zustandsraum:* Menge aller Zustände.

Auf das 8er-Puzzle angewendet erhalten wir

*Zustand:* 3×3-Matrix S mit den Werten 1, 2, 3, 4, 5, 6, 7, 8 (je einmal) und einer leeren Quadrat.

*Ausgangszustand:* Ein beliebiger Zustand.

*Zielzustand:* Ein beliebiger Zustand, z. B. der rechts in Abb. 6.2 auf Seite 86 angegebene Zustand.

*Aktionen:* Bewegung des leeren Quadrats Sij nach links (wenn j = 1), nach rechts (wenn j = 3), nach oben (wenn i = 1), nach unten (wenn i = 3).

*Kostenfunktion:* Die konstante Funktion 1, da alle Aktionen gleiche Kosten haben.

*Zustandsraum:* Der Zustandsraum ist in Domänen degeneriert, die für beide Seiten nicht erreichbar sind (Übung 6.4 auf Seite 110). Somit gibt es unlösbare 8-Puzzle-Probleme.

Zur Analyse der Suchalgorithmen werden folgende Begriffe benötigt:

**Definition 6.2** • Die Anzahl der Nachfolgezustände eines Staates s wird als *Verzweigungsfaktor* bezeichnet b(s) oder b, wenn der Verzweigungsfaktor konstant ist.

• Der *effektive Verzweigungsfaktor* eines Baums der Tiefe d mit insgesamt n Knoten ist definiert als der Verzweigungsfaktor, den ein Baum mit konstantem Verzweigungsfaktor, gleicher Tiefe und gleichem n hätte (siehe Übung 6.3 auf Seite 110). • Ein Suchalgorithmus heißt *vollständig* , wenn er für jedes lösbare Problem eine Lösung findet. Wenn ein vollständiger Suchalgorithmus abbricht, ohne eine Lösung zu finden, ist das Problem unlösbar.

Für eine gegebene Tiefe d und Knotenanzahl n kann der effektive Verzweigungsfaktor sein berechnet durch Lösen der Gleichung

$$n = \frac{b^{d+1} - 1}{b - 1} \tag{6.1}$$

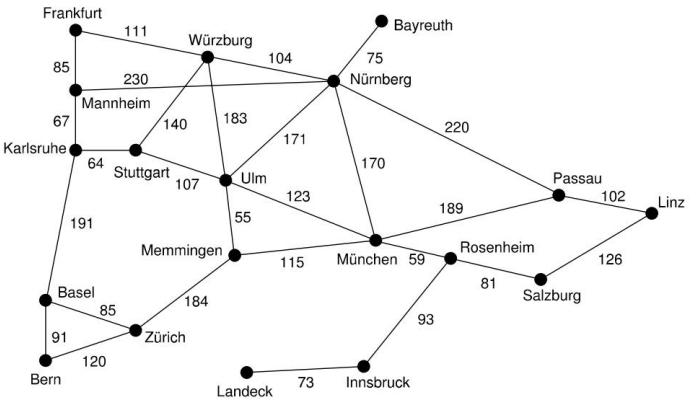



**Abb. 6.5 Der Graph von Süddeutschland als Beispiel für eine Suchaufgabe mit Kostenfunktion**

**für b, weil ein Baum mit konstantem Verzweigungsfaktor und Tiefe d insgesamt hat**

$$n = \sum_{i=0}^{D} b^i = \frac{b^{d+1} - 1}{b - 1} \tag{6.2}$$

**Knoten.**

**Für die praktische Anwendung von Suchalgorithmen für endliche Suchbäume das Letzte Ebene ist besonders wichtig, weil**

**Satz 6.1** ***Bei stark verzweigten endlichen Suchbäumen mit einem großen konstanten Verzweigungsfaktor liegen fast alle Knoten auf der letzten Ebene.***

**Der einfache Beweis dieses Theorems wird dem Leser als Übung empfohlen (Übung 6.1 auf Seite 110).**

***Beispiel 6.2*** **Wir erhalten eine Karte, wie sie in Abb. 6.5 dargestellt ist, als Diagramm mit Städten als Knoten und Autobahnverbindungen zwischen den Städten als gewichtete Kanten mit Entfernungen. Wir suchen nach einer optimalen Route von Stadt A nach Stadt B. Die Beschreibung des entsprechenden Schemas lautet**

***Bundesland:*** **Eine Stadt als aktueller Standort des Reisenden.**

***Ausgangszustand:*** **Eine beliebige Stadt.**

***Zielzustand:*** **Eine beliebige Stadt.**

***Aktionen:*** **Reisen Sie von der aktuellen Stadt in eine Nachbarstadt.**

***Kostenfunktion:*** **Die Entfernung zwischen den Städten. Jede Aktion entspricht einer Kante in der Grafik mit der Entfernung als Gewicht.**

***Zustandsraum:*** **Alle Städte, also Knoten des Graphen.**

**Um die Route mit minimaler Länge zu finden, müssen die Kosten berücksichtigt werden, da diese nicht wie beim 8-Rätsel konstant sind.**

**Definition 6.3 Ein Suchalgorithmus heißt *optimal*, wenn er, sofern eine Lösung existiert, immer die Lösung mit den niedrigsten Kosten findet.**

Das 8-Puzzle-Problem ist *deterministisch,* was bedeutet, dass jede Aktion von einem Zustand zu einem eindeutigen Nachfolgezustand führt. Darüber hinaus ist es *beobachtbar,* das heißt, der Agent weiß immer, in welchem Zustand er sich befindet. Bei der Routenplanung in realen Anwendungen sind beide Eigenschaften nicht immer gegeben. Die Aktion *„Fahrt von München nach Ulm“* kann – beispielsweise aufgrund eines Unfalls – zum Nachfolgeland *„München“ führen.* Es kann auch vorkommen, dass der Reisende nicht mehr weiß, wo er ist, weil er sich verlaufen hat. Solche Komplikationen wollen wir zunächst einmal ignorieren. Daher werden wir in diesem Kapitel nur Probleme betrachten, die deterministisch und beobachtbar sind.

Probleme wie das 8-Puzzle, die deterministisch und beobachtbar sind, machen die Aktionsplanung relativ einfach, da es aufgrund eines abstrakten Modells möglich ist, Aktionssequenzen zur Lösung des Problems zu finden, ohne die Aktionen tatsächlich in der realen Welt auszuführen . Beim 8er-Rätsel ist es nicht notwendig, die Felder in der realen Welt tatsächlich zu verschieben, um die Lösung zu finden. Mit sogenannten *Offline-Algorithmen können wir optimale Lösungen finden.* Ganz andere Herausforderungen stellt man beispielsweise beim Bau von Robotern, die Fußball spielen sollen. Dabei wird es nie ein exaktes abstraktes Modell der Handlungen geben. Beispielsweise kann ein Roboter, der den Ball in eine bestimmte Richtung kickt, nicht mit Sicherheit vorhersagen, wohin sich der Ball bewegen wird, weil er unter anderem nicht weiß, ob ein Gegner den Ball fängt oder abwehrt. Hier sind dann *Online-Algorithmen* gefragt, die in jeder Situation Entscheidungen auf Basis von Sensorsignalen treffen. *Reinforcement Learning,* beschrieben in Abschn. 10 arbeitet daran, diese Entscheidungen auf der Grundlage von Erfahrungen zu optimieren.

## 6.2 Uninformierte Suche

### 6.2.1 Breitensuche

Bei der Breitensuche wird der Suchbaum von oben nach unten nach dem in Abb. 6.6 auf Seite 90 angegebenen Algorithmus durchsucht , bis eine Lösung gefunden ist. Zunächst wird jeder Knoten in der Knotenliste darauf getestet, ob es sich um einen Zielknoten handelt, und im Erfolgsfall wird das Programm gestoppt. Ansonsten werden alle Nachfolger des Knotens generiert. Anschließend wird die Suche rekursiv auf der Liste aller neu generierten Knoten fortgesetzt. Das Ganze wiederholt sich, bis keine Nachfolger mehr generiert werden.

Dieser Algorithmus ist generisch. Das heißt, es funktioniert für beliebige Anwendungen, wenn die beiden anwendungsspezifischen Funktionen „GoalReached“ und „Successors“ bereitgestellt werde „GoalReached“ berechnet, ob das Argument ein Zielknoten ist, und „Successors“ berechnet die Liste aller Nachfolgerknoten seines Arguments. Abbildung 6.7 auf Seite 90 zeigt eine Momentaufnahme der Breitensuche.

```
BREADTH-FIRST-SEARCH(NodeList, Goal)
Neue Knoten = ÿ
Für alle Node ÿ NodeList
    If GoalReached(Knoten, Ziel)
        Return(„Lösung gefunden“, Knoten)
    NewNodes = Append(NewNodes, Successors(Node))
Wenn NewNodes = ÿ
    Return(BREADTH-FIRST-SEARCH(NewNodes, Goal))
Anders
    Return(“Keine Lösung”)
```

**Abb. 6.6 Der Algorithmus für die Breitensuche**

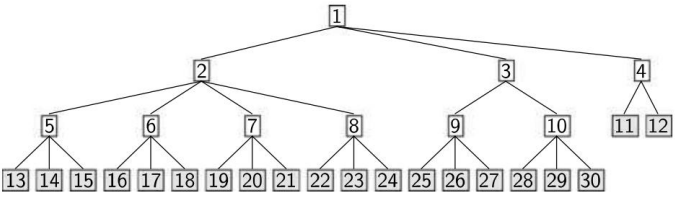



**Abb. 6.7 Breitensuche während der Erweiterung der Knoten der dritten Ebene. Die Knoten werden entsprechend der Reihenfolge ihrer Generierung nummeriert. Die Nachfolger der Knoten 11 und 12 wurden noch nicht generiert**

**Analyse Da die Breitensuche jede Tiefe vollständig durchsucht und jede Tiefe in endlicher Zeit erreicht, ist sie vollständig, wenn der Verzweigungsfaktor b endlich ist.**
**Die optimale (also die kürzeste) Lösung liegt dann vor, wenn die Kosten aller Aktionen gleich sind (siehe Aufgabe 6.7 auf Seite 110). Rechenzeit und Speicherplatz wachsen exponentiell mit der Tiefe des Baums. Für einen Baum mit konstantem Verzweigungsfaktor b und Tiefe d ist die Gesamtrechenzeit somit gegeben durch**

$$C \cdot \sum_{ich=0}^{D} b^{ich} = \frac{b^{d+1} ÿ 1}{b ÿ 1} = O(b^{d}).$$

**Obwohl nur die letzte Ebene im Speicher gespeichert wird, beträgt der Speicherplatzbedarf ebenfalls O(bd ).**

**Bei der Geschwindigkeit heutiger Computer, die innerhalb von Minuten Milliarden von Knoten generieren können, ist der Hauptspeicher schnell voll und die Suche endet. Das Problem, dass nicht immer die kürzeste Lösung gefunden wird, kann durch die sogenannte *Uniform Cost Search gelöst werden,* bei der der Knoten mit den niedrigsten Kosten aufsteigend sortiert wird**

```
DEPTH-FIRST-SEARCH (Knoten, Ziel)

If GoalReached(Node, Goal) Return("Lösung gefunden")
NewNodes = Nachfolger(Knoten)
Während NewNodes = ÿ
    Ergebnis = DEPTH-FIRST-SEARCH(First(NewNodes), Goal)
    Wenn Ergebnis = „Lösung gefunden“ Rückgabe („Lösung gefunden“)
    NewNodes = Rest(NewNodes)
Return("Keine Lösung")
```

**Abb. 6.8 Der Algorithmus für die Tiefensuche. Die Funktion „First“ gibt das erste Element einer Liste zurück und „Rest“ den Rest der Liste**

Die Liste der Knoten wird immer erweitert und die neuen Knoten einsortiert. So finden wir die optimale Lösung. Das Speicherproblem ist jedoch noch nicht gelöst. Eine Lösung für dieses Problem bietet die Tiefensuche.

## 6.2.2 Tiefensuche

Bei der Tiefensuche werden jeweils nur wenige Knoten gleichzeitig im Speicher gespeichert. Nach der Erweiterung eines Knotens werden nur dessen Nachfolger gespeichert und der erste Nachfolgerknoten wird sofort erweitert. Dadurch wird die Suche schnell sehr tiefgründig. Erst wenn ein Knoten keine Nachfolger hat und die Suche in dieser Tiefe fehlschlägt, wird der nächste offene Knoten durch *Backtracking* zum letzten Zweig usw. erweitert. Am besten erkennen wir dies am eleganten rekursiven Algorithmus in Abb. 6.8 und am Suchbaum in Abb. 6.9 auf Seite 92.

Analyse Die Tiefensuche erfordert viel weniger Speicher als die Breitensuche, da in jeder Tiefe höchstens b Knoten gespeichert werden. Wir benötigen also b · d Speicherzellen.

Allerdings ist die Tiefensuche für unendlich tiefe Bäume nicht vollständig, da die Tiefensuche in eine Endlosschleife gerät, wenn es im Zweig ganz links keine Lösung gibt. Daher ist die Frage nach der optimalen Lösung obsolet. Aufgrund der Endlosschleife kann keine Grenze für die Rechenzeit angegeben werden. Bei einem endlich tiefen Suchbaum mit der Tiefe d werden insgesamt etwa b Knoten erzeugt.
D
Somit wächst die Rechenzeit, genau wie bei der Breitensuche, exponentiell mit der Tiefe.

Wir können den Suchbaum endlich machen, indem wir eine Tiefenbegrenzung festlegen. Wenn nun im beschnittenen Suchbaum keine Lösung gefunden wird, kann es dennoch Lösungen außerhalb der Grenze geben. Dadurch wird die Suche unvollständig. Es gibt jedoch naheliegende Ideen, um die Suche zu vervollständigen.

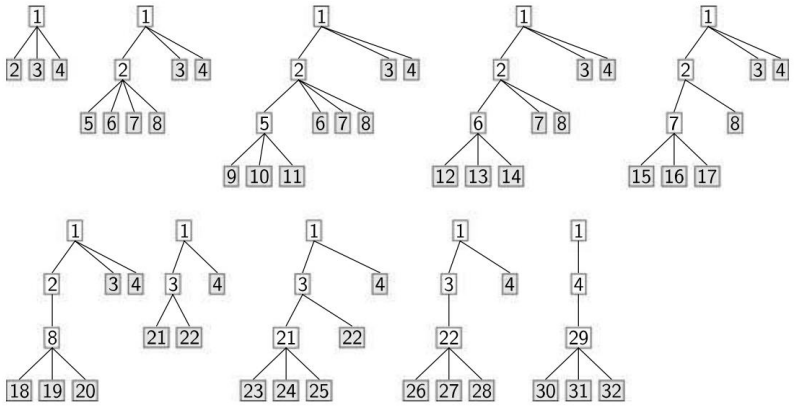



**Abb. 6.9 Durchführung der Tiefensuche. Alle Knoten in Tiefe drei sind nicht erfolgreich und führen zu einem Backtracking. Die Knoten werden in der Reihenfolge nummeriert, in der sie generiert wurden**

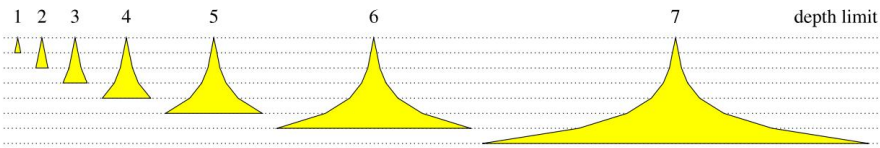



**Abb. 6.10 Schematische Darstellung der Entwicklung des Suchbaums bei der iterativen Vertiefung mit Grenzen von 1 bis 7. Die Breite des Baums entspricht einem Verzweigungsfaktor von 2**

## 6.2.3 Iterative Vertiefung

**Wir beginnen die Tiefensuche mit einem Tiefenlimit von 1. Wenn keine Lösung gefunden wird, erhöhen wir das Limit um 1 und beginnen die Suche von vorne und so weiter, wie in Abb. 6.10 dargestellt . Diese iterative Erhöhung der Tiefengrenze wird als *iterative Vertiefung bezeichnet.***

Wir müssen das in Abb. 6.8 **auf Seite 91 dargestellte Tiefensuchprogramm um die beiden zusätzlichen Parameter „Depth“ und „Limit“ erweitern. „Depth“ wird beim rekursiven Aufruf um eins erhöht und die Kopfzeile der while-Schleife wird durch „While NewNodes = ÿ And Depth < Limit“ ersetzt. Der modifizierte Algorithmus ist in Abb. 6.11 auf Seite 93 dargestellt.**

**Analyse Der Speicherbedarf ist derselbe wie bei der Tiefensuche. Man könnte argumentieren, dass das wiederholte Starten der Tiefensuche bei Tiefe Null eine Menge überflüssiger Arbeit verursacht. Bei großen Verzweigungsfaktoren ist dies nicht der Fall. Wir zeigen nun, dass die Summe der Anzahl der Knoten aller Tiefen bis zum vorletzten *dmax* ÿ 1 in allen durchsuchten Bäumen viel kleiner ist als die Anzahl der Knoten im zuletzt durchsuchte**

```
ITERATIVEDEEPENING(Knoten, Ziel)
DepthLimit = 0
Wiederholen
    Ergebnis = DEPTHFIRSTSEARCH-B(Knoten, Ziel, 0, DepthFirstSearch)
    DepthLimit = DepthLimit + 1
Bis Ergebnis = „Lösung gefunden“
```

```
DEPTHFIRSTSEARCH-B(Knoten, Ziel, Tiefe, Limit)
If GoalReached(Node, Goal) Return(“Lösung gefunden”)
NewNodes = Nachfolger(Knoten)
Während NewNodes = ÿ und Depth < Limit
    Ergebnis =
        DEPTHFIRSTSEARCH-B(First(NewNodes), Ziel, Tiefe + 1, Limit)
    Wenn Ergebnis = „Lösung gefunden“ Rückgabe („Lösung gefunden“)
    NewNodes = Rest(NewNodes)
Return(“Keine Lösung”)
```

**Abb. 6.11 Der Algorithmus zur iterativen Vertiefung, der die leicht modifizierte Tiefensuche mit Tiefenbegrenzung aufruft (TIEFENSUCHE-B)**

Sei Nb(d) die Anzahl der Knoten eines Suchbaums mit dem Verzweigungsfaktor b und Tiefe d und *dmax* sind die zuletzt gesuchte Tiefe. Der zuletzt durchsuchte Baum enthält

$$Nb(dmax) = \sum_{ich=0}^{D_{max}} ich\ b = \frac{b\ dmax+1\ \ddot{y}\ 1}{b\ \ddot{y}\ 1}$$

Knoten. Alle Bäume wurden vorher gemeinsam durchsucht

$$Nb(d) = \sum_{d=1}^{dmax\ \ddot{y}1\ dmax\ \ddot{y}1} b\ d+1\ \ddot{y}\ 1 = \frac{1}{b\ \ddot{y}\ 1} \sum_{d=1}^{dmax} \ddot{y}1\ d+1\ b \quad \ddot{y}\ dmax + 1$$

$$= \frac{1}{b\ \ddot{y}\ 1} \sum_{d=2}^{D_{max}} d\ b \quad \ddot{y}\ dmax + 1$$

$$= \frac{1}{b\ \ddot{y}\ 1} \left( \frac{b\ dmax+1\ \ddot{y}}{b\ \ddot{y}\ 1} 1\ \ddot{y}\ 1\ \ddot{y}\ b\ \ddot{y}\ dmax + 1 \right)$$

$$\ddot{y} \frac{1}{b\ \ddot{y}\ 1} \left( \frac{b\ dmax+1\ \ddot{y}\ 1}{b\ \ddot{y}\ 1} \right) = \frac{1}{b\ \ddot{y}\ 1} Nb(dmax)$$

Tabelle 6.1 Vergleich der uninformierten Suchalgorithmen. (*) bedeutet, dass die Aussage nur ist wahr bei konstanten Aktionskosten. ds ist die maximale Tiefe für einen endlichen Suchbaum

| | Breite zuerst suchen | Einheitliche Kosten suchen | Tiefenorientiert suchen | Iterativ Vertiefung |
|---|---|---|---|---|
| Vollständigkeit | Ja | Ja | NEIN | Ja |
| Optimale Lösung | Ja (*) | Ja | NEIN | Ja (*) |
| Rechenzeit | d b | d b | ÿ oder b ds | $B^D$ |
| Speichernutzung | $B^D$ | $B^D$ | bd | bd |

Knoten. Für b > 2 ist dies kleiner als die Anzahl *Nb(dmax)* der Knoten im letzten Baum. Für b = 20 enthalten die ersten *dmax* ÿ 1 Bäume zusammen nur etwa = 1/19 der Anzahl der Knoten im letzten Baum. Die Rechenzeit für alle Iterationen außer Letzte kann ignoriert werden.

Genau wie die Breitensuche ist diese Methode vollständig und mit konstanten Kosten verbunden Für alle Aktionen wird die kürzeste Lösung gefunden.

### 6.2.4 Vergleich

Die beschriebenen Suchalgorithmen sind in Tabelle 6.1 einander gegenübergestellt .

Wir können deutlich erkennen, dass die iterative Vertiefung der Gewinner dieses Tests ist, weil sie erhält in allen Kategorien die beste Note. Tatsächlich ist es von allen vier vorgestellten Algorithmen der nur praktisch verwendbar.

Wir haben tatsächlich einen Testsieger, wenn auch für realistische Anwendungen ist in der Regel nicht erfolgreich. Sogar für das 15er-Puzzle, den großen Bruder des 8er-Puzzles (siehe Aufgabe 6.4 auf Seite 110) gibt es etwa 2 × 1013 verschiedene Zustände. Für nicht trivial Bei Inferenzsystemen ist der Zustandsraum um viele Größenordnungen größer. Wie gezeigt in Sekte. 6.1, die ganze Rechenleistung der Welt wird nicht viel mehr helfen. Stattdessen Was benötigt wird, ist eine intelligente Suche, die nur einen winzigen Teil der Suche untersucht Raum und findet dort eine Lösung.

## 6.3 Heuristische Suche

*Heuristiken* sind Problemlösungsstrategien, die in vielen Fällen schneller zu einer Lösung führen als uninformierte Suche. Dies ist jedoch nicht garantiert. Heuristische Suche könnte erfordern viel mehr Zeit und können sogar dazu führen, dass die Lösung nicht gefunden wird.

Wir Menschen nutzen heuristische Verfahren erfolgreich für alles Mögliche. Wann Wenn wir zum Beispiel Gemüse im Supermarkt kaufen, beurteilen wir die verschiedenen Möglichkeiten für ein Pfund Erdbeeren anhand einiger einfacher Kriterien wie Preis, Aussehen, Produktionsquelle und Vertrauen in den Verkäufer, und dann entscheiden wir uns für die beste Option nach Bauchgefühl. Theoretisch könnte es besser sein, die Erdbeeren einer Grundierung zu unterziehen chemische Analyse, bevor Sie sich für den Kauf entscheiden. Beispielsweise könnten die Erdbeeren vergiftet sein. Wenn das der Fall wäre, hätte sich die Analyse gelohnt

```
HEURISTIKSEARCH(Start, Ziel)

NodeList = [Start]
Während wahr
    Wenn NodeList = ÿ Return("Keine Lösung")
    Knoten = First(NodeList)
    NodeList = Rest(NodeList)
    If GoalReached(Node, Goal) Return(„Lösung gefunden", Node)
    NodeList = SortIn(Successors(Node),NodeList)
```

**Abb. 6.12 Der Algorithmus für die heuristische Suche**

Problem. Eine solche Analyse führen wir jedoch nicht durch, da die Wahrscheinlichkeit sehr hoch ist, dass unsere heuristische Auswahl gelingt und wir schnell an unser Ziel, leckere Erdbeeren zu essen, gelangen.

Heuristische Entscheidungen sind eng mit der Notwendigkeit verbunden, *Entscheidungen in Echtzeit mit begrenzten Ressourcen zu treffen.* In der Praxis wird eine schnell gefundene gute Lösung einer Lösung vorgezogen, die optimal ist, deren Ableitung aber sehr teuer ist.

Zur mathematischen Modellierung einer Heuristik wird eine heuristische Bewertungsfunktion f(s) für Zustände verwendet. Ziel ist es, mit geringem Aufwand eine Lösung für das gestellte Suchproblem mit minimalen Gesamtkosten zu finden. Bitte beachten Sie, dass es einen subtilen Unterschied zwischen dem Aufwand, eine Lösung zu finden, und den Gesamtkosten dieser Lösung gibt. Beispielsweise kann es eine halbe Sekunde dauern, bis Google Maps eine Route vom Rathaus in San Francisco nach Tuolumne Meadows im Yosemite-Nationalpark findet, aber die Fahrt von San Francisco nach Tuolumne Meadows mit dem Auto kann vier Stunden und etwas Geld dauern für Benzin usw. (Gesamtkosten).

Als nächstes modifizieren wir den Breitensuchalgorithmus, indem wir ihm die Bewertungsfunktion hinzufügen. Die aktuell geöffneten Knoten werden nicht mehr zeilenweise von links nach rechts erweitert, sondern entsprechend ihrer heuristischen Bewertung. Aus der Menge der offenen Knoten wird immer zuerst der Knoten mit der minimalen Bewertung erweitert. Dies wird dadurch erreicht, dass Knoten beim Erweitern sofort ausgewertet und in die Liste der offenen Knoten ein
Die Liste kann dann Knoten aus unterschiedlichen Tiefen im Baum enthalten.

Da die heuristische Auswertung von Zuständen für die Suche sehr wichtig ist, unterscheiden wir im Folgenden zwischen Zuständen und den ihnen zugeordneten Knoten. Der Knoten con speichert den Zustand und weitere für die Suche relevante Informationen, wie z. B. seine Tiefe im Suchbaum und die heuristische Bewertung des Zustands. Dies hat zur Folge, dass die Funktion „Nachfolger", die die Nachfolger (Kinder) eines Knotens generiert, für diese Nachfolgerknoten sofort auch deren heuristische Bewertungen als Bestandteil jedes Knotens berechnen muss. Wir definieren den allgemeinen Suchalgorithmus HEURISTICSEARCH in Abb. 6.12.

Die Knotenliste wird mit den Startknoten initialisiert. Anschließend wird in der Schleife der erste Knoten aus der Liste entfernt und daraufhin geprüft, ob es sich um einen Lösungsknoten handelt. Wenn nicht, wird es um die Funktion „Nachfolger" erweitert und seine Nachfolger zur Liste hinzugefügt

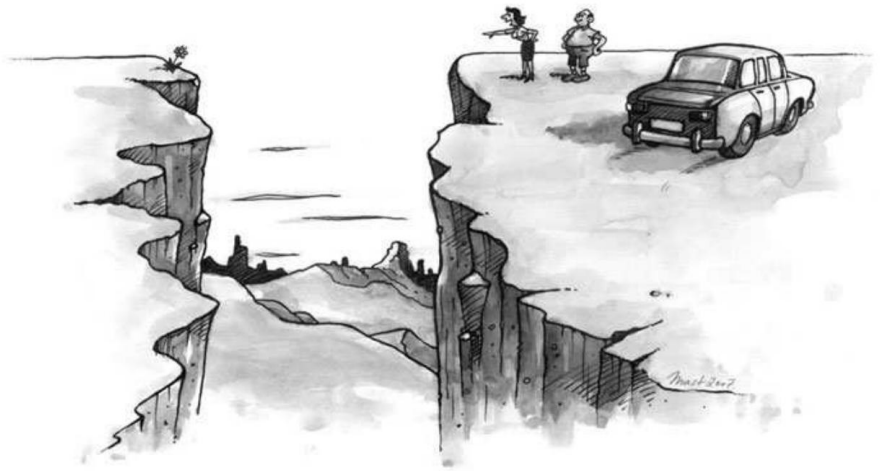


Abb. 6.13 Er: „Lieber, denken Sie an die Treibstoffkosten! Ich werde dir woanders eins pflücken.“ Sie: „Nein, das da drüben will ich haben!“

mit der Funktion „SortIn“. „SortIn(X,Y)“ fügt die Elemente aus der unsortierten Liste X in die aufsteigend sortierte Liste Y ein. Als Sortierschlüssel wird die heuristische Bewertung verwendet. Somit ist gewährleistet, dass der beste Knoten (also der mit dem niedrigsten heuristischen Wert) immer am Anfang der Liste steht.3 Auch die Tiefensuche und die Breitensuche sind Sonderfälle der Funktion HEURISTICSEARCH. Wir können sie einfach generieren, indem wir die entsprechende Auswertungsfunktion einbinden (Übung 6.11 auf Seite 111).

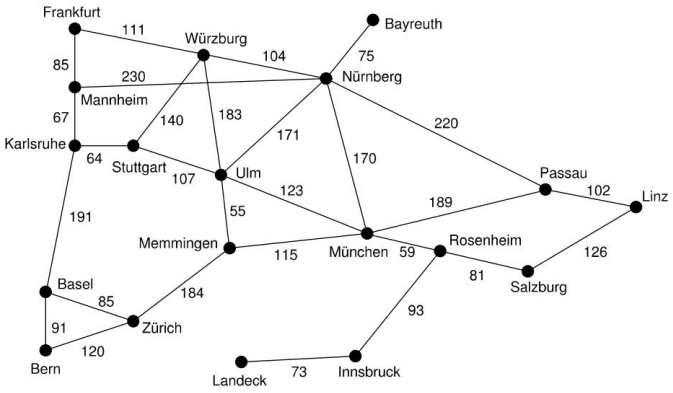
Die beste Heuristik wäre eine Funktion, die die tatsächlichen Kosten von jedem Knoten bis zum Ziel berechnet. Dazu wäre allerdings eine Durchquerung des gesamten Suchraums erforderlich, was die Heuristik genau verhindern soll. Daher benötigen wir eine Heuristik, die schnell und einfach zu berechnen ist. Wie finden wir eine solche Heuristik?

Eine interessante Idee zum Finden einer Heuristik ist die Vereinfachung des Problems. Die ursprüngliche Aufgabe ist so vereinfacht, dass sie mit geringem Rechenaufwand gelöst werden kann. Die Kosten von einem Zustand zum Ziel im vereinfachten Problem dienen dann als Schätzung für das eigentliche Problem (siehe Abb. 6.13). Diese *Kostenschätzfunktion* bezeichnen wir mit h.

## 6.3.1 Gierige Suche

Es erscheint sinnvoll, aus der Liste der aktuell verfügbaren Zustände den Zustand mit dem niedrigsten geschätzten h-Wert (also den mit den niedrigsten geschätzten Kosten) auszuwählen. D

3Beim Einsortieren eines neuen Knotens aus der Knotenliste kann es von Vorteil sein, zu prüfen, ob der Knoten bereits verfügbar ist, und gegebenenfalls das Duplikat zu löschen.




| | |
|---|---|
| Basel | 204 |
| Bayreuth | 207 |
| Bern | 247 |
| Frankfurt | 215 |
| Innsbruck | 163 |
| Karlsruhe | 137 |
| Landeck | 143 |
| Linz | 318 |
| München | 120 |
| Mannheim | 164 |
| Memmingen | 47 |
| Nürnberg | 132 |
| Passau | 257 |
| Rosenheim | 168 |
| Stuttgart | 75 |
| Salzburg | 236 |
| Würzburg | 153 |
| Zürich | 157 |

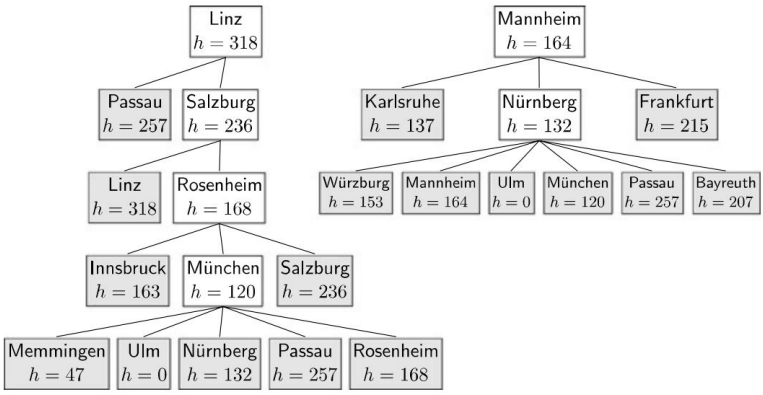
**Abb. 6.14 Stadtdiagramm mit Flugentfernungen von allen Städten nach Ulm**

**Die Kostenschätzung kann dann direkt als Bewertungsfunktion verwendet werden. Für die Auswertung in der Funktion HEURISTICSEARCH setzen wir f (s) = h(s). Dies ist am Beispiel der Reiseplanung (Beispiel 6.2 auf Seite 88) deutlich zu erkennen. Als Vereinfachung des Problems haben wir uns die Aufgabe gestellt, den geradlinigen Weg von Stadt zu Stadt (also die Flugstrecke) zu finden. Anstatt die optimale Route zu suchen, ermitteln wir zunächst von jedem Knoten aus eine Route mit minimaler Flugentfernung zum Ziel. Als Ziel wählen wir Ulm. Somit wird die Kostenschätzungsfunktion**

**h(s) = Flugentfernung von Stadt s nach Ulm.**

**Die Flugentfernungen von allen Städten nach Ulm sind in Abb. 6.14 neben der Grafik dargestellt.**

**Der Suchbaum für den Start in Linz ist in Abb. 6.15 auf Seite 98 links dargestellt. Wir können sehen, dass der Baum sehr schlank ist. Die Suche ist somit schnell beendet. Leider findet diese Suche nicht immer die optimale Lösung. Beispielsweise gelingt es diesem Algorithmus beim Start in Mannheim nicht, die optimale Lösung zu finden (Abb. 6.15 auf Seite 98 rechts). Der Weg Mannheim–Nürnberg–Ulm hat eine Länge von 401 km. Die Strecke Mannheim–Karlsruhe–Stuttgart–Ulm wäre mit 238 km deutlich kürzer. Wenn wir die Grafik betrachten, wird die Ursache dieses Problems klar. Nürnberg liegt zwar etwas näher an Ulm als Karlsruhe, allerdings ist die Entfernung von Mannheim nach Nürnberg deutlich größer als die von Mannheim nach Karlsruhe. Die Heuristik blickt nur „gierig“ auf das Ziel voraus, anstatt auch die bereits zurückgelegte Strecke bis zum aktuellen Knoten zu berücksichtigen. Deshalb geben wir ihr den Namen *Greedy Search.***




| Node | München | Innsbruck | Salzburg | Passau | Linz |
|---|---|---|---|---|---|
| heuristic rating | 120 | 163 | 236 | 257 | 318 |

**Abb. 6.15 Greedy-Suche: von Linz nach Ulm *(links)* und von Mannheim nach Ulm *(rechts).* Die Knotenlisten-Datenstruktur für den linken Suchbaum, sortiert nach der Knotenbewertung, bevor die Erweiterung des Knotens München angegeben wird**

### 6.3.2 Aÿ -Suche

**Wir wollen nun die Kosten berücksichtigen, die bei der Suche bis zum aktuellen Knoten s entstanden sind. Zuerst definieren wir die *Kostenfunktion***

**g(s) = Summe der angefallenen Kosten vom Start bis zum aktuellen Knoten,**

**Fügen Sie dann die geschätzten Kosten für das Ziel hinzu und erhalten Sie die *heuristische Bewertungsfunktion***

**f (s) = g(s) + h(s).**

**Nun fügen wir noch eine weitere kleine, aber wichtige Anforderung hinzu.**

**Definition 6.4 Eine heuristische Kostenschätzfunktion h(s), die die tatsächlichen Kosten vom Zustand s zum Ziel niemals überschätzt, wird als *zulässig bezeichnet.***

**Die Funktion HEURISTICSEARCH wird zusammen mit einer Bewertungsfunktion f (s) = g(s) + h(s) und einer zulässigen heuristischen Funktion h** Aÿ ***-Algorithmus genannt.*** **Dieser berühmte Algorithmus ist vollständig und optimal.** Aÿ **findet somit immer die kürzeste Lösung für jedes lösbare Suchproblem. Wir werden dies in der folgenden Diskussion erklären und beweisen.**

**Zuerst wenden wir den** Aÿ **-Algorithmus auf das Beispiel an. Wir suchen den Kürzesten Weg von Frankfurt nach Ulm.**

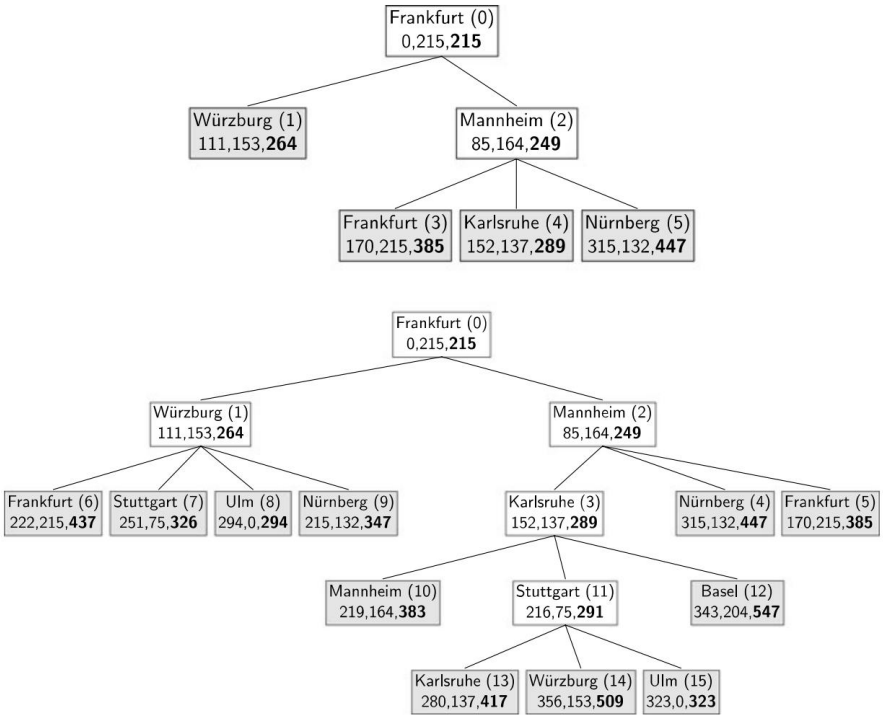



**Abb. 6.16 Zwei Schnappschüsse des** Aÿ- **Suchbaums für die optimale Route von Frankfurt nach Ulm. In den Feldern unter dem Namen der Stadt** s **zeigen wir** g(s), h(s), **f** (s). **Zahlen in Klammern hinter den Städtenamen zeigen die Reihenfolge, in der die Knoten durch die Funktion „Nachfolger“ generiert wur**

**Im oberen Teil von Abb. 6.16 sehen wir, dass die Nachfolger von Mannheim vor den Nachfolgern von Würzburg erzeugt werden. Die optimale Lösung Frankfurt–Würzburg–Ulm wird kurz darauf im achten Schritt generiert, aber noch nicht als solche erkannt. Somit terminiert der Algorithmus noch nicht, da der Knoten Karlsruhe (3) einen besseren (niedrigeren)** f- **Wert hat und somit in der Reihe vor dem Knoten Ulm (8) liegt. Erst wenn alle** f- **Werte größer oder gleich dem Lösungsknoten Ulm (8) sind, haben wir sichergestellt, dass wir eine optimale Lösung haben. Andernfalls könnte es möglicherweise eine andere Lösung mit geringeren Kosten geben. Wir werden nun zeigen, dass dies allgemein gilt.**

**Satz 6.2** ***Der*** Aÿ- ***Algorithmus ist optimal. Das heißt, es wird immer die Lösung mit den geringsten Gesamtkosten gefunden, wenn die Heuristik*** h ***zulässig ist.***

***Beweis*** **Im HEURISTICSEARCH -Algorithmus wird jeder neu generierte Knoten** s **durch die Funktion „SortIn“ entsprechend seiner heuristischen Bewertung f(s) einsortiert . Der Knoten mit dem kleinsten Bewertungswert steht somit am Anfang der Liste. Wenn der Knoten** l **am**

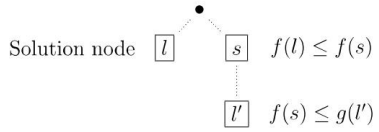



Abb. 6.17 Der erste von Aÿ gefundene Lösungsknoten l hat niemals höhere Kosten als ein anderer beliebiger Knoten l′

**Wenn am Anfang der Liste ein Lösungsknoten steht, hat kein anderer Knoten eine bessere heuristische Bewertung. Für alle anderen Knoten** s **gilt dann** f (l) **ÿ** f (s). **Da die Heuristik zulässig ist, kann auch nach Erweiterung aller anderen Knoten** keine bessere Lösung l gefunden werden (siehe Abb. **6.17). Formal geschrieben:**

$$g(l) = g(l) + h(l) = f(l) \ddot{y} f(s) = g(s) + h(s) \ddot{y} g(l\ddot{y}).$$

**Die erste Gleichheit gilt, weil** l **ein Lösungsknoten mit** h(l) **= 0 ist. Die zweite ist die Definition von** f . **Die dritte (Un-)Gleichheit gilt, weil die Liste der offenen Knoten in aufsteigender Reihenfolge sortiert ist. Die vierte Gleichung ist wiederum die Definition von** f . **Schließlich ist die letzte (Un-)Gleichheit die Zulässigkeit der Heuristik, die niemals die Kosten von Knoten** zu **einer beliebigen Lösung überschätzt. Somit wurde gezeigt, dass** g(l) **ÿ** g(lÿ ), **das heißt, dass die gefundene Lösung** l **optimal ist.**

### 6.3.3 IDAÿ -Suche

**Die** Aÿ- **Suche übernimmt eine Eigenart der Breitensuche. Es müssen viele Knoten im Speicher gespeichert werden, was zu einer sehr hohen Speichernutzung führen kann. Darüber hinaus muss die Liste der offenen Knoten sortiert werden. Das Einfügen von Knoten in die Liste und das Entfernen von Knoten aus der Liste kann daher nicht mehr in konstanter Zeit erfolgen, was die Komplexität des Algorithmus leicht erhöht. Basierend auf dem Heapsort-Algorithmus können wir die Knotenliste als Heap mit logarithmischer Zeitkomplexität für das Einfügen und Entfernen von Knoten strukturieren (siehe [CLR90]).**

**Beide Probleme können – ähnlich wie bei der Breitensuche – durch iterative Vertiefung gelöst werden. Wir arbeiten mit der Tiefensuche und heben das Limit sukzessive an. Anstatt jedoch mit einem Tiefenlimit zu arbeiten, verwenden wir hier ein Limit für die heuristische Auswertung** f (s). **Dieser Vorgang wird als IDAÿ -Algorithmus bezeichnet.**

### 6.3.4 Empirischer Vergleich der Suchalgorithmen

**In** Aÿ **, oder (alternativ) IDAÿ , Wir haben einen Suchalgorithmus mit vielen guten Eigenschaften. Es ist vollständig und optimal. Es kann somit ohne Risiko verwendet werden. Das Wichtigste ist jedoch, dass es mit Heuristiken arbeitet und so die Rechenzeit zur Lösungsfindung deutlich reduzieren kann. Wir möchten dies am Beispiel des 8-Puzzles empirisch untersuchen.**

**Für das 8-Puzzle gibt es zwei einfache zulässige Heuristiken. Die Heuristik h1 zählt einfach die Anzahl der Quadrate, die nicht an der richtigen Stelle sind. Diese Heuristik ist eindeutig zulässig. Die Heuristik h2 misst die *Manhattan-Entfernung.* Für jeden**

Tabelle 6.2 Vergleich des Rechenaufwands der uninformierten Suche und der heuristischen Suche nach lösbare 8-Puzzle-Aufgaben mit unterschiedlicher Tiefe. Die Messungen erfolgen in Schritten und Sekunden. Alle Werte sind Durchschnittswerte über mehrere Läufe (siehe letzte Spalte)

| Tiefe | Iterative Vertiefungsschritte | Zeit [Sek.] | Aÿ- Algorithmus Heuristik h1 Schritte | Zeit [Sek.] | Heuristik h2 Schritte | Zeit [Sek.] | Num. läuft |
|---|---|---|---|---|---|---|---|
| 2 | 20 | 0,003 | 3,0 | 0,0010 | 3,0 | 0,0010 | 10 |
| 4 | 81 | 0,013 | 5.2 | 0,0015 | 5,0 | 0,0022 | 24 |
| 6 | 806 | 0,13 | 10.2 | 0,0034 | 8.3 | 0,0039 | 19 |
| 8 | 6455 | 1,0 | 17.3 | 0,0060 | 12.2 | 0,0063 | 14 |
| 10 | 50512 | 7.9 | 48.1 | 0,018 | 22.1 | 0,011 | 15 |
| 12 | 486751 | 75,7 | 162.2 | 0,074 | 56,0 | 0,031 | 12 |
| | | | IDAÿ | | | | |
| 14 | – | – | 10079.2 | 2.6 | 855,6 | 0,25 | 16 |
| 16 | – | – | 69386.6 | 19.0 | 3806.5 | 1.3 | 13 |
| 18 | – | – | 708780,0 | 161,6 | 53941,5 | 14.1 | 4 |

Quadrieren Sie die horizontalen und vertikalen Abstände zur Position dieses Quadrats im Zielzustand werden addiert. Dieser Wert wird dann über alle Quadrate summiert. Zum Beispiel die Manhattan-Entfernung der beiden Staaten


wird berechnet als

$$h2(s) = 1 + 1 + 1 + 1 + 2 + 0 + 3 + 1 = 10.$$

Auch die Zulässigkeit der Manhattan-Distanz liegt auf der Hand (siehe Aufgabe 6.13 ) . Seite 111).

Die beschriebenen Algorithmen wurden in Mathematica implementiert. Zum Vergleich bei uninformierter Suche der Aÿ- Algorithmus mit den beiden Heuristiken h1 und h2 und Die iterative Vertiefung wurde auf 132 zufällig generierte 8-Puzzle-Aufgaben angewendet. Der Durchschnittswerte für die Anzahl der Schritte und die Rechenzeit sind in Tabelle 6.2 angegeben. Wir sehen, dass die Heuristiken die Suchkosten im Vergleich zur uninformierten Suche erheblich reduzieren.

Wenn wir beispielsweise die iterative Vertiefung mit Aÿ mit h1 in Tiefe 12 vergleichen , ist es Es wird deutlich, dass h1 die Anzahl der Schritte um etwa den Faktor 3.000 reduziert, aber die Rechenzeit nur um den Faktor 1.023. Dies liegt an den höheren Kosten pro Schritt zur Berechnung der Heuristik.

Bei näherer Betrachtung erkennt man einen Sprung in der Anzahl der Schritte zwischen Tiefe 12 und Tiefe 14 in der Spalte für h1. Dieser Sprung lässt sich nicht allein durch das Wiederholen erklären Arbeit von IDAÿ · Es entsteht durch die Implementierung des Aÿ- Algorithmus

löscht Duplikate identischer Knoten und verkleinert dadurch den Suchraum. Dies ist mit IDAÿ nicht möglich, da es fast keine Knoten speichert. Trotzdem kann Aÿ über die Tiefe 14 hinaus nicht mehr mit IDAÿ konkurrieren, da die Kosten für das Sortieren in neuen Knoten die Zeit pro Schritt so stark in die Höhe treiben.

Eine Berechnung des effektiven Verzweigungsfaktors nach (6.1) auf Seite 87 ergibt Werte von etwa 2,8 für die uninformierte Suche. Diese Zahl deckt sich mit dem Wert aus Abschn. 6.1. Die Heuristik h1 reduziert den Verzweigungsfaktor auf Werte von etwa 1,5 und h2 auf etwa 1,3. Wir können in der Tabelle sehen, dass eine kleine Reduzierung des Verzweigungsfaktors von 1,5 auf 1,3 uns einen großen Vorteil bei der Rechenzeit verschafft.

Die heuristische Suche hat daher eine wichtige praktische Bedeutung, da sie Probleme lösen kann, die für eine uninformierte Suche weit außerhalb der Reichweite liegen.

### 6.3.5 Zusammenfassung

Von den verschiedenen Suchalgorithmen für die uninformierte Suche ist die iterative Vertiefung der einzig praktikable, da sie vollständig ist und mit sehr wenig Speicher auskommt.
Bei schwierigen kombinatorischen Suchproblemen scheitert jedoch selbst eine iterative Vertiefung meist an der Größe des Suchraums. Hier hilft die heuristische Suche durch die Reduzierung des effektiven Verzweigungsfaktors. Der IDAÿ- Algorithmus ist ebenso wie die iterative Vertiefung vollständig und benötigt sehr wenig Speicher.

Heuristiken bringen natürlich nur dann einen signifikanten Vorteil, wenn die Heuristik „gut“ ist. Bei der Lösung schwieriger Suchprobleme besteht die eigentliche Aufgabe des Entwicklers darin, Heuristiken zu entwerfen, die den effektiven Verzweigungsfaktor stark reduzieren. In Abschn. In Abschn . 6.5 gehen wir auf dieses Problem ein und zeigen außerdem, wie Techniken des maschinellen Lernens zur automatischen Generierung von Heuristiken eingesetzt werden können.

Abschließend bleibt festzuhalten, dass Heuristiken bei unlösbaren Problemen keinen Leistungsvorteil haben, da die Unlösbarkeit eines Problems erst festgestellt werden kann, wenn der gesamte Suchbaum durchsucht wurde. Für entscheidbare Probleme wie das 8-Puzzle bedeutet dies, dass der gesamte Suchbaum bis zu einer maximalen Tiefe durchlaufen werden muss, unabhängig davon, ob eine Heuristik verwendet wird oder nicht. Die Heuristik ist in diesem Fall immer ein Nachteil, was auf den Rechenaufwand für die Auswertung der Heuristik zurückzuführen ist. Dieser Nachteil kann in der Regel durch einen konstanten Faktor abgeschätzt werden, der von der Größe des Problems unabhängig ist. Für unentscheidbare Probleme wie den Beweis von PL1-Formeln kann der Suchbaum unendlich tief sein. Das bedeutet, dass die Suche im unlösbaren Fall möglicherweise nie endet. Zusammenfassend lässt sich Folgendes sagen: Bei lösbaren Problemen reduzieren Heuristiken die Rechenzeit oft drastisch, bei unlösbaren Problemen kann der Aufwand bei Heuristiken jedoch sogar höher sein.

## 6.4 Spiele mit Gegnern

Spiele für zwei Spieler, wie Schach, Dame, Othello und Go, sind deterministisch, da jede Aktion (ein Zug) bei gleichem übergeordneten Zustand zum gleichen untergeordneten Zustand führt. Im Gegensatz dazu ist Backgammon nicht deterministisch, da sein untergeordneter Zustand davon abhängt

auf das Ergebnis eines Würfelwurfs. Diese Spiele sind alle beobachtbar, da jeder Spieler immer den kompletten Spielstand kennt. Viele Kartenspiele, wie zum Beispiel Poker, sind nur teilweise beobachtbar, weil der Spieler die Karten der anderen Spieler nicht oder nur teilweise kennt.

Die bisher in diesem Kapitel diskutierten Probleme waren deterministisch und beobachtbar. Im Folgenden betrachten wir Spiele, die ebenfalls deterministisch und beobachtbar sind. Darüber hinaus beschränken wir uns auf Nullsummenspiele. Dabei handelt es sich um Spiele, bei denen jeder Gewinn eines Spielers einen gleichwertigen Verlust für den Gegner bedeutet. Die Summe aus Gewinn und Verlust ist immer gleich Null. Dies gilt für die oben erwähnten Spiele Schach, Dame, Othello und Go.

### 6.4.1 Minimax-Suche

Das Ziel jedes Spielers ist es, optimale Züge zu machen, die zum Sieg führen. Im Prinzip ist es möglich, einen Suchbaum zu konstruieren und diesen (wie beim 8er-Puzzle) vollständig nach einer Reihe von Zügen zu durchsuchen, die zum Sieg führen. Es gibt jedoch einige Besonderheiten, auf die man achten sollte:

1. Der effektive Verzweigungsfaktor im Schach liegt bei etwa 30 bis 35. In einem typischen Spiel mit 50 Zügen pro Spieler hat der Suchbaum mehr als 30100 ÿ 10148 Blattknoten . Daher besteht keine Möglichkeit, den Suchbaum vollständig zu erkunden. Außerdem wird Schach oft mit Zeitlimit gespielt. Aufgrund dieser *Echtzeitanforderung* muss die Suche auf eine geeignete Tiefe im Baum beschränkt werden, beispielsweise auf acht Halbzüge. Da es unter den Blattknoten dieses tiefenbegrenzten Baums normalerweise keine Lösungsknoten (also Knoten, die das Spiel beenden) gibt, wird eine heuristische *Bewertungsfunktion* B für Brettpositionen verwendet. Der Spielgrad des Programms hängt stark von der Qualität dieser Bewertungsfunktion ab. Daher werden wir dieses Thema in Abschn. 6.5.
2. Im Folgenden nennen wir den Spieler, dessen Spiel wir optimieren wollen, Max und seinen Gegner Min. Die Bewegungen des Gegners (Min) sind nicht im Voraus bekannt, und daher ist auch der eigentliche Suchbaum nicht bekannt. Dieses Problem lässt sich elegant lösen, indem man annimmt, dass der Gegner immer den bestmöglichen Zug macht. Je höher die Bewertung B(s) für Position s ist, desto besser ist Position s für den Spieler Max und desto schlechter ist sie für seinen Gegner Min. Max versucht, die Bewertung seiner Züge zu maximieren, während Min Bewegungen ausführt, die zu einer möglichst niedrigen Bewertung führen.

Ein Suchbaum mit vier Halbbewegungen und Bewertungen aller Blätter ist in Abb. 6.18 auf Seite 104 dargestellt . Die Bewertung eines inneren Knotens wird rekursiv als Maximum oder Minimum seiner untergeordneten Knoten abgeleitet, abhängig von der Ebene des Knotens.

### 6.4.2 Alpha-Beta-Beschneidung

Durch den Wechsel zwischen Maximierung und Minimierung können wir uns unter Umständen viel Arbeit ersparen. Das Alpha-Beta-Pruning funktioniert mit der Tiefensuche bis zu einer voreingestellten Tiefengrenze. Auf diese Weise wird der Suchbaum von links durchsucht

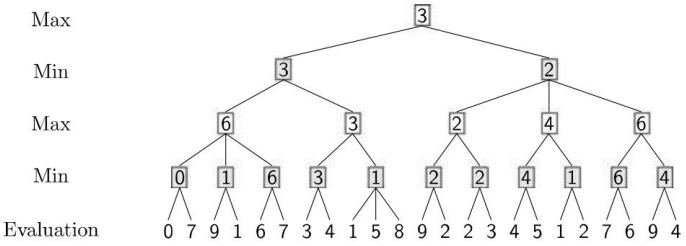



**Abb. 6.18 Ein Minimax-Spielbaum mit Vorausschau auf vier Halbzüge**

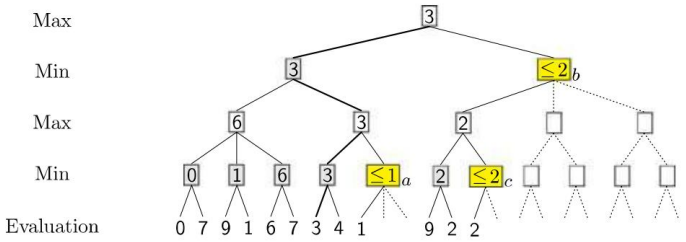



**Abb. 6.19 Ein Alpha-Beta-Spielbaum mit Vorausschau auf vier Halbzüge. Die gepunkteten Teile des Baums werden nicht durchlaufen, da sie keinen Einfluss auf das Endergebnis haben**

**nach rechts. Wie bei der Minimax-Suche wird bei den Minimum-Knoten das Minimum aus dem Minimalwert der Nachfolgerknoten generiert, bei den Maximum-Knoten ebenfalls das Maximum. In Abb. 6.19 ist dieser Vorgang für den Baum aus Abb. 6.18 dargestellt. An dem mit a gekennzeichneten Knoten können alle anderen Nachfolger ignoriert werden, nachdem das erste Kind als Wert 1 ausgewertet wurde, da das Minimum mit Sicherheit ÿ1 ist. Es könnte sogar noch kleiner werden, aber das spielt keine Rolle, da das Maximum eine Ebene darüber bereits ÿ3 liegt. Unabhängig davon, wie die Bewertung der verbleibenden Nachfolger ausfällt, behält das Maximum den Wert 3. Analog wird der Baum am Knoten b beschnitten. Da das erste Kind von b den Wert 2 hat, kann das zu erzeugende Minimum für b nur kleiner oder gleich 2 sein. Das Maximum am Wurzelknoten ist aber schon sicher ÿ3.**
**Dies kann durch Werte ÿ2 nicht geändert werden. Somit können die verbleibenden Teilbäume von b beschnitten werden.**

**Die gleiche Argumentation gilt für den Knoten c. Allerdings ist der relevante maximale Knoten ist nicht der direkte Elternknoten, sondern der Wurzelknoten. Dies lässt sich verallgemeinern. • An jedem Blattknoten wird die Bewertung berechnet. • Für jeden Maximumknoten wird der aktuell größte Kindwert in ÿ gespeichert. • Für jeden minimalen Knoten wird der aktuell kleinste Kindwert in ÿ gespeichert. • Wenn an einem minimalen Knoten k der aktuelle Wert ÿ ÿ ÿ ist, kann die Suche unter k beendet werden.**

**Dabei ist ÿ der größte Wert eines maximalen Knotens im Pfad von der Wurzel nach k. • Wenn an einem maximalen Knoten l der aktuelle Wert ÿ ÿ ÿ ist, kann die Suche unter l beendet werden.**

**Dabei ist ÿ der kleinste Wert eines minimalen Knotens im Pfad von der Wurzel nach l.**

```
ALPHABETAMAX(Knoten, ÿ, ÿ)

If DepthLimitReached(Node) Return(Rating(Node))
NewNodes = Nachfolger(Knoten)
Während NewNodes = ÿ
    ÿ = Maximum(ÿ, ALPHABETAMIN(First(NewNodes), ÿ, ÿ))
    Wenn ÿ ÿ ÿ Return(ÿ)
    NewNodes = Rest(NewNodes)
Rückgabe(ÿ)
```

```
ALPHABETAMIN(Knoten, ÿ, ÿ)

If DepthLimitReached(Node) Return(Rating(Node))
NewNodes = Nachfolger(Knoten)
Während NewNodes = ÿ
    ÿ = Minimum(ÿ, ALPHABETAMAX(First(NewNodes), ÿ, ÿ))
    Wenn ÿ ÿ ÿ Return(ÿ)
    NewNodes = Rest(NewNodes)
Rückkehr(ÿ)
```

**Abb. 6.20 Der Algorithmus zur Alpha-Beta-Suche mit den beiden Funktionen ALPHABETAMIN und AL PHABETAMAX**

**Der in Abb. 6.20** dargestellte Algorithmus **ist eine Erweiterung der Tiefensuche mit zwei Funktionen, die abwechselnd aufgerufen werden. Es verwendet die oben definierten Werte für ÿ und ÿ.**

**Komplexität: Die durch Alpha-Beta-Pruning eingesparte Rechenzeit hängt stark von der Reihenfolge ab, in der untergeordnete Knoten durchlaufen werden. Im schlimmsten Fall bringt der Alpha-Beta-Schnitt keinen Vorteil. Für einen konstanten Verzweigungsfaktor b ist die Anzahl nd der in der Tiefe d auszuwertenden Blattknoten gleich**

$$nd = b^{D}.$$

**Im besten Fall, wenn die Nachfolger maximaler Knoten absteigend und die Nachfolger minimaler Knoten aufsteigend sortiert sind, reduziert sich der effektive Verzweigungsfaktor auf ÿ b. Im Schach bedeutet dies eine erhebliche Reduzierung des effektiven Verzweigungsfaktors von 35 auf etwa 6. Dann nur**

$$nd = ÿ\, b^{—D} = b\, d/2$$

Blattknoten würden erstellt. Das bedeutet, dass die Tiefenbegrenzung und damit auch der Suchhorizont beim Alpha-Beta-Pruning verdoppelt werden. Dies trifft jedoch nur bei optimal sortierten Nachfolgern zu, da die Bewertungen der untergeordneten Knoten zum Zeitpunkt ihrer Erstellung unbekannt sind. Wenn die untergeordneten Knoten zufällig sortiert sind, wird der Verzweigungsfaktor auf b 3/4 und die Anzahl der Blattknoten auf b reduziert

$$nd = b^{\frac{3}{4}d}.$$

Mit der gleichen Rechenleistung kann ein Schachcomputer mit Alpha-Beta-Pruning beispielsweise acht statt sechs Halbzüge voraus berechnen, mit einem effektiven Verzweigungsfaktor von etwa 14. Eine ausführliche Analyse mit einer Ableitung dieser Parameter finden Sie in [Pea84].

Um die Suchtiefe wie oben erwähnt zu verdoppeln, müssten die untergeordneten Knoten optimal angeordnet sein, was in der Praxis nicht der Fall ist. Sonst wäre die Suche unnötig. Mit einem einfachen Trick können wir eine relativ gute Knotenreihenfolge erreichen. Wir verbinden Alpha-Beta-Pruning mit iterativer Vertiefung über die Tiefengrenze hinaus. Somit können wir bei jeder neuen Tiefengrenze auf die Bewertungen aller Knoten vorheriger Ebenen zugreifen und die Nachfolger bei jedem Zweig ordnen. Dadurch erreichen wir einen effektiven Verzweigungsfaktor von etwa 7 bis 8, was nicht weit vom theoretischen O

### 6.4.3 Nichtdeterministische Spiele

Die Minimax-Suche kann auf alle Spiele mit nicht deterministischen Aktionen, wie z. B. Backgammon, verallgemeinert werden. Jeder Spieler würfelt vor seinem Zug, der durch das Ergebnis des Würfelwurfs beeinflusst wird. Im Spielbaum gibt es nun in der Reihenfolge drei Arten von Levels

Max, Würfel, Min, Würfel, . . . ,

wobei jeder Würfelwurfknoten sechs Wege verzweigt. Da wir den Wert des Würfels nicht vorhersagen können, mitteln wir die Werte aller Würfe und führen die Suche wie beschrieben mit den Durchschnittswerten aus [RN10] durch.

## 6.5 Heuristische Auswertungsfunktionen

Wie finden wir eine gute heuristische Bewertungsfunktion für die Suchaufgabe? Hier gibt es grundsätzlich zwei Ansätze. Der klassische Weg nutzt das Wissen menschlicher Experten. Dem Wissensingenieur wird die meist schwierige Aufgabe übertragen, das implizite Wissen des Experten in Form eines Computerprogramms zu formalisieren. Wie dieser Vorgang vereinfacht werden kann, wollen wir nun am Beispiel eines Schachprogramms zeigen.

Im ersten Schritt werden Experten zu den wichtigsten Faktoren bei der Umzugsauswahl befragt. Anschließend wird versucht, diese Faktoren zu quantifizieren. Wir erhalten eine Liste relevanter Merkmale oder Attribute. Diese werden dann (im einfachsten Fall) zusammengefasst

eine lineare Bewertungsfunktion B(s) für Positionen, die wie folgt aussehen könnte:

$$B(s) = a1 \cdot \text{Material} + a2 \cdot \text{Bauernstruktur} + a3 \cdot \text{Königssicherheit} + a4 \cdot \text{knight\_in\_center} + a5 \cdot \text{Bishop\_diagonal\_coverage} + \cdots, \quad (6.3)$$

wobei „Material“ das mit Abstand wichtigste Merkmal ist und nach berechnet wird

$$\text{material} = \text{material}(\text{own\_team}) \ddot{y} \text{material}(\text{gegner})$$

mit

$$\text{material}(\text{team}) = \text{num\_pawns}(\text{team}) \cdot 100 + \text{num\_knights}(\text{team}) \cdot 300 + \text{num\_bishops}(\text{team}) \cdot 300 + \text{num\_rooks}(\text{team}) \cdot 500 + \text{num\_queens}(\text{team}) \cdot 900$$

Fast alle Schachprogramme nehmen eine ähnliche Materialbewertung vor. Allerdings gibt es bei allen anderen Features große Unterschiede, auf die wir hier nicht näher eingehen [Fra05, Lar00].

Im nächsten Schritt müssen die Gewichte ai aller Merkmale bestimmt werden. Diese werden nach Diskussion mit Experten intuitiv festgelegt und dann nach jedem Spiel aufgrund positiver und negativer Erfahrungen geändert. Die Tatsache, dass dieser Optimierungsprozess sehr aufwändig ist und darüber hinaus die lineare Kombination von Merkmalen sehr begrenzt ist, legt den Einsatz von maschinellem Lernen nahe.

## 6.5.1 Erlernen von Heuristiken

Wir wollen nun die Gewichte ai der Bewertungsfunktion B(s) aus (6.3) automatisch optimieren . Bei diesem Ansatz wird der Experte lediglich nach den relevanten Merkmalen f1(s), ... befragt. . . fn(s) für Spielstatus s. Anschließend kommt ein maschinelles Lernverfahren zum Einsatz mit dem Ziel, eine möglichst optimale Bewertungsfunktion zu finden. Wir beginnen mit einer zunächst voreingestellten Bewertungsfunktion (bestimmt durch den Lernprozess) und lassen dann das Schachprogramm spielen. Am Ende des Spiels wird aus dem Ergebnis (Sieg, Niederlage oder Unentschieden) eine Wertung abgeleitet. Basierend auf dieser Bewertung wird die Bewertungsfunktion geändert mit dem Ziel, beim nächsten Mal weniger Fehler zu machen. Im Prinzip wird nun das Gleiche, was der Entwickler tut, automatisch durch den Lernprozess erledigt.

So einfach das klingt, so schwierig ist es in der Praxis. Ein zentrales Problem bei der Verbesserung der Positionsbewertung anhand gewonnener oder verlorener Spiele ist heute als *Credit-Assignment-* Problem bekannt. Wir haben zwar eine Wertung am Ende der Partie, aber keine Wertung für die einzelnen Spielzüge. Der Agent führt also viele Aktionen aus, erhält aber bis zum Schluss weder positives noch negatives Feedback. Wie soll es dieses Feedback dann den vielen in der Vergangenheit ergriffenen Maßnahmen zuordnen? Und wie sollte es in diesem Fall sein Handeln verbessern? Mit solchen Fragen beschäftigt sich das spannende junge Feld des *Reinforcement Learning* (siehe Abschn. 10).

Die meisten der besten Schachcomputer der Welt funktionieren heute noch ohne Techniken des maschinellen Lernens. Dafür gibt es zwei Gründe. Einerseits benötigen die bisher entwickelten Reinforcement-Learning-Algorithmen bei großen Zustandsräumen sehr viel Rechenzeit. Andererseits sind die manuell erstellten Heuristiken von Hochleistungsschachcomputern bereits stark optimiert. Das bedeutet, dass nur ein sehr gutes Lernsystem zu Verbesserungen führen kann. In den nächsten zehn Jahren wird voraussichtlich die Zeit kommen, in der ein lernender Computer Weltmeister wird.

## 6.6 Stand der Technik

Zur Bewertung der Qualität der heuristischen Suchprozesse möchte ich noch einmal wiederholen Elaine Richs Definition [Ric83]:

> Künstliche Intelligenz ist die Untersuchung, wie man Computer dazu bringt, Dinge zu tun, in denen Menschen derzeit besser sind.

Es gibt kaum einen besser geeigneten Test zur Entscheidung, ob ein Computerprogramm intelligent ist, als den direkten Vergleich von Computer und Mensch in einem Spiel wie Schach, Dame, Backgammon oder Go.

1950 stellten Claude Shannon, Konrad Zuse und John von Neumann die ersten Schachprogramme vor, deren Umsetzung jedoch entweder nicht oder nur mit sehr viel Zeitaufwand umsetzbar war. Nur wenige Jahre später, im Jahr 1955, schrieb Arthur Samuel ein Programm, das Dame spielte und durch einen einfachen Lernprozess seine eigenen Parameter verbessern konnte. Dazu nutzte er den ersten programmierbaren Logikrechner, den IBM 701. Im Vergleich zu heutigen Schachcomputern verfügte er jedoch über Zugriff auf eine Vielzahl archivierter Partien, bei denen jeder einzelne Zug von Experten bewertet worden war. Dadurch verbesserte das Programm seine Auswertungsfunktion. Um weitere Verbesserungen zu erzielen, ließ Samuel sein Programm gegen sich selbst spielen. Er hat das Problem der Kreditvergabe auf einfache Weise gelöst. Für jede einzelne Position während eines Spiels vergleicht es die Bewertung durch die Funktion B(s) mit der durch Alpha-Beta-Pruning berechneten und ändert B(s) entsprechend. 1961 schlug sein Dameprogramm den viertbesten Damespieler der USA. Mit dieser bahnbrechenden Arbeit war Samuel seiner Zeit sicherlich fast 30 Jahre voraus.

Erst Anfang der neunziger Jahre, als Reinforcement Learning aufkam, baute Gerald Tersauro ein Lern-Backgammonprogramm namens TD-Gammon auf, das auf Weltmeisterniveau spielte (siehe Abschn. 10).

Heutzutage gibt es mehrere Schachprogramme, die auf Großmeisterniveau spielen, und einige werden kommerziell für den PC verkauft. Der Durchbruch gelang 1997, als IBMs Deep Blue den Schachweltmeister Gary Kasparov mit einer Punktzahl von 3,5 zu 2,5 besiegte. Deep Blue konnte mit Alpha-Beta-Bereinigung und heuristischer Stellungsauswertung durchschnittlich 12 Halbzüge voraus berechnen.

Einer der bislang leistungsstärksten Schachcomputer ist Hydra, ein Parallelcomputer einer Firma in den Vereinigten Arabischen Emiraten. Die Software wurde von den Wissenschaftlern Christian Donninger (Österreich) und Ulf Lorenz (Deutschland) sowie dem deutschen Schachgroßmeister Christopher Lutz entwickelt. Hydra verwendet 64 parallele Xeon

Prozessoren mit ca. 3 GHz Rechenleistung und jeweils 1 GByte Speicher. Für die Positionsauswertung verfügt jeder Prozessor über einen FPGA-Coprozessor (Field Programmable Gate Array). Dadurch wird es möglich, auch mit einer teuren Auswertefunktion 200 Millionen Positionen pro Sekunde auszuwerten.

Mit dieser Technologie kann Hydra im Durchschnitt etwa 18 Züge im Voraus berechnen. In besonderen, kritischen Situationen kann der Suchhorizont sogar auf 40 Halbzüge ausgedehnt werden. Offensichtlich übersteigt diese Art von Horizont selbst große Champions, denn Hydra macht oft Bewegungen, die große Champions nicht verstehen können, die aber am Ende zum Sieg führen. Im Jahr 2005 besiegte Hydra den siebtplatzierten Großmeister Michael Adams mit 5,5 zu 0,5 Spielen.

Hydra verwendet wenig spezielles Lehrbuchwissen über Schach, eher eine Alpha-Beta-Suche mit relativ allgemeinen, bekannten Heuristiken und einer guten handcodierten Stellungsbewertung. Insbesondere ist Hydra nicht lernfähig. Zwischen den Spielen werden von den Entwicklern Verbesserungen vorgenommen. Somit wird der Computer weiterhin vom Lernen entlastet. Hydra verfügt auch über keine speziellen Planungsalgorithmen. Die Tatsache, dass Hydra ohne Lernen funktioniert, ist ein Hinweis darauf, dass trotz vieler Fortschritte noch Forschungsbedarf im Bereich maschinelles Lernen besteht. Darauf gehen wir, wie oben erwähnt, im Abschn. 10 und Kap. 8.

Im Jahr 2009 gewann das auf einem PDA laufende System Pocket Fritz 4 das Copa Mercosur-Schachturnier in Buenos Aires mit neun Siegen und einem Unentschieden gegen zehn hervorragende menschliche Schachspieler, darunter drei Großmeister. Auch wenn nicht viele Informationen über den internen Aufbau der Software verfügbar sind, stellt diese Schachmaschine einen Trend weg von reiner Rechenleistung hin zu mehr Intelligenz dar. Diese Maschine spielt auf Großmeisterniveau und ist mit Hydra vergleichbar, wenn nicht Laut Pocket Fritz-Entwickler Stanislav Tsukrov [Wik10] durchsucht Pocket Fritz mit seiner Schachsuchmaschine HIARCS 13 weniger als 20.000 Positionen pro Sekunde und ist damit etwa um den Faktor 10.000 langsamer als Hydra. Dies führt zu dem Schluss, dass HIARCS 13 definitiv bessere Heuristiken zur Verringerung des effektiven Verzweigungsfaktors verwendet als Hydra und daher durchaus als intelligenter als Hydra bezeichnet werden kann. HIARCS ist übrigens eine Abkürzung für *Higher Intelligence Auto Response C*

Auch wenn bald kein Mensch mehr eine Chance gegen die besten Schachcomputer haben wird, gibt es noch viele Herausforderungen für die KI. Gehen Sie zum Beispiel. Bei diesem alten japanischen Spiel, das auf einem quadratischen Brett mit 361 Feldern, 181 weißen und 180 schwarzen Steinen gespielt wird, beträgt der effektive Verzweigungsfaktor etwa 300. Nach vier Halbzügen sind es bereits etwa 8×109 Stellungen . Alle bekannten klassischen Spielbaum-Suchverfahren haben auf diesem Komplexitätsniveau keine Chance gegen gute menschliche Go-Spieler. Die Experten sind sich einig, dass es hier „wirklich intelligente“ Systeme braucht. Die kombinatorische Aufzählung aller Möglichkeiten ist die falsche Methode. Vielmehr werden Prozesse benötigt, die Muster auf dem Board erkennen, langfristige Entwicklungen verfolgen und schnell „intuitive“ Entscheidungen treffen können. Ähnlich wie bei der Erkennung von Objekten in komplexen Bildern ist der Mensch Computerprogrammen immer noch meilenweit voraus. Wir verarbeiten das Bild als Ganzes hochgradig parallel, während der Computer die Millionen Pixel nacheinander verarbeitet und es schwer hat, in der Fülle der Pixel das Wesentliche zu erkennen. Das Programm „The Many Faces of Go“ kann 1.100 verschiedene Muster erkennen und kennt 200 Spielstrategien. Aber alle Go-Programme haben immer no

## 6.7 Übungen

**Aufgabe 6.1**

(a) Beweisen Sie Satz 6.1 auf Seite 88, mit anderen Worten, beweisen Sie, dass sich für einen Baum mit großem konstanten Verzweigungsfaktor b fast alle Knoten auf der letzten Ebene in der Tiefe d befinden. (b) Zeigen Sie, dass dies nicht immer zutrifft, wenn der effektive Verzweigungsfaktor groß und nicht konstant ist.

**Aufgabe 6.2**

(a) Berechnen Sie den durchschnittlichen Verzweigungsfaktor für das 8-Puzzle ohne Prüfung auf Zyklen. Der durchschnittliche Verzweigungsfaktor ist der Verzweigungsfaktor, den ein Baum mit einer gleichen Anzahl von Knoten auf der letzten Ebene, einem konstanten Verzweigungsfaktor und gleicher Tiefe hätte. (b) Berechnen Sie den durchschnittlichen Verzweigungsfaktor für das 8-Rätsel für die uninformierte Suche unter Vermeidung von Zyklen der Länge 2.

**Aufgabe 6.3**

(a) Was ist der Unterschied zwischen dem Durchschnitt und dem effektiven Verzweigungsfaktor? (Definition 6.2 auf Seite 87)?
(b) Warum eignet sich der effektive Verzweigungsfaktor besser zur Analyse und zum Vergleich der Rechenzeit von Suchalgorithmen als der durchschnittliche Verzweigungsfaktor? (c) Zeigen Sie, dass für einen stark verzweigten Baum mit n Knoten und der Tiefe d der effektive Verzweigungsfaktor $b^-$ ungefähr gleich dem durchschnittlichen Verzweigungsfaktor und damit gleich $\sqrt[d]{n}$ ist.

**Aufgabe 6.4**

(a) Berechnen Sie die Größe des Zustandsraums für das 8-Puzzle, für das analoge 3-Puzzle (2 × 2-Matrix) sowie für das 15-Puzzle (4 × 4-Matrix). (b) Beweisen Sie, dass der Zustandsgraph, der aus den Zuständen (Knoten) und den Aktionen (Kanten) für das 3-Puzzle besteht, in zwei verbundene Teilgraphen zerfällt, zwischen denen es keine Verbindungen gibt.

**Übung 6.5** Finden Sie mit der Breitensuche für das 8-Rätsel einen Pfad (manuell) von der Startknoten  zum Zielknoten  .

**ÿ Übung 6.6**

(a) Programmieren Sie die Breitensuche, die Tiefensuche und die iterative Vertiefung in der Sprache Ihrer Wahl und testen Sie sie am Beispiel mit 8 Rätseln. (b) Warum macht es wenig Sinn, beim 8-Puzzle die Tiefensuche zu verwenden?

**Aufgabe 6.7**

(a) Zeigen Sie, dass die Breitensuche bei konstanten Kosten für alle Aktionen garantiert ist um die kürzeste Lösung zu finden.

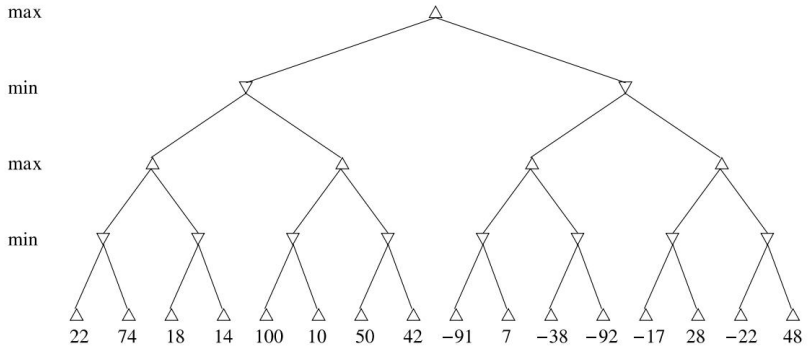



**Abb. 6.21 Minimax-Suchbaum**

**(b) Zeigen Sie, dass dies für unterschiedliche Kosten nicht der Fall ist.**

**Übung 6.8 Suchen Sie mithilfe** der Aÿ- **Suche für das 8-Rätsel (manuell) nach einem Pfad von der Startknoten**

| | | |
|---|---|---|
| 1 | | 3 |
| 4 | 2 | 6 |
| 7 | 5 | 8 |

**zum Zielknoten**

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

**(a) unter Verwendung der Heuristik h1 (Abschn. 6.3.4). (b) unter Verwendung der Heuristik h2 (Abschn. 6.3.4).**

**Aufgabe 6.9 Konstruieren Sie den** Aÿ- **Suchbaum für den Stadtgraphen aus Abb. 6.14 auf Seite 97 und verwenden Sie die Flugentfernung nach Ulm als Heuristik. Starten Sie in Bern mit Ulm als Ziel. Achten Sie darauf, dass jede Stadt pro Pfad nur einmal vorkommt.**

**ÿ Übung 6.10 Programm** Aÿ **Suche in der Programmiersprache Ihrer Wahl**
**Verwenden Sie die Heuristiken h1 und h2 und testen Sie diese am 8-Puzzle-Beispiel.**

**ÿ Aufgabe 6.11 Geben Sie eine heuristische Bewertungsfunktion für Zustände an, mit denen HEURIS TICSEARCH kann als Tiefensuche und als Implementierung einer Breitensuche implementiert werden.**

**Aufgabe 6.12 Welcher Zusammenhang besteht zwischen dem Bild des Paares in der Schlucht aus Abb. 6.13 auf Seite 96 und zulässigen Heuristiken?**

**Aufgabe 6.13 Zeigen Sie, dass die Heuristiken h1 und h2 für das 8-Puzzle aus Abschn. 6.3.4 sind zulässig.**

**Aufgabe 6.14 (a)**

**Der Suchbaum für ein Zwei-Personen-Spiel ist in Abb. 6.21 mit den Bewertungen aller Blattknoten dargestellt. Verwenden Sie die Minimax-Suche mit ÿ-ÿ- Beschneidung von links nach rechts. Streichen Sie alle nicht besuchten Knoten durch und geben Sie für jeden inneren Knoten die optimale Ergebnisbewertung an. Markieren Sie den gewählten Weg. (b) Testen Sie sich selbst mit dem Applet [Win] von P. Winston.**

# Argumentieren mit Unsicherheit

# 7

Wir haben bereits in Abschn. gezeigt. 4 mit dem Tweety-Problem, dass zweiwertige Logik zu Problemen im alltäglichen Denken führt. In diesem Beispiel führen die Aussagen *Tweety ist ein Pinguin, Pinguine sind Vögel* und *Alle Vögel können fliegen* zu der (semantisch falschen) Schlussfolgerung, dass *Tweety fliegen kann.* Die Wahrscheinlichkeitstheorie bietet eine Sprache, in der wir die Aussage formalisieren können. *Fast alle Vögel können fliegen* und daraus Rückschlüsse ziehen. Die Wahrscheinlichkeitstheorie ist eine bewährte Methode, die wir hier verwenden können, da die Unsicherheit darüber, ob Vögel fliegen können, gut durch einen Wahrscheinlichkeitswert modelliert werden kann. Wir werden zeigen, dass Aussagen wie „ 99 % *aller Vögel können fliegen“* zusammen mit der Wahrscheinlichkeitslogik zu korrekten Schlussfolgerungen führen

Das Denken unter Unsicherheit und begrenzten Ressourcen spielt in Alltagssituationen und auch in vielen technischen Anwendungen von KI eine große Rolle. In diesen Bereichen sind heuristische Verfahren sehr wichtig, wie wir bereits in Kap. 6. Beispielsweise nutzen wir heuristische Techniken bei der Parkplatzsuche im Stadtverkehr. Heuristiken allein reichen oft nicht aus, insbesondere wenn bei unvollständigem Wissen eine schnelle Entscheidung erforderlich ist, wie das folgende Beispiel zeigt. Ein Fußgänger überquert die Straße und ein Auto nähert sich schnell. Um einen schweren Unfall zu verhindern, muss der Fußgänger schnell reagieren. Er ist nicht in der Lage, sich um vollständige Informationen über den Zustand der Welt zu kümmern, die er für die in Kap. besprochenen Suchalgorithmen benötigen würde. 6. Er muss daher unter den gegebenen Randbedingungen (wenig Zeit und wenig, möglicherweise unsicheres Wissen) zu einer optimalen Entscheidung kommen. Wenn er zu lange nachdenkt, wird es gefährlich. In dieser und vielen ähnlichen Situationen (siehe Abb. 7.1 auf Seite 114) ist eine Methode zum Schlussfolgern mit unsicherem und unvollständigem Wissen erforderlich.

Wir wollen die verschiedenen Möglichkeiten des Denkens unter Unsicherheit anhand eines einfachen medizinischen Diagnosebeispiels untersuchen. Wenn ein Patient Schmerzen im rechten Unterbauch und eine erhöhte Anzahl weißer Blutkörperchen (Leukozyten) verspürt, liegt der Verdacht nahe, dass es sich um eine Blinddarmentzündung handeln könnte. Wir modellieren diese Beziehung mithilfe der Aussagenlogik mit der Formel

*Magenschmerzen rechts unten* ÿ *Leukozyten* > 10000 ÿ *Blinddarmentzündung*

W. Ertel, *Einführung in die künstliche Intelligenz,* Bachelor-Themen in Informatik,
DOI 10.1007/978-0-85729-299-5_7, 

**Abb. 7.1 „Lehnen wir uns einfach zurück und überlegen, was zu tun ist!“**

**Wenn wir das dann wissen**

***Magenschmerzen rechts unten*** **ÿ** ***Leukozyten*** **> 10000**

**wahr ist, dann können wir den Modus Ponens verwenden, um *eine Appendizitis abzuleiten.* Dieses Modell ist eindeutig zu grob. 1976 erkannten Shortliffe und Buchanan dies, als sie ihr medizinisches Expertensystem MYCIN bauten [Sho76]. Sie entwickelten einen Kalkül mithilfe sogenannter Gewissheitsfaktoren, der es ermöglichte, die Gewissheit von Fakten und Regeln darzustellen.**
**Einer Regel A ÿ B wird ein Sicherheitsfaktor ÿ zugewiesen. Die Semantik einer Regel A ÿÿ B wird über die bedingte Wahrscheinlichkeit P (B|A) = ÿ definiert. Im obigen Beispiel könnte die Regel dann lauten**

***Magenschmerzen rechts unten*** **ÿ** ***Leukozyten*** **> 10000 ÿ0,6** ***Blinddarmentzündung.***

**Um mit dieser Art von Formeln zu argumentieren, verwendeten sie eine Infinitesimalrechnung, um die Faktoren von Regeln zu verbinden. Es stellte sich jedoch heraus, dass mit dieser Rechnung inkonsistente Ergebnisse abgeleitet werden konnten.**

Wie in Kap. besprochen. 4 gab es auch Versuche, dieses Problem durch den Einsatz nicht-monotoner Logik und Default-Logik zu lösen, die jedoch letztlich erfolglos blieben. Die Dempster-Schäfer-Theorie weist einem logischen Satz A eine Glaubensfunktion Bel(A) zu, deren Wert den Beweisgrad für die Wahrheit von A angibt. Aber auch dieser Formalismus weist Schwächen auf, was in [Pea88] anhand einer Variante von gezeigt wird das Tweety-Beispiel. Selbst die Fuzzy-Logik, die vor allem in der Kontrolltheorie erfolgreich ist, weist bei der Argumentation unter Unsicherheit in komplexeren Anwendungen erhebliche Schwächen auf [Elk93].

Seit etwa Mitte der 1980er Jahre hat die Wahrscheinlichkeitstheorie immer mehr Einfluss auf die KI gehabt [Pea88, Che85, Whi96, Jen01]. Im Bereich des Denkens mit Bayes'schen Netzen bzw. der subjektiven Wahrscheinlichkeit hat es sich einen festen Platz unter den erfolgreichen KI-Techniken gesichert. Anstelle der in der Logik bekannten Implikation (materielle Implikation) wird hier die bedingte Wahrscheinlichkeit verwendet, die das alltägliche Kausaldenken deutlich besser modelliert. Das Schließen mit Wahrscheinlichkeiten profitiert stark von der Tatsache, dass die Wahrscheinlichkeitstheorie ein Hunderte Jahre alter, gut etablierter Zweig der Mathematik ist.

In diesem Kapitel wählen wir einen eleganten, aber für ein Lehrbuch etwas ungewöhnlichen Einstiegspunkt in dieses Gebiet. Nach einer kurzen Einführung in die wichtigsten Grundlagen, die hier für das Schließen mit Wahrscheinlichkeit benötigt werden, beginnen wir mit einem einfachen, aber wichtigen Beispiel für das Schließen mit unsicherem und unvollständigem Wissen. Auf ganz natürliche, fast zwingende Weise werden wir zur Methode der maximalen Entropie (MaxEnt) geführt. Anschließend zeigen wir den Nutzen dieser Methode in der Praxis anhand des medizinischen Expertensystems LEXYMED. Abschließend stellen wir die inzwischen weit verbreitete Argumentation mit Bayes'schen Netzen vor und zeigen die Beziehung zwischen den beiden Methoden.

## 7.1 Rechnen mit Wahrscheinlichkeiten

Der Leser, der mit der Wahrscheinlichkeitstheorie vertraut ist, kann diesen Abschnitt überspringen. Für alle anderen geben wir einen kurzen Überblick und empfehlen ein paar geeignete Lehrbücher wie [Ros09, FPP07].

Die Wahrscheinlichkeit eignet sich besonders gut zur Modellierung des Denkens unter Unsicherheit. Ein Grund dafür ist, dass Wahrscheinlichkeiten intuitiv leicht zu interpretieren sind, was im folgenden elementaren Beispiel zu sehen ist.

*Beispiel 7.1* Bei einem einzelnen Würfelwurf (Experiment) beträgt die Wahrscheinlichkeit des Ereignisses „eine Sechs würfeln" 1/6, während die Wahrscheinlichkeit des Ereignisses „eine ungerade Zahl würfeln" 1/2 beträgt.

**Definition 7.1** Sei ÿ die endliche Menge von *Ereignissen* für ein Experiment. Jedes Ereignis ÿ ÿ ÿ repräsentiert ein mögliches Ergebnis des Wurfs. Wenn sich diese Ereignisse wi ÿ ÿ gegenseitig ausschließen, aber alle möglichen Ergebnisse des Versuchs abdecken, werden sie *Elementarereignisse genannt.*

***Beispiel 7.2*** **Für einen einzelnen Wurf eines Spielwürfels**

$$ÿ = \{1, 2, 3, 4, 5, 6\}$$

**weil keine zwei dieser Ereignisse gleichzeitig passieren können. Eine gerade Zahl würfeln ({2, 4, 6}) ist daher kein Elementarereignis und auch kein Würfeln einer Zahl kleiner als fünf ({1, 2, 3, 4}), weil {2, 4, 6}ÿ{1, 2, 3, 4}={2, 4} = ÿ.**

**Gegeben zwei Ereignisse A und B, ist A ÿB ebenfalls ein Ereignis. ÿ selbst wird als das *Gewisse*** bezeichnet ***Ereignis*** **und die leere Menge ÿ das *unmögliche Ereignis.***

**Im Folgenden verwenden wir die aussagenlogische Notation für Mengenoperationen. Das heißt, für die Menge AÿB schreiben wir AÿB. Dies ist nicht nur eine syntaktische Transformation, Vielmehr ist es auch semantisch korrekt, da der Schnittpunkt zweier Mengen definiert ist als**

$$x ÿ A ÿ B ÿ x ÿ A ÿ x ÿ B.$$

**Da dies die Semantik von A ÿ B ist, können und werden wir diese Notation verwenden. Das ist auch wahr für die anderen Mengenoperationen Vereinigung und Komplement, und wir werden es tun, wie in gezeigt In der folgenden Tabelle verwenden Sie auch für sie die aussagenlogische Notation.**

| Mengennotation | Aussagenlogik | Beschreibung |
|---|---|---|
| A ÿ B | A ÿ B | Kreuzung/und |
| A ÿ B | A ÿ B | Union/oder |
| A | ¬A | Ergänzung/Negation |
| ÿ | w | Bestimmtes Ereignis/wahr |
| ÿ | F | Unmögliches Ereignis/falsch |

**Die hier verwendeten Variablen (zum Beispiel A, B usw.) werden *Zufallsvariablen*** genannt **in der Wahrscheinlichkeitstheorie. Wir werden hier nur diskrete Zufallsvariablen mit endlichen Grundfunktionen verwenden. Die Variable *face_number* für einen Würfelwurf ist diskret mit den Werten 1, 2, 3, 4, 5, 6. Die Wahrscheinlichkeit, eine Fünf oder Sechs zu würfeln, beträgt 1/3. Das kann sein beschrieben von**

$$P(\textit{Gesichtsnummer} ÿ \{5, 6\}) = P(\textit{Gesichtsnummer} = 5 ÿ \textit{Gesichtsnummer} = 6) = 1/3.$$

**Der Begriff der Wahrscheinlichkeit soll uns eine möglichst objektive Beschreibung geben möglicher Teil unseres „Glaubens“ oder unserer „Überzeugung“ über das Ergebnis eines Experiments. Alle Zahlen im Intervall [0, 1] sollten möglich sein, wobei 0 die Wahrscheinlichkeit dafür ist unmögliches Ereignis und 1 die Wahrscheinlichkeit des bestimmten Ereignisses. Wir kommen hierher die folgende Definition.**

**Definition 7.2** Sei $\ddot{y} = \{\ddot{y}1, \ddot{y}2, ..., \ddot{y}n\}$ endlich. Es gibt kein bevorzugtes Elementarereignis, was bedeutet, dass wir eine Symmetrie annehmen, die mit der Häufigkeit des Auftretens jedes Elementarereignisses zusammenhängt. Die *Wahrscheinlichkeit* P (A) des Ereignisses A beträgt dann

$$P(A) = \frac{|A|}{|\ddot{y}|} = \frac{\text{Anzahl günstiger Fälle für A}}{\text{Anzahl möglicher Fälle}}.$$

Daraus folgt sofort, dass jedes Elementarereignis die Wahrscheinlichkeit $1/|\ddot{y}|$ hat. Die Forderung, dass Elementarereignisse die gleiche Wahrscheinlichkeit haben, wird *Laplace-Annahme* genannt und die dadurch berechneten Wahrscheinlichkeiten werden *Laplace-Wahrscheinlichkeiten genan* Diese Definition stößt an ihre Grenzen, wenn die Zahl der Elementarereignisse unendlich wird. Da wir hier jedoch nur endliche Ereignisräume betrachten, stellt dies kein Problem dar. Zur Beschreibung von Ereignissen verwenden wir Variablen mit der entsprechenden Anzahl an Werten. Beispielsweise kann eine Variable *eye_color* die Werte *grün, blau,* braun annehmen . *eye_color = blue* beschreibt dann ein Ereignis, da es sich um einen Satz mit den Wahrheitswerten t [illegible] (booleschen) Variablen ist die Variable selbst bereits ein Satz. Hier genügt es beispielsweise, P *(JohnCalls)* statt P *(JohnCalls =* t) zu schreiben.

*Beispiel 7.3* Nach dieser Definition beträgt die Wahrscheinlichkeit, eine gerade Zahl zu würfeln

$$P(\textit{face\_number} \in \{2, 4, 6\}) = \frac{|\{2, 4, 6\}|}{|\{1, 2, 3, 4, 5, 6\}|} = \frac{3}{6} = \frac{1}{2}.$$

Die folgenden wichtigen Regeln ergeben sich direkt aus der Definition.

**Satz 7.1** 1. $P(\ddot{y}) = 1$.
2. $P(\ddot{y}) = 0$, *was bedeutet, dass das unmögliche Ereignis eine Wahrscheinlichkeit von* 0 hat.
3. *Für paarweise exklusive Ereignisse* A *und* B *gilt* $P(A \ddot{y} B) = P(A) + P(B)$.
4. *Für zwei komplementäre Ereignisse* A *und* $\neg$A *gilt* $P(A) + P(\neg A) = 1$.
5. *Für beliebige Ereignisse* A *und* B *gilt* $P(A \ddot{y} B) = P(A) + P(B) \ddot{y} P(A \ddot{y} B)$.
6. *Für* $A \ddot{y} B$ *gilt* $P(A) \ddot{y} P(B)$.
7. *Wenn* A1,...,An *Elementarereignisse sind,* $\sum_{i=1}^{N} P(Ai) = 1$ *(Normalisierung dann Bedingung).*

Der Ausdruck P (A ÿ B) oder äquivalent P (A, B) steht für die Wahrscheinlichkeit der Ereignisse A ÿ B. Wir sind oft an den Wahrscheinlichkeiten aller Elementarereignisse interessiert Ereignisse, also aller Kombinationen aller Werte der Variablen A und B. Für die binäre Variablen A und B sind dies P (A, B), P (A,¬B), P (¬A,B), P (¬A,¬B).
Wir nennen den Vektor

$$(P(A, B), P(A,\neg B), P(\neg A, B), P(\neg A,\neg B))$$

bestehend aus diesen vier Werten eine *Verteilung* oder *gemeinsame Wahrscheinlichkeitsverteilung* der Variablen A und B. Eine Kurzform hierfür ist P(A, B). Die Verteilung im Fall von Zwei Variablen können gut in Form einer Tabelle (Matrix) visualisiert werden, dargestellt als folgt:

| P(A, B) | B = w | B = f |
|---|---|---|
| A = w | P (A, B) | P (A,¬B) |
| A = f | P(¬A,B) | P (¬A,¬B) |

Für die d Variablen X1,...,Xd mit jeweils n Werten hat die Verteilung die Werte P (X1 = x1,...,Xd = xd ) und x1,...,xd , die jeweils n verschiedene Werte annehmen. Die Verteilung kann daher als d-dimensionale Matrix mit einer Summe dargestellt werden von $n^D$ Elemente. Aufgrund der Normalisierungsbedingung aus Satz 7.1 auf Seite 117 gilt jedoch einer dieser n $^D$ Die Werte sind redundant und die Verteilung ist charakterisiert von $n^D$ ÿ 1 eindeutige Werte.

## 7.1.1 Bedingte Wahrscheinlichkeit

*Beispiel 7.4* Auf der Landsdowne Street in Boston wird die Geschwindigkeit von 100 Fahrzeugen gemessen. Bei jeder Messung wird auch vermerkt, ob es sich beim Fahrer um einen Schüler handelt. Der Ergebnisse sind

| Ereignis | Frequenz | Relative Frequenz |
|---|---|---|
| Fahrzeug beobachtet | 100 | 1 |
| Fahrer ist Student (S) | 30 | 0,3 |
| Geschwindigkeit zu hoch (G) | 10 | 0,1 |
| Der Fahrer ist Student und fährt zu schnell (S ÿ G) | 5 | 0,05 |

Wir stellen die Frage: *Beschleunigen Schüler häufiger als der Durchschnitt? Sohn, oder als Nicht-Studenten?1*

Die Antwort ergibt sich aus der Wahrscheinlichkeit

$$P(G|S) = \frac{|\text{Fahrer ist Student und fährt zu schnell}|}{|\text{Fahrer ist Student}|} = \frac{5}{30} = \frac{1}{6} \text{ ÿ } 0{,}17$$

1Die berechneten Wahrscheinlichkeiten können nur dann für fortgesetzte Aussagen verwendet werden, wenn die gemessene Stichprobe vorliegt (100 Fahrzeuge) ist repräsentativ. Ansonsten können nur Aussagen zu den beobachteten 100 Fahrzeugen gemacht werden gemacht sein.

wegen Geschwindigkeitsüberschreitung, sofern der Fahrer ein Student ist. Dies unterscheidet sich offensichtlich von der A-priori-Wahrscheinlichkeit P (G) = 0,1 für Geschwindigkeitsüberschreitungen. Für die A-priori-Wahrscheinlichkeit ist der Ereignisraum nicht durch zusätzliche Bedingungen begrenzt.

**Definition 7.3** Für zwei Ereignisse A und B ist die Wahrscheinlichkeit P (A|B) für A unter der Bedingung B *(bedingte Wahrscheinlichkeit)* definiert durch

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

In Beispiel 7.4 sehen wir, dass im Fall eines endlichen Ereignisraums die bedingte Wahrscheinlichkeit P (A|B) als die Wahrscheinlichkeit von A ∩ B verstanden werden kann, wenn wir nur das Ereignis B betrachten, also als

$$P(A|B) = \frac{|A \cap B|}{|B|}.$$

Diese Formel kann leicht mithilfe von Definition 7.2 auf Seite 117 abgeleitet werden

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{|A \cap B|}{|\Omega|}}{\frac{|B|}{|\Omega|}} = \frac{|A \cap B|}{|B|}.$$

**Definition 7.4** Wenn für zwei Ereignisse A und B

$$P(A|B) = P(A),$$

dann heißen diese Ereignisse unabhängig.

Somit sind A und B unabhängig, wenn die Wahrscheinlichkeit des Ereignisses A nicht durch das Ereignis B beeinflusst wird.

**Satz 7.2** *Für unabhängige Ereignisse* A *und* B *folgt aus der Definition, dass*

$$P(A \cap B) = P(A) \cdot P(B).$$

*Beispiel 7.5* Bei einem Wurf mit zwei Würfeln beträgt die Wahrscheinlichkeit, zwei Sechsen zu würfeln, 1/36, wenn die beiden Würfel unabhängig sind, weil

$$P(D1 = 6 \ddot{y} D2 = 6) = P(D1 = 6) \cdot P(D2 = 6) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36},$$

wobei die erste Gleichung nur dann wahr ist, wenn die beiden Würfel unabhängig sind. Wenn zum Beispiel durch eine magische Kraft Würfel 2 immer gleich Würfel 1 ist, dann

$$P(D1 = 6 \ddot{y} D2 = 6) = \frac{1}{6} \cdot$$

**Kettenregel**

Das Lösen der Definition der bedingten Wahrscheinlichkeit für $P(A \ddot{y} B)$ führt zur sogenannten Produktregel

$$P(A \ddot{y} B) = P(A|B)P(B),$$

was wir sofort für den Fall von n Variablen verallgemeinern. Durch wiederholte Anwendung der obigen Regel erhalten wir die *Kettenregel*

$$\begin{aligned} &P(X1,...,Xn) \\ &= P(Xn|X1,...,Xn\ddot{y}1) \cdot P(X1,...,Xn\ddot{y}1) \\ &= P(Xn|X1,...,Xn\ddot{y}1) \cdot P(Xn\ddot{y}1|X1,...,Xn\ddot{y}2) \cdot P(X1,...,Xn\ddot{y}2) \\ &= P(Xn|X1,...,Xn\ddot{y}1) \cdot P(Xn\ddot{y}1|X1,...,Xn\ddot{y}2) \cdot \cdots \cdot P(X2|X1) \cdot P(X1) \\ &= \prod_{i=1}^{N} P(Xi|X1 ...,Xi\ddot{y}1), \end{aligned} \tag{7.1}$$

mit dem wir eine Verteilung als Produkt bedingter Wahrscheinlichkeiten darstellen können. Da die Kettenregel für alle Werte der Variablen X1,...,Xn gilt, wurde sie für die Verteilung mit dem Symbol P formuliert.

**Marginalisierung**

Weil $A \ddot{y} (A \ddot{y} B) \ddot{y} (A \ddot{y} \neg B)$ für binäre Variablen A und B gilt

$$P(A) = P((A \ddot{y} B) \ddot{y} (A \ddot{y} \neg B)) = P(A \ddot{y} B) + P(A \ddot{y} \neg B).$$

Durch Summation über die beiden Werte von B wird die Variable B eliminiert. Analog kann für beliebige Variablen X1,...,Xd eine Variable, beispielsweise Xd , durch Summation über alle ihre Variablen eliminiert werden, und wir erhalten

$$P(X1 = x1,...,Xd\ddot{y}1 = xd\ddot{y}1) = \sum_{xd} P(X1 = x1,...,Xd\ddot{y}1 = xd\ddot{y}1, Xd = xd).$$

**Die Anwendung dieser Formel nennt man Marginalisierung.** Diese Summation kann mit den Variablen **X1,...,Xdÿ1 fortgesetzt werden , bis nur noch eine Variable übrig ist. Marginalisierung** kann auch auf die Verteilung **P(X1,...,Xd ) angewendet werden . Die resultierende Verteilung P(X1,...,Xdÿ1) heißt *Randverteilung.* Es ist vergleichbar mit der Projektion eines rechteckigen Quaders auf eine ebene Fläche. Hier ist das dreidimensionale Objekt auf der Kante oder dem „Rand“ des Quaders gezeichnet, also auf einer zweidimensionalen Menge. Sowohl In diesen Fällen wird die Dimensionalität um eins reduziert.**

***Beispiel 7.6*** **Wir beobachten die Menge aller Patienten, die akut zum Arzt kommen Magenschmerzen. Für jeden Patienten wird der Leukozytenwert gemessen, der ein Maß darstellt für die relative Häufigkeit weißer Blutkörperchen im Blut. Wir definieren die Variable *Leuko,* was genau dann zutrifft, wenn der Leukozytenwert größer als 10.000 ist. Das weist auf eine Infektion im Körper hin. Ansonsten definieren wir die Variable *App,* die erzählt Wir informieren Sie, ob der Patient an einer Blinddarmentzündung, also einem entzündeten Blinddarm, leidet. Die Verteilung *P(App,Leuko)* dieser beiden Variablen ist in der folgenden Tabelle angegeben:**

| *P(App,Leuko)* | *App* | *¬App* | Total |
|---|---|---|---|
| *Leuko* | 0,23 | 0,31 | 0,54 |
| *¬Leuko* | 0,05 | 0,41 | 0,46 |
| Gesamt | 0,28 | 0,72 | 1 |

**In der letzten Zeile ist die Summe über die Zeilen angegeben und in der letzten Spalte die Summe der Spalten ist angegeben. Zu diesen Beträgen kommt es durch Marginalisierung. Zum Beispiel wir ablesen**

$$P(Leuko) = P(App, Leuko) + P(\neg App, Leuko) = 0{,}54.$$

**Die angegebene Verteilung *P(App,Leuko)* könnte aus einer Befragung deutscher Ärzte stammen, Zum Beispiel. Daraus können wir dann die bedingte Wahrscheinlichkeit berechnen**

$$P(Leuko|App) = \frac{P(Leuko,App)}{P(App)} = 0{,}82$$

**Daraus lässt sich schließen, dass etwa 82 % aller Blinddarmentzündungen zu einem hohen Leukozytenwert führen. Solche Werte werden in der medizinischen Fachliteratur veröffentlicht. Allerdings wäre die bedingte Wahrscheinlichkeit P *(App|Leuko)* eigentlich viel hilfreicher für die Diagnose Blinddarmentzündung, wird nicht veröffentlicht. Um dies zu verstehen, leiten wir zunächst ein einfaches Aber ab sehr wichtige Formel.**

## Satz von Bayes

**Das Vertauschen von A und B in Definition 7.3 auf Seite 119 führt zu**

$$P(A|B) = \frac{P(A \ddot{y} B)}{P(B)} \quad \text{und} \quad P(B|A) = \frac{P(A \ddot{y} B)}{P(A)}.$$

Indem wir die beiden Gleichungen nach P (AÿB) lösen und gleichsetzen, erhalten wir den *Satz von Bayes*

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}, \tag{7.2}$$

was wir sofort auf das Blinddarmentzündungsproblem anwenden und erhalten

$$P(App|Leuko) = \frac{P(Leuko|App) \cdot P(App)}{P(Leuko)} = \frac{0,82 \cdot 0,28}{0,54} = 0,43. \tag{7.3}$$

Warum wird nun P *(Leuko|App)* veröffentlicht, aber nicht P *(App|Leuko)?*

Unter der Annahme, dass Blinddarmentzündung den menschlichen Körper unabhängig von der Rasse betrifft, ist P *(Leuko|App)* ein universeller Wert, der auf der ganzen Welt gilt. In (7.3) sehen wir, dass P *(App|Leuko)* nicht universell ist, da dieser Wert von den A-priori-Wahrscheinlichkeiten P *(App)* und P *(Leuko) beeinflusst wird.* Jedes davon kann je nach Lebensumständen v Beispielsweise hängt P *(Leuko)* davon ab, ob es in einer Population viele oder wenige Infektionskrankheiten gibt. Dieser Wert könnte zwischen den Tropen und kälteren Regionen möglicherweise erheblich variieren. Der Satz von Bayes ermöglicht jedoch eine einfache Berechnung des für die Diagnose relevanten P *(App|Leuko)* aus dem allgemeingültigen Wert P (Leuko|App).

Bevor wir näher auf dieses Beispiel eingehen und es zu einem medizinischen Expertensystem für Blinddarmentzündung erweitern, müssen wir zunächst die notwendige probabilistische Schlussfolgerung einführ Mechanismus.

## 7.2 Das Prinzip der maximalen Entropie

Wir werden nun anhand eines Inferenzbeispiels zeigen, dass ein Kalkül zum Schließen unter Unsicherheit mithilfe der Wahrscheinlichkeitstheorie realisiert werden kann. Allerdings werden wir bald sehen, dass die eingefahrenen probabilistischen Pfade schnell ein Ende haben. Insbesondere wenn zu wenig Wissen vorhanden ist, um die notwendigen Gleichungen zu lösen, sind neue Ideen gefragt. Der amerikanische Physiker ET Jaynes leistete in den 1950er Jahren Pionierarbeit auf diesem Gebiet. Er behauptete, dass man bei fehlendem Wissen die Entropie der gewünschten Wahrscheinlichkeitsverteilung maximieren kann, und wandte dieses Prinzip auf viele Beispiele in [Jay57, Jay03] an. Dieses Prinzip wurde dann weiterentwickelt [Che83, Nil86, Kan89, KK92] und ist mittlerweile ausgereift und technologisch anwendbar, was wir am Beispiel des LEXMED-Projekts in Abschn. 7.3.

### 7.2.1 Eine Inferenzregel für Wahrscheinlichkeiten

Wir wollen eine zum Modus Ponens analoge Inferenzregel für unsicheres Wissen ableiten. Aus der Kenntnis eines Satzes A und einer Regel A ÿ B soll die Schlussfolgerung B gezogen werden. Kurz und bündig formuliert lautet das:

$$\frac{A, A \ddot{y} B}{B}.$$

Die Verallgemeinerung für Wahrscheinlichkeitsregeln ergibt

$$\frac{P(A) = \ddot{y}, P(B|A) = \ddot{y}}{P(B) = ?}.$$

Gegeben seien die beiden Wahrscheinlichkeitsregeln ÿ, ÿ und der gewünschte Wert P(B). Durch Marginalisierung erhalten wir die gewünschte Randverteilung

$$P(B) = P(A, B) + P(\neg A, B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A).$$

Die drei Werte P (A), P (¬A), P (B|A) auf der rechten Seite sind bekannt, der Wert P (B|¬A) ist jedoch unbekannt. Über P (B) können wir mit der klassischen Wahrscheinlichkeitstheorie keine exakte Aussage treffen, aber wir können allenfalls P (B) ÿ P (B|A) · P (A) abschätzen.

Wir betrachten nun die Verteilung

$$P(A, B) = (P(A, B), P(A, \neg B), P(\neg A, B), P(\neg A, \neg B))$$

und stellen Sie stenografisch die vier Unbekannten vor

$$p1 = P(A, B), p2 = P(A, \neg B), p3 = P(\neg A, B), p4 = P(\neg A, \neg B).$$

Diese vier Parameter bestimmen die Verteilung. Wenn sie alle bekannt sind, kann jede Wahrscheinlichkeit für die beiden Variablen A und B berechnet werden. Um die vier Unbekannten zu berechnen, werden vier Gleichungen benötigt. Eine Gleichung ist in Form der Normalisierungsbedingung bereits bekannt

$$p1 + p2 + p3 + p4 = 1.$$

Daher sind drei weitere Gleichungen erforderlich. In unserem Beispiel sind jedoch nur zwei Gleichungen bekannt.

Aus den gegebenen Werten P (A) = ÿ und P (B|A) = ÿ berechnen wir

$$P(A, B) = P(B|A) \cdot P(A) = \ddot{y}\ddot{y}$$

Und

$$P(A) = P(A, B) + P(A, \neg B).$$

Daraus können wir das folgende Gleichungssystem aufstellen und soweit wie möglich lösen:

$$p1 = \ddot{y}, \tag{7.4}$$

$$p1 + p2 = \ddot{y}, \tag{7,5}$$

$$p1 + p2 + p3 + p4 = 1, \tag{7,6}$$

(7.4) in (7.5): $p2 = \ddot{y} \ddot{y} \ddot{y} = \ddot{y}(1 \ddot{y} \ddot{y})$, (7.7)

(7.5) in (7.6): $p3 + p4 = 1 \ddot{y} \ddot{y}$. (7.8)

Damit sind die Wahrscheinlichkeiten p1, p2 für die Interpretationen (A, B) und (A,¬B) bekannt, für die Werte p3, p4 bleibt jedoch nur noch eine Gleichung übrig. Um trotz dieses fehlenden Wissens zu einer eindeutigen Lösung zu kommen, ändern wir unseren Standpunkt. Wir verwenden die gegebene Gleichung als Randbedingung für die Lösung eines Optimieru

Gesucht wird eine Verteilung p (für die Variablen p3, p4) , die die Entropie maximiert

$$H(p) = \ddot{y}_{i=1}^{n} pi \ln pi = \ddot{y}p3 \ln p3 \ddot{y} p4 \ln p4 \tag{7.9}$$

unter der Nebenbedingung p3 +p4 = 1ÿÿ (7.8). Warum genau sollte die Entropiefunktion maximiert werden? Da uns Informationen über die Verteilung fehlen, müssen diese irgendwie hinzugefügt werden. Wir könnten einen Ad-hoc-Wert festlegen, zum Beispiel p3 = 0,1. Es ist jedoch besser, die Werte p3 und p4 so zu bestimmen, dass die hinzugefügte Informatio Wir können zeigen (Abschn. 8.4.2 und [SW76]), dass die Entropie die Unsicherheit einer Verteilung bis zu einem konstanten Faktor misst. Die negative Entropie ist dann ein Maß für die Menge an Informationen, die eine Verteilung enthält. Durch die Maximierung der Entropie wird der Informationsgehalt der Verteilung minimiert. Um dies zu veranschaulichen, wird die Entropiefunktion für den zweidimensionalen Fall in Abb. 7.2 auf Seite 125 g

Um das Maximum der Entropie unter der Nebenbedingung p3 + p4 ÿ 1 + ÿ = 0 zu bestimmen, verwenden wir die Methode der Lagrange-Multiplikatoren [Ste07]. Die Lagrange-Funktion lautet

$$L = \ddot{y}p3 \ln p3 \ddot{y} p4 \ln p4 + \ddot{y}(p3 + p4 \ddot{y} 1 + \ddot{y}).$$

Durch die partiellen Ableitungen nach p3 und p4 erhalten wir:

$$\frac{\ddot{y}L}{} = \ddot{y}\ln p3 \ddot{y} 1 + \ddot{y} = 0, \ddot{y}p3$$

$$\frac{\ddot{y}L}{} = \ddot{y}\ln p4 \ddot{y} 1 + \ddot{y} = 0 \ddot{y}p4$$

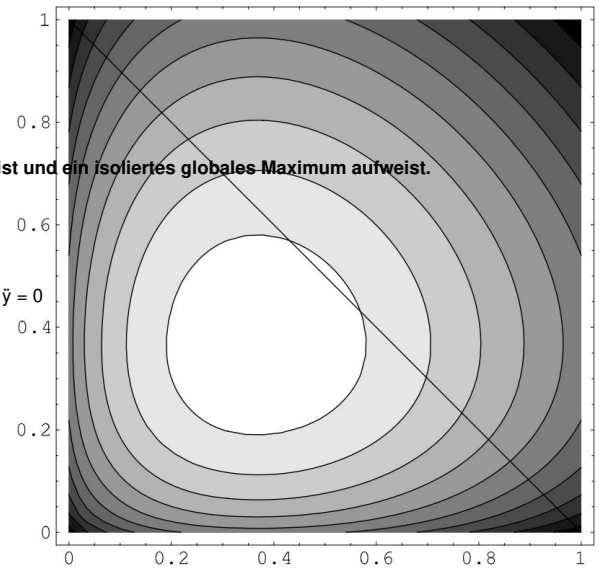



**Abb. 7.2**

***Höhenliniendiagramm*** **der zweidimensionalen Entropiefunktion. Wir sehen, dass es im gesamten Einheitsquadrat streng konvex ist und ein isoliertes globales Maximum aufweist. Ebenfalls markiert ist die Nebenbedingung p3 + p4 = 1 als Sonderfall der hier relevanten** Bedingung **p3 + p4** ÿ 1 + ÿ = 0 für ÿ = 0

**und berechnen**

$$p3 = p4 = \frac{1 \ddot{y} \ddot{y}}{2}.$$

**Jetzt können wir den gewünschten Wert berechnen**

$$P(B) = P(A, B) + P(\neg A, B) = p1 + p3 = \ddot{y}\ddot{y} + = \ddot{y}\ddot{y}\ddot{y}\,2 \qquad \frac{1 \ddot{y} \ddot{y} 1}{2} \qquad - \quad + \frac{1}{2}.$$

**Das Einsetzen von ÿ und ÿ ergibt**

$$P(B) = P(A)\,P(B|A) \ddot{y}\,2 \qquad \frac{1}{-} \quad 1 \; \underset{+\,2}{-}.$$

**P (B) ist in Abb. 7.3 auf Seite 126 für verschiedene Werte von P (B|A) dargestellt. Wir sehen, dass im Fall der zweiwertigen Kante, das heißt, wenn P (B) und P (B|A) die Werte 0 oder 1 annehmen, die probabilistische Inferenz denselben Wert für P (B) zurückgibt wie der Modus Ponens. Wenn A und B|A beide wahr sind, ist auch B wahr. Ein interessanter Fall ist P (A) = 0, in dem ¬A wahr ist. Modus ponens kann hier nicht angewendet werden, aber unsere Formel ergibt den Wert 1/2 für P (B), unabhängig von P (B|A). Wenn A falsch ist, wissen wir nichts über B, was genau unsere Intuition widerspiegelt. Der Fall, dass P (A) = 1 und P (B|A) = 0 ist, wird auch von der Aussagenlogik abgedeckt. Hier ist A wahr und A ÿ B falsch und somit A ÿ ¬B wahr. Dann ist B falsch. Die horizontale Linie in der Abbildung bedeutet, dass wir im Fall von P (B|A) = 1/2 keine Vorhersage über B machen können. Zwischen diesen Punkten ändert sich P (B) linear bei Änderungen von P (A) oder P (B|A).**

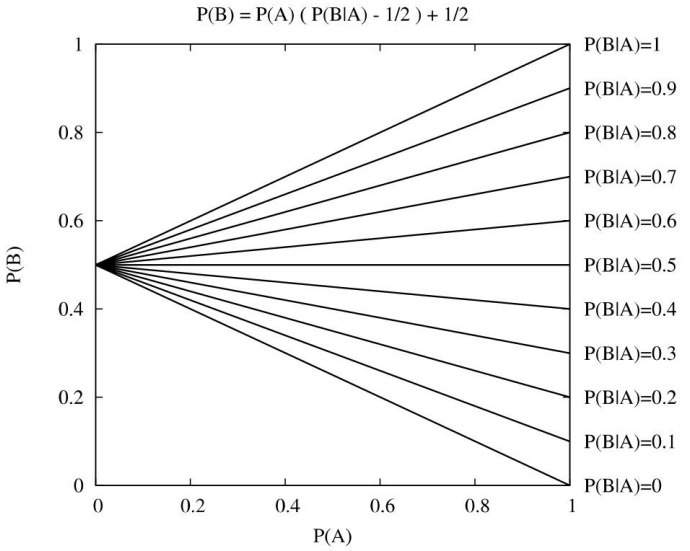



**Abb. 7.3 Kurvenfeld für P (B) als Funktion von P (A) für verschiedene Werte von P (B|A)**

**Satz 7.3** ***Es gebe einen konsistenten2 Satz linearer Wahrscheinlichkeitsgleichungen. Dann gibt es ein eindeutiges Maximum für die Entropiefunktion mit den gegebenen Gleichungen als Randbedingungen. Die dadurch definierte MaxEnt-Verteilung hat unter den Randbedingungen einen minimalen Informationsgehalt.***

**Aus diesem Satz folgt, dass es keine Verteilung gibt, die die Einschränkungen erfüllt und eine höhere Entropie aufweist als die MaxEnt-Verteilung. Eine Rechnung, die zu einer geringeren Entropie führt, fügt zusätzliche Ad-hoc-Informationen ein, die nicht gerechtfertigt sind.**

**Wenn wir die obige Berechnung von P (B) genauer betrachten, sehen wir, dass die beiden Werte p3 und p4 immer symmetrisch auftreten. Das bedeutet, dass sich durch den Austausch der beiden Variablen das Ergebnis nicht ändert. Somit ist das Endergebnis p3 = p4. Die sogenannte Indifferenz dieser beiden Variablen führt dazu, dass sie von MaxEnt gleichgesetzt werden. Dieser Zusammenhang gilt allgemein:**

**Definition 7.5 Wenn ein beliebiger Austausch von zwei oder mehr Variablen in den La-Grange-Gleichungen zu äquivalenten Gleichungen führt, werden diese Variablen als** ***indifferent bezeichnet.***

**2Ein Satz probabilistischer Gleichungen heißt konsistent, wenn es mindestens eine Lösung gibt, also eine Verteilung, die alle Gleichungen erfüllt.**

**Satz 7.4** *Wenn eine Menge von Variablen* {pi1 ,...,pik } *indifferent ist, dann liegt das Maximum der Entropie unter den gegebenen Einschränkungen an dem Punkt, an dem* pi1 = pi2 =···= pik .

Mit diesem Wissen hätten wir die beiden Variablen p3 und p4 sofort gleichsetzen können (ohne die Lagrange-Gleichungen zu lösen).

### 7.2.2 Maximale Entropie ohne explizite Einschränkungen

Wir betrachten nun den Fall, in dem kein Wissen gegeben ist. Dies bedeutet, dass es sich nicht um die Normalisierungsbedingung handelt

$$p1 + p2 + \cdots + pn = 1,$$

es gibt keine Einschränkungen. Alle Variablen sind daher indifferent. Deshalb können wir [3] Setze sie gleich und es folgt, dass p1 = p2 =···= pn = 1/n. Für das Denken unter Unsicherheit bedeutet dies, dass bei völligem Mangel an Wissen alle Welten gleich wahrscheinlich sind. Das heißt, die Verteilung ist gleichmäßig. Im Fall der beiden Variablen A und B wäre dies beispielsweise der Fall

$$P(A, B) = P(A, \neg B) = P(\neg A, B) = P(\neg A, \neg B) = 1/4,$$

woraus P (A) = P (B) = 1/2 und P (B|A) = 1/2 folgen. Das Ergebnis für den zweidimensionalen Fall ist in Abb. 7.2 auf Seite 125 zu sehen , da die markierte Bedingung genau die Normalisierungsbedingung ist. Wir sehen, dass das Maximum der Entropie genau bei (1/2, 1/2) auf der Linie liegt.

Sobald der Wert einer Bedingung von dem aus der Gleichverteilung abgeleiteten Wert abweicht, verschieben sich die Wahrscheinlichkeiten der Welten. Wir zeigen dies in einem weiteren Beispiel. Bei den gleichen Beschreibungen wie oben gehen wir nur davon aus

$$P(B|A) = ÿ$$

ist bekannt. Somit ist P (A, B) = P (B|A)P (A) = ÿP (A), woraus p1 = ÿ(p1 + p2) folgt und wir die beiden Einschränkungen ableiten

$$ÿp2 + (ÿ ÿ 1)p1 = 0,$$

$$p1 + p2 + p3 + p4 ÿ 1 = 0.$$

---

3Der Leser kann dieses Ergebnis durch Maximierung der Entropie unter der Normalisierungsbedingung berechnen (Übung 7.5 auf Seite 158).

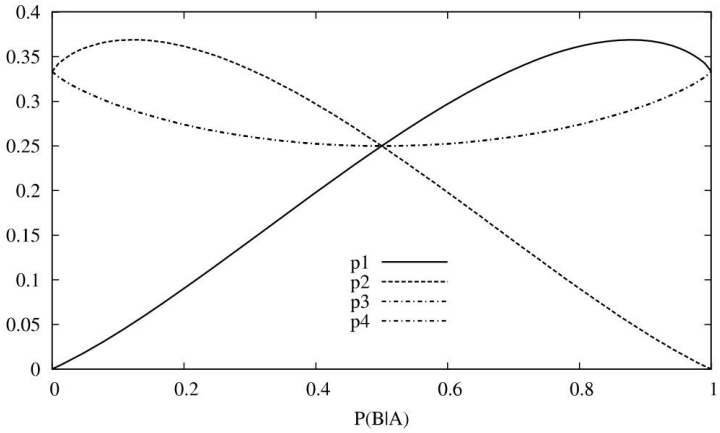



**Abb. 7.4 p1, p2, p3, p4 in Abhängigkeit von ÿ**

**Tabelle 7.1 Wahrheitstabelle für materielle Implikation und bedingte Wahrscheinlichkeit für Grenze der Aussagenlogik**

| ABA ÿ B | P (A) | P (B) | P (B\|A) |
|---|---|---|---|
| ttt 111 | | | |
| t ff 100 | | | |
| ft w fft 0 | 0 | 1 | *Nicht definiert* |
| | | 0 | *Nicht definiert* |

**Hier lassen sich die Lagrange-Gleichungen nicht mehr so einfach symbolisch lösen. Eine numerische Lösung der Lagrange-Gleichungen ergibt das in Abb. 7.4 dargestellte Bild, was zeigt, dass p3 = p4. Wir können dies bereits an den Einschränkungen erkennen, in denen p3 und p4 sind gleichgültig. Für P (B|A) = 1/2 erhalten wir die Gleichverteilung, die ist keine Überraschung. Dies bedeutet, dass die Einschränkung für diesen Wert keine Einschränkung der Verteilung mit sich bringt. Darüber hinaus können wir sehen, dass für kleine P (B|A) P (A, B) gil auch klein.**

## 7.2.3 Bedingte Wahrscheinlichkeit versus materielle Implikation

**Wir werden nun zeigen, dass die bedingte Wahrscheinlichkeit für die Modellierung des Denkens besser ist als das, was in der Logik als materielle Implikation bekannt ist (siehe hierzu auch [Ada75]).**
Zunächst betrachten wir die in Tabelle 7.1 **dargestellte Wahrheitstabelle , in der die bedingte Wahrscheinlichkeit und die materielle Implikation für die Extremfälle der Wahrscheinlichkeiten Null und Eins sind verglichen. In beiden Fällen mit falschen Prämissen (die intuitiv kritische Fälle sind) P (B|A) ist undefiniert, was Sinn macht.**

Nun fragen wir uns, welchen Wert P (B|A) annimmt, wenn beliebige Werte P (A) = ÿ und P (B) = ÿ gegeben sind und keine anderen Informationen bekannt sind. Auch hier maximieren wir die Entropie unter den gegebenen Randbedingungen. Wie oben haben wir eingestellt

p1 = P (A, B), p2 = P (A,¬B), p3 = P (¬A, B), p4 = P (¬A,¬B)

und als Nebenbedingungen erhalten

$$p1 + p2 = ÿ, \tag{7.10}$$

$$p1 + p3 = ÿ , \tag{7.11}$$

$$p1 + p2 + p3 + p4 = 1. \tag{7.12}$$

Damit rechnen wir mittels Entropiemaximierung (siehe Übung 7.8 auf Seite 159)

p1 = ÿÿ, p2 = ÿ(1 ÿ ÿ ), p3 = ÿ (1 ÿ ÿ), p4 = (1 ÿ ÿ)(1 ÿ ÿ ).

Aus p1 = ÿÿ folgt P (A, B) = P (A) · P (B), was bedeutet, dass A und B unabhängig sind. Da es keine Einschränkungen zwischen A und B gibt, führt das MaxEnt-Prinzip zur Unabhängigkeit dieser Variablen. Die rechte Hälfte von Tabelle 7.1 auf Seite 128 erleichtert das Verständnis. Aus der Definition

$$P(B|A) = \frac{P(A, B)}{P(A)}$$

Daraus folgt für den Fall P (A) = 0, das heißt, wenn die Prämisse nicht falsch ist, weil A und B unabhängig sind, dass P (B|A) = P (B). Für den Fall P (A) = 0 bleibt P (B|A) undefiniert.

### 7.2.4 MaxEnt-Systeme

Wie bereits erwähnt, kann die MaxEnt-Optimierung aufgrund der Nichtlinearität der Entropiefunktion für nichttriviale Probleme normalerweise nicht symbolisch durchgeführt werden. Daher wurden zwei Systeme zur numerischen Entropiemaximierung entwickelt. Das erste System, SPIRIT (Symmetrical Probabilistic Intensional Reasoning in Inference Net Works in Transition, www.xspirit.de), [RM96] wurde an der Fernuniversität Hagen gebaut.
Das zweite, PIT (Probability Induction Tool), wurde an der Technischen Universität München entwickelt [Sch96, ES99, SE00]. Wir werden PIT nun kurz vorstellen.

Das PIT-System verwendet die Methode der sequentiellen quadratischen Programmierung (SQP), um ein Extremum der Entropiefunktion unter den gegebenen Einschränkungen zu finden. Als Eingabe erwartet PIT Daten, die die Einschränkungen enthalten. Beispielsweise sind die Nebenbedingungen P (A) = ÿ und P (B|A) = ÿ aus Abschn. 7.2.1 haben die Form

```
var A{t,f}, B{t,f};

P([A=t]) = 0,6;
P([B=t] | [A=t]) = 0,3;

QP([B=t]);
QP([B=t] | [A=t]);
```

Da PIT eine numerische Berechnung durchführt, müssen wir explizite Wahrscheinlichkeitswerte eingeben. Die vorletzte Zeile enthält die Abfrage QP([B=t]). Das heisst dass P(B) der gewünschte Wert ist. Auf www.pit-systems.de unter „Beispiele“ finden Sie jetzt Fügen Sie diese Eingabe in eine leere Eingabeseite („Blank Page“) ein und starten Sie PIT. Als Ergebnis haben wir erhalten

| Nr. | Wahrheitswert | Wahrscheinlichkeitsabfrage |
|---|---|---|
| 1 | NICHT SPEZIFIZIERT | 3.800e–01 QP([B = t]); |
| 2 | NICHT SPEZIFIZIERT | 3.000e–01 QP([A = t]-|> [B = t]); |

und daraus P (B) = 0,38 und P (B|A) = 0,3 ablesen.

## 7.2.5 Das Tweety-Beispiel

Wir zeigen nun anhand des Tweety-Beispiels aus Abschn. 4.3, diese probabilistische Argumentation und insbesondere MaxEnt sind nicht monoton und modellieren das alltägliche Denken sehr Also. Wir modellieren die relevanten Regeln mit Wahrscheinlichkeiten wie folgt:

| | |
|---|---|
| P *(Vogel|Pinguin)* = 1 P | „Pinguine sind Vögel“ |
| (Fliegen|Vogel) ÿ [0,95, 1] | „(fast alle) Vögel können fliegen“ |
| P *(Fliegen|Pinguin)* = 0 | „Pinguine können nicht fliegen“ |

Die erste und dritte Regel stellen feste Vorhersagen dar, die sich auch leicht in der Logik formulieren lassen. Im zweiten bringen wir jedoch unser Wissen zum Ausdruck, dass fast alle Vögel können mithilfe eines Wahrscheinlichkeitsintervalls fliegen. Mit den PIT-Eingabedaten

```
Var-Pinguin{ja, nein}, Vogel{ja, nein}, fliegt{ja, nein};

P([Vogel=Ja] | [Pinguin=Ja]) = 1;
P([flies=yes] | [bird=yes]) IN [0.95,1];
P([flies=yes] | [pinguin=yes]) = 0;

QP([flies=yes]| [pinguin=yes]);
```

Wir bekommen die richtige Antwort zurück

| Nr. | Wahrheitswert | Wahrscheinlichkeitsabfrage |
|---|---|---|
| 1 | UNSPECIFIED | 0.000e+00 QP([penguin=yes]-|> [flies=yes]); |

mit der These, dass Pinguine nicht fliegen können.4 Die Erklärung hierfür ist sehr einfach. Mit P (Fliegen|Vogel) ÿ [0,95, 1] ist es möglich, dass es nicht fliegende Vögel gibt. Wenn dies würde die Regel durch P (Fliegen|Vogel) = 1 ersetzt , dann könnte PIT nichts tun und würde eine Fehlermeldung über inkonsistente Einschränkungen ausgeben.

An diesem Beispiel können wir leicht erkennen, dass Wahrscheinlichkeitsintervalle oft sehr hilfreich sind zur Modellierung unserer Unwissenheit über genaue Wahrscheinlichkeitswerte. Wir hätten ein machen können Eine noch unschärfere Formulierung der zweiten Regel im Sinne von „Normalerweise fliegen Vögel“ mit P (Fliegen|Vogel) ÿ (0,5, 1]. Die Verwendung des halboffenen Intervalls schließt den Wert 0,5 aus.

In [Pea88] wurde bereits gezeigt, dass dieses Beispiel auch ohne MaxEnt mit proba bilistischer Logik gelöst werden kann. In [Sch96] wird für eine Reihe anspruchsvoller Benchmarks für nichtmonotones Denken gezeigt, dass diese elegant gelöst werden können MaxEnt. Im folgenden Abschnitt stellen wir eine erfolgreiche praktische Anwendung vor MaxEnt in Form eines medizinischen Expertensystems.

## 7.3 LEXMED, ein Expertensystem zur Diagnose einer Blinddarmentzündung

Das medizinische Expertensystem LEXMED, das die MaxEnt-Methode nutzt, wurde an der Hochschule Ravensburg-Weingarten von Manfred entwickelt Schramm, Walter Rampf und der Autor, in Zusammenarbeit mit dem Weingarten 14-Nothelfer-Krankenhaus [SE00, Le999].5 Das Akronym LEXMED steht für *Lernendes Expertensystem für medizinische Diagnose.*

### 7.3.1 Appendizitis-Diagnose mit formalen Methoden

Die häufigste schwerwiegende Ursache für akute Bauchschmerzen [dD91] ist eine Blinddarmentzündung – eine Entzündung des Blinddarms, eines blinden Schlauchs, der mit dem Blinddarm verbunden ist. Sogar Heutzutage kann die Diagnose in vielen Fällen schwierig sein [OFY+95]. Zum Beispiel bis zu ca 20 % der entfernten Anhänge sind ohne pathologischen Befund, das heißt dass die Operationen unnötig waren. Ebenso gibt es regelmäßig Fälle, in denen Eine Blinddarmentzündung wird nicht als solche erkannt.

Bereits seit Anfang der 1970er Jahre gab es Versuche, die Diagnose einer Appendizitis zu automatisieren, mit dem Ziel, die Rate falscher Diagnosen zu reduzieren [dDLS+72, OPB94, OFY+95 ] . Besonders hervorzuheben ist das Expertensystem

4QP([penguin=yes]-|> [flies=yes]) ist eine alternative Form der PIT-Syntax für QP([flies=yes] | [pinguin=yes]).

5Das Projekt wurde vom Land Baden-Württemberg, der Krankenkasse AOK Baden-Württemberg, der Hochschule Ravensburg-Weingarten und finanziert das 14 Nothelfer Krankenhaus in Weingarten.

zur Diagnose akuter Bauchschmerzen, entwickelt von de Dombal in Großbritannien. Es wurde 1972 veröffentlicht, also deutlich früher als das berühmte System MYCIN.

Fast alle formalen diagnostischen Verfahren in der Medizin basieren bisher auf Scores. Punktesysteme sind denkbar einfach anzuwenden: Für jeden Wert eines Symptoms (zum Beispiel *Fieber* oder *Schmerzen im rechten Unterbauch)* notiert der Arzt eine bestimmte Anzahl von Punkten. Liegt die Summe der Punkte über einem bestimmten Wert (Schwellenwert), wird eine bestimmte Entscheidung empfohlen (z. B. Betrieb). Für n Symptome S1,...,Sn kann ein Score für Blinddarmentzündung formal beschrieben werden als

$$\text{Diagnose} = \begin{cases} \text{Appendizitis, wenn } w_1S_1 + \cdots + w_nS_n > \ddot{y}, \text{ negativ} \\ \text{anders.} \end{cases}$$

Bei Scores wird also eine lineare Kombination von Symptomwerten mit einem Schwellenwert ÿ verglichen. Die Gewichte der Symptome werden mithilfe statistischer Methoden aus Datenbanken extrahiert. Der Vorteil von Scores liegt in ihrer einfachen Anwendung. Die gewichtete Summe der Punkte kann leicht von Hand berechnet werden und für die Diagnose ist kein Computer erforderlich.

Aufgrund der Linearität dieser Methode sind die Ergebnisse zu schwach, um komplexe Zusammenhänge zu modellieren. Da der Beitrag wiSi eines Symptoms Si zum Score unabhängig von den anderen Symptomen berechnet wird, ist klar, dass Scoresysteme keinen „Kontext“ berücksichtigen können. Grundsätzlich können sie nicht zwischen Symptomkombinationen unterscheiden, beispielsweise können sie nicht zwischen der Anzahl der weißen Blutkörperchen eines alten Patienten und der eines jungen Patienten unterscheiden.

Für einen festen gegebenen Satz von Symptomen ist die bedingte Wahrscheinlichkeit viel aussagekräftiger als Scores für die Erstellung von Vorhersagen, da letztere die Abhängigkeiten zwischen verschiedenen Symptomen nicht beschreiben können. Wir können zeigen, dass Scores implizit davon ausgehen, dass alle Symptome unabhängig sind.

Bei der Verwendung von Partituren tritt noch ein weiteres Problem auf. Um eine gute Diagnosequalität zu erreichen, müssen hohe Anforderungen an die Datenbanken gestellt werden, die zur statistischen Ermittlung der Gewichte wi insbesondere müssen sie repräsentativ für die Patientengruppe im Einsatzgebiet des Diagnosesystems sein. Dies ist oft nur schwer, wenn nicht sogar unmöglich, zu gewährleisten. In solchen Fällen können Scores und andere statistische Methoden entweder nicht verwendet werden oder weisen eine hohe Fehlerquote auf.

## 7.3.2 Hybride probabilistische Wissensdatenbank

Komplexe Wahrscheinlichkeitsbeziehungen treten in der Medizin häufig auf. Mit LEXMED lassen sich diese Zusammenhänge gut modellieren und schnell berechnen. Wesentlich ist hierbei der Einsatz probabilistischer Aussagen, mit denen sich unsichere und unvollständige Informationen intuitiv und mathematisch fundiert ausdrücken und verarbeiten lassen. Als typische Abfrage an das Expertensystem kann folgende Frage dienen: „Wie hoch ist die Wahrscheinlichkeit einer Blinddarmentzündung, wenn es sich bei dem Patienten um einen 23-jährigen Mann mit Schmerzen im rechten Unterbauch und einer Anzahl weißer Blutkörperchen von 13.000 handelt?“ ?“ Als bedingte Wahrscheinlichkeit formuliert, unter Verwendung der in Tabelle 7.2 auf Seite 133 verwendeten Namen und Wertebereiche für die Symptome , lautet dies:

**Tabelle 7.2 Für die Abfrage in LEXMED** verwendete Symptome **und ihre Werte. Die Anzahl der Werte für Jedes Symptom ist in der mit # gekennzeichneten Spalte aufgeführt.**

| Symptom | Werte | # Kurz |
|---|---|---|
| Geschlecht | *Männlich, weiblich* | 2 *Sex2* |
| Alter | *0–5, 6–10, 11–15, 16–20, 21–25, 26–35, 36–45, 46–55, 56–65, 65–* | 10 *Alter10* |
| Schmerzen im 1. Quadrizeps. | *Ja, nein* | 2 *P1Q2* |
| Schmerzen im 2. Quadrizeps. | *Ja, nein* | 2 *P2Q2* |
| Schmerzen im dritten Quadranten. | *Ja, nein* | 2 *P3Q2* |
| Schmerzen im vierten Quadranten. | *Ja, nein* | 2 *P4Q2* |
| Bewachung | *Lokal, global, keine* | 3 *Gua3* |
| Rebound-Zärtlichkeit | *Ja, nein* | 2 *Reb2* |
| Schmerzen beim Klopfen | *Ja, nein* | 2 *Tippen Sie auf2* |
| Rektaler Schmerz | *Ja, nein* | 2 *RecP2* |
| Darmgeräusche | *Schwach, normal, erhöht, keine* | 4 *BogenS4* |
| Abnormaler Ultraschall | *Ja, nein* | 2 *Sono2* |
| Abnormaler Urinsedim. | *Ja, nein* | 2 *Urin2* |
| Temperatur (rektal) | *–37,3, 37,4–37,6, 37,7–38,0, 38,1–38,4, 38,5–38,9, 39,0–* | 6 *TRec6* |
| Leukozyten | *0–6k, 6k–8k, 8k–10k, 10k–12k, 12k–15k, 15k–20k, 20k–* | 7 *Leuko7* |
| Diagnose | *Entzündet, perforiert, negativ, anders* | 4 *Diag4* |

**P** *(Diag4 = entzündet* ÿ *Diag4 = perforiert* |

*Geschlecht2 = männlich* ÿ *Alter 10* ÿ 21–25 ÿ *Leuko7* ÿ 12.000–15.000).

**Durch die Verwendung probabilistischer Aussagen** ist **LEXMED in der Lage, Informationen zu nutzen aus nicht repräsentativen Datenbanken, da diese Informationen ergänzt werden können entsprechend aus anderen Quellen. Zugrunde** liegt **LEXMED eine Datenbank, die nur enthält Daten über Patienten, deren Blinddarm operativ entfernt wurde. Mit statistischen Methoden werden (ca. 400) Regeln generiert, die das in der Datenbank enthaltene Wissen in einer abstrahierten Form zusammenfassen[ES99]. Weil es keine Patienten gibt in dieser Datenbank, bei denen der Verdacht auf eine Blinddarmentzündung bestand, die jedoch negative Diagnosen hatten (d. h. keine Behandlung benötigten),6 gibt es keine Erkenntnisse über negative Patienten in der Datenbank. Daher muss Wissen aus anderen Quellen hinzugefügt werden. In LEXMED werden daher die aus der Datenbank erfassten Regeln durch ergänzt (ca. 100) Regeln von medizinischen Experten und der medizinischen Literatur. Das führt zu eine hybride probabilistische Datenbank, die aus Daten extrahiertes Wissen enthält sowie von Experten explizit formuliertes Wissen. Weil es beide Arten von Regeln gibt**

**6Diese negativen Diagnosen werden als „unspezifische Bauchschmerzen“ (NSAP)** *bezeichnet.*

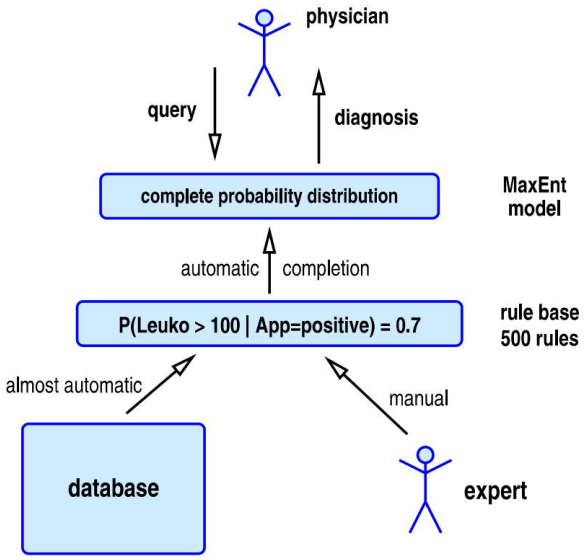



Abb. 7.5 Aus Daten und Expertenwissen werden probabilistische Regeln generiert, die in eine Regelbasis (Wissensbasis) integriert und schließlich mit der MaxEnt-Methode vervollständigt werden

Als bedingte Wahrscheinlichkeiten formuliert (siehe zum Beispiel (7.14) auf Seite 138), können sie leicht kombiniert werden, wie in Abb. 7.5 und mit weiteren Einzelheiten in Abb. 7.7 auf Seite 136 dargestellt.

LECMED berechnet die Wahrscheinlichkeiten verschiedener Diagnosen anhand der Wahrscheinlichkeitsverteilung aller relevanten Variablen (siehe Tabelle 7.2 auf Seite 133). Da alle 14 in LEXMED verwendeten Symptome und die Diagnosen als diskrete Variablen modelliert werden (sogar kontinuierliche Variablen wie der Leukozytenwert werden in Bereiche unterteilt), kann die Größe der Verteilung (d. h. die Größe des Ereignisraums) anhand der Tabelle bestimmt werden 7.2 auf Seite 133 als Produkt der Anzahl der Werte aller Symptome, d

$$10\,2 \cdot 10 \cdot 3 \cdot 4 \cdot 6 \cdot 7 \cdot 4 = 20\,643\,840$$

Elemente. Aufgrund der Normalisierungsbedingung aus Satz 7.1 hat er somit 20.643.839 unabhängige Werte. Jeder Regelsatz mit weniger als 20.643.839 Wahrscheinlichkeitswerten beschreibt diesen Ereignisraum potenziell nicht vollständig. Um jede beliebige Frage beantworten zu können, benötigt das Expertensystem eine vollständige Verteilung. Die Konstruktion einer so umfangreichen, konsistenten Verteilung mit statistischen Methoden ist sehr schwierig.7 Von einem menschlichen Experten alle 20.643.839 Werte für die Verteilung (anstelle der oben genannten 100 Regeln) zu verlangen, wäre praktisch unmöglich.

Hier kommt die MaxEnt-Methode ins Spiel. Die Verallgemeinerung von etwa 500 Regeln zu einem vollständigen Wahrscheinlichkeitsmodell erfolgt in LEXMED durch Maximierung der Entropie mit den 500 Regeln als Einschränkungen. Eine effiziente Kodierung der resultierenden MaxEnt-Verteilung führt zu Antwortzeiten für die Diagnose von etwa einer Sekunde.

7Die Aufgabe, aus einem Datensatz eine Funktion zu generieren, wird als maschinelles Lernen bezeichnet. Wir werden dies in Kap. ausführlich behandeln. 8.

| Personal Details | unknown | values | |
|---|---|---|---|
| Gender | ◉ | ○ male ○ female | ? |
| Age-group | ○ | ○ 0-5 ○ 6-10 ○ 11-15 ○ 16-20 ◉ 21-25 ○ 26-35 ○ 36-45 ○ 46-55 ○ 56-65 ○ 65- | ? |
| **Results of examination** | **not done** | **values** | |
| 1st quadrant | ◉ | ○ yes ○ no | ? |
| 2nd quadrant | ◉ | ○ yes ○ no | ? |
| 3rd quadrant | ○ | ◉ yes ○ no | ? |
| 4th quadrant | ◉ | ○ yes ○ no | ? |
| guarding | ○ | ◉ local ○ global ○ none | ? |
| rebound tenderness | ○ | ◉ yes ○ no | ? |
| pain on tapping | ○ | ◉ yes ○ no | ? |
| rectal pain | ◉ | ○ yes ○ no | ? |
| bowel sounds | ○ | ○ weak ◉ normal ○ increased ○ none | ? |
| abnormal ultrasound | ◉ | ○ yes ○ no | ? |
| abnormal urine sediment | ◉ | ○ yes ○ no | ? |
| temperature range (rectal) | ○ | ○ -37.3 ○ 37.4-37.6 ○ 37.7-38.0 ○ 38.1-38.4 ◉ 38.5-38.9 ○ 39.0- | ? |
| leucocyte count | ○ | ○ 0-6k ○ 6k-8k ○ 8k-10k ○ 10k-12k ◉ 12k-15k ○ 15k-20k ○ 20k- | ? |

| | Result of the PIT diagnosis | | | |
|---|---|---|---|---|
| Diagnosis | App. inflamed | App. perforated | Negative | Other |
| Probability | 0.70 | 0.17 | 0.06 | 0.07 |

**Abb. 7.6 Die LEXMED- Eingabemaske zur Eingabe der untersuchten Symptome und darunter die Ausgabe der resultierenden Diagnosewahrscheinlichkeiten**

### 7.3.3 Anwendung von LEXMED

**Die Nutzung von LEXMED ist einfach und selbsterklärend. Der Arzt besucht für eine Homepage von LEXMED** unter [8] **automatische Diagnose den Arzt www.lexmed.de. gibt die Ergebnisse seiner Untersuchung in das Eingabeformular in Abb. 7.6 ein. Nach ein bis zwei Sekunden erhält er die Wahrscheinlichkeiten für die vier verschiedenen Diagnosen sowie einen Behandlungsvorschlag (Abschn. 7.3.5). Fehlen bestimmte Untersuchungsergebnisse als Eingabe (zum Beispiel die Sonogramm-Ergebnisse), wählt der Arzt den Eintrag *nicht untersucht.* Natürlich ist die Sicherheit der Diagnose umso höher, je mehr Symptomwerte eingegeben werden.**

8Eine Version mit eingeschränkter Funktionalität ist ohne Passwort zugänglich.

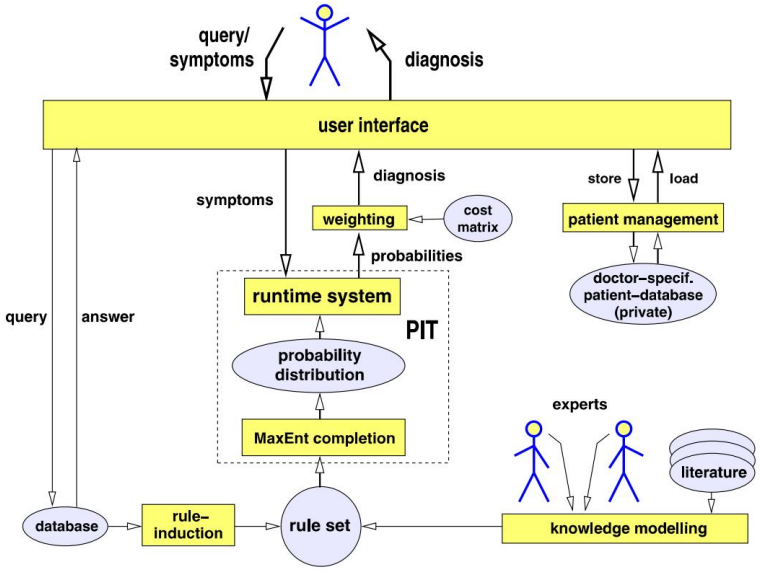



**Abb. 7.7 Regeln werden sowohl aus der Datenbank als auch aus Expertenwissen generiert. Daraus erstellt MaxEnt eine vollständige Wahrscheinlichkeitsverteilung. Für eine Benutzerabfrage wird die Wahrscheinlichkeit jeder möglichen Diagnose berechnet. Anhand der Kostenmatrix (siehe Abschn. 7.3.5) wird dann eine Entscheidung vorgeschlag**

**Jeder registrierte Benutzer hat Zugriff auf eine private Patientendatenbank, in der Eingabedaten archiviert werden können. Somit können Daten und Diagnosen früherer Patienten problemlos mit denen eines neuen Patienten verglichen werden.**

## 7.3.4 Funktion von LEXMED

**Wissen wird mithilfe probabilistischer Aussagen formalisiert. Zum Beispiel der Vorschlag**

$$P(Leuko7 > 20000 | Diag4 = entzündet) = 0{,}09$$

**gibt eine Häufigkeit von 9 % für einen Leukozytenwert von mehr als 20.000 bei einer Blinddarmentzündung an.[9]**

**Lernen von Regeln durch statistische Induktion Die Rohdaten in der** Datenbank **von LEXMED enthalten 54 verschiedene (anonymisierte) Werte für 14.646 Patienten. Wie bereits erwähnt, sind in dieser Datenbank nur Patienten enthalten, deren Blinddarm operativ entfernt wurde. Von den 54 in der Datenbank verwendeten Attributen sind nach einer statistischen Analyse die 14 Symptome in Tabelle 7.2 auf Seite 133** aufgeführt

[9] **Anstelle einzelner Zahlenwerte können hier auch Intervalle verwendet werden (z. B. [0,06, 0,12]).**

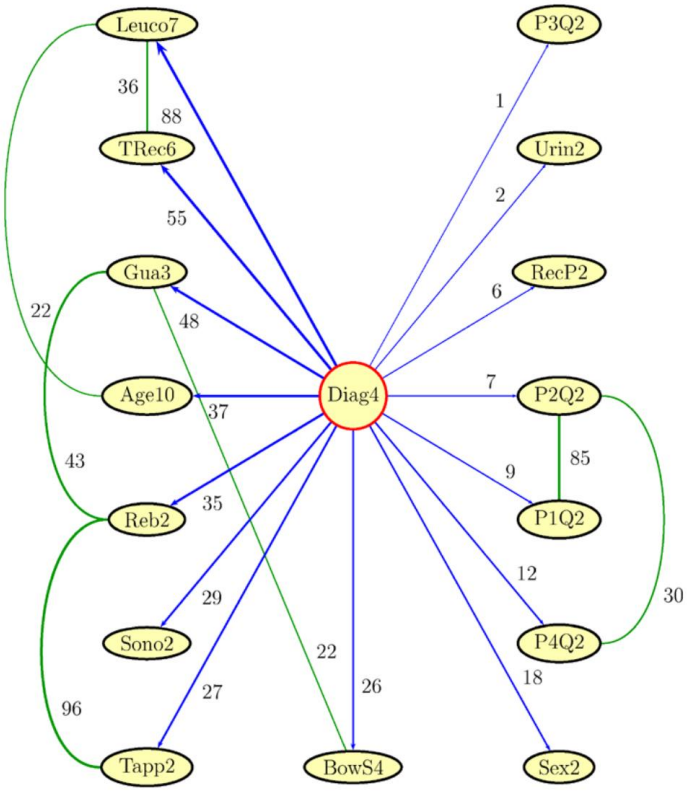



**Abb. 7.8 Aus der Datenbank berechneter Abhängigkeitsgraph**

**wurden benutzt. Aus dieser Datenbank werden nun in zwei Schritten die Regeln erstellt. Im ersten Schritt wird die Abhängigkeitsstruktur der Symptome ermittelt. Im zweiten Schritt wird diese Struktur mit den entsprechenden Wahrscheinlichkeitsregeln gefüllt.10**

**Bestimmung des Abhängigkeitsgraphen Der Graph in Abb. 7.8 enthält für jede Variable (das Symptom und die Diagnose) einen Knoten und gerichtete Kanten, die verschiedene Knoten verbinden. Die Dicke der Kanten zwischen den Variablen stellt ein Maß für die statistische Abhängigkeit bzw. Korrelation der Variablen dar. Die Korrelation zweier unabhängiger Variablen ist gleich Null. Die Paarkorrelation für jedes der 14 Symptome mit *Diag4* wurde berechnet und in der Grafik aufgeführt. Darüber hinaus wurden alle dreifachen Korrelationen zwischen der Diagnose und zwei Symptomen berechnet. Von diesen wurden nur die stärksten Werte als zusätzliche Kanten zwischen den beiden beteiligten Symptomen eingezeichnet.**

10Für eine systematische Einführung in maschinelles Lernen verweisen wir den Leser auf Kap. 8.

```
 1 P([Leuco7=0-6k]                | [Diag4=negativ] * [Alter10=16-20]) = [0,132,0,156];
 2 P([Leuco7=6-8k]                | [Diag4=negativ] * [Alter10=16-20]) = [0,257,0,281];
 3 P([Leuco7=8-10k] | [Diag4=negativ] * [Alter10=16-20]) = [0,250,0,274];
 4 P([Leuco7=10-12k] | [Diag4=negativ] * [Alter10=16-20]) = [0,159,0,183];
 5 P([Leuco7=12-15k] | [Diag4=negativ] * [Alter10=16-20]) = [0,087,0,112];
 6 P([Leuco7=15-20k] | [Diag4=negativ] * [Alter10=16-20]) = [0,032,0,056];
 7 P([Leuco7=20k-]                | [Diag4=negativ] * [Alter10=16-20]) = [0,000,0,023];
 8 P([Leuco7=0-6k]                | [Diag4=negativ] * [Alter10=21-25]) = [0,132,0,172];
 9 P([Leuco7=6-8k]                | [Diag4=negativ] * [Alter10=21-25]) = [0,227,0,266];
10 P([Leuco7=8-10k] | [Diag4=negativ] * [Alter10=21-25]) = [0,211,0,250];
11 P([Leuco7=10-12k] | [Diag4=negativ] * [Alter10=21-25]) = [0,166,0,205];
12 P([Leuco7=12-15k] | [Diag4=negativ] * [Alter10=21-25]) = [0,081,0,120];
13 P([Leuco7=15-20k] | [Diag4=negativ] * [Alter10=21-25]) = [0,041,0,081];
14 P([Leuco7=20k-]                | [Diag4=negativ] * [Alter10=21-25]) = [0,004,0,043];
```

**Abb. 7.9 Einige der LEXMED- Regeln mit Wahrscheinlichkeitsintervallen. „*“ steht hier für „ÿ“.**

**Abschätzen der Regelwahrscheinlichkeiten Die Struktur des Abhängigkeitsgraphen beschreibt die Struktur der erlernten Regeln.11 Die Regeln sind dabei unterschiedlich komplex: Es gibt Regeln, die nur die Verteilung der möglichen Diagnosen beschreiben (A-priori-Regeln, zum Beispiel (7.13)), Regeln, die die Abhängigkeit zwischen beschreiben die Diagnose und ein Symptom (Regeln mit einfachen Bedingungen, zum Beispiel (7.14)), und schließlich Regeln, die die Abhängigkeit zwischen der Diagnose und zwei Symptomen beschreiben, wie in Abb. 7.9 in PIT- Syntax dargestellt.**

$$P(Diag4 = entzündet) = 0{,}40, \quad P \tag{7.13}$$

$$(Sono2 = ja \mid Diag4 = entzündet) = 0{,}43, \tag{7.14}$$

$$P(P4Q2 = ja \mid Diag4 = entzündet \text{ ÿ } P2Q2 = ja) = 0{,}61. \tag{7.15}$$

**Um die Kontextabhängigkeit des gespeicherten Wissens möglichst gering zu halten, Alle Regeln enthalten die Diagnose in ihren Bedingungen und nicht als Schlussfolgerungen. Das ist ganz ähnlich dem Aufbau vieler medizinischer Bücher mit Formulierungen des Art „Bei einer Blinddarmentzündung sehen wir normalerweise . . . “. Wie zuvor in Beispiel 7.6 gezeigt Auf Seite 121 stellt dies jedoch kein Problem dar, da die Bayesian-Methode verwendet wird Mithilfe der Formel bringt LECMED diese Regeln automatisch in die richtige Form.**

**Die numerischen Werte für diese Regeln werden durch Zählen ihrer Häufigkeit geschätzt die Datenbank. Beispielsweise ergibt sich der Wert in (7.14) durch Zählen und Rechnen**

$$\frac{|Diag4 = entzündet \text{ ÿ } Sono2 = ja|}{|Diag4 = entzündet|} = 0{,}43.$$

**11Der Unterschied zu einem Bayes'schen Netzwerk besteht beispielsweise darin, dass die Regeln ausgestattet sind mit Wahrscheinlichkeitsintervallen und dass erst nach Anwendung des Prinzips der maximalen Entropie ein Unikat vorliegt Wahrscheinlichkeitsmodell erstellt.**

### Expertenregeln

Denn die Appendizitis-Datenbank enthält nur Patienten, die sich einer Blinddarmentzündung unterzogen haben
Operation, Regeln für unspezifische Bauchschmerzen *(NSAP)* erhalten ihre Werte von
Vorschläge medizinischer Experten. Die Erfahrungen in LEXMED bestätigen, dass die Wahrscheinlichkeitsregeln
leicht zu lesen sind und direkt in natürliche Sprache übersetzt werden können.
Aussagen medizinischer Experten zu Häufigkeitszusammenhängen bestimmter Symptome
und die Diagnose, sei es aus der Literatur oder als Ergebnis eines Interviews, kann
können daher mit geringem Aufwand in die Regelbasis integriert werden. Zur Modellierung der Unsicherheit
von Expertenwissen hat sich die Verwendung von Wahrscheinlichkeitsintervallen bewährt.
Das Expertenwissen wurde in erster Linie von den beteiligten Chirurgen Dr.
Rampf und Dr. Hontschik und ihre Veröffentlichungen [Hon94].

Sobald die Expertenregeln erstellt wurden, ist die Regelbasis fertig. Dann wird das vollständige
Wahrscheinlichkeitsmodell mit der Methode der maximalen Entropie berechnet
PIT-System.

### Diagnoseabfragen

Mithilfe seines effizient hinterlegten Wahrscheinlichkeitsmodells berechnet LECMED die Wahrscheinlichkeiten
für die vier möglichen Diagnosen innerhalb weniger Sekunden. Wir gehen zum Beispiel davon aus, dass
folgende Ausgabe:

| | Ergebnisse der PIT-Diagnose | | |
|---|---|---|---|
| Diagnose | Blinddarm entzündet Blinddarm perforiert 0,24 0,16 | Negativ | Andere |
| Wahrscheinlichkeit | | 0,57 | 0,03 |

Auf der Grundlage dieser vier Wahrscheinlichkeitswerte muss eine Entscheidung getroffen werden, einen dieser Werte zu verfolgen
Die vier Behandlungen: Operation, Notoperation, stationäre Beobachtung oder ambulante Beobachtung.12
Während die Wahrscheinlichkeit für eine negative Diagnose in diesem Fall hoch ist
Da die Entscheidung überwiegt, ist es keine gute Entscheidung, den Patienten als gesund nach Hause zu schicken.
Wir können das deutlich sehen, auch wenn die Wahrscheinlichkeiten der Diagnosen hoch waren
berechnet, die Diagnose ist noch nicht abgeschlossen.

Die Aufgabe besteht nun vielmehr darin, aus diesen Wahrscheinlichkeiten eine optimale Entscheidung abzuleiten. Zu
Zu diesem Zweck kann der Nutzer von LEXYMED eine empfohlene Entscheidung berechnen lassen.

## 7.3.5 Risikomanagement mithilfe der Kostenmatrix

Wie können nun die berechneten Wahrscheinlichkeiten optimal in Entscheidungen umgesetzt werden?
Ein naiver Algorithmus würde jeder Diagnose eine Entscheidung zuweisen und letztendlich auswählen
die Entscheidung, die der höchsten Wahrscheinlichkeit entspricht. Gehen Sie davon aus, dass die berechnete
Die Wahrscheinlichkeiten für die Diagnose ***Blinddarmentzündung*** (entzündet oder perforiert) liegen bei 0,40 und bei 0,55
für die Diagnose *„negativ“* und 0,05 für die Diagnose *„andere“*. Ein naiver Algorithmus würde es tun
Wählen Sie nun die (zu riskante) Entscheidung „keine Operation“, da diese der Diagnose mit der höheren
Wahrscheinlichkeit entspricht. Eine bessere Methode besteht darin, die Kosten zu vergleichen

12Ambulante Beobachtung bedeutet, dass der Patient zu Hause entlassen wird.

Tabelle 7.3 Die Kostenmatrix von LEXMED zusammen mit den berechneten Diagnosewahrscheinlichkeiten eines Patienten

| Therapie | Wahrscheinlichkeit verschiedener Diagnosen | | | | |
|---|---|---|---|---|---|
| | *Entzündet* | *Perforiert* | *Negativ* | *Andere* | |
| | 0,25 | 0,15 | 0,55 | 0,05 | |
| Betrieb | 0 | 500 | 5800 | 6000 | 3565 |
| Notoperation | 500 | 0 | 6300 | 6500 | 3915 |
| Ambulante Beobachtung. | 12000 | 150000 | 0 | 16500 | 26325 |
| Andere | 3000 | 5000 | 1300 | 0 | 2215 |
| Stationärer Beobachter | 3500 | 7000 | 400 | 600 | 2175 |

der möglichen Fehler, die bei jeder Entscheidung auftreten können. Der Fehler wird in der quantifiziert Form von „(hypothetischen) zusätzlichen Kosten der aktuellen Entscheidung im Vergleich zum Optimum“. Die angegebenen Werte beinhalten die Kosten für das Krankenhaus, für die Versicherung, dem Patienten (z. B. Risiko postoperativer Komplikationen) und anderen Parteien (z. B. Abwesenheit vom Arbeitsplatz) unter Berücksichtigung langfristiger Folgen.
Diese Kosten sind in Tabelle 7.3 aufgeführt.

Die Eingaben werden abschließend für jede Entscheidung gemittelt, also beim Treffen summiert Berücksichtigung ihrer Frequenzen. Diese sind in der letzten Spalte der Tabelle 7.3 aufgeführt. Abschließend wird die Entscheidung mit den geringsten durchschnittlichen Fehlerkosten vorgeschlagen. In Tabelle 7.3 Die Matrix wird zusammen mit dem für einen Patienten berechneten Wahrscheinlichkeitsvektor angegeben (in in diesem Fall: (0,25, 0,15, 0,55, 0,05)). Die letzte Spalte der Tabelle enthält das Ergebnis von die Berechnungen der durchschnittlich erwarteten Kosten der Fehler. Der Wert der *Operation* in der ersten Zeile errechnet sich somit zu 0,25 ·0+0,15 ·500+0,55 ·5800+0,05 ·6000 = 3565, ein gewichteter Durchschnitt aller Kosten. Die optimalen Entscheidungen werden mit (Zusatz-)Kosten von 0 eingetragen. Das System entscheidet sich für die Behandlung mit dem minimalen Durchschnitt kosten. Es handelt sich somit um ein Beispiel für einen kostenorientierten Agenten.

## Kostenmatrix im binären Fall

Um die Kostenmatrix und das Risikomanagement besser zu verstehen, werden wir nun die einschränken LEXMED- System zur zweiwertigen Entscheidung zwischen der Diagnose *Blinddarmentzündung* und *NSAP.* Als mögliche Behandlungen sind nur *Operation* mit Wahrscheinlichkeit p1 und *ambulante Behandlung* möglich Es kann *eine Beobachtung* mit der Wahrscheinlichkeit p2 gewählt werden. Die Kostenmatrix ist somit eine 2 × 2 Matrix des Formulars

$$\begin{matrix} 0 & k2 \\ k1 & 0 \end{matrix} .$$

Die beiden Nullen in der Diagonale stehen für die jeweils richtige *Entscheidungsoperation* der *Blinddarmentzündung* und *ambulante Beobachtung* auf *NSAP.* Der Parameter k2 steht für die zu erwartenden Kosten, die entstehen, wenn ein Patient keinen entzündeten Blinddarm hat operiert. Dieser Fehler wird als falsch positiv bezeichnet . Andererseits die Entscheidung

*Eine ambulante Beobachtung* ist bei einer Blinddarmentzündung *falsch negativ.* Der Wahrscheinlichkeitsvektor (p1, p2)T wird von dieser Matrix multipliziert und wir erhalten den Vektor

$$(2 \text{ re. } 2 \text{ p., } 1 \text{ re. p. } 1)^T$$

mit den durchschnittlichen Mehrkosten für die beiden möglichen Behandlungen. Da die Entscheidung nur das Verhältnis der beiden Komponenten berücksichtigt, kann der Vektor mit einem beliebigen Skalarfaktor multipliziert werden. Wir wählen 1/k1 und erhalten ((k2/k1)p2, p1). Daher ist hier nur die Beziehung k = k2/k1 relevant. Das gleiche Ergebnis erhält man mit der einfacheren Kostenmatrix

$$\begin{matrix} 0 & k \\ 1 & 0 \end{matrix},$$

die nur die Variable k enthält. Dieser Parameter ist sehr wichtig, da er das Risikomanagement bestimmt. Durch die Änderung von k können wir den „Arbeitspunkt“ des Diagnosesystems anpassen. Für k ÿ ÿ befindet sich das System in einer äußerst riskanten Umgebung, da kein Patient jemals operiert wird, mit der Konsequenz, dass es keine falsch-positiven Klassifizierungen, aber viele falsch-negative Klassifizierungen liefert. Bei k = 0 sind die Verhältnisse genau umgekehrt und alle Patienten werden operiert.

### 7.3.6 Leistung

LEUXMED ist für den Einsatz in einer Arztpraxis oder einem Krankenwagen bestimmt. Voraussetzung für die Anwendung von LEXYMED sind akute Bauchschmerzen über mehrere Stunden (jedoch weniger als fünf Tage). Darüber hinaus ist LEXMED (derzeit) auf Blinddarmentzündung spezialisiert, was bedeutet, dass das System für andere Krankheiten nur sehr wenige In

Im Rahmen einer prospektiven Studie wurde eine repräsentative Datenbank mit 185 Fällen im 14. Nothelfer Krankenhaus erstellt. Es enthält die Patienten des Krankenhauses, die nach mehreren Stunden akuter Bauchschmerzen und Verdacht auf Blinddarmentzündung in die Klinik kamen. Von diesen Patienten werden die Symptome und die Diagnose (im Falle einer Operation anhand einer Gewebeprobe überprüft) notiert.

Konnten die Patienten nach mehrstündigem oder 1–2-tägigem Aufenthalt beschwerdefrei nach Hause (ohne Operation) entlassen werden, wurde anschließend telefonisch erfragt, ob der Patient beschwerdefrei blieb oder ob eine positive Diagnose vorliegt anschließende Behandlung.

Um die Darstellung zu vereinfachen und einen besseren Vergleich mit ähnlichen Studien zu ermöglichen, wurde LEXMED auf die zweiwertige Unterscheidung zwischen *Appendizitis* und *NSAP beschränkt,* wie in Abschn. 7.3.5. Jetzt wird k zwischen Null und Unendlich variiert und für jeden Wert von k werden die *Sensitivität* und *Spezifität* anhand der Testdaten gemessen. Empfindlichkeitsmaße

$$\text{P (positiv klassifiziert|positiv)} = \left| \frac{\text{|positiv und als positiv eingestuft|}}{\text{positiv|}} \right.,$$

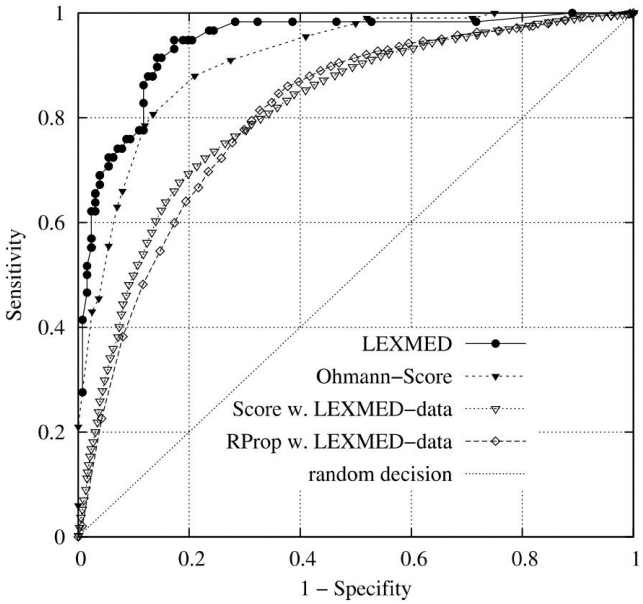



**Abb. 7.10 ROC-Kurve von LEXMED im Vergleich zum Ohmann-Score und zwei weiteren Modellen**

**das heißt, der relative Anteil der positiven Fälle, die korrekt identifiziert wurden. Sie gibt an, wie empfindlich das Diagnosesystem ist. Spezifität hingegen misst**

$$\text{P (klassifiziert negativ|negativ)} = \frac{|\text{negativ und als negativ eingestuft}|}{|\text{negativ}|},$$

**das heißt, der relative Anteil der korrekt identifizierten Negativfälle.**

**Die Ergebnisse der Sensitivität und Spezifität geben wir in Abb. 7.10 für 0 ÿ k < ÿ. Diese Kurve wird als *ROC-Kurve* oder Receiver Operating Characteristic bezeichnet. Bevor wir zur Analyse der Qualität von LEXMED kommen, noch ein paar Worte zur Bedeutung der ROC-Kurve. Zur Orientierung ist die Linie eingezeichnet, die das Diagramm diagonal ha Alle Punkte auf dieser Linie entsprechen einer Zufallsentscheidung. Beispielsweise entspricht der Punkt (0,2, 0,2) einer Spezifität von 0,8 bei einer Sensitivität von 0,2. Wir können dies ganz einfach erreichen, indem wir einen neuen Fall, ohne ihn anzusehen, mit Wahrscheinlichkeiten von 0,2 für positiv und 0,8 für negativ klassifizieren. Jedes wissensbasierte Diagnosesystem muss daher einen ROC generieren, der deutlich über der Diagonale**

**Interessant sind auch die Extremwerte in der ROC-Kurve. Im Punkt (0, 0) schneiden sich alle drei Kurven. Das entsprechende Diagnosesystem würde alle Fälle als negativ klassifizieren. Der andere Extremwert (1, 1) entspricht einem System, das entscheiden würde, die Operation bei jedem Patienten durchzuführen und hat daher eine Sensitivität von 1. Wir könnten die ROC-Kurve als charakteristische Kurve für zweiwertige Diagnosesysteme bezeichnen. Das ideale Diagnosesystem hätte eine Kennlinie, die nur aus dem Punkt (0, 1) besteht und somit 100 % Spezifität und 100 % Sensitivität aufweist.**

Lassen Sie uns nun die ROC-Kurve analysieren. Bei einer Sensitivität von 88 % erreicht LEXMED eine Spezifität von 87 % (k = 0,6). Zum Vergleich wird der Ohmann-Score, ein etablierter, bekannter Score für Appendizitis, angegeben [OMYL96, ZSR+99]. Da LEXMED fast überall über oder links vom Ohmann-Score liegt, ist die durchschnittliche Diagnosequalität deutlich besser. Dies ist nicht überraschend, da die Ergebnisse einfach zu schwach sind, um interessante Aussagen zu modellieren. In Abschn. 8.6 und in Übung 8.16 auf Seite 219 werden wir zeigen, dass Scores äquivalent zum Spezialfall des naiven Bayes sind, also der Annahme, dass alle Symptome paarweise unabhängig sind, wenn die Diagnose bekannt ist. Beim Vergleich von LEXMED mit Scores ist jedoch zu erwähnen, dass für den Ohmann-Score eine statistisch repräsentative Datenbank verwendet wurde, für LEXMED jedoch eine nicht repräsentative, mit Expertenwissen angereicherte Datenbank. Um einen Eindruck von der Qualität der LEXMED- Daten im Vergleich zu den Ohmann-Daten zu bekommen, wurde ein linearer Score nach der Methode der kleinsten Quadrate berechnet (siehe Abschn. 9.4.1), der auch zum Vergleich herangezogen wird. Darüber hinaus wurde ein neuronales Netzwerk auf den LEXMED- Daten mit dem RProp-Algorithmus trainiert (siehe Abschn. 9.5). Die Stärke der Kombination von Daten und Expertenwissen zeigt sich deutlich im Unterschied zwischen der LEXMED- Kurve und den Kurven des Score-Systems und

## 7.3.7 Anwendungsbereiche und Erfahrungen

LEXYMED sollte nicht das Urteil eines erfahrenen Chirurgen ersetzen. Da im klinischen Umfeld jedoch nicht immer ein Facharzt zur Verfügung steht, bietet eine LEXMED- Anfrage eine fundierte Zweitmeinung. Besonders interessant und lohnenswert ist der Einsatz des Systems im klinischen Rettungswagen und bei niedergelassenen Ärzten.

Die Lernfähigkeit von LEXMED, die die Berücksichtigung weiterer Symptome, weiterer Patientendaten und weiterer Regeln ermöglicht, eröffnet auch in der Klinik neue Möglichkeiten. Für besonders seltene und schwierig zu diagnostizierende Gruppen, beispielsweise Kinder unter sechs Jahren, kann LEXYMED Daten von Kinderärzten oder anderen speziellen Datenbanken nutzen, um auch erfahrene Chirurgen zu unterstützen.

Neben dem direkten Einsatz in der Diagnostik unterstützt LEXMED auch Maßnahmen zur Qualitätssicherung. Beispielsweise können Versicherungen die Diagnosequalität von Krankenhäusern mit der von Expertensystemen vergleichen. Durch die Weiterentwicklung der in LEXMED erstellten Kostenmatrix (mit Zustimmung von Ärzten, Versicherungen und Patienten) wird die Qualität von Arztdiagnosen, Computerdiagnosen und anderen medizinischen Einrichtungen besser vergleichbar.

LEXYMED hat einen neuen Weg zum Aufbau automatischer Diagnosesysteme aufgezeigt. Mithilfe der Sprache der Wahrscheinlichkeitstheorie und des MaxEnt-Algorithmus wird induktiv statistisch abgeleitetes Wissen mit Wissen von Experten und aus der Literatur kombiniert. Der auf probabilistischen Modellen basierende Ansatz ist theoretisch elegant, allgemein anwendbar und hat in einer kleinen Studie zu sehr guten Ergebnissen geführt.

Seit 1999 ist LEXMED im Nothelfer-Krankenhaus 14 in Weingarten im praktischen Einsatz und hat sich dort sehr gut bewährt. Es ist auch unter www.lexmed.de erhältlich, natürlich ohne Gewähr. Die Diagnosequalität ist vergleichbar mit der eines erfahrenen Chirurgen und damit besser als die eines durchschnittlichen Allgemeinmediziners oder eines unerfahrenen Arztes in der Klinik.

Trotz dieses Erfolgs zeigt sich, dass es sehr schwierig ist, ein solches System im deutschen Medizinsystem kommerziell zu vermarkten. Ein Grund dafür ist, dass es keinen freien Markt gibt, der durch seine Selektionsmechanismen eine bessere Qualität (hier bessere Diagnosen) fördert. Darüber hinaus ist in der Medizin auch im Jahr 2010 noch nicht die Zeit für einen breiten Einsatz intelligenter Techniken gekommen. Eine Ursache hierfür könnten konservative Lehren in dieser Hinsicht an deutschen medizinischen Fakultäten sein.

Ein weiteres Thema ist der Wunsch vieler Patienten nach persönlicher Beratung und Betreuung durch den Arzt sowie die Befürchtung, dass der Patient mit der Einführung von Expertensystemen nur noch mit der Maschine kommuniziert. Diese Angst ist jedoch völlig unbegründet. Auch auf lange Sicht können medizinische Expertensysteme den Arzt nicht ersetzen Sie können jedoch ebenso wie Laserchirurgie und Magnetresonanztomographie für alle Beteiligten vorteilhaft eingesetzt werden. Seit dem ersten medizinischen Computer-Diagnosesystem von de Dombal im Jahr 1972 sind fast 40 Jahre vergangen. Es bleibt abzuwarten, ob die Medizin noch weitere 40 Jahre warten wird, bis die Computerdiagnostik ein etabliertes medizinisches Instrument wird.

## 7.4 Argumentation mit Bayes'schen Netzwerken

Auf ein Problem des Wahrscheinlichkeitsschlusses in der Praxis wurde bereits in Abschn. 7.1. Werden d Variablen X1,...,Xd mit jeweils n Werten verwendet, dann die zugehörigen d Die Wahrscheinlichkeitsverteilung Gesamtwerte. Dies bedeutet, dass im schlimmsten Fall die benötigt n Speicher und die Rechenzeit zur Bestimmung der angegebenen Wahrscheinlichkeiten wächst exponentiell mit der Anzahl der Variablen.

In der Praxis sind die Anwendungen meist sehr strukturiert und die Verteilung weist viele Redundanzen auf. Dies bedeutet, dass er mit geeigneten Methoden deutlich reduziert werden kann. Der Einsatz von Bayes'schen Netzwerken hat sich hier bewährt und gehört zu den KI-Techniken, die in der Praxis erfolgreich eingesetzt werden. Bayesianische Netze nutzen das Wissen über die Unabhängigkeit von Variablen, um das Modell zu verein

### 7.4.1 Unabhängige Variablen

Im einfachsten Fall sind alle Variablen paarweise unabhängig und das ist auch der Fall

$$P(X1,...,Xd) = P(X1) \cdot P(X2) \cdots P(Xd).$$

Alle Einträge in der Verteilung können somit aus den d-Werten P (X1), ... berechnet werden, . . . P (Xd ). Interessante Anwendungen können jedoch meist nicht modelliert werden, weil bedingte Wahrscheinlichkeiten trivial werden.13 Wegen

$$P(A|B) = \frac{P(A, B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A)$$

13Bei der naiven Bayes-Methode wird die Unabhängigkeit aller Attribute vorausgesetzt und diese Methode wurde erfolgreich auf die Textklassifizierung angewendet (siehe Abschn. 8.6).

Alle bedingten Wahrscheinlichkeiten werden auf die A-priori-Wahrscheinlichkeiten reduziert. Interessanter wird die Situation, wenn nur ein Teil der Variablen unter bestimmten Bedingungen unabhängig oder abhängig ist. Für die Argumentation in der KI sind die Abhängigkeiten zwischen Variablen wichtig und müssen genutzt werden.

Wir möchten die Argumentation mit Bayes'schen Netzwerken anhand eines einfachen und sehr anschaulichen Beispiels von J. Pearl [Pea88] skizzieren, das durch [RN10] bekannt wurde und mittlerweile zum KI-Grundwissen gehört.

*Beispiel 7.7* (Alarm-Beispiel) Bob, der Single ist, hat in seinem Haus eine Alarmanlage zum Schutz vor Einbrechern installieren lassen. Bob kann den Alarm nicht hören, wenn er im Büro arbeitet. Deshalb hat er seine beiden Nachbarn, John im Haus nebenan links und Mary im Haus rechts, gebeten, ihn in seinem Büro anzurufen, wenn sie seinen Alarm hören. Nach ein paar Jahren weiß Bob, wie zuverlässig John und Mary sind und modelliert ihr Anrufverhalten mithilfe der bedingten Wahrscheinlichkeit wie folgt.14

$$P(J \mid Al) = 0{,}90 \quad P(M \mid Al) = 0{,}70, \quad P(J \mid \neg Al) = 0{,}05 \quad P(M \mid \neg Al) = 0{,}01.$$

Da Mary schwerhörig ist, hört sie den Alarm häufiger als John.
Allerdings verwechselt John manchmal die Alarmanlage in Bobs Haus mit der Alarmanlage in anderen Häusern. Der Alarm wird durch einen Einbruch ausgelöst, kann aber auch durch ein (schwaches) Erdbeben ausgelöst werden, was zu einem Fehlalarm führen kann, da Bob nur in seinem Büro über Einbrüche Bescheid wissen möchte. Diese Beziehungen werden modelliert von

$$P(Al \mid Bur, Ohr) = 0{,}95,$$

$$P(Al \mid Bur, \neg Ear) = 0{,}94,$$

$$P(Al \mid \neg Bur, Ohr) = 0{,}29,$$

$$P(Al \mid \neg Bur, \neg Ear) = 0{,}001,$$

sowie die A-priori-Wahrscheinlichkeiten $P(Bur) = 0{,}001$ und $P(Ear) = 0{,}002$. Diese beiden Variablen sind unabhängig voneinander, da Erdbebenpläne nicht auf der Grundlage der Gewohnheiten von Einbrechern erstellt werden und es umgekehrt keine Möglichkeit gibt, Erdbeben vorherzusagen, sodass Einbrecher nicht die Möglichkeit haben, ihren Zeitplan entsprechend festzule

Nun werden Abfragen gegen diese Wissensdatenbank durchgeführt. Bob könnte beispielsweise an $P(Bur \mid J \text{ ÿ } M)$, $P(J \mid Bur)$ oder $P(M \mid Bur)$ interessiert sein. Das heißt, er möchte wissen, wie empfindlich die Variablen J und M auf eine Einbruchsmeldung reagieren.

14Die binären Variablen J und M stehen für die beiden Ereignisse „John ruft“ bzw. „Maria ruft“, *Al* für „Alarmsirene ertönt“, *Bur* für „Einbruch“ und *Ohr* für „Erdbeben“.

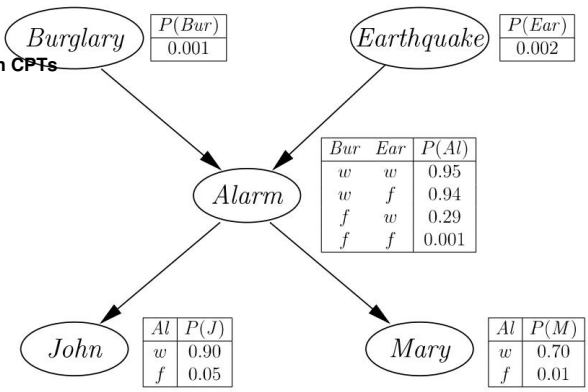



**Abb. 7.11 Bayesianisches Netzwerk für das Alarmbeispiel mit den zugehörigen CPTs**

## 7.4.2 Grafische Darstellung von Wissen als Bayesianisches Netzwerk

**Wir können die praktische Arbeit erheblich vereinfachen, indem wir Wissen, das als bedingte Wahrscheinlichkeit formuliert ist, grafisch darstellen. Abbildung 7.11 zeigt das Bayes'sche Netzwerk für das Alarmbeispiel. Jeder Knoten im Netzwerk repräsentiert eine Variable und jede gerichtete Kante eine Aussage über die bedingte Wahrscheinlichkeit. Die Kante von *Al* nach J stellt beispielsweise die beiden Werte P (J |*Al)* und P (J |*¬Al) dar,* die in Form einer Tabelle, der sogenannten CPT (Conditional Probability Table), angegeben werden. Die CPT eines Knotens listet alle bedingten Wahrscheinlichkeiten der Variablen des Knotens auf, die von allen Knoten abhängig sind, die durch eingehende Kanten verbunden sind.**

**Beim Studium des Netzwerks fragen wir uns vielleicht, warum außer den vier eingezeichneten Kanten keine weiteren Kanten enthalten sind. Die beiden Knoten *Bur* und *Ear* sind nicht verknüpft, da die Variablen unabhängig sind. Alle anderen Knoten haben einen übergeordneten Knoten, was die Argumentation etwas komplexer macht. Wir brauchen zunächst das Konzept der bedingten Unabhängigkeit.**

## 7.4.3 Bedingte Unabhängigkeit

**Analog zur Unabhängigkeit von Zufallsvariablen geben wir an**

**Definition 7.6 Zwei Variablen A und B heißen *bedingt unabhängig,* gegeben C wenn**

$$P(A, B|C) = P(A|C) \cdot P(B|C).$$

**Diese Gleichung gilt für alle Wertekombinationen für alle drei Variablen (also für die Verteilung), die wir in der Notation sehen. Wir betrachten nun im Alarmbeispiel die Knoten J und M, die den gemeinsamen Elternknoten *Al haben.* Wenn John und Mary**

unabhängig auf einen Alarm reagieren, dann sind die beiden Variablen J und M bei gegebenem *Al unabhängig,* das heißt:

$$P(J,M|Al) = P(J|Al) \cdot P(M|Al).$$

Ist der Wert von *Al* bekannt, beispielsweise weil ein Alarm ausgelöst wurde, sind die Variablen J und M unabhängig (unter der Bedingung *Al* = w). Aufgrund der bedingten Unabhängigkeit der beiden Variablen J und M wird keine Kante zwischen diesen beiden Knoten hinzugefügt. Allerdings sind J und M nicht unabhängig (siehe Aufgabe 7.11 auf Seite 159).

Ganz ähnlich ist der Zusammenhang zwischen den beiden Variablen J und *Bur,* denn John reagiert nicht auf einen Einbruch, sondern auf den Alarm. Dies könnte zum Beispiel daran liegen, dass eine hohe Mauer ihm die Sicht auf Bobs Grundstück versperrt, er den Alarm aber dennoch hören kann. Somit sind J und *Bur* bei gegebenem *Al* und unabhängig

$$P(J,Bur|Al) = P(J|Al) \cdot P(Bur|Al).$$

Bei einem Alarm sind auch die Variablen J und *Ear,* M und *Bur* sowie M und *Ear* unabhängig. Für die Berechnung mit bedingter Unabhängigkeit sind die folgenden Charakterisierungen hilfreich, die der obigen Definition entsprechen:

**Satz 7.5** ***Die folgenden Gleichungen sind paarweise äquivalent, was bedeutet, dass jede einzelne Gleichung die bedingte Unabhängigkeit für die Variablen*** **A** ***und*** **B** ***bei gegebenem*** **C** beschreibt.

$$P(A, B|C) = P(A|C) \cdot P(B|C), P(A|B,C) \tag{7.16}$$

$$= P(A|C), P(B|A,C) = P(B|C). \tag{7.17}$$

$$\tag{7.18}$$

***Beweis*** **Einerseits können wir** mithilfe der bedingten Unabhängigkeit (7.16) daraus schließen

$$P(A, B, C) = P(A, B|C)P(C) = P(A|C)P(B|C)P(C).$$

Andererseits gibt uns die Produktregel

$$P(A, B, C) = P(A|B,C)P(B|C)P(C).$$

Somit ist P(A|B,C) = P(A|C) äquivalent zu (7.16). Analog erhalten wir (7.18) durch Vertauschen von A und B in dieser Ableitung.

### 7.4.4 Praktische Anwendung

Nun wenden wir uns wieder dem Alarmbeispiel zu und zeigen, wie das Bayes-Netzwerk in Abb. 7.11 auf Seite 146 zur Argumentation genutzt werden kann. Bob interessiert sich beispielsweise für die Sensibilität seiner beiden Alarmreporter John und Mary, also für $\mathbf{P}(J|Bur)$ und $\mathbf{P}(M|Bur)$. Noch wichtiger sind ihm jedoch die Werte $\mathbf{P}(Bur|J)$ und $\mathbf{P}(Bur|M)$ sowie $\mathbf{P}(\text{Bur}|J,M)$. Wir beginnen mit $\mathbf{P}(J|Bur)$ und berechnen

$$\mathbf{P}(J|Bur) = \frac{\mathbf{P}(J,Bur)}{\mathbf{P}(Bur)} = \frac{\mathbf{P}(J,Bur,Al) + \mathbf{P}(J,Bur,\neg Al)}{\mathbf{P}(Bur)} \tag{7.19}$$

Und

$$P(J,Bur,Al) = \mathbf{P}(J|Bur,Al)P(Al|Bur)P(Bur) = \mathbf{P}(J|Al)P(Al|Bur)P(Bur), \tag{7.20}$$

wobei wir für die letzten beiden Gleichungen die Produktregel und die bedingte Unabhängigkeit von J und *Bur* bei gegebenem *Al verwendet haben.* Eingefügt in (7.19) erhalten wir

$$\begin{aligned}\mathbf{P}(J|Bur) &= \frac{\mathbf{P}(J|Al)\mathbf{P}(Al|Bur)\mathbf{P}(Bur) + \mathbf{P}(J|\neg Al)\mathbf{P}(\neg Al|Bur)\mathbf{P}(Bur)}{\mathbf{P}(Bur)} \\ &= \mathbf{P}(J|Al)\mathbf{P}(Al|Bur) + \mathbf{P}(J|\neg Al)\mathbf{P}(\neg Al|Bur).\end{aligned} \tag{7.21}$$

Hier fehlen $\mathbf{P}(Al|Bur)$ und $\mathbf{P}(\neg \text{Al}|\text{Bur})$. Deshalb rechnen wir

$$\begin{aligned}\mathbf{P}(Al|Bur) &= \frac{\mathbf{P}(Al,Bur)}{\mathbf{P}(Bur)} = \frac{\mathbf{P}(Al,Bur,Ohr) + \mathbf{P}(Al,Bur,\neg Ohr)}{\mathbf{P}(Bur)} \\ &= \frac{\mathbf{P}(Al|Bur,Ohr)\mathbf{P}(Bur)\mathbf{P}(Ohr) + \mathbf{P}(\text{Al}|\text{Bur},\neg\text{Ohr})\mathbf{P}(Bur)\mathbf{P}(\neg Ohr)}{\mathbf{P}(Bur)} \\ &= \mathbf{P}(Al|Bur,Ohr)\mathbf{P}(Ohr) + \mathbf{P}(\text{Al}|\text{Bur},\neg\text{Ohr})\mathbf{P}(\neg Ohr) \\ &= 0{,}95 \cdot 0{,}002 + 0{,}94 \cdot 0{,}998 = 0{,}94\end{aligned}$$

sowie $\mathbf{P}(\neg \text{Al}|\text{Bur}) = 0{,}06$ und fügen Sie dies in (7.21) ein , was das Ergebnis ergibt

$$\mathbf{P}(J|Bur) = 0{,}9 \cdot 0{,}94 + 0{,}05 \cdot 0{,}06 = 0{,}849.$$

Analog berechnen wir $\mathbf{P}(M|Bur) = 0{,}659$. Wir wissen jetzt, dass John für etwa 85 % aller Einbrüche verantwortlich ist und Mary für etwa 66 % aller Einbrüche. Die Wahrscheinlichkeit, die beide nennen, wird aufgrund der bedingten Unabhängigkeit wie folgt berecl

$$\mathbf{P}(J,M|\text{Bur}) = \mathbf{P}(J|Bur)\mathbf{P}(M|Bur) = 0{,}849 \cdot 0{,}659 = 0{,}559.$$

Interessanter ist jedoch die Wahrscheinlichkeit eines Anrufs von John oder Mary

$$P(J \mid M, Bur) = P(\neg(\neg J, \neg M) \mid Bur) = 1 - P(\neg J, \neg M \mid Bur)$$
$$= 1 - P(\neg J \mid Bur) P(\neg M \mid Bur) = 1 - 0{,}051 = 0{,}948.$$

Somit wird Bob bei etwa 95 % aller Einbrüche benachrichtigt. Um nun $P(Bur \mid J)$ zu berechnen, wenden wir die Bayes-Formel an, die uns ergibt

$$P(Bur \mid J) = \frac{P(J \mid Bur) P(Bur)}{P(J)} = \frac{0{,}849 \cdot 0{,}001}{0{,}052} = 0{,}016.$$

Offenbar sind nur etwa 1,6 % aller Anrufe von John tatsächlich auf einen Einbruch zurückzuführen. Da die Wahrscheinlichkeit von Fehlalarmen bei Mary fünfmal geringer ist, haben wir mit $P(Bur \mid M) = 0{,}056$ ein deutlich höheres Vertrauen bei einem Anruf von Mary. Bob sollte sich nur ernsthafte Sorgen um sein Zuhause machen, wenn beide anrufen, denn $P(Bur \mid J, M) = 0{,}284$ (siehe Übung 7.11 auf Seite 159).

In (7.21) auf Seite 148 haben wir mit gezeigt

$$P(J \mid Bur) = P(J \mid Al) P(Al \mid Bur) + P(J \mid \neg Al) P(\neg Al \mid Bur)$$

wie wir eine neue Variable „einschieben“ können. Diese Beziehung gilt im Allgemeinen für zwei Variablen A und B bei Einführung einer zusätzlichen Variablen C und wird als *Konditionierung bezeichnet:*

$$P(A \mid B) = \sum_c P(A \mid B, C = c) P(C = c \mid B).$$

Wenn außerdem A und B bei gegebenem C bedingt unabhängig sind, vereinfacht sich diese Formel zu

$$P(A \mid B) = \sum_c P(A \mid C = c) P(C = c \mid B).$$

## 7.4.5 Software für Bayesianische Netzwerke

Am Beispiel Alarm geben wir eine kurze Einführung in zwei Tools. Das System PIT ist uns bereits bekannt. Die Werte aus den CPTs geben wir in PIT-Syntax in das Online-Eingabefenster unter www.pit-systems.de ein. Nach der in Abb. 7.12 auf Seite 150 dargestellten Eingabe erhalten wir die Antwort:

P([Einbruch=t] | [John=t] AND [Mary=t]) = 0,2841.

Obwohl PIT kein klassisches Bayes'sches Netzwerktool ist, kann es beliebige bedingte Wahrscheinlichkeiten und Abfragen als Eingabe verwenden und korrekte Ergebnisse berechnen. Es kann gezeigt werden [Sch96], dass das MaxEnt-Prinzip bei Eingabe von CPTs oder äquivalenten Regeln die gleichen bedingten Unabhängigkeiten und damit auch die gleichen Antworten impliziert wie ein Bayesianisches Netzwerk. Bayesianische Netzwerke sind somit ein Sond

```
1 var Alarm{t,f}, Einbruch{t,f}, Erdbeben{t,f}, John{t,f}, Mary{t,f};
2
3 P([Erdbeben=t]) = 0,002; 4 P([Einbruch=t])
= 0,001; 5 P([Alarm=t] | [Einbruch=t]
UND [Erdbeben=t]) = 0,95; 6 P([Alarm=t] | [Einbruch=t] UND [Erdbeben=f]) = 0,94; 7
P([Alarm=t] | [Einbruch=f] UND [Erdbeben=t]) = 0,29; 8 P([Alarm=t] | [Einbruch=f]
UND [Erdbeben=f]) = 0,001; 9 P([John=t] | [Alarm=t]) = 0,90; 10 P([John=t] | [Alarm=f])
= 0,05; 11 P([Mary=t] | [Alarm=t]) = 0,70; 12 P([Mary=t] | [Alarm=f]) = 0,01; 13 14
QP([Einbruch=t] | [John=t] AND [Mary=t]);
```

**Abb. 7.12 PIT-Eingabe für das Alarmbeispiel**

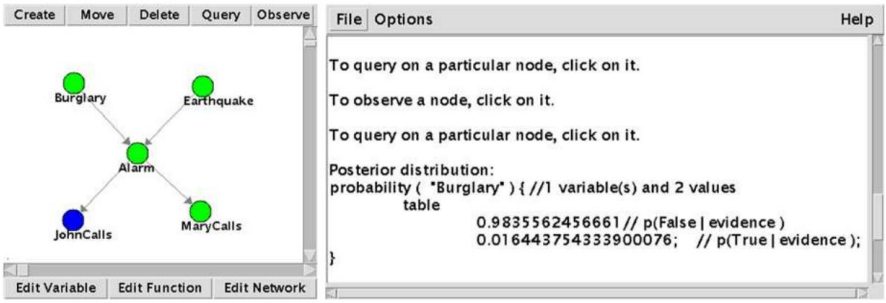



**Abb. 7.13 Die Benutzeroberfläche von JavaBayes: *Links* der grafische Editor und *rechts* die Konsole, in der die Antworten als Ausgabe ausgegeben werden**

Als nächstes betrachten wir JavaBayes [Coz98], ein klassisches System, das ebenfalls frei im Internet verfügbar ist und über die in Abb. 7.13 dargestellte grafische Oberfläche verfügt. Mit dem grafischen Netzwerkeditor können Knoten und Kanten manipuliert und die Werte in den CPTs bearbeitet werden. Darüber hinaus können mit „Observe“ die Werte von Variablen zugewiesen und mit „Query“ die Werte anderer Variablen abgerufen werden. Die Antworten auf Abfragen erscheinen dann im Konsolenfenster.

Das professionelle, kommerzielle System Hugin ist deutlich leistungsfähiger und komfortabler. Beispielsweise kann Hugin neben diskreten Variablen auch kontinuierliche Variablen verwenden. Es kann auch Bayes'sche Netzwerke lernen, also das Netzwerk vollautomatisch aus statistischen Daten generieren (siehe Abschn. 8.5).

## 7.4.6 Entwicklung Bayes'scher Netzwerke

Ein kompaktes Bayes-Netzwerk ist sehr übersichtlich und für den Leser deutlich informativer als eine vollständige Wahrscheinlichkeitsverteilung. Darüber hinaus wird viel weniger benö

Erinnerung. Für die Variablen v1,...,vn mit |v1|,...,|vn| Jeweils unterschiedliche Werte, die Verteilung hat insgesamt

$$\prod_{i=1}^{N} |vi| \ddot{y} 1$$

unabhängige Einträge. Im Alarmbeispiel sind alle Variablen binär. Somit gilt für alle Variablen |vi | = 2, und die Verteilung hat 25 ÿ 1 = 31 unabhängige Einträge. Um die Anzahl der unabhängigen Einträge für das Bayes-Netzwerk zu berechnen, muss die Gesamtzahl aller Einträge aller CPTs ermittelt werden. Für einen Knoten vi mit ki- Elternknoten ei1,...,eiki , das zugehörige CPT hat

$$(|vi| \ddot{y} 1) \prod_{j=1}^{ki} |eij|$$

Einträge. Dann haben alle CPTs im Netzwerk zusammen

$$\prod_{i=1}^{N} (|vi| \ddot{y} 1) \prod_{j=1}^{ki} |eij| \qquad (7.22)$$

Einträge.15 Für das Alarmbeispiel lautet das Ergebnis dann

$$2 + 2 + 4 + 1 + 1 = 10$$

unabhängige Einträge, die das Netzwerk eindeutig beschreiben. Der Vergleich der Speicherkomplexität zwischen der Vollverteilung und dem Bayes'schen Netzwerk wird klarer, wenn wir annehmen, dass alle n Variablen die gleiche Anzahl b an Werten haben und jeder Knoten k Elternknoten hat. Dann kann (7.22) vereinfacht werden und alle CPTs zusammen haben n(b ÿ 1)bk Einträge. Die vollständige Verteilung enthält b. $^{N}$ ÿ 1 Einträge. Ein Signifikat Überhöhungsgewinn wird nur dann erzielt, wenn die durchschnittliche Anzahl der übergeordneten Knoten viel kleiner ist als die Anzahl der Variablen. Dies bedeutet, dass die Knoten nur lokal Durch die lokale Anbindung wird das Netzwerk modularisiert, was – wie im Software-Engineering – zu einer Reduzierung der Komplexität führt. Im Alarmbeispiel trennt der Alarmknoten die Knoten *Bur* und *Ear* von den Knoten J und M. Dies können wir auch im LEXMED- Beispiel deutlich sehen.

15Für den Fall eines Knotens ohne Vorfahren ist das Produkt in dieser Summe leer. Dafür ersetzen wir den Wert 1, da die CPT für Knoten ohne Vorfahren mit ihrer A-priori-Wahrscheinlichkeit genau einen Wert enthält.

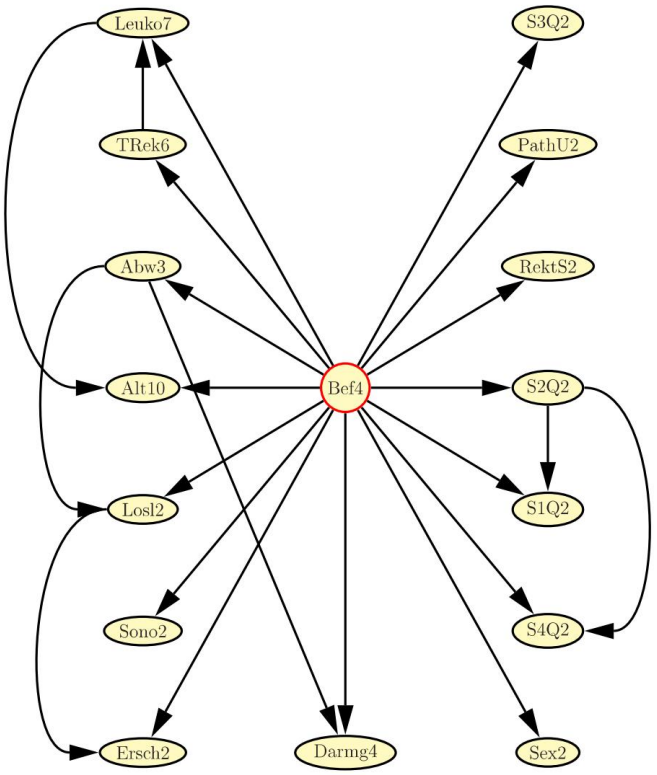



**Abb. 7.14 Bayesianisches Netzwerk für die LEXMED- Anwendung**

**LEXMED als Bayesianisches Netzwerk**

**Das in Abschn. beschriebene LEXMED-** System. **7.3 kann auch als Bayes'sches Netzwerk modelliert werden. Indem man die äußeren, dünn gezeichneten Linien gerichtet macht (sie mit Pfeilen versehen), kann der Unabhängigkeitsgraph in Abb. 7.8 auf Seite 137 als Bayes'sches Netzwerk interpretiert werden. Das resultierende Netzwerk ist in Abb. 7.14 dargestellt .**

**In Abschn. 7.3.2 Die Größe der Verteilung für LEXMED wurde berechnet als Wert 20 643 839. Das Bayes'sche Netzwerk hingegen kann mit nur 521 Werten vollständig beschrieben werden. Dieser Wert kann ermittelt werden, indem die Variablen aus Abb. 7.14 in (7.22) auf Seite 151 eingegeben werden. In der Reihenfolge (Leuko, TRek, Gua, Age, Reb, Sono, Tapp, BowS, Sex, P4Q, P1Q, P2Q, RecP , Urin, P3Q, Diag4) berechnen wir**

$$\sum_{i=1}^{N} (|v_i \mid \ddot{y} 1) \qquad |e_{ij}| = 6 \cdot 6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 7 \cdot 4 + 1 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 2 \cdot 4 \; j=1^{k_i}$$

$$+ 3 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 + 1 \cdot 4 + 1 \cdot 4$$

$$+ 1 \cdot 4 + 1 = 521.$$

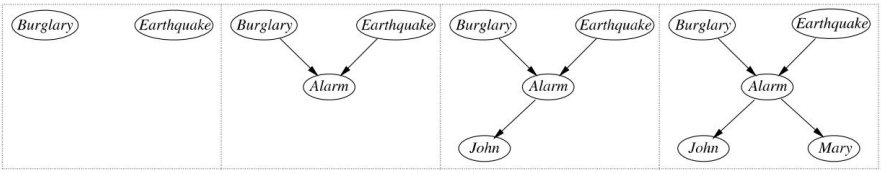



**Abb. 7.15 Schrittweiser Aufbau des Alarmnetzwerks unter Berücksichtigung der Kausalität**

**Dieses Beispiel zeigt, dass es praktisch unmöglich ist, eine vollständige Distribution für reale Anwendungen zu erstellen. Ein Bayes-Netzwerk mit 22 Kanten und 521 Wahrscheinlichkeitswerten ist dagegen noch beherrschbar.**

**Kausalität und Netzwerkstruktur Der**
**Aufbau eines Bayes'schen Netzwerks erfolgt normalerweise in zwei Schritten.**

1. ***Entwurf der Netzwerkstruktur:*** **Dieser Schritt wird in der Regel manuell durchgeführt und im Folgenden beschrieben.**
2. ***Eingabe der Wahrscheinlichkeiten in die CPTs:*** **Bei vielen Variablen ist die manuelle Eingabe der Werte sehr mühsam. Wenn (wie zum Beispiel bei LEXMED) eine Datenbank verfügbar ist, kann dieser Schritt automatisiert werden, indem die CPT-Einträge durch Zählen von Häufigkeiten geschätzt werden.**

**Wir beschreiben nun den Aufbau des Netzwerks am Alarmbeispiel (siehe Abb. 7.15). Zu Beginn kennen wir die beiden Ursachen *Einbruch* und *Erdbeben* und die beiden Symptome *John* und *Mary.* Da John und Mary jedoch nicht direkt auf einen Einbrecher oder ein Erdbeben reagieren, sondern nur auf den Alarm, ist es angebracht, dies als zusätzliche Variable hinzuzufügen, die von Bob nicht beobachtbar ist. Der Prozess des Hinzufügens von Kanten beginnt bei den Ursachen, also bei den Variablen, die keine übergeordneten Knoten haben. Zuerst wählen wir *Einbruch* und als nächstes *Erdbeben.* Jetzt müssen wir prüfen, ob *Erdbeben* unabhängig von *Einbruch ist.* Dies ist gegeben und daher wird kein Vorteil von *Einbruch* zu *Erdbeben hinzugefügt.* Da *„Alarm“* direkt von *„Einbruch* “ und „Erdbeben“ abhängt , werden diese Variablen als Nächstes ausgewählt und eine Kante von „ Einbruch “ und *„Erdbeben“* zu *„Alarm“* hinzugefügt . Dann wählen wir *John.* Da *Alarm* und *John* nicht unabhängig sind, wird eine Kante von Alarm zu John hinzugefügt. Dasselbe gilt auch für *Maria.* Nun müssen wir prüfen, ob *John* bei *Einbruchalarm* bedingt unabhängig ist . Ist dies nicht der Fall, muss eine weitere Kante von *Einbruch* zu *John eingefügt werden.* Wir müssen auch prüfen, ob Kanten von *Erdbeben* zu *John* und von *Einbruch* oder *Erdbeben* zu *Mary* benötigt werden . Aufgrund der bedingten Unabhängigkeit sind diese vier Kanten nicht notwendig. Kanten zwischen *John* und *Mary* sind auch unnötig, da *John* und *Mary* bei *Alarm* bedingt unabhängig sind .**

**Die Struktur des Bayes'schen Netzwerks hängt stark von der gewählten Variablenreihenfolge ab. Wenn die Reihenfolge der Variablen so gewählt wird, dass sie den Kausalzusammenhang widerspiegelt, beginnend bei den Ursachen und fortschreitend zu den Diagnosevariablen, dann wird das Ergebnis ein einfaches Netzwerk sein. Andernfalls kann das Netzwerk deutlich mehr Kanten ent Solche nicht-kausalen Netzwerke sind oft sehr schwer zu verstehen und haben eine höhere Relevanz**

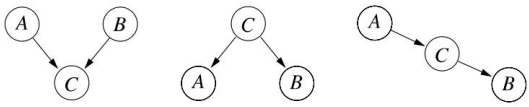



**Abb. 7.16 Es gibt keine Kante zwischen A und B, wenn sie unabhängig *(links)* oder bedingt unabhängig *(Mitte, rechts)*** sind.

**Komplexität für das Denken.** Zum besseren Verständnis **kann der Leser auf Übung 7.11 auf Seite 159** verweisen .

## 7.4.7 Semantik Bayes'scher Netzwerke

Wie wir im vorherigen Abschnitt gesehen haben, wird einem Bayes'schen Netzwerk zwischen zwei Variablen A und B keine Kante hinzugefügt, wenn A und B unabhängig oder bedingt unabhängig sind und eine dritte Variable C gegeben ist. Diese Situation ist in Abb. **7.16 dargestellt.**

**Wir fordern nun, dass das Bayes'sche Netzwerk keine Zyklen hat, und gehen davon aus, dass die Variablen so nummeriert sind, dass keine Variable eine niedrigere Nummer hat als jede Variable, die ihr vorangeht. Dies ist immer dann möglich, wenn das Netzwerk keine Zyklen hat.16 Dann gilt, wenn alle bedingten Unabhängigkeiten verwendet werden**

$$P(Xn|X1,...,Xnÿ1) = P(Xn|Eltern(Xn)).$$

**Diese Gleichung ist somit eine Aussage, dass eine beliebige Variable Xi in einem Bayes'schen Netzwerk angesichts ihrer Eltern bedingt unabhängig von ihren Vorfahren ist. Der etwas allgemeinere Satz, der in Abb. 7.17 auf Seite 155** dargestellt ist , **lässt sich kompakt formulieren als**

**Satz 7.6** Ein Knoten in einem Bayes'schen Netzwerk ist ***aufgrund seiner Eltern bedingt unabhängig von allen Nicht-Nachfolgeknoten .***

**Jetzt können wir die Kettenregel ((7.1) auf Seite 120) stark vereinfachen:**

$$P(X1,...,Xn) = \prod_{i=1}^{N} P(Xi\ |X1\ ...,Xiÿ1) = \prod_{i=1}^{N} P(Xi\ |Eltern(Xi)). \quad (7.23)$$

**Mit dieser Regel könnten wir beispielsweise (7.20) direkt auf Seite 148** schreiben

$$P(J,Bur,Al) = P(J\ |Al)P(Al|Bur)P(Bur).$$

---

**16Wenn zum Beispiel drei Knoten X1, X2, X3** einen Kreis bilden , **dann gibt es die Kanten (X1, X2 ), ( X2, X3) und ( X3 ,**

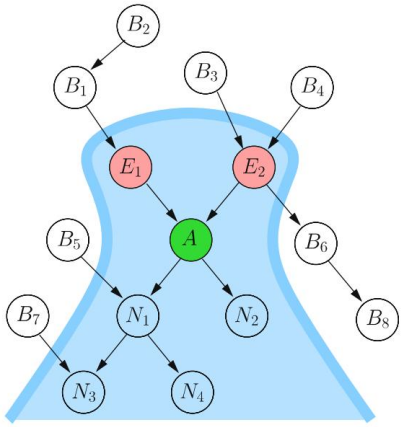



**Abb. 7.17 Beispiel für bedingte Unabhängigkeit in einem Bayes'schen Netzwerk. Sind die Elternknoten E1 und E2 gegeben, dann sind alle Nicht-Nachfolgeknoten B1,...,B8 unabhängig von A**

**Wir kennen jetzt die wichtigsten Konzepte und Grundlagen des Bayes'schen Netzes funktioniert. Fassen wir sie zusammen [Jen01]:**

**Definition 7.7 Ein Bayes'sches Netzwerk ist definiert durch: • Eine Menge von Variablen und eine Menge gerichteter Kanten zwischen diesen Variablen. • Jede Variable hat endlich viele mögliche Werte. • Die Variablen bilden zusammen mit den Kanten einen gerichteten azyklischen Grapher**
**Ein DAG ist ein Graph ohne Zyklen, also ohne Pfade der Form (A, . . . , A).**
**• Für jede Variable A die CPT (d. h. die Tabelle der bedingten Wahrscheinlichkeiten). P(A|Eltern(A))) ist gegeben.**

**Zwei Variablen A und B heißen *bedingt unabhängig* bei gegebenem C, wenn P(A, B|C) = P(A|C) · P(B|C) oder äquivalent, wenn P(A|B,C) = P( A|C).**
**Neben den Grundregeln der Wahrscheinlichkeitsberechnung gelten auch die folgenden Regeln: P (B|A)·P (A)**
***Satz von Bayes:*** **P (A|B) = P (B)**
***Marginalisierung:*** **P (B) = P (A, B) + P (¬A,B) = P (B|A) · P (A) + P (B|¬A) · P (¬A)**

***Konditionierung:*** **P (A|B) = P (A|B,C = c)P (C = c|B)**
**Eine Variable in einem Bayes'schen Netzwerk ist aufgrund ihrer übergeordneten Variablen bedingt unabhängig von allen Nicht-Nachfolgevariablen. Wenn X1,...,Xnÿ1 keine Nachfolger von Xn sind, gilt P(Xn|X1,...,Xnÿ1) = P(Xn|Parents(Xn)). Diese Bedingung muss beim Aufbau eines Netzwerks berücksichtigt werden.**
**Beim Aufbau eines Bayes'schen Netzwerks sollten die Variablen nach Kausalität geordnet werden. Zuerst die Ursachen, dann die versteckten Variablen und zuletzt die Diagnosevariablen.**
***Kettenregel:*** **P(X1,...,Xn) = $\prod_{i=1}^{n}$ P(Xi |Eltern(Xi))**

In [Pea88] und [Jen01] wird der Begriff d-Separation für Bayes'sche Netze eingeführt, woraus ein Theorem ähnlich zu Theorem 7.6 auf Seite 154 folgt. Wir verzichten auf die Einführung dieses Begriffs und kommen dadurch zu einer etwas einfacheren, wenn auch theoretisch nicht ganz so sauberen Darstellung.

## 7.5 Zusammenfassung

In einer Weise, die den anhaltenden, anhaltenden Trend zu probabilistischen Systemen widerspiegelt, haben wir probabilistische Logik für das Denken mit unsicherem Wissen eingeführt. Nach der Einführung der Sprache – um Wahrscheinlichkeiten oder Wahrscheinlichkeitsintervalle erweiterte Aussagenlogik – wählten wir den natürlichen, wenn auch ungewöhnlichen Ansatz über die Methode der maximalen Entropie als Einstiegspunkt und zeigten, wie wir mit dieser Methode nichtmonotones Denken modellieren können. Bayesianische Netzwerke wurden dann als Sonderfall der MaxEnt-Methode eingeführt.

Warum sind Bayesianische Netzwerke ein Sonderfall von MaxEnt? Beim Aufbau eines Bayes'schen Netzwerks werden Annahmen zur Unabhängigkeit getroffen, die für die MaxEnt-Methode nicht erforderlich sind. Darüber hinaus müssen beim Aufbau eines Bayes'schen Netzwerks alle CPTs vollständig ausgefüllt werden, damit eine vollständige Wahrscheinlichkeitsverteilung er Andernfalls ist die Argumentation eingeschränkt oder unmöglich. Mit MaxEnt hingegen kann der Entwickler sämtliches Wissen, das ihm zur Verfügung steht, in Form von Wahrscheinlichkeiten formulieren. Anschließend vervollständigt MaxEnt das Modell und generiert die Verteilung. Auch wenn (beispielsweise bei der Befragung eines Experten) nur vage Aussagen vorliegen, kann dies geeignet modelliert werden. Eine Aussage wie „Ich bin mir ziemlich sicher, dass A wahr ist.“ kann beispielsweise mit P (A) ÿ [0,6, 1] als Wahrscheinlichkeitsintervall modelliert werden. Beim Aufbau eines Bayes'schen Netzwerks muss, ggf. durch Schätzung, ein konkreter Wert für die Wahrscheinlichkeit angegeben werden. Dies bedeutet jedoch, dass der Experte oder der Entwickler Ad-hoc-Informationen in das System einpflegt. Ein weiterer Vorteil von MaxEnt ist die Möglichkeit, (fast) beliebige Aussagen zu formulieren. Für Bayes'sche Netzwerke müssen die CPTs gefüllt sein.

Die Freiheit, die der Entwickler bei der Modellierung mit MaxEnt hat, kann (insbesondere für einen Anfänger) von Nachteil sein, da im Gegensatz zum Bayesianischen Ansatz nicht unbedingt klar ist, welches Wissen modelliert werden soll. Bei der Modellierung mit Bayes’schen Netzen ist die Vorgehensweise ganz klar: Entsprechend der kausalen Abhängigkeiten, von den Ursachen bis zu den Wirkungen, wird durch Prüfung der bedingten Unabhängigkeit eine Kante nach der anderen in das Netz eingetragen.17 Am Ende werden alle CPTs mit Wer

Folgende interessante Kombinationen der beiden Methoden sind jedoch möglich: Wir bauen zunächst ein Netzwerk nach der Bayes’schen Methodik auf, geben alle Kanten entsprechend ein und füllen dann die CPTs mit Werten. Sollten bestimmte Werte für die CPTs nicht verfügbar sein, können sie durch Intervalle oder andere probabilistische Logikformeln ersetzt werden. Natürlich gibt es ein solches Netzwerk – oder besser: ein Regelwerk – nicht

17Das ist auch nicht immer ganz so einfach.

hat nicht mehr die spezielle Semantik eines Bayes'schen Netzwerks. Anschließend muss es von einem MaxEnt-System verarbeitet und vervollständigt werden.

Die Möglichkeit, MaxEnt mit beliebigen Regelsätzen zu verwenden, hat jedoch einen Nachteil. Ähnlich wie in der Logik können solche Regelsätze inkonsistent sein. Beispielsweise sind die beiden Regeln P (A) = 0,7 und P (A) = 0,8 inkonsistent. Während beispielsweise das MaxEnt-System PIT die Inkonsistenz erkennen kann, kann es keinen Hinweis darauf geben, wie das Problem behoben werden kann.

Wir stellten das medizinische Expertensystem LEXMED vor, eine klassische Anwendung zur Argumentation mit unsicherem Wissen, und zeigten, wie es mithilfe von MaxEnt und Bayesianischen Netzwerken modelliert und implementiert werden kann und wie diese Tools das etablierte, aber zu schwache lineare Scoring ersetzen können Systeme, die in der Medizin verwendet werd

Im Beispiel von LEXMED haben wir gezeigt, dass es möglich ist, ein Expertensystem für das Denken unter Unsicherheit aufzubauen, das in der Lage ist, Wissen aus den Daten in einer Datenbank zu entdecken (zu lernen). Weitere Einblicke in die Lernmethoden Bayes'scher Netzwerke geben wir in Kap. 8, nachdem die notwendigen Grundlagen für maschinelles Lernen gelegt wurden.

Heute ist das Bayesianische Denken ein großes, eigenständiges Feld, das wir hier nur kurz beschreiben können. Auf die Behandlung kontinuierlicher Variablen haben wir komplett verzichtet. Für den Fall normalverteilter Zufallsvariablen gibt es Verfahren und Systeme. Bei beliebigen Verteilungen ist jedoch der Rechenaufwand ein großes Problem.
Neben den gerichteten Netzwerken, die stark auf Kausalität basieren, gibt es auch ungerichtete Netzwerke. Damit verbunden ist eine Diskussion über die Bedeutung und den Nutzen von Kausalität in Bayes'schen Netzwerken. Der interessierte Leser wird auf hervorragende Lehrbücher wie [Pea88, Jen01, Whi96, DHS01] sowie auf die Tagungsberichte der Jahreskonferenz der Association for Uncertainty *in Artificial Intelligence (AUAI)* verwiesen (www.auai.org).

## 7.6 Übungen

**Aufgabe 7.1** Beweisen Sie den Satz aus Satz 7.1 auf Seite 117.

**Übung 7.2** Der Hobbygärtner Max möchte seine jährliche Erbsenernte statistisch analysieren. Für jede Erbsenschote, die er pflückt, misst er deren Länge xi in Zentimetern und ihr Gewicht yi in Gramm. Er teilt die Erbsen in zwei Klassen ein, die guten und die schlechten (leeren Schoten). Die gemessenen Daten (xi, yi) sind

| | x | | | | |
|---|---|---|---|---|---|
| gute Erbsen: | 122 | 334 | 456 | und | 234 |
| | 455 | 666 | | | |
| schlechte Erbsen: | x | 4667 | und | | |
| | 2233 | | | | |

18In Abschn. 8.6 und in Übung 8.16 auf Seite 219 werden wir zeigen, dass die Scores äquivalent zum Spezialfall naive Bayes sind, also zur Annahme, dass alle Symptome angesichts der Diagnose bedingt unabhängig sind.

(a) Berechnen Sie aus den Daten die Wahrscheinlichkeiten P (y > 3 /*Klasse = gut)* und P (y ÿ 3 /*Klasse = gut).* Verwenden Sie dann die Bayes-Formel, um P *(Klasse =)* zu bestimmen *gut* | y > 3) und P *(Klasse = gut* | y ÿ 3).

(b) Welche der in Teilaufgabe (a) berechneten Wahrscheinlichkeiten widerspricht der Aussage? „Alle guten Erbsen sind schwerer als 3 Gramm“?

Übung 7.3 Sie sollen das Nachmittagswetter mithilfe einiger einfacher Vorhersagen vorhersagen Wetterwerte vom Morgen dieses Tages. Die klassische Wahrscheinlichkeitsrechnung Hierzu ist ein vollständiges Modell erforderlich, das in der folgenden Tabelle aufgeführt ist.

| *Himmel* | *Bar* | *Prec* P *(Sky, Bar, Prec)* |
|---|---|---|
| Klar | Steigende | 0,40 |
| Klar | Trockenheit Steigende Regenfälle | 0,07 |
| Klar | Trockenfallen | 0,08 |
| Klar | Fallender Regen | 0,10 |
| Bewölkt | Steigend Trocken | 0,09 |
| Bewölkt | Steigend Regen | 0.11 |
| Bewölkt | fallend trocken | 0,03 |

*Himmel:* Der Himmel ist klar oder bewölkt am Morgen
*Balken:* Barometer steigend bzw am Morgen fallen
*Prec:* Regen oder Trockenheit im Nachmittag

(a) Wie viele Ereignisse enthält die Verteilung für diese drei Variablen?

(b) Berechnen Sie P *(Prec* = trocken, Himmel = klar, Balken = steigend).

(c) Berechnen Sie P *(Prec* = Regen|Sky = bewölkt).

(d) Was würden Sie tun, wenn die letzte Zeile in der Tabelle fehlen würde?

ÿ Übung 7.4 In einer Fernsehquizshow muss der Teilnehmer zwischen drei wählen verschlossene Türen. Hinter einer Tür wartet der Preis: ein Auto. Hinter beiden Türen sind Ziegen. Der Teilnehmer wählt eine Tür, z. B. Nummer eins. Der Gastgeber, der weiß, wo das Auto ist, öffnet eine andere Tür, z. B. Nummer drei, und eine Ziege erscheint. Der Teilnehmer hat nun die Möglichkeit, zwischen den beiden verbleibenden zu wählen Türen (eine und zwei). Was ist aus seiner Sicht die bessere Wahl? Bleiben mit die Tür, die er ursprünglich gewählt hatte, oder zu der anderen geschlossenen Tür wechseln?

Übung 7.5 Zeigen Sie dies mithilfe der Lagrange-Multiplikatormethode, ohne explizit Einschränkungen stellt die Gleichverteilung p1 = p2 =···= pn = 1/n die maximale Entropie dar. Vergessen Sie nicht die implizit allgegenwärtige Einschränkung p1 + p2 +···+ pn = 1. Wie können wir dasselbe Ergebnis mithilfe der Indifferenz zeigen?

Aufgabe 7.6 Verwenden Sie das PIT-System (www.pit-systems.de) , um den MaxEnt zu berechnen Lösung für P (B) unter der Randbedingung P (A) = ÿ und P (B|A) = ÿ. Welchen Nachteil der PIT im Vergleich zur Berechnung per Hand fällt Ihnen hier auf?

Übung 7.7 Berechnen Sie P (B) anhand der Randbedingungen P (A) = ÿ und P (A ÿ B) = ÿ manuell mit der MaxEnt-Methode. Verwenden Sie PIT, um Ihre Lösung zu überprüfen.

ÿ **Aufgabe 7.8 Gegeben seien die Nebenbedingungen aus (7.10), (7.11), (7.12): p1 + p2 = ÿ, p1 + p3 = ÿ, p1 + p2 + p3 + p4 = 1. Zeigen Sie, dass p1 = ÿÿ, p2 = ÿ (1 ÿ ÿ ), p3 = ÿ (1 ÿ ÿ), p4 = (1 ÿ ÿ)(1 ÿ ÿ ) stellt das Entropiemaximum unter diesen Einschränkungen dar.**

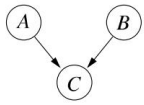
ÿ **Übung 7.9 Ein probabilistischer Algorithmus berechnet die Wahrscheinlichkeit p, dass es sich bei einer eingehenden E-Mail um Spam handelt. Um die E-Mails in die Klassen *„Löschen“* und *„Lesen“* *zu klassifizieren,* wird**

**anschließend eine Kostenmatrix auf das Ergebnis angewendet. (a) Geben Sie eine Kostenmatrix (2 × 2-Matrix) für den Spamfilter an. Gehen Sie hier davon aus, dass das Löschen einer E-Mail den Benutzer 10 Cent kostet, während der Verlust einer E-Mail 10 Dollar kostet (vergleichen Sie dies mit Beispiel 1.1 auf Seite 11 und Übung 1.7 auf Seite 14). (b) Zeigen Sie, dass für den Fall einer 2 × 2-Matrix die Anwendung der Kostenmatrix äquivalent zur Anwendung eines Schwellenwerts auf die Spam-Wahrscheinlichkeit ist, und bestimmen Sie den Schwellenwert.**

**Aufgabe 7.10 Gegeben sei ein Bayes-Netzwerk mit den drei Binärvariablen A,B,C und P (A) = 0,2, P (B) = 0,9 sowie der unten gezeigten CPT:**

**(a) Berechnen Sie P (A|B).**
**(b) Berechnen Sie P (C|A).**

| ABP (C) wf 0,1 |
|---|
| ww 0,2 |
| |
| fw 0,9 ff 0,4 |
| |

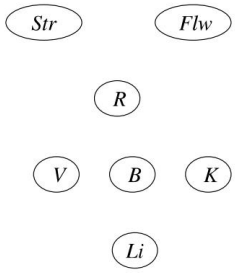



**Aufgabe 7.11 Berechnen Sie für das Alarmbeispiel (Beispiel 7.7 auf Seite 145) die folgenden bedingten Wahrscheinlichkeiten: (a) Berechnen Sie**
**die A-priori-Wahrscheinlichkeiten P *(Al),* P (J), P (M). (b) Berechnen Sie**
**P *(M|Bur)* unter Verwendung der Produktregel, der Marginalisierung, der Kettenregel und bedingte Unabhängigkeit. (c)**
**Verwenden Sie die Bayes-Formel, um P (Bur|M) zu**
**berechnen . (d) Berechnen Sie P (Al|J,M) und P (Bur|**
**J,M). (e) Zeigen Sie, dass die Variablen J und M nicht unabhängig**
**sind. (f) Überprüfen Sie alle Ihre Ergebnisse mit JavaBayes und mit PIT (siehe [Ert11] für Demo). Programme).**
**(g) Entwerfen Sie ein Bayes'sches Netzwerk für das Alarmbeispiel, jedoch mit der geänderten Variablenreihenfolge *M,Al,Ear,Bur,* J . Gemäß der Semantik Bayes'scher Netzwerke müssen nur die notwendigen Kanten eingezeichnet werden. (Hinweis: Die hier angegebene Variablenreihenfolge stellt KEINE Kausalität dar. Daher wird es schwierig sein, bedingte Unabhängigkeiten intuitiv zu bestimmen.)**
**(h) Im ursprünglichen Bayes'schen Netzwerk des Alarmbeispiels wurden die Erdbebenknoten entfernt. Welche CPTs ändert sich dadurch? (Warum gerade diese?)**
**(i) Berechnen Sie die CPT des Alarmknotens im neuen Netzwerk.**

**Aufgabe 7.12 Es soll ein Diagnosesystem für ein dynamobetriebenes Fahrradlicht unter Verwendung eines Bayes'schen Netzwerks erstellt werden. Die Variablen in der folgenden Tabelle sind angegeben.**

| Abk. | Das bedeutet, | *Werte* |
|---|---|---|
| dass *Li* Light im | | t/f |
| ***Str*** | Straßenzustand ist | *trocken, nass, schneebedeckt* |
| ***Flw*** | Dynamo-Schwungrad verschlissen | t/f |
| **R** | Dynamo gleitend | t/f |
| **V** | Dynamo zeigt Spannung | t/f an |
| **B** | Glühbirne | ok t/f |
| **K** | Kabel | ok t/f |

**Die folgenden Variablen sind paarweise unabhängig: *Str, Flw, B, K.* Weiterhin: (R, B), (R, K), (V, B), (V, K) sind unabhängig und die folgende Gleichung hält:**




$$P(Li|V, R) = P(Li|V)$$

$$P(V|R, Str) = P(V|R)$$

$$P(V|R, Flw) = P(V|R)$$

| VBK *P(Li)* |
|---|
| ttt 0,99 |
| ttf 0,01 |
| t ft 0,01 |
| t ff 0,001 |
| ft t 0,3 |
| ft f 0,005 |
| fft 0,005 |
| fff 0 |

**(a) Zeichnen Sie alle Kanten in den Graphen ein (unter Berücksichtigung der Kausalität).**

**(b) Tragen Sie alle fehlenden CPTs in die Grafik (Tabelle der bedingten Wahrscheinlichkeiten) ein. Geben Sie für die Wahrscheinlichkeiten frei plausible Werte ein.**

**(c) Zeigen Sie, dass das Netzwerk keine Kante enthält (Str,Li).**

**(d) Berechnen Sie P (V |*Str = schneebedeckt).***

# Maschinelles Lernen und Data Mining

# 8

Wenn wir KI wie in Elaine Richs Buch [Ric83] definieren:

> Künstliche Intelligenz ist die Untersuchung, wie man Computer dazu bringt, bestimmte Dinge zu tun
> Moment, die Menschen sind besser.

und wenn man bedenkt, dass die Lernfähigkeit des Computers diesem besonders unterlegen ist
des Menschen, dann ist die Erforschung von Lernmechanismen und die Entwicklung maschineller
Lernalgorithmen einer der wichtigsten Zweige
der KI.

Auch aus Sicht der Software besteht Bedarf an maschinellem Lernen
Entwickler, der beispielsweise das Verhalten eines autonomen Roboters programmiert. Der
Die Struktur des intelligenten Verhaltens kann so komplex werden, dass es sehr schwierig ist
Selbst mit modernen Hochsprachen wie PROLOG und Python ist es sogar unmöglich, sie annähernd
optimal zu programmieren.1 Es kommen sogar Algorithmen des maschinellen Lernens zum Einsatz
heute Roboter auf ähnliche Weise zu programmieren, wie Menschen lernen (siehe Kap. 10 bzw
[BCDS08, RGH+06]), oft in einer hybriden Mischung aus programmiert und gelernt
Verhalten.

Die Aufgabe dieses Kapitels besteht darin, die wichtigsten Algorithmen des maschinellen Lernens
und ihre Anwendungen zu beschreiben. Das Thema wird in diesem Abschnitt vorgestellt und anschließend verfolgt
durch wichtige grundlegende Lernalgorithmen in den nächsten Abschnitten. Parallel dazu werden
Theorie und Terminologie aufgebaut. Das Kapitel wird mit einer Zusammenfassung abgeschlossen
und Überblick über die verschiedenen Algorithmen und ihre Anwendungen. Wir beschränken uns in
diesem Kapitel auf überwachte und unüberwachte Lernalgorithmen. Als wichtige Klasse lernender
Algorithmen werden neuronale Netze in Kap. behandelt. 9.
Aufgrund seiner besonderen Stellung und wichtigen Rolle für autonome Roboter, Verstärkung
Auch zum Lernen wird es ein eigenes Kapitel geben (Kap. 10).

Was ist Lernen? Erlernen des Wortschatzes einer Fremdsprache oder einer technischen Sprache
Begriffe zu lernen oder sich sogar ein Gedicht zu merken, kann für viele Menschen schwierig sein. Für Computer,

1Python ist eine moderne Skriptsprache mit sehr gut lesbarer Syntax, leistungsstarken Datentypen und umfangreichen Standardbibliotheken, die zu diesem Zweck verwendet werden können.

W. Ertel, *Einführung in die Künstliche Intelligenz,*
Bachelor-Themen in Informatik,
DOI 10.1007/978-0-85729-299-5_8, 

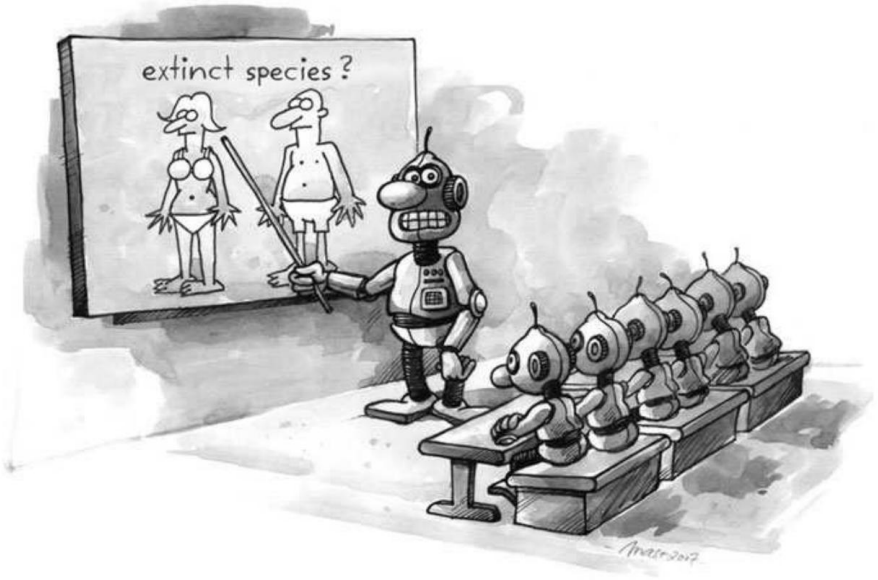



**Abb. 8.1 Überwachtes Lernen. . .**

**Diese Aufgaben sind jedoch recht einfach, da sie kaum mehr als das Speichern von Text in einer Datei umfassen. Daher ist das Auswendiglernen für die KI uninteressant. Im Gegensatz dazu erfolgt der Erwerb mathematischer Fähigkeiten meist nicht durch Auswendiglernen. Bei der Addition natürlicher Zahlen ist dies überhaupt nicht möglich, da es für jeden Term in der Summe x + y unendlich viele Werte gibt. Für jede Kombination der beiden Werte x und y müsste das Tripel (x, y, x + y) gespeichert werden, was unmöglich ist. Bei Dezimalzahlen ist dies völlig unmöglich. Dabei stellt sich die Frage: Wie lernen wir Mathematik? Die Antwort lautet: Der Lehrer erklärt den Prozess und die Schüler üben ihn an Beispielen, bis sie an neuen Beispielen keine Fehler mehr machen. Nach 50 Beispielen versteht der Schüler (hoffentlich) die Addition. Das heißt, nach nur 50 Beispielen kann er das Gelernte auf unendlich viele neue Beispiele anwenden, die bis zu diesem Zeitpunkt noch nicht gesehen wurden. Dieser Vorgang wird als *Generalisierung bezeichnet.* Wir beginnen mit einem einfachen Beispiel.**

***Beispiel 8.1*** **Ein Obstbauer möchte geerntete Äpfel automatisch in die Warenklassen A und B einteilen. Die Sortieranlage ist mit Sensoren ausgestattet, um zwei *Merkmale,* Größe und Farbe, zu messen und dann zu entscheiden, zu welcher der beiden Klassen der Apfel gehört. Dies ist eine typische *Klassifizierungsaufgabe.* Systeme, die Merkmalsvektoren in eine endliche Anzahl von Klassen unterteilen können, werden als *Klassifikatoren bezeichnet.***

**Zur Konfiguration der Maschine werden die Äpfel von einem Fachmann handverlesen, also klassifiziert. Anschließend werden die beiden Messungen zusammen mit ihrer Klassenbezeichnung in eine Tabelle eingetragen (Tabelle 8.1 auf Seite 163). Die Größe wird in Form des Durchmessers in Zentimetern und die Farbe durch einen Zahlenwert zwischen 0 (für Grün) und**

**Tabelle 8.1 Trainingsdaten für den Apfelsortieragenten**

| | | |
|---|---|---|
| **Größe [cm]** | **8863. . .** | |
| **Farbe** | **0,1 0,3 0,9 0,8 . . .** | |
| **Warenklasse** | **BAAB** | **. . .** |

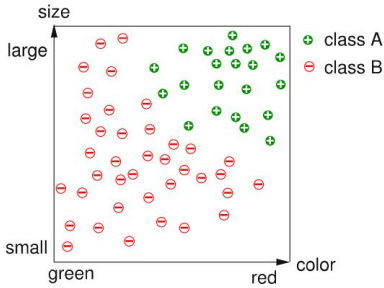
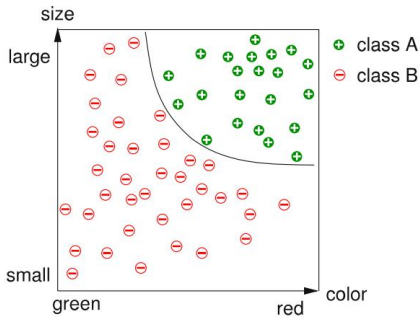



**Abb. 8.2 Apfelsortieranlage der BayWa-Firma in Kressbronn und Einordnung einiger Äpfel in die Warenklassen A und B im Objektraum (Foto: BayWa)**

**Abb. 8.3 Die in das Diagramm eingezeichnete *Kurve*** unterteilt die Klassen und kann dann auf beliebige neue Äpfel angewendet werden




**Eine Visualisierung der Daten ist als Punkte in einem Streudiagramm rechts in Abb. 8.2 aufgeführt.**

**Die Aufgabe beim maschinellen Lernen besteht darin, aus den gesammelten, klassifizierten Daten eine Funktion zu generieren, die aus den beiden Merkmalen Größe und Farbe den Klassenwert (A oder B) für einen neuen Apfel berechnet. In Abb. 8.3 ist eine solche Funktion durch die durch das Diagramm gezogene Trennlinie dargestellt. Alle Äpfel mit einem Merkmalsvektor unten links in der Linie werden in die Klasse B eingeordnet, alle anderen in die Klasse A.**

**In diesem Beispiel ist es noch sehr einfach, eine solche Trennlinie für die beiden Klassen zu finden. Es ist eindeutig eine schwierigere und vor allem viel weniger visualisierbare Aufgabe, wenn die zu klassifizierenden Objekte nicht nur durch zwei, sondern durch viele Merkmale beschrieben werden. In der Praxis werden üblicherweise 30 oder mehr Features verwendet. Für n Merkmale besteht die Aufgabe darin, eine n ÿ 1-dimensionale Hyperebene innerhalb des n-dimensionalen *Merkmalsraums*** zu finden , **die die Klassen möglichst gut unterteilt. Eine „gute“ Einteilung bedeutet, dass der Anteil falsch klassifizierter Objekte möglichst gering ist.**


**Abb. 8.4 Funktionsstruktur eines Lernagenten zur Apfelsortierung *(links)* und allgemein *(rechts)***

**Ein Klassifikator ordnet einen Merkmalsvektor einem Klassenwert zu. Hier gibt es eine feste, meist kleine Anzahl an Alternativen.** Die gewünschte Abbildung wird auch ***Zielfunktion*** **genannt . Wenn sich die Zielfunktion nicht auf einen endlichen Bereich abbilden lässt, handelt es sich nicht um eine Klassifikation, sondern um ein *Näherungsproblem.* Ein solches Näherungsproblem ist die Bestimmung des Marktwerts einer Aktie aus gegebenen Merkmalen. In den folgenden Abschnitten stellen wir mehrere Lernagenten für beide Arten von Zuordnungen vor.**

**Der Lernagent Wir können einen Lernagenten formal als eine Funktion beschreiben, die einen Merkmalsvektor auf einen diskreten Klassenwert oder allgemein auf eine reelle Zahl abbildet. Diese Funktion ist nicht programmiert, sondern entsteht bzw. verändert sich während der *Lernphase,* beeinflusst durch die *Trainingsdaten.* In Abb. 8.4 ist ein solcher Agent am Beispiel der Apfelsortierung dargestellt. Beim Lernen wird der Agent mit den bereits klassifizierten Daten aus Tabelle 8.1 auf Seite 163 gefüttert . Anschließend stellt der Agent eine möglichst gute Abbildung vom Merkmalsvektor auf den Funktionswert (z. B. Warenklasse) her.**

**Wir können nun versuchen, uns einer Definition des Begriffs „maschinelles Lernen“ anzunähern. Tom Mitchell [Mit97] gibt diese Definition:**

**„ Maschinelles *Lernen ist das Studium von Computeralgorithmen, die Automatisierung verbessern.“* *durch Erfahrung.***

**Darauf aufbauend geben wir**

> **Definition 8.1 Ein Agent ist ein lernender Agent, wenn er seine Leistung (gemessen an einem geeigneten Kriterium) bei neuen, unbekannten Daten im Laufe der Zeit verbessert (nachdem er viele Trainingsbeispiele gesehen hat).**

**Es ist wichtig, die Generalisierungsfähigkeit des Lernalgorithmus an unbekannten Daten, den *Testdaten, zu testen.* Andernfalls würde jedes System, das gerade die Trainingsdaten gespeichert hat, allein durch den Aufruf der gespeicherten Daten eine optimale Leistung erbringen. Ein lernender Agent wird durch folgende**
**Begriffe charakterisiert: *Aufgabe:* Die Aufgabe des lernenden Algorithmus besteht darin, eine Abbildung zu lernen. Dies könnte beispielsweise die Zuordnung der Größe und Farbe eines Apfels zu seiner Handelsklasse sein, aber auch die Zuordnung der 15 Symptome eines Patienten zur Entscheidung, ob sein Blinddarm entfernt werden soll oder nicht.**


Abb. 8.5 Data Mining

*Variabler Agent* (genauer gesagt eine Klasse von Agenten): Hier müssen wir entscheiden, mit welchem Lernalgorithmus gearbeitet wird. Wenn dies gewählt wurde, wird somit die Klasse aller lernbaren Funktionen bestimmt.

*Trainingsdaten* (Erfahrung): Die Trainingsdaten (Probe) enthalten das Wissen, das der Lernalgorithmus extrahieren und lernen soll. Bei der Auswahl der Trainingsdaten muss darauf geachtet werden, dass es sich um eine repräsentative Stichprobe für die zu erlernende Aufgabe h

*Testdaten:* wichtig für die Beurteilung, ob der trainierte Agent die Trainingsdaten gut auf neue Daten verallgemeinern kann.

*Leistungsmaß:* Bei der Apfelsortieranlage die Anzahl der korrekt klassifizierten Äpfel. Wir benötigen es, um die Qualität eines Agenten zu testen. Die Kenntnis des Leistungsmaßes ist in der Regel viel einfacher als die Kenntnis der Funktion des Agenten. Beispielsweise ist es einfach, die Leistung (Zeit) eines 10.000-Meter-Läufers zu messen. Dies bedeutet jedoch keineswegs, dass der Schiedsrichter, der die Zeit misst, genauso schnell laufen kann. Der Schiedsrichter weiß nur, wie er die Leistung messen soll, nicht aber die „Funktion“ des Agenten, dessen Leistung er misst.

Was ist Data Mining? Die Aufgabe einer lernenden Maschine ist es, aus Trainingsdaten Wissen zu extrahieren. Oftmals möchte der Entwickler oder Anwender, dass die lernende Maschine das extrahierte Wissen auch für Menschen lesbar macht. Noch besser ist es, wenn der Entwickler das Wissen sogar ändern kann. Der Prozess der Induktion von Entscheidungsbäumen in Abschn. 8.4 ist ein Beispiel für eine solche Methode.

Ähnliche Herausforderungen ergeben sich aus dem elektronischen Geschäftsverkehr und dem Wissensmanagement. Hier zeigt sich ein klassisches Problem: Von den Aktionen der Besucher auf seinem Webportal,

Der Inhaber eines Internetunternehmens möchte eine Beziehung zwischen dem herstellen
Eigenschaften eines Kunden und die Produktklasse, die für ihn interessant ist
Kunde. Dann kann ein Verkäufer kundenspezifische Anzeigen schalten. Das
Dies wird anschaulich auf www.amazon.com demonstriert, wo dem Kunden Produkte empfohlen werden,
die denen des vorherigen Besuchs ähneln. In vielen
In den Bereichen Werbung und Marketing sowie im Customer Relationship Management (CRM) kommen
Data-Mining-Techniken zum Einsatz. Immer dann, wenn es um große Mengen geht
Je mehr Daten vorhanden sind, desto mehr kann man versuchen, diese Daten für die Kundenanalyse zu nutzen
Präferenzen, um kundenspezifische Werbung anzuzeigen. Das aufstrebende Feld
Diesem Zweck ist die Studie des Präferenzlernens gewidmet.

Der Prozess der Wissensgewinnung aus Daten sowie deren Darstellung
und Anwendung wird als *Data Mining bezeichnet.* Die verwendeten Methoden werden in der Regel übernommen
aus Statistik oder maschinellem Lernen und sollte auf sehr große Bereiche anwendbar sein
Datenmengen zu angemessenen Kosten.

Im Rahmen der Informationsbeschaffung, beispielsweise im Internet oder in einem Intranet, spielt
Text Mining eine immer wichtigere Rolle. Typische Aufgaben sind das Auffinden ähnlicher Texte in einer
Suchmaschine oder das Klassifizieren von Texten, die z.B
wird in Spamfiltern für E-Mails angewendet. In Abschn. 8.6.1 werden wir das weit verbreitete vorstellen
Naiver Bayes-Algorithmus zur Klassifizierung von Text. Eine relativ neue Herausforderung für
Unter Data Mining versteht man die Extraktion struktureller, statischer und dynamischer Informationen aus
Diagrammstrukturen wie soziale Netzwerke, Verkehrsnetzwerke oder Internetverkehr.

Da sich die beiden beschriebenen Aufgaben des maschinellen Lernens und des Data Mining
weitgehend ähneln, sind die grundlegenden Methoden, die in beiden Bereichen zum Einsatz kommen,
größtenteils identisch. Daher wird in der Beschreibung der Lernalgorithmen nicht unterschieden
zwischen maschinellem Lernen und Data Mining.

Aufgrund der enormen kommerziellen Auswirkungen von Data-Mining-Techniken gibt es diese mittlerweile
viele ausgefeilte Optimierungen und eine ganze Reihe leistungsstarker Data-Mining-Systeme,
die eine große Palette praktischer Werkzeuge zur Wissensextraktion bieten
Daten. Ein solches System wird in Abschn. 2.1 vorgestellt. 8.8.

## 8.1 Datenanalyse

Statistiken bieten eine Reihe von Möglichkeiten, Daten mit einfachen Parametern zu beschreiben. Aus
Aus diesen wählen wir einige aus, die für die Trainingsanalyse besonders wichtig sind
Daten und testen Sie diese an einer Teilmenge der LEXMED- Daten aus Abschn. 7.3. In diesem Beispiel
Datensatz, die Symptome x1,...,x15 von N = 473 Patienten, prägnant beschrieben in Tabelle 8.2 auf Seite
167, sowie die Klassenbezeichnung – also die Diagnose (Blinddarmentzündung).
positiv/negativ) – werden aufgelistet. Patient Nummer eins wird zum Beispiel von beschrieben
Vektor

$$^{1}x_{-} = (26, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 37{,}9, 38{,}8, 23100, 0, 1)$$

Tabelle 8.2 Beschreibung von Variablen x1,...,x16.

Ein etwas anderes Formalisierung wurde in verwendet

Tabelle 7.2 auf Seite 133

| Var. Num. | Beschreibung | Werte |
|---|---|---|
| 1 | Alter | Kontinuierlich |
| 2 | Geschlecht (1=männlich, 2=weiblich) | 1, 2 |
| 3 | Schmerzquadrant 1 | 0, 1 |
| 4 | Schmerzquadrant 2 | 0, 1 |
| 5 | Schmerzquadrant 3 | 0, 1 |
| 6 | Schmerzquadrant 4 | 0, 1 |
| 7 | Lokaler Muskelschutz | 0, 1 |
| 8 | Generalisierte Muskelprotektion | 0, 1 |
| 9 | Rebound-Zärtlichkeit | 0, 1 |
| 10 | Schmerzen beim Klopfen | 0, 1 |
| 11 | Schmerzen bei rektaler Untersuchung | 0, 1 |
| 12 | Axiale Temperatur | Kontinuierlich |
| 13 | Rektale Temperatur | Kontinuierlich |
| 14 | Leukozyten | Kontinuierlich |
| 15 | Diabetes Mellitus | 0, 1 |
| 16 | Appendizitis | 0, 1 |

**und Patient Nummer zwei von**

$$^{2}x_{\_} = (17, 2, 0, 0, 1, 0, 1, 0, 1, 1, 0, 36{,}9, 37{,}4, 8100, 0, 0)$$

**Patient Nummer zwei hat den Leukozytenwert** $x_{14}^{2} = 8100$.

**Für jede Variable xi** ist **ihr Durchschnitt** $\bar{x}i$ **definiert als**

$$\bar{x}i := \frac{1}{N} \sum_{p=1}^{N} x_{\_}^{P}$$

**und die Standardabweichung si als Maß für seine durchschnittliche Abweichung vom Durchschnitt Wert als**

$$si := \sqrt{\frac{1}{N \ddot{y} 1} \sum_{p=1}^{N} (x_{\_}^{p} \ddot{y} \bar{x}i)^{2}}.$$

Für die Analyse mehrdimensionaler Daten ist **die Frage wichtig, ob zwei Variablen xi und xj statistisch abhängig (korreliert) sind. Zum Beispiel die Kovarianz**

$$\ddot{y}ij = \frac{1}{N \ddot{y} 1} \sum_{p=1}^{N} (x_{\_}^{p} \ddot{y} \bar{x}i)(x_{J}^{p} \ddot{y} \bar{x}j)$$

**Tabelle 8.3 Korrelationsmatrix für die 16 in 473 Fällen gemessenen Blinddarmentzündungsvariablen**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | ÿ0,009 | 0,14 | 0,037 | ÿ0,096 | 0,12 | 0,018 | 0,051 | ÿ0,034 | ÿ0,041 | 0,034 | 0,037 | 0,05 | ÿ0,037 | 0,37 | 0,012 |
| ÿ0,009 | 1. | ÿ0,0074 | ÿ0,019 | ÿ0,06 | 0,063 | ÿ0,17 | 0,0084 | ÿ0,17 | ÿ0,14 | ÿ0,13 | ÿ0,017 | ÿ0,034 | ÿ0,14 | 0,045 | ÿ0,2 |
| 0,14 | ÿ0,0074 | 1. | 0,55 | ÿ0,091 | 0,24 | 0,13 | 0,24 | 0,045 | 0,18 | 0,028 | 0,02 | 0,045 | 0,03 | 0,11 | 0,045 |
| 0,037 | ÿ0,019 | 0,55 | 1. | ÿ0,24 | 0,33 | 0,051 | 0,25 | 0,074 | 0,19 | 0,087 | 0,11 | 0,12 | 0,11 | 0,14 | ÿ0,0091 |
| ÿ0,096 | ÿ0,06 | ÿ0,091 | ÿ0,24 | 1. | 0,059 | 0,14 | 0,034 | 0,14 | 0,049 | 0,057 | 0,064 | 0,058 | 0,11 | 0,017 | 0,14 |
| 0,12 | 0,063 | 0,24 | 0,33 | 0,059 | 1. | 0,071 | 0,19 | 0,086 | 0,15 | 0,048 | 0,11 | 0,12 | 0,063 | 0,21 | 0,053 |
| 0,018 | ÿ0,17 | 0,13 | 0,051 | 0,14 | 0,071 | 1. | 0,16 | 0,4 | 0,28 | 0,2 | 0,24 | 0,36 | 0,29 | ÿ0,0001 | 0,33 |
| 0,051 | 0,0084 | 0,24 | 0,25 | 0,034 | 0,19 | 0,16 | 1. | 0,17 | 0,23 | 0,24 | 0,19 | 0,24 | 0,27 | 0,083 | 0,084 |
| ÿ0,034 | ÿ0,17 | 0,045 | 0,074 | 0,14 | 0,086 | 0,4 | 0,17 | 1. | 0,53 | 0,25 | 0,19 | 0,27 | 0,27 | 0,026 | 0,38 |
| ÿ0,041 | ÿ0,14 | 0,18 | 0,19 | 0,049 | 0,15 | 0,28 | 0,23 | 0,53 | 1. | 0,24 | 0,15 | 0,19 | 0,23 | 0,02 | 0,32 |
| 0,034 | ÿ0,13 | 0,028 | 0,087 | 0,057 | 0,048 | 0,2 | 0,24 | 0,25 | 0,24 | 1. | 0,17 | 0,17 | 0,22 | 0,098 | 0,17 |
| 0,037 | ÿ0,017 | 0,02 | 0,11 | 0,064 | 0,11 | 0,24 | 0,19 | 0,19 | 0,15 | 0,17 | 1. | 0,72 | 0,26 | 0,035 | 0,15 |
| 0,05 | ÿ0,034 | 0,045 | 0,12 | 0,058 | 0,12 | 0,36 | 0,24 | 0,27 | 0,19 | 0,17 | 0,72 | 1. | 0,38 | 0,044 | 0,21 |
| ÿ0,037 | ÿ0,14 | 0,03 | 0,11 | 0,11 | 0,063 | 0,29 | 0,27 | 0,27 | 0,23 | 0,22 | 0,26 | 0,38 | 1. | 0,051 | 0,44 |
| 0,37 | 0,045 | 0,11 | 0,14 | 0,017 | 0,21 | ÿ0,0001 | 0,083 | 0,026 | 0,02 | 0,098 | 0,035 | 0,044 | 0,051 | 1. | ÿ0,0055 |
| 0,012 | ÿ0,2 | 0,045 | ÿ0,0091 | 0,14 | 0,053 | 0,33 | 0,084 | 0,38 | 0,32 | 0,17 | 0,15 | 0,21 | 0,44 | ÿ0,0055 | 1. |

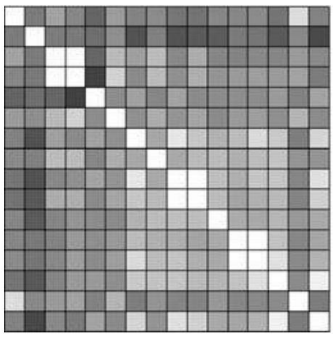
gibt Auskunft darüber. In dieser Summe liefert der Summand einen positiven Eintrag für der p-te Datenvektor genau dann, wenn die Abweichungen der i-ten und j-ten Komponenten von im Durchschnitt haben beide das gleiche Vorzeichen. Wenn sie unterschiedliche Vorzeichen haben, dann ist der Eintrag Negativ. Daher sollte die Kovarianz ÿ12,13 der beiden unterschiedlichen Fieberwerte betragen ganz klar positiv.

Allerdings hängt die Kovarianz auch vom Absolutwert der Variablen ab, was den Vergleich der Werte erschwert. Um den Abschluss vergleichen zu können Als Abhängigkeit bei mehreren Variablen definieren wir daher die *Korrelation Koeffizient*

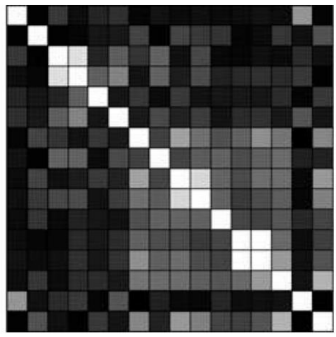
$$Kij = \frac{ÿij}{si \cdot sj}$$

für zwei Werte xi und xj , was nichts anderes als eine normalisierte Kovarianz ist. Die Matrix K aller Korrelationskoeffizienten enthält Werte zwischen ÿ1 und 1, ist symmetrisch, und alle seine Diagonalelemente haben den Wert 1. Die Korrelationsmatrix für alle 16 Variablen sind in Tabelle 8.3 angegeben.

Diese Matrix wird etwas besser lesbar, wenn wir sie als Dichte darstellen Handlung. Anstelle der Zahlenwerte die Matrixelemente in Abb. 8.6 auf Seite 169 sind mit Grauwerten gefüllt. Im rechten Diagramm sind die absoluten Werte dargestellt. Daher Wir können sehr schnell erkennen, welche Variablen eine schwache oder starke Abhängigkeit aufweisen. Wir Man erkennt zum Beispiel, dass die Variablen 7, 9, 10 und 14 die stärkste Korrelation mit der Klassenvariable *Blinddarmentzündung* aufweisen und daher für die wichtiger sind Diagnose als die andere Variable. Wir sehen jedoch auch, dass die Variablen 9 und 10 sind stark korreliert. Dies könnte bedeuten, dass einer dieser beiden Werte potenziell vorhanden ist ausreichend für die Diagnose.



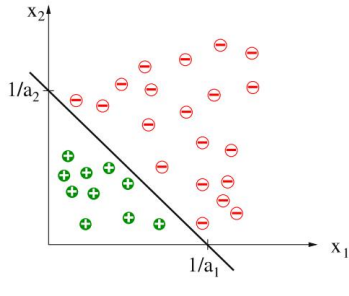
$K_{ij} = -1$: black, $K_{ij} = 1$: white



$|K_{ij}| = 0$: black, $|K_{ij}| = 1$: white

**Abb. 8.6 Die Korrelationsmatrix als Häufigkeitsgraph. Im *linken Diagramm* steht *Dunkel* für Negativ und *Hell* für Positiv. Im *rechten Bild* wurden die absoluten Werte aufgeführt. Dabei bedeutet *Schwarz* Kij ÿ 0 (unkorreliert) und *Weiß* |Kij | ÿ 1 (stark korreliert)**

**Abb. 8.7 Ein linear trennbarer zweidimensionaler Datensatz. Die Gleichung für die Teilungsgerade lautet a1x1 + a2x2 = 1**




## 8.2 Das Perzeptron, ein linearer Klassifikator

Im Beispiel der Sortierung von Äpfeln ist in Abb. 8.3 auf Seite 163 eine gekrümmte Trennlinie zwischen den beiden Klassen eingezeichnet. Ein einfacherer Fall ist in Abb. 8.7 dargestellt. Hier können die zweidimensionalen Trainingsbeispiele durch eine Gerade getrennt werden. Wir nennen einen solchen Satz von Trainingsdaten *linear trennbar.* In n Dimensionen wird zur Trennung eine Hyperebene benötigt. Dies stellt einen linearen Unterraum der Dimension n ÿ 1 dar.

Weil jede (n ÿ 1)-dimensionale Hyperebene in der R- $^{N}$ kann durch ein beschrieben werden Gleichung

$$\sum_{i=1}^{N} a_i x_i = \ddot{y}$$

Es ist sinnvoll, die lineare Trennbarkeit wie folgt zu definieren.

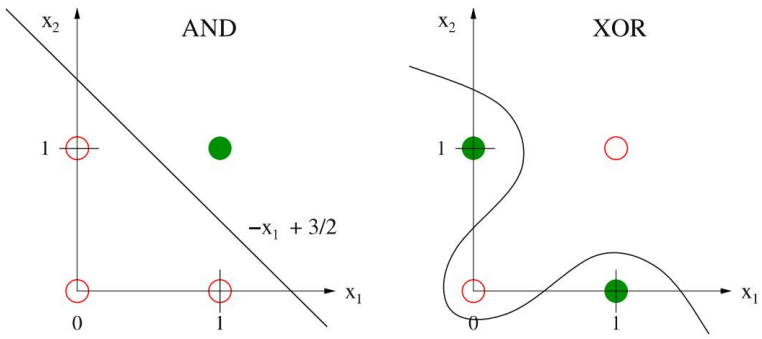



**Abb. 8.8 Die boolesche Funktion AND ist linear separierbar, XOR jedoch nicht ( ˆ= wahr, ○ ˆ= falsch)**

**Definition 8.2 Zwei Mengen M1 ÿ $^N$ und M2 ÿ R $^N$ heißen *linear separierbar* R , wenn reelle Zahlen a1,...,an, ÿ existieren mit**

$$\sum_{i=1}^{N} a_i x_i > \ddot{y} \text{ für alle } x \ddot{y} M1 \text{ und } \sum_{i=1}^{N} a_i x_i \ddot{y} \ddot{y} \text{ für alle } x \ddot{y} M2.$$

**Der Wert ÿ wird als Schwelle bezeichnet.**

**In Abb. 8.8 sehen wir, dass die AND-Funktion linear separierbar ist, die XOR-Funktion jedoch nicht. Für AND trennt beispielsweise die Linie ÿx1 + 3/2 wahre und falsche Interpretationen der Formel x1 ÿx2. Im Gegensatz dazu hat die XOR-Funktion keine gerade Trennlinie. Offensichtlich hat die XOR-Funktion in dieser Hinsicht eine komplexere Struktur als die AND-Funktion.**

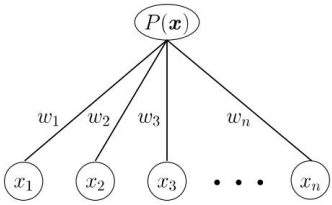
**Mit dem Perzeptron stellen wir einen sehr einfachen Lernalgorithmus vor, der sep bilden linear trennbare Mengen.**

**Definition 8.3 Sei w = (w1,...,wn) ÿ R ein $^N$ sei ein Gewichtsvektor und x ÿ $^N$ Eingabevektor. Ein *Perzeptron* stellt eine Funktion P dar: R $^N$ R ÿ {0, 1} was entspricht der folgenden Regel:**

$$P(x) = \begin{cases} 1 \text{ wenn } wx = \sum_{i=1}^{N} w_i x_i > 0, \\ 0 \text{ sonst.} \end{cases}$$

**Das Perzeptron [Ros58, MP69] ist ein sehr einfacher Klassifizierungsalgorithmus. Es entspricht einem zweischichtigen neuronalen Netzwerk mit Aktivierung durch eine Schwellenwertfunktion, dargestellt in Abb. 8.9 auf Seite 171. Wie in Kap. In 9 repräsentiert jeder Knoten im Netzwerk ein Neuron und jede Kante eine Synapse. Im Moment werden wir jedoch nur**

Abb. 8.9 Grafische Darstellung eines Perzeptrons als zweischichtiges neuronales Netzwerk




Betrachten Sie das Perzeptron als Lernagent, das heißt als mathematische Funktion, die einen Merkmalsvektor auf einen Funktionswert abbildet. Dabei werden die Eingabevariablen xi als Features bezeichnet .

Wie wir in der Formel $\sum_{i=1}^{N} w_i x_i > 0$ sehen können, werden alle Punkte x oberhalb der Hyperebene $\sum_{i=1}^{N} w_i x_i = 0$ als positiv (P (x) = 1) und alle anderen als negativ (P (x) klassifiziert ) = 0). Die trennende Hyperebene geht durch den Ursprung, weil ÿ = 0.
Mit einem kleinen Trick zeigen wir, dass das Fehlen einer willkürlichen Schwelle keine Einschränkung der Macht darstellt. Zunächst wollen wir jedoch einen einfachen Lernalgorithmus für das Perzeptron vorstellen.

## 8.2.1 Die Lernregel

Mit der Notation M+ und Mÿ für die Sätze positiver bzw. negativer Trainingsmuster lautet die Perzeptron-Lernregel [MP69]

```
PERCEPTRONLEARNING[M+,Mÿ] w =
beliebiger Vektor reeller Zahlen
Wiederholen
    Für alle x ÿ M+
        Wenn wx ÿ 0, dann ist w = w + x
    Für alle x ÿ Mÿ
        Wenn wx > 0, dann ist w = w ÿ x
Bis alle x ÿ M+ ÿ Mÿ korrekt klassifiziert sind
```

Das Perzeptron sollte für alle x ÿ M+ den Wert 1 ausgeben . Nach Definition 8.3 auf Seite 170 ist dies der Fall, wenn wx > 0. Ist dies nicht der Fall, wird x zum Gewichtsvektor w addiert, wodurch der Gewichtsvektor genau in die richtige Richtung geändert wird.
Wir sehen dies, wenn wir das Perzeptron auf den geänderten Vektor w + x anwenden, weil

$$2\ (w + x) \cdot x = wx + x \qquad \cdot$$

Wenn dieser Schritt oft genug wiederholt wird, wird der Wert wx irgendwann positiv, wie es sein sollte. Analog sehen wir, dass für negative Trainingsdaten die

perceptron errechnet einen immer kleineren Wert

$$(w \ddot{y} x) \cdot x = wx \ddot{y} x^2$$

was irgendwann negativ wird.2

*Beispiel 8.2* Ein Perzeptron soll auf den Mengen M+ = {(0, 1,8), (2, 0,6)} und Mÿ = {(ÿ1,2, 1,4), (0,4,ÿ1)} trainiert werden. w = (1, 1) wurde als anfänglicher Gewichtsvektor verwendet. Die Trainingsdaten und die durch den Gewichtsvektor wx = x1 + x2 = 0 definierte Linie sind in Abb. 8.10 auf Seite 173 im ersten Bild in der oberen Reihe dargestellt. Zusätzlich wird der Gewichtsvektor als gestrichelte Linie eingezeichnet. Da wx = 0 ist, ist dies orthogonal zur Gerad

In der ersten Iteration durch die Schleife des Lernalgorithmus ist das nur falsch Das klassifizierte Trainingsbeispiel ist (ÿ1,2, 1,4), weil

$$(\ddot{y}1{,}2,\ 1{,}4) \cdot \begin{matrix} \\ 1\ 1 \end{matrix} = 0{,}2 > 0.$$

Daraus ergibt sich w = (1, 1)ÿ(ÿ1,2, 1,4) = (2,2,ÿ0,4), wie im zweiten Bild in der oberen Reihe in Abb. 8.10 auf Seite 173 gezeichnet. Die anderen Bilder zeigen, wie es danach geht insgesamt fünf Änderungen, die Trennlinie verläuft zwischen den beiden Klassen. Das Perzeptron klassifiziert somit alle Daten korrekt. Wir sehen im Beispiel deutlich, dass jeder falsch klassifizierte Datenpunkt aus M+ den Gewichtsvektor w in seine Richtung „zieht“ und jeder falsch klassifizierte Punkt aus Mÿ den Gewichtsvektor in die entgegengesetzte Richtung „drückt“.

Es wurde gezeigt [MP69], dass das Perzeptron für lineare Separierungen immer konvergiert Ackerdaten. Wir haben

**Satz 8.1** ***Seien die Klassen*** **M+** ***und*** **Mÿ** durch eine Hyperebene **wx = 0** ***linear trennbar. Dann*** konvergiert **PERCEPTRONLEARNING** ***für jede Initialisierung des Vektors*** **w.** ***Das Perzeptron*** **P** ***mit dem so berechneten Gewichtsvektor teilt die Klassen*** **M+** ***und*** **Mÿ,** ***das heißt:***

$$P(x) = 1 \ddot{y} x \ddot{y} M+$$

***Und***

$$P(x) = 0 \ddot{y} x \ddot{y} M\ddot{y}.$$

Wie wir in Beispiel 8.2 deutlich sehen können, können Perzeptrone wie oben definiert keine beliebigen linear trennbaren Mengen teilen, sondern nur solche, die durch eine Linie durch den Ursprung oder in $\mathbb{R}^N$ durch eine Hyperebene durch den Ursprung teilbar sind, da der konstante Term ÿ ist fehlt in der Gleichung $\sum_{i=1}^{N} wixi = 0$.

2Vorsicht! Dies ist kein Konvergenzbeweis für die Perzeptron-Lernregel. Es zeigt nur, dass das Perzeptron konvergiert, wenn der Trainingsdatensatz aus einem einzelnen Beispiel besteht.

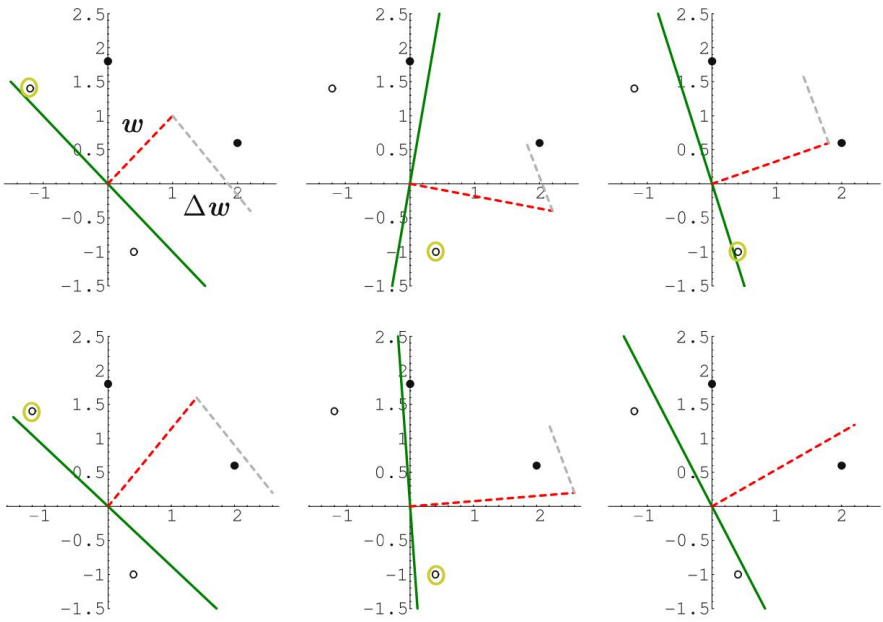



**Abb. 8.10 Anwendung der Perzeptron-Lernregel auf zwei positive (•) und zwei negative (ÿ) Datenpunkte. Die *durchgezogene Linie* zeigt die aktuelle Trennlinie wx = 0. Die *orthogonale gestrichelte Linie* ist der Gewichtsvektor w und die *zweite gestrichelte Linie* der hinzuzufügende Änderungsvektor w = x oder w = ÿx, der aus dem aktuell aktiven Datenpunkt berechnet wird *grau* umrandet**

**Mit dem folgenden Trick können wir den konstanten Term erzeugen. Wir halten die letzte Komponente xn des Eingabevektors x konstant und setzen sie auf den Wert 1. Nun funktioniert das Gewicht wn =: ÿÿ wie ein Schwellenwert, weil**

$$\sum_{i=1}^{N} wixi = \sum_{i=1}^{n\ddot{y}1} wixi \ddot{y} \ddot{y} > 0 \ddot{y} \sum_{i=1}^{n\ddot{y}1} wixi > \ddot{y}.$$

**Ein solcher konstanter Wert xn = 1 in der Eingabe wird als *Bias-Einheit bezeichnet.* Da das damit verbundene Gewicht eine ständige Verschiebung der Hyperebene bewirkt, passt der Begriff „Bias“ gut.**

**Bei der Anwendung des Perzeptron-Lernalgorithmus wird an den Trainingsdatenvektor ein Bit mit dem konstanten Wert 1 angehängt. Wir beobachten, dass das Gewicht wn oder der Schwellenwert ÿ ist, wird während des Lernprozesses gelernt.**

**Nun wurde gezeigt, dass ein Perzeptron Pÿ : $R^{n\ddot{y}1}$ ÿ {0, 1}**

$$P\ddot{y}(x1,...,xn\ddot{y}1) = \begin{cases} 1 \text{ wenn } \sum_{i=1}^{n\ddot{y}1} wixi > \ddot{y}, \\ 0 \text{ sonst} \end{cases} \quad (8.1)$$

**Mit einem beliebigen Schwellenwert kann durch ein Perzeptron P : $R^{N}$ der ÿ {0, 1} mit Schwellenwert 0 simuliert werden. Vergleichen wir (8.1) mit der Definition von linear trennbar, dann sehen wir, dass beide Aussagen äquivalent sind. Zusammenfassend haben wir Folgendes gezeigt:**

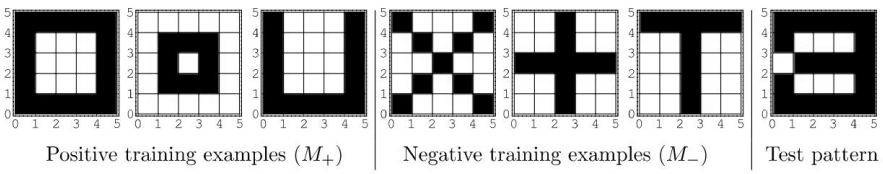
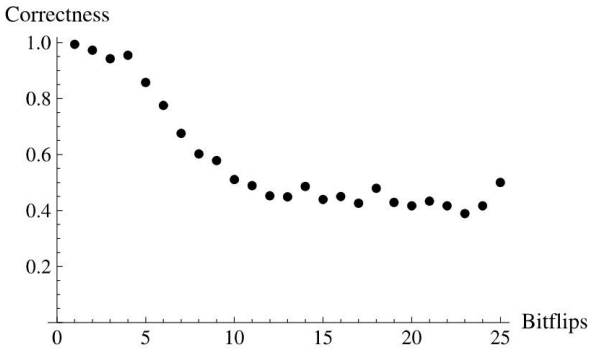



**Abb. 8.11 Die sechs für das Training verwendeten Muster. Das *gesamte rechte Muster* ist eines der 22 Testmuster für das *erste Muster* mit einer Folge von vier invertierten Bits**

**Abb. 8.12 Relative Korrektheit des Perzeptrons als Funktion der Anzahl invertierter Bits in den Testdaten**




**Satz 8.2** ***Eine Funktion*** **f : R ÿ {0, 1}$^N$** ***kann genau dann durch ein Perzeptron dargestellt werden, wenn die beiden Mengen positiver und negativer Eingabevektoren linear trennbar sind.***

***Beispiel 8.3*** **Wir trainieren nun ein Perzeptron mit einem Schwellenwert auf sechs einfachen, grafischen Binärmustern, dargestellt in Abb. 8.11, mit jeweils 5 × 5 Pixeln.**

**Die Trainingsdaten können von PERCEPTRONLEARNING in vier Iterationen über alle Muster gelernt werden. Muster mit einer variablen Anzahl invertierter Bits, die als Rauschen eingeführt werden, werden verwendet, um die Generalisierungsfähigkeit des Systems zu testen. Die invertierten Bits im Testmuster liegen jeweils der Reihe nach hintereinander vor. In Abb. 8.12 ist der Prozentsatz korrekt klassifizierter Muster als Funktion der Anzahl falscher Bits aufgetrag**

**Nach etwa fünf aufeinanderfolgenden invertierten Bits nimmt die Korrektheit stark ab, was angesichts der Einfachheit des Modells nicht verwunderlich ist. Im nächsten Abschnitt stellen wir einen Algorithmus vor, der in diesem Fall deutlich besser abschneidet.**

## 8.2.2 Optimierung und Ausblick

**Als einer der einfachsten auf neuronalen Netzwerken basierenden Lernalgorithmen kann das zweischichtige Per-Zeptron nur linear trennbare Klassen unterteilen. In Abschn. 9.5 werden wir sehen, dass mehrschichtige Netzwerke deutlich leistungsfähiger sind. Trotz seiner einfachen Str**

Das Perzeptron in der dargestellten Form konvergiert sehr langsam. Es kann durch Normalisierung des gewichtsverändernden Vektors beschleunigt werden. Die Formeln w = w ± x werden durch w = w ± x/|x| ersetzt. Dadurch hat jeder Datenpunkt beim Lernen das gleiche Gewicht, unabhängig von seinem Wert.

Die Konvergenzgeschwindigkeit hängt stark von der Initialisierung des Vektors w ab. Im Idealfall müsste es überhaupt nicht geändert werden und der Algorithmus würde nach einer Iteration konvergieren. Durch den Einsatz der heuristischen Initialisierung können wir diesem Ziel näher kommen

$$w0 = \frac{x \ddot{y}}{x\ddot{y}M+} \quad \frac{}{x\ddot{y}M\ddot{y}} X,$$

was wir in Aufgabe 8.5 auf Seite 217 genauer untersuchen werden .

Vergleicht man die Perzeptronformel mit der in Abschn. 7.3.1 sehen wir sofort ihre Äquivalenz. Darüber hinaus ist das Perzeptron als einfachstes neuronales Netzwerkmodell äquivalent zu Naive Bayes, der einfachsten Art von Bayes'schen Netzwerken (siehe Übung 8.16 auf Seite 219). Somit haben offenbar mehrere sehr unterschiedliche Klassifikationsalgorithmen einen gemeinsamen Ursprung.

Im Kap. 9 lernen wir eine Verallgemeinerung des Perzeptrons in Form des Backpropagation-Algorithmus kennen, der nichtlinear trennbare Mengen durch die Verwendung mehrerer Schichten teilen kann und über eine bessere Lernregel verfügt.

## 8.3 Die Nearest-Neighbor-Methode

Für ein Perzeptron wird das in den Trainingsdaten vorhandene Wissen extrahiert und in komprimierter Form in den Gewichten wi gespeichert . Dadurch gehen Informationen über die Daten verloren. Genau das ist jedoch dann erwünscht, wenn das System von den Trainingsdaten auf neue Daten verallgemeinern soll. Die Generalisierung ist in diesem Fall ein zeitintensiver Prozess mit dem Ziel, eine kompakte Darstellung der Daten in Form einer Funktion zu finden, die neue Daten möglichst gut klassifiziert.

Das Speichern aller Daten durch einfaches Speichern ist etwas ganz anderes. Hier ist das Lernen äußerst einfach. Allerdings lässt sich das gespeicherte Wissen, wie bereits erwähnt, nicht so einfach auf neue, unbekannte Beispiele übertragen. Ein solcher Ansatz ist beispielsweise für das Erlernen des Skifahrens sehr ungeeignet. Ein Anfänger kann nie ein guter Skifahrer werden, indem er sich nur Videos von guten Skifahrern ansieht. Offensichtlich passiert bei der automatischen Ausführung solcher Lernbewegungen etwas Ähnliches wie beim Perzeptron. Nach ausreichend langer Übung wird das in Trainingsbeispielen gespeicherte Wissen in eine interne Repräsentation im Gehirn umgewandelt.

Es gibt jedoch gelungene Beispiele des Auswendiglernens, bei denen auch eine Verallgemeinerung möglich ist. Während der Diagnose eines schwierigen Falles könnte ein Arzt versuchen, sich an ähnliche Fälle aus der Vergangenheit zu erinnern. Wenn sein Gedächtnis in Ordnung ist, könnte er auf diesen Fall stoßen, ihn in seinen Akten nachschlagen und schließlich zu einer ähnlichen Diagnose kommen. Für dieses Vorgehen muss der Arzt ein gutes Gespür für *Ähnlichkeiten haben,* um sich an den ähnlichsten Fall zu erinnern. Wenn er dies gefunden hat, muss er sich fragen, ob es ähnlich genug ist, um dieselbe Diagnose zu rechtfertigen.

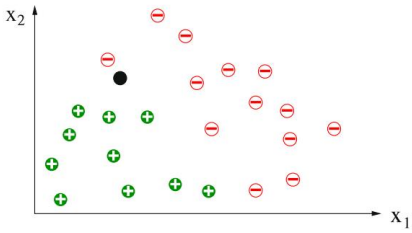



**Abb. 8.13 In diesem Beispiel mit negativen und positiven Trainingsbeispielen gruppiert die Methode des nächsten Nachbarn den neuen, schwarz markierten Punkt in die negative Klasse**

**Was bedeutet Ähnlichkeit in dem formalen Kontext, den wir konstruieren? Wir stellen die Trainingsbeispiele wie gewohnt in einem mehrdimensionalen Merkmalsraum dar und definieren: *Je kleiner ihr Abstand im Merkmalsraum, desto ähnlicher sind sich zwei Beispiele.***

**Wir wenden diese Definition nun auf das einfache zweidimensionale Beispiel aus Abb. 8.13 an. Hier ist der nächste Nachbar des schwarzen Punktes ein negatives Beispiel. Somit wird es der negativen Klasse zugeordnet.**

**Der Abstand d(x, y) zwischen zwei Punkten x ÿ** $\mathbb{R}^N$ **und y ÿ** $\mathbb{R}^N$ **kann zum Beispiel sein gemessen durch den euklidischen Abstand**

$$d(x, y) = |x \ddot{y} y| = \sqrt{\sum_{i=1}^{N} (xi \ddot{y} yi)^2}.$$

**Da es neben dieser noch viele andere Distanzmetriken gibt, ist es sinnvoll, über Alternativen für eine konkrete Anwendung nachzudenken. In vielen Anwendungen sind bestimmte Funktionen wichtiger als andere.** Daher ist es oft sinnvoll, die Features durch Gewichte **wi unterschiedlich zu skalieren . Die Formel lautet dann**

$$dw(x, y) = |x \ddot{y} y| = \sqrt{\sum_{i=1}^{N} wi(xi \ddot{y} yi)^2}.$$

**Das folgende einfache** Programm ***zur Klassifizierung des nächsten Nachbarn*** **durchsucht die Trainingsdaten nach dem nächsten Nachbarn t zum neuen Beispiel s und klassifiziert dann s genau 3 wie t.**

```
NEARESTNEIGHBOR[M+,Mÿ,s] t =
    argminxÿM+ÿMÿ {d(s, x)}
    Wenn t ÿ M+, dann Return („+“)
                 Else Return(„–“)
```

**3Die Funktionale argmin und argmax bestimmen, ähnlich wie min und max, das Minimum bzw. Maximum einer Menge oder Funktion. Sie geben jedoch nicht den Wert des Maximums oder Minimums zurück, sondern die Position, also das Argument, in dem das Extremum auftritt.**

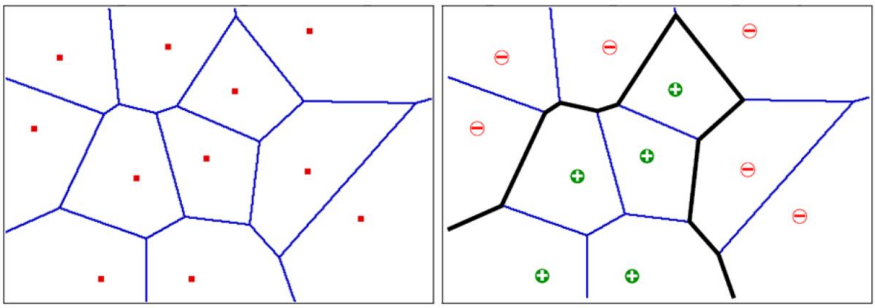
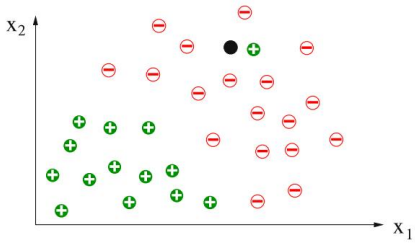


**Abb. 8.14 Eine Menge von Punkten zusammen mit ihrem Voronoi-Diagramm *(links)* und der erzeugten *Trennlinie*** für die beiden Klassen **M+ und Mÿ**

Abb. 8.15 Die Methode „Nächster Nachbar“ ordnet den neuen, ***schwarz*** **markierten Punkt der falschen (positiven) Klasse zu, da der nächste Nachbar höchstwahrscheinlich falsch klassifiziert ist**




**Im Gegensatz zum Perzeptron erzeugt die Methode des nächsten Nachbarn keine Linie, die die Trainingsdatenpunkte teilt. Es gibt jedoch sicherlich eine imaginäre Linie, die die beiden Klassen trennt. Wir können dies erzeugen, indem wir zunächst das sogenannte *Voronoi-Diagramm erstellen.* Im Voronoi-Diagramm ist jeder Datenpunkt von einem konvexen Polygon umgeben, das somit eine Umgebung um ihn herum definiert. Das Voronoi-Diagramm hat die Eigenschaft, dass für einen beliebigen neuen Punkt der nächste Nachbar unter den Datenpunkten der Datenpunkt ist, der in derselben Nachbarschaft liegt. Wenn das Voronoi-Diagramm für einen Satz von Trainingsdaten bestimmt wird, ist es einfach, den nächsten Nachbarn für einen neuen zu klassifizierenden Punkt zu finden. Die Klassenzugehörigkeit wird dann vom nächsten Nachbarn übern**

**In Abb. 8.14 sehen wir deutlich, dass die Methode des nächsten Nachbarn deutlich leistungsfähiger ist als das Perzeptron. Es ist in der Lage, beliebig komplexe Trennlinien (allgemein: Hyperebenen) korrekt darzustellen. Allerdings besteht hier eine Gefahr. Ein einzelner fehlerhafter Punkt kann unter Umständen zu sehr schlechten Klassifizierungsergebnissen führen. Ein solcher Fall tritt in Abb. 8.15 bei der Klassifikation des Schwarzpunktes auf. Die Methode des nächsten Nachbarn klassifiziert dies möglicherweise falsch. Wenn der schwarze Punkt unmittelbar neben einem positiven Punkt liegt, der ein Ausreißer der positiven Klasse ist, wird er als positiv und nicht wie hier beabsichtigt als negativ klassifiziert. Eine fehlerhafte Anpassung an zufällige Fehler (Rauschen) wird als *Überanpassung bezeichnet.***

```
K-NEARESTNEIGHBOR(M+,Mÿ,s)
    V = {k nächste Nachbarn in M+ ÿ Mÿ}
    Wenn    |M+ ÿ V | > |Mÿ ÿ V | Dann Return(„+“)
    Elself |M+ ÿ V | < |Mÿ ÿ V | Dann Return(„–“)
    Anders                          Return(Random(„+“, „–“))
```

**Abb. 8.16 Der K-NEARESTNEIGHBOR -ALGORITHMUS**

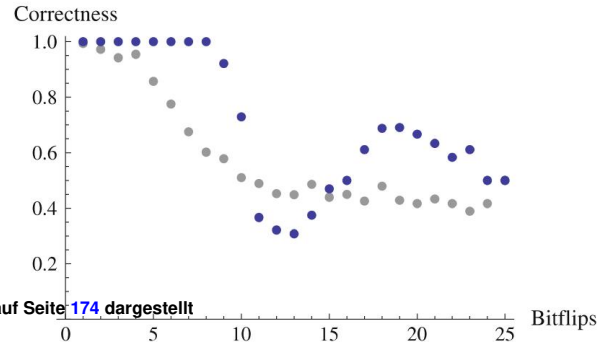



**Abb. 8.17 Relative Korrektheit der Klassifikation des nächsten Nachbarn als Funktion der Anzahl invertierter Bits. Die Struktur der Kurve mit ihrem Minimum bei 13 und ihrem Maximum bei 19 hängt mit der speziellen Struktur der Trainingsdaten zusammen. Zum Vergleich sind die Werte des Perzeptrons aus Beispiel 8.3 auf Seite 174 dargestellt *grau***

**Um Fehleinstufungen aufgrund einzelner Ausreißer zu vermeiden, empfiehlt es sich, die Teilungsfläche etwas zu glätten. Dies kann beispielsweise mit dem K-NEARESTNEIGHBOR- Algorithmus in Abb. 8.16 erreicht werden, der eine Mehrheitsentscheidung unter den k nächsten Nachbarn trifft.**

***Beispiel 8.4*** Wir wenden **NEARESTNEIGHBOR nun auf Beispiel 8.3 auf Seite 174 an. Da es sich um binäre Daten handelt, verwenden wir die Hamming-Distanz als Distanzmetrik.4 Als Testbeispiel verwenden wir wiederum modifizierte Trainingsbeispiele mit jeweils n aufeinanderfolgenden invertierten Bits. In Abb. 8.17 ist der Prozentsatz korrekt klassifizierter Testbeispiele als Funktion der Anzahl invertierter Bits b dargestellt. Bei bis zu acht invertierten Bits werden alle Muster korrekt erkannt. Ab diesem Zeitpunkt steigt die Zahl der Fehler schnell an. Dies ist nicht überraschend, da das Trainingsmuster Nummer 2 aus Abb. 8.11 auf Seite 174 aus der Klasse M+ eine Hamming-Distanz von 9 zu den beiden Trainingsbeispielen Nummer 4 und 5 aus der anderen Klasse hat. Dies bedeutet, dass das Testmuster mit hoher Wahrscheinlichkeit den Mustern der anderen Klasse nahe kommt. Wir sehen ganz deutlich, dass die Klassifizierung des nächsten Nachbarn in dieser Anwendung dem Perzeptron für bis zu acht falsche**

**4Der Hamming-Abstand zwischen zwei Bitvektoren ist die Anzahl der unterschiedlichen Bits der beiden Vektoren.**

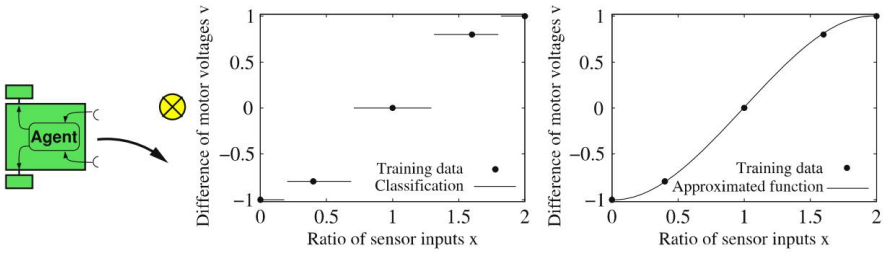



**Abb. 8.18 Der Lernagent, der Licht meiden soll *(links),* dargestellt als Klassifikator *(Mitte)* und als Näherungswert *(rechts)***

## 8.3.1 Zwei Klassen, viele Klassen, Näherung

**Die Klassifizierung „Nächster Nachbar“ kann auch auf mehr als zwei Klassen angewendet werden. Genau wie bei zwei Klassen wird die Klasse des zu klassifizierenden Merkmalsvektors einfach als die Klasse des nächsten Nachbarn festgelegt. Für die k-nächste-Nachbarn-Methode ist die Klasse als die Klasse mit den meisten Mitgliedern unter den k nächsten Nachbarn zu bestimmen.**

**Ist die Anzahl der Klassen groß, ist der Einsatz von Klassifikationsalgorithmen in der Regel nicht mehr sinnvoll, da mit der Anzahl der Klassen auch die Größe der notwendigen Trainingsdaten schnell wächst. Darüber hinaus gehen bei der Klassifizierung vieler Klassen unter bestimmten Umständen wichtige Informationen verloren. Dies wird im folgenden Beispiel deutlich.**

***Beispiel 8.5*** **Ein autonomer Roboter mit einfachen Sensoren ähnlich den in Abb. 1.1 auf Seite 2 vorgestellten Braitenberg-Fahrzeugen soll lernen, sich vom Licht zu entfernen. Das heißt, es soll möglichst optimal lernen, seine Sensorwerte auf ein Lenksignal abzubilden, das die Fahrtrichtung steuert. Der Roboter ist auf seiner Vorderseite mit zwei einfachen Lichtsensoren ausgestattet. Aus den beiden Sensorsignalen (mit sl für den linken und sr für den rechten Sensor) wird die Beziehung x = sr/sl berechnet.**
**Um ab diesem Wert x die Elektromotoren der beiden Räder anzusteuern, ist die Differenz v = Ur ÿUl der beiden Spannungen Ur und Ul des linken bzw. rechten Motors.**
**Die Aufgabe des Lernagenten besteht nun darin, einem Lichtsignal auszuweichen. Es muss also eine 5- Abbildung f lernen, die den „richtigen“ Wert v = f(x) berechnet.**

**Dazu führen wir ein Experiment durch, bei dem wir für einige Messwerte x einen möglichst optimalen Wert v finden. Diese Werte sind als Datenpunkte in Abb. 8.18** dargestellt **und sollen als Trainingsdaten für den Lernagenten dienen. Bei der Klassifizierung des nächsten Nachbarn wird jeder Punkt im Merkmalsraum (d. h. auf der x-Achse) genau wie sein nächster Nachbar in den Trainingsdaten klassifiziert. Die Funktion zur Steuerung der Motoren ist dann eine Schrittfunktion mit großen Sprüngen (Abb. 8.18 Mitte). Wenn wir feinere Schritte wollen, müssen wir entsprechend mehr Trainingsdaten bereitstellen. An**

**5Um das Beispiel einfach und lesbar zu halten, wurde der Merkmalsvektor x bewusst eins beibehalten dimensional.**

Andererseits können wir eine stetige Funktion erhalten, wenn wir eine glatte Funktion annähern, um die fünf Punkte anzupassen (Abb. 8.18 auf Seite 179 rechts). Die Anforderung, dass die Funktion f stetig sein muss, führt zu sehr guten Ergebnissen, auch ohne zusätzliche Datenpunkte

Für die Approximation von Funktionen an Datenpunkten gibt es viele mathematische Methoden, beispielsweise die Polynominterpolation, die Spline-Interpolation oder die Methode der kleinsten Quadrate. In höheren Dimensionen wird die Anwendung dieser Methoden problematisch. Die besondere Schwierigkeit bei der KI besteht darin, dass modellfreie Approximationsverfahren benötigt werden. Das heißt, eine gute Näherung der Daten muss ohne Kenntnisse über besondere Eigenschaften der Daten oder der Anwendung erfolgen. Sehr gute Ergebnisse wurden hier mit neuronalen Netzen und anderen nichtlinearen Funktionsapproximatoren erzielt, die in Kap. 9.

Die k-nächste-Nachbarn-Methode kann auf einfache Weise auf das Approximationsproblem angewendet werden. Im Algorithmus K-NEARESTNEIGHBOR wird nach der Bestimmung der Menge V = {x1, x2, ..., xk } der durchschnittliche Funktionswert der k nächsten Nachbarn ermittelt

$$f(\hat{x}) = \frac{1}{k} \sum_{i=1}^{k} f(xi) \qquad (8.2)$$

wird berechnet und als Näherung fˆ für den Abfragevektor x verwendet. Je größer k wird, desto glatter ist die Funktion fˆ.

### 8.3.2 Entfernung ist relevant

Bei der praktischen Anwendung sowohl diskreter als auch kontinuierlicher Varianten der k-Nächste-Nachbarn-Methode treten häufig Probleme auf. Wenn k größer wird, gibt es typischerweise mehr Nachbarn mit großem Abstand als solche mit kleinem Abstand. Dabei wird die Berechnung von fˆ von weit entfernten Nachbarn dominiert. Um dies zu verhindern, werden die k Nachbarn so gewichtet, dass die weiter entfernten Nachbarn einen geringeren Einfluss auf das Ergebnis haben. Bei der Mehrheitsentscheidung im Algorithmus K-NEARESTNEIGHBOR werden die „Stimmen“ mit dem Gewicht gewichtet

$$wi = \frac{1}{1 + \ddot{y}d(x, xi)^2}, \qquad (8.3)$$

die mit dem Quadrat der Entfernung abnimmt. Die Konstante ÿ bestimmt die Geschwindigkeit der Abnahme der Gewichte. Gleichung (8.2) wird nun ersetzt durch

$$f(\hat{x}) = \frac{\sum_{i=1}^{k} wi\, f(xi)}{\sum_{i=1}^{k} wi}.$$

Bei einer gleichmäßig verteilten Punktkonzentration im Merkmalsraum wird dadurch sichergestellt, dass der Einfluss der Punkte mit zunehmendem Abstand asymptotisch gegen Null geht. Dadurch wird es möglich, viele oder sogar alle Trainingsdaten zu verwenden, um einen gegebenen Eingabevektor zu klassifizieren oder anzunähern.

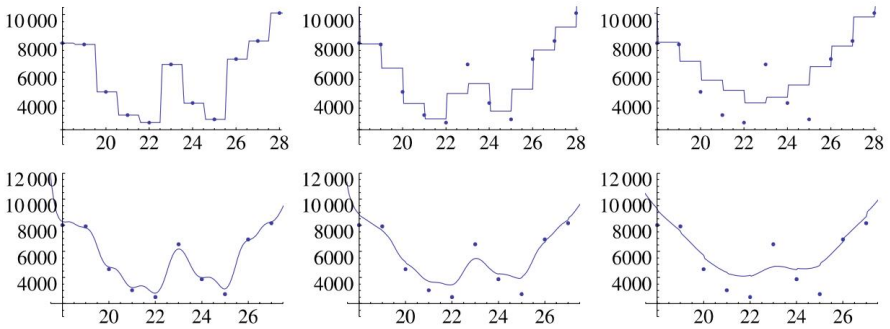



**Abb. 8.19 Vergleich der k-Nearest-Neighbor-Methode *(obere Reihe)* mit k = 1 *(links),* k = 2 *(Mitte)* und k = 6 *(rechts)* zu ihrer abstandsgewichteten Variante *(untere Reihe)* mit ÿ = 20 *(links),* ÿ = 4 *(Mitte)* und ÿ = 1 *(rechts)* für einen eindimensionalen Datensatz**

**Um ein Gefühl für diese Methoden zu bekommen, wird in Abb. 8.19 die k-Nearest-Neighbor-Methode (in der oberen Reihe) mit ihrer abstandsgewichteten Optimierung verglichen. Aufgrund der Mittelung können beide Methoden verallgemeinern, also Rauschen auslöschen, wenn die Anzahl der Nachbarn für k-nächster Nachbar oder der Parameter ÿ entsprechend eingestellt ist. Die Diagramme zeigen gut, dass die abstandsgewichtete Methode eine viel glattere Näherung liefert als die k-nächste Nachbarmethode. Hinsichtlich der Approximationsqualität kann dieses sehr einfache Verfahren gut mit anspruchsvollen Approximationsalgorithmen wie nichtlinearen neuronalen Netzen, Support-Vektor-Maschinen und Gauß-Prozessen mithalten.**

Es gibt viele Alternativen zu der in **(8.3) auf Seite 180 angegebenen Gewichtsfunktion (auch Kernel genannt). Beispielsweise kann eine Gauß-Funktion oder eine ähnliche glockenförmige Funktion verwendet werden. Bei den meisten Anwendungen sind die Ergebnisse bei der Auswahl des Kernels nicht sehr aussagekräftig. Allerdings hat der Breitenparameter ÿ, der bei all diesen Funktionen manuell eingestellt werden muss, großen Einfluss auf die Ergebnisse, wie in Abb. 8.19 dargestellt. Um eine solche umständliche manuelle Anpassung zu vermeiden, wurden Optimierungsverfahren zur automatischen Einstellung dieses Parameters entwickelt [SA94, SE10].**

## 8.3.3 Rechenzeiten

**Wie bereits erwähnt, erfolgt das Training in allen Varianten der Nearest-Neighbor-Methode durch einfaches Speichern aller Trainingsvektoren zusammen mit ihren Labels (Klassenwerten) oder dem Funktionswert f(x). Daher gibt es keinen anderen Lernalgorithmus, der so schnell lernt. Allerdings kann die Beantwortung einer Anfrage zur Klassifizierung oder Approximation eines Vektors x sehr teuer werden. Allein das Finden der k nächsten Nachbarn für n Trainingsdaten erfordert einen Aufwand, der linear mit n wächst. Für die Klassifizierung bzw. Approximation kommt zusätzlich ein Aufwand hinzu, der linear in k ist. Die Gesamtrechenzeit wächst somit mit ÿ(n + k). Bei großen Mengen an Trainingsdaten kann dies zu Problemen führen.**

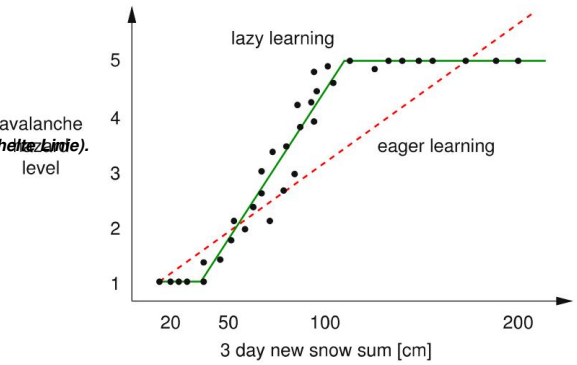



**Abb. 8.20 Zur Bestimmung der Lawinengefahr wird eine Funktion aus Trainingsdaten approximiert. Hier zum Vergleich ein lokales Modell *(durchgezogene Linie)* und ein globales Modell *(gestrichelte Linie).***

### 8.3.4 Zusammenfassung und Ausblick

**Da in der Lernphase der vorgestellten Nearest-Neighbor-Methoden nichts passiert, werden solche Algorithmen auch als *Lazy Learning bezeichnet,* im Gegensatz zum *Eager Learning,* bei dem die Lernphase teuer sein kann, die Anwendung auf neue Beispiele aber sehr effizient ist. Das Perzeptron und alle anderen neuronalen Netze, das Lernen von Entscheidungsbäumen und das Lernen von Bayes'schen Netzen sind Methoden des eifrigen Lernens. Da die Lazy-Learning-Methoden zur Annäherung an eine neue Eingabe Zugriff auf den Speicher mit allen Trainingsdaten benötigen, werden sie auch als *gedächtnisbasiertes L***

**Um diese beiden Klassen von Lernprozessen zu vergleichen, verwenden wir als Beispiel die Aufgabe, aus der Neuschneemenge in einem bestimmten Gebiet der Schweiz die aktuelle Lawinengefahr zu ermitteln.6 In Abb. 8.20 sind von Experten ermittelte Werte eingetragen , die Wir wollen als Trainingsdaten verwenden. Bei der Anwendung eines Eager-Learning-Algorithmus, der eine lineare Approximation der Daten vornimmt, wird die in der Abbildung dargestellte gestrichelte Linie berechnet. Aufgrund der Beschränkung auf eine Gerade ist der Fehler mit maximal etwa 1,5 Gefährdungsstufen relativ groß. Beim Lazy Learning wird nichts berechnet, bevor eine Abfrage nach der aktuellen Gefährdungsstufe eintrifft. Dann wird die Antwort aus mehreren nächsten Nachbarn, also lokal, berechnet. Dadurch könnte die in der Abbildung gezeigte Kurve entstehen, die aus Liniensegmenten zusammengesetzt ist und deutlich kleinere Fehler aufweist. Der Vorteil der Lazy-Methode ist ihre Lokalität. Die Näherung erfolgt lokal aus der aktuellen Neuschneehöhe und nicht global. Für grundsätzlich gleiche Klassen von Funktionen (z. B. lineare Funktionen) sind daher die Lazy-Algorithmen besser.**

**Nearest-Neighbor-Methoden eignen sich gut für alle Problemsituationen, in denen eine gute lokale Näherung erforderlich ist, die aber keine hohen Anforderungen an die Geschwindigkeit des Systems stellen. Der hier erwähnte Lawinenvorhersager, der einmal täglich läuft, ist eine solche Anwendung. Nearest-Neighbor-Methoden sind nicht geeignet, wenn eine Beschreibung des aus den Daten extrahierten Wissens für den Menschen verständlich sein muss, was heute bei vielen Data-Mining-Anwendungen der Fall ist (siehe Abschn. 8.4 )**

---

**6Die dreitägige Gesamtschneemenge ist tatsächlich ein wichtiges Merkmal zur Bestimmung der Gefährdungsstufe. In der Praxis kommen jedoch noch weitere Attribute zum Einsatz [Bra01]. Das hier verwendete Beispiel ist vere**

| Besonderheit | Abfrage | Koffer aus Kofferbasis |
|---|---|---|
| Defektes Teil: | Rücklicht | Vorderlicht |
| Fahrradmodell: | Marin Pine Mountain | VSF T400 |
| Jahr: | 1993 | 2001 |
| Energiequelle: | Batterie | Dynamo |
| Zustand der Glühbirne: | ok | OK |
| Zustand des Lichtkabels: | ? | OK |
| | Lösung | |
| Diagnose: | ? | Elektrischer Kontakt vorn fehlt |
| Reparatur: | ? | Elektrischen Kontakt vorn herstellen |

Abb. 8.21 Einfaches Diagnosebeispiel für eine Abfrage und den zugehörigen Fall aus der Fallbasis

In den letzten Jahren erfreuen sich diese gedächtnisbasierten Lernmethoden immer größerer Beliebtheit, und es wurden verschiedene verbesserte Varianten (z. B. lokal gewichtete lineare Regression) entwickelt angewendet [Cle79].

Um die beschriebenen Methoden nutzen zu können, müssen die Trainingsdaten verfügbar sein in Form von Vektoren aus ganzen Zahlen oder reellen Zahlen. Sie sind daher ungeeignet für Anwendungen, in denen die Daten symbolisch dargestellt werden, beispielsweise als erstes Ordnungslogikformeln. Darauf wollen wir nun kurz eingehen.

## 8.3.5 Fallbasiertes Denken

Beim Case-based Reasoning (CBR) wird die Nearest-Neighbor-Methode auf symbolische Problembeschreibungen und deren Lösungen erweitert. CBR wird zur Diagnose technischer Probleme im Kundendienst oder bei Telefon-Hotlines eingesetzt. Das gezeigte Beispiel In Abb. 8.21 über die Diagnose eines Fahrradlichtausfalls wird dieser Typ veranschaulicht Situation.

Für die Anfrage eines Kunden mit einem defekten Hinterrad wird nach einer Lösung gesucht Licht. In der rechten Spalte ist ein Fall angegeben, der der Abfrage in der mittleren Spalte ähnelt. Dies ergibt sich aus der Fallbasis, die den Trainingsdaten am nächsten kommt Nachbarmethode. Wenn wir einfach den ähnlichsten Fall nehmen würden, wie wir es im nächsten tun Nachbarmethode, dann würden wir am Ende versuchen, das Frontlicht zu reparieren, wenn das Rücklicht ist kaputt. Wir benötigen daher eine Rücktransformation der Lösung des entdeckten ähnlichen Problems zurück in die Abfrage. Die wichtigsten Schritte zur Lösung auf einen CBR-Fall werden in Abb. 8.22 auf Seite 184 durchgeführt. Die Transformation in diesem Das Beispiel ist einfach: Das Rücklicht wird dem Vorderlicht zugeordnet.

So schön und einfach diese Methode in der Theorie auch erscheint, in der Praxis ist der Aufbau von CBR-Diagnosesystemen eine sehr schwierige Aufgabe. Die drei Hauptschwierigkeiten Sind:

*Modellierung* Die Domänen der Anwendung müssen in einem formalen Kontext modelliert werden. Hier kommt es zur logischen Monotonie, die wir aus Kap. 4, bereitet Schwierigkeiten. Kann der Entwickler alle möglichen Sonderfälle und Problemvarianten vorhersagen und abbilden?

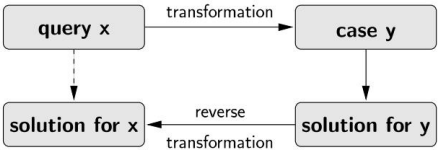



**Abb. 8.22 Wenn für einen Fall x ein ähnlicher Fall y gefunden wird, muss, um eine Lösung für x zu erhalten, die Transformation bestimmt und ihre Umkehrung auf den entdeckten Fall y angewendet werden**

***Ähnlichkeit*** **Finden einer geeigneten Ähnlichkeitsmetrik für symbolische, nicht numerische Fea turen.**

***Transformation*** **Auch wenn ein ähnlicher Fall gefunden wird, ist noch nicht klar, wie die Transformationsabbildung und ihre Umkehrung aussehen sollen.**

**Tatsächlich sind heute praktische CBR-Systeme für diagnostische Anwendungen im Einsatz. Allerdings bleiben diese aus den genannten Gründen hinsichtlich Leistung und Flexibilität weit hinter menschlichen Experten zurück. Eine interessante Alternative zu CBR sind die in Kap. vorgestellten Bayes'schen Netze. 7. Oft lässt sich die symbolische Problemdarstellung auch recht gut auf diskrete oder kontinuierliche numerische Merkmale abbilden. Dann können die genannten induktiven Lernmethoden wie Entscheidungsbäume oder neuronale Netze erfolgreich eingesetzt werden.**

## 8.4 Entscheidungsbaum-Lernen

**Das Lernen von Entscheidungsbäumen ist ein außerordentlich wichtiger Algorithmus für die KI, da er sehr leistungsfähig, aber auch einfach und effizient ist, um Wissen aus Daten zu extrahieren. Gegenüber den beiden bereits vorgestellten Lernalgorithmen weist es einen wichtigen Vorteil auf. Das extrahierte Wissen ist nicht nur als Black-Box-Funktion verfügbar und nutzbar, sondern kann in Form eines lesbaren Entscheidungsbaums auch vom Menschen leicht verstanden, interpretiert und kontrolliert werden. Dies macht das Lernen von Entscheidungsbäumen auch zu einem wichtigen Werkzeug für das Data Mining.**

**Wir werden Funktion und Anwendung des Entscheidungsbaumlernens mithilfe des *C4.5*-Algorithmus diskutieren. C4.5 wurde 1993 vom Australier Ross Quinlan eingeführt und ist eine Verbesserung seines Vorgängers *ID3* (Iterative Dichotomiser 3, 1986). Es ist für die nichtkommerzielle Nutzung frei verfügbar [Qui93]. Eine Weiterentwicklung, die noch effizienter arbeitet und die Kosten von Entscheidungen berücksichtigen kann, ist C5.0 [Qui93].**

**Das von Leo Breiman [BFOS84] entwickelte *CART*-** System (Classification and Regression Trees, 1984) funktioniert ähnlich wie C4.5. **Es verfügt über eine praktische grafische Benutzeroberfläche, ist aber sehr teuer.**

**Zwanzig Jahre zuvor, im Jahr 1964, wurde von J. Sonquist und J. Morgan das CHAID-System (Chi-square Automatic Interaction Detectors) eingeführt, das automatisch Entscheidungsbäume generieren kann. Es hat die bemerkenswerte Eigenschaft, dass es das Wachstum des Baumes stoppt, bevor er zu groß wird, aber heute hat es keine Bedeutung mehr.**

Tabelle 8.4 Variablen für das Ski-Klassifizierungsproblem

| Variable | Wert | Beschreibung |
|---|---|---|
| *Ski* (Zielvariabel) | Ja Nein | Soll ich zum nächsten Skigebiet fahren? mit genügend Schnee? |
| *Sonne* (Feature) | Ja Nein | Gibt es heute Sonnenschein? |
| *Snow_Dist* (Funktion) | ÿ100, >100 | Entfernung zum nächstgelegenen Skigebiet mit Gute Schneeverhältnisse (über/unter). 100 km) |
| *Wochenende* (Feature) | Ja Nein | Ist heute Wochenende? |

Interessant ist auch das Data-Mining-Tool *KNIME* (Konstanz Information Miner), das über eine benutzerfreundliche Oberfläche verfügt und mithilfe der *WEKA*- Java-Bibliothek auch macht Induktion von Entscheidungsbäumen möglich. In Abschn. 8.8 werden wir KNIME vorstellen.

Nun zeigen wir zunächst an einem einfachen Beispiel, wie ein Entscheidungsbaum konstruiert werden kann aus Trainingsdaten, um dann den Algorithmus zu analysieren und auf weitere anzuwenden komplexes LEXMED- Beispiel für die medizinische Diagnose.

## 8.4.1 Ein einfaches Beispiel

Ein begeisterter Skifahrer, der in der Nähe der High Sierra, einer wunderschönen Bergkette in Kalifornien, lebt, möchte einen Entscheidungsbaum, der ihm bei der Entscheidung hilft, ob es sich lohnt, mit dem Auto z mit seinem Auto zu einem Skigebiet in den Bergen. Wir haben also einen Zwei-Klassen-Problemski *ja/nein* basierend auf den in Tabelle 8.4 aufgeführten Variablen.

Abbildung 8.23 auf Seite 186 zeigt einen *Entscheidungsbaum* für dieses Problem. Ein Entscheidungsbaum ist ein Baum, dessen innere Knoten Merkmale (Attribute) darstellen. Jede Kante steht für ein Attributwert. An jedem Blattknoten wird ein Klassenwert angegeben.

Die für die Konstruktion des Entscheidungsbaums verwendeten Daten sind in Tabelle 8.5 aufgeführt Seite 186. Jede Zeile in der Tabelle enthält die Daten für einen Tag und stellt somit einen dar Probe. Bei näherer Betrachtung erkennen wir, dass Zeile 6 und Zeile 7 einander widersprechen. Daher kann kein deterministischer Klassifizierungsalgorithmus alle Daten korrekt klassifizieren. Die Anzahl der falsch klassifizierten Daten muss daher ÿ1 sein. Der Baum in Abb. 8.23 auf Seite 186 klassifiziert die Daten somit optimal.

Wie entsteht aus den Daten ein solcher Baum? Um diese Frage zu beantworten, werden wir unter Beschränken wir uns zunächst auf diskrete Attribute mit endlich vielen Werten. Weil das Die Anzahl der Attribute ist ebenfalls endlich und jedes Attribut kann höchstens einmal pro Pfad vorkommen. Es gibt endlich viele verschiedene Entscheidungsbäume. Ein einfacher, offensichtlicher Algorithmus für die Die Konstruktion eines Baumes würde einfach alle Bäume generieren und dann für jeden Baum berechnen die Anzahl der fehlerhaften Klassifizierungen der Daten und wählen Sie am Ende den Baum aus mit der minimalen Fehleranzahl. Somit hätten wir sogar einen optimalen Algorithmus (im Sinne von Fehlern für die Trainingsdaten) für das Lernen im Entscheidungsbaum.

Der offensichtliche Nachteil dieses Algorithmus ist seine unzumutbar hohe Rechenzeit, sobald die Anzahl der Attribute etwas größer wird. Wir werden Entwickeln Sie nun einen heuristischen Algorithmus, der ausgehend von der Wurzel rekursiv aufbaut

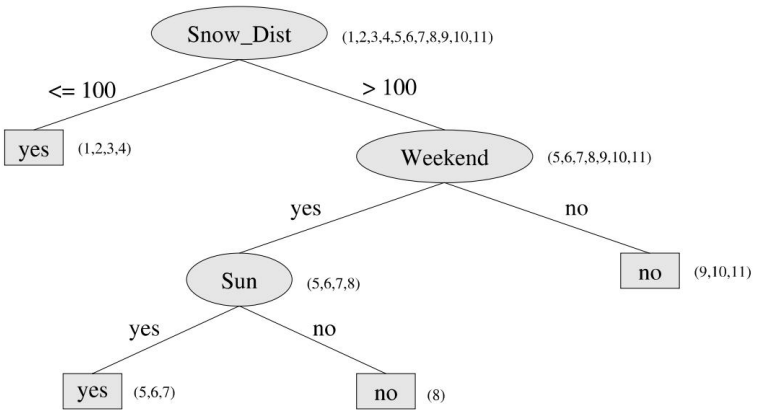



**Abb. 8.23 Entscheidungsbaum für das Ski-Klassifizierungsproblem. In den Listen rechts *neben* den Knoten: die Nummern der entsprechenden Trainingsdaten sind angegeben. Beachten Sie, dass bei den Blattknoten *sonnig = ja* ist nur zwei der drei Beispiele sind korrekt klassifiziert**

**Tabelle 8.5 Datensatz für die Klassifizierungsproblem im Skisport**

| Tag | *Snow_Dist* | *Wochenende* | *Sonnenskilauf* |
|---|---|---|---|
| 1 | ÿ100 | Ja | Ja ja |
| 2 | ÿ100 | Ja | Ja ja |
| 3 | ÿ100 | Ja | Nein Ja |
| 4 | ÿ100 | NEIN | Ja ja |
| 5 | >100 | Ja | Ja ja |
| 6 | >100 | Ja | Ja ja |
| 7 | >100 | Ja | ja Nein |
| 8 | >100 | Ja | Nein Nein |
| 9 | >100 | NEIN | ja Nein |
| 10 | >100 | NEIN | ja Nein |
| 11 | >100 | NEIN | Nein Nein |

**ein Entscheidungsbaum.** Zuerst **wird das Attribut mit dem höchsten Informationsgewinn *(Snow_Dist)*** ausgewählt **aus der Menge aller Attribute für den Wurzelknoten ausgewählt. Für jeden Attributwert (ÿ100, >100) gibt es einen Ast im Baum. Nun wird dieser Vorgang für jeden Zweig wiederholt rekursiv. Bei der Generierung der Knoten wird immer das Attribut mit dem höchsten Informationsgewinn unter den noch nicht genutzten Attributen ausgewählt Geist einer gierigen Strategie.**

## 8.4.2 Entropie als Maß für Informationsinhalte

**Der beschriebene Top-Down-Algorithmus für die Konstruktion eines Entscheidungsbaums Schritt wählt das Attribut mit dem höchsten Informationsgewinn aus. Wir stellen jetzt vor**

**Entropie als *Maß* für den Informationsgehalt eines Trainingsdatensatzes D. Wenn wir im obigen Beispiel** nur die binäre Variable ***Skiing*** betrachten, dann kann D beschrieben werden **als**

$$D = (ja, ja, ja, ja, ja, ja, nein, nein, nein, nein, nein)$$

**mit geschätzten Wahrscheinlichkeiten**

$$p1 = P(ja) = 6/11 \text{ und } p2 = P(nein) = 5/11.$$

**Hier haben wir offensichtlich eine Wahrscheinlichkeitsverteilung p = (6/11, 5/11). Im Allgemeinen lautet dies für ein n-Klassen-Problem**

$$p = (p1,...,pn)$$

**mit**

$$\sum_{i=1}^{N} pi = 1.$$

**Um den Informationsgehalt einer Verteilung einzuführen, beobachten wir zwei Extremfälle. Erst mal lassen**

$$p = (1, 0, 0,..., 0). \tag{8.4}$$

**Das heißt, das erste der n Ereignisse wird mit Sicherheit eintreten und alle anderen nicht. Die Unsicherheit über den Ausgang der Ereignisse ist daher minimal. Im Gegensatz dazu für die Gleichverteilung**

$$p = \left(\frac{1}{N}, \frac{1}{N}, ..., \frac{1}{N}\right) \tag{8.5}$$

**Die Unsicherheit ist maximal, da kein Ereignis von den anderen unterschieden werden kann. Hier stellte sich Claude Shannon die Frage, wie viele Bits nötig wären, um ein solches Ereignis zu kodieren. Im bestimmten Fall von (8.4) werden Nullbits benötigt, da wir wissen, dass der Fall i = 1 immer auftritt. Im gleichverteilten Fall von (8.5) gibt es n gleichwahrscheinliche Möglichkeiten. Für binäre Kodierungen werden hier log2 n Bits benötigt.**
**Da alle Einzelwahrscheinlichkeiten pi = 1/n sind, werden für diese Kodierung log2-Bits benötigt.** $\sum_{i} \frac{1}{pi}$

**Im allgemeinen Fall p = (p1,...,pn) wird, wenn die Wahrscheinlichkeiten der Elementarereignisse von der Gleichverteilung abweichen, der Erwartungswert H für die Anzahl der Bits berechnet. Dazu gewichten wir alle Werte log2 = ÿlog2 pi mit ihren Wahrscheinlichkeiten** $\sum_{i} pi$ **und erhalten**

$$H = \sum_{i=1}^{N} pi(ÿlog2\ pi) = \sum_{i=1}^{ÿn} pi \log2\ pi\ .$$

**Je mehr Bits wir zum Kodieren eines Ereignisses benötigen, desto größer ist eindeutig die Unsicherheit über das Ergebnis. Deshalb definieren wir:**

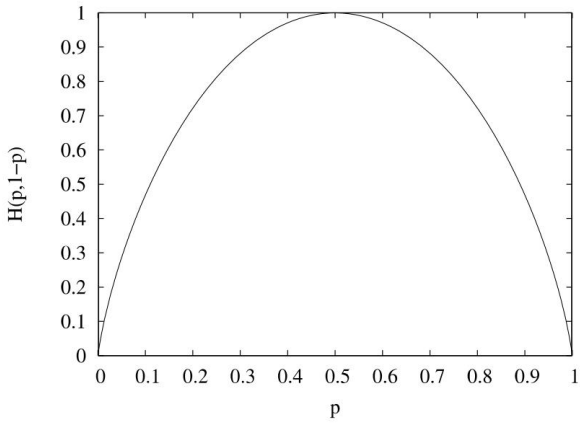



**Abb. 8.24 Die Entropiefunktion für den Fall zweier Klassen. Wir sehen das Maximum bei p = 1/2 und die Symmetrie hinsichtlich der Vertauschung von p und 1 ÿ p**

**Definition 8.4 Die *Entropie* H als Maß für die Unsicherheit einer Wahrscheinlichkeitsverteilung ist definiert durch7**

$$H(p) = H(p_1,\ldots,p_n) := \ddot{y}_{i=1}^{n} p_i \log_2 p_i .$$

**Eine detaillierte Herleitung dieser Formel findet sich in [SW76]. Wenn wir das bestimmte Ereignis p = (1, 0, 0,..., 0), dann 0 log2 0 einsetzen, ergibt sich ein undefinierter Ausdruck. Wir lösen dieses Problem durch die Definition 0 log2 0 := 0 (siehe Aufgabe 8.9 auf Seite 218).**

**Jetzt können wir H (1, 0,..., 0) = 0 berechnen. Wir werden zeigen, dass die Entropie in der Hypercube [0, 1] $^N$ unter der Einschränkung $\sum_{i=1}^{N} p_i = 1$ nimmt seinen Maximalwert an). Im mit gleichmäßiger Verteilung $\frac{1}{N},\ldots,\frac{1}{N}$ Falle einer Veranstaltung mit zwei möglichen (Ergebnissen, die zwei Klassen entsprechen, ist das Ergebnis**

$$H(p) = H(p_1, p_2) = H(p_1, 1 \ddot{y} p_1) = \ddot{y}(p_1 \log_2 p_1 + (1 \ddot{y} p_1)\log_2 (1 \ddot{y} p_1)).$$

**Dieser Ausdruck ist als Funktion von p1 in Abb. 8.24 dargestellt , wobei sein Maximum bei p1 = 1/2 liegt.**

**Da jedem klassifizierten Datensatz D durch Schätzung der Klassenwahrscheinlichkeiten eine Wahrscheinlichkeitsverteilung p zugewiesen wird, können wir das Konzept der Entropie auf Daten erweite**

---

7 In (7.9) auf Seite 124 **wird zur Definition der Entropie der natürliche Logarithmus anstelle von log2** verwendet. **Da hier, wie auch bei der MaxEnt-Methode, nur Entropien verglichen werden, spielt dieser Unterschied keine Rolle. (siehe Übung 8.11 auf Seite 218).**

die Definition

$$H(D) = H(p).$$

Da nun der Informationsgehalt I (D) des Datensatzes D das Gegenteil von Unsicherheit sein soll. So definieren wir:

**Definition 8.5** Der Informationsgehalt eines Datensatzes ist definiert als

$$I(D) := 1 - H(D). \tag{8.6}$$

### 8.4.3 Informationsgewinn

Wenn wir die Entropieformel auf das Beispiel anwenden, ist das Ergebnis

$$H(6/11, 5/11) = 0{,}994$$

Beim Aufbau eines Entscheidungsbaums wird der Datensatz nach jedem neuen Attribut weiter unterteilt. Je mehr ein Attribut den Informationsgehalt der Verteilung durch Aufteilung der Daten erhöht, desto besser ist dieses Attribut. Wir definieren entsprechend:

**Definition 8.6** Der *Informationsgewinn* G(D, A) durch die Verwendung des Attributs A wird durch die Differenz des durchschnittlichen Informationsgehalts des Datensatzes D = D1 ∪ D2 ∪ ··· ∪ Dn dividiert durch das n-Wert-Attribut A bestimmt und der Informationsgehalt I (D) des ungeteilten Datensatzes, der ergibt

$$G(D, A) = \sum_{i=1}^{N} \frac{|Di|}{|D|} I(Di) - I(D).$$

Mit (8.6) erhalten wir daraus

$$G(D, A) = \sum_{i=1}^{N} \frac{|Di|}{|D|} I(Di) - I(D) = \sum_{i=1}^{N} \frac{|Di|}{|D|} (1 - H(Di)) - (1 - H(D))$$

$$= 1 - \sum_{i=1}^{N} \frac{|Di|}{|D|} H(Di) - 1 + H(D)$$

$$= H(D) - \sum_{i=1}^{N} \frac{|Di|}{|D|} H(Di). \tag{8.7}$$

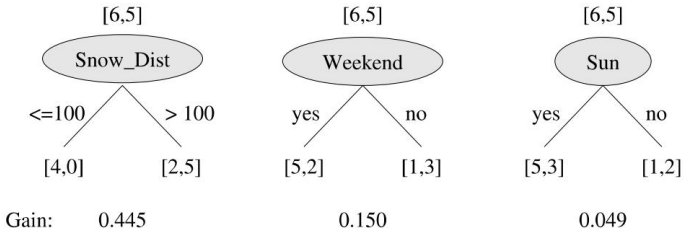



**Abb. 8.25 Der berechnete Gewinn für die verschiedenen Attribute spiegelt wider, ob die Division der Daten durch das jeweilige Attribut zu einer besseren Klasseneinteilung führt. Je stärker die durch das Attribut erzeugten Verteilungen von der Gleichverteilung abweichen, desto höher ist der Informationsgewinn**

**Auf unser Beispiel für das Attribut *Snow_Dist*** angewendet **ergibt dies**

$$G(D, \textit{Snow\_Dist}) = H(D) - \frac{4}{11} H(D_{\leq 100}) + \frac{7}{11} H(D_{>100})$$

$$= 0{,}994 - \frac{4}{11} \cdot 0 + \frac{7}{11} \cdot 0{,}863 = 0{,}445.$$

**Analog erhalten wir**

$$G(D, \textit{Wochenende}) = 0{,}150$$

**Und**

$$G(D, \textit{Sonne}) = 0{,}049.$$

**Das Attribut *Snow_Dist* wird nun zum Wurzelknoten des Entscheidungsbaums. Die Situation der Auswahl dieses Attributs wird in Abb. 8.25 noch einmal verdeutlicht.**

Die beiden Attributwerte ≤100 und >100 erzeugen zwei Kanten im Baum, die den Teilmengen **D≤100 und** D>100 **entsprechen . Für die Teilmenge D≤100 ist die Klassifizierung eindeutig *Ja.* somit endet der Baum hier. Im anderen Zweig D>100 gibt es kein eindeutiges Ergebnis. Somit wiederholt sich der Algorithmus rekursiv. Von den beiden noch verfügbaren Attributen *So* und *Wochenende* muss das bessere ausgewählt werden. Wir berechnen**

$$G(D_{>100}, \textit{Wochenende}) = 0{,}292$$

**Und**

$$G(D_{>100}, \textit{Sonne}) = 0{,}170.$$

**Der Knoten bekommt somit das Attribut *Wochenende* zugewiesen. Für *Weekend = no* endet der Baum mit der Entscheidung *Ski = no.* Eine Berechnung der Verstärkung ergibt hier den Wert 0. Für *Weekend = ja, So* ergibt sich ein Gewinn von 0,171. Dann bricht der Aufbau des Baums ab, da keine weiteren Attribute mehr vorhanden sind, obwohl Beispiel Nummer 7 falsch klassifiziert ist. Den fertigen Baum kennt man bereits aus Abb. 8.23 auf Seite 186.**

### 8.4.4 Anwendung von C4.5

Der gerade generierte Entscheidungsbaum kann auch von C4.5 generiert werden. Die Trainingsdaten werden in einer Datendatei ski.data im folgenden Format gespeichert:

```
<=100, ja, ja, ja <=100, ja, ja, ja <=100,
ja, nein, ja <=100, nein, ja, ja >100, ja,
ja, ja >100, ja, ja , ja >100, ja, ja, nein
>100, ja, nein, nein >100, nein, ja,
nein >100, nein, ja, nein >100, nein,

nein, nein
```

Die Informationen zu Attributen und Klassen werden in der Datei ski.names gespeichert (Zeilen, die mit „|“ beginnen, sind Kommentare):

```
|Unterricht: nein: nicht Skifahren, ja: Skifahren gehen |

Nein Ja.
|
|Attribute |

Snow_Dist:          <=100,>100.
Wochenende:         Nein Ja.
Sonne:              Nein Ja.
```

C4.5 wird dann über die Unix-Befehlszeile aufgerufen und generiert den unten gezeigten Entscheidungsbaum, der mithilfe von Einrückungen formatiert ist. Die Option -f gibt den Namen der Eingabedatei an und die Option -m gibt die Mindestanzahl an Trainingsdatenpunkten an, die zum Generieren eines neuen Zweigs im Baum erforderlich sind. Da die Anzahl der Trainingsdatenpunkte in diesem Beispiel äußerst gering ist, ist -m 1 hier sinnvoll. Für größere Datensätze sollte ein Wert von mindestens -m 10 verwendet werden.

```
unixprompt> c4.5 -f ski -m 1

C4.5 [Release 8] Entscheidungsbaumgenerator

                                                    Mi, 23. August 10:44:49 2010
---------------------------------------

      Optionen:
            Dateistamm <Ski>
            Sinnvoller Test erfordert 2 Zweige mit >=1 Fällen

Lesen Sie 11 Fälle (3 Attribute) aus ski.data

Entscheidungsbaum:

Snow_Dist = <=100: ja (4.0)
Snow_Dist = >100:
| Wochenende = nein: nein (3,0)
| Wochenende = ja:
| | Sonne = nein: nein (1,0)
| | Sonne = ja: ja (3,0/1,0)

Vereinfachter Entscheidungsbaum:

Snow_Dist = <=100: ja (4.0/1.2)
Snow_Dist = >100: nein (7,0/3,4)

Auswertung der Trainingsdaten (11 Items):

          Vor dem Beschneiden              Nach dem Beschneiden
          ---------------                  --------------
          Größe     Fehler                 Größe     Fehler          Schätzen

            7       1(9,1%)                  3       2(18,2%)        (41,7 %) <<
```

Zusätzlich wird ein vereinfachter Baum mit nur einem Attribut angegeben. Dieser Baum, der durch Beschneiden entstanden ist (siehe Abschn. 8.4.7), wird in zunehmendem Maße wichtig sein von Trainingsdaten. In diesem kleinen Beispiel macht es noch nicht viel Sinn. Der Fehler Die Rate für beide Bäume in den Trainingsdaten ist ebenfalls angegeben. Die Zahlen in Klammern Geben Sie nach den Entscheidungen die Größe des zugrunde liegenden Datensatzes und die Anzahl der Fehler an. Beispielsweise zeigt die Zeile Sun = ja: ja (3.0/1.0) im oberen Baum an Für diesen Blattknoten Sun = ja gibt es drei Trainingsbeispiele, eines davon ist falsch eingestuft. Daraus kann der Nutzer also ablesen, ob die Entscheidung statistisch fundiert und/oder sicher ist.

In Abb. 8.26 auf Seite 193 können wir nun das Schema des Lernalgorithmus wiedergeben zur Generierung eines Entscheidungsbaums.

Wir kennen nun die Grundlagen der automatischen Entscheidungsgenerierung Bäume. Für die praktische Anwendung sind jedoch wichtige Erweiterungen erforderlich. Wir wird diese anhand der bereits bekannten **LECMED-** Anwendung **einführen .**

```
GENERATEDECISIONTREE(Daten,Knoten)

Amax = Attribut mit maximalem Informationsgewinn Wenn
G(Amax ) = 0,
    dann wird der Knoten zum Blattknoten mit der häufigsten Klasse in den
    Daten . Andernfalls weisen Sie dem Knoten
        das Attribut Amax zu . Generieren Sie für jeden Wert
                                        a1,...,an von Amax einen Nachfolgeknoten: K1, ,,Kn
        Teilen Sie Daten in D1,...,Dn mit Di = {x ÿ Daten|Amax(x) = ai}
        Für alle i ÿ {1,...,n}
            Wenn alle x ÿ Di zur gleichen Klasse Ci gehören
                Generieren Sie dann den Blattknoten Ki der Klasse Ci
                Else GENERATEDECISIONTREE(Di, Ki)
```

Abb. 8.26 Algorithmus zur Konstruktion eines Entscheidungsbaums

## 8.4.5 Lernen der Blinddarmentzündungsdiagnose

Im Forschungsprojekt LEXMED wurde auf Basis einer Datenbank mit Patientendaten ein Expertensystem zur Diagnose der Blinddarmentzündung entwickelt [ES99, SE00]. Das System, das mit der Methode der maximalen Entropie arbeitet, wird in Abschn. 2.1 beschrieben. 7.3

Wir verwenden nun die LEXMED- Datenbank, um einen Entscheidungsbaum für die Diagnose einer Blinddarmentzündung mit C4.5 zu erstellen. Die als Attribute verwendeten Symptome sind in der Datei app.names definiert:

```
|Definition der Klassen und Attribute | |Klassen

0=Blinddarmentzündung negativ | 1=Appendizitis
positiv 0,1. | |Attribute | Alter:


ununterbrochen.


Sex_(1=m___2=w): 1,2.
Pain_Quadrant1_(0=nein__1=ja): 0,1.
Pain_Quadrant2_(0=nein__1=ja): 0,1.
Pain_Quadrant3_(0=nein__1=ja): 0,1.
Pain_Quadrant4_(0=nein__1=ja): 0,1.
Local_guarding_(0=nein__1=ja): 0,1.
Generalized_guarding_(0=nein__1=ja): 0,1.
Rebound_tenderness_(0=nein__1=ja): 0,1.
Pain_on_tapping_(0=nein__1=ja): 0,1.
```

```
Pain_during_rectal_examination_(0=nein__1=ja): 0,1.
Temp_axial: kontinuierlich.
Temp_rectal: kontinuierlich.
Leukozyten: kontinuierlich.
Diabetes_mellitus_(0=nein__1=ja): 0,1
```

Wir sehen, dass neben vielen binären Merkmalen wie den verschiedenen Schmerzsymptomen auch kontinuierliche Symptome wie Alter und Fiebertemperatur auftreten. In der folgenden Trainingsdatendatei, app.data, wird in jeder Zeile ein Fall beschrieben. In der ersten Zeile steht ein 19-jähriger männlicher Patient mit Schmerzen im dritten Quadranten (unten rechts, wo sich der Blinddarm befindet), den beiden Fieberwerten 36,2 und 37,8 Grad Celsius, einem Leukozytenwert von 13400 und einer positiven Diagnose ist ein entzünd

```
19,1,0,0,1,0,1,0,1,1,0,362,378,13400,0,1
13,1,0,0,1,0,1,0,1,1,1,383,385,18100 ,0,1
32,2,0,0,1,0,1,0,1,1,0,364,374,11800,0,1 18,2,0,0,1,1,0,0,0,0
,0,362,370,09300,0,0 73,2,1,0,1,1,1,0,1,1,1,376,380,13600,1,1
30,1,1,1,1,1,0,1 ,1,1,1,377,387,21100,0,1
56,1,1,1,1,1,0,1,1,1,0,390,?,14100,0,1 36,1,0,0,1
,0,1,0,1,1,0,372,382,11300,0,1
36,2,0,0,1,0,0,0,1,1,1,370,379,15300,0,1 33,1,0
,0,1,0,1,0,1,1,0,367,376,17400,0,1
19,1,0,0,1,0,0,0,1,1,0,361,375,17600,0,1 12
,1,0,0,1,0,1,0,1,1,0,364,370,12900,0,0

...
```

Ohne näher auf die Datenbank einzugehen, ist es wichtig zu erwähnen, dass nur Patienten in die Datenbank aufgenommen werden, bei denen bei der Ankunft im Krankenhaus der Verdacht auf eine Blinddarmentzündung bestand und die dann operiert wurden. In der siebten Zeile sehen wir, dass C4.5 auch mit fehlenden Werten umgehen kann. Die Daten umfassen 9764 Fä

```
unixprompt> c4.5 -f app -u -m 100

C4.5 [Release 8] Entscheidungsbaumgenerator
                                                    Mi, 23. August 13:13:15 2006
--------------------------------------

Lesen Sie 9764 Fälle (15 Attribute) aus app.data

Entscheidungsbaum:
```

```
Leukozyten <= 11030 :
| Rebound_tenderness = 0:
| | Temp_rektal > 381 : 1 (135,9/54,2)
| | Temp_rectal <= 381 :
| | | Local_guarding = 0: 0 (1453,3/358,9)
| | | Local_guarding = 1:
| | | | Sex_(1=m___2=w) = 1: 1 (160,1/74,9)
| | | | Sex_(1=m___2=w) = 2: 0 (286,3/97,6)
| Rebound_tenderness = 1:
| | Leukozyten <= 8600 :
| | | Temp_rektal > 378 : 1 (176,0/59,4)
| | | Temp_rectal <= 378 :
| | | | Sex_(1=m___2=w) = 1:
| | | | | Local_guarding = 0: 0 (110,7/51,7)
| | | | | Local_guarding = 1: 1 (160,6/68,5)
| | | | Sex_(1=m___2=w) = 2:
| | | | | Alter <= 14: 1 (131,1/63,1)
| | | | | Alter > 14: 0 (398,3/137,6)
| | Leukozyten > 8600:
| | | Sex_(1=m___2=w) = 1: 1 (429,9/91,0)
| | | Sex_(1=m___2=w) = 2:
| | | | Local_guarding = 1: 1 (311,2/103,0)
| | | | Local_guarding = 0:
| | | | | Temp_rektal <= 375 : 1 (125,4/55,8)
| | | | | Temp_rektal > 375 : 0 (118,3/56,1)
Leukozyten > 11030:
| Rebound_tenderness = 1: 1 (4300,0/519,9)
| Rebound_tenderness = 0:
| | Leukozyten > 14040 : 1 (826,6/163,8)
| | Leukozyten <= 14040 :
| | | Pain_on_tapping = 1: 1 (260,6/83,7)
| | | Pain_on_tapping = 0:
| | | | Local_guarding = 1: 1 (117,5/44,4)
| | | | Local_guarding = 0:
| | | | | Temp_axial <= 368 : 0 (131,9/57,4)
| | | | | Temp_axial > 368 : 1 (130,5/57,8)

Vereinfachter Entscheidungsbaum:

Leukozyten > 11030 : 1 (5767,0/964,1)
Leukozyten <= 11030 :
| Rebound_tenderness = 0:
| | Temp_rektal > 381 : 1 (135,9/58,7)
| | Temp_rectal <= 381 :
| | | Local_guarding = 0: 0 (1453,3/370,9)
| | | Local_guarding = 1:
| | | | Sex_(1=m___2=w) = 1: 1 (160,1/79,7)
| | | | Sex_(1=m___2=w) = 2: 0 (286,3/103,7)
| Rebound_tenderness = 1:
| | Leukozyten > 8600 : 1 (984,7/322,6)
| | Leukozyten <= 8600 :
```

```
| Temp_rektal > 378 : 1 (176,0/64,3)
| Temp_rectal <= 378 :
| Sex_(1=m___2=w) = 1:
| | Local_guarding = 0: 0 (110,7/55,8)
| | Local_guarding = 1: 1 (160,6/73,4)
| Sex_(1=m___2=w) = 2:
| | Alter <= 14: 1 (131,1/67,6)
| | Alter > 14: 0 (398,3/144,7)

Auswertung der Trainingsdaten (9764 Items):

Vor dem Beschneiden Nach dem Beschneiden
Größenfehler Schätzung der Größenfehler
37 2197(22,5%) 21 2223(22,8%) (23,6%) <<

Auswertung anhand von Testdaten (4882 Items):

Vor dem Beschneiden Nach dem Beschneiden
Größenfehler Schätzung der Größenfehler
37 1148(23,5%) 21 1153(23,6%) (23,6%) <<

 (a) (b) <-klassifiziert als
758 885 (a): Klasse 0
268 2971 (b): Klasse 1
```

## 8.4.6 Kontinuierliche Attribute

In den für die Blinddarmentzündungsdiagnose generierten Bäumen gibt es einen Knoten Leukozyten > 11030, was eindeutig auf das kontinuierliche Attribut Leukozyten zurückzuführen ist durch Festlegen eines Schwellenwerts auf den Wert 11030. C4.5 hat somit aus dem kontinuierlichen Attribut Leukozyten ein binäres Attribut Leukozyten > 11030 erstellt.

Der Schwellenwert ÿD,A für ein Attribut A wird durch den folgenden Algorithmus bestimmt: Für alle Werte v, die in den Trainingsdaten D vorkommen, gilt das binäre Attribut A>v generiert und deren Informationsgewinn berechnet. Der Schwellenwert ÿD,A wird dann auf gesetzt der Wert v mit dem maximalen Informationsgewinn, also:

$$\ddot{y}_{D,A} = \text{argmax}_v \{G(D, A > v)\}.$$

Für ein Attribut wie den Leukozytenwert oder das Alter des Patienten kann eine Entscheidung getroffen werden eine binäre Diskretisierung ist vermutlich zu ungenau. Dennoch besteht keine Notwendigkeit Diskretisieren Sie feiner, da jedes kontinuierliche Attribut bei jedem neu generierten Attribut getestet wird Knoten und kann daher in einem Baum mit unterschiedlichem Schwellenwert ÿD,A wiederholt vorkommen. Daher letztlich erhalten wir eine sehr gute Diskretisierung, deren Feinheit zum Problem passt.

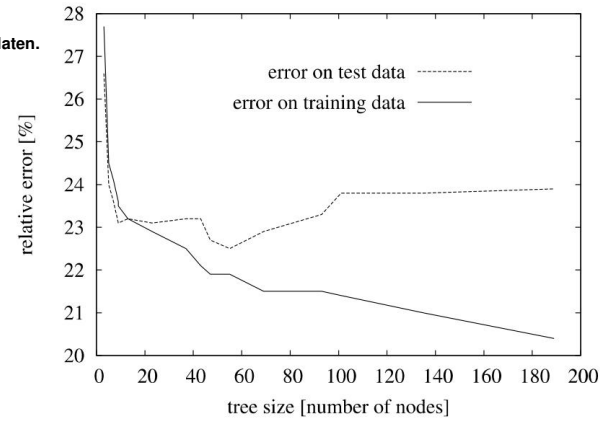



**Abb. 8.27 Lernkurve von C4.5 anhand der Blinddarmentzündungsdaten. Wir sehen deutlich die Überanpassung von Bäumen mit mehr als 55 Knoten**

## 8.4.7 Beschneiden – Den Baum fällen

**Seit Aristoteles gilt die Regel, dass von zwei wissenschaftlichen Theorien, die den gleichen Sachverhalt gleich gut erklären, die einfachere zu bevorzugen ist. Dieses Gesetz der Ökonomie, heute auch als *Occams Rasiermesser bekannt,* ist für maschinelles Lernen und Data Mining von großer Bedeutung.**

**Ein Entscheidungsbaum ist eine Theorie zur Beschreibung der Trainingsdaten. Eine andere Theorie zur Beschreibung der Daten sind die Daten selbst. Wenn der Baum alle Daten fehlerfrei klassifiziert, aber viel kompakter und damit für den Menschen leichter verständlich ist, dann ist er nach Occams Rasiermesser vorzuziehen. Das Gleiche gilt für zwei Entscheidungsbäume unterschiedlicher Größe. Daher muss das Ziel jedes Algorithmus zur Generierung eines Entscheidungsbaums darin bestehen, für eine gegebene Fehlerrate einen möglichst kleinen Entscheidungsbaum zu generieren. Unter allen Bäumen mit einer festen Fehlerrate sollte immer der kleinste Ba**

**Bisher haben wir den Begriff Fehlerquote nicht genau definiert. Wie bereits mehrfach erwähnt, ist es wichtig, dass der Lernbaum die Trainingsdaten nicht nur im Gedächtnis behält, sondern dass diese gut verallgemeinert werden können. Um die Generalisierungsfähigkeit eines Baums zu testen, teilen wir die verfügbaren Daten in einen Satz *Trainingsdaten* und einen Satz *Testdaten auf.* Die Testdaten werden vor dem Lernalgorithmus verborgen und nur zum Testen verwendet. Wenn ein großer Datensatz verfügbar ist, beispielsweise die Blinddarmentzündungsdaten, können wir beispielsweise zwei Drittel der Daten zum Lernen und das verbleibend**

**Neben der besseren Verständlichkeit hat Occams Rasiermesser noch eine weitere wichtige Begründung: die Fähigkeit zur Generalisierung. Je komplexer das Modell (hier ein Entscheidungsbaum) ist, desto mehr Details werden abgebildet, aber im gleichen Maße ist das Modell umso weniger auf neue Daten übertragbar. Dieser Zusammenhang ist in Abb. 8.27 dargestellt . Entscheidungsbäume unterschiedlicher Größe wurden anhand der Blinddarmentzündungsdaten trainiert. In der Grafik sind Klassifizierungsfehler sowohl der Trainingsdaten als auch der Testdaten angegeben. Die Fehlerrate der Trainingsdaten nimmt monoton mit der Größe des Baums ab. Bis zu einer Baumgröße von 55 Knoten sinkt auch die Fehlerquote bei Testdaten. Wächst der Baum jedoch weiter, steigt die Fehlerquote wieder an! Dieser Effekt, den wir bereits bei der Methode des nächsten Nachbarn gesehen haben, wird als *Überanpassung*

Wir werden dieses Konzept, das für fast alle Lernprozesse wichtig ist, weitergeben:
eine allgemeine Definition aus [Mit97]:

**Definition 8.7 Es sei ein spezifischer Lernalgorithmus, also ein Lernagent, gegeben. Wir nennen einen Agenten A eine Überanpassung an die Trainingsdaten, wenn es einen anderen Agenten Aÿ gibt, dessen Fehler in den Trainingsdaten größer als der von A, dessen Fehler in der gesamten Datenverteilung jedoch kleiner als der Fehler von A ist.**

Wie können wir nun diesen Punkt des minimalen Fehlers in den Testdaten finden? Der offensichtlichste Algorithmus heißt *Kreuzvalidierung.* Während der Baumkonstruktion wird parallel der Fehler der Testdaten gemessen. Sobald der Fehler deutlich ansteigt, wird der Baum mit dem minimalen Fehler gespeichert. Dieser Algorithmus wird vom zuvor erwähnten CART-System verwendet.

C4.5 funktioniert etwas anders. Zunächst wird mit dem Algorithmus GENERATEDECISIONTREE aus Abb. 8.26 auf Seite 193 ein Baum generiert, der in der Regel überfit ist .
Mithilfe *von Pruning* wird dann versucht, Knoten des Baums wegzuschneiden, bis der Fehler in den Testdaten, der durch den Fehler in den Trainingsdaten geschätzt wird, zuzunehmen beginnt.8 Wie die Konstruktion des Baums ist auch dies ein gieriger Algorithmus . Dies bedeutet, dass ein einmal beschnittener Knoten nicht erneut eingefügt werden kann, auch wenn sich dies später als besser herausstellt.

## 8.4.8 Fehlende Werte

Häufig fehlen einzelne Attributwerte in den Trainingsdaten. Im LEX MED -Datensatz kommt der folgende Eintrag vor:

56, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 390, ?, 14100, 0, 1,

bei dem einer der Fieberwerte fehlt. Dennoch können solche Daten beim Aufbau des Entscheidungsbaums verwendet werden. Wir können dem Attribut den häufigsten Wert aus dem gesamten Datensatz oder den häufigsten aller Datenpunkte derselben Klasse zuweisen. Noch besser ist es, den fehlenden Attributwert durch die Wahrscheinlichkeitsverteilung aller Attributwerte zu ersetzen und das Trainingsbeispiel entsprechend dieser Verteilung in Zweige aufzuteilen. Dies ist übrigens ein Grund für das Auftreten nicht ganzzahliger Werte in den Ausdrücken in Klammern neben den Blattknoten des C4.5-Baums.

Fehlende Werte können nicht nur beim Lernen, sondern auch beim Klassifizieren auftreten. Diese werden auf die gleiche Weise gehandhabt wie beim Lernen.

---

8 Es wäre besser, den Fehler direkt auf die Testdaten anzuwenden. Zumindest dann, wenn die Menge an Trainingsdaten ausreicht, um einen separaten Testsatz zu rechtfertigen.

### 8.4.9 Zusammenfassung

Das Erlernen von Entscheidungsbäumen ist ein beliebter Ansatz für Klassifizierungsaufgaben. Die Gründe hierfür liegen in der einfachen Anwendung und Schnelligkeit. Bei einem Datensatz von etwa 10.000 LEXMED- Datenpunkten mit jeweils 15 Attributen benötigt C4.5 etwa 0,3 Sekunden zum Lerner Dies ist im Vergleich zu anderen Lernalgorithmen sehr schnell.

Für den Anwender ist es aber auch wichtig, dass der Entscheidungsbaum als erlerntes Modell verstanden und ggf. verändert werden kann. Es ist auch nicht schwierig, einen Entscheidungsbaum automatisch in eine Reihe von If-Then-Else-Anweisungen umzuwandeln und ihn so effizient in ein bestehendes Programm einzubauen.

Da sowohl beim Aufbau des Baums als auch beim Beschneiden ein Greedy-Algorithmus verwendet wird, sind die Bäume im Allgemeinen suboptimal. Der entdeckte Entscheidungsbaum weist normalerweise eine relativ geringe Fehlerquote auf. Es gibt jedoch möglicherweise einen besseren Baum, da die heuristische gierige Suche von C4.5 kleine Bäume und Attribute mit hohem Informationsgewinn an der Spitze des Baums bevorzugt. Bei Attributen mit vielen Werten weist die vorgestellte Formel für den Informationsgewinn Schwächen auf. Alternativen hierzu werden in [Mit97] angegeben.

## 8.5 Lernen von Bayes'schen Netzwerken

In Abschn. In Abschnitt 7.4 wurde gezeigt, wie man manuell ein Bayes'sches Netzwerk aufbaut. Jetzt werden wir Algorithmen zur Induktion von Bayes'schen Netzwerken vorstellen. Ähnlich wie beim zuvor beschriebenen Lernprozess wird ein Bayes-Netzwerk automatisch aus einer Datei mit Trainingsdaten generiert. Dieser Prozess gliedert sich typischerweise in zwei Teile.

1. *Erlernen der Netzwerkstruktur:* Für gegebene Variablen wird aus den Trainingsdaten die Netzwerktopologie generiert. Dieser erste Schritt ist bei weitem der schwierigere und wird später genauer beachtet.
2. *Lernen der bedingten Wahrscheinlichkeiten:* Für bekannte Netzwerktopologien müssen die CPTs mit Werten gefüllt werden. Wenn genügend Trainingsdaten verfügbar sind, können alle notwendigen bedingten Wahrscheinlichkeiten durch Zählen der Häufigkeiten in den Daten geschätzt w Dieser Schritt lässt sich relativ einfach automatisieren.

Wir erklären nun, wie Bayesianische Netzwerke mithilfe eines einfachen Algorithmus aus [Jen01] lernen.

### 8.5.1 Erlernen der Netzwerkstruktur

Bei der Entwicklung eines Bayes'schen Netzwerks (siehe Abschn. 7.4.6) muss die kausale Abhängigkeit der Variablen berücksichtigt werden, um ein einfaches Netzwerk guter Qualität zu erhalten. Der menschliche Entwickler ist auf Hintergrundwissen angewiesen, das der Maschine nicht zur Verfügung steht. Daher kann dieser Vorgang nicht einfach automatisiert werden.

Das Finden einer optimalen Struktur für ein Bayes'sches Netzwerk kann als klassisches Suchproblem formuliert werden. Gegeben seien ein Satz Variablen V1,...,Vn und eine Datei mit Trainingsdaten. Wir suchen nach einer Menge gerichteter Kanten ohne Kreise zwischen den Knoten

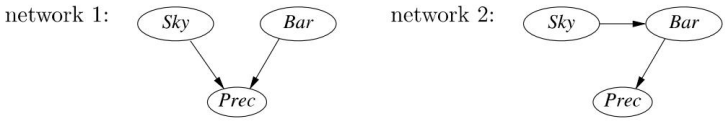



**Abb. 8.28 Zwei Bayes'sche Netze zur Modellierung des Wettervorhersagebeispiels aus Übung 7.3 auf Seite 158**

**V1,...,Vn, also ein gerichteter azyklischer Graph (DAG), der die zugrunde liegenden Daten bestmöglich wiedergibt.**

**Zuerst beobachten wir den Suchraum. Die Anzahl verschiedener DAGs wächst mehr als exponentiell mit der Anzahl der Knoten. Für fünf Knoten gibt es 29281 und für neun Knoten etwa 1015 verschiedene DAGs [MDBM00]. Somit ist eine uninformierte kombinatorische Suche (siehe Abschn. 6.2) im Raum aller Graphen mit einer gegebenen Menge von Variablen aussichtslos, wenn die Anzahl der Variablen wächst. Daher müssen heuristische Algorithmen verwendet werden. Dies wirft die Frage nach einer Bewertungsfunktion für Bayes'sche Netzwerke auf. Es ist möglich, den Klassifizierungsfehler eines Netzwerks bei der Anwendung auf einen Testdatensatz zu messen, wie es beispielsweise in C4.5 durchgeführt wird (siehe Abschn. 8.4 ). Hierzu müssen allerdings die vom Bayes'schen Netzwerk berechneten Wahrscheinlichkeiten**

**Über die Wahrscheinlichkeitsverteilung kann eine direkte Messung der Qualität eines Netzwerks erfolgen. Wir gehen davon aus, dass wir vor dem Aufbau des Netzwerks aus den Daten die Verteilung bestimmen (schätzen) konnten. Dann beginnen wir mit der Suche im Raum aller DAGs, schätzen anhand der Daten den Wert der CPTs für jede DAG (also für jedes Bayes'sche Netzwerk), berechnen daraus die Verteilung und vergleichen sie mit der bekannten Verteilung die Daten. Für den Vergleich von Verteilungen benötigen wir natürlich eine Distanzmetrik.**

**Betrachten wir das Wettervorhersagebeispiel aus Übung 7.3 auf Seite 158 mit den drei Variablen *Sky, Bar, Prec* und der Verteilung**

$$P(Sky,Bar,Prec) = (0{,}40,\ 0{,}07,\ 0{,}08,\ 0{,}10,\ 0{,}09,\ 0{,}11,\ 0{,}03,\ 0{,}12).$$

**In Abb. 8.28 sind zwei Bayes-Netzwerke dargestellt, die wir nun hinsichtlich ihrer Qualität vergleichen werden. Jedes dieser Netzwerke geht von einer Unabhängigkeitsannahme aus, die bestätigt wird, indem wir die Verteilung des Netzwerks bestimmen und diese dann mit der ursprünglichen Verteilung vergleichen (siehe Übung 8.15 auf Seite 219).**

**Da die Verteilung für konstante vorgegebene Variablen eindeutig durch einen Vektor konstanter Länge dargestellt wird, können wir die euklidische Norm der Differenz der beiden Vektoren als Abstand zwischen Verteilungen berechnen. Wir definieren**

$$dq(x, y) = \sum (x_i - y_i)^2$$

**als Summe der Quadrate der Abstände der Vektorkomponenten und berechnen Sie den Abstand dq (Pa,P) = 0,0029 der Verteilung Pa von Netzwerk 1 von der ursprünglichen Verteilung. Für Netzwerk 2 berechnen wir dq (Pb,P) = 0,014. Offensichtlich ist Netzwerk 1 eine bessere Annäherung an die Verteilung. Oftmals wird anstelle des Quadratabstandes der**

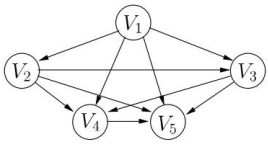



**Abb. 8.29 Das maximale Netzwerk mit fünf Variablen und Kanten (Vi, Vj ), die die Bedingung i<j erfüllen**

sogenannte Kullback-Leibler-Distanz

$$dk(x, y) = \sum yi(\log 2\, yi \ddot{y} \log 2\, xi),$$

Es wird eine Metrik der Informationstheorie verwendet. Damit berechnen wir dk(Pa,P) = 0,017 und dk(Pb,P) = 0,09 und kommen zum gleichen Ergebnis wie zuvor. Es ist zu erwarten, dass Netzwerke mit vielen Kanten die Verteilung besser annähern als solche mit wenigen Kanten. Wenn alle Kanten im Netzwerk konstruiert werden, wird es sehr unübersichtlich und es besteht die Gefahr einer Überanpassung, wie es bei vielen anderen Lernalgorithmen der Fall ist. Um eine Überanpassung zu vermeiden, geben wir kleinen Netzwerken mithilfe einer heuristischen Bewertungsfunktion ein größeres Gewicht

$$f(N) = \textit{Größe}(N) + w \cdot dk(P_N, P).$$

Dabei ist *Size(N )* die Anzahl der Einträge in den CPTs und P N die Verteilung des Netzwerks N. w ist ein Gewichtsfaktor, der manuell angepasst werden muss.

Der Lernalgorithmus für Bayes'sche Netzwerke berechnet also die heuristische Bewertung f(N) für viele verschiedene Netzwerke und wählt dann das Netzwerk mit dem kleinsten Wert aus. Wie bereits erwähnt besteht die Schwierigkeit in der Reduzierung des Suchraums für die gesuchte Netzwerktopologie. Als einfacher Algorithmus ist es möglich, ausgehend von einer (z. B. kausalen) Ordnung der Variablen V1,...,Vn, nur solche Kanten in den Graphen aufzunehmen, für die i<j gilt. Wir beginnen mit dem Maximalmodell, das diese Bedingung erfüllt. Dieses Netzwerk ist in Abb. 8.29 für fünf geordnete Variablen dargestellt.

Nun wird beispielsweise im Sinne einer Greedy Search (vgl. Abschn. 6.3.1) eine Kante nach der anderen entfernt, bis der Wert f nicht mehr abnimmt.

Für größere Netzwerke ist dieser Algorithmus in dieser Form nicht praktikabel. Der große Suchraum, die manuelle Abstimmung des Gewichts w und der notwendige Vergleich mit einer Zielverteilung P sind Gründe dafür, denn diese können einfach zu groß werden, oder der verfügbare Datensatz könnte zu klein sein.

Tatsächlich ist die Forschung zum Lernen von Bayes'schen Netzwerken immer noch in vollem Gange, und es gibt eine große Anzahl vorgeschlagener Algorithmen, zum Beispiel den EM-Algorithmus (siehe Abschn. 8.7.2), Markov-Ketten-Monte-Carlo-Methoden und Gibbs-Sampling [ DHS01, Jor99, Jen01, HTF09]. Neben dem hier vorgestellten Batch-Learning, bei dem das Netzwerk einmalig aus dem gesamten Datensatz generiert wird, gibt es auch inkrementelle Algorithmen, bei denen jeder einzelne neue Fall zur Verbesserung des Netzwerks genutzt wird. Es gibt auch Implementierungen dieser Algorithmen, beispielsweise von Hugin (www.hugin.com) und Bayesware (www.bayesware.com).

## 8.6 Der Naive-Bayes-Klassifikator

**In Abb. 7.14 auf Seite 152 wurde die Diagnose einer Blinddarmentzündung als Bayes'sches Netzwerk modelliert. Da gerichtete Kanten an einem Diagnoseknoten beginnen und keine Kanten dort enden, muss zur Beantwortung einer Diagnoseabfrage die Bayes-Formel verwendet werden. Für die Symptome S1,...,Sn und die k-Wert-Diagnose D mit den Werten b1,...,bk berechnen wir die Wahr**

$$P(D|S1,...,Sn) = P \frac{P(S1,...,Sn|D) \cdot P(D)}{(S1,...,Sn)},$$

**für die Diagnose anhand der Symptome des Patienten. Im schlimmsten Fall, also wenn es keine unabhängigen Variablen gäbe, müssten alle Kombinationen aller Symptome und *D* für alle 20.643.840 Wahrscheinlichkeiten der Verteilung P(S1,...,Sn,D) bestimmt werden. Dies würde eine riesige Datenbank erfordern. Im Fall des** Bayes'schen Netzwerks **von LEX MED reduziert sich die Anzahl der notwendigen Werte (in den CPTs) auf 521. Das Netzwerk kann jedoch weiter vereinfacht werden, indem wir annehmen, dass alle Symptomvariablen bei gegebenem D bedingt unabhängig sind, das heißt:**

$$P(S1,...,Sn|D) = P(S1|D) \cdot\cdot\cdot\cdot\cdot P(Sn|D).$$

**Das Bayes'sche Netzwerk für Blinddarmentzündung wird dann zu dem in Abb. 8.30 auf Seite 203 gezeigten Stern vereinfacht.**
**Somit erhalten wir die Formel**

$$S1,...,Sn) = \frac{P(D)n\ P(Si|D)\ i=1\ P(D|}{P(S1,...,Sn)}. \qquad (8.8)$$

Die berechneten Wahrscheinlichkeiten werden durch einen einfachen naiven Bayes-Klassifikator in eine Entscheidung umgewandelt, der **aus allen Werten di** in D **das Maximum P (D = di | S1,...,Sn)** auswählt . **Das heißt, es bestimmt**

$$dNaive\text{-}Bayes = argmaxiÿ\{1,...,k\}\ P(D = di\ |S1,...,Sn).$$

**Da der Nenner in (8.8) konstant ist, kann er bei der Maximierung weggelassen werden, was zur *naiven Bayes-Formel*** führt

$$dNaive\text{-}Bayes = argmaxiÿ\{1,...,k\}\ P(D = di) \prod_{j=1}^{N} P(Sj\ |D).$$

Da mehrere Knoten nun weniger Vorfahren haben, verringert sich die Anzahl der zur Beschreibung der **LECMED-** Verteilung in den CPTs **notwendigen Werte gemäß (7.22) auf Seite 151 auf**

$$6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 4 + 3 \cdot 4 + 10 \cdot (1 \cdot 4) + 1 = 141.$$

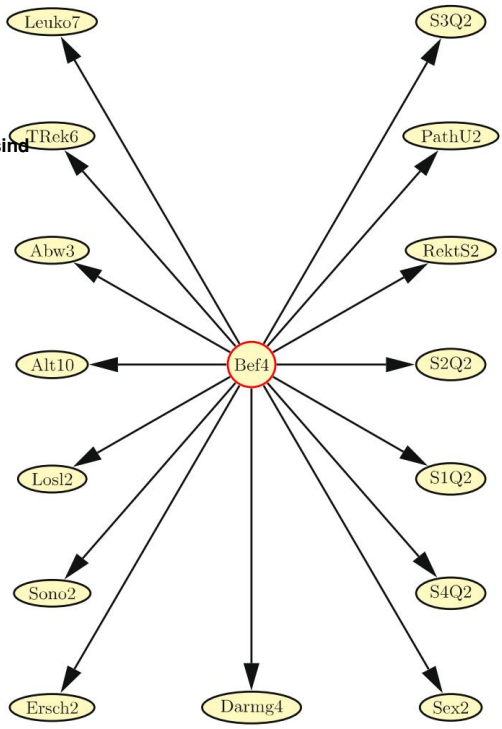



**Abb. 8.30 Bayesianisches Netzwerk für die LEXMED-Anwendung mit der Annahme, dass alle Symptome bei gegebener Diagnose bedingt unabhängig sind**

**Für ein medizinisches Diagnosesystem wie LEXYMED wäre diese Vereinfachung nicht akzeptabel. Aber für Aufgaben mit vielen unabhängigen Variablen ist Naive Bayes teilweise oder sogar sehr gut geeignet, wie wir am Beispiel der Textklassifizierung sehen werden.**

**Die naive Bayes-Klassifikation ist übrigens genauso aussagekräftig wie das in Abschn. 2.1 beschriebene lineare Score-System. 7.3.1 (siehe Übung 8.16 auf Seite 219). Das heißt, allen Scores liegt die Annahme zugrunde, dass alle Symptome angesichts der Diagnose bedingt unabhängig sind. Dennoch werden Scores auch heute noch in der Medizin verwendet. Trotz der Tatsache, dass er aus einer besseren, repräsentativen Datenbasis generiert wurde, weist der Ohmann-Score im Vergleich zu LEXMED in Abb. 7.10 auf Seite 142 eine schlechtere Diagnosequalität auf. Die eingeschränkte Aussagekraft ist sicherlich ein Grund dafür. Beispielsweise ist es wie bei Naive Bayes nicht möglich, Abhängigkeiten zwischen Sym Partituren.**

**Schätzung von Wahrscheinlichkeiten Wenn wir die naive Bayes-Formel in (8.8) auf Seite 202 betrachten, sehen wir, dass der gesamte Ausdruck Null wird, sobald einer der Faktoren P (Si |D) auf der rechten Seite Null wird. Theoretisch ist hier nichts falsch. In der Praxis kann dies jedoch zu sehr unangenehmen Effekten führen, wenn die P (Si |D) klein sind, da diese durch Zählen von Frequenzen und deren Substitution geschätzt werden**

hinein

$$P(Si = x|D = y) = \frac{|Si = x \ddot{y} D = y|}{|D = y|}.$$

Nehmen Sie an, dass für die Variablen Si : P (Si = x|D = y) = 0,01 und dass es 40 Trainingsfälle mit D = y gibt. Dann gibt es mit hoher Wahrscheinlichkeit keinen Trainingsfall mit Si = x und D = y, und wir schätzen P (Si = x|D = y) = 0. Für einen anderen Wert D = z nehmen wir an, dass die Beziehungen ähnlich liegen, aber die Schätzung ergibt Werte größer als Null für alle P (Si = x|D = z). Daher wird immer der Wert D = z bevorzugt, der nicht die tatsächliche Wahrscheinlichkeitsverteilung widerspiegelt. Daher gilt bei der Schätzung von Wahrscheinlichkeiten die Formel

$$P(A|B) \ddot{y} \frac{|A \ddot{y} B|}{|B|} = \frac{\text{schnappen}}{nB}$$

wird ersetzt durch

$$P(A|B) \ddot{y} \frac{nAB + mp}{nB + m},$$

wobei p = P (A) die A-priori-Wahrscheinlichkeit für A ist und m eine frei wählbare Konstante ist, die als „äquivalente Datengröße“ bekannt ist. Je größer m wird, desto größer ist das Gewicht der A-priori-Wahrscheinlichkeit im Vergleich zu dem aus der gemessenen Häufigkeit ermittelten Wert.

## 8.6.1 Textklassifizierung mit Naive Bayes

Naive Bayes ist heute sehr erfolgreich und produktiv in der Textklassifizierung. Seine primäre und zugleich sehr wichtige Anwendung ist die automatische Filterung von E-Mails in gewünschte und unerwünschte bzw. Spam-E-Mails. In Spam-Filtern wie SpamAssassin [Sch04] kommt unter anderem ein naiver Bayes-Klassifikator zum Einsatz, der lernt, gewünschte E-Mails von Spam zu trennen. SpamAssassin ist ein Hybridsystem, das eine anfängliche Filterung mithilfe von Black- und Whitelists durchführt. Schwarze Listen sind Listen blockierter E-Mail-Adressen von Spam-Absendern, deren E-Mails immer gelöscht werden, und weiße Listen sind Listen mit Absendern, deren E-Mails immer zugestellt werden. Nach dieser Vorfilterung werden die verbleibenden E-Mails durch den naiven Bayes-Klassifikator nach ihrem eigentlichen Inhalt, also nach dem Text, klassifiziert. Der erkannte Klassenwert wird dann zusammen mit anderen Attributen aus dem Header der E-Mail wie der Domäne des Absenders, dem MIME-Typ usw. anhand eine

Hier ist die Lernfähigkeit des naiven Bayes-Filters sehr wichtig. Hierzu muss der Benutzer zunächst eine große Anzahl an E-Mails manuell als Wunsch- oder Spam-E-Mails klassifizieren. Anschließend wird der Filter trainiert. Um auf dem neuesten Stand zu bleiben, muss der Filter regelmäßig neu trainiert werden. Hierzu sollte der Benutzer alle E-Mails, die vom Filter fälschlicherweise klassifiziert wurden, korrekt klassifizieren, also in die entsprechenden Ordner ablegen. Der Filter wird dann mit diesen E-Mails kontinuierlich neu trainiert.

Neben der Spam-Filterung gibt es viele weitere Anwendungen zur automatischen Textklassifizierung. Wichtige Anwendungen sind das Filtern unerwünschter Einträge in Internet-Diskussionsforen und das Tracking von Websites mit kriminellen Inhalten wie militanten oder terroristischen Aktivitäten, Kinderpornografie oder Rassismus. Darüber hinaus lassen sich damit auch Suchmaschinen an die Präferenzen des Nutzers anpassen, um die Suchergebnisse besser einzuordnen. Im industriellen und wissenschaftlichen Umfeld steht die unternehmensweite Suche in Datenbanken oder in der Literatur im Vordergrund der Forschung. Durch seine Lernfähigkeit kann sich ein Filter an die Gewohnheiten und Wün

Wir werden die Anwendung von Naive Bayes auf die Textanalyse anhand eines kurzen Beispiels vorstellen ausführlicher Text von Alan Turing aus [Tur50]:

> „Wir können hoffen, dass Maschinen irgendwann in allen rein intellektuellen Bereichen mit Menschen konkurrieren werden. Aber welche sind die besten für den Anfang? Auch das ist eine schwierige Entscheidung. Viele Leute denken, dass eine sehr abstrakte Aktivität, wie das Schachspielen, am besten wäre. Man kann auch behaupten, dass es am besten ist, die Maschine mit den besten Sinnesorganen auszustatten, die man für Geld kaufen kann, und ihr dann beizubringen, Englisch zu verstehen und zu sprechen. Dieser Prozess könnte dem normalen Unterricht eines Kindes folgen. Dinge würden hervorgehoben und benannt usw. Auch hier weiß ich nicht, was die richtige Antwort ist, aber ich denke, dass b

Nehmen wir an, dass Texte wie dieser in zwei Klassen eingeteilt werden sollten: „I“ für „interessant“ und „¬I“ für „uninteressant“. Nehmen wir außerdem an, dass eine Datenbank mit Texten existiert, die bereits klassifiziert sind. Welche Attribute sollen verwendet werden? Bei einem klassischen Ansatz zum Aufbau eines Bayes'schen Netzwerks definieren wir eine Reihe von Attributen wie die Länge des Textes, die durchschnittliche Satzlänge, die relative Häufigkeit bestimmter Satzzeichen, die Häufigkeit mehrerer wichtiger Wörter wie „I“ , „Maschinen“ usw. Bei der Klassifizierung mittels Naive Bayes wird dagegen ein überraschend primitiver Algorithmus gewählt. Für jede der n Wortpositionen im Text wird ein Attribut si definiert. Als mögliche Werte für alle Positionen sind alle im Text vorkommenden Wörter erlaubt . Nun für die Klassen I und ¬I die Werte

$$P(I|s1,...,sn) = c \cdot P(I) \prod_{i=1}^{n} P(si|I) \qquad (8.9)$$

und P (¬I |s1,...,sn) berechnet werden und dann die Klasse mit dem Maximalwert ausgewählt werden. Im obigen Beispiel mit insgesamt 113 Wörtern ergibt dies

$$P(I|s1,...,sn) = c \cdot P(I) \cdot P(s1 = \text{„Wir“}|I) \cdot P(s2 = \text{„kann“}|I) \cdot \dots \cdot P(s113 = \text{„sollte“}|I)$$

Und

$$P(\neg I|s1,...,sn) = c \cdot P(\neg I) \cdot P(s1 = \text{„Wir“}|\neg I) \cdot P(s2 = \text{„kann“}|\neg I) \cdot \dots \cdot P(s113 = \text{„sollte“}|\neg I).$$

Das Lernen hier ist ganz einfach. Die bedingten Wahrscheinlichkeiten P (si |I ), P (si |¬I ) und die A-priori-Wahrscheinlichkeiten P (I ), P (¬I ) müssen einfach berechnet werden. Wir gehen nun zusätzlich davon aus, dass die P (si |I ) unabhängig von der Position im Text sind. Das

bedeutet zum Beispiel das

$$P(s61 = \text{„und“}|l) = P(s69 = \text{„und“}|l) = P(s86 = \text{„und“}|l).$$

Wir könnten also den Ausdruck $P(und|l)$ mit der neuen binären Variablen *und* als Wahrscheinlichkeit des Auftretens von „und“ an einer beliebigen Position verwenden.

Die Umsetzung kann etwas beschleunigt werden, wenn wir die Häufigkeit $n_i$ von finden jedes Wort , das im Text vorkommt, und verwenden Sie die Formel

$$P(l|s_1,\ldots,s_n) = c \cdot P(l) \prod_{i=1}^{l} P(w_i|l)^{n_i} \qquad (8.10)$$

was äquivalent zu (8.9) auf Seite 205 ist. Bitte beachten Sie, dass der Index i im Produkt nur auf die Anzahl l verschiedener Wörter verweist, die im Text vorkommen.

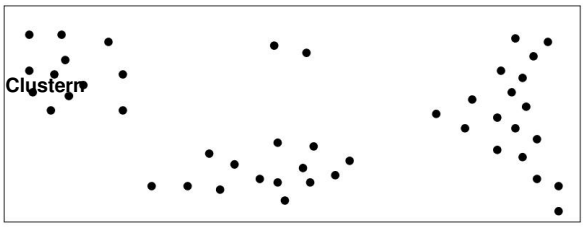
Trotz seiner Einfachheit liefert Naive Bayes hervorragende Ergebnisse für die Textklassifizierung. Spamfilter, die mit Naive Bayes arbeiten, erreichen Fehlerraten von deutlich unter einem Prozent. Die Systeme DSPAM und CRM114 können sogar so gut trainiert werden, dass sie nur eine von 7000 bzw. 8000 E-Mails falsch klassifizieren. Dies entspricht einer Richtigkeit von nahezu 99,99 %.

## 8.7 Clustering

Wenn wir in einer Suchmaschine nach dem Begriff „Mars“ suchen, erhalten wir Ergebnisse wie „der Planet Mars“ und „Schokoladen-, Süßwaren- und Getränkekonglomerat“, die semantisch recht unterschiedlich sind. In der Menge der entdeckten Dokumente gibt es zwei deutlich unterschiedliche *Cluster.* Google beispielsweise listet die Ergebnisse immer noch unstrukturiert auf. Es wäre besser, wenn die Suchmaschine die Cluster trennen und dem Benutzer entsprechend präsentieren würde, da der Benutzer normalerweise nur an einem der Cluster interessiert ist.

Der Unterschied zwischen *Clustering* und überwachtem Lernen besteht darin, dass die Trainingsdaten unbeschriftet sind. Somit fehlt die Vorstrukturierung der Daten durch den Betreuer Vielmehr geht es beim Clustering darum, Strukturen zu finden. Im Bereich der Trainingsdaten finden sich Datenansammlungen wie in Abb. 8.31 auf Seite 207 . In einem Cluster ist der Abstand benachbarter Punkte typischerweise kleiner als der Abstand zwischen Punkten verschiedener Cluster. Daher ist die Wahl einer geeigneten Distanzmetrik für Punkte, also für zu gruppierende Objekte und für Cluster, von grundlegender Bedeutung. Wie zuvor gehen wir im Folgenden davon aus, dass jedes Datenobjekt durch einen Vektor numerischer Attribute beschrieben wird.

Abb. 8.31
Einfaches zweidimensionales Beispiel mit vier klar getrennten Clustern



## 8.7.1 Distanzmetriken

Dementsprechend werden für jede Anwendung die verschiedenen Abstandsmetriken für den Abstand d zwischen zwei Vektoren x und y im Rn definiert. Am gebräuchlichsten ist der euklidische Abstand

$$de(x, y) = \sqrt{\sum_{i=1}^{N} (xi \ddot{y} yi)\, 2.}$$

Etwas einfacher ist die Summe quadratierter Abstände

$$dq(x, y) = \sum_{i=1}^{N} (xi \ddot{y} yi)^{2},$$

was für Algorithmen, in denen nur Abstände verglichen werden, dem euklidischen Abstand entspricht (Übung 8.19 auf Seite 220). Ebenfalls verwendet wird die bereits erwähnte Manhattan-Distanz

$$dm(x, y) = \sum_{i=1}^{N} |xi \ddot{y} yi|$$

sowie der Abstand der maximalen Komponente

$$d\ddot{y}(x, y) = \max_{i=1,\ldots,n} |xi \ddot{y} yi|$$

was auf der Maximalnorm basiert. Bei der Textklassifizierung wird die normalisierte Projektion der beiden Vektoren aufeinander, also das normalisierte Skalarprodukt, verwendet

$$\frac{xy\,|}{x||y|}$$

wird häufig berechnet, wobei |x| ist die euklidische Norm von x. Denn diese Formel ist eine Metrik für die Ähnlichkeit der beiden Vektoren, als Distanzmetrik die Umkehrung

$$ds(x, y) = \frac{|x||y|}{xy}$$

kann verwendet werden, oder „>“ und „<“ können für alle Vergleiche vertauscht werden.
Bei der Suche nach einem Text werden die Attribute x1,...,xn ähnlich wie bei Naive Bayes als Komponenten des Vektors x wie folgt berechnet. Für ein Wörterbuch mit 50.000 Wörtern beträgt der W

entspricht der Häufigkeit des i-ten Wörterbuchworts im Text. Da in einem solchen Vektor normalerweise fast alle Komponenten Null sind, sind bei der Berechnung des Skalarprodukts fast alle Terme der Summation Null. Durch die Nutzung dieser Art von Informationen kann die Implementierung erheblich beschleunigt werden (Übung 8.20 auf Seite 220).

## 8.7.2 k-Means und der EM-Algorithmus

Immer wenn die Anzahl der Cluster bereits im Voraus bekannt ist, kann der *k-Means*-Algorithmus verwendet werden. Wie der Name schon sagt, werden k Cluster durch ihren Durchschnittswert definiert. Zunächst werden die k Clustermittelpunkte ÿ1 ,...,ÿk zufällig oder manuell initialisiert. Anschließend werden die folgenden zwei Schritte wiederholt durchgeführt: • Klassifizierung aller Daten auf ihren nächsten Clustermittelpunkt • Neuberechnung des Clustermittelpunkts.

**Als Algorithmus ergibt sich folgendes Schema:**

```
K-MEANS(x1,..., xn,k)
initialisiert ÿ1 ,...,ÿk (z. B. zufällig)

Wiederholen Sie die Klassifizierung von
   x1 , ... , _ _ _ _ _
```

Die Berechnung des Clustermittelpunkts ÿ für die Punkte x1,..., xl erfolgt durch

$$\ddot{y} = \frac{1}{l} \sum_{i=1}^{l} xi .$$

Die Ausführung an einem Beispiel ist in Abb. 8.32 auf Seite 209 für den Fall zweier Klassen dargestellt. Wir sehen, wie sich nach drei Iterationen die zunächst zufällig ausgewählten Klassenzentren stabilisieren. Obwohl dieser Algorithmus keine Konvergenz garantiert, konvergiert er normalerweise sehr schnell. Das bedeutet, dass die Anzahl der Iterationsschritte typischerweise viel kleiner ist als die Anzahl der Datenpunkte. Seine Komplexität beträgt O(ndkt), wobei n die Gesamtzahl der Punkte, d die Dimensionalität des Merkmalsraums und t die Anzahl der Iterationsschritte ist.

In vielen Fällen stellt die Notwendigkeit, die Anzahl der Unterrichtsstunden im Voraus anzugeben, eine unangenehme Einschränkung dar. Deshalb werden wir als nächstes einen Algorithmus vorstellen, der flexibler ist.

Zuvor erwähnen wir jedoch den *EM-Algorithmus,* der eine kontinuierliche Variante von k-means ist, da er keine feste Zuordnung der Daten zu Klassen vornimmt, sondern für jeden Punkt die Wahrscheinlichkeit seiner Zugehörigkeit zu den zurückgibt verschieden

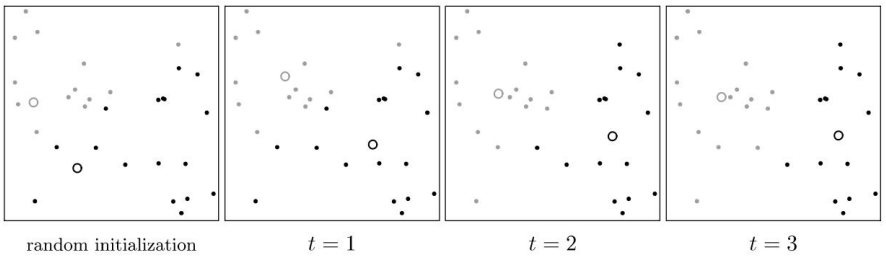



**Abb. 8.32 k-Mittelwerte mit zwei Klassen (k = 2), angewendet auf 30 Datenpunkte. Ganz links ist der Datensatz mit den anfänglichen Zentren und rechts *der* Cluster nach jeder Iteration. Nach drei Iterationen ist Konvergenz erreicht**

**Dabei müssen wir davon ausgehen, dass die Art der Wahrscheinlichkeitsverteilung bekannt ist. Häufig wird die Normalverteilung verwendet. Die Aufgabe des EM-Algorithmus besteht darin, die Parameter (Mittelwert ÿi und Kovarianzmatrizen ÿi der k mehrdimensionalen Normalverteilungen) für jeden Cluster zu bestimmen. Ähnlich wie bei k-means werden die beiden folgenden Schritte wiederholt ausgeführt:**

***Erwartung:*** **Für jeden Datenpunkt wird die Wahrscheinlichkeit P (Cj |xi) berechnet, dass er zu jedem Cluster gehört.**

***Maximierung:*** **Anhand der neu berechneten Wahrscheinlichkeiten werden die Parameter der Verteilung neu berechnet.**

**Dadurch wird eine weichere Clusterbildung erreicht, die in vielen Fällen zu besseren Ergebnissen führt. Dieser Wechsel zwischen Erwartung und Maximierung gibt dem Algorithmus seinen Namen. Neben dem Clustering wird beispielsweise der EM-Algorithmus zum Erlernen von Bayes'schen Netzwerken verwendet [DHS01].**

## 8.7.3 Hierarchisches Clustering

**Beim hierarchischen Clustering beginnen wir mit n Clustern, die jeweils aus einem Punkt bestehen. Dann werden die Cluster der nächsten Nachbarn kombiniert, bis alle Punkte zu einem einzigen Cluster zusammengefasst wurden oder bis ein Abbruchkriterium erreicht wurde. Wir erhalten das Schema**

**HIERARCHICALCLUSTERING(x1,..., xn,k) initialisiert**
**C1 = {x1},...,Cn = {xn}**

**Wiederholen. Finden Sie zwei Cluster Ci und Cj mit dem kleinsten Abstand. Kombinieren**
**Sie Ci und Cj , bis die Beendigungsbedingung**
**erreicht ist. Zurückgeben (Baum mit Clustern).**

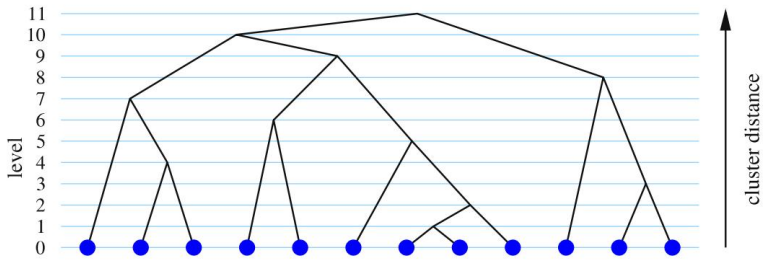



**Abb. 8.33 Beim hierarchischen Clustering werden in jedem Schritt die beiden Cluster mit dem kleinsten Abstand zusammengefasst**

Als Abbruchbedingung könnte beispielsweise eine gewünschte Anzahl von Clustern oder ein maximaler Abstand zwischen Clustern gewählt werden. In Abb. 8.33 ist dieser Algorithmus schematisch als Binärbaum dargestellt, bei dem von unten nach oben in jedem Schritt, also auf jeder Ebene, zwei Teilbäume verbunden sind. Auf der obersten Ebene werden alle Punkte zu einem großen Cluster zusammengefasst.

Bisher ist unklar, wie die Abstände zwischen den Clustern berechnet werden. Tatsächlich haben wir im vorherigen Abschnitt verschiedene Distanzmetriken für Punkte definiert, diese können jedoch nicht auf Clustern verwendet werden. Eine praktische und häufig verwendete Metrik ist der Abstand zwischen den beiden nächstgelegenen Punkten in den beiden Clustern Ci und Cj :

$$d_{min}(C_i, C_j) = \min d(x, y).\ x \ddot{y} C_i,\ y \ddot{y} C_j$$

Damit erhalten wir den *Nearest-Neighbor-Algorithmus,* dessen Anwendung in Abb. 8.34 auf Seite 211 dargestellt ist. Baum.10 [9] Wir sehen, dass dieser Algorithmus eine minimale Spannweite erzeugt Das Beispiel zeigt außerdem, dass die beiden beschriebenen Algorithmen recht unterschiedliche Cluster erzeugen. Dies zeigt uns, dass bei Diagrammen mit Clustern, die nicht klar getrennt sind, das Ergebnis stark vom Algorithmus oder der gewählten Distanzmetrik abhängt.

Für eine effiziente Implementierung dieses Algorithmus erstellen wir zunächst eine Adjazenzmatrix, in der die Abstände zwischen allen Punkten gespeichert werden, was O(n2 ) Zeit und Speicher benötigt. Wenn die Anzahl der Cluster keine Obergrenze hat, wird die Schleife n ÿ 1 Mal durchlaufen und die asymptotische Berechnungszeit beträgt O(n3 ).

Um den Abstand zwischen zwei Clustern zu berechnen, können wir auch den Abstand be verwenden zwischen den beiden am weitesten entfernten Punkten

$$d_{max}(C_i, C_j) = \max d(x, y)\ x \ddot{y} C_i, y \ddot{y} C_j$$

9Der Nearest-Neighbor-Algorithmus ist nicht mit der Nearest-Neighbor-Methode für Klassen zu verwechseln Sifizierung aus Abschn. 8.3.

10Ein minimaler Spannbaum ist ein azyklischer, ungerichteter Graph mit der minimalen Summe der Kantenlängen.

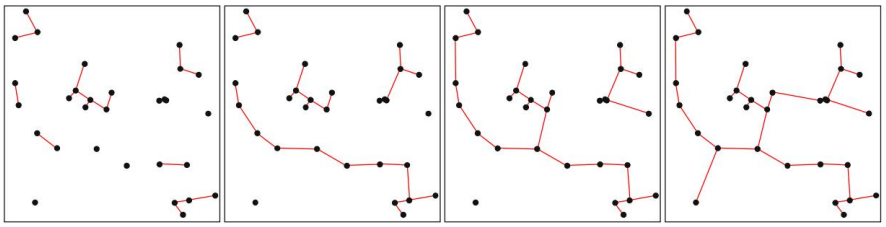



**Abb. 8.34 Der Algorithmus für den nächsten Nachbarn, angewendet auf die Daten aus Abb. 8.32 auf Seite 209 auf verschiedenen Ebenen mit 12, 6, 3, 1 Clustern**

**und erhalten Sie den *Algorithmus für den am weitesten entfernten Nachbarn.* Alternativ wird der Abstand des Clustermittelpunkts dÿ(Ci, Cj ) = d(ÿi,ÿj ) verwendet. Neben dem Clustering-Algorithmus Hier vorgestellt, gibt es viele andere, für die wir den Leser auf [DHS01] verweisen. zum weiteren Studium.**

## 8.8 Data Mining in der Praxis

**Alle bisher vorgestellten Lernalgorithmen können als Werkzeuge für das Data Mining verwendet werden. Für Für den Benutzer ist es jedoch manchmal recht mühsam, sich an neue Software zu gewöhnen Tools für jede Anwendung bereitzustellen und darüber hinaus die zu analysierenden Daten in die zu übertragen für jeden Einzelfall das passende Format.**

**Eine Reihe von Data-Mining-Systemen befasst sich mit diesen Problemen. Die meisten dieser Systeme bieten eine komfortable grafische Benutzeroberfläche mit vielfältigen Tools zur Visualisierung der Daten, zur Vorverarbeitung, z. B. Manipulation fehlender Werte, und zur Analyse. Für Zur Analyse kommen unter anderem die hier vorgestellten Lernalgorithmen zum Einsatz.**

**Besonders hervorzuheben ist die umfangreiche Open-Source-Java-Bibliothek WEKA. Es bietet eine große Anzahl an Algorithmen und vereinfacht die Entwicklung neuer Algorithmen.**

**Das frei verfügbare System KNIME, das wir im Folgenden kurz vorstellen Der folgende Abschnitt bietet eine praktische Benutzeroberfläche und alle Arten von Tools oben erwähnt. KNIME verwendet auch WEKA-Module. Darüber hinaus bietet es eine Einfache Möglichkeit, den Datenfluss der gewählten Visualisierungs-, Vorverarbeitungs- und Analysetools mit einem grafischen Editor zu steuern. Mittlerweile bieten eine Vielzahl anderer Systeme durchaus ähnliche Funktionalitäten, etwa das Open-Source-Projekt RapidMiner (www.rapidminer.com), das System Clementine (www.spss.com/clementine), verkauft von SPSS, und das KXEN-Analyse-Framework (www.kxen.com).**

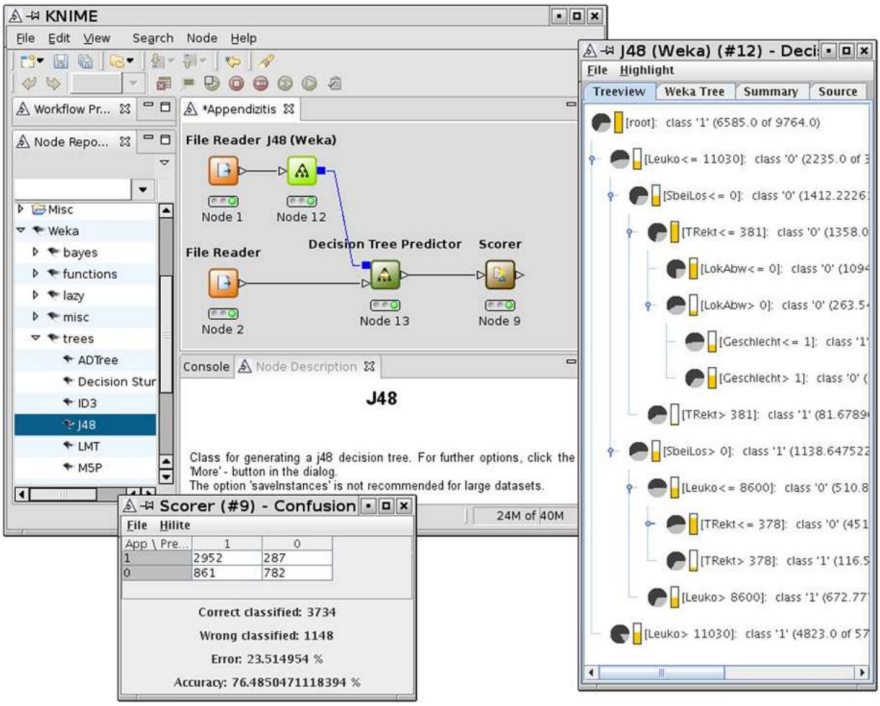



**Abb. 8.35 Die KNIME-Benutzeroberfläche mit zwei zusätzlichen Ansichten, die den Entscheidungsbaum und die Verwirrungsmatrix anzeigen**

## 8.8.1 Das Data-Mining-Tool KNIME

Anhand der LEXMED- Daten zeigen wir nun, wie man mit *KNIME* (Konstanz Information Miner, www.knime.org) Wissen aus Daten extrahieren kann. Zunächst erstellen wir einen Entscheidungsbaum wie in Abb. 8.35 dargestellt. Nach dem Anlegen eines neuen Projekts wird ein Workflow grafisch aufgebaut. Dazu werden die entsprechenden Tools einfach mit der Maus aus dem Node-Repository entnommen und in das Haupt-Workflow-Fenster gezo

Die Trainings- und Testdaten aus der C4.5-Datei können mit den beiden File-Reader-Knoten problemlos eingelesen werden. Diese Knoten können jedoch auch ganz einfach für andere Dateiformate konfiguriert werden. Die seitliche Ampel unter dem Knoten zeigt seinen Status an (nicht bereit, konfiguriert, ausgeführt). Anschließend wird Knoten J48 aus der WEKA-Bibliothek [WF01] ausgewählt, die eine Java-Implementierung von C4.5 enthält. Die Konfiguration hierfür ist recht einfach. Nun wird ein Prädiktorknoten ausgewählt, der den generierten Baum auf die Testdaten anwendet. Es fügt eine neue Spalte mit der vom Baum generierten Klassifizierung in die Testdatentabelle „Vorhersage“ ein. Daraus berechnet der Scorer-Knoten die in der Abbildung dargestellte Verwirrungsmatrix, die die Anzahl der korrekt klassifizierten Fälle für beide Klassen in der Diagonale und zusätzlich die Anzahl der falsch positiven und falsch negativen Datenpunkte angib

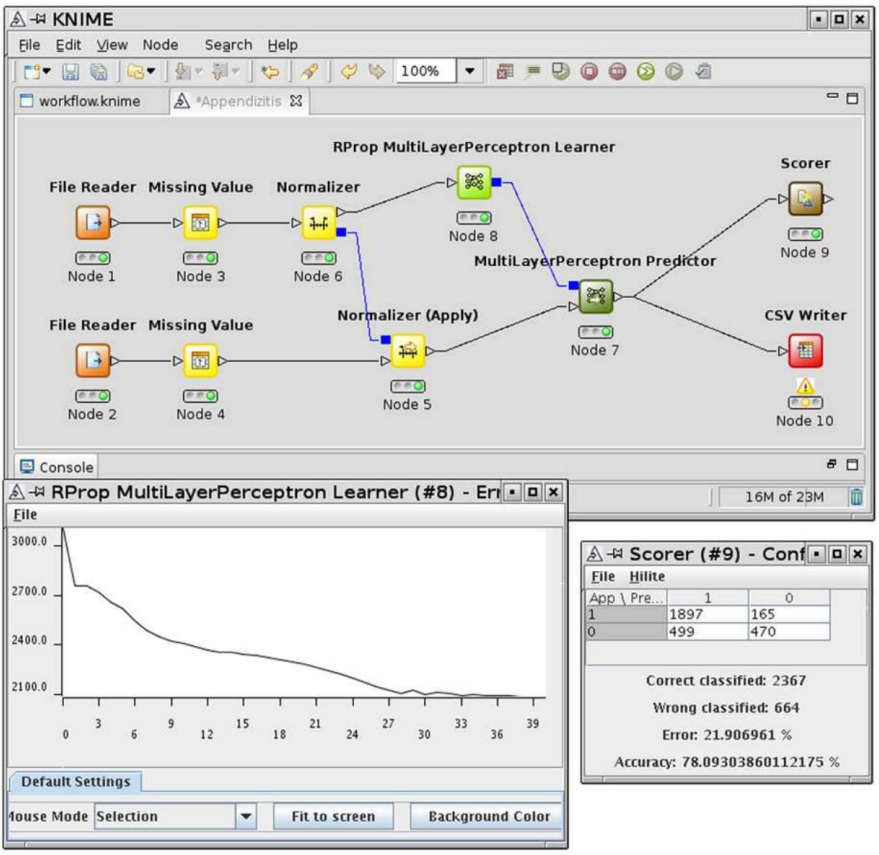



**Abb. 8.36 Die KNIME-Benutzeroberfläche mit dem Workflow-Fenster, der Lernkurve und der Verwirrungsmatrix**

Sobald der Fluss vollständig aufgebaut und alle Knoten konfiguriert sind, folgt ein beliebiger Knoten ausgeführt werden kann. Es stellt automatisch sicher, dass Vorgängerknoten ausgeführt werden, wenn notwendig. Der J48-Knoten generiert die rechts dargestellte Ansicht des Entscheidungsbaums der Figur. Dieser Baum ist identisch mit dem von C4.5 in Abschn. 8.4.5, obwohl hier der Knoten TRekt<=378 kollabiert dargestellt ist.

Zum Vergleich dient ein Projekt zum Erlernen eines mehrschichtigen Perzeptrons (siehe Abschn. 9.5). in Abb. 8.36 dargestellt . Dies funktioniert ähnlich wie das zuvor eingeführte lineare Perceptron, kann aber auch nicht linear trennbare Klassen unterteilen. Hier ist der Fluss etwas komplexer. Für die Fehlerbehandlung fehlender Werte ist ein zusätzlicher Knoten erforderlich Vorverarbeitung jeder der beiden Dateien. Wir haben es so eingestellt, dass diese Zeilen gelöscht werden. Da neuronale Netze nicht mit beliebigen Werten, den Werten aller Variablen, umgehen können werden mithilfe des „Normalizer“-Knotens linear in das Intervall [0, 1] skaliert.

Nach der Anwendung des RProp-Lerners wurde eine Verbesserung der Backpropagation (siehe Sekte. 9.5) können wir den zeitlichen Verlauf des Approximationsfehlers analysieren die gezeigte Lernkurve. In der Verwirrungsmatrix gibt der Bewerter die Analyse aus der Testdaten. Der Knoten „CSV Writer“ unten rechts dient zum Exportieren der Ergebnisdateien, die dann extern verwendet werden können, um die in gezeigte ROC-Kurve zu generieren Abb. 7.10 auf Seite 142, für die es leider (noch) kein KNIME-Tool gibt.

Zusammenfassend können wir über KNIME (und ähnliche Tools) Folgendes sagen: z Projekte mit Datenanalyseanforderungen, die nicht zu exotisch sind, lohnt sich mit einer so leistungsstarken Workbench zur Datenanalyse zu arbeiten. Der Benutzer speichert bereits viel Zeit mit der Vorverarbeitungsphase. Es gibt eine große Auswahl an leicht verwendbaren „Mining-Tools“, wie z. B. Klassifikatoren für den nächsten Nachbarn, einfache Bayesian-Netzwerk-Lernalgorithmen sowie der k-Means-Clustering-Algorithmus (Abschn. 8.7.2). Auswertung der Ergebnisse, beispielsweise eine Kreuzvalidierung, können einfach durchgeführt werden. Es bleibt dabei Es sei erwähnt, dass es neben den gezeigten noch viele andere Tools zur Visualisierung der Daten gibt. Darüber hinaus haben die Entwickler von KNIME eine Erweiterung vorgenommen KNIME verfügbar, mit dem der Anwender eigene Tools in Java oder Python programmieren kann.

Es sollte jedoch auch erwähnt werden, dass die Benutzer dieser Art von Data Mining System sollte solide Vorkenntnisse über maschinelles Lernen und deren Nutzung mitbringen von Data-Mining-Techniken. Die Software allein kann keine Daten analysieren, wohl aber die Hand Für einen Spezialisten wird es zu einem leistungsstarken Werkzeug zur Gewinnung von Wissen aus Daten. Für den Einsteiger in das faszinierende Gebiet des maschinellen Lernens bietet sich ein solches System an eine ideale und einfache Möglichkeit, sein Wissen praktisch zu testen und zu vergleichen die verschiedenen Algorithmen. Der Leser kann dies in Übung 8.21 auf Seite 220 überprüfen.

## 8.9 Zusammenfassung

Wir haben mehrere Algorithmen aus dem etablierten Bereich des überwachten Lernens ausführlich behandelt, darunter Entscheidungsbaumlernen, Bayesianische Netzwerke und die nächstgelegenen Nachbarmethode. Diese Algorithmen sind in verschiedenen Anwendungen stabil und effizient einsetzbar und gehören somit zum Standardrepertoire in der KI und im Data Mining. Der Gleiches gilt für die Clustering-Algorithmen, die ohne „Supervisor“ und funktionieren finden sich beispielsweise in Suchmaschinenanwendungen. Verstärkungslernen da ein anderer Bereich des maschinellen Lernens ebenfalls keinen Vorgesetzten benötigt. Im Gegensatz zum überwachten Lernen, bei dem der Lernende die richtigen Aktionen oder Antworten als Etiketten erhä In den Trainingsdaten erhält man beim Reinforcement Learning nur hin und wieder positives oder negatives Feedback aus der Umgebung. Im Kap. 10 zeigen wir, wie das geht funktioniert. Nicht ganz so schwer ist die Aufgabe beim teilüberwachten Lernen, einem jungen Teilgebiet des maschinellen Lernens, wo nur sehr wenige aus einer großen Menge an Trainingsdaten vorhanden sind beschriftet.

Überwachtes Lernen ist mittlerweile ein etablierter Bereich mit vielen erfolgreichen Anwendungen. Für das überwachte Lernen von Daten mit kontinuierlichen Beschriftungen kann jeder Funktionsnäherungsalgorithmus verwendet werden. Daher gibt es viele Algorithmen aus verschiedenen Bereic Bereiche Mathematik und Informatik. In Abschn. 9 werden wir verschiedene vorstellen Arten neuronaler Netze, Algorithmen der kleinsten Quadrate und Support-Vektor-Maschinen,

die allesamt Funktionsnäherungen sind. Heutzutage erfreuen sich Gauß-Prozesse großer Beliebtheit, da sie sehr universell und einfach anzuwenden sind und dem Benutzer eine Schätzung der Unsicherheit der Ausgabewerte liefern [RW06].

Die folgende Taxonomie gibt einen Überblick über die wichtigsten Lernalgorithmen und deren Klassifizierung.

*Überwachtes Lernen*

- Faules Lernen
  - k-Nearest-Neighbor-Methode (Klassifizierung + Näherung)
  - Lokal gewichtete Regression (Näherung)
  - Fallbasiertes Lernen (Klassifizierung + Approximation) •
- Eager Learning –
  - Entscheidungsbaum-Induktion (Klassifizierung)
  - Erlernen von Bayes'schen Netzwerken (Klassifizierung + Approximation)
  - Neuronale Netze (Klassifikation + Approximation)
  - Gaußsche Prozesse (Klassifikation + Näherung)
  - Wavelets, Splines, radiale Basisfunktionen, . . .

*Unüberwachtes Lernen (Clustering)*

- Nearest-Neighbor-Algorithmus
- Farthest-Neighbor-Algorithmus
- k-means •
- Neuronale Netze

*Reinforcement Learning*

- Value Iteration •
- Q Learning •
- TD Learning •
- Policy-Gradient-Methoden •
- Neuronale Netze

Was über überwachtes Lernen gesagt wurde, ist jedoch nur wahr , wenn mit einem festen Satz bekannter Attribute gearbeitet wird. Ein interessantes, noch offenes Feld, auf dem intensiv geforscht wird, ist die automatische Funktionsauswahl. In Abschn. Für das Lernen mit Entscheidungsbäumen haben wir in Abschn. 8.4 einen Algorithmus zur Berechnung des Informationsgewinns von Attributen vorgestellt, der die Attribute nach ihrer Relevanz sortiert und nur diejenigen verwendet, die die Qualität der Klassifizierung verbessern . Mit dieser Art von Methode ist es möglich, die relevanten Attribute automatisch aus einer p Dieses Basisset muss jedoch manuell ausgewählt werden.

Noch offen und auch nicht klar definiert ist die Frage, wie die Maschine neue Attribute finden kann. Stellen wir uns einen Roboter vor, der Äpfel pflücken soll. Dazu muss er lernen, zwischen reifen und unreifen Äpfeln und anderen Gegenständen zu unterscheiden. Traditionell würden wir bestimmte Attribute wie die Farbe und Form von Pixelregionen bestimmen und dann einen Lernalgorithmus anhand manuell klassifizierter Bilder trainieren. Es ist auch möglich, dass beispielsweise ein neuronales Netz direkt mit allen Pixeln des Bildes als Eingabe trainiert wird, was bei hoher Auflösung allerdings mit erheblichen Rechenzeitproblemen verbunden ist. Wünschenswert wären hier Ansätze, die automatisch Vorschläge für relevante Features machen. Aber das ist immer noch Science-Fiction.

Clustering bietet einen Ansatz zur Merkmalsauswahl. Bevor wir die Apfelerkennungsmaschine trainieren, lassen wir das Clustering auf den Daten laufen. Für das (überwachte) Lernen der Klassen *Apple* und *Non Apple* sind die Eingabe nicht mehr alle Pixel, sondern nur noch die beim Clustering gefundenen Klassen, möglicherweise zusammen mit anderen Attributen. Clustering kann jedenfalls zur automatischen, kreativen „Entdeckung“ von Features genutzt werden. Es ist jedoch ungewiss, ob die entdeckten Merkmale relevant sind.

Noch schwieriger ist folgendes Problem: Gehen Sie davon aus, dass die zur Apfelerkennung verwendete Videokamera nur Schwarzweißbilder überträgt. Die Aufgabe lässt sich nicht mehr gut lösen. Es wäre schön, wenn die Maschine von sich aus kreativ wäre, indem sie beispielsweise vorschlägt, die Kamera durch eine Farbkamera zu ersetzen.
Das wäre heute zu viel verlangt.

Neben Fachwerken zu allen Teilgebieten des maschinellen Lernens gibt es die hervorragenden Lehrbücher [Mit97, Bis06, Alp04, DHS01, HTF09]. Aktuelle Forschungsergebnisse finden Sie im frei verfügbaren Journal of Machine Learning Research (http://jmlr.csail.mit.edu), im Machine Learning Journal sowie in den Tagungsbänden der International Conference on Machine Learning (ICML). ) ist empfohlen. Für jeden Entwickler von Lernalgorithmen ist das Machine Learning Repository [DNM98] der University of California at Irvine (UCI) mit seiner großen Sammlung an Trainings- und Testdaten für Lernalgorithmen und Data-Mining-Tools interessant. MLOSS steht für Machine Learning Open Source Software und ist ein hervorragendes Verzeichnis mit Links zu frei verfügbarer Software (www.mloss.org).

## 8.10 Übungen

### 8.10.1 Einführung

**Aufgabe 8.1**

(a) Geben Sie die Aufgabe eines Agenten an, der anhand der Messwerte für Temperatur, Luftdruck und Luftfeuchtigkeit das Wetter für den nächsten Tag vorhersagen soll. Das Wetter sollte in eine der drei Klassen eingeteilt werden: sonnig, bewölkt und regnerisch.
(b) Beschreiben Sie den Aufbau einer Datei mit den Trainingsdaten für diesen Agenten.

**Aufgabe 8.2** Zeigen Sie, dass die Korrelationsmatrix symmetrisch ist und dass alle Diagonalelemente gleich 1 sind.

### 8.10.2 Das Perzeptron

**Übung 8.3** Wenden Sie die Perzeptron-Lernregel auf die Mengen an

$$M+ = \{(0, 1{,}8), (2, 0{,}6)\} \text{ und } M\ddot{y} = \{(\ddot{y}1{,}2, 1{,}4), (0{,}4, \ddot{y}1)\}$$

aus Beispiel 8.2 auf Seite 172 und geben Sie das Ergebnis der Werte des Gewichtsvektors an.

**Übung 8.4** Gegeben sei die Tabelle rechts mit den Trainingsdaten: (a) Zeigen Sie anhand eines Diagramms, dass die Daten linear trennbar sind. (b) Bestimmen Sie manuell die Gewichte w1 und w2 sowie den Schwellenwert ÿ eines Perzeptrons (mit Schwellenwert), das die Daten korrekt klassifiziert. (c) Programmieren Sie die Perzeptron-Lernregel und wenden Sie Ihr Programm auf die Tabelle an. Vergleichen Sie die ermittelten Gewichte mit den manuell berechneten.

| Num. | x1 | x2 | Klasse |
|---|---|---|---|
| 1 | 6 | 10 | ⊖ |
| 2 | 7 | 30 | ⊖ |
| 3 | 8 | 20 | ⊕ |
| 4 | 9 | 00 | ⊖ |
| 5 | 8 | 41 | ⊕ |
| 6 | 8 | 61 | ⊖ |
| 7 | 9 | 21 | ⊕ |
| 8 | 9 | 51 | ⊖ |

**Aufgabe 8.5**

(a) Geben Sie eine visuelle Interpretation der heuristischen Initialisierung

$$w0 = \frac{\sum_{xi \in M+} xi}{|M+|} - \frac{\sum_{xi \in M-} xi}{|M-|},$$

des in Abschn. 2.1 beschriebenen Gewichtsvektors 8.2.2.

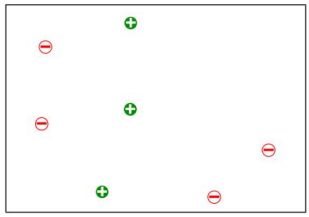
(b) Geben Sie ein Beispiel für einen linear trennbaren Datensatz an, für den diese Heuristik keine Trennlinie erzeugt.

## 8.10.3 Methode des nächsten Nachbarn

**Aufgabe 8.6**

(a) Zeigen Sie das Voronoi-Diagramm für benachbarte Punktmengen.

(b) Zeichnen Sie dann die Klassentrennungslinien ein.



Aufgabe 8.7 **Gegeben sei die Tabelle mit Trainingsdaten aus Aufgabe 8.4 . Verwenden Sie im Folgenden den Manhattan-Abstand d(a, b), definiert als d(a, b) = |a1 ÿ b1| + |a2 ÿ b2|, um den Abstand d zwischen zwei Datenpunkten a = (a1, a2) und b = (b1, b2) zu bestimmen. (a) Klassifizieren Sie den Vektor v = (8, 3,5) mit der Methode des nächsten Nachbarn. (b) Klassifizieren Sie den Vektor v = (8, 3.5) mit der k-Nächste-Nachbarn-Methode für k = 2, 3, 5.**

**ÿ Übung 8.8**

(a) Zeigen Sie, dass es in einem zweidimensionalen Merkmalsraum sinnvoll ist, wie in (8.3) auf Seite 180 behauptet, die k nächsten Nachbarn mit dem Kehrwert des quadrierten Abstands zu gewichten.

(b) Warum sollte eine Gewichtung mit $w' = \frac{1}{1+\ddot{y}d(x,xi)}$ macht weniger Sinn?

### 8.10.4 Entscheidungsbäume

**Aufgabe 8.9 Zeigen Sie, dass die Definition 0 log2 0 := 0 sinnvoll ist, mit anderen Worten, dass die Funktion f (x) = x log2 x dadurch im Ursprung stetig wird.**

**Aufgabe 8.10 Bestimmen Sie die Entropie für die folgenden Verteilungen.**

(a) (1, 0, 0, 0, 0) (B) $\frac{1}{2}, \frac{1}{2}, 0, 0, 0$ (C) $\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, 0, 0$

(D) $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}$ (e) $\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}$ (F) $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \ldots$

**Aufgabe 8.11**

**(a) Zeigen Sie, dass sich die beiden unterschiedlichen Definitionen der Entropie aus (7.9) auf Seite 124 und Definition 8.4 auf Seite 188 nur um einen konstanten Faktor unterscheiden, nämlich das**

$$\sum_{i=1}^{N} pi \log_2 pi = c \sum_{i=1}^{N} pi \ln pi$$

**und geben Sie die Konstante**

**c an. (b) Zeigen Sie, dass es für die MaxEnt-Methode und für Entscheidungsbäume keinen Unterschied macht, welche der beiden Formeln wir verwenden.**

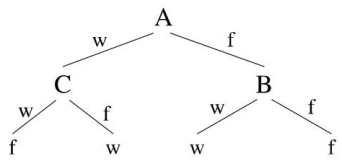
**Aufgabe 8.12 Entwickeln Sie einen Entscheidungsbaum für den Datensatz D aus Übung 8.4 auf Seite 217.**

**(a) Behandeln Sie beide Attribute als**

**diskret. (b) Behandeln Sie nun das Attribut x2 als kontinuierlich und**

**x1 als diskret. (c) Lassen Sie C4.5 einen Baum mit beiden Varianten generieren. Verwenden Sie -m 1 -t 10 als Parameter, um unterschiedliche Vorschläge zu erhalten.**

**Übung 8.13 Gegeben seien der folgende Entscheidungsbaum und die folgenden Tabellen für Trainings- und Testdaten:**




| Trainingsdaten |
|---|
| ABC-Klasse |
| ttft |
| tfft |
| tttf |
| ffft |
| ffff |
| fttt |

| Testdaten |
|---|
| ABC-Klasse |
| ttft |
| tfft |
| ftff |
| ffft |

**(a) Geben Sie die Korrektheit des Baums für die Trainings- und Testdaten**

**an. (b) Geben Sie eine Aussagelogikformel an, die dem Baum**

**entspricht. (c) Beschneiden Sie den Baum, zeichnen Sie den resultierenden Baum und geben Sie seine Richtig für die Trainings- und Testdaten.**

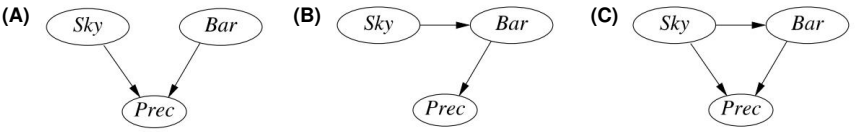
ÿ Übung 8.14

(a) Der Algorithmus zur Generierung von Entscheidungsbäumen (Abb. 8.26 auf Seite 193) eliminiert bei der Ermittlung des aktuellen Attributs nicht die Attribute, die bereits weiter oben im Baum verwendet wurden. Trotzdem kommt ein diskretes Attribut in einem Pfad höchstens einmal vor. Warum?

(b) Warum können kontinuierliche Attribute mehrfach vorkommen?

## 8.10.5 Lernen von Bayes'schen Netzwerken

Aufgabe 8.15 Verwenden Sie die in Übung 7.3 auf Seite 158 angegebene Verteilung und bestimmen Sie die CPTs für die drei Bayes'schen Netzwerke:




(d) Bestimmen Sie die Verteilung für die beiden Netzwerke aus (a) und (b) und vergleichen Sie diese mit der Originalverteilung. Welches Netzwerk ist „besser“?

(e) Bestimmen Sie nun die Verteilung für Netzwerk (c). Was fällt Ihnen ein? Jus tify!

ÿ ÿ Übung 8.16 Zeigen Sie, dass für binäre Variablen S1,...,Sn und die binäre Klassenvariable K ein linearer Score der Form vorliegt

$$\text{Entscheidung} = \begin{cases} \text{Positiv, wenn } w1S1 + \cdots + wnSn > ÿ, \text{ sonst} \\ \text{negativ} \end{cases}$$

ist in Bezug auf das Perzeptron und den naiven Bayes-Klassifikator gleichermaßen ausdrucksstark, die beide nach der Formel entscheiden

$$\text{Entscheidung} = \begin{cases} \text{positiv, wenn } P(K|S1,...,Sn) > 1/2, \text{ sonst} \\ \text{negativ} \end{cases}$$

Übung 8.17 Bei der Implementierung der Textklassifizierung mit Naive Bayes kann es schnell zu einem Exponenunterlauf kommen, da die Faktoren P (wi |K) (die in (8.10) auf Seite 206 erscheinen ) typischerweise alle sehr klein sind, was dazu führen kann, dass sie extrem klein sind Ergebnisse. Wie können wir dieses Problem entschärfen?

ÿ ÿ Übung 8.18 Schreiben Sie ein Programm zur naiven Bayes-Textanalyse. Anschließend trainieren und testen Sie es anhand von Text-Benchmarks mit einem Tool Ihrer Wahl. Das Zählen der Häufigkeit von Wörtern im Text kann unter Linux mit dem Befehl ganz einfach durchgeführt werden

cat <datei> | tr -d „[:punct:]“ | tr -s „[:space:]“ „\n“ | sortieren | uniq -ci

Erhalten Sie die Twenty Newsgroups-Daten von Tom Mitchell in der Benchmark-Sammlung für maschinelles Lernen der UCI (Machine Learning Repository) [DNM98]. Dort finden Sie auch einen Hinweis auf ein naives Bayes-Programm zur Textklassifizierung von Mitchell.

## 8.10.6 Clustering

Aufgabe 8.19 Zeigen Sie, dass für Algorithmen, die nur Distanzen vergleichen, die Anwendung einer streng monoton steigenden Funktion f auf die Distanz keinen Unterschied macht. Mit anderen Worten müssen Sie zeigen, dass der Abstand d1(x, y) und der Abstand d2(x, y) := f (d1(x, y)) bezüglich der Ordnungsrelation zum gleichen Ergebnis führen.

Aufgabe 8.20 Bestimmen Sie die Abstände ds (Skalarprodukt) der folgenden Texte zueinander. x1: Wir werden kurz

die Anwendung von Naive Bayes auf die Textanalyse vorstellen
Beispieltext von Alan Turing aus [Tur50].

x2: Wir können hoffen, dass Maschinen irgendwann überhaupt mit Männern konkurrieren werden geistige Felder. Aber welche sind die besten für den Anfang?

x3: Auch hier weiß ich nicht, was die richtige Antwort ist, aber ich denke, beide Ansätze sollten es tun müde sein.

## 8.10.7 Data Mining

Übung 8.21 Verwenden Sie KNIME (www.knime.de) und (a)

Laden Sie die Beispieldatei mit den Iris-Daten aus dem KNIME-Verzeichnis und experimentieren Sie mit den verschiedenen Datendarstellungen, insbesondere mit den Streudiagrammen.

(b) Trainieren Sie zunächst einen Entscheidungsbaum für die drei Klassen und dann ein RProp-Netzwerk. (c) Laden Sie die Blinddarmentzündungsdaten auf die Website dieses Buches. Vergleichen Sie die Klassifizierungsqualität der Methode „k nächster Nachbar“ mit der eines RProp-Netzwerks. Optimieren Sie k sowie die Anzahl der versteckten Neuronen des RProp-Netzwerks. (d) Besorgen Sie sich einen Datensatz Ihrer Wahl aus der UCI-Datensammlung für Data Mining unter http://kdd.ics.uci.edu oder für maschinelles Lernen unter http://mlearn.ics.uci.edu/MLRepository.html und experimentiere damit.

# Neuronale Netze

# 9

Neuronale Netze sind Netzwerke von Nervenzellen im Gehirn von Menschen und Tieren. Das menschliche Gehirn verfügt über etwa 100 Milliarden Nervenzellen. Wir Menschen verdanken unsere Intelligenz und unsere Fähigkeit, verschiedene motorische und intellektuelle Fähigkeiten zu erlernen, den komplexen Relais und der Adaptivität des Gehirns. Seit vielen Jahrhunderten versuchen Biologen, Psychologen und Ärzte zu verstehen, wie das Gehirn funktioniert. Um 1900 kam die revolutionäre Erkenntnis, dass diese winzigen physischen Bausteine des Gehirns, die Nervenzellen und ihre Verbindungen, für Bewusstsein, Assoziationen, Gedanken, Bewusstsein und Lernfähigkeit verantwortlich sind.

Der erste große Schritt hin zu neuronalen Netzen in der KI wurde 1943 von McCulloch und Pitts in einem Artikel mit dem Titel „Ein logischer Kalkül der der Nervenaktivität immanenten Ideen“ gemacht [AR88]. Sie waren die ersten, die ein mathematisches Modell des Neurons als grundlegendes Schaltelement des Gehirns vorstellten. Dieser Artikel legte den Grundstein für den Aufbau künstlicher neuronaler Netze und damit für diesen sehr wichtigen Zweig der KI.

Wir könnten den Bereich der Modellierung und Simulation neuronaler Netze als den Zweig der Bionik innerhalb der KI betrachten.1 Nahezu alle Bereiche der KI versuchen, kognitive Prozesse nachzubilden, beispielsweise in der Logik oder im probabilistischen Denken. Allerdings haben die zur Modellierung verwendeten Werkzeuge – nämlich Mathematik, Programmiersprachen und digitale Computer – nur sehr wenig mit dem menschlichen Gehirn gemein. Bei künstlichen neuronalen Netzen ist der Ansatz anders. Ausgehend von Erkenntnissen über die Funktion natürlicher neuronaler Netze versuchen wir, diese in Hardware zu modellieren, zu simulieren und sogar zu rekonstruieren. Jeder Forscher auf diesem Gebiet steht vor der faszinierenden und spannenden Herausforderung, Ergebnisse mi

In diesem Kapitel werden wir versuchen, den historischen Fortschritt zu skizzieren, indem wir ausgehend von den wichtigsten biologischen Erkenntnissen ein Modell des Neurons und seiner Interkonnektivität definieren. Anschließend stellen wir einige wichtige und grundlegende Modelle vor: das Hopfield-Modell, zwei einfache assoziative Speichermodelle und den – in der Praxis überaus wichtigen – Backpropagation-Algorithmus.

1Bionics beschäftigt sich mit der Erschließung der „Entdeckungen der belebten Natur“ und ihrer innovativen Umsetzung in Technologie [Wik10].

W. Ertel, *Einführung in die künstliche Intelligenz,*
Bachelor-Themen in Informatik, DOI
10.1007/978-0-85729-299-5_9, 

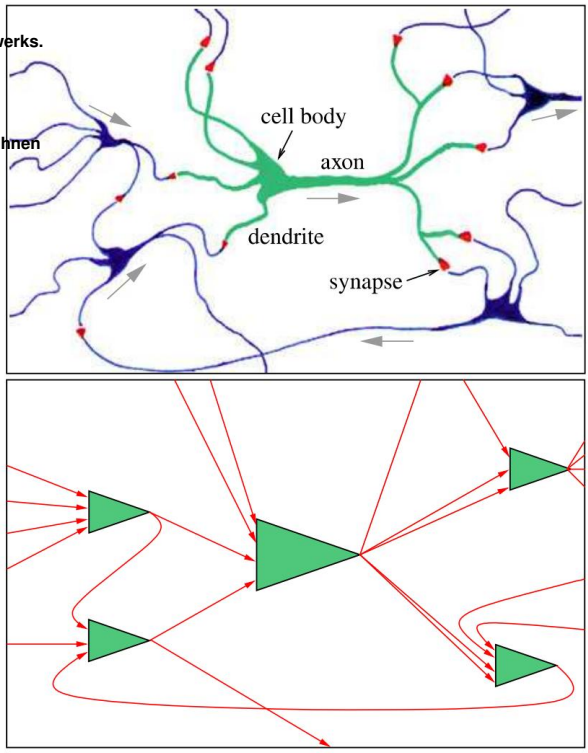



**Abb. 9.1 Zwei Phasen der Modellierung eines neuronalen Netzwerks. *Oben* ein biologisches Modell und *unten* ein formales Modell mit Neuronen und gerichteten Verbindungen zwischen ihnen**

## 9.1 Von der Biologie zur Simulation

Jedes der rund 100 Milliarden Neuronen im menschlichen Gehirn hat, wie in Abb. 9.1 **vereinfacht dargestellt , die folgende Struktur und Funktion. Neben dem Zellkörper verfügt das Neuron über ein Axon, das über die Dendriten lokale Verbindungen zu anderen Neuronen herstellen kann. Das Axon kann jedoch bis zu einem Meter lang in Form einer Nervenfaser durch den Körper wachsen.**

**Der Zellkörper des Neurons kann kleine elektrische Ladungen speichern, ähnlich wie ein Kondensator oder eine Batterie. Dieser Speicher wird durch eingehende elektrische Impulse anderer Neuronen geladen. Je mehr elektrische Impulse eingehen, desto höher ist die Spannung. Wenn die Spannung einen bestimmten Schwellenwert überschreitet, feuert das Neuron. Das bedeutet, dass es seinen Vorrat entlädt, indem es einen Impuls über das Axon und die Synapsen sendet. Der elektrische Strom teilt sich und gelangt über die Synapsen zu vielen anderen Neuronen, in denen der gleiche Vorgang abläuft.**

**Nun stellt sich die Frage nach der Struktur des neuronalen Netzwerks. Jedes der etwa 1011 Neuronen im Gehirn ist mit etwa 1000 bis 10.000 anderen Neuronen verbunden, was insgesamt über 1014 Verbindungen ergibt. Wenn wir weiter bedenken, dass diese gigantische Anzahl extrem dünner Verbindungen aus weichem, dreidimensionalem Gewebe besteht und Experimente am menschlichen Gehirn nicht einfach durchzuführen sind, dann wird klar, warum wir keinen detaillierten Schaltplan davon haben das Gehirn. Vermutlich**

Wir werden niemals in der Lage sein, den Schaltplan unseres Gehirns allein aufgrund seiner immensen Größe vollständig zu verstehen.

Aus heutiger Sicht lohnt es sich nicht mehr, einen vollständigen Schaltplan des Gehirns zu erstellen, da die Struktur des Gehirns adaptiv ist. Es verändert sich spontan und passt sich den Aktivitäten und Umwelteinflüssen des Einzelnen an. Die zentrale Rolle spielen dabei die Synapsen, die die Verbindung zwischen Neuronen herstellen. An der Verbindungsstelle zwischen zwei Neuronen ist es, als ob sich zwei Kabel treffen. Allerdings sind die beiden Leitungen nicht perfekt leitend miteinander verbunden, vielmehr besteht eine kleine Lücke, über die die Elektronen nicht direkt springen können. Diese Lücke wird mit chemischen Substanzen, sogenannten Neurotransmittern, gefüllt. Diese können durch eine angelegte Spannung ionisiert werden und dann eine Ladung über die Lücke transportieren. Die Leitfähigkeit dieser Lücke hängt von vielen Parametern ab, beispielsweise der Konzentration und der chemischen Zusammensetzung des Neurotransmitters. Aufschlussreich ist, dass die Funktion des Gehirns sehr empfindlich auf Veränderungen dieser synaptischen Verbindung reagiert, beispielsweise durch den Einfluss von Alko

Wie funktioniert das Lernen in einem solchen neuronalen Netzwerk? Das Überraschende dabei ist, dass nicht die eigentlichen aktiven Einheiten, nämlich die Neuronen, adaptiv sind, sondern die Verbindungen zwischen ihnen, also die Synapsen. Konkret kann sich dadurch ihre Leitfähigkeit verändern. Wir wissen, dass eine Synapse stärker wird, je mehr elektrischer Strom sie transportieren muss. Stärker bedeutet hier, dass die Synapse eine höhere Leitfähigkeit aufweist. Die genutzten Synapsen erhalten oft ein immer höheres Gewicht. Bei selten genutzten oder gar nicht aktiven Synapsen nimmt die Leitfähigkeit weiter ab. Dies kann sogar zu ihrem Absterben führen.

Alle Neuronen im Gehirn arbeiten asynchron und parallel, allerdings im Vergleich zu einem Computer mit sehr geringer Geschwindigkeit. Die Zeit für einen Nervenimpuls beträgt etwa eine Millisekunde, genau die Zeit, die die Ionen für den Transport über den synaptischen Spa Die Taktfrequenz des Neurons liegt dann unter einem Kilohertz und ist damit um den Faktor 106 niedriger als bei modernen Computern. Dieser Nachteil wird jedoch bei vielen komplexen kognitiven Aufgaben, etwa der Bilderkennung, durch die sehr hohe Taktfrequenz mehr als ausgeglichen Grad der Parallelverarbeitung im Netzwerk von Nervenzellen.

Die Verbindung zur Außenwelt erfolgt über sensorische Neuronen, zum Beispiel auf der Netzhaut in den Augen, oder über Nervenzellen mit sehr langen Axonen, die vom Gehirn bis zu den Muskeln reichen und so Aktionen wie die Bewegung eines Menschen ausführen können Bein.

Allerdings ist noch unklar, wie die diskutierten Prinzipien intelligentes Verhalten ermöglichen. Wie viele Forscher in den Neurowissenschaften werden wir versuchen, anhand von Simulationen eines einfachen mathematischen Modells zu erklären, wie kognitive Aufgaben, beispielsweise die Mustererkennung, möglich werden.

### 9.1.1 Das mathematische Modell

Zuerst ersetzen wir die kontinuierliche Zeitachse durch eine diskrete Zeitskala. Das Neuron i führt die folgende Berechnung in einem Zeitschritt durch. Das „Laden“ der Aktivierung

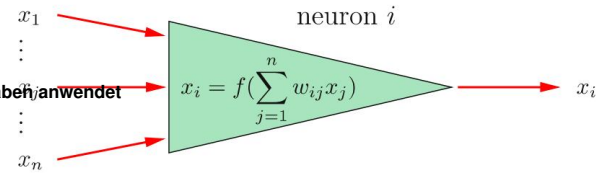



**Abb. 9.2 Die Struktur eines formalen Neurons, das die Aktivierungsfunktion f auf die gewichtete Summe aller Eingaben anwendet**

**Die Berechnung des Potentials erfolgt einfach durch Summation der gewichteten Ausgangswerte x1,...,xn aller eingehenden Verbindungen über die Formel**

$$\sum_{j=1}^{N} w_{ij} x_j .$$

**Diese gewichtete Summe wird von den meisten neuronalen Modellen berechnet. Darauf wird dann eine *Aktivierungsfunktion* f angewendet und das Ergebnis ermittelt**

$$x_i = f\left(\sum_{j=1}^{N} w_{ij} x_j\right)$$

**wird als Ausgabe über die synaptischen Gewichte an die benachbarten Neuronen weitergegeben. In Abb. 9.2 ist ein solches modelliertes Neuron dargestellt. Für die Aktivierungsfunktion gibt es mehrere Möglichkeiten. Am einfachsten ist die Identität: f (x) = x. Das Neuron berechnet also nur die gewichtete Summe der Eingabewerte und gibt diese weiter. Dies führt jedoch häufig zu Konvergenzproblemen mit der neuronalen Dynamik, da die Funktion f (x) = x unbegrenzt ist und die Funktionswerte mit der Zeit über alle Grenzen hinaus wachsen können.**

**Sehr gut eingeschränkt ist dagegen die Schwellenfunktion (Heaviside-Stufenfunktion)**

$$H_\ddot{y}(x) = \begin{cases} 0 & \text{wenn } x<\ddot{y}, \\ 1 & \text{sonst.} \end{cases}$$

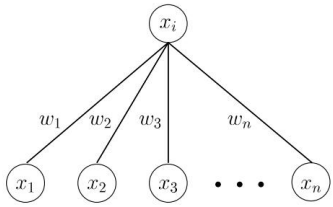
**Das gesamte Neuron berechnet dann seine Ausgabe als**

$$x_i = \begin{cases} 0 & \text{wenn } \sum_{j=1}^{N} w_{ij} x_j < \ddot{y}, \\ 1 & \text{sonst.} \end{cases}$$

**Diese Formel ist identisch mit (8.1) auf Seite 173, also mit einem Perzeptron mit der Schwelle ÿ (Abb. 9.3 auf Seite 225).** Die Eingabeneuronen 1,...,n haben hier nur die Funktion von Variablen, die ihre von außen vorgegebenen Werte **x1,...,xn unverändert weitergeben .**

**Für binäre Neuronen ist die Schrittfunktion durchaus sinnvoll, da die Aktivierung eines Neurons ohnehin nur die Werte Null oder Eins annehmen kann. Im Gegensatz dazu für kontinuierlich**

**Abb. 9.3 Das Neuron mit einer Stufenfunktion funktioniert wie ein Perzeptron mit einer Schwelle**



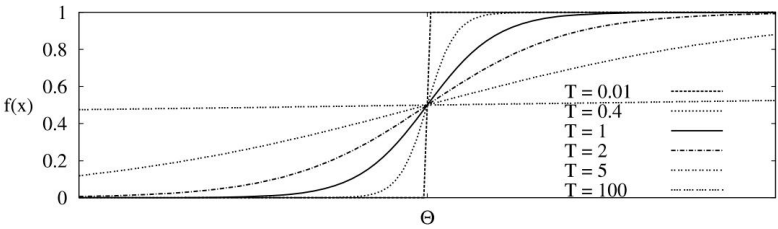




**Abb. 9.4 Die Sigmoidfunktion für verschiedene Werte des Parameters TT ÿ . Das sehen wir im Limit 0 ergibt die Sprungfunktion**

**Bei Neuronen mit Aktivierungen zwischen 0 und 1 erzeugt die Schrittfunktion eine Diskontinuität.** Dies kann jedoch durch eine ***Sigmoidfunktion*** **geglättet werden , z**

$$f(x) = \frac{1}{1 + e^{\frac{x\ddot{y}\ddot{y}}{T}}}$$

**mit der Grafik in Abb. 9.4. In der Nähe des kritischen Bereichs um den Schwellenwert ÿ verhält sich diese Funktion nahezu linear und hat einen asymptotischen Grenzwert. Die Glättung kann durch den Parameter Ŧ variiert werden**

**Die Modellierung des Lernens ist von zentraler Bedeutung für die Theorie neuronaler Netze. Wie bereits erwähnt, besteht eine Möglichkeit des Lernens darin, eine Synapse entsprechend zu stärken, je nachdem wie viele elektrische Impulse sie übertragen muss. Dieses Prinzip wurde 1949 von D. Hebb postuliert und ist als *Hebb-Regel bekannt:***

Wenn eine Verbindung **wij** zwischen Neuron j und Neuron i **besteht und wiederholte Signale von Neuron j an Neuron i gesendet werden, was dazu führt, dass beide Neuronen gleichzeitig aktiv sind, dann wird das Gewicht wij verstärkt. Eine mögliche Formel für die Gewichtsänderung wij lautet**

$$wij = \ddot{y}xixj$$

**mit der Konstanten ÿ (Lernrate), die die Größe der einzelnen Lernschritte bestimmt.**

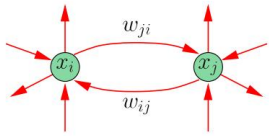



Abb. 9.5 Wiederkehrende Verbindungen zwischen zwei Neuronen in einem Hopfield-Netzwerk

Es gibt viele Modifikationen dieser Regel, die dann zu unterschiedlichen Arten von Netzwerken oder Lernalgorithmen führen. In den folgenden Abschnitten werden wir einige davon kennenlernen.

## 9.2 Hopfield-Netzwerke

Wenn wir uns die Hebb-Regel ansehen, sehen wir, dass bei Neuronen mit Werten zwischen Null und Eins die Gewichte nur mit der Zeit zunehmen können. Nach dieser Regel ist es nicht möglich, dass ein Neuron schwächer wird oder gar stirbt. Dies kann beispielsweise durch eine Zerfallskonstante modelliert werden, die ein ungenutztes Gewicht pro Zeitschritt um einen konstanten Faktor abschwächt, beispielsweise 0,99.

Ganz anders löst dieses Problem das von Hopfield 1982 vorgestellte Modell [Hop82]. Es verwendet binäre Neuronen, jedoch mit den beiden Werten ÿ1 für inaktiv und 1 für aktiv. Mit der Hebb-Regel erhalten wir immer dann einen positiven Beitrag zum Gewicht, wenn zwei Neuronen gleichzeitig aktiv sind. Ist jedoch nur eines der beiden Neuronen aktiv, ist wij negativ.

Auf dieser Idee basieren *Hopfield-Netzwerke,* die ein schönes und anschauliches Beispiel für *autoassoziatives Gedächtnis* darstellen. Muster können im autoassoziativen Speicher gespeichert werden. Um ein gespeichertes Muster aufzurufen, reicht es aus, ein ähnliches Muster bereitzustellen. Der Shop findet dann das ähnlichste gespeicherte Muster. Eine klassische Anwendung hierfür ist die Handschrifterkennung.

In der Lernphase eines Hopfield-Netzwerks werden N binär codierte Muster gespeichert die Vektoren $q^1, \ldots, q^N$ eines $q^j$, j sollen gelernt werden. Jede Komponente $q^j_i \in \{ÿ1, 1\}$ solchen Vektors q repräsentiert ein Pixel eines Musters. Für Vektoren, die aus n Pixeln bestehen, wird ein neuronales Netzwerk mit n Neuronen verwendet, eines für jede Pixelposition. Die Neuronen sind vollständig verbunden mit der Einschränkung, dass die Gewichtsmatrix symmetrisch ist und alle diagonalen Elemente null sind. Das heißt, es gibt keine Verbindung zwischen einem Neuron und sich selbst.

Das vollständig vernetzte Netzwerk beinhaltet komplexe Rückkopplungsschleifen, sogenannte *Rekursionen,* im Netzwerk (Abb. 9.5).

N Muster können gelernt werden, indem einfach alle Gewichte mit der Formel berechnet werden

$$w_{ij} = \frac{1}{N} \sum_{k=1}^{N} q_i^k q_j^k. \qquad (9.1)$$

Diese Formel weist auf eine interessante Beziehung zur Hebb-Regel hin. Jedes Muster, in dem die Pixel i und j den gleichen Wert haben, trägt positiv zum Gewicht wij bei . Jedes andere Muster leistet einen negativen Beitrag. Da jedes Pixel

entspricht einem Neuron, hier werden die Gewichte zwischen Neuronen, die gleichzeitig den gleichen Wert haben, verstärkt. Bitte beachten Sie diesen kleinen Unterschied zur Hebb-Regel.

Sind alle Muster gespeichert, kann das Netzwerk zur Mustererkennung genutzt werden. Wir geben dem Netzwerk ein neues Muster x und aktualisieren die Aktivierungen aller Neuronen in einem asynchronen Prozess gemäß der Regel

$$x_i = \begin{cases} +1 & \text{wenn } \sum_{\substack{j=1 \\ j \neq i}}^{N} w_{ij} x_j < 0, \\ -1 & \text{anders} \end{cases} \tag{9.2}$$

bis das Netzwerk stabil wird, also bis sich keine Aktivierungen mehr ändern. Als Programmschema lautet dies wie folgt:

**HOPFIELDASSOCIATOR(q)**
Alle Neuronen initialisieren: $x = q$
  Wiederholen i = Random(1, n)
    Aktualisieren Sie Neuron i gemäß (9.2)
Bis x konvergiert
Rückgabe (x)

## 9.2.1 Anwendung auf ein Mustererkennungsbeispiel

Wir wenden den beschriebenen Algorithmus auf ein einfaches Mustererkennungsbeispiel an. Es sollte Ziffern in einem 10×10 Pixel großen Feld erkennen. Das Hopfield-Netzwerk verfügt somit über 100 Neuronen mit insgesamt

$$\frac{100 \cdot 99}{2} = 4950$$

Gewichte. Zunächst werden die Muster der Ziffern 1, 2, 3, 4 in Abb. 9.6 auf Seite 228 oben trainiert. Das heißt, die Gewichte werden nach (9.1) auf Seite 226 berechnet . Dann fügen wir das Muster mit zusätzlichem Rauschen ein und lassen die Hopfield-Dynamik bis zur Konvergenz laufen. In den Zeilen 2 bis 4 der Abbildung sind fünf Momentaufnahmen der Netzwerkentwicklung während der Erkennung dargestellt. Bei 10 % Rauschen werden alle vier gelernten Muster sehr zuverlässig erkannt. Oberhalb von etwa 20 % Rauschen konvergiert der Algorithmus häufig mit anderen gelernten Mustern oder sogar mit Mustern, die nicht gelernt wurden. Mehrere solcher Muster sind in Abb. 9.6 auf Seite 228 unten dargestellt .

Nun speichern wir die Ziffern 0 bis 9 (Abb. 9.7 auf Seite 229 oben) im selben Netzwerk und testen das Netzwerk erneut mit Mustern, die einen zufälligen Anteil von etwa 10 % invertierten Pixeln haben. In der Abbildung sehen wir deutlich, dass die Hopfield-Iteration selbst bei nur 10 % Rauschen oft nicht zum ähnlichsten erlernten Zustand konvergiert. Offensichtlich

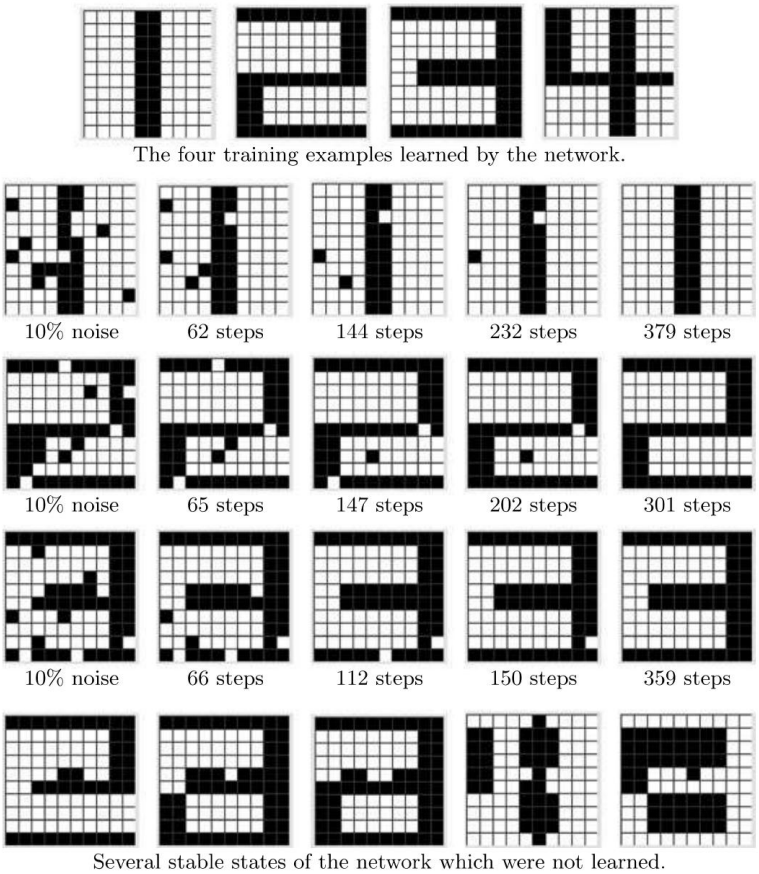



**Abb. 9.6 Dynamik eines Hopfield-Netzwerks. In *den Zeilen* 2, *3* und *4* können wir gut erkennen, wie das Netzwerk konvergiert und das gelernte Muster nach etwa 300 bis 400 Iterationen erkannt wird. In der *letzten Zeile* werden mehrere stabile Zustände angezeigt, die das Netzwerk erreicht, wenn das Eingabemuster zu stark von allen gelernten Mustern abweicht**

**Das Netzwerk kann vier Muster sicher speichern und erkennen, für zehn Muster ist jedoch die Speicherkapazität überschritten. Um dies besser zu verstehen, werfen wir einen kurzen Blick auf die Theorie dieses Netzwerks.**

## 9.2.2 Analyse

**1982 zeigte John Hopfield in [Hop82], dass dieses Modell formal einem physikalischen Modell des Magnetismus entspricht. Kleine Elementarmagnete, sogenannte Spins, beeinflussen sich gegenseitig über ihre Magnetfelder (siehe Abb. 9.8 auf Seite 229). Wenn wir**

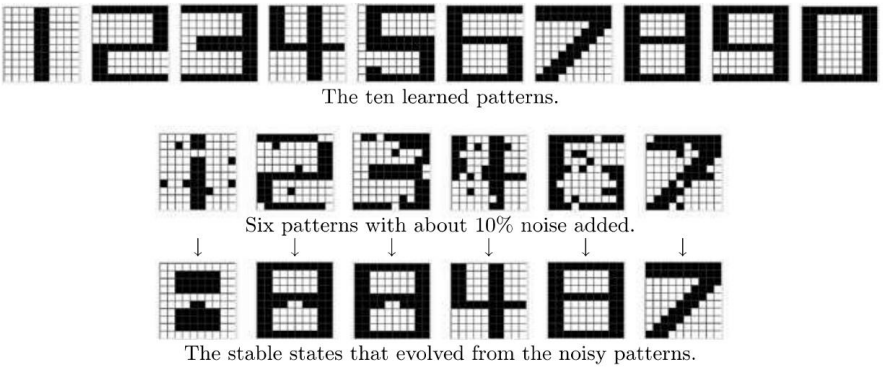



**Abb. 9.7 Für zehn erlernte Zustände zeigt das Netzwerk chaotisches Verhalten. Selbst bei geringem Rauschen konvergiert das Netzwerk mit falschen Mustern oder Artefakten**

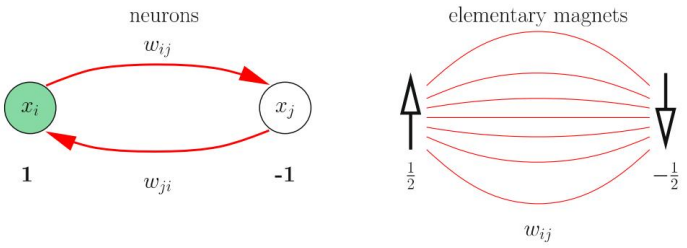



**Abb. 9.8 Vergleich zwischen der neuronalen und physikalischen Interpretation des Hopfield-Modells**

**Beobachten Sie zwei solcher Spins i und j , sie interagieren über eine Konstante wij und die Gesamtenergie des Systems beträgt dann**

$$E = \ddot{y}\ 2^{-1} \sum_{i,j} w_{ij}\, x_i x_j .$$

**Übrigens gilt wii = 0 auch in der Physik, weil Teilchen keine Selbstwechselwirkung haben. Da physikalische Wechselwirkungen symmetrisch sind, gilt wij = wj i.**

**Ein physikalisches System im Gleichgewicht nimmt einen (stabilen) Zustand minimaler Energie ein und minimiert somit E(x, y). Wird ein solches System in einen beliebigen Zustand gebracht, dann bewegt es sich in einen Zustand minimaler Energie. Die in (9.2) auf Seite 227 definierte Hopfield-Dynamik entspricht genau diesem Prinzip, da sie den Zustand in jeder Iteration so aktualisiert, dass von den beiden Zuständen ÿ1 und 1 derjenige mit der kleineren Gesamtenergie eingenommen wird. Der Beitrag des Neurons i zur Gesamtenergie beträgt**

$$-\frac{1}{2}\, x_i \sum_{j=i}^{N} w_{ij}\, x_j .$$

Wenn jetzt

$$\sum_{j=i}^{N} w_{ij} x_j < 0,$$

dann führt $x_i = \ddot{y}1$ zu einem negativen Beitrag zur Gesamtenergie und $x_i = 1$ zu einem positiven Beitrag. Für $x_i = \ddot{y}1$ nimmt das Netzwerk einen Zustand niedrigerer Energie ein als für $x_i = 1$. Analog können wir das im Fall von behaupten

$$\sum_{j=i}^{N} w_{ij} x_j \ddot{y} 0,$$

es muss wahr sein, dass $x_i = 1$.

Wenn jede einzelne Iteration der neuronalen Dynamik zu einer Verringerung der Energiefunktion führt, dann nimmt die Gesamtenergie des Systems monoton mit der Zeit ab. Da es nur endlich viele Zustände gibt, bewegt sich das Netzwerk mit der Zeit in einen Zustand minimaler Energie. Nun stellt sich uns die spannende Frage: Was bedeuten diese Minima der Energiefunktion?

Wie wir im Mustererkennungsexperiment gesehen haben, konvergiert das System bei wenigen gelernten Mustern zu einem der gelernten Muster. Die erlernten Muster repräsentieren Minima der Energiefunktion im Zustandsraum. Werden jedoch zu viele Muster gelernt, konvergiert das System zu Minima, die nicht den gelernten Mustern entsprechen. Hier haben wir einen Übergang von einer geordneten Dynamik in eine chaotische
eins.

Hopfield und andere Physiker haben genau diesen Prozess untersucht und gezeigt, dass es bei einer kritischen Anzahl erlernter Muster tatsächlich zu einem Phasenübergang kommt. Übersteigt die Anzahl der gelernten Muster diesen Wert, dann geht das System von der geordneten Phase in die chaotische Phase über.

In der magnetischen Physik gibt es einen solchen Übergang vom ferromagnetischen Modus, bei dem alle Elementarmagnete versuchen, sich parallel auszurichten, zu einem sogenannten Spinglas, bei dem die Spins chaotisch interagieren. Ein besser sichtbares Beispiel für einen solchen physikalischen Phasenübergang ist das Schmelzen eines Eiskristalls. Der Kristall befindet sich in einem hohen Ordnungszustand, da die H2O- Moleküle streng geordnet sind. Im flüssigen Wasser hingegen ist die Struktur der Moleküle aufgelöst und ihre Position

In einem neuronalen Netz kommt es dann zu einem Phasenübergang vom geordneten Lernen und Erkennen von Mustern zum chaotischen Lernen bei zu vielen Mustern, die nicht mehr sicher erkannt werden können. Hier sehen wir durchaus Parallelen zu Wirkungen, die wir gelegentlich selbst erlebt haben.

Wir können diesen Phasenübergang [RMS92] verstehen, wenn wir alle Neuronen in a Musterzustand, zum Beispiel $^{1}$, bringen und die gelernten Gewichte aus (9.1) auf Seite 226 $q$ in den Begriff $^{N}$ einfügen, was für die Aktualisierung von Neuron i relevant ist. Daraus ergibt sich $\sum_{j=1,j=i}^{N} w_{ij} q_j$,
In

$$w_{ij} = \frac{1}{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \sum_{k=1}^{N} q_i^k q_j^k = \frac{1}{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \sum_{k=2}^{N} q_i^k q_j^k + \frac{1}{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} q_i^1 q_j^1$$

$$= q_i^1 + \frac{1}{N} \sum_{j=1 \atop j \neq i}^{N} \sum_{k=2}^{N} q_i^k q_j^k q_j^1 .$$

Hier sehen wir die i-te Komponente des Eingabemusters plus eine Summe mit (n ÿ 1)(N ÿ 1) Termen. Wenn diese Summanden alle statistisch unabhängig sind, können wir die Summe durch eine normalverteilte Zufallsvariable mit Standardabweichung beschreiben

$$\frac{1}{N} \sqrt{(n-1)(N-1)} \approx \sqrt{\frac{N-1}{N}} .$$

Statistische Unabhängigkeit kann beispielsweise mit unkorrelierten Zufallsmustern erreicht werden. Die Summe erzeugt dann Rauschen, das nicht störend ist, solange N ÿ n, was bedeutet, dass die Anzahl der gelernten Muster viel kleiner bleibt als die Anzahl der Neuronen. Ist jedoch N ÿ n, dann wird der Einfluss des Rauschens so groß wie das Muster und das Netzwerk reagiert chaotisch. Eine genauere Berechnung des Phasenübergangs ergibt N = 0,146 n als kritischen Punkt. Auf unser Beispiel übertragen bedeutet das, dass für 100 Neuronen bis zu 14 Muster gespeichert werden können. Da die Muster im Beispiel jedoch stark korrelieren, liegt der kritische Wert viel niedriger, offenbar zwischen 0,04 und 0,1. Selbst ein Wert von 0,146 ist viel niedriger als die Speicherkapazität einer herkömmlichen Speicherliste (Übung 9.3 auf Seite 255).

Hopfield-Netzwerke in der dargestellten Form funktionieren nur dann gut, wenn Muster mit etwa 50 % 1-Bits gelernt werden. Sind die Bits sehr asymmetrisch verteilt, müssen die Neuronen mit einer Schwelle ausgestattet werden [Roj96]. In der Physik ist dies analog zum Anlegen eines äußeren Magnetfeldes, das ebenfalls eine Asymmetrie der Spin 1/2- und Spin ÿ1/2-Zustände bewirkt.

### 9.2.3 Zusammenfassung und Ausblick

Durch seine biologische Plausibilität, das gut verstandene mathematische Modell und vor allem durch die beeindruckenden Simulationen in der Mustererkennung trug das Hopfield-Modell zu einer Welle der Begeisterung für neuronale Netze und zum Aufstieg bei der Neuroinformatik als einem wichtigen Zweig der KI.[2] Anschließend wurden viele weitere Netzwerkmodelle entwickelt. Einerseits wurden Netzwerke ohne Rückkopplungen untersucht, da deren Dynamik deutlich einfacher zu verstehen ist als bei aktuellen Hopfield-Netzwerken. Andererseits wurde versucht, die Speicherkapazität der Netzwerke zu verbessern, worauf wir im nächsten Abschnitt eingehen werden.

---

2Auch der Autor war von dieser Welle erfasst, die ihn 1987 von der Physik in die KI führte.

Ein besonderes Problem vieler neuronaler Modelle zeigte sich bereits beim Hopfield-Modell. Selbst wenn eine Konvergenzgarantie besteht, ist es nicht sicher, ob das Netzwerk in einen erlernten Zustand konvergiert oder bei einem lokalen Minimum hängen bleibt. Als Versuch, dieses Problem zu lösen, wurde die Boltz-Mann-Maschine mit kontinuierlichen Aktivierungswerten und einer probabilistischen Aktualisierungsregel für ihre Netzwerkdynamik entwickelt. Mithilfe eines „Temperatur"-Parameters können wir die Menge zufälliger Zustandsänderungen variieren und so versuchen, lokale Minima zu umgehen, mit dem Ziel, ein stabiles globales Minimum zu finden. Dieser Algorithmus wird „Simulated Annealing" genannt. Glühen ist ein Prozess der Wärmebehandlung von Metallen mit dem Ziel, das Metall fester und „stabiler" zu machen.

Das Hopfield-Modell sucht nach einem Minimum der Energiefunktion im Raum der Aktivierungswerte. Dadurch wird das in den Gewichten gespeicherte Muster gefunden, das somit in der Energiefunktion abgebildet wird. Die Hopfield-Dynamik kann auch auf andere Energiefunktionen angewendet werden, sofern die Gewichtsmatrix symmetrisch ist und die Diagonalelemente Null sind. Dies wurde von Hopfield und Tank am Problem des Handlungsreisenden erfolgreich demonstriert [HT85, Zel94]. Die Aufgabe besteht hier darin, anhand von n Städten und ihrer Entfernungsmatrix den kürzesten Hin- und Rückweg zu finden, der jede Stadt genau einmal besucht.

## 9.3 Neuronales Assoziatives Gedächtnis

Ein klassischer Listenspeicher kann im einfachsten Fall eine Textdatei sein, in der Ziffernfolgen zeilenweise gespeichert werden. Wenn die Datei zeilenweise sortiert ist, kann die Suche nach einem Element auch bei sehr großen Dateien sehr schnell in logarithmischer Zeit erfolgen.

Der Listenspeicher kann jedoch auch zur Erstellung von Zuordnungen verwendet werden. Ein Telefonbuch ist beispielsweise eine Abbildung der Menge aller eingegebenen Namen auf die Menge aller Telefonnummern. Diese Zuordnung wird als einfache Tabelle implementiert, die normalerweise in einer Datenbank gespeichert wird.

Eine ähnliche Aufgabe stellt die Zugangskontrolle zu einem Gebäude mittels Gesichtserkennung dar. Hier könnten wir auch eine Datenbank verwenden, in der von jeder Person ein Foto zusammen mit dem Namen der Person und möglicherweise weiteren Daten gespeichert ist. Die Kamera am Eingang macht dann ein Foto der Person und durchsucht die Datenbank nach einem identischen Foto. Wird das Foto gefunden, ist die Person identifiziert und erhält Zutritt zum Gebäude. Allerdings würde ein Gebäude mit einem solchen Kontrollsystem nicht viele Besucher bekommen, da die Wahrscheinlichkeit, dass das aktuelle Foto genau mit dem gespeicherten Foto übereinstimmt, sehr gering ist.

In diesem Fall reicht es nicht aus, das Foto nur in einer Tabelle zu speichern. Was wir vielmehr wollen, ist ein *assoziatives Gedächtnis,* das in der Lage ist, nicht nur dem Foto den richtigen Namen zuzuordnen, sondern auch einem beliebigen aus einer potenziell unendlichen Menge „ähnlicher" Fotos. Aus einem endlichen Satz an Trainingsdaten, nämlich den mit den Namen beschrifteten gespeicherten Fotos, soll eine Funktion zur Ähnlichkeitsfindung generiert werden. Ein einfacher Ansatz hierfür ist die in Abschn. 2.1 vorgestellte Methode des nächsten Nachbarn. 8.3. Beim Lernen werden alle Fotos einfach gespeichert.

Um diese Funktion nutzen zu können, muss in der Datenbank das Foto gefunden werden, das dem aktuellen am ähnlichsten ist. Bei einer Datenbank mit vielen hochauflösenden Fotos kann dieser Vorgang je nach verwendeter Entfernungsmetrik sehr lange Rechenzeiten und erfordern

ist daher in dieser einfachen Form nicht umsetzbar. Daher bevorzugen wir anstelle eines solchen Lazy-Algorithmus einen, der die Daten in eine Funktion überführt, die dann bei ihrer Anwendung eine sehr schnelle Assoziation herstellt.

Ein weiteres Problem stellt die Suche nach einer geeigneten Distanzmetrik dar. Wir möchten, dass eine Person auch dann erkannt wird, wenn das Gesicht der Person an einer anderen Stelle auf dem Foto erscheint (Übersetzung), oder wenn es kleiner, größer oder sogar gedreht ist. Auch der Betrachtungswinkel und die Beleuchtung können variieren.

Hier zeigen neuronale Netze ihre Stärken. Ohne dass sich der Entwickler Gedanken über eine geeignete Ähnlichkeitsmetrik machen muss, liefern sie dennoch gute Ergebnisse.
Wir stellen zwei der einfachsten assoziativen Gedächtnismodelle vor und beginnen mit einem Modell von Teuvo Kohonen, einem der Pioniere auf diesem Gebiet.

Das im vorherigen Kapitel vorgestellte Hopfield-Modell wäre aus zwei Gründen zu schwierig anzuwenden. Erstens handelt es sich nur um ein *autoassoziatives Gedächtnis,* also um eine annähernd identische Abbildung, die ähnliche Objekte auf das erlernte Original abbildet. Zweitens ist die komplexe wiederkehrende Dynamik in der Praxis oft schwer zu bewältigen. Daher betrachten wir nun einfache zweischichtige Feedforward-Netzwerke.

## 9.3.1 Korrelationsmatrixspeicher

In [Koh72] führte Kohonen ein assoziatives Gedächtnismodell ein, das auf elementarer linearer Algebra basiert. Dadurch werden Abfragevektoren $x \in \mathbb{R}^N$ auf Ergebnisvektoren $y \in \mathbb{R}^m$ abgebildet. Wir suchen nach einer Matrix W, die einen Satz von Trainingsdaten korrekt abbildet

$$T = \{(q^1, t^1), \ldots, (q^N, t^N)\}$$

mit N Anfrage-Antwort-Paaren alle Anfragevektoren zu ihren Antworten.3 Das heißt, für p = 1,...,N muss es so sein

$$t^p = W \cdot q^P, \tag{9.3}$$

oder

$$t_i^p = \sum_{j=1}^{N} w_{ij} q_j^P. \tag{9.4}$$

Um die Matrixelemente $w_{ij}$ zu berechnen , gilt die Regel

$$w_{ij} = \sum_{p=1}^{N} t_i^p q_j^p \tag{9.5}$$

3Um eine klare Unterscheidung zwischen Trainingsdaten und anderen Werten eines Neurons zu ermöglichen, bezeichnen wir in der folgenden Diskussion den Abfragevektor als q und die gewünschte Antwort als t (Ziel).

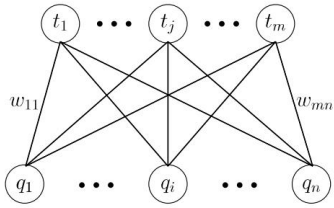



**Abb. 9.9 Darstellung des Kohonen-Assoziativgedächtnisses als zweischichtiges neuronales Netzwerk**

**wird eingesetzt. Diese beiden linearen Gleichungen können einfach als neuronales Netzwerk verstanden werden, wenn wir, wie in Abb. 9.9, ein zweischichtiges Netzwerk mit q als Eingangsschicht und t als Ausgangsschicht definieren. Die Neuronen der Ausgabeschicht haben eine lineare Aktivierungsfunktion und als Lernregel wird (9.5)** verwendet, was genau der Hebb-Regel entsp

**Bevor wir zeigen, dass das Netzwerk die Trainingsdaten erkennt, benötigen wir folgende Definition:**

**Definition 9.1 Zwei Vektoren x und y heißen *orthonormal* wenn**

$$\mathbf{T}\ \mathbf{x} \cdot \mathbf{y} = \begin{cases} \mathbf{1, wenn\ x = y,} \\ \mathbf{sonst\ 0.} \end{cases}$$

**Daher**

**Satz 9.1** ***Wenn alle*** **N** ***Abfragevektoren*** **q p** ***in den Trainingsdaten orthonormal sind, dann wird jeder Vektor*** **q p** ***durch Multiplikation mit der Matrix*** **wij** ***aus*** **(9.5)** ***auf Seite*** 233 auf den **Zielvektor** t **p** ***abgebildet .***

***Beweis*** **Wir setzen (9.5) auf Seite 233 in (9.4) auf Seite 233** ein **und erhalten**

$$(W \cdot q^P)_i = \sum_{jj=1}^{N} w_{ij} q_j^P = \sum_{j=1}^{N} \sum_{r=1}^{N} r_{qf}^R{}_{ji} q_j^p$$

$$= \sum_{j=1}^{N} q_j^P q_j^{pp} t_{+i} \sum_{r=p}^{N} \sum_{j=1}^{N} q_j^R q_j^p t_i^R$$

$$= t_i^p \sum_{j=1}^{N} Q_{jq}^P{}_j^P p + \sum_{r=p}^{N} t_i^R \sum_{j=1}^{N} Q_{jq}^R{}_j^P$$

$$= t_i^p \underbrace{(Q^P)^T}_{=1}{}_{pq} + t_i^R \sum_{r=p}^{N} \underbrace{(Q^R)^T}_{=0}{}_{pp} q = t_i$$

Wenn also die Abfragevektoren orthonormal sind, werden alle Muster korrekt den jeweiligen Zielen zugeordnet. Allerdings ist die Orthonormalität eine zu starke Einschränkung. In Abschn. In Abschnitt 9.4 werden wir einen Ansatz vorstellen, der diese Einschränkung überwindet.

Da lineare Abbildungen kontinuierlich und injektiv sind, wissen wir, dass die Abbildung von Abfragevektoren auf Zielvektoren die Ähnlichkeit bewahrt. Aufgrund der Kontinuität werden ähnliche Abfragen somit auf ähnliche Ziele abgebildet. Gleichzeitig wissen wir jedoch, dass unterschiedliche Abfragen unterschiedlichen Zielen zugeordnet werden. Wenn das Netzwerk darauf trainiert wurde, Gesichtern Namen zuzuordnen, und wenn einem Gesicht der Name Henry zugewiesen wird, dann sind wir sicher, dass bei der Eingabe eines ähnlichen Gesichts eine Ausgabe ähnlich „Henry“ erzeugt wird, aber „Henry“ selbst wird garantiert nicht berechnet. Wenn die Ausgabe als String interpretiert werden kann, könnte es sich beispielsweise um „Genry“ oder „Gfnry“ handeln. Um zu dem ähnlichsten erlernten Fall zu gelangen, verwendet Kohonen eine binäre Kodierung für das Ausgabeneuron. Das berechnete Ergebnis einer Abfrage wird gerundet, wenn sein Wert nicht Null oder Eins ist. Selbst dann haben wir keine Garantie, dass wir den Zielvektor erreichen. Alternativ könnten wir eine nachträgliche Abbildung der berechneten Antwort auf den gelernten Zielvektor mit dem kleinsten Abstand h

## 9.3.2 Die Pseudoinverse

Es gibt einen anderen Ansatz, den wir zur Berechnung der Gewichtsmatrix W wählen können: indem wir uns alle Abfragevektoren als Spalten einer n × N-Matrix Q = (q ) und $_{1},\ldots,q_{N}$ vorstellen Analog können die Zielvektoren als Spalten einer m × N-Matrix T = (t) dargestellt ). werden. Somit kann (9.3) auf Seite 233 in die Form gebracht werden

$$T = W \cdot Q. \tag{9.6}$$

Jetzt versuchen wir, diese Gleichung nach W zu lösen. Formal invertieren wir und erhalten

$$W = T \cdot Q^{-1}. \tag{9.7}$$

Voraussetzung für diese Umrechnung ist die Invertierbarkeit von Q. Dazu muss die Matrix quadratisch sein und aus linear unabhängigen Spaltenvektoren bestehen. Das heißt, es muss gelten, dass n = N ist und die n Abfragevektoren alle linear unabhängig sein müssen. Diese Bedingung ist zwar unbequem, aber immer noch nicht ganz so streng wie die oben geforderte Orthonormalität. Das Lernen nach der Hebb-Regel hat jedoch den Vorteil, dass wir sie auch dann einfach anwenden können, wenn die Abfragevektoren nicht orthonormal sind, in der Hoffnung, dass der Assoziator trotzdem einigermaßen funktioniert. Hier ist dies jedoch nicht so einfach, denn wie können wir eine nicht invertierbare Matrix umkehren?

Eine Matrix Q ist genau dann invertierbar, wenn es eine Matrix $Q^{-1}$ mit der Eigenschaft gibt

$$Q \cdot Q^{-1} = I \quad , \tag{9.8}$$

wobei ich die Identitätsmatrix ist. Wenn Q nicht invertierbar ist (z. B. weil Q nicht quadratisch ist), dann gibt es keine Matrix $Q^{-1}$ mit dieser Eigenschaft. Es gibt jedoch sicherlich eine Matrix, die dieser Eigenschaft nahe kommt. In diesem Sinne definieren wir

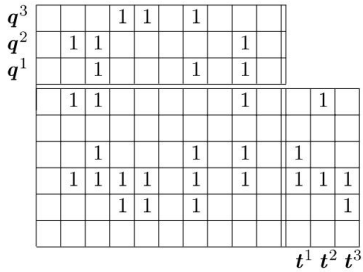



**Abb. 9.10** Die Matrix W nach dem Speichern von drei Paaren (q 1 ,t 1 ), (q 2 ,t 2 ), (q 3 ,t 3 ). Leere Felder entsprechen dem Wert 0

**Definition 9.2** Sei Q eine reelle n × m-Matrix. Eine m × n-Matrix Q+ heißt *pseudoinvers* zu Q, wenn sie minimiert ist

$$Q \cdot Q+ \ddot{y} I.$$

Hier ist M die euklidische Norm.

Jetzt verwenden

$$W = T \cdot Q+ \tag{9.9}$$

Wir können eine Gewichtsmatrix berechnen, die Übersprechen (Assoziationsfehler) minimiert. T ÿ W · Q. Es gibt verschiedene Möglichkeiten, die Pseudoinverse zu berechnen, zum Beispiel die Methode der kleinsten Quadrate, die wir in Abschn. 2.1 vorstellen werden. 9.4.1.

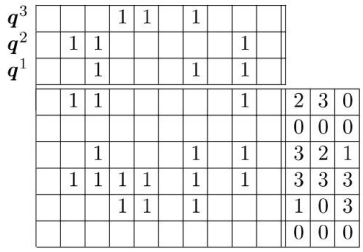
## 9.3.3 Die binäre Hebb-Regel

Im Zusammenhang mit dem assoziativen Gedächtnis wurde die sogenannte binäre Hebb-Regel vorgeschlagen. Es erfordert, dass das Muster binär codiert ist. Dies bedeutet, dass wir für alle Muster q p ÿ {0, 1} N und t p ÿ {0, 1} m. Darüber hinaus ist die Summe aus (9.5) auf Seite 233 durch ein einfaches logisches ODER ersetzen und die binäre Hebb-Regel erhalten

$$w_{ij} = \bigvee_{p=1}^{N} t_i^p q_j^p . \tag{9.10}$$

Die Gewichtsmatrix ist somit ebenfalls binär und ein Matrixelement wij ist gleich eins, wenn und nur wenn mindestens einer der Einträge q 1 j ,...,qN j es nicht Null ist. Alle anderen Matrix Elemente sind Null. Wir sind versucht zu glauben, dass hier beim Lernen viele Informationen verloren gehen, denn wenn ein Matrixelement einmal den Wert 1 annimmt, kann es durch weitere Muster nicht mehr verändert werden. Abbildung 9.10 zeigt, wie die Matrix für ein Beispiel mit n = 10, m = 6 nach dem Lernen von drei Paaren mit Einsen gefüllt wird.

Abb. 9.11 Berechnung des Produkts W q1 , W q2 , W q3




Um die gespeicherten Muster abzurufen, multiplizieren wir einfach einen Abfragevektor q mit der Matrix und schauen Sie sich das Ergebnis W q an. Wir testen dies am Beispiel und erhalten Abb. 9.11.

Wir sehen, dass im Zielvektor auf der rechten Seite an der Stelle, an der der gelernte Zielvektor eine Eins hatte, der Wert 3 steht. Die korrekten Ergebnisse würden durch Festlegen eines Schwellenwerts von 3 erzielt. Im allgemeinen Fall wählen wir die Anzahl der Einsen im Abfragevektor als Schwellenwert. Jedes Ausgabeneuron funktioniert somit wie ein Perzeptron, allerdings mit variabler Schwelle.

Solange die Gewichtsmatrix dünn besetzt ist, funktioniert dieser Algorithmus gut. Wenn jedoch viele verschiedene Muster gespeichert werden, wird die Matrix immer dichter. Im Extremfall enthält es nur Einsen. Dann würden nach Festlegung des Schwellenwerts alle Antworten nur noch aus Einsen bestehen und keine Informationen mehr enthalt

Dieser Fall tritt selten ein, solange die Anzahl der in der Matrix gespeicherten Bits nicht zu groß wird. Die Matrix hat eine Größe von mn Elementen. Ein zu speicherndes Paar hat m + n Bits. Man kann [Pal80] zeigen, dass die Anzahl der merkbaren Muster Nmax durch die folgende Bedingung bestimmt wird:

$$\ddot{y} = \frac{\text{Anzahl der speicherbaren Bits}}{\text{Anzahl der binären Matrixelemente}} = \frac{(m+n)N_{max}}{mn}\ddot{y} \ln 2 \ddot{y} 0{,}69. \tag{9.11}$$

Für einen Listenspeicher gilt ÿ = 1. Assoziativer Speicher mit der binären Hebb-Regel hat eine maximale Speichereffizienz von ÿ = 0,69 im Vergleich zu ÿ = 0,72 für koho nen assoziativen Speicher und ÿ = 0,292 für Hopfield-Netzwerke [Pal80, Pal91]. Die Speicherkapazität der binären Hebb-Regel ist damit im Vergleich zum Kohonen-Modell mit kontinuierlichen Neuronen überraschend hoch.

Es ist offensichtlich, dass ein solcher Speicher weniger schnell „voll“ wird, wenn die Abfrage- und Zielvektoren nur spärlich mit Einsen gefüllt sind. Nicht nur deshalb ist die Kodierung von Ein- und Ausgabe bei assoziativen Speichern – wie auch bei anderen neuronalen Netzen – für eine gute Leistung sehr wichtig. Wir werden dies nun an einer Anwendung dieses Speichers mit geeignet gewählten Kodierungen von Ein- und Ausgabe demonstrieren.

### 9.3.4 Ein Rechtschreibkorrekturprogramm

Als Anwendung des beschriebenen assoziativen Gedächtnisses mit der binären Hebb-Regel wählen wir ein Programm, das fehlerhafte Eingaben korrigiert und sie auf gespeicherte Wörter aus einem Wörterbuch abbildet. Hier wäre eindeutig ein autoassoziatives Gedächtnis erforderlich. Da wir die Abfrage- und Zielvektoren jedoch unterschiedlich kodieren, ist dies nicht der Fall. Für die Abfragevektoren q wählen wir eine Paarkodierung. Für ein Alphabet mit 26 Zeichen gibt es 26 · 26 = 676 geordnete Buchstabenpaare. Bei 676 Bit hat der Abfragevektor für jedes der möglichen Paare ein Bit

aa, ab,..., az, ba,..., bz,..., za,..., zz.

Kommt in dem Wort ein Buchstabenpaar vor, so wird an der entsprechenden Stelle eine Eins eingetragen. Für das Wort „Henry“ beispielsweise sind die Plätze für „ha“, „an“ und „ns“ mit Einsen gefüllt. Für den Zielvektor t sind für jede Position im Wort bis zu einer maximalen Länge (z. B. zehn Zeichen) 26 Bit reserviert. Für den i-ten Buchstaben im Alphabet an Position j im Wort wird dann die Bitzahl (j ÿ 1)· 26 + i gesetzt. Für das Wort „Henry“ sind die Bits 8, 27, 66 und 97 gesetzt. Bei maximal 10 Buchstaben pro Wort hat der Zielvektor somit eine Länge von 260 Bit.

Die Gewichtsmatrix W hat somit eine Größe von 676 · 260 Bits = 199420 Bits, was durch (9.11) auf Seite 237 kann höchstens speichern

$$N_{max} \ddot{y} 0{,}69 = 0{,}69 \ddot{y} 130\,676 + 260 \frac{mn}{m+n} \frac{676 \cdot 260}{}$$

Wörter. Bei 72 Vornamen sparen wir etwa die Hälfte ein und testen das System. Die gespeicherten Namen und die Ausgabe des Programms für mehrere Beispieleingaben sind in Abb. 9.12 auf Seite 239 dargestellt . Der Schwellenwert wird immer auf die Anzahl der Bits in der codierten Abfrage initialisiert. Hier ist dies die Anzahl der Buchstabenpaare, also die Wortlänge minus eins. Dann wird es schrittweise auf zwei reduziert. Wir könnten die Auswahl des Schwellenwerts weiter automatisieren, indem wir für jeden versuchten Schwellenwert einen Vergleich mit dem Wörterbuch durchführen und das gefundene Wort ausgeben, wenn der Vergleich erfolgreich ist.

Interessant ist die Reaktion auf die mehrdeutigen Eingaben „andr“ und „johanne“. In beiden Fällen erstellt das Netzwerk eine Mischung aus zwei gespeicherten Wörtern, die passen. Wir sehen hier eine wichtige Stärke neuronaler Netze. Sie sind in der Lage, Assoziationen zu ähnlichen Objekten ohne explizite Ähnlichkeitsmetrik herzustellen. Allerdings gibt es, ähnlich wie bei der heuristischen Suche und der menschlichen Entscheidungsfindung, keine Garantie für eine „richtige“ Lösung.

Da für alle bisher vorgestellten neuronalen Modelle die Trainingsdaten in Form von Input-Output-Paaren vorliegen müssen, handelt es sich um überwachtes Lernen, was auch für die in den folgenden Abschnitten vorgestellten Netzwerke der Fall ist.

Gespeicherte Wörter:

agathe, agnes, alexander, andreas, andree, anna, annemarie, astrid, august, bernhard, björn, cathrin, christian, christoph, corinna, corrado, dieter, elisabeth, elvira, erdmut, ernst, evelyn, fabrizio, frank, franz, geoffrey, georg, gerhard, hannelore, harry, herbert, ingilt, irmgard, jan, johannes, johnny, juergen, karin, klaus, ludwig, luise, manfred, maria, mark, markus, marleen, martin, matthias, norbert, otto, Patricia, Peter, Phillip, Quit, Reinhold, Renate, Robert, Robin, Sabine, Sebastian, Stefan, Stephan, Sylvie, Ulrich, Ulrike, Ute, Uwe, Werner, Wolfgang, Xavier

Verbände des Programms:

Eingabemuster: Harry

Schwellenwert: 4, Antwort: Harry
Schwellenwert: 3, Antwort: Harry
Schwellenwert: 2, Antwort: horryrrde

---

Eingabemuster: ute

Schwelle: 2, Antwort: ute

---

Eingabemuster: Gerhar

Schwelle: 5, Antwort: Gerhard Schwelle: 4, Antwort: Gerrarn Schwelle: 3, Antwort: Jerrhrrd Schwelle: 2, Antwort: Jurtyrrde

---

Eingabemuster: egrhard

Schwelle: 6, Antwort: Schwelle: 5, Antwort: Schwelle: 4, Antwort: Gerhard Schwelle: 3, Antwort: gernhrrd Schwelle: 2, Antwort: irrryrrde

---

Eingabemuster: andr

Schwellenwert: 3, Antwort: Andrees
Schwellenwert: 2, Antwort: Anexenser

Eingabemuster: andrees

Schwellenwert: 6, Antwort: a
Schwellenwert: 5, Antwort: Andree
Schwellenwert: 4, Antwort: Andrees
Schwellenwert: 3, Antwort: Mnnrens
Schwellenwert: 2, Antwort: Morxsnssr

---

Eingabemuster: johanne

Schwellenwert: 6, Antwort: johnnnes
Schwellenwert: 5, Antwort: johnnnes
Schwellenwert: 4, Antwort: jornnnrse
Schwellenwert: 3, Antwort: sorrnyrse
Schwellenwert: 2, Antwort: wtrrsyrse

---

Eingabemuster: johnnnes

Schwelle: 6, Antwort: joh Schwelle: 5, Antwort: johnnnes Schwelle: 4, Antwort: johnnyes Schwelle: 3, Antwort: jonnnyes Schwelle: 2, Antwort: jornsyrse

---

Eingabemuster: johnnyes

Schwellenwert: 7, Antwort:
Schwellenwert: 6, Antwort: Joh
Schwellenwert: 5, Antwort: Johnny
Schwellenwert: 4, Antwort: Johnnyes
Schwellenwert: 3, Antwort: Johnnyes
Schwellenwert: 2, Antwort: Jonnnyes

**Abb. 9.12 Anwendung des Rechtschreibkorrekturprogramms auf verschiedene gelernte oder fehlerhafte Eingaben. Die richtigen Eingaben werden mit dem maximalen (d. h. dem ersten Versuch) Schwellenwert gefunden. Bei fehlerhaften Eingaben muss für eine korrekte Zuordnung die Schwelle gesenkt werden**

## 9.4 Lineare Netzwerke mit minimalen Fehlern

Die in den bisher vorgestellten neuronalen Modellen verwendete Hebb-Regel arbeitet mit Assoziationen zwischen benachbarten Neuronen. Im assoziativen Gedächtnis wird dies ausgenutzt, um eine Zuordnung von Abfragevektoren zu Zielen zu lernen. Dies funktioniert in vielen Fällen sehr gut, insbesondere wenn die Abfragevektoren linear unabhängig sind. Ist diese Bedingung nicht erfüllt, beispielsweise weil zu viele Trainingsdaten vorliegen, stellt sich die Frage: Wie finden wir die optimale Gewichtsmatrix? Optimal bedeutet, dass der durchschnittliche Fehler minimiert wird.

Wir Menschen sind in der Lage, aus Fehlern zu lernen. Die Hebb-Regel bietet diese Möglichkeit nicht. Der im Folgenden beschriebene Backpropagation-Algorithmus verwendet eine elegante, aus der Funktionsnäherung bekannte Lösung, um die Gewichte so zu ändern, dass der Fehler in den Trainingsdaten minimiert wird.

Es seien N Paare von Trainingsvektoren

$$T = \{(q^{1}, t^{1}), \ldots, (Q^{N}, t^{N})\}$$

gegeben sein mit $q^{p} \in [0,1]^{N}$, $t^{p} \in [0,1]^{M}$. Wir suchen eine Funktion $f : [0,1]^{N} \to [0,1]^{M}$, was den quadratischen Fehler minimiert

$$\sum_{p=1}^{N} (f(q^{P}) - t^{p})^{2}$$

auf die Daten. Nehmen wir zunächst an, dass die Daten keine Widersprüche enthalten. Das heißt, es gibt keinen Abfragevektor in den Trainingsdaten, der zwei verschiedenen Zielen zugeordnet werden sollte. In diesem Fall ist es nicht schwierig, eine Funktion zu finden, die den quadratischen Fehler minimiert. Tatsächlich gibt es unendlich viele Funktionen, die den Fehler auf Null bringen. Wir definieren die Funktion

$$f(q) = 0, \text{ wenn } q \in \{q^{1}, \ldots, q^{N}\}$$

Und

$$f(q^{P}) = t^{p} \quad \forall p \in \{1, \ldots, N\}.$$

Dies ist eine Funktion, die sogar den Fehler in den Trainingsdaten auf Null setzt. Was wollen wir mehr? Warum sind wir mit dieser Funktion nicht zufrieden?

Die Antwort lautet: Weil wir ein intelligentes System bauen wollen! Intelligent bedeutet unter anderem, dass sich die gelernte Funktion gut von den Trainingsdaten auf neue, unbekannte Daten aus demselben repräsentativen Datensatz verallgemeinern lässt. Mit anderen Worten bedeutet es: Wir wollen keine Überanpassung der Daten durch Auswendiglernen. Was wollen wir denn wirklich?

Wir wollen eine Funktion, die glatt ist und den Raum zwischen den Punkten „ausgleicht“. Kontinuität und die Fähigkeit, mehrere Derivate zu übernehmen, wären sinnvolle Anforderungen. Da es auch unter diesen Bedingungen noch unendlich viele Funktionen gibt, die den Fehler zu Null machen, müssen wir diese Klasse von Funktionen noch weiter einschränken.

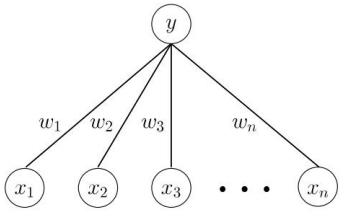



Abb. 9.13 Ein zweischichtiges Netzwerk mit einem Ausgang Neuron

### 9.4.1 Methode der kleinsten Quadrate

Die einfachste Wahl ist eine lineare Abbildung. Wir beginnen mit einem zweischichtigen Netzwerk (Abb. 9.13) , in dem das einzelne Neuron der zweiten Schicht seine Aktivierung anhand berechnet

$$y = f\left(\sum_{i=1}^{N} W_i x_i\right)$$

mit f(x) = x. Die Tatsache, dass wir hier nur Output-Neuronen betrachten, stellt keine wirkliche Einschränkung dar, da ein zweischichtiges Netzwerk mit zwei oder mehr Output-Neuronen immer in unabhängige Netzwerke mit identischen Input-Neuronen für jedes der ursprünglichen Output-Neuronen aufgeteilt werden kann. Die Gewichte der Teilnetze sind alle unabhängig. Die Verwendung einer Sigmoidfunktion anstelle der linearen Aktivierung bringt hier keinen Vorteil, da die Sigmoidfunktion streng monoton wachsend ist und die Ordnungsbeziehung zwischen verschiedenen Ausgabewerten nicht v

Gewünscht ist ein Vektor w, der den quadratischen Fehler minimiert

$$E(w) = \sum_{p=1}^{N} (w_{qp} \ddot{y} t_p)^2 = \sum_{p=1}^{N} \left(\sum_{i=1}^{N} p_{wiq_i} \ddot{y} t_p\right)^2 .$$

Als notwendige Bedingung für ein Minimum dieser Fehlerfunktion müssen alle partiellen Ableitungen Null sein. Wir fordern also für j = 1,...,n:

$$\frac{\ddot{y}E}{\ddot{y}w_j} = 2 \sum_{p=1}^{N} \left(\sum_{i=1}^{N} w_{iq_i} p \ddot{y} t_p\right) q_j p = 0.$$

Multipliziert man dies, erhält man ein Ergebnis

$$\sum_{p=1}^{N} \left(\sum_{i=1}^{N} p_{wiq_i}\right) P_{q_j} p \ddot{y} t_{p_{q_j}} = 0,$$

und das Vertauschen der Summen ergibt das lineare Gleichungssystem

$$\sum_{i=1}^{N} w_i \sum_{p=1}^{P} q_i^p q_j^p = \sum_{p=1}^{P} t_p q_j^p,$$

welches mit

$$A_{ij} = \sum_{p=1}^{P} q_i^p q_j^p \quad \text{und} \quad b_j = \sum_{p=1}^{P} t_p q_j^p \tag{9.12}$$

kann als Matrixgleichung geschrieben werden

$$Aw = b. \tag{9.13}$$

Diese sogenannten Normalgleichungen haben immer mindestens eine Lösung und, wenn A invertierbar ist, genau eine. Darüber hinaus ist die Matrix A positiv-definit, was zur Folge hat, dass die gefundene Lösung im Einzelfall ein globales Minimum ist.

Dieser Algorithmus ist als Methode der kleinsten Quadrate bekannt.

Die Rechenzeit für den Aufbau der Matrix A wächst mit $\ddot{y}(N \cdot n^2)$ und die Zeit zum Lösen des Gleichungssystems als $O(n^3)$. Diese Methode kann sehr einfach auf die Einbeziehung mehrerer Ausgabeneuronen erweitert werden, da, wie bereits erwähnt, bei zweischichtigen Feedforward-Netzwerken die Ausgabeneuronen unabhängig voneinander sind.

## 9.4.2 Anwendung auf die Appendizitis-Daten

Als Anwendung ermitteln wir nun einen linearen Score für die Blinddarmentzündungsdiagnose. Aus den aus Abschn . 8.4.5 verwenden wir die Methode der kleinsten Quadrate, um eine lineare Abbildung von Symptomen auf die kontinuierlichen Klassenvariablen *AppScore* mit Werten im Intervall [0, 1] zu bestimmen und die lineare Kombination zu erhalten

$$\begin{aligned} \textit{AppScore} = {} & 0{,}00085\text{Alter} \ddot{y} 0{,}125\, \textit{Geschlecht} + 0{,}025\text{P1Q} + 0{,}035\text{P2Q} \ddot{y} 0{,}021\text{P3Q} \\ & \ddot{y} 0{,}025\text{P4Q} + 0{,}12\, \textit{TensLoc} + 0{,}031\, \textit{TensGlo} + 0{,}13\, \textit{Losl} \\ & + 0{,}081\text{Conv} + 0{,}0034\text{RectS} + 0{,}0027\, \textit{TAxi} + 0{,}0031\, \textit{TRec} \\ & + 0{,}000021\, \textit{Leuko} \ddot{y} \textit{0{,}11Diab} \ddot{y} 1{,}83. \end{aligned}$$

Diese Funktion liefert kontinuierliche Variablen für *AppScore,* obwohl die eigentliche binäre Klassenvariable *App* nur die Werte 0 und 1 annimmt. Wir müssen uns also wie beim Perzeptron für einen Schwellenwert entscheiden. Der Klassifizierungsfehler des Scores als Funktion des Schwellenwerts ist in Abb. 9.14 auf Seite 243 für die Trainingsdaten und die Testdaten aufgeführt. Wir sehen deutlich, dass beide Kurven nahezu gleich sind und haben

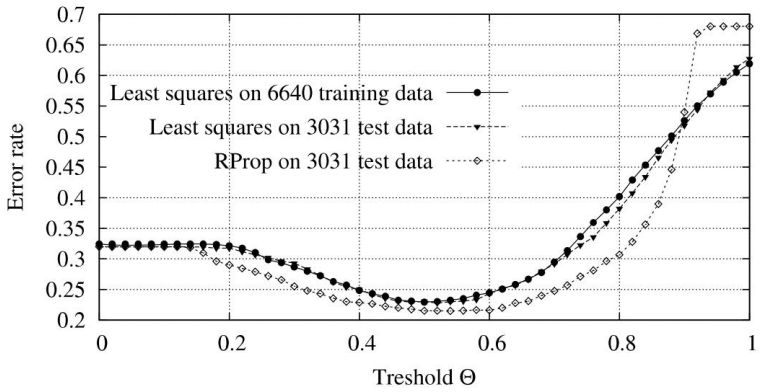



**Abb. 9.14 Fehler der kleinsten Quadrate für Trainings- und Testdaten**

**ihr Minimum bei ÿ = 0,5. An der geringen Differenz der beiden Kurven erkennen wir, dass eine Überanpassung für diese Methode kein Problem darstellt, da das Modell sehr gut aus den Testdaten verallgemeinert.**

**Ebenfalls in der Abbildung dargestellt ist das Ergebnis für das nichtlineare, dreischichtige RProp-Netzwerk (Abschn. 9.5) mit einem etwas geringeren Fehler für Schwellenwerte zwischen 0,2 und 0,9. Für die praktische Anwendung des abgeleiteten Scores und die korrekte Bestimmung des Schwellenwerts ÿ ist es wichtig, nicht nur den Fehler zu betrachten, sondern auch nach Fehlerart (nämlich falsch positiv und falsch negativ) zu unterscheiden, wie dies in der LEXMED-Anwendung geschieht in Abb. 7.10 auf Seite 142. In der dort gezeigten ROC-Kurve ist auch der hier berechnete Score dargestellt. Wir sehen, dass das einfache lineare Modell dem LEXMED-System deutlich unterlegen ist. Offensichtlich sind lineare Approximationen für viele komplexe Anwendungen nicht leistungsstark genug.**

### 9.4.3 Die Delta-Regel

**Die Methode der kleinsten Quadrate ist, wie das Perzeptron- und Entscheidungsbaum-Lernen, ein sogenannter *Batch-Learning-Algorithmus,* im Gegensatz zum *inkrementellen Lernen.* Beim Batch-Lernen müssen alle Trainingsdaten in einem Durchlauf gelernt werden. Wenn neue Trainingsdaten hinzukommen, können diese nicht einfach zusätzlich zu den bereits vorhandenen gelernt werden. Der gesamte Lernvorgang muss mit dem erweiterten Satz wiederholt werden. Dieses Problem wird durch inkrementelle Lernalgorithmen gelöst, die das gelernte Modell an jedes weitere neue Beispiel anpassen können. In den Algorithmen, die wir in der folgenden Diskussion betrachten werden, werden wir die Gewichte für jedes neue Trainingsbeispiel durch die Regel addit**

$$wj = wj + wj .$$

**Um eine inkrementelle Variante der Methode der kleinsten Quadrate abzuleiten, betrachten wir die oben berechneten n partiellen Ableitungen der Fehlerfunktion noch einmal**

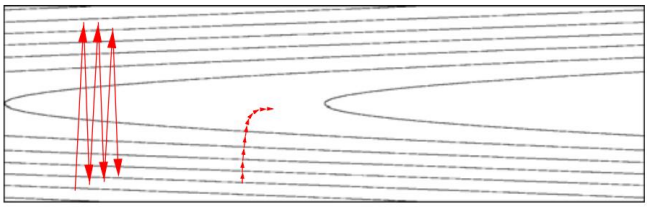


**Abb. 9.15 Gradientenabstieg für ein großes ÿ *(links)* und ein sehr kleines ÿ *(rechts)* in ein Tal, das flach nach *rechts abfällt.* Für ein großes ÿ gibt es Schwingungen um das Tal herum. Bei einem zu kleinen ÿ hingegen erfolgt die Konvergenz im flachen Tal sehr langsam**

$$\frac{\ddot{y}E}{\ddot{y}wj} = 2 \sum_{p=1}^{N} \sum_{i=1}^{N} wiq\ i^{p}\ \ddot{y}\ t^{p} \quad p_{q\,j}\ .$$

**Der Farbverlauf**

$$\ddot{y}E = \frac{\ddot{y}E}{\ddot{y}w1},\ldots, \frac{\ddot{y}E}{\ddot{y}wn}$$

**als Vektor aller partiellen Ableitungen der Fehlerfunktion zeigt in die Richtung des stärksten Anstiegs der Fehlerfunktion im n-dimensionalen Raum der Gewichte. Bei der Suche nach einem Minimum folgen wir daher der Richtung des negativen Gradienten. Als Formel zur Änderung der Gewichte erhalten wir**

$$\ddot{y}\ wj = \ddot{y}\ \frac{}{2} \frac{\ddot{y}E}{\ddot{y}wj} = \ddot{y}\ddot{y} \sum_{p=1}^{N} \sum_{i=1}^{N} wiq\ i^{p}\ \ddot{y}\ t^{p} \quad p_{q\,j}\ ,$$

**wobei die *Lernrate* ÿ eine frei wählbare positive Konstante ist. Ein größeres ÿ beschleunigt die Konvergenz, erhöht aber gleichzeitig das Risiko einer Oszillation um Minima oder flache Täler. Daher ist die optimale Wahl von ÿ keine einfache Aufgabe (siehe Abb. 9.15).**
**Am Anfang wird oft ein großes ÿ verwendet, zum Beispiel ÿ = 1, das dann langsam schrumpft.**
**Durch Ersetzen der Aktivierung**

$$p\ y = \sum_{i=1}^{N} p\ wiq\ i$$

**des Ausgabeneurons für das angewandte Trainingsbeispiel p die Formel ist vereinfacht und q erhalten wir die *Delta-Regel***

$$wj = \ddot{y} \sum_{p=1}^{N} (tp\ \ddot{y}\ y \quad p\ )qp_{j}\ .$$

Abb. 9.16 Lernen eines zweischichtigen linearen Netzwerks mit der Delta-Regel. Beachten Sie, dass die Gewichtsänderungen immer dann auftreten, nachdem alle Trainingsdaten angewendet wurden

```
DELTALEARNING(TrainingExamples, ÿ)
Initialisieren Sie alle Gewichte wj zufällig
Wiederholen
    Sie w = 0
  Für alle (q p ,tp ) ÿ Trainingsbeispiele
      Berechnen Sie die                P = wpq p
        Netzwerkausgabe yw = w +
  ÿ(tp ÿ y p )q p w = w + w
Bis w konvergiert
```

```
DELTALEARNINGINCREMENTAL(TrainingExamples, ÿ)
Initialisieren Sie alle Gewichte wj zufällig
Wiederholen
  Für alle (q p ,tp ) ÿ Trainingsbeispiele
      Berechnen Sie die                P = wpq p
      Netzwerkausgabe yw = w + ÿ(tp ÿ y p )q p
Bis w konvergiert
```

Abb. 9.17 Inkrementelle Variante der Delta-Regel

Somit wird für jedes Trainingsbeispiel die Differenz zwischen dem Ziel t p und dem tatsächlichen p des Netzwerks für die gegebene Eingabe q berechnet. Nach der Summierung aller Muster über die Ausgabe y werden die Gewichte dann proportional zur Summe geändert. Dieser Algorithmus ist in Abb. 9.16 dargestellt .

Wir sehen, dass der Algorithmus immer noch nicht wirklich inkrementell ist, da die Gewichtsänderungen erst auftreten, nachdem alle Trainingsbeispiele einmal angewendet wurden. Diesen Mangel können wir beheben, indem wir nach jedem Trainingsbeispiel direkt die Gewichte ändern (inkrementeller Gradient de Scent) (Abb. 9.17) , was streng genommen keine korrekte Umsetzung der Delta-Regel mehr darstellt.

## 9.4.4 Vergleich zum Perzeptron

Die in Abschn. eingeführte Lernregel für Perzeptrone. 8.2.1, die Methode der kleinsten Quadrate und die Delta-Regel können verwendet werden, um lineare Funktionen aus Daten zu generieren. Bei Perzeptronen wird jedoch im Gegensatz zu den anderen Methoden ein Klassifikator für linear trennbare Klassen durch die Schwellenwertentscheidung gelernt. Die anderen beiden Methoden erzeugen jedoch eine lineare Näherung der Daten. Wie in Abschn. 9.4.2 kann aus der linearen Abbildung bei Bedarf durch Anwendung einer Schwellenwertfunktion e

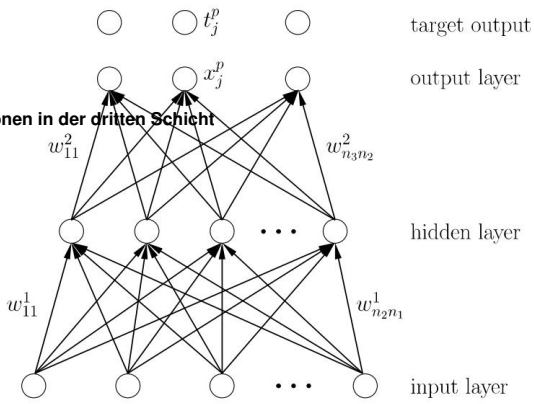



**Abb. 9.18 Ein dreischichtiges Backpropagation-Netzwerk mit n1 Neuronen in der ersten, n2 Neuronen in der zweiten und n3 Neuronen in der dritten Schicht**

Das Perzeptron und die Delta-Regel sind iterative Algorithmen, bei denen die Zeit bis zur Konvergenz stark von den Daten abhängt. Bei linear trennbaren Daten lässt sich für das Perzeptron eine Obergrenze für die Anzahl der Iterationsschritte finden. Für die Delta-Regel hingegen gibt es nur eine Garantie der asymptotischen Konvergenz ohne Limit [HKP91].

Bei der Methode der kleinsten Quadrate besteht das Lernen darin, ein lineares Gleichungssystem für den Gewichtsvektor aufzustellen und zu lösen. Es gibt also eine harte Grenze für die Rechenzeit. Aus diesem Grund ist die Methode der kleinsten Quadrate immer dann vorzuziehen, wenn kein inkrementelles Lernen erforderlich ist.

## 9.5 Der Backpropagation-Algorithmus

Mit dem Backpropagation-Algorithmus stellen wir nun das am häufigsten verwendete neuronale Modell vor. Der Grund für seine weite Verbreitung ist seine universelle Vielseitigkeit für beliebige Approximationsaufgaben. Der Algorithmus geht direkt auf die inkrementelle Delta-Regel zurück. Im Gegensatz zur Delta-Regel wendet sie als Aktivierungsfunktion eine nichtlineare Sigmoidfunktion auf die gewichtete Summe der Eingaben an. Darüber hinaus kann ein Backpropagation-Netzwerk mehr als zwei Neuronenschichten haben. Bekannt wurde der Algorithmus durch den Artikel [RHR86] in der legendären PDP-Sammlung [RM86].

In Abb. 9.18 ist ein typisches Backpropagation-Netzwerk mit einer Eingabeschicht und einer verborgenen und eine Ausgabeebene wird angezeigt. Da der aktuelle Ausgabewert $x_j$ der $P$ Ausgabeschicht dargestellt Neuronenschicht $j$ mit dem Zielausgabewert $t_j$ verglichen wird, werden diese parallel zu $j$ gezeichnet . Mit Ausnahme der Eingabeneuronen berechnen alle Neuronen ihren aktuellen Wert $x_j$ nach der Regel

$$x_j = f\left(\sum_{i=1}^{N} w_{ji} x_i\right) \tag{9.14}$$

Dabei ist n die Anzahl der Neuronen in der vorherigen Schicht. Wir verwenden die Sigmoidfunktion

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Analog zur inkrementellen Delta-Regel werden die Gewichte proportional zum negativen Gradienten der über die Ausgabeneuronen summierten quadratischen Fehlerfunktion geändert

$$E_p(w) = \frac{1}{2} \sum_{k \in \text{Ausgabe}} (t_{pk} - x_{pk})^2$$

für das Trainingsmuster p:

$$\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}.$$

Zur Ableitung der Lernregel wird Ep durch den obigen Ausdruck ersetzt. Innerhalb des Ausdrucks wird xk durch (9.14) auf Seite 246 ersetzt . Innerhalb der Gleichung erfolgen die Ausgaben xi der Neuronen der nächsten, tieferen Schicht rekursiv usw. Durch mehrfache Anwendung der Kettenregel (siehe [RHR86] bzw [Zel94]) erhalten wir die Backpropagation-Lernregel

$$\Delta_p w_{ji} = \eta \delta_{pj} x_{pi},$$

mit

$$\delta_{pj} = \begin{cases} x'_j(t_{pj} - x_{pj}) & \text{wenn } j \text{ ein Ausgabeneuron ist,} \\ x'_j \sum_k \delta_{pk} w_{kj} & \text{wenn } j \text{ ein verstecktes Neuron ist,} \end{cases}$$

was auch als verallgemeinerte Delta-Regel bezeichnet wird. Für alle Neuronen enthält die Formel zur Gewichtsänderung wj i von Neuron i zu Neuron j (siehe Abb. 9.19 auf Seite 248) wie die Hebb-Regel einen Term ÿx S. x j . Der neue Faktor (1ÿx j P ) erzeugt das Symmetrie try, der in der Hebb-Regel fehlt, zwischen den Aktivierungen 0 und 1 des Neurons. Für die Ausgabeneuronen der Faktor (tp j . J ÿ x p J ) kümmert sich um einen Gewichtsveränderungs-Profi proportional zum Fehler. Für die verborgenen Neuronen der P des Neurons j wird berechnet Wert ÿ j rekursiv aus allen P k der Neuronen der nächsthöheren Ebene.

Änderungen ÿ Der gesamte Ablauf des Lernprozesses ist in Abb. 9.20 auf Seite 248 dargestellt. Nach der Berechnung der Ausgabe des Netzwerks (Vorwärtsausbreitung) für ein Trainingsbeispiel wird der Approximationsfehler berechnet. Dies wird dann während der Rückwärtsausbreitung verwendet, um die Gewichte von Schicht zu Schicht rückwärts zu ändern. Der gesamte Vorgang wird dann auf alle Trainingsbeispiele angewendet und wiederholt, bis sich die Gewichte nicht mehr ändern oder ein Zeitlimit erreicht ist.

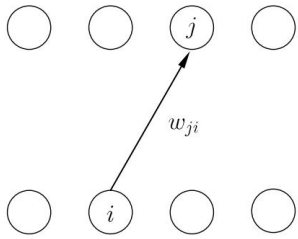



**Abb. 9.19 Bezeichnung der Neuronen und Gewichte für die Anwendung der Backpropagation-Regel**

***BACKPROPAGATION(TrainingExamples,*** **ÿ)**
**Initialisieren Sie alle Gewichte wj auf Zufallswerte.**

**Wiederholen Sie den Vorgang für alle (q p ,t**
**p auf die Eingabeschicht 1. p ) ÿ *Trainingsbeispiele***
**Wenden Sie den Abfragevektor**
**q an 2. Vorwärtsausbreitung: Für alle Schichten ab der**
**ersten verborgenen Schicht aufwärts Für jedes Neuron de Schicht**
**Berechnen Sie die Aktivierung xj = f(ni=1 wj ixi)**
**3. Berechnung des quadratischen Fehlers Ep(w)**
**4. Rückwärtsausbreitung: Für**
**alle Gewichtungsebenen von der letzten abwärts. Für**
**jedes Gewicht wj i = wj**
**wj i i + ÿÿp J x i p**
**Bis w konvergiert oder das Zeitlimit erreicht ist**

**Abb. 9.20 Der Backpropagation-Algorithmus**

**Wenn wir ein Netzwerk mit mindestens einer verborgenen Schicht aufbauen, können nichtlineare Abbildungen gelernt werden. Ohne verborgene Schichten sind die Ausgabeneuronen trotz der Sigmoidfunktion nicht leistungsfähiger als ein lineares Neuron. Der Grund dafür ist, dass die Sigmoidfunktion streng monoton ist. Gleiches gilt für mehrschichtige Netzwerke, die lediglich eine lineare Funktion als Aktivierungsfunktion verwenden, beispielsweise die Identitätsfunktion. Dies liegt daran, dass verkettete Ausführungen linearer Zuordnungen insgesamt linear sind.**

**Ebenso wie beim Perzeptron die Klasse der darstellbaren Funktionen werden auch vergrößert, wenn wir eine variable Sigmoidfunktion verwenden**

$$f(x) = 1 \frac{1}{+ e^{ÿ(x ÿ ÿ)}}.$$

**mit Schwelle ÿ. Die Umsetzung erfolgt analog zu dem in Abschn. 8.2, bei dem in die Eingabeschicht und in jede verborgene Schicht ein Neuron eingefügt wird, dessen Aktivierung immer den Wert Eins hat und das mit Neuronen der nächsthöheren Ebene verbunden ist. Die Gewichte dieser Verbindungen werden normal gelernt und stellen den Schwellenwert ÿ der Nachfolgeneuronen dar.**

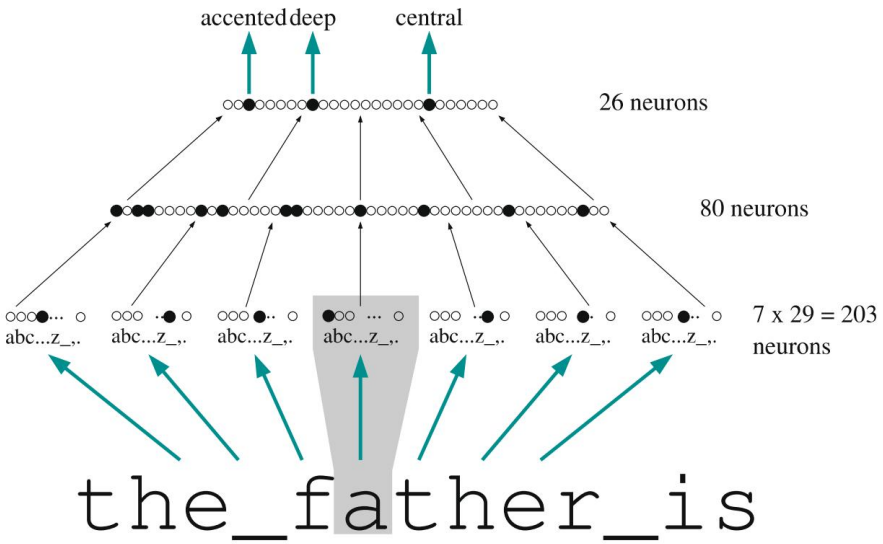



**Abb. 9.21 Das NETtalk-Netzwerk ordnet einen Text seinen Ausspracheattributen zu**

## 9.5.1 NETtalk: Ein Netzwerk lernt sprechen

**Sejnowski und Rosenberg haben 1986 sehr eindrucksvoll gezeigt, wozu Backprop-Agation in der Lage ist [SR86]. Sie haben ein System entwickelt, das in der Lage ist, englischen Text aus einer Textdatei verständlich vorzulesen.** Die Architektur des in Abb. 9.21 **dargestellten Netzwerks besteht aus einer Eingabeschicht mit 7 × 29 = 203 Neuronen**
**Dabei werden der aktuelle Buchstabe und drei vorangegangene Buchstaben sowie drei nachfolgende Buchstaben kodiert. Für jeden dieser sieben Buchstaben sind 29 Neuronen reserviert Zeichen „a...z ,.“ Die Eingabe wird auf die 26 Ausgabeneuronen über 80 abgebildet versteckte Neuronen, von denen jedes für ein bestimmtes Phonem steht. Zum Beispiel die „a“ in „Vater“ wird tief, akzentuiert und zentral ausgesprochen. Das Netzwerk wurde geschult mit 1.000 Wörtern, die zufällig Buchstabe für Buchstabe nacheinander angewendet wurden. Für jeden Buchstaben wurde manuell die Zielausgabe für seine Intonation angegeben. Um die Intonationsattribute in tatsächliche Klänge zu übersetzen, Teil des Sprachsynthesesystems Es wurde DECtalk verwendet. Durch vollständige Interkonnektivität enthält das Netzwerk ein Ganzes von 203 × 80 + 80 × 26 = 18320 Gewichte.**

**Das System wurde mithilfe eines Simulators auf einer VAX 780 mit etwa 50 Zyklen trainiert über alle Wörter. Bei etwa 5 Zeichen pro Wort im Durchschnitt also etwa 5·50·1000 = Es waren 250.000 Iterationen des Backpropagation-Algorithmus erforderlich. Mit einer Geschwindigkeit von ca Ein Zeichen pro Sekunde bedeutet dies etwa 69 Stunden Rechenzeit. Der Die Entwickler beobachteten viele Eigenschaften des Systems, die denen des Menschen sehr ähnlich sind Lernen. Das System kann zunächst nur undeutlich sprechen oder einfache Wörter verwenden. Mit der Zeit Es verbesserte sich weiter und erreichte schließlich eine 95-prozentige Korrektheit der ausgesprochenen Buchstaben.**

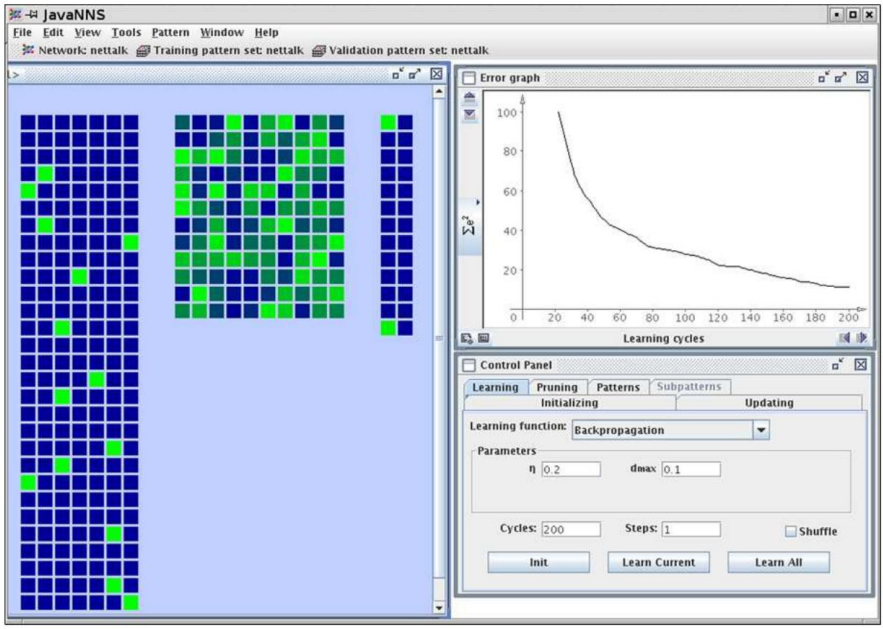



**Abb. 9.22 Das NETtalk-Netzwerk in JNNS. Im *linken Fenster* sehen wir von *links* nach *rechts* die 7 · 29 Eingabeneuronen, 80 versteckte und 26 Ausgabeneuronen. Aufgrund ihrer großen Anzahl werden die Gewichte im Netzwerk weggelassen. *Oben rechts* ist eine Lernkurve zu sehen, die die Entwicklung des quadratischen Fehlers über die Zeit zeigt**

**Für visuelle Experimente mit neuronalen Netzen empfiehlt sich der Java Neural Network Simulator JNNS [Zel94]. Das mit JNNS geladene und trainierte NETtalk-Netzwerk ist in Abb. 9.22 dargestellt .**

## 9.5.2 Erlernen von Heuristiken für Theorembeweiser

**Im Kap. In Kapitel 6 wurden Algorithmen für die heuristische Suche, wie der Aÿ -Algorithmus und der IDAÿ -Algorithmus, diskutiert. Um eine signifikante Reduzierung des Suchraums zu erreichen, ist eine gute Heuristik für die Implementierung dieser Algorithmen erforderlich. In Abschn. In Abschnitt 4.1 wurde das Problem des explodierenden Suchraums während der Beweissuche von Theorembeweisern aufgezeigt. Dieses Problem wird durch die große Anzahl möglicher Inferenzschritte in jedem Schritt verursacht.**

**Wir versuchen nun, heuristische Proof-Controlling-Module zu erstellen, die für den nächsten Schritt die verschiedenen Alternativen bewerten und dann die Alternative mit der besten Bewertung auswählen. Im Falle einer Auflösung könnte die Bewertung der verfügbaren Klauseln durch eine Funktion erfolgen, die basierend auf bestimmten Attributen von Klauseln wie der Anzahl der Literale, der Komplexität der Begriffe usw. einen Wert für jedes Paar von Auflösungen berechnet able-Klauseln. In [ESS89, SE90] wurde eine solche Heuristik für den Theorembeweis SETHEO mittels Backpropagation erlernt. Beim Erlernen der Heuristik we**

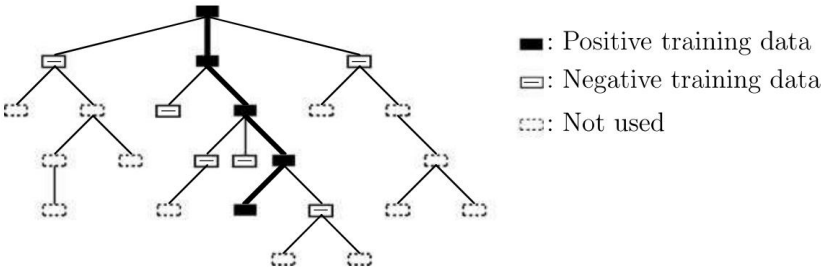



**Abb. 9.23 Suchbaum mit einem erfolgreichen Pfad, dessen Knoten als positiv bewertet werden, und den negativ bewerteten erfolglosen Zweigen**

Die aktuelle Klausel wurde für jeden gefundenen Beweis als Trainingsdaten für die positive Klasse gespeichert in jeder Filiale während der Suche. Bei allen Zweigen, die nicht zu einem Beweis führten, wird der Attribute der aktuellen Klausel wurden als Trainingsdaten für die negative Klasse gespeichert. Es wird ein solcher Baum mit einem erfolgreichen Pfad und den entsprechenden Knotenbewertungen skizziert in Abb. 9.23.

Auf diesen Daten wurde ein Backpropagation-Netzwerk trainiert und anschließend zur Auswertung genutzt Klauseln im Prüfer. Für jede Klausel wurden 16 numerische Attribute berechnet, normiert und in einem Eingabeneuron kodiert. Das Netzwerk wurde mit 25 versteckten trainiert Neuronen und ein Ausgabeneuron für die Klasse (positiv/negativ).

Es zeigte sich, dass die Zahl der Schlussfolgerungsversuche um ein Vielfaches reduziert werden kann bei schwierigen Problemen um Größenordnungen, was letztendlich die Rechenzeit von Stunden auf Sekunden reduziert. Dadurch wurde es möglich, Theoreme zu beweisen die ohne Heuristik unerreichbar waren.

## 9.5.3 Probleme und Verbesserungen

Backpropagation ist mittlerweile 25 Jahre alt und hat sich in verschiedenen Anwendungen bewährt, zum Beispiel in der Mustererkennung und in der Robotik. Allerdings ist seine Anwendung manchmal problematisch. Vor allem, wenn das Netzwerk viele tausend Gewichte hat und Da viele Trainingsdaten gelernt werden müssen, treten zwei Probleme auf:

Das Netzwerk konvergiert häufig gegen lokale Minima der Fehlerfunktion. Außerdem, Die Rückausbreitung konvergiert oft sehr langsam. Das bedeutet, dass viele Iterationen vorbei sind Alle Trainingsmuster werden benötigt. Zur Linderung dieser Probleme wurden viele Verbesserungen vorgeschlagen. Wie in Abschn. 9.4.3 können Schwingungen vermieden werden durch langsame Reduzierung der Lernrate ÿ, wie in Abb. 9.15 auf Seite 244 dargestellt.

Eine weitere Methode zur Reduzierung von Schwingungen ist die Verwendung eines Impulsterms Aktualisierung der Gewichte, wodurch sichergestellt wird, dass sich die Richtung des Gradientenabfalls nicht ändert sich von einem Schritt zum nächsten zu dramatisch verändern. Hier zum aktuellen Gewicht Ändere pwj i(t) zum Zeitpunkt t wird ein weiterer Teil der Änderung pwj i(t ÿ 1) aus dem vorherigen Schritt hinzugefügt. Die Lernregel ändert sich dann zu

$$\text{pwj i(t)} = \ddot{y}\ddot{y}p \quad {}_{J}{}^{X\,P}{}_{\ddot{}} + \ddot{y}\ \text{pwj i(t } \ddot{y}\ 1)$$

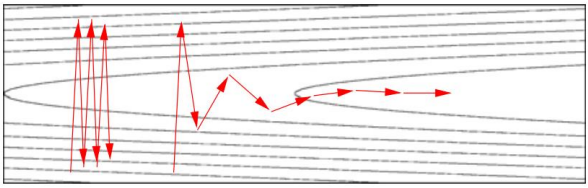


Abb. 9.24 Abrupte Richtung Änderungen werden durch geglättet die Nutzung des Impulses Begriff. Eine Iteration ohne die Impulsterm *(links)* in Vergleich zur Iteration mit der Impulsterm *(rechts)*

mit einem Parameter ÿ zwischen Null und Eins. Dies wird zweidimensional dargestellt Beispiel in Abb. 9.24.

Eine andere Idee besteht darin, die lineare Fehlerfunktion anstelle des quadratischen Fehlers zu minimieren Funktion, die das Problem der langsamen Konvergenz in flache Täler verringert.

Der Gradientenabstieg bei der Backpropagation basiert letztlich auf einer linearen Approximation der Fehlerfunktion. Quickprop, ein Algorithmus, der eine quadratische Näherung an die Fehlerfunktion verwendet und so eine schnellere Konvergenz erreicht, wurde von entwickelt Scott Fahlmann.

Durch geschickte Vereinheitlichung der genannten Verbesserungen und weiterer heuristischer Tricks gelang Martin Riedmiller eine weitere Optimierung mit dem RProp-Algorithmus [RB93]. RProp hat die Backpropagation ersetzt und ist der neue Zustand von Kunst-Feedforward-Approximationsalgorithmus für neuronale Netze. In Abschn. 8,8 haben wir angewendet RProp hat bei der Klassifizierung der Blinddarmentzündungsdaten einen Fehler festgestellt ungefähr die gleiche Größe wie ein erlernter Entscheidungsbaum.

## 9.6 Support Vector Machines

Feedforward-Neuronale Netze mit nur einer Gewichtsschicht sind linear. Linearität führt zu einfachen Netzwerken und schnellem Lernen mit garantierter Konvergenz. Darüber hinaus ist die Gefahr einer Überanpassung bei linearen Modellen gering. Für viele Anwendungen, Allerdings sind die linearen Modelle nicht stark genug, etwa weil die relevanten Klassen nicht linear trennbar sind. Hier kommen mehrschichtige Netzwerke wie Backpropagation zum Einsatz, mit der Folge, dass lokale Minima, Konvergenz entstehen

Es kann zu Problemen und einer Überanpassung kommen.

Ein vielversprechender Ansatz, der die Vorteile linearer und nichtlinearer Modelle vereint, folgt der Theorie der Support Vector Machines (SVM), die wir vorstellen werden Grobe Gliederung anhand eines Zwei-Klassen-Problems.4

Im Fall zweier linear separierbarer Klassen ist es leicht, einen dividierenden Hyper zu finden Ebene, zum Beispiel mit der Perzeptron-Lernregel. Allerdings gibt es sie in der Regel unendlich viele solcher Ebenen, wie im zweidimensionalen Beispiel in Abb. 9.25 auf Seite 253. Gesucht wird eine Ebene, die zu beiden den größten Mindestabstand hat Klassen. Diese Ebene wird in der Regel durch wenige Punkte im Randbereich eindeutig definiert. Diese Punkte, die sogenannten *Stützvektoren,* haben alle den gleichen Abstand zur Teilung

4Support-Vektor-Maschinen sind keine neuronalen Netze. Aufgrund ihrer historischen Entwicklung und ihrer mathematischen Verwandtschaft zu linearen Netzwerken werden sie hier jedoch diskutiert.

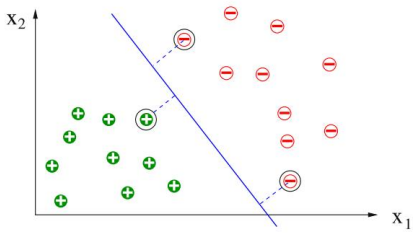



Abb. 9.25 Zwei Klassen mit der maximalen Trennlinie. Die *eingekreisten Punkte* sind die Stützvektoren

Linie. Um die Unterstützungsvektoren zu finden, gibt es einen effizienten Optimierungsalgorithmus. Interessant ist, dass die optimale teilende Hyperebene durch einige Parameter bestimmt wird, nämlich durch die Stützvektoren. Daher ist die Gefahr einer Überanpassung gering.

Support-Vektor-Maschinen wenden diesen Algorithmus in einem zweistufigen Prozess auf nichtlinear trennbare Probleme an: Im ersten Schritt wird eine nichtlineare Transformation auf die Daten angewendet, mit der Eigenschaft, dass die transformierten Daten linear trennbar sind. Im zweiten Schritt werden dann die Stützvektoren im transformierten Raum bestimmt.

Der erste Schritt ist hochinteressant, aber nicht ganz einfach. Tatsächlich ist es immer möglich, die Klassen durch Transformation des Vektorraums linear trennbar zu machen, solange die Daten keine Widersprüche enthalten.5 Eine solche Trennung kann beispielsweise durch Einführung einer neuen (n + 1)-ten Dimension und erreicht werden die Definition

$$x_{n+1} = \begin{cases} 1, \text{ wenn } x \text{ ÿ } \textit{Klasse} \\ 1, 0, \text{ wenn } x \text{ ÿ } \textit{Klasse } 0. \end{cases}$$

Diese Formel hilft jedoch nicht viel, da sie nicht auf neue Punkte einer unbekannten Klasse anwendbar ist, die klassifiziert werden sollen. Wir benötigen daher eine allgemeine Transformation, die möglichst unabhängig von den aktuellen Daten ist. Es kann gezeigt werden, dass es solche generischen Transformationen auch für beliebig geformte Klassenteilungsgrenzen im ursprünglichen Vektorraum gibt. Im transformierten Raum sind die Daten dann linear trennbar. Allerdings wächst die Anzahl der Dimensionen des neuen Vektorraums exponentiell mit der Anzahl der Dimensionen des ursprünglichen Vektorraums.
Allerdings ist die Vielzahl der neuen Dimensionen nicht so problematisch, da bei der Verwendung von Stützvektoren die Teilungsebene, wie oben erwähnt, nur durch wenige Parameter bestimmt wird.

Die zentrale nichtlineare Transformation des Vektorraums wird als *Kernel bezeichnet,* weshalb Support-Vektor-Maschinen auch als Kernel-Methoden bezeichnet werden. Die ursprünglich für Klassifikationsaufgaben entwickelte SVM-Theorie wurde erweitert und kann nun auch auf Regressionsprobleme angewendet werden.

Die hier verwendete Mathematik ist sehr interessant, aber für eine erste Einführung zu umfangreich. Um tiefer in diesen vielversprechenden jungen Zweig des maschinellen Lernens einzutauchen, verweisen wir den Leser auf [SS02, Alp04] und [Bur98].

5Ein Datenpunkt ist widersprüchlich, wenn er zu beiden Klassen gehört.

## 9.7 Anwendungen

Neben den hier aufgeführten Beispielen gibt es unzählige Anwendungen für neuronale Netze in allen Industriezweigen. Ein sehr wichtiger Bereich ist die Mustererkennung in all ihren Formen, sei es die Analyse von Fotos zur Erkennung von Personen oder Gesichtern, die Erkennung von Fischschwärmen anhand von Sonarbildern, die Erkennung und Klassifizierung von Militärfahrzeugen anhand von Radarscans und vieles mehr. Aber auch neuronale Netze können darauf trainiert werden, gesprochene Sprache und handgeschriebenen Text zu e

Neuronale Netze dienen nicht nur der Erkennung von Objekten und Szenen. Sie werden außerdem darauf trainiert, einfache Roboter mithilfe von Sensordaten zu steuern sowie die Suche in Backgammon- und Schachcomputern heuristisch zu steuern. Besonders interessant in Spielen sind Reinforcement-Learning-Techniken Abschn. 10.8, zum Beispiel mit neuronalen Netzen.

Neuronale Netze werden neben statistischen Methoden seit langem erfolgreich zur Prognose von Aktienmärkten und zur Beurteilung der Bonität von Bankkunden eingesetzt.

Für viele dieser Anwendungen können auch andere Algorithmen des maschinellen Lernens verwendet werden. Allerdings sind neuronale Netze bislang erfolgreicher und beliebter als alle anderen Algorithmen des maschinellen Lernens. Aufgrund des großen kommerziellen Erfolgs von Data Mining und Support-Vektor-Maschinen wird die historische Verbreitung neuronaler Netze jedoch langsam zugunsten anderer Methoden zurückgedrängt.

## 9.8 Zusammenfassung und Ausblick

Mit dem Perzeptron, der Delta-Regel und der Backpropagation haben wir die wichtigste Klasse neuronaler Netze und ihre Beziehung zu Scores und zum naiven Bayes einerseits, aber auch zur Methode der kleinsten Quadrate eingeführt. Als nächstes sahen wir die faszinierenden Hopfield-Netzwerke, inspiriert von biologischen Modellen, die jedoch aufgrund ihrer komplexen Dynamik in der Praxis schwer zu handhaben sind. Von größerer Bedeutung für die Praxis sind die verschiedenen assoziativen Gedächtnismodelle.

In allen neuronalen Modellen werden Informationen über viele Gewichte verteilt gespeichert. Aus diesem Grund hat das Absterben einiger Neuronen im Gehirn keinen spürbaren Einfluss auf die Gehirnfunktion. Durch die verteilte Datenspeicherung ist das Netzwerk robust gegenüber kleinen Störungen. Dies hängt auch mit der Fähigkeit zusammen, Muster mit Fehlern zu erkennen, die in mehreren Beispielen gezeigt werden.

Die verteilte Darstellung von Wissen hat jedoch den Nachteil, dass es für den Wissensingenieur schwierig ist, Informationen zu lokalisieren. Es ist praktisch unmöglich, die vielen Gewichte in einem vollständig trainierten neuronalen Netzwerk zu analysieren und zu verstehen. Bei einem erlernten Entscheidungsbaum hingegen ist es einfach, das erlernte Wissen zu verstehen und sogar als logische Formel darzustellen. Besonders ausdrucksstark und elegant ist die Prädikatenlogik, die die Formalisierung von Beziehungen ermöglicht. Beispielsweise ist das Prädikat „Großmutter“ (karen, clyde) leicht zu verstehen. Auch mit neuronalen Netzen lässt sich ein solcher Zusammenhang erlernen, allerdings ist es nicht einmal möglich, das „Großmutterneuron“ im Netz zu lokalisieren. Daher gibt es immer noch Probleme bei der Verbindung neuronaler Netze mit Symbolverarbeitungssyster

Viele der vielen interessanten neuronalen Modelle können hier nicht behandelt werden. Beispielsweise handelt es sich bei den von Kohonen eingeführten selbstorganisierenden Karten um eine biologisch motivierte Abbildung von einer Sensorschicht von Neuronen auf eine zweite Schicht von Neuronen mit der Eigenschaft, dass diese Abbildung adaptiv ist und die Ähnlichkeit bewahrt.

Ein Problem mit den hier vorgestellten Netzwerken tritt beim inkrementellen Lernen auf. Wenn beispielsweise ein vollständig trainiertes Backpropagation-Netzwerk mit neuen Mustern weiter trainiert wird, geraten viele (und möglicherweise alle) der alten Muster schnell in Vergessenheit. Um dieses Problem zu lösen, entwickelten Carpenter und Grossberg die adaptive Resonanztheorie (ART), die zu einer ganzen Reihe neuronaler Modelle führte.

Als zusätzliche Literatur empfehlen wir die Lehrbücher [Bis05, RMS92, Roj96, Zel94]. Wer sich für die Lektüre des wichtigsten Originalwerks auf diesem spannenden Gebiet interessiert, kann die beiden Sammelbände [AR88, APR90] konsultieren.

## 9.9 Übungen

### 9.9.1 Von der Biologie zur Simulation

**Aufgabe 9.1** Zeigen Sie die Punktsymmetrie der Sigmoidfunktion zu $(\ddot{y}, \frac{1}{2})$.

### 9.9.2 Hopfield-Netzwerke

**Übung 9.2** Verwenden Sie das Hopfield-Netzwerk-Applet unter http://www.cbu.edu/~pong/ai/hopfield/hopfieldapplet.html und testen Sie die Speicherkapazität für korrelierte und unkorrelierte Muster (mit zufällig gesetzten Bits) mit einem 10 × 10 Gittergröße. Verwenden Sie zum Testen ein Muster mit 10 % Rauschen.

**Aufgabe 9.3** Vergleichen Sie den theoretischen Grenzwert N = 0,146 n für die maximale Anzahl speicherbarer Muster aus Abschn. 9.2.2, mit der Kapazität eines klassischen Binärspeichers gleicher Größe.

### 9.9.3 Lineare Netzwerke mit minimalen Fehlern

**ÿ Übung 9.4**

**(a) Schreiben Sie ein Programm zum Erlernen einer linearen Abbildung mit der Methode der kleinsten mittleren Quadrate.** Das ist ganz einfach: Man muss nur die Normalgleichungen nach (9.12) auf Seite 242 aufstellen und dann das Gleichungssystem lösen.

**(a) Wenden Sie dieses Programm auf die Blinddarmentzündungsdaten auf der Website dieses Buches an und ermitteln Sie einen linearen Wert. Geben Sie den Fehler als Funktion des Schwellenwerts**

**an, wie in Abb. 9.14 auf Seite 243. (c) Bestimmen Sie nun die ROC-Kurve für diesen Score.**

## 9.9.4 Backpropagation

**Übung 9.5 Erstellen Sie mit einem Backpropagation-Programm (zum Beispiel in JNNS oder KNIME) ein Netzwerk mit acht Eingabeneuronen, acht Ausgabeneuronen und drei versteckten Neuronen. Trainieren Sie dann das Netzwerk mit acht Trainingsdatenpaaren q, so dass im p-ten pp mit der Requisite , q Vektor q das p-te Bit eins und alle anderen Bits null sind. Das Netzwerk hat somit die einfache Aufgabe, eine identische Abbildung zu lernen. Ein solches Netzwerk wird als 8-3-8-Encoder bezeichnet. Beobachten Sie nach dem Lernprozess die Kodierung der drei verborgenen Neuronen für die Eingabe aller acht gelernten Eingabevektoren. Was fällt dir auf? Wiederholen Sie nun das Experiment und reduzieren Sie die Anzahl der verborgenen Neuronen auf zwei und dann auf eins.**

**Übung 9.6 Um zu zeigen, dass Backpropagation-Netzwerke nicht linear trennbare Mengen teilen können, trainieren Sie die XOR-Funktion. Die Trainingsdaten hierfür sind: ((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 0). (a) Erstellen Sie ein Netzwerk mit jeweils zwei**
**Neuronen in der Eingabeschicht und der verborgenen Schicht, und**
**ein Neuron in der Ausgabeschicht und trainieren dieses Netzwerk.**
**(b) Löschen Sie nun die verborgenen Neuronen und verbinden Sie die Eingabeschicht direkt mit den Ausgabeneuronen. Was beobachten Sie?**

**Aufgabe 9.7 (a)**
**Zeigen Sie, dass jedes mehrschichtige Backpropagation-Netzwerk mit einer linearen Aktivierungsfunktion genauso leistungsfähig ist wie ein zweischichtiges. Dazu reicht es zu zeigen, dass es sich bei aufeinanderfolgenden Ausführungen linearer Abbildungen um eine lineare**
**Abbildung handelt. (b) Zeigen Sie, dass ein zweischichtiges Backpropagation-Netzwerk mit einer streng monotonen Aktivierungsfunktion für Klassifizierungsaufgaben nicht leistungsfähiger ist als eines ohne Aktivierungsfunktion oder mit einer linearen Aktivierungsfunktion.**

## 9.9.5 Support Vector Machines

**ÿ Übung 9.8 Zwei nicht linear trennbare zweidimensionale Sätze von Trainingsdaten M+ 2 2 = 1 und alle + und Mÿ sind gegeben. Alle Punkte in M+ liegen innerhalb des Einheitskreises x x 2 2 2 ÿ R , was ergibt 1 Punkte in Mÿ liegen außerhalb. Geben Sie eine Koordinatentransformation f :** R an , **um die Daten linear trennbar zu machen. Geben Sie die Gleichung der Trennlinie an und skizzieren Sie die beiden Räume und die Datenpunkte.**

# Verstärkungslernen

# 10

## 10.1 Einführung

Alle bisher beschriebenen Lernalgorithmen – mit Ausnahme der Clustering-Algorithmen – gehören zur Klasse des *überwachten Lernens.* Beim überwachten Lernen soll der Agent eine Zuordnung von den Eingabevariablen zu den Ausgabevariablen lernen. Dabei ist es wichtig, dass für jedes einzelne Trainingsbeispiel alle Werte sowohl für Eingabevariablen als auch für Ausgabevariablen bereitgestellt werden. Mit anderen Worten: Wir benötigen einen Lehrer oder eine Datenbank, in der die zu erlernende Abbildung für eine ausreichende Anzahl von Eingabewerten näherungsweise definiert ist. Die einzige Aufgabe des maschinellen Lernalgorithmus besteht darin, das Rauschen aus den Daten herauszufiltern und eine Funktion zu finden, die die Zuordnung auch zwischen den gegebenen Datenpunkten gut annähert.

Beim Reinforcement Learning ist die Situation anders und schwieriger, da keine Trainingsdaten verfügbar sind. Wir beginnen mit einem einfachen Anschauungsbeispiel aus der Robotik, das dann als Anwendung für die verschiedenen Algorithmen dient.

Reinforcement Learning ist im Bereich der Robotik sehr wertvoll, wo die auszuführenden Aufgaben häufig komplex genug sind, um sich der Kodierung als Programme zu entziehen, und wo keine Trainingsdaten verfügbar sind. Die Aufgabe des Roboters besteht darin, durch Versuch und Irrtum (oder Erfolg) herauszufinden, welche Handlungen in einer bestimmten Situation *gut* sind und welche nicht. In vielen Fällen lernen wir Menschen auf ganz ähnliche Weise. Wenn ein Kind beispielsweise laufen lernt, geschieht dies meist ohne Anleitung, sondern einfach durch Verstärkung. Erfolgreiche Gehversuche werden mit Fortschritten belohnt, erfolglose Gehversuche werden mit oft schmerzhaften Stürzen bestraft. Positive und negative Verstärkung sind auch wichtige Faktoren für erfolgreiches Lernen in der Schule und in vielen Sportarten (siehe Abb. 10.1 auf Seite 258).

Im folgenden Beispiel wird eine stark vereinfachte Bewegungsaufgabe erlernt.

*Beispiel 10.1* Ein Roboter, dessen Mechanismus nur aus einem rechteckigen Block und einem Arm mit zwei Gelenken gy und gx besteht , ist in Abb. 10.2 auf Seite 258 dargestellt (siehe [KMK97]). Die einzigen möglichen Aktionen des Roboters sind die Bewegung von gy nach oben oder unten und von gx nach rechts oder links. Darüber hinaus erlauben wir nur Bewegungen fester diskreter Einheiten (z. B. in 10-Grad-Schritten). Die Aufgabe des Agenten besteht darin, eine *Richtlini*

W. Ertel, *Einführung in die künstliche Intelligenz,* Bachelor-Themen in Informatik,
DOI 10.1007/978-0-85729-299-5_10, 

**Abb. 10.1 „Vielleicht sollte ich das nächste Mal etwas früher mit dem Abbiegen beginnen oder langsamer fahren?“ – Lernen aus negativer Verstärkung**

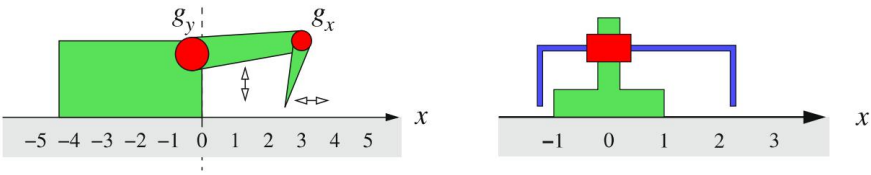


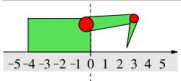
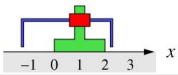
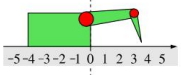
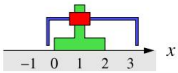
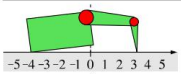
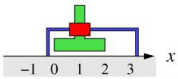
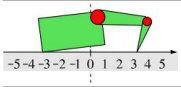
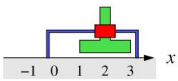

**Abb. 10.2 Durch die Bewegung seiner beiden Gelenke kann sich der Krabbelroboter links *im* Bild vorwärts und rückwärts bewegen. Der Laufroboter auf der *rechten Seite* muss den Rahmen entsprechend *nach oben* und *unten* bzw. *nach links* und *rechts bewegen.* Das Feedback für die Bewegung des Roboters ist bei einer Bewegung nach rechts positiv *und* bei einer Bewegung nach *links* negativ**

**Dadurch kann es sich möglichst schnell nach rechts bewegen. Der Laufroboter in Abb. 10.2 arbeitet analog innerhalb desselben zweidimensionalen Zustandsraums.**

**Eine erfolgreiche Aktionssequenz ist in Tabelle 10.1 auf Seite 259 dargestellt . Die Aktion zum Zeitpunkt t = 2 führt dazu, dass der belastete Arm den Körper um eine Längeneinheit nach rechts bewegt. Schöne Animationen dieses Beispiels finden Sie in [KMK97] und [Tok06].**

**Bevor wir auf den Lernalgorithmus eingehen, müssen wir die Aufgabe mathematisch passend modellieren. Wir beschreiben den *Zustand* des Roboters durch die beiden Variablen gx und gy für die Position der Gelenke mit jeweils endlich vielen diskreten Werten. Der Zustand des Roboters wird somit als Vektor (gx, gy) kodiert. Die Anzahl der möglichen** oder **Gelenkpositionen beträgt nx bzw. ny . Zur Bewertung der Aktionen des Roboters nutzen wir die horizontale Position des Roboterkörpers, die reale Werte annehmen kann. Bewegungen nach rechts werde**

Tabelle 10.1 Ein Zyklus einer periodischen Reihe von Bewegungen mit systematischer Vorwärtsbewegung

| Kriechender Roboter | Laufender Roboter | Zeit | Zustand | | Belohnungsaktion | |
|---|---|---|---|---|---|---|
| | | T | gy | gx | X | bei |
|  |  | 0 | Hoch | Links | 0 | Rechts |
|  |  | 1 | Hoch | Rechts | 0 | Runter |
|  |  | 2 | Unten rechts | | 0 | Links |
|  |  | 3 | Unten links1 | | | Hoch |

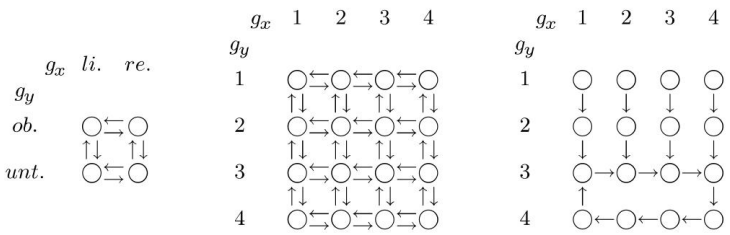



Abb. 10.3 Der Zustandsraum des Beispielroboters bei jeweils zwei möglichen Positionen die Gelenke *(links)* und bei jeweils vier horizontalen und vertikalen Positionen *(Mitte).* Im *Mit dem richtigen Bild* ist eine optimale Politik gegeben

mit positiven Änderungen an x und Bewegungen nach links werden mit negativen bestraft Änderungen.

In Abb. 10.3 ist der Zustandsraum für zwei Varianten davon vereinfacht dargestellt.1 Im linken Beispiel haben beide Gelenke zwei Positionen, im mittleren Beispiel beide jeder hat vier Positionen. Eine optimale Richtlinie ist in Abb. 10.3 rechts dargestellt.

## 10.2 Die Aufgabe

Wie in Abb. 10.4 auf Seite 260 dargestellt, unterscheiden wir zwischen dem Agenten und seiner Umgebung. Zum Zeitpunkt t ist die Welt, die den Agenten und seine Umgebung umfasst, vorhanden beschrieben durch einen *Zustand* st ÿ S. Die Menge S ist eine Abstraktion der tatsächlich möglichen Zustände der Welt, weil einerseits die Welt nicht genau beschrieben werden kann, und andererseits Andererseits verfügt der Agent oft nur über unvollständige Informationen über die tatsächliche Situation

1Der aus Bögen bestehende Armbewegungsraum wird als rechtwinkliges Gitter dargestellt.

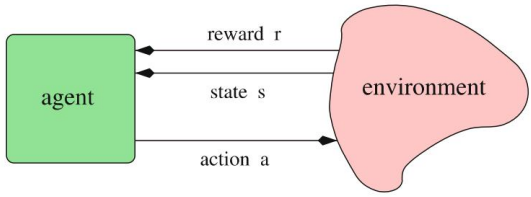



**Abb. 10.4 Der Agent und seine Interaktion mit der Umgebung**

**Zustand aufgrund von Messfehlern.** Der Agent führt dann **zum Zeitpunkt t** eine ***Aktion*** **bei ÿ** A **aus . Diese Aktion verändert die Welt und führt somit zum Zustand st+1 zum Zeitpunkt t + 1. Die** durch die Umgebung definierte ***Zustandsübergangsfunktion*** ÿ **bestimmt den neuen Zustand st+1 =** ÿ(st, at). **Es kann vom Agenten nicht beeinflusst werden.**

**Nach Ausführung der Aktion** at **erhält der Agent eine sofortige Belohnung** rt **=** r(st, at) **(siehe Abb. 10.4). Die unmittelbare Belohnung** rt **=** r(st, at) **ist immer abhängig vom aktuellen Zustand und der ausgeführten Aktion.** r(st, at) = 0 bedeutet, dass der Agent zum Zeitpunkt **keine unmittelbare Rückmeldung für die Aktion erhält . Während des Lernens** sollte rt > **0 zu einer positiven und** rt < **0 zu einer negativen Verstärkung der Bewertung der Aktion** im **Zustand** st führen . **Insbesondere beim Reinforcement Learning werden Anwendungen untersucht, bei denen es über einen längeren Zeitraum zu keiner unmittelbaren Belohnung kommt. Ein Schachspieler beispielsweise lernt, sein Spiel durch gewonnene oder verlorene Partien zu verbessern, auch wenn er in einzelnen Zügen keine sofortige Belohnung für alle erhält. Hier zeigt sich die Schwierigkeit, die Belohnung am Ende einer Handlungsfolge allen Handlungen in der Folge zuzuordnen, die zu diesem Punkt geführt haben (Credit-Assignment-Problem).**

**Beim kriechenden Roboter besteht der Zustand aus der Stellung der beiden Gelenke, das heißt,** s **=** (gx, gy ). **Die Belohnung ergibt sich aus der zurückgelegten Strecke** x .

**Eine *Richtlinie*** ÿ : S **ÿ** A **ist eine Abbildung von Zuständen auf Aktionen. Das Ziel des Verstärkungslernens besteht darin, dass der Agent auf der Grundlage seiner Erfahrungen eine optimale Richtlinie Eine Richtlinie ist optimal, wenn sie langfristig, also über viele Schritte hinweg, den Nutzen maximiert. Aber was bedeutet „Belohnung maximieren“ genau? Wir definieren den *Wert* bzw. die *ermäßigte Belohnung***

$$V^{\ddot{y}}(st) = rt + \ddot{y}\, rt+1 + \ddot{y}^{2}\, rt+2 + \cdots = \ddot{y}^{i}\, rt+i \qquad (10.1)$$

**einer Richtlinie** ÿ , **wenn wir im Startzustand** st beginnen**. Dabei ist 0 ÿ** ÿ < **1 eine Konstante, die sicherstellt, dass zukünftiges Feedback umso mehr abgezinst wird, je weiter es in der Zukunft liegt. Die unmittelbare Belohnung** rt **wird am stärksten gewichtet. Diese Belohnungsfunktion wird am häufigsten verwendet. Eine manchmal interessante Alternative ist die durchschnittliche Belohnung**

$$V^{\ddot{y}}(st) = \lim_{h \ddot{y} \ddot{y}} \frac{1}{h} \sum_{i=0}^{H} rt+i\ . \qquad (10.2)$$

***Eine Politik*** ÿ ÿ ***heißt optimal,*** **wenn für alle Zustände** s

$$V^{\ddot{y}\,\ddot{y}}(s) \ddot{y}\, V^{\ddot{y}}(S). \qquad (10.3)$$

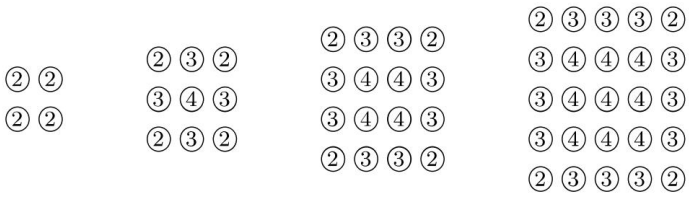


**Abb. 10.5 Der Zustandsraum für das Beispiel mit den Werten 2, 3, 4, 5 für nx und ny. Die Anzahl der möglichen Aktionen ist für jedes Bundesland in den jeweiligen *Kreisen* angegeben**

**Tabelle 10.2 Anzahl der Richtlinien für unterschiedlich große Zustandsräume im Beispiel**

| nx , ny | Anzahl der Staaten | Anzahl der Policen |
|---|---|---|
| 24 24 = 16 | | |
| 39 243 44 = 5184 | | |
| 4 | 16 | 243 84 4 ÿ 2,7 × 107 |
| 5 | 25 | 243 124 9 ÿ 2,2 × 1012 |

**Das heißt, sie ist hinsichtlich des definierten Wertes mindestens so gut wie alle anderen Policen Funktion. Zur besseren Lesbarkeit ist die optimale Wertfunktion V wird mit** V bezeichnet .

**Die hier besprochenen Agenten bzw. deren Richtlinien verwenden nur Informationen über den aktuellen Zustand,** um **den nächsten Zustand zu bestimmen, und nicht die Vorgeschichte. Dies ist gerechtfertigt, wenn die Belohnung einer Aktion nur vom aktuellen Zustand und den aktuellen Aktionen** Solche Prozesse werden ***Markov-Entscheidungsprozesse*** (MDP) **genannt . In vielen Anwendungen, insbesondere in der Robotik, ist der tatsächliche Zustand des Agenten nicht genau bekannt, was die Planung von Aktionen zusätzlich erschwert. Der Grund dafür kann ein verrauschtes Sensorsignal sein. Wir nennen einen solchen Prozess einen *teilweise beobachtbaren Markov-Entscheidungsprozess (POMDP).***

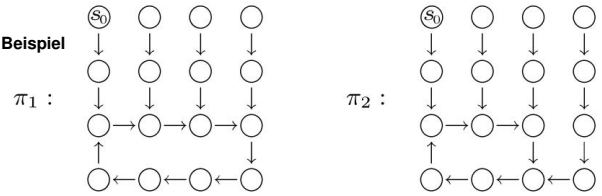
## 10.3 Uninformierte kombinatorische Suche

**Die einfachste Möglichkeit, eine erfolgreiche Politik zu finden, ist die kombinatorische Aufzählung aller Politiken, wie in Kap. 6. Allerdings gibt es bereits im einfachen Beispiel 10.1 sehr viele Richtlinien, was dazu führt, dass die kombinatorische Suche mit einem extrem hohen Rechenaufwand verbunden ist. In Abb. 10.5 ist für jeden Zustand die Anzahl der möglichen Aktionen angegeben. Daraus wird die Anzahl der möglichen Richtlinien als Produkt der angegebenen Werte berechnet, wie in Tabelle 10.2 dargestellt.**

**Für beliebige Werte von** nx **und** ny **gibt es immer vier Eckknoten mit zwei möglichen Aktionen,** 2(nx **ÿ** 2) **+** 2(ny **ÿ** 2) **Randknoten mit drei Aktionen und** (nx **ÿ** 2)(ny **ÿ** 2) **innere Knoten mit vier Aktionen. So gibt es**

**4 2 3 2(nxÿ2)+2(nyÿ2) 4** (nxÿ2)(nyÿ2)

Abb. 10.6 Zwei unterschiedliche Richtlinien für das Beispiel




Unterschiedliche Richtlinien für feste nx und ny .Die Zahl der Policen wächst also exponentiell mit der Zahl der Staaten. Dies gilt im Allgemeinen, wenn es mehr als eine mögliche Aktion pro Zustand gibt. Für praktische Anwendungen ist dieser Algorithmus daher unbrauchbar. Sogar die heuristische Suche, beschrieben in Kap. 6, kann hier nicht verwendet werden. Da die direkte Belohnung für fast alle Aktionen Null ist, kann sie nicht als heuristische Bewertungsfunktion verwendet werden.

Der Rechenaufwand steigt sogar noch höher, wenn man bedenkt, dass (zusätzlich zu (s) für jedes Aufzählung aller Richtlinien), der Wert V ÿ generierte (s) berechnet werden muss , für das Politik ÿ und jeder Startzustand s. Die unendliche Summe in V ÿ abgeschnitten werden muss Verwendung in einer praktischen Berechnung; Aufgrund der exponentiellen Reduzierung der ÿ- Faktoren in (10.1) auf Seite 260 stellt dies jedoch kein Problem dar.

In Beispiel 10.1 auf Seite 257 kann die Differenz xt+1 ÿ xt als unmittelbare Belohnung für eine Aktion bei verwendet werden, was bedeutet, dass jede Bewegung des Roboterkörpers nach rechts mit 1 und jede Bewegung des Roboterkörpers nach rechts mit 1 belohnt wird die Linke wird mit ÿ1 bestraft. In Abb. 10.6 sind zwei Richtlinien dargestellt. Hier ist die unmittelbare Belohnung überall außer in der unteren Reihe des Zustandsraums Null. Die linke Politik ÿ1 ist langfristig besser, da bei langen Aktionssequenzen der durchschnittliche Fortschritt pro Aktion 3/8 = 0,375 für ÿ1 und 2/6 ÿ 0,333 für ÿ2 beträgt. Wenn wir (10.1) (s) verwenden, ist das Ergebnis die folgende Tabelle mit dem auf Seite 260 für V ÿ Startzustand s0 am oben links und verschiedene ÿ- Werte:

| ÿ | 0,9 | 0,8375 | 0,8 |
|---|---|---|---|
| V ÿ1 (s0) | 2,52 | 1,156 | V ÿ2 (s0) 0,77 |
| 2,39 | 1,156 | | 0,80 |

Hier sehen wir, dass die Richtlinie ÿ1 der Richtlinie ÿ2 überlegen ist, wenn Gamma = 0,9, und das Gegenteil gilt, wenn Gamma = 0,8. Für ÿ ÿ 0,8375 sind beide Richtlinien gleich gut.
Wir können deutlich erkennen, dass ein größeres ÿ zu einem größeren Zeithorizont für die Bewertung von Richtlinien führt.

## 10.4 Wertiteration und dynamische Programmierung

Bei dem naiven Ansatz, alle Richtlinien aufzuzählen, wird viel redundante Arbeit geleistet, da viele Richtlinien größtenteils identisch sind. Sie dürfen sich nur geringfügig unterscheiden. Dennoch wird jede Police komplett neu erstellt und bewertet.
Dies legt nahe, Zwischenergebnisse für Teile von Richtlinien zu speichern und wiederzuverwenden.

**Dieser Ansatz zur Lösung von Optimierungsproblemen wurde bereits 1957 von Richard Bellman** als ***dynamische Programmierung*** eingeführt [Bel57]. **Bellman erkannte das Für eine optimale Politik gilt:**

> **Unabhängig vom Startzustand st und der ersten Aktion bei** müssen **alle Folgeentscheidungen ausgehend von jedem möglichen Nachfolgezustand st+1 optimal sein.**

**Basierend auf dem sogenannten Bellman-Prinzip wird es möglich, eine globale Lösung zu finden optimale Politik durch lokale Optimierung einzelner Maßnahmen. Wir werden dies ableiten Prinzip für MDPs zusammen mit einem geeigneten Iterationsalgorithmus.**

**Gewünscht ist eine optimale Politik** ÿ was **(10.3) auf Seite 260 und (10.1) auf** Seite 260 **erfüllt Seite 260. Wir schreiben die beiden Gleichungen um und erhalten**

$$V^{\ddot{y}}(st) = \max_{bei,bei+1,bei+2,\ldots} (r(st, at) + \ddot{y}\, r(st+1, \mathbf{at+1}) + \ddot{y}^{2}\, r(st+2, \mathbf{at+2}) + \cdots). \quad \textbf{(10.4)}$$

**Da die unmittelbare Belohnung** r(st, at) **nur von** st **und** at abhängt , **nicht aber von der Nachfolgezustände und -aktionen können maximiert werden, was letztendlich der Fall ist ergibt die folgende rekursive Charakterisierung von** $V^{\ddot{y}}$ **:**

$$V^{\ddot{y}}(st) = \max_{bei} [r(st, at) + \ddot{y} \max_{bei+1,bei+2,\ldots} (r(st+1, \mathbf{at+1}) + \ddot{y}\, r(st+2, \mathbf{at+2}) + \cdots)]$$

$$= \max_{bei} [r(st, at) + \ddot{y}\, V^{\ddot{y}}(\mathbf{St+1})]. \quad \textbf{(10.5)}$$

**Gleichung (10.5) ergibt sich aus der Substitution** t **ÿ** t **+ 1 in (10.4). Etwas geschrieben einfacher:**

$$V^{\ddot{y}}(s) = \max_{A} [r(s, a) + \ddot{y}\, V^{\ddot{y}}(\ddot{y}(s, a))]. \quad \textbf{(10.6)}$$

**Diese Gleichung impliziert, ebenso wie (10.1) auf Seite 260, dass zur Berechnung von** $V^{\ddot{y}}(s)$ **das im Die mittlere Belohnung wird zur Belohnung aller Nachfolgestaaten addiert, abgezinst von der Faktor** ÿ . **Wenn** $V^{\ddot{y}}(\ddot{y}(s, a))$ **bekannt ist, dann** $V^{\ddot{y}}(s)$ **ergibt sich eindeutig aus einfachen lokalen Operationen Timing über alle möglichen Aktionen** a **im Zustand s. Dies entspricht dem Bellman Prinzip, weshalb (10.6) auch Bellman-Gleichung genannt wird.**

**Die optimale Richtlinie** $\ddot{y}^{\ddot{y}}(s)$ führt eine Aktion im Zustand s **aus , die zur Folge hat Maximalwert** $V^{\ddot{y}}$ **. Daher,**

$$\ddot{y}^{\ddot{y}}(s) = \operatorname*{argmax}_{A} [r(s, a) + \ddot{y}\, V^{\ddot{y}}(\ddot{y}(s, a))]. \quad \textbf{(10.7)}$$

**Aus der Rekursionsgleichung (10.6) eine Iterationsregel zur Approximation von** $V^{\ddot{y}}$ **: folgt auf einfache Weise:**

$$\hat{V}(s) = \max_{A} [r(s, a) + \ddot{y}\, \hat{V}(\ddot{y}(s, a))]. \quad \textbf{(10.8)}$$

**Zu Beginn werden beispielsweise die Näherungswerte** $\hat{V}(s)$ **für alle Zustände initialisiert mit dem Wert Null. Nun wird** $\hat{V}(s)$ **für jeden Zustand wiederholt rekursiv aktualisiert Zurückgreifen auf den Wert** $\hat{V}(\ddot{y}(s, a))$ **des besten Nachfolgezustandes. Dieser Prozess von Berechnung von** $\hat{V}$ **wird *Werteiteration* genannt und ist in Abb. 10.7** schematisch dargestellt

**Abb. 10.7 Der Algorithmus für Wertiteration**

**BEWERTUNG()**

**Für alle s ÿ** S

**V(s) = 0**

**Wiederholen**

**Für alle s ÿ** S

**V (s) = maxa[r(s, a) +** ÿ V (ÿ(s, a)) ]

**Bis sich** V(s) **nicht ändert**

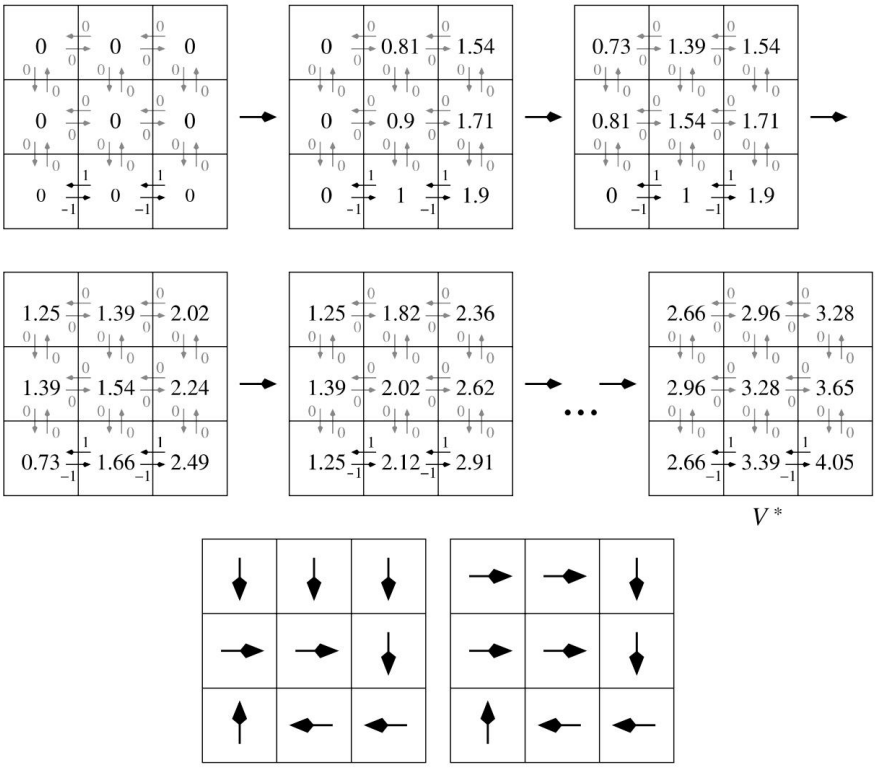



**Abb. 10.8 Werteiteration im Beispiel mit 3 × 3 Zuständen. Die *letzten beiden Bilder* zeigen zwei optimale Richtlinien. Die Zahlen neben den *Pfeilen* geben die unmittelbare Belohnung** r(s, a) **jeder Aktion** an

**auf Seite 264. Es kann gezeigt werden, dass die Werteiteration gegen** V konvergiert **[SB98]. Ein Eine hervorragende Analyse dynamischer Programmieralgorithmen finden Sie in [Sze10], wobei basierend auf den Kontraktionseigenschaften der jeweiligen Algorithmen (z. B. Werteiteration) die Konvergenz mithilfe der Fixpunkttheorie von Banach nachgewiesen werden kann rem.**

In Abb. 10.8 wird dieser Algorithmus auf Beispiel 10.1 auf Seite 257 mit ÿ = 0,9 angewendet . In jeder Iteration werden die Zustände zeilenweise von links unten nach rechts oben abgearbeitet. Dargestellt sind mehrere Anfangsiterationen und im zweiten Bild in der unteren Reihe die stabilen Grenzwerte für V ÿ .

In dieser Reihenfolge sehen wir deutlich den Fortschritt des Lernens. Der Agent durchsucht wiederholt alle Zustände, führt für jeden Zustand eine Werteiteration durch und speichert die in Form einer tabellierten Funktion V ÿ, Richtlinie, die dann weiter in eine Richtlinie kompiliert werden ka effizient nutzbare Tabelle ÿ ÿ .

Um übrigens eine optimale Police von V zu finden ÿ Es wäre falsch, die Aktion im Zustand st zu wählen , die zu dem Zustand mit dem maximalen V gemäß (10.7) auf ÿ Wert. Korre Seite 263 führt. Die unmittelbare Belohnung r(st, at) muss ebenfalls hinzugefügt werden, da wir in Abb. nach V suchen . 10.8 auf Seite ÿ (st) und nicht V ÿ (St+1). Angewendet auf den Zustand s = (2, 3) 264, das heißt

$$\ddot{y}^{\ddot{y}}(2, 3) = \text{argmax}\,[r(s, a) + \ddot{y}\, V^{\ddot{y}}(\ddot{y}(s, a))]$$
$$\text{a}\ddot{y}\{\text{links, rechts, oben}\}$$

$$\{\text{links, rechts, oben}\} \quad \{1 + 0{,}9 \cdot 2{,}66, \ddot{y}1 + 0{,}9 \cdot 4{,}05, 0 + 0{,}9 \cdot 3{,}28\} = \text{argmax}$$

$$\text{argmax} \quad \{3{,}39, 2{,}65, 2{,}95\} = \text{links.} = \{\text{links, rechts, oben}\}$$

In (10.7) auf Seite 263 sehen wir, dass der Agent im Zustand st die unmittelbare Belohnung rt und den Nachfolgezustand st+1 = ÿ(st, at) kennen muss , um die optimale Aktion bei auszuwählen . Es muss auch ein Modell der Funktionen r und ÿ vorhanden sein. Da dies für viele praktische Anwendungen nicht der Fall ist, werden Algorithmen benötigt, die auch ohne Kenntnis von r und ÿ funktionieren können. Abschnitt 10.6 ist einem solchen Algorithmus gewidmet.

## 10.5 Ein lernender Laufroboter und seine Simulation

Eine grafische Benutzeroberfläche für einfache Experimente mit Reinforcement Learning ist in Abb. 10.9 auf Seite 266 dargestellt [TEF09]. Der Benutzer kann verstärkendes Lernen für unterschiedlich große zweidimensionale Zustandsräume beobachten. Zur besseren Verallgemeinerung werden Backpropagation-Netzwerke zur Zustandsspeicherung eingesetzt (siehe Abschn. 10.8). Besonders interessant für Experimente ist der unten rechts dargestellte Feedback-Editor, mit dem der Nutzer manuell Rückmeldungen über die Umgebung abgeben kann. Nicht gezeigt ist das Menü zum Einrichten der Parameter für die Werteiteration und das Backpropagation-Lernen.

Neben der Simulation wurden zwei kleine, echte Kriechroboter mit demselben zweidimensionalen diskreten Zustandsraum speziell für den Unterricht entwickelt [TEF09].2 Die beiden Roboter sind in Abb. 10.10 auf Seite 266 dargestellt . Jeder bewegt sich mit einem Servoantrieb

2Weitere Informationen und verwandte Quellen zu Crawling-Robotern finden Sie unter www. hs-weingarten.de/~ertel/kibuch.

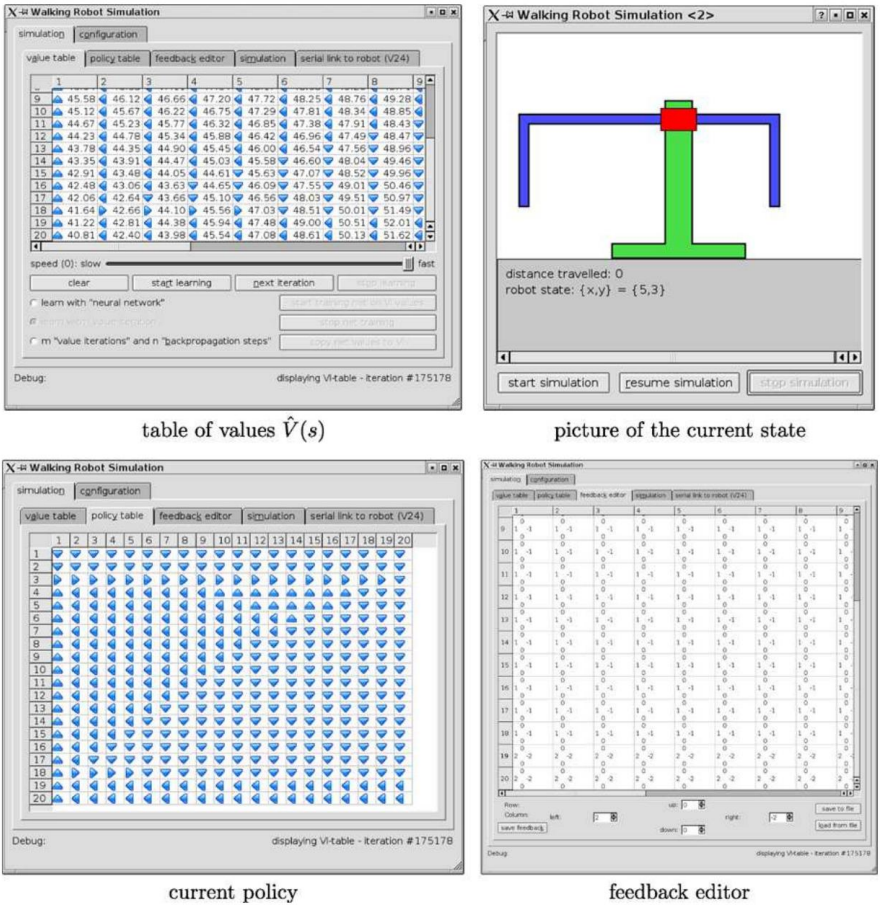



**Abb. 10.9 Vier verschiedene Fenster des Laufrobotersimulators**

**Abb. 10.10 Zwei Versionen des Krabbelroboters**

ator. Die Servos werden über einen Mikrocontroller oder über eine drahtlose Schnittstelle direkt von einem PC aus gesteuert. Mithilfe einer Simulationssoftware kann die Feedback-Matrix des Roboters auf dem PC visualisiert werden. Mit diesem gespeicherten Feedback kann eine Richtlinie auf dem PC trainiert werden (der schneller rechnet), dann wieder in den Roboter geladen und ausgeführt werden. Der Roboter kann jedoch auch autonom lernen. Bei einem Zustandsraum der Größe 5×5 dauert dies etwa 30 Sekunden.

Es ist interessant, den Unterschied zwischen der Simulation und dem „echten“ Roboter zu beobachten. Im Gegensatz zur Simulation erlernt der Crawler Verhaltensweisen, bei denen er seinen Arm nie vom Boden hebt, sich aber dennoch sehr effizient vorwärtsbewegt. Der Grund dafür liegt darin, dass die Spitze des „Unterarms“ je nach Untergrundoberfläche bei der Rückwärtsbewegung den Boden festhalten kann, bei der Vorwärtsbewegung jedoch durchrutscht. Dieser Effekt wird durch die Abstandsmesssensoren sehr gut wahrgenommen und beim Lernen entsprechend ausgewertet.

Die Adaptivität des Roboters führt zu überraschenden Effekten. Wir können beispielsweise beobachten, wie die Raupe trotz eines defekten Servos, das in einem bestimmten Winkel durchrutscht, dennoch das Laufen lernt (eher humpelt). Es ist sogar in der Lage, sich durch geänderte Richtlinien an veränderte Situationen anzupassen. Ein durchaus wünschenswerter Effekt ist die Fähigkeit, bei unterschiedlich glatten Oberflächen (z. B. unterschiedlich rauen Teppichen) jeweils eine optimale Vorgehensweise zu erlernen. Es zeigt sich auch, dass der reale Roboter angesichts eines kleinen Zustandsraums der Größe 5 × 5 tatsächlich sehr anpassungsfähig ist.

Der Leser kann (in Ermangelung eines echten Roboters) verschiedene Oberflächen oder Servodefekte modellieren, indem er die Rückkopplungswerte variiert und dann die daraus resultierenden Richtlinien beachtet (Übung 10.3 auf Seite 276).

## 10.6 Q-Learning

Eine Politik, die auf der Bewertung möglicher Nachfolgezustände basiert, ist eindeutig nicht anwendbar, wenn der Agent kein Modell der Welt hat, das heißt, wenn er nicht weiß, zu welchem Zustand eine mögliche Aktion führt. In den meisten realistischen Anwendungen kann der Agent nicht auf ein solches Weltmodell zurückgreifen. Beispielsweise kann ein Roboter, der komplexe Objekte greifen soll, nicht vorhersagen, ob das Objekt nach einem Greifvorgang sicher im Griff gehalten wird oder ob es an Ort und Stelle bleibt.

Liegt kein Modell der Welt vor, ist eine Bewertung einer im Zustand st durchgeführten Aktion erforderlich, auch wenn noch unbekannt ist, wohin diese Aktion führt. Daher arbeiten wir nun mit einer Bewertungsfunktion Q(st, at) für Zustände mit ihren zugehörigen Aktionen. Bei dieser Funktion erfolgt die Wahl der optimalen Aktion durch die Regel

$$\ddot{y}^{\ddot{y}}(s) = \underset{A}{\mathbf{argmax}}\, Q(s, a). \qquad (10.9)$$

Um die Bewertungsfunktion zu definieren, verwenden wir wiederum eine schrittweise Diskontierung der Bewertung für Zustands-Aktionspaare, die weiter in der Zukunft liegen, genau wie in (10.1) auf Seite 260. Wir wollen also $r_t + \ddot{y} r_{t+1} + \ddot{y}^2 r_{t+2} + \cdots$ maximieren. Um die Aktion im Zustand st zu bewerten, definieren wir daher in Analogie zu (10.4) on

**Seite 263:**

$$Q(st, at) = \max_{bei+1,bei+2,\ldots} (r(st, at)+ÿ\, r(st+1, at+1)+ÿ^2\, r(st+2, at+2)+\cdots). \quad (10.10)$$

**Analog zum Ansatz zur Werteiteration bringen wir diese Gleichung durch in eine einfache rekursive Form**

$$\begin{aligned} Q(st, at) &= \max_{bei+1,bei+2,\ldots} (r(st, at) + ÿ\, r(st+1, at+1) + ÿ^2\, r(st+2, at+2) +\cdots) \\ &= r(st, at) + ÿ \max_{bei+1,bei+2,\ldots} (r(st+1, at+1) + ÿ\, r(st+2, at+2) +\cdots) \\ &= r(st, at) + ÿ \max_{bei+1} (r(st+1, at+1) + ÿ \max_{bei+2} (r(st+2, at+2) +\cdots)) \\ &= r(st, at) + ÿ \max_{bei+1} Q(st+1, at+1) \\ &= r(st, at) + ÿ \max_{bei+1} Q(ÿ(st, at), at+1) \\ &= r(s, a) + ÿ \max_{a'} Q(ÿ(s, a), aÿ). \end{aligned} \quad (10.11)$$

**Was ist dann der Vorteil gegenüber der Wertiteration? Die alte Gleichung wird nur leicht umgeschrieben, aber das erweist sich als genau der richtige Ansatz für einen neuen Algorithmus. Anstatt** V zu speichern, wird **nun die Funktion** Q **gespeichert, und der Agent kann seine Aktionen aus den Funktionen** ÿ **und** r auswählen , **ohne ein Modell der Welt zu h**

**Allerdings gibt es noch keinen Prozess, der** Q **direkt, also ohne Kenntnis von** V , lernen kann . **Aus der**

**rekursiven Formulierung von** Q(s, a) lässt sich **ein Iterationsalgorithmus zur Bestimmung** von Q(s, a) **in a ableiten unkomplizierte Art und Weise. Wir initialisieren eine Tabelle** Q(s, a) **für alle Zustände beliebig, beispielsweise mit Nullen, und führen sie iterativ aus**

$$\hat{Q}(s, a) = r(s, a) + ÿ \max_{a'} \hat{Q}(ÿ(s, a), a'). \quad (10.12)$$

Es bleibt zu beachten, dass wir die Funktionen r **und** ÿ **nicht kennen . Wir lösen dieses Problem ganz pragmatisch, indem wir den Agenten in seiner Umgebung im Zustand** s **die Aktion a ausführen** lassen . **Der Nachfolgezustand ist dann eindeutig** ÿ(s, a) **und der Agent erhält seine Belohnung von der Umgebung. Der in Abb. 10.11 auf Seite 269** dargestellte Algorithmus **implementiert diesen Algorithmus für Q-Learning.**

**Die Anwendung des Algorithmus auf Beispiel 10.1 auf Seite 257 mit** ÿ = 0,9 **und** nx = 3, ny **= 2 (also in einem 2 × 3-Raster) ist in Abb. 10.12 auf Seite 269 beispielhaft dargestellt. Im ersten Bild werden alle** Q- **Werte auf Null initialisiert. Im zweiten Bild werden nach der ersten Aktionssequenz die vier** r- **Werte, die ungleich Null sind, als** Q- **Werte sichtbar. Im letzten Bild ist die erlernte optimale Richtlinie dargestellt. Der folgende Satz, dessen Beweis in [Mit97] zu finden ist, zeigt, dass dieser Algorithmus nicht nur im Beispiel, sondern allgemein konvergiert.**

**Q-LEARNING()**

**Für alle s ÿ S, a ÿ** A

**Q(s, a) = 0 (oder zufällig)**

**Wiederholen**

**Wählen Sie (z. B. zufällig) einen Status s aus.**

**Wiederholen**

**Wählen Sie eine Aktion a aus und führen Sie sie aus. Erhalten Sie eine Belohnung r und einen neuen Status s**

**Q(s, a) := r(s, a) + ÿ maxa ÿ** Q(s , aÿ )

' s := s

**Bis** s **ein Endzustand oder ein Zeitlimit erreicht ist**

**Bis Qˆ konvergiert**

**Abb. 10.11 Der Algorithmus für Q-Learning**

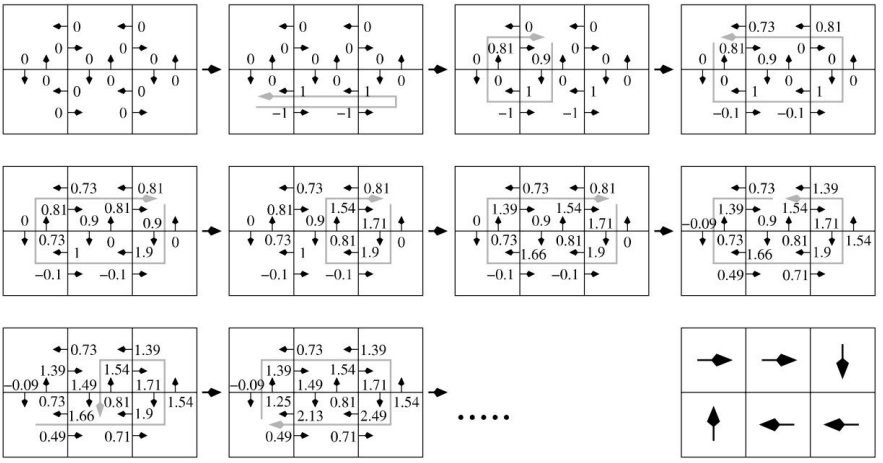


**Abb. 10.12 Q-Learning angewendet auf das Beispiel mit** nx **= 3,** ny **= 2. Die *grauen Pfeile* markieren die durchgeführten Aktionen in jedem Bild. Die aktualisierten** Q- **Werte werden angezeigt. Im *letzten Bild* ist die aktuelle, ebenfalls optimale Politik dargestellt**

**Satz 10.1** ***Es sei ein deterministischer MDP mit begrenzter unmittelbarer Belohnung*** r(s, a) ***gegeben.*** Zum Lernen wird ***Gleichung*** (10.12) ***auf Seite*** 268 ***mit*** **0 ÿ** ÿ < **1** ***verwendet. Sei*** **Qˆ** n(s, a) ***der Wert für*** Q(s, a) ***nach*** n ***Aktualisierungen. Wenn jedes Zustandsaktionspaar unendlich oft besucht wird, dann konvergiert*** **Qˆ** n(s, a) ***für alle Werte*** s ***und*** a ***für*** n ÿ ÿ gegen Q(s, a) .

***Beweis*** **Da jeder Zustands-Aktions-Übergang unendlich oft auftritt, betrachten wir aufeinanderfolgende Zeitintervalle mit der Eigenschaft, dass in jedem Intervall alle Zustands-Aktions-Übergänge mindestens einmal auftreten.** Wir zeigen nun, dass der maximale Fehler für alle Einträge in der Qˆ-Tabelle **in jedem dieser Intervalle mindestens um den Faktor** ÿ reduziert wird. **Lassen**

$$_{N} = \max_{s,a} |Q\hat{}\ n(s, a) \ddot{y} Q(s, a)|$$

**sei der maximale Fehler in der Tabelle Q**ˆ n **und** s ' **=** ÿ(s, a). **Für jeden Tabelleneintrag Q**ˆ n(s, a) **Wir berechnen seinen Beitrag zum Fehler nach einem Intervall als**

$$|Q\hat{}\ n+1(s, a) \ddot{y} Q(s, a)| = r + \ddot{y} \max_{ein'} Q\hat{}\ n(s\ddot{y}, a\ddot{y}) \ddot{y} r + \ddot{y} \max_{ein'} Q(s\ddot{y}, a\ddot{y})$$

$$= \ddot{y} \max_{A'} Q\hat{}\ n(s\ddot{y}, a\ddot{y}) \ddot{y} \max_{A'} Q(s\ddot{y}, a\ddot{y})$$

$$\ddot{y} \ddot{y} \max_{ein'} |Q\hat{}\ n(s\ddot{y}, a\ddot{y}) \ddot{y} Q(s\ddot{y}, a\ddot{y})|$$

$$\ddot{y} \ddot{y} \max_{„aÿ} s |Q\hat{}\ n(s\ddot{y}\ddot{y}, a\ddot{y}) \ddot{y} Q(s\ddot{y}\ddot{y}, a\ddot{y})| = \ddot{y} n.$$

**Die erste Ungleichung ist wahr, weil für beliebige Funktionen** f **und g**

$$\max_{x} f(x) \ddot{y} \max_{x} g(x) \ddot{y} \max_{x} |f(x) \ddot{y} g(x)|$$

**und die zweite Ungleichung ist wahr, weil durch zusätzliche Variation des Zustands s** das **resultierende Maximum nicht kleiner werden kann. Somit wurde gezeigt, dass n+1 ÿ Da der Fehler** ÿ **n. in jedem Intervall mindestens um den Faktor** ÿ reduziert wird , **beträgt er nach** k **Intervallen höchstens** ÿ **k0** und **ist daher** auf **0** beschränkt. **Da jeder Staat unendlich oft besucht wird, gibt es unendlich viele Intervalle und null.** n **konvergiert**

**Gemäß Satz 10.1 auf Seite 269 konvergiert Q-Learning unabhängig von den beim Lernen gewählten Aktionen. Das bedeutet, dass es für die Konvergenz keine Rolle spielt, welche Aktionen der Agent auswählt, solange jede davon unendlich oft ausgeführt wird.**
**Die Konvergenzgeschwindigkeit hängt jedoch sicherlich davon ab, welche Wege der Agent beim Lernen einschlägt (siehe Abschn. 10.7).**

## 10.6.1 Q-Learning in einer nichtdeterministischen Umgebung

**In vielen Robotikanwendungen ist die Umgebung des Agenten nichtdeterministisch. Dies bedeutet, dass die Reaktion der Umwelt auf die Aktion** a **in Zuständen** zu **zwei unterschiedlichen Zeitpunkten zu unterschiedlichen Folgezuständen und Belohnungen führen kann. Ein solcher nichtdeterministischer Markov-Prozess wird durch eine probabilistische Übergangsfunktion** ÿ(s, a) **und eine probabilistische unmittelbare Belohnung** r(s, a) modelliert. **Um die** Q- **Funktion zu definieren, muss jeweils der Erwartungswert über alle möglichen Folgezustände berechnet werden. Gleichung (10.**

auf Seite 268 wird somit verallgemeinert

$$Q(st, at) = E(r(s, a)) + \ddot{y} \sum_{s'} P(s\ddot{y} | s, a) \max_{ein'} Q(s\ddot{y}, a\ddot{y}), \quad (10.13)$$

Dabei ist P (sÿ |s, a) die Wahrscheinlichkeit, mit der Aktion a vom Zustand s zum Nachfolgezustand s zu wechseln . Leider gibt es für Q-Learning im nichtdeterministischen Fall keine Konvergenzgarantie, wenn wir wie zuvor gemäß (10.12) auf Seite 268 vorgehen. Dies liegt daran, dass bei aufeinanderfolgenden Durchläufen der äußeren Schleife des Algorithmus in Abb. 10.11 auf Seite 269 Belohnungs- und Nachfolgerzustand für denselben Zustand s und dieselbe Aktion a völlig unterschiedlich sein können. Dabei kann es zu einer alternierenden Sequenz kommen, die zwischen mehreren Werten hin- und herspringt. Um solche stark springenden Q- Werte zu vermeiden, fügen wir den alten gewichteten Q- Wert auf der rechten Seite von (10.12) auf Seite 268 hinzu . Dies stabilisiert die Iteration. Die Lernregel lautet dann

$$\hat{Q}_n(s, a) = (1 - \ddot{y}_n)\hat{Q}_{n-1}(s, a) + \ddot{y}_n r(s, a) + \ddot{y} \max_{ein'} \hat{Q}_{n-1}(\ddot{y}(s, a), a\ddot{y}) \quad (10.14)$$

mit einem zeitlich veränderlichen Gewichtungsfaktor

$$\ddot{y}_n = \frac{1}{1 + Mrd.(s, a)}.$$

Der Wert bn(s, a) gibt an, wie oft die Aktion a im Zustand s bei der n-ten Iteration ausgeführt wurde. Für kleine Werte von bn (also zu Beginn des Lernens) kommt der stabilisierende Term $\hat{Q}_{n-1}(s, a)$ nicht ins Spiel, da wir wollen, dass der Lernprozess schnell voranschreitet. Später wird bn jedoch größer und verhindert dadurch zu große Sprünge in der Folge der $\hat{Q}$-Werte. Bei der Integration von (10.14) in Q-Learning müssen für alle Zustands-Aktionspaare die Werte bn(s, a) gespeichert werden. Dies kann durch eine Erweiterung der Tabelle der $\hat{Q}$-Werte erreicht werden.

Zum besseren Verständnis von (10.14) vereinfachen wir dies, indem wir annehmen, dass $\ddot{y}_n = \ddot{y}$ a ist konstant und transformiere es wie folgt:

$$\hat{Q}_n(s, a) = (1 - \ddot{y})\hat{Q}_{n-1}(s, a) + \ddot{y} r(s, a) + \ddot{y} \max_{ein'} \hat{Q}_{n-1}(\ddot{y}(s, a), a\ddot{y})$$

$$= \hat{Q}_{n-1}(s, a) + \ddot{y} r(s, a) + \ddot{y} \max_{ein'} \hat{Q}_{n-1}(\ddot{y}(s, a), a\ddot{y}) \ddot{y} \hat{Q}_{n-1}(s, a).$$

TD-Fehler

Der neue Q- Wert $\hat{Q}_n(s, a)$ kann eindeutig als der alte $\hat{Q}_{n-1}(s, a)$ plus ÿ mal einem Korrekturterm dargestellt werden, der der Änderung des Q- Werts in diesem Schritt entspricht . Der Korrekturterm wird TD-Fehler oder zeitlicher Differenzfehler genannt, und die obige Gleichung zur Änderung des Q -Werts ist ein Sonderfall von TD-Learning, einer wichtigen Klasse von Lernalgorithmen [SB98]. Für ÿ = 1 erhalten wir das oben beschriebene Q-Learning. Für ÿ = 0 bleiben die $\hat{Q}$-Werte völlig unverändert. Somit findet kein Lernen statt.

## 10.7 Exploration und Ausbeutung

Für Q-Learning wurde bisher nur ein grobes Algorithmusschema angegeben. Besonders
Es fehlt eine Beschreibung der jeweiligen Wahl des Startzustands und der auszuführenden Aktionen in
der inneren Schleife von Abb. 10.11 auf Seite 269. Für die Auswahl der nächsten Aktion gibt es zwei
Möglichkeiten. Zu den möglichen Aktionen zählen
man kann zufällig ausgewählt werden. Langfristig führt dies zu einer einheitlichen Erkundung
aller möglichen Aktionen oder Richtlinien, jedoch mit sehr langsamer Konvergenz. Eine Alternative
Dazu gehört die Nutzung zuvor erlernter Qˆ-Werte. Hier immer der Agent
wählt die Aktion mit dem höchsten Qˆ-Wert. Dies führt zu einer relativ schnellen Konvergenz einer
bestimmten Flugbahn. Andere Wege bleiben jedoch durchgehend unbegangen
bis zum Ende. Im Extremfall können wir dann nicht optimale Richtlinien erhalten. In Theo rem 10.1 auf
Seite 269 ist es daher erforderlich, dass jedes Zustands-Aktionspaar besucht wird
unendlich oft. Es wird empfohlen, eine Kombination aus Erkundung und zu verwenden
Die Ausbeutung mit einem hohen Explorationsanteil am Anfang steigern und diesen immer weiter reduzieren
mehr im Laufe der Zeit.

Auch die Wahl des Ausgangszustandes beeinflusst die Lerngeschwindigkeit. In der ersten
In den drei Bildern in Abb. 10.12 auf Seite 269 ist deutlich zu erkennen, dass für die ersten Iterationen nur
die Q-Werte in der unmittelbaren Umgebung von Zustands-Aktionspaaren geändert werden
durch sofortige Belohnung. Wenn man weiter von einem solchen Punkt entfernt anfängt, ergibt sich viel
unnötige Arbeit. Dies legt nahe, Vorwissen über staatliches Handeln zu übertragen
Paare mit sofortiger Belohnung in Startzustände in der Nähe dieser Punkte. Im Kurs
des Lernens können weiter entfernte Ausgangszustände gewählt werden.

## 10.8 Approximation, Generalisierung und Konvergenz

Da Q-Learning bisher beschrieben wurde, ist explizit eine Tabelle mit allen Q-Werten erforderlich
in einer Tabelle gespeichert. Dies ist nur möglich, wenn mit einem endlichen Zustandsraum gearbeitet wird
endlich viele Aktionen. Wenn der Zustandsraum jedoch beispielsweise im Fall unendlich ist
von kontinuierlichen Variablen ist es weder möglich, alle Q-Werte zu speichern noch zu besuchen
alle Zustands-Aktionspaare während des Lernens.

Dennoch gibt es eine einfache Möglichkeit, Q-Learning und Werteiteration auf kontinuierliche
Variablen anzuwenden. Die Q(s, a)-Tabelle wird durch ein neuronales Netzwerk ersetzt, zum Beispiel a
Backpropagation-Netzwerk mit den Eingabevariablen s, a und dem Q-Wert als Ziel
Ausgabe. Für jede Aktualisierung eines Q-Wertes wird dem neuronalen Netzwerk ein Training vorgelegt
Beispiel mit (s, a) als Eingabe und Q(s, a) als Zielausgabe. Am Ende haben wir eine endliche Darstellung
der Funktion Q(s, a). Da wir immer nur endlich viele haben
Trainingsbeispiele, aber die Funktion Q(s, a) ist für unendlich viele Eingaben definiert, wir
Damit erhält man bei geeigneter Wahl der Netzwerkgröße automatisch eine Verallgemeinerung (siehe
Kap. 9). Anstelle eines neuronalen Netzwerks können wir auch ein anderes überwachtes Netzwerk verwenden
Lernalgorithmus oder ein Funktionsnäherungsgerät wie eine Support-Vektor-Maschine oder
ein Gaußscher Prozess.

Allerdings kann der Schritt von endlich vielen Trainingsbeispielen zu einer kontinuierlichen Funktion in bestimmten Situationen sehr teuer werden. Q-Learning mit Funktionsnäherung konvergiert möglicherweise nicht, da Satz 10.1 auf Seite 269 nur dann wahr ist, wenn jedes Zustands-Aktionspaar unendlich oft besucht wird.

Allerdings kann es auch bei endlich vielen Zustands-Aktionspaaren beim Einsatz von Q-Learning auf einem POMDP zu Konvergenzproblemen kommen. Q-Learning kann – in beiden beschriebenen Varianten – auf deterministische und nichtdeterministische Markov-Prozesse (MDPs) angewendet werden. Bei einem POMDP kann es vorkommen, dass der Agent, beispielsweise aufgrund verrauschter Sensoren, viele verschiedene Zustände als einen einzigen wahrnimmt. Oft werden viele Zustände in der realen Welt gezielt auf eine sogenannte *Beobachtung abgebildet.* Der resultierende Beobachtungsraum ist dann deutlich kleiner als der Zustandsraum, wodurch das Lernen schneller wird und eine Überanpassung vermieden werden kann (sie

Durch die Bündelung mehrerer Zustände kann der Agent jedoch nicht mehr zwischen den tatsächlichen Zuständen unterscheiden, und eine Aktion kann ihn in viele verschiedene Nachfolgezustände führen, je nachdem, in welchem Zustand er sich tatsächlich befindet. Dies kann zu Konvergenzproblemen für den Wert führen Iteration oder für Q-Learning. In der Literatur (z. B. in [SB98]) werden viele unterschiedliche Lösungsansätze vorgeschlagen.

Sehr vielversprechend sind auch sogenannte Policy-Verbesserungsmethoden und daraus abgeleitete Policy-Gradienten-Methoden, bei denen Q-Werte nicht verändert werden, sondern die Policy direkt verändert wird. Bei diesem Schema wird im Raum aller Policen nach einer Police gesucht, die die kumulative abgezinste Belohnung maximiert ((10.1) auf Seite 260). Eine Möglichkeit, dies zu erreichen, besteht darin, dem Gradienten der kumulativen Belohnung bis zum Maximum zu folgen. Die so gefundene Policy optimiert dann deutlich die kumulierte Belohnung. In [PS08] wird gezeigt, dass dieser Algorithmus das Lernen in Anwendungen mit großen Zustandsräumen, wie sie beispielsweise bei humanoiden Robotern auftreten, erheblich beschleunige

## 10.9 Anwendungen

Der praktische Nutzen von Reinforcement Learning wurde mittlerweile vielfach nachgewiesen. Aus einer Vielzahl von Beispielen hierzu stellen wir Ihnen kurz eine kleine Auswahl vor.

TD-Learning wurde zusammen mit einem Backpropagation-Netzwerk mit 40 bis 80 versteckten Neuronen sehr erfolgreich in TD-Gammon, einem Backgammon-Spielprogramm, eingesetzt [Tes95]. Die einzige unmittelbare Belohnung für das Programm ist das Ergebnis am Ende des Spiels. Eine optimierte Version des Programms mit einem Lookahead mit zwei Zügen wurde in 1,5 Millionen Spielen gegen sich selbst trainiert. Anschließend wurden Weltklassespieler und -spiele sowie die drei besten menschlichen Spieler besiegt.

Es gibt viele Anwendungen in der Robotik. In der RoboCup Soccer Simulation League beispielsweise setzen die besten Roboterfußballteams mittlerweile erfolgreich Reinforcement Learning ein [SSK05, Robb]. Das Balancieren einer Stange, das für einen Menschen relativ einfach ist, wurde mit Verstärkungslernen schon oft erfolgreich gelöst.

Einen eindrucksvollen Beweis für die Lernfähigkeit von Robotern lieferte Russ Tedrake auf der IROS 2008 mit seinem Vortrag über ein Modellflugzeug, das lernt, an einem genauen Punkt zu landen, genau wie ein Vogel, der auf einem Ast landet [Ted08]. Weil Luft

Da die Strömungen bei solch einem hochdynamischen Landeanflug sehr turbulent werden, ist die zugehörige Differentialgleichung, die Navier-Stokes-Gleichung, unlösbar. Die Landung kann daher nicht auf klassische mathematische Weise gesteuert werden. Tedrakes Kommentar dazu:

> „Vögel lösen Navier-Stokes nicht!“

Vögel können offensichtlich auch ohne die Navier-Stokes-Gleichung fliegen und landen lernen. Tedrake zeigte, dass dies nun auch für Flugzeuge möglich ist.

Heute ist es auch möglich, mithilfe von Q-Learning und Funktionsnäherung [RMD07] zu lernen, ein echtes Auto in nur 20 Minuten zu steuern. Dieses Beispiel zeigt, dass reale Industrieanwendungen, bei denen wenige Messungen in Aktionen abgebildet werden müssen, sehr gut in kurzer Zeit erlernt werden können.

Echte Roboter haben immer noch Schwierigkeiten beim Lernen in hochdimensionalen Zustands- und Aktionsräumen, da echte Roboter im Vergleich zu einer Simulation relativ langsam Feedback von der Umgebung erhalten. Aus zeitlichen Gründen sind die vielen Millionen notwendigen Trainingszyklen daher nicht realisierbar. Hier sind neben schnellen Lernalgorithmen auch Methoden gefragt, die es ermöglichen, zumindest Teile des Lernens offline, also ohne Rückmeldung aus der Umgebung, ablaufen zu lassen.

## 10.10 Fluch der Dimensionalität

Trotz der Erfolge in den letzten Jahren bleibt Reinforcement Learning ein aktives Forschungsgebiet in der KI, nicht zuletzt, weil selbst die besten heute bekannten Lernalgorithmen aufgrund ihrer gigantischen Rechenzeit noch immer für hochdimensionale Zustands- und Aktionsräume unpraktisch sind. Dieses Problem ist als „Fluch der Dimensionalität“ bekannt.

Auf der Suche nach Lösungen für dieses Problem beobachten Wissenschaftler Tiere und Menschen beim Lernen. Dabei fällt uns auf, dass Lernen in der Natur auf vielen Abstraktionsebenen stattfindet. Auf der untersten Stufe erlernt ein Baby zunächst einfache motorische und sprachliche Fähigkeiten. Wenn diese gut erlernt sind, werden sie gespeichert und können später jederzeit abgerufen und verwendet werden. In die Sprache der Informatik übersetzt heißt das, dass jede erlernte Fähigkeit in einem Modul gekapselt wird und dann auf einer höheren Ebene eine Handlung darstellt. Durch die Verwendung solch komplexer Aktionen auf einer höheren Ebene wird der Aktionsraum stark reduziert und somit das Lernen beschleunigt. Auf ähnliche Weise können Zustände abstrahiert und damit der Zustandsraum verkleinert werden. Dieses Lernen auf mehreren Ebenen wird hierarchisches Lernen genannt [BM03].

Ein weiterer Ansatz zur Modularisierung des Lernens ist verteiltes Lernen oder Multi-Agent-Lernen [PL05]. Beim Erlernen der motorischen Fähigkeiten eines humanoiden Roboters müssen bis zu 50 verschiedene Motoren gleichzeitig gesteuert werden, wodurch ein 50-dimensionaler Zustandsraum und auch ein 50-dimensionaler Aktionsraum entstehen. Um diese gigantische Komplexität zu reduzieren, wird die zentrale Steuerung durch eine verteilte Steuerung ersetzt. Beispielsweise könnte jeder einzelne Motor eine eigene Steuerung erhalten, die ihn direkt steuert, möglichst unabhängig von den anderen Motoren. In der Natur finden wir diese Art der Bekämpfung bei Insekten. Beispielsweise werden die vielen Beine eines Tausendfüßlers nicht von einem zentralen Gehirn gesteuert, sondern jedes Beinpaar hat sein eigenes kl

Ähnlich wie bei der uninformierten kombinatorischen Suche besteht beim Reinforcement Learning die Aufgabe, aus einer Vielzahl von Richtlinien die beste zu finden. Die Lernaufgabe wird erheblich einfacher, wenn der Agent vor Beginn des Lernens eine mehr oder weniger gute Richtlinie hat. Dann können die hochdimensionalen Lernaufgaben schneller gelöst werden. Aber wie finden wir eine solche anfängliche Politik? Hier gibt es zwei Hauptmöglichkeiten.

Die erste Möglichkeit ist die klassische Programmierung. Der Programmierer stellt dem Agenten eine Richtlinie zur Verfügung, die ein Programm umfasst, das er für gut hält. Dann erfolgt eine Umstellung, beispielsweise auf Q-Learning. Der Agent wählt, zumindest zu Beginn des Lernens, seine Handlungen entsprechend der programmierten Politik und wird so in „interessante“ Bereiche des staatlichen Handlungsraums geführt. Dies kann zu einer dramatischen Verkleinerung des Suchraums des Reinforcement Learning führen.

Wenn die herkömmliche Programmierung zu komplex wird, können wir damit beginnen, den Roboter oder Agenten zu trainieren, indem wir ihm die richtigen Aktionen von einem Menschen vorschreiben lassen. Im einfachsten Fall erfolgt dies durch manuelle Fernbedienung des Roboters. Der Roboter speichert dann die vorgeschriebene Aktion für jeden Zustand und verallgemeinert sie mithilfe eines überwachten Lernalgorithmus wie Backpropagation oder Entscheidungsbaumlernen. Dieses sogenannte Demonstrationslernen [BCDS08, SE10] bietet somit auch eine erste Richtlinie für das anschließende Verstärkungslernen.

## 10.11 Zusammenfassung und Ausblick

Heute stehen uns gut funktionierende und etablierte Lernalgorithmen für das Training unserer Maschinen zur Verfügung. Allerdings ist die Aufgabe für den menschlichen Trainer oder Entwickler bei komplexen Anwendungen immer noch anspruchsvoll. Es gibt nämlich viele Möglichkeiten, das Training eines Roboters zu gestalten, und ohne Experimente wird es nicht gelingen. Dieses Experimentieren kann in der Praxis sehr mühsam sein, da jedes neue Lernprojekt entworfen und programmiert werden muss. Hier sind Werkzeuge gefragt, die dem Trainer neben den verschiedenen Lernalgorithmen auch die Möglichkeit bieten, diese mit klassischem Programmieren und Demonstrationslernen zu kombinieren. Eines der ersten Tools dieser Art ist die Teaching-Box [ESCT09], die neben einer umfangreichen Programmbibliothek auch Vorlagen für die Konfiguration von Lernprojekten und für die Kommunikation zwischen Roboter und Umgebung bietet. Beispielsweise kann der menschliche Lehrer dem Roboter zusätzlich zum Feedback aus der Umgebung weiteres Feedback über die Tastatur oder über eine Sprachschnittstelle geben.

Reinforcement Learning ist ein faszinierendes und aktives Forschungsgebiet, das in Zukunft zunehmend genutzt werden wird. Immer mehr Robotersteuerungen, aber auch andere Programme, lernen durch Feedback aus der Umgebung. Heutzutage gibt es eine Vielzahl von Variationen der vorgestellten Algorithmen und auch völlig unterschiedliche Algorithmen. Das Skalierungsproblem bleibt ungelöst. Für kleine Aktions- und Zustandsräume mit wenigen Freiheitsgraden können beeindruckende Ergebnisse erzielt werden. Wenn die Anzahl der Freiheitsgrade im Zustandsraum beispielsweise bei einem einfachen humanoiden Roboter auf 18 anwächst, wird das Lernen sehr teuer.

Für weitere Grundlagenvorlesungen empfehlen wir die kompakte Einführung in das Reinforcement Learning in Tom Mitchells Buch [Mit97]. Das Standardwerk von Suton und Barto [SB98] ist gründlich und umfassend, ebenso wie der Übersichtsartikel von Kaelbling, Littman und Moore [KLM96].

## 10.12 Übungen

**Aufgabe 10.1 (a)**

Berechnen Sie die Anzahl verschiedener Richtlinien für n Zustände und n Aktionen. Somit sind Übergänge von jedem Staat zum anderen möglich. (b) Wie ändert sich die Anzahl der Richtlinien in Teilproblem (a), wenn leere Aktionen, also Aktionen von einem Zustand zu sich selbst, nicht erlaubt sind? (c) Geben Sie mithilfe von Pfeildiagrammen wie denen in Abb. 10.3 auf Seite 259 alle Richtlinien für an zwei Staaten.

(d) Geben Sie mithilfe von Pfeildiagrammen alle Richtlinien ohne leere Aktionen für drei Zustände an.

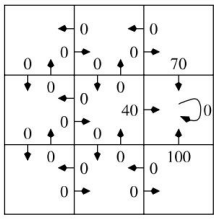
**Übung 10.2** Verwenden Sie die Werteiteration manuell in Beispiel 10.1 auf Seite 257 mit nx = ny = 2.

**Übung 10.3** Führen Sie verschiedene Experimente mit einem Werteiterationssimulator durch. (a) Installieren Sie den Werteiterationssimulator von [Tok06]. (b) Reproduzieren Sie die Ergebnisse aus Übung 10.2 , indem Sie zunächst das Feedback mit eingeben des Feedback-Editors und anschließende Durchführung der Werteiteration. (c) Modellieren Sie Oberflächen unterschiedlicher Glätte und beobachten Sie, wie sich die Richtlinie ändert. (d) Erweitern Sie mit einer ähnlichen Rückkopplungsmatrix den Zustandsraum schrittweise auf etwa 100 × 100 und passen Sie den Abzinsungsfaktor ÿ so an, dass sich eine sinnvolle Richtlinie ergibt.

**ÿ Aufgabe 10.4** Zeigen Sie, dass für die Beispielrechnung in Abb. 10.8 auf Seite 264 ) **ÿ 4,05499** ist . Der genaue Wert ist $\hat{V}^{6}(3, 3) = 1{,}9/(1 - 0{,}9$

**Übung 10.5**

Führen Sie Q-Learning im 3×3-Raster rechts durch. Der Zustand in der Mitte rechts ist ein absorbierender Zielzustand.




Aufgabe 10.6 Gegeben ist ein Roboterarm mit n Gelenken (Dimensionen) und ÿ diskreten Zuständen pro Gelenk. Aktionen von jedem Zustand zu jedem Zustand sind möglich (wenn der Roboter nichts tut, wird dies als (leere) Aktion gewertet).

**(a) Geben Sie eine Formel für die Anzahl der Zustände und die Anzahl der Aktionen in jedem Zustand an und für die Anzahl der Richtlinien für den Roboter.**

**(b) Erstellen Sie eine Tabelle mit der Anzahl der Strategien für n = 1, 2, 3, 4, 8 und ÿ = 1, 2, 3, 4, 10.**

**(c) Um die Anzahl möglicher Strategien zu reduzieren, nehmen Sie an, dass die Anzahl möglicher Aktionen pro Gelenk immer gleich 2 ist und dass der Roboter jeweils nur ein Gelenk bewegen kann. Geben Sie eine neue Formel für die Anzahl der Strategien an und erstellen Sie die zugehörige Tabelle.**

**(d) Begründen Sie mit dem berechneten Ergebnis, dass ein Agent, der autonom und adaptiv mit n = 8 und l = 10 agiert, durchaus als intelligent bezeichnet werden kann.**

# 11 Lösungen zu den Übungen

## 11.1 Einführung

**Aufgabe 1.3 Viele bekannte Inferenzprozesse, Lernprozesse etc. sind NP-vollständig oder sogar unentscheidbar. Was bedeutet das für KI?**

**Aufgabe 1.4 Wenn ein Problem NP-vollständig ist oder als „schwer“ beschrieben werden kann, bedeutet das, dass es Fälle gibt, in denen das Problem nicht in akzeptabler Zeit gelöst werden kann. Dies ist der sogenannte *Worst Case.* Bei manchen Anwendungen müssen wir damit leben, dass im schlimmsten Fall keine effiziente Lösung möglich ist. Das bedeutet, dass es auch in Zukunft praktisch relevante Probleme geben wird, die in bestimmten Sonderfällen unlösbar sind.**

**KI wird daher weder eine universelle Formel finden, noch eine Supermaschine bauen, mit der alle Probleme lösbar werden. Es stellt sich vielmehr die Aufgabe, Systeme mit einer höheren Wahrscheinlichkeit, eine Lösung zu finden, bzw. mit einer höheren Wahrscheinlichkeit, schnelle, optimale Lösungen zu finden, aufzubauen. Wir Menschen kommen im Alltag recht gut mit suboptimalen Lösungen zurecht. Der Grund liegt ganz einfach in den zu hohen Kosten für die Suche nach der optimalen Lösung. Beispielsweise brauche ich nur sieben Minuten, um mit einer Karte in einer unbekannten Stadt den Weg von Punkt A nach Punkt B zu finden. Der kürzeste Weg hätte nur sechs Minuten gedauert. Den kürzesten Weg zu finden, hätte jedoch vielleicht eine Stunde gedauert. Der Beweis der Optimalität des Pfades dürfte noch aufwändiger sein.**

**Aufgabe 1.5**

**(a) Die Ausgabe hängt nicht nur von der Eingabe ab, sondern auch vom Inhalt des Speichers. Für eine Eingabe x könnte die Ausgabe je nach Inhalt des Speichers y1 oder y2 sein. Es ist also nicht eindeutig und daher keine**

**Funktion. (b) Betrachtet man den Inhalt des Speichers als weiteren Input, dann ist der Output eindeutig (da der Agent deterministisch ist) und der Agent stellt eine Funktion dar.**

W. Ertel, *Einführung in die künstliche Intelligenz,* Bachelor-Themen in Informatik, DOI 10.1007/978-0-85729-299-5_11, 

**Aufgabe 1.6 (a)**

**Geschwindigkeit vx (t) = $\frac{\ddot{y}x}{\ddot{y}t}$ limtÿ0 $\frac{x(t)ÿx(tÿt) =}{\tau}$ ÿ $\frac{x(t)ÿx(tÿt)}{\tau}$. vy wird berechnet analog. ÿ 2x vx (t)ÿvx (tÿt) (b) Beschleunigung ax(t) = vx(t) = limtÿ0 ÿt2 x(t)ÿx(tÿt) x(tÿt)ÿ x(tÿ2t) (t)2 (t)2 $\frac{}{\ddot{y}\ddot{y}t}$ $\frac{}{\tau}$ = ay wird analog berechnet . Man benötigt außerdem die Position zu den drei Zeitpunkten t ÿ 2t, t = x(t)ÿ2x(tÿt)+x(tÿ2t) limtÿ0 . ÿ t, t.**

**Aufgabe 1.7**

**(a) Kosten für Agent 1 = 11 · 100 Cent + 1 · 1 Cent = 1, 101 Cent.**
**Kosten für 2 = 0 · 100 Cent + 38 · 1 Cent = 38 Cent.**
**Daher spart Agent 2 1.101 Cent ÿ 38 Cent = 1.063 Cent. (b) Gewinn für Agent 1 = 189 · 100 Cent + 799 · 1 Cent = 19.699 Cent.**
**Gewinn für Agent 2 = 200 · 100 Cent + 762 · 1 Cent = 20.762 Cent.**
**Agent 2 hat also 20.762 Cent ÿ 19.699 Cent = 1.063 Cent höheren Gewinn.**
**Bewertet man die durch Fehler entgangenen Gewinne, trifft der nutzenbasierte Agent die gleichen Entscheidungen wie ein kostenbasierter Agent.**

## 11.2 Aussagenlogik

**Aufgabe 2.1 Mit der Signatur = {ÿ1, ÿ2,...,ÿn} und der Grammatikvariablenformel *lässt* sich die Syntax der Aussagenlogik wie folgt definieren:**

***Formel ::=*** **ÿ1|ÿ2|···|ÿn|w|f**
**| ¬Formel|(Formel)| FormelÿFormel**
**| FormelÿFormel|FormelÿFormel**
**| FormelÿFormel**

**Aufgabe 2.2 Beweis mit der Wahrheitstabellenmethode.**

**Aufgabe 2.3 (a) (¬A ÿ B) ÿ (¬B ÿ A) (b) (¬A ÿ B) ÿ (¬B ÿ A) (c) t**

**Aufgabe 2.4 (a) erfüllbar (b) wahr (c) unerfüllbar**

**Aufgabe 2.6**

**(a) In Aufgabe 2.3(c) wurde bereits gezeigt, dass A ÿ (A ÿ B) ÿ B eine Tautologie ist. Der Abzugssatz stellt somit die Richtigkeit der Inferenzregel sicher.**

**(b) Wir zeigen mit der Wahrheitstabellenmethode, dass (A ÿ B) ÿ (¬B ÿ C) ÿ (A ÿ C) ist eine Tautologie.**

**Aufgabe 2.7** Die Anwendung der Resolutionsregel auf die Klausel (f ÿ B) und (¬B ÿ f ) ergibt das Resolvente (f ÿ f ) ÿ (f ). Nun wenden wir die Auflösungsregel auf die Klauseln B und ¬B an und erhalten die leere Klausel als Resolvente. Weil (f ÿ B) ÿ B und (¬B ÿ f ) ÿ ¬B, (f ) ÿ (). In der Praxis ist es wichtig, dass die Ableitung des Leersatzes immer auf einen Widerspruch zurückzuführen ist.

**Aufgabe 2.8** Wenn *KB* einen Widerspruch enthält, dann gibt es zwei Klauseln A und ¬A, aus denen sich die leere Klausel ableiten lässt. Der Widerspruch in *KB* besteht eindeutig immer noch in *KB* ÿ ¬Q. Daher ermöglicht es auch die Ableitung der Leerklausel.

**Aufgabe 2.9** (a) (A ÿ B) ÿ (¬A ÿ ¬B) (b) (A ÿ B) ÿ (B ÿ C) ÿ (A ÿ C)

**Übung 2.10** *Formalisierung:* Komplize: A, Auto: C, Schlüssel: K

$$WB \ \ddot{y} \ (A \ \ddot{y} \ C) \ \ddot{y} \ [(\neg A \ \ddot{y} \ \neg K) \ \ddot{y} \ (A \ \ddot{y} \ K)] \ \ddot{y} \ K$$

*Transformation in CNF:* (¬A ÿ ¬K) ÿ (A ÿ K) ÿ (¬K ÿ A) ÿ (¬A ÿ K)

Versuchen Sie, C zu beweisen und ¬C zur Klauselmenge hinzuzufügen. Der CNF-Klauselsatz ist

$$(\neg A \ \ddot{y} \ C)1 \ \ddot{y} \ (\neg K \ \ddot{y} \ A)2 \ \ddot{y} \ (\neg A \ \ddot{y} \ K)3 \ \ddot{y} \ (K)4 \ \ddot{y} \ (\neg C)5.$$

*Auflösungsbeweis :* Res(2, 4) : (A)6
Res(1, 6) : (C)7
Res(7, 5) : ()8
Somit wurde C gezeigt.

**Aufgabe 2.11**

(a) *KB* ÿ (A ÿ B) ÿ (¬B ÿ C), Q ÿ (A ÿ C)
*KB* ÿ ¬Q ÿ (A ÿ B)1 ÿ (¬B ÿ C)2 ÿ (¬A)3 ÿ (¬C)4
*Auflösungsnachweis :* Res(1, 3): (B)5
Res(2, 4) : (¬B)6
Res(5, 6) : ()

(b) ¬(¬B ÿ (B ÿ ¬A) ÿ ¬A) ÿ (¬B)1 ÿ (B ÿ ¬A)2 ÿ (A)3
*Auflösungsnachweis :* Res(1, 2): (¬A)4
Res(3, 4) : ()

**Aufgabe 2.12** Durch Anwendung der Äquivalenzen aus Satz 2.1 auf Seite 18 können wir die Behauptungen sofort beweisen.

**Übung 2.13**

Res(8, 9) : (C ÿ F ÿ E ÿ f )10
Res(3, 10) : (F ÿ E ÿ f )11
Res(6, 11) : (A ÿ B ÿ C ÿ E ÿ f )12
Res(1, 12) : (B ÿ C ÿ E ÿ f )13
Res(2, 13) : (C ÿ E ÿ f )14
Res(3, 14) : (E ÿ f )15
Res(5, 15) : ()

## 11.3 Prädikatenlogik erster Ordnung

**Aufgabe 3.1**

**(a) ÿx männlich(x) ÿ ¬weiblich(x)**
**(b) ÿx ÿy ÿz Vater(x, y) ÿ männlich(x) ÿ Kind(y, x, z) (c)**
**ÿ x ÿy Geschwister(x, y) ÿ [(ÿz Vater(z, x) ÿ Vater(z, y)) ÿ (ÿz Mutter(z, x) ÿ Mutter(z, y))]**
**(d) ÿx ÿy ÿz Eltern(x, y, z) ÿ Vater(x, z) ÿ Mutter(y, z) (e) ÿx ÿy**
**Onkel(x, y) ÿ ÿz ÿu Kind(y, z, u) ÿ Geschwister(z, x) ÿ männlich(x) (f) ÿx**
**ÿy Vorfahr(x, y) ÿ ÿz Kind(y, x, z ) ÿ ÿu ÿv kind(u, x, v) ÿ Vorfahr(u, y))**

**Aufgabe 3.2**

**(a) ÿx ÿy ÿz Vater(y, x) ÿ Mutter(z, x) (b)**
**ÿx ÿy Kind(y, x, z) (c)**
**ÿx Vogel(x) ÿ fliegt (x) (d)**
**ÿx ÿy ÿz Tier(x) ÿ Tier(y) ÿ isst(x, y) ÿ isst(y, z) ÿ Getreide(z) (e) ÿx Tier(x)**
**ÿ (ÿy (isst(x, y) ÿ (Pflanze(y) ÿ (Tier(y) ÿ ÿz Pflanze(z) ÿ isst(y, z) ÿ viel_kleiner(y, x)))**

**Aufgabe 3.3 ÿx ÿy ÿzx = Vater(y) ÿ männlich(x) ÿ Kind(y, x, z)**

**Aufgabe 3.4**
**ÿx ÿy x<y ÿ y<x ÿ x = y, ÿx ÿy**
**x<y ÿ ¬y<x, ÿx ÿy ÿz x<y**
**ÿ y<z ÿ x<z**

**Übung 3.5**

**(a) MGU: x/f (z), u/f (y), Term: p(f (z), f (y)) (b)**
**nicht vereinbar**
**(c) MGU: x/ cos y, z/ 4 ÿ 7 · cos y, Term: cos y = 4 ÿ 7 · cos y**
**(d) nicht**
**vereinbar (e) MGU: u/f (g(w, w), g(g(w, w), g( w, w)), g(g(g(w, w), g(w, w)), g(g(w, w), g(w, w)))), x/g(w, w), y/g(g(w, w), g(w, w)), z/g(g(g(w, w), g(w, w)), g(g(w, w ), g(w, w))) Term:**
**q(f (g(w, w), g(g(w, w), g(w, w)), g(g(g(w, w) , g(w, w)), g(g(w, w), g(w, w)))), f (g(w, w), g(g(w, w), g(w, w)), g(g(g(w, w), g(w, w)), g(g(w, w), g(w, w)))))**

**Aufgabe 3.7**

**(a) Gegeben sei die unerfüllbare Formel p(x) ÿ ¬p(x) ÿ r(x). Wir wählen die Klausel r(x) als SOS, sodass kein Widerspruch abgeleitet werden**
**kann. (b) Wenn das SOS bereits unerfüllbar ist, kann kein Widerspruch abgeleitet werden. Wenn nicht, sind Auflösungsschritte zwischen Klauseln aus SOS und *(KB* ÿ ¬Q)\SOS erforderlich.**

(c) Wenn es kein Komplement zu einem Literal L in einer Klausel K gibt, dann verbleibt das Literal L in jeder Klausel, die durch Auflösung aus Klausel K abgeleitet wird. Somit kann die leere Klausel weder aus K oder seiner Auflösung abgeleitet werden, noch aus irgendeinem anderen zukünftige Entschlossenheit.

Aufgabe 3.8 ¬Q ÿ *KB* ÿ (e = n)1 ÿ (n · x = n)2 ÿ (e · x = x)3 ÿ (¬a = b)4 Beweis: Dem(1, 2) : (e · x = e)5 Tra,Sym(3, 5) : (x = e)6 Dem(4, 6) : (¬e = b)7 Dem(7, 6) : (¬e = e)8 ( )

Dabei steht „Dem“ für Demodulation. Klausel Nummer 6 wurde durch Anwendung der Transitivität und Symmetrie der Gleichheit in den Klauseln 3 und 5 abgeleitet.

Übung 3.9 Die LOP-Eingabedateien sind:

(a)
```
a;b<-. <-
a,b.
b;c<-. <-
b,c.
c;a<-.
<-c,a.
```

(b)
```
<- rasiert(barb,X),
   rasiert(X,X).
   rasiert(barb,X);
   rasiert(X,X) <-.
```

(c)
```
e = n. <-
   a = b.
   m(m(X,Y),Z)=m(X,m(Y,Z)). m(e,X) =
   X. m(n,X) = n.
```

## 11.4 Grenzen der Logik

Aufgabe 4.1

(a) Richtig: *Wir nehmen eine vollständige Beweisrechnung für PL1. Damit finden wir für jede wahre Formel in endlicher Zeit einen Beweis. Für alle unerfüllbaren Formeln gehe ich wie folgt vor: Ich wende den Kalkül auf* ¬ÿ an *und zeige, dass* ¬ÿ *wahr ist. Somit* ist ÿ *falsch. Somit können wir jede wahre Formel aus PL1 beweisen und jede falsche Formel widerlegen.* Leider ist dieses Verfahren für erfüllbare Formeln ungeeignet.

Übung 4.2 (a)

Er rasiert sich genau dann, wenn er sich nicht rasiert. (Widerspruch) (b) Die Menge aller Mengen, die sich selbst nicht enthalten. Es enthält genau dann sich selbst wenn es sich nicht selbst enthält.

## 11.5 PROLOG

Übung 5.1 PROLOG signalisiert einen Stapelüberlauf. Der Grund ist die Tiefensuche von PROLOG, die immer das erste vereinbare Prädikat in der Eingabedatei wählt.
Bei rekursiven Prädikaten wie Gleichheit führt dies zu einer nicht terminierenden Rekursion.

**Übung 5.2**

```
write_path( [H1,H2|T] ) :- write_path([H2|T]), write_move(H2,H1). write_path( [_X] ).
write_move( state(X,W,Z,K),
state(Y,W,Z,K) ) :-
        write('Farmer from '), write(X), write(' to '), write(Y), nl.
write_move( state(X,X,Z,K), state(Y,Y,Z,K) ) :-
        write('Bauer und Wolf aus '), write(X), write(' bis '), write(Y),nl.
write_move( state(X,W,X,K), state(Y,W,Y,K) ) :-
        write('Bauer und Ziege von '),write(X),write(' bis '), write(Y), nl.
write_move( state(X,W,Z,X), state(Y,W,Z,Y) ) :-
        write('Bauer und Kohl aus '),write(X),write(' bis '), write(Y), nl.
```

**Aufgabe 5.3**

**(a) Es wird benötigt, die gefundenen Lösungen auszugeben. Die Vereinheitlichung im Faktenplan (Ziel, Ziel, Pfad, Pfad). kümmert sich darum. (b)**

**Die Bedingungen für den Eintritt in die Klauseln des Prädikats sicher überschneiden sich. Das durch den Fehler verursachte Zurückverfolgen führt zur Ausführung der zweiten oder dritten Alternative von sicher, wobei die gleiche Lösung offensichtlich wieder gefunden wird. Eine Kürzung am Ende der ersten beiden Safe-Klauseln löst das Problem. Alternativ können alle sicheren Zustände explizit angegeben werden, wie zum Beispiel „safe(state(left,left,left,right)“).**

**Aufgabe 5.5**

```
?- eins(10). one(0) :-
        write(1). eins(N) :- N1 ist N-1,
        eins(N1), eins(N1).
```

**Aufgabe 5.6**

**(a) Mit n1 = |L1|, n2 = |L2| es ist wahr, dass Tappend(n1, n2) = ÿ(n1). (b) Mit n = |L| es gilt, dass Tnrev(n) = ÿ(n2 ). (c) Mit n = |L| es ist wahr, dass Taccrev(n) = ÿ(n).**

**Übung 5.7 Für die Bäume mit Symbolen auf den inneren Knoten können wir beispielsweise die Terme a(b,c) und a(b(e,f,g),c(h),d) verwenden. Bäume ohne Symbole: Baum(b,c) oder Baum(Baum(e,f,g),Baum(h),d).**

**Aufgabe 5.8 SWI-PROLOG-Programme:**

**(a)**

```
fib(0,1). fib(1,1).
    fib(N,R):- N1 ist N-1, fib(N1,R1), N2 ist N-2, fib(N2,R2), R ist R1 + R2.
```

**(b) T (n) = ÿ(2 (c) :- dynamische fib/2.**

```
fib(0,1). fib(1,1).  N ).

    fib(N,R) :- N1 ist N-1, fib(N1,R1), N2 ist N-2, fib(N2,R2), R ist R1 + R2,
    Asserta(fib(N,R)).
```

**(d) Wenn die Fakten fib(0,1) bis fib(k,f ib(k)) bereits berechnet wurden, dann gilt für den Aufruf fib(n,X) das**

$$T(n) = \begin{cases} \ddot{y}(1) & \text{wenn } n \ddot{y} k, \\ \ddot{y}(n \ddot{y} k), \text{ wenn } n > k. \end{cases}$$

**(e) Weil nach den ersten n Aufrufen des fib-Prädikats alle weiteren Aufrufe direkt sind Zugang zu Fakten.**

**Übung 5.9**

**(a) Die Lösung wird hier nicht offengelegt.**

**(B)**

```
Start :-
    fd_domain([Brite, Schwede, Däne, Norwegisch, Deutsch],1,5),
    fd_all_different([Brite, Schwede, Däne, Norweger, Deutsch]),
    fd_domain([Tee, Kaffee, Wasser, Bier, Milch],1,5),
    fd_all_different([Tee, Kaffee, Wasser, Bier, Milch]),
    fd_domain([Rot, Weiß, Grün, Gelb, Blau],1,5),
    fd_all_different([Rot, Weiß, Grün, Gelb, Blau]),
    fd_domain([Hund, Vogel, Katze, Pferd, Fisch],1,5),
    fd_all_different([Hund, Vogel, Katze, Pferd, Fisch]),
    fd_domain([Pallmall, Dunhill, Marlboro, Winfield, Rothmanns],1,5),
    fd_all_different([Pallmall, Dunhill, Marlboro, Winfield, Rothmanns]),
    fd_labeling([Brite, Schwede, Däne, Norwegisch, Deutsch]),
    Brite #= Rot, % Der Brite lebt im roten Haus
    Schwede #= Hund, % Der Schwede hat keinen Hund
    Däne #= Tee, % Der Däne trinkt gern Tee
    Grün #= Weiß - 1, % Das grüne Haus befindet sich links vom weißen Haus
    Grün #= Kaffee, % Der Besitzer des Gewächshauses trinkt Kaffee
    Pallmall #= Bird, % Die Person, die Pall Mall raucht, hat einen Vogel
    Milch #= 3, % Der Mann im mittleren Haus trinkt Milch
    Gelb #= Dunhill, % Der Besitzer des gelben Hauses raucht Dunhill
    Norweger #= 1, % Der Norweger wohnt im ersten Haus
    dist(Marlboro,Cat)#= 1, % Der Marlboro-Raucher wohnt neben der Katze
    dist(Horse,Dunhill) #= 1, % Der Mann mit dem Pferd wohnt neben dem Dunhill-Raucher
    Winfield #= Beer,                    % Der Winfield-Raucher trinkt gern Bier
    dist(Norwegian,Blue) #= 1, % Der Norweger wohnt neben dem blauen Haus
    Deutsch #= Rothmanns,                % Der Deutsche raucht Rothmanns
    dist(Marlboro,Water)#=1, % Der Nachbar des Marlboro-Rauchers trinkt Wasser.
    write([Brite, Schwede, Däne, Norwegisch, Deutsch]), nl,
    write([Hund, Vogel, Katze, Pferd, Fisch]), nl.
```

# 11.6 Suche, Spiele und Problemlösung

**Übung 6.1**

**(a) Auf der letzten Ebene gibt es b $^{D}$ Knoten. Alle vorherigen Level zusammen haben**

$$Nb(dmax) = \sum_{ich=0}^{dÿ1} ich\ b = \frac{d\ b\ ÿ\ 1}{b\ ÿ\ 1}\ ÿ\ b^{\ dÿ1}$$

**Knoten, wenn b groß wird. Weil b $^{d\ /bdÿ1 = b}$ gibt es ungefähr b mal as viele Knoten auf der letzten Ebene wie auf allen anderen Ebenen zusammen.**

**(b) Für einen nicht konstanten Verzweigungsfaktor gilt diese Aussage nicht mehr, wie das folgende Gegenbeispiel zeigt: Ein Baum, der bis zur vorletzten Ebene stark verzweigt, gefolgt von einer Ebene mit einem konstanten Verzweigungsfaktor von 1 , hat genau so viele Knoten auf der letzten Ebene wie auf der vorletzten Ebene.**

**Übung 6.2**

**(a) In Abb. 6.3 auf Seite 86 die Struktur des Baums auf der zweiten Ebene mit seinen acht Knoten wiederholt sich.** Somit können wir **aus b den durchschnittlichen Verzweigungsfaktor bm** berechnen $^{2}_{M}$ **8 zu bm = ÿ 8.** $^{=}$

(b) Hier ist die Berechnung nicht so einfach, da der Wurzelknoten des Baums stärker verzweigt als alle anderen. Wir können jedoch sagen, dass im Inneren des Baumes der Verzweigungsfaktor genau um 1 kleiner ist, als er ohne die Zyklusprüfung wäre. Somit ist bm ÿ ÿ 8 ÿ 1 ÿ 1,8.

**Aufgabe 6.3**

(a) Für den durchschnittlichen Verzweigungsfaktor ist die Anzahl der Blattknoten festgelegt. Beim effektiven Verzweigungsfaktor hingegen ist die Anzahl der Knoten im gesamten Baum festgelegt.

(b) Weil die Anzahl aller Knoten im Baum normalerweise ein besseres Maß für die Rechenzeit für die Suche eines gesamten Baums ist als die Anzahl der Blattknoten. (c) Für ein großes b gilt nach (6.1) auf Seite 87 n ÿ b¯d+1 /b¯ = b¯d , ergibt b¯ = ÿd n.

**Aufgabe 6.4**

(a) 3-Puzzle: 4! = 24 Staaten, 8-Puzzle: 9! = 362 880 Staaten, 15-Puzzle: 16! = 20 922 789 888 000 Staaten.

(b) Nachdem wir das leere Quadrat 12-mal im Uhrzeigersinn verschoben haben, erreichen wir wieder den Ausgangszustand und erzeugen so einen zyklischen Unterraum mit 12 Zuständen.

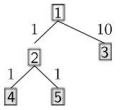
**Aufgabe 6.6 (a) Mathematica-Programm:**

```
0 BreadthFirstSearch[Node_, Goal_, Depth_] := Module[{i, NewNodes={}},
1 For[i=1, i<= Länge[Knoten], i++, 2
                NewNodes = Join[ NewNodes, Successors[ Node[[i]] ] ];
3               If[MemberQ[Successors[ Node[[i]] ], Goal],
4                       Return[{"Lösung gefunden", Tiefe+1}], 0
5                   ]
6           ];
7 Wenn[Länge[NeueNodes] > 0, 8
              BreadthFirstSearch[NewNodes, Ziel, Tiefe+1],
9             Return[{"Fail", Depth}]
10        ]
11]
```

(b) Weil die Tiefe des Suchraums nicht begrenzt ist.

**Aufgabe 6.7**

(a) Für konstante Kosten 1 sind die Kosten aller Pfade in der Tiefe d kleiner als die Kosten aller Pfade in der Tiefe d + 1. Da alle Knoten in der Tiefe d vor dem ersten Knoten in der Tiefe d + 1 getestet werden, Es wird garantiert, dass eine Lösung der Länge d vor einer Lösung der Länge d + 1 gefunden wird. (b) Für den benachbarten Suchbaum werden zuerst Knoten Nummer 2 und der Lösungsknoten Nummer 3 mit Kosten 10 generiert. Die Suche wird beendet. Die Knoten 4 und 5 mit Pfadkosten von jeweils 2 werden nicht generiert. Die optimale Lösung wird daher nicht gefunden.

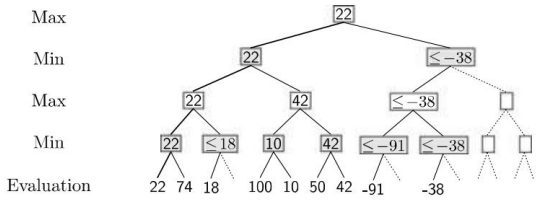


**Übung 6.11 Tiefensuche: Der neue Knoten muss eine niedrigere Bewertung haben als alle offenen Knoten. Breitensuche: Der neue Knoten muss eine höhere Bewertung haben als alle offenen Knoten.**

**Übung 6.12 Wie bei einer zulässigen Heuristik unterschätzt die Frau die Entfernung zum Ziel. Dies könnte dazu führen, dass die beiden den schnellsten Weg zum Ziel finden – wenn auch mit großem Aufwand. Dies gilt jedoch nur, wenn die Dame die Distanz stets unterschätzt.**

**Übung 6.14**

**(A)**




## 11.7 Argumentieren mit Unsicherheit

**Aufgabe 7.1 |**

**1. P (ÿ) = 2. ÿ| = 1. |ÿ|**

**P (ÿ) = 1 ÿ P (ÿ) = 0.**

**3. Für paarweise exklusive Ereignisse A und B: P (A ÿ B) = |ÿ| |AÿB| = |A|+|B| | = P (A)+ ÿ|**

**P (B).**

**4. Für zwei komplementäre Ereignisse A und ¬A: P (A) + P (¬A) = P (A) + P (ÿ ÿ |ÿÿ A| |ÿ| |AÿB|**

**A) = | 5. P = |ÿ| = 1.**
**A| |ÿ|+ (A |ÿ|**

**|ÿ| ÿ B) = = |A|+|B|ÿ|AÿB| = P(A) + P (B) ÿ P (A ÿ B). |ÿ| |B| = P(B). |ÿ|**

**6. Für A ÿ B: P (A) = | A| |ÿ|ÿ**

**7. Gemäß Definition 7.1 auf Seite 115: A1 ÿ···ÿ An = ÿ und |A1 ÿ··· ÿAn| |ÿ| N i=1 P (Ai) =**

**ni=1 |Ai | |ÿ| = Ni=1 |Ai | = = |ÿ| = 1. |ÿ|**
**|ÿ|**

**Aufgabe 7.2**

**(a)**

**P (y > 3 /Klasse = gut) = 7/9 P (Klasse = gut) = 9/13 P (y > 3) = 7/13,**

$$P(Klasse = gut \mid y > 3) = \frac{P(y > 3 \mid Klasse = gut) \cdot P(Klasse = gut)}{P(y > 3)}$$

$$= \frac{7/9 \cdot 9/13}{1,\ 7/13} =$$

$$P(\textit{Klasse} = \text{gut} \mid y \ddot{y} 3) = \frac{P(y \ddot{y} 3 \mid \textit{Klasse} = \text{gut}) \cdot P(\textit{Klasse} = \text{gut})}{P(y \ddot{y} 3)}$$

$$= \frac{2/9 \cdot 9/13}{6/13} = \frac{1}{3}.$$

**(b) P (y > 3 |*Klasse* = gut) = 7/9.**

**Aufgabe 7.3**

**(a) acht**
**Ereignisse. (B)**

$$P(\textit{Prec} = \text{trocken} \mid \text{Himmel} = \text{klar}, \text{Balken} = \text{steigend})$$

$$= \frac{P(\textit{Prec} = \text{trocken}, \textit{Himmel} = \text{klar}, \text{Balken} = \text{steigend})}{P(\textit{Himmel} = \text{klar}, \text{Balken} = \text{steigend})} = \frac{0,4}{0,47} = 0,85.$$

**(c) Fehlende Zeile in der Tabelle: Bewölkt Fallend Regen 0,12**

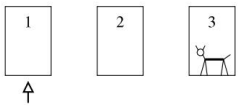
$$P(\textit{Prec} = \text{regnet} \mid \text{Sky} = \text{bewölkt}) = \frac{P(\textit{Prec} = \text{regnet}, \textit{Sky} = \text{bewölkt})}{P(\textit{Himmel} = \text{bewölkt})}$$

$$= \frac{0,23}{0,35} = 0,66.$$

**(d) Das fehlende Wahrscheinlichkeitsmaß beträgt 0,15. Das Indifferenzprinzip (Definition 7.5 auf Seite 126) erfordert nun, dass beiden fehlenden Zeilen symmetrisch der Wert 0,075 zugewiesen wird.**

**Übung 7.4**

**Der Kandidat wählte zunächst Tür Nummer 1. Dann öffnete der Moderator Tür Nummer 3. Wir führen die Abkürzung Ai für „Auto steht hinter Tür Nummer i (bevor das Experiment beginnt)“ und Mi für „Gastgeber (Moderator) öffnet Tür Nummer i“** ein. . **Dann ist P (A1) = P (A2) = P (A3) = 1/3 und wir berechnen**




$$P(A1|M3) = \frac{P(M3|A1)P(A1)}{P(M3)}$$

$$= \frac{P(M3|A1)P(A1)}{P(M3|A1)P(A1) + P(M3|A2)P(A2) + P(M3|A3)P(A3)\ 1/2 \cdot 1/3}$$

$$= \frac{1/2 \cdot 1/3}{+ 1 \cdot 1/3 + 0 \cdot 1/3} = \frac{1/2 \cdot 1/3}{= 1/3,\ 1/2}$$

$$P(A2|M3) = \frac{P(M3|A2)P(A2)}{P(M3)} = \frac{1 \cdot 1/3}{= 2/3.\ 1/2}$$

**Somit ist klar, dass es besser ist, zur anderen Tür zu wechseln.**

**Aufgabe 7.5 Die Lagrange-Funktion lautet L = ÿn i=1 pi lnpi + ÿ(n i=1pi ÿ 1). Wenn man die partiellen Ableitungen nach pi und pj gleich Null setzt, erhält man**

$$\frac{\ddot{y}L}{} = \ddot{y}\ln pi \ddot{y} 1 + \ddot{y} = 0 \text{ und } \ddot{y}pi \qquad \frac{\ddot{y}L}{} = \ddot{y}\ln pj \ddot{y} 1 + \ddot{y} = 0. \ddot{y}pj$$

**Die Subtraktion dieser beiden Gleichungen ergibt pi = pj für alle i, j ÿ {1,...,n}. Somit ist p1 = p2 =···= pn = 1/n.**

**Übung 7.6**

| **PIT-Eingabedaten:** | **PIT-Ausgabe:** |
|---|---|
| **var A{t,f}, B{t,f}; P([A=t]) = 0,5; P([B=t] \| [A=t]) = 0,94; QP([B=t]);** | **Status der Abfragen melden**<br><br>**Nr Wahrheitswert-Wahrscheinlichkeitsabfrage 1 NICHT ANGEGEBEN 7.200e-01 QP([B=t]);** |

**Da PIT numerisch arbeitet, kann es mit dem pa nicht symbolisch rechnen Parameter ÿ und ÿ, eher nur mit konkreten Zahlen.**

**Übung 7.7**

**Wenn: p1 = P (A, B), p2 = P (A,¬B), p3 = P (¬A, B), p4 = P (¬A,¬B)**

$$\text{Die Einschränkungen sind: } p1 + p2 = \ddot{y} \tag{11.1}$$

$$p4 = 1 \ddot{y} \ddot{y} \tag{11.2}$$

$$p1 + p2 + p3 + p4 = 1 \tag{11.3}$$

**Daraus folgt p3 = ÿ ÿ ÿ. Aus (11.1) schließen wir auf der Grundlage der Indifferenz, dass p1 = p2 = ÿ/2. Somit ist P (B) = p1 + p3 = ÿ/2 + ÿ ÿ ÿ = ÿ ÿ ÿ/2 = P (A ÿ B) ÿ 1/2P (A). Das entsprechende PIT-Programm liest**

```
var A{t,f}, B{t,f}; P([A=t]) = 0,6;
P([B=t] ODER [A=t]) =
0,94; QP([B=t]).
```

**Aufgabe 7.8 Die Lagrange-Funktion lautet**

$$L = \ddot{y}_{i=1}^{4} pi \ln pi + \ddot{y}1(p1+p2\ddot{y}\ddot{y}) + \ddot{y}2(p1+p3\ddot{y}\ddot{y}) + \ddot{y}3(p1+p2+p3+p4\ddot{y}1). \tag{11.4}$$

**Durch partielle Ableitungen für p1, p2, p3, p4 erhalten wir**

$$\frac{\ddot{y}L}{} = \ddot{y}\ln p1 \ddot{y} 1 + \ddot{y}1 + \ddot{y}2 + \ddot{y}3 = 0, \ddot{y}p1 \tag{11.5}$$

$$\frac{\ddot{y}L}{} = \ddot{y}lnp2\ \ddot{y}\ 1 + \ddot{y}1 + \ddot{y}3 = 0,\ \ddot{y}p2 \tag{11.6}$$

$$\frac{\ddot{y}L}{} = \ddot{y}lnp3\ \ddot{y}\ 1 + \ddot{y}2 + \ddot{y}3 = 0,\ \ddot{y}p3 \tag{11.7}$$

$$\frac{\ddot{y}L}{} = \ddot{y}lnp4\ \ddot{y}\ 1 + \ddot{y}3 = 0\ \ddot{y}p4 \tag{11.8}$$

und berechnen

$$(11.5)–(11.6): lnp2\ \ddot{y}\ lnp1 + \ddot{y}2 = 0, \tag{11.9}$$

$$(11.7)–(11.8): lnp4\ \ddot{y}\ lnp3 + \ddot{y}2 = 0, \tag{11.10}$$

woraus folgt, dass p3/p4 = p1/p2, was wir sofort in (7.12) auf Seite 129 einsetzen:

$$\frac{p2p3}{S.\ 4} + p2 + p3 + p4 = 1$$

$$\ddot{y}\ p2\ 1 + p4 \qquad \frac{p3}{} + p3 + p4 = 1 \tag{11.11}$$

$$(7.10)–(7.11): p2 = p3 + \ddot{y}\ \ddot{y}\ \ddot{y}$$

was, ersetzt durch p2, $(p3 + \ddot{y}\ \ddot{y}\ \ddot{y}\ )(1 +$ ergibt $\frac{p3}{S.\ 4}) + p3 + p4 = 1$

$$(7.10)\ in\ (7.12): \ddot{y} + p3 + p4 = 1. \tag{11.12}$$

Daher eliminieren wir p4 in (11.8), was ergibt

$$p3\ (p3 + \ddot{y}\ \ddot{y}\ \ddot{y}\ )\ 1 + \frac{}{1\ \ddot{y}\ \ddot{y}\ \ddot{y}\ p3} + 1\ \ddot{y}\ \ddot{y} = 1 \tag{11.13}$$

$$\ddot{y}\ (p3 + \ddot{y}\ \ddot{y}\ \ddot{y}\ )(1\ \ddot{y}\ \ddot{y}\ \ddot{y}\ p3 + p3) = \ddot{y}(1\ \ddot{y}\ \ddot{y}\ \ddot{y}\ p3) \tag{11.14}$$

$$\ddot{y}\ (p3 + \ddot{y}\ \ddot{y}\ \ddot{y}\ )(1\ \ddot{y}\ \ddot{y}) = \ddot{y}(1\ \ddot{y}\ \ddot{y}\ \ddot{y}\ p3) \tag{11.15}$$

$$\ddot{y}\ p3 + \ddot{y}\ \ddot{y}\ \ddot{y}\ \ddot{y}\ \ddot{y}p3\ \ddot{y}\ \ddot{y} \qquad ^{2} + \ddot{y}\ddot{y} = \ddot{y}\ \ddot{y}\ \ddot{y} \qquad 2\ \ddot{y}\ \ddot{y}p3 \tag{11.16}$$

$$\ddot{y}\ p3 = \ddot{y}\ (1\ \ddot{y}\ \ddot{y}). \tag{11.17}$$

Mit (11.12) folgt p4 = (1 ÿ ÿ)(1 ÿ ÿ ) und mit (11.11) erhalten wir p2 = ÿ(1 ÿ ÿ ) und p1 = ÿÿ .

**Aufgabe 7.9**

**(a)**

$$M = \begin{matrix} 0 & 1000 & 10 \\ 0 & & \end{matrix}.$$

**(B)**

$$M \cdot \begin{matrix} P \\ 1\ \ddot{y}\ S \end{matrix} = \begin{matrix} 1000 \cdot (1\ \ddot{y}\ p) \\ 10 \cdot p \end{matrix}.$$

Da die Entscheidung zwischen *Löschen* oder *Lesen* durch die Festlegung des Minimums der beiden Werte getroffen wird, reicht es aus, die beiden Werte zu vergleichen: 1000(1 ÿ p) < 10p, was p > 0,99 ergibt. Im Allgemeinen berechnen wir für eine Matrix $M = \begin{smallmatrix} 0 & k \\ 1 & 0 \end{smallmatrix}$ den Schwellenwert k/(k + 1).

**Aufgabe 7.10**

(a) Da A und B unabhängig sind, gilt P (A|B) = P (A) = 0,2. (b) Durch Anwendung der Konditionierungsregel (Seite 155) erhalten wir

$$P(C|A) = P(C|A,B)P(B)+P(C|A,\neg B)P(\neg B) = 0{,}2\cdot 0{,}9+0{,}1\cdot 0{,}1 = 0{,}19.$$

**Aufgabe 7.11**

(a)

$$\begin{aligned}
P(Al) &= P(Al,Bur,Ohr) + P(Al,\neg Bur,Ohr) \\
&\quad + P(Al,Bur,\neg Ohr) + P(Al,\neg Bur,\neg Ohr) \\
&= P(Al|Bur,Ohr)P(Bur,Ohr) + P(Al|\neg Bur,Ohr)P(\neg Bur,Ohr) \\
&\quad + P(Al|Bur,\neg Ear)P(Bur,\neg Ear) \\
&\quad + P(Al|\neg Bur,\neg Ohr)P(\neg Bur,\neg Ohr) \\
&= 0{,}95 \cdot 0{,}001 \cdot 0{,}002 + 0{,}29 \cdot 0{,}999 \cdot 0{,}002 + 0{,}94 \cdot 0{,}001 \cdot 0{,}998 \\
&\quad + 0{,}001 \cdot 0{,}999 \cdot 0{,}998 \\
&= 0{,}00252,
\end{aligned}$$

$$\begin{aligned}
P(J) &= P(J,Al) + P(J,\neg Al) = P(J|Al)P(Al) + P(J|\neg Al)P(\neg Al) \\
&= 0{,}9 \cdot 0{,}0025 + 0{,}05 \cdot (1 - 0{,}0025) = 0{,}052,
\end{aligned}$$

$$\begin{aligned}
P(M) &= P(M,Al) + P(M,\neg Al) = P(M|Al)P(Al) + P(M|\neg Al)P(\neg Al) = 0{,}7 \cdot 0{,}0025 \\
&\quad + 0{,}01 \cdot (1 - 0{,}0025) = 0{,}0117.
\end{aligned}$$

(B)

$$\begin{aligned}
P(Al|Bur) &= P(Al|Bur,Ohr)P(Ohr) + P(Al|Bur,\neg Ohr)P(\neg Ohr) \\
&= 0{,}95 \cdot 0{,}002 + 0{,}94 \cdot 0{,}998 = 0{,}94002,
\end{aligned}$$

$$\begin{aligned}
P(M|Bur) &= P(M,Bur)/P(Bur) = [P(M,Al,Bur) + P(M,\neg Al,Bur)] \\
&\quad /P(Bürer) \\
&= [P(M|Al)P(Al|Bur)P(Bur) + P(M|\neg Al)P(\neg Al|Bur)P(Bur)] \\
&\quad /P(Bürer) \\
&= P(M|Al)P(Al|Bur) + P(M|\neg Al)P(\neg Al|Bur) \\
&= 0{,}7 \cdot 0{,}94002 + 0{,}01 \cdot 0{,}05998 = 0{,}659.
\end{aligned}$$

(C)

$$P(Bur|M) = \frac{P(M|Bur)P(Bur)}{P(M)} = \frac{0{,}659 \cdot 0{,}001}{0{,}0117} = 0{,}056.$$

**(D)**

$$\mathbf{P}(Al|J,M) = \frac{\mathbf{P}(Al,J,M)}{\mathbf{P}(J,M)} = \frac{\mathbf{P}(Al,J,M)}{\mathbf{P}(Al,J,M)+\mathbf{P}(\neg Al,J,M)}$$

$$= \frac{1}{1+\frac{\mathbf{P}(\neg Al,J,M)}{\mathbf{P}(Al,J,M)}} = \frac{1}{\frac{\mathbf{P}(J|\neg Al)\mathbf{P}(M|\neg Al)\mathbf{P}(\neg Al)}{\mathbf{P}(J|Al)\mathbf{P}(M|Al)\mathbf{P}(Al)}+1}$$

$$= \frac{1}{1+\frac{0{,}05\cdot 0{,}01\cdot 0{,}9975}{0{,}9\cdot 0{,}7\cdot 0{,}0025}} = 0{,}761,$$

$$\mathbf{P}(J,M|Bur) = \mathbf{P}(J,M|Al)\mathbf{P}(Al|Bur) + \mathbf{P}(J,M|\neg Al)\mathbf{P}(\neg Al|Bur))$$

$$= \mathbf{P}(J|Al)\mathbf{P}(M|Al)\mathbf{P}(Al|Bur) + \mathbf{P}(J|\neg Al)\mathbf{P}(M|\neg Al)\mathbf{P}(\neg Al|Bur))$$

$$= 0{,}9\cdot 0{,}7\cdot 0{,}94 + 0{,}05\cdot 0{,}01\cdot 0{,}06 = 0{,}5922,$$

$$\mathbf{P}(Al|\neg Bur) = \mathbf{P}(Al|\neg Bur,Ohr)\mathbf{P}(Ohr) + \mathbf{P}(Al|\neg Bur,\neg Ohr)\mathbf{P}(\neg Ohr)$$

$$= 0{,}29\cdot 0{,}002 + 0{,}001\cdot 0{,}998 = 0{,}00158,$$

$$\mathbf{P}(J,M|\neg Bur) = \mathbf{P}(J,M|Al)\mathbf{P}(Al|\neg Bur) + \mathbf{P}(J,M|\neg Al)\mathbf{P}(\neg Al|\neg Bur))$$

$$= \mathbf{P}(J|Al)\mathbf{P}(M|Al)\mathbf{P}(Al|\neg Bur) + \mathbf{P}(J|\neg Al)\mathbf{P}(M|\neg Al)\mathbf{P}(\neg Al|\neg Bur))$$

$$= 0{,}9\cdot 0{,}7\cdot 0{,}00158 + 0{,}05\cdot 0{,}01\cdot 0{,}998 = 0{,}00149,$$

$$\mathbf{P}(J,M) = \mathbf{P}(J,M|Bur)\mathbf{P}(Bur) + \mathbf{P}(J,M|\neg Bur)\mathbf{P}(\neg Bur)$$

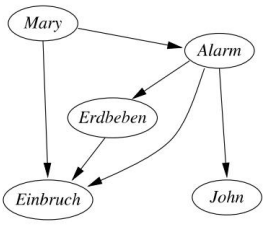
$$= 0{,}5922\cdot 0{,}001 + 0{,}00149\cdot 0{,}999 = 0{,}00208,$$

$$\mathbf{P}(Bur|J,M) = \frac{\mathbf{P}(J,M|Bur)\mathbf{P}(Bur)}{\mathbf{P}(J,M)} = \frac{0{,}5922\cdot 0{,}001}{0{,}00208} = 0{,}284.$$

**(e)** $\mathbf{P}(J)\mathbf{P}(M) = 0{,}052\cdot 0{,}0117 = 0{,}00061$, aber $\mathbf{P}(J,M) = 0{,}00208$ (siehe oben). Daher ist $\mathbf{P}(J)\mathbf{P}(M) = \mathbf{P}(J,M)$.

**(f)** Siehe Abschn. 7.4.5 auf Seite 149.

**(G)**




**(h)** Nur die CPT des Alarmknotens ändert sich. Alle anderen Knoten sind erdbebenunabhängig oder bei Erdbebenalarm unabhängig.

**(i)**

| *Bur* | $\mathbf{P}(Al)$ |
|---|---|
| T | 0,94 |
| F | 0,0016 |

Aufgrund der kontraproduktiven Variablenordnung sind auch andere Lösungen möglich. So lernen wir: Verwenden Sie immer eine Variablenreihenfolge, die die Kausalität berücksichtigt!

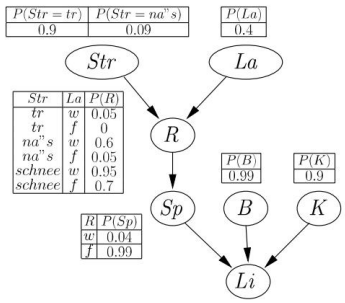



**Abb. 11.1 Bayesianisches Netzwerk für das Fahrradlicht. Die CPT für Li ist in der Aufgabe angegeben**
**Stellungnahme**

**Aufgabe 7.12**

**(a), (b) Siehe Abb. 11.1.**

**(c) Die Kante *(Str, Li)* fehlt, weil *Li* und *Str* bei gegebenem V bedingt unabhängig sind, also P *(Li*|*V , Str)* = P *(Li*|*V ),* weil der Straßenzustand keinen direkten Einfluss auf das Licht hat , sondern nur über den Dynamo und die von ihm erzeugte Spannung. (Die bedingte Unabhängigkeit *Li* und *Str* bei gegebenem V kann hier nicht durch Berechnung anhand der verfügbaren Daten für P *(Li*|*V , Str)* und P *(Li*|*V )* gezeigt, sondern nur behauptet werden.) (d) Für das Gegebene CPTs:**

$$\mathbf{P}(R|Str) = \mathbf{P}(R|Str,Flw)\mathbf{P}(Flw) + \mathbf{P}(R|Str,\neg Flw)\mathbf{P}(\neg Flw)$$
$$= 0{,}95 \cdot 0{,}4 + 0{,}7 \cdot 0{,}6 = 0{,}8,$$
$$\mathbf{P}(V|Str) = \mathbf{P}(V|R)\mathbf{P}(R|Str) + \mathbf{P}(V|\neg R)\mathbf{P}(\neg R|Str)$$
$$= 0{,}04 \cdot 0{,}8 + 0{,}99 \cdot 0{,}2 = 0{,}23.$$

## 11.8 Maschinelles Lernen und Data Mining

**Aufgabe 8.1**

**(a) Der Agent ist eine Funktion A, die den Vektor (t1,t2,t3, d1, d2, d3, f1, f2, f3) bestehend aus jeweils drei Werten für Temperatur, Druck und Feuchtigkeit einer Klasse** zuordnet **Wert k ÿ {sonnig, bewölkt, regnerisch}. (b) Die Trainingsdatendatei besteht aus einer Zeile des Formulars**

**ti1,ti2,ti3, di1, di2, di3, fi1, fi2, fi3, ki**

**für jedes einzelne Trainingsdatenelement. Der Index läuft über alle Trainingsdaten. Eine konkrete Datei könnte zum Beispiel beginnen:**

**17,17,17,980,983,986,63,61,50,sonnig**
**20,22,25,990,1014,1053,65,66,69, sonnig**
**20,22,18,990,979,929,52,60,61,regnerisch**

**Übung 8.2 Wir können die Symmetrie direkt anhand der Definition von Korrelationskoeffizienten oder der Kovarianz erkennen. Das Vertauschen von i und j ändert den Wert nicht. Für die**

**Diagonalelemente berechnen wir**

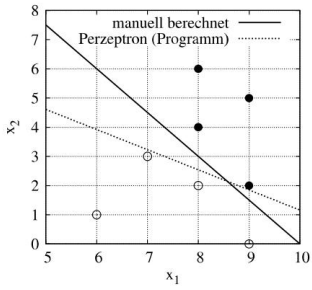
$$K_{ii} = \frac{\ddot{y}_{ii}}{s_i \cdot s_i} = \frac{\sum_{p=1}^{N} (x_p \ddot{y} \bar{x}_i)(x_p \ddot{y} \bar{x}_i)}{\sum_{(x_p \ddot{y} \bar{x}_i)\, p=1}^{N} {}^{2}} = 1.$$

**Aufgabe 8.3 Die Wertefolge für w ist**

**(1, 1), (2,2, ÿ0,4), (1,8, 0,6), (1,4, 1,6), (2,6, 0,2), (2,2, 1,2)**

**Übung 8.4**

**(A)**




**(b) Das Zeichnen einer geraden Linie im Diagramm ergibt:**

**1,5x1 + x2 ÿ 15 > 0.**

**Nach 442 Iterationen sind die Perzeptron-Lernalgorithmen mit dem Startvektor w = (1, 1, 1) kehrt zurück:**

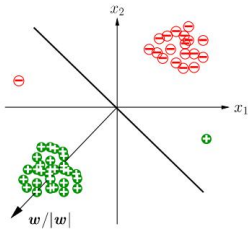
**w = (11, 16, ÿ129).**

**Dies entspricht: 0,69x1 +x2 ÿ8,06 > 0**

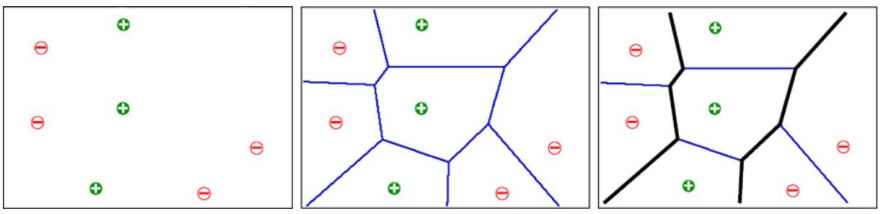
**Aufgabe 8.5**

**(a) Der Vektor $\ddot{y}_{M\ddot{y}}$ x zeigt in eine „durchschnittliche“ Richtung aller positiven Punkte und xÿM+ x zeigt in eine „durchschnittliche“ Richtung aller negativen Punkte. Die Differenz dieser Vektoren zeigt von den negativen Punkten zu den positiven Punkten. Die Trennlinie steht dann senkrecht auf diesem Differenzv**

**(b) Bei der Berechnung von w dominiert die Punktwolke der positiven und negativen Punkte. Die beiden Ausreißer spielen hier kaum eine Rolle. Für die Bestimmung der Trennlinie sind sie jedoch wichtig.**




**Übung 8.6**



**Aufgabe 8.7**

**(a) Nächster Nachbar ist (8, 4), also Klasse 1.**

**(b) k = 2: Klasse undefiniert, da eine Instanz von Klasse 1 und eine Instanz von Klasse 2.**
**k = 3: Entscheidung 2:1 für Klasse**
**0. k = 5: Entscheidung 2:3 für Klasse 1.**

**Aufgabe 8.8**

**(a) Um allgemeine Aussagen treffen zu können, müssen wir davon ausgehen, dass die Punkte gleichmäßig im Merkmalsraum verteilt sind, das heißt, dass die Anzahl der Punkte pro Fläche insgesamt annähernd gleich ist. Nun berechnen wir die Fläche Ad eines schmalen Reihenrings der Breite ÿd und des Abstands d um den Punkt x:**

$$Ad = ÿ(d + d)2 ÿ ÿd2 = ÿ(d2 + 2dd + d2 ) ÿ ÿd2 ÿ 2ÿdd.$$

**Das Gesamtgewicht aller Punkte im Ring der Breite d und des Abstands d ist somit proportional zu dwi = d/(1 + ÿd2 ) ÿ 1/(ÿd) für d ÿ ÿ. (b) Für diese Gewichtung ist das Gesamtgewicht aller Punkte im Ring der Breite d und = d/d = Abstand d wäre proportional zu dwÿ 1. Somit hätte jeder Ring der Breite d das gleiche Gewicht, unabhängig von seinem Abstand zum Punkt x. Dies ist sicherlich nicht sinnvoll, da die unmittelbare Umgebung für die Annäherung am wichtigsten ist.**

**Aufgabe 8.9** $\lim_{x\to 0} x \log_2 x = \lim_{x\to 0} \frac{\log_2 x}{1/x} = \lim_{x\to 0} \frac{1/(x \ln 2)}{-1/x^2} = 0$, **wobei für die zweite Gleichung die l'Hospitalsche Regel verwendet wurde.** $= \lim_{x\to 0} \frac{\ln x}{\ln 2}$ =

**Aufgabe 8.10 (a) 0 (b) 1 (c) 1,5 (d) 1,875 (e) 2,32 (f) 2**

**Aufgabe 8.11**

**(a) Aus log2 x = ln x/ ln 2 folgt c = 1/ ln 2 ÿ 1,44. (b) Da sich beide Entropiefunktionen nur um einen konstanten Faktor unterscheiden, ändert sich die Lage des Entropiemaximums nicht. Extrema unter Einschränkungen behalten ebenfalls die gleiche Position. Somit stellt der Faktor c für die Max-Ent-Methode kein Problem dar. Beim Lernen von Entscheidungsbäumen werden die Informationsgewinne verschiedener Attribute verglichen. Da hier nur die Reihenfolge zählt, nicht aber der absolute Wert, stellt der Faktor c auch hier kein Problem dar.**

**Aufgabe 8.12**

**(a) Für das erste Attribut berechnen wir**

$$G(D, x1) = H(D) ÿ \sum_{i=6}^{9} \frac{|Dx1=i|}{|D|} H(Dx1=i)$$

$$= 1 ÿ \frac{1}{8} H(Dx1=6) + \frac{1}{8} H(Dx1=7) + \frac{3}{8} H(Dx1=8) + \frac{3}{8} H(Dx1=9)$$

3 3 = 1 ÿ 0 + 0
+ · 0,918 + · 0,918 = 0,311 8 8 —

G(D, x2) = 0,75, also ist x2 ausgewählt. Für x2 = 0, 1, 3, 4, 5, 6 ist die Entscheidung klar. Für x2 = 2 wird x1 ausgewählt. Der Baum hat dann die Form

x2 = 0: 0 (1/0) x2 = 1: 0 (1/0)
x2 = 2:
| x1 = 6: 0 (0/0) | x1 = 7: 0 (0/0) | x1 = 8: 0 (1/0) | x1 = 9: 1 (1/0) x2 = 3: 0 (1/0) x2 = 4: 1 (1/0) x2 = 5: 1 (1/0)
x2 = 6: 1 (1/0)

**(b) Informationsgewinn für das kontinuierliche Attribut x2 als Wurzelknoten:**

| Schwelle ÿ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| G(D, x2 ÿ ÿ) | 0,138 | 0,311 | 0,189 | 0,549 | 0,311 | 0,138 |

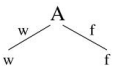
Da G(D, x2 ÿ 3) = 0,549 > 0,311 = G(D, x1), wird x2 ÿ 3 ausgewählt. Für x2 ÿ 3 ist die Klassifikation nicht eindeutig. Wir berechnen G(Dx2ÿ3, x1) = 0,322, G(Dx2ÿ3, x2 ÿ 0) = 0,073, G(Dx2ÿ3, x2 ÿ 1) = 0,17, G(Dx2ÿ3, x2 ÿ 2) = 0,073. Somit ist x1 ausgewählt. Für x1 = 9 ist die Klassifikation nicht eindeutig. Hier ist die Entscheidung eindeutig: G(D(x2ÿ3,x1=9), x1) = 0, G(D(x2ÿ3,x1=9), x2 ÿ 1) = 1, x2 ÿ 1 wird gewählt, und Der Baum ist wieder zu 100 % korrekt.

Baum: x2
<= 3: | x1 = 6: 0 (1/0) | x1 = 7: 0 (0/0) | x1 = 8: 0 (1/0) | x1 = 9: | | x2 <= 1: 0 (1/0) | | x2 > 1: 1 (1/0) x2 > 3: 1 (3/0)

**Übung 8.13**

**(a) 100 % für die Trainingsdaten, 75 % für die Testdaten, 80 % Gesamtkorrektheit.**

**(b) (A ÿ ¬C) ÿ (¬A ÿ B) (c)**




66,6 % Korrektheit für die Trainingsdaten,
100 % Korrektheit für die Testdaten,
80 % Gesamtkorrektheit

**Aufgabe 8.14**

**(a) Gleichung (8.7) auf Seite 189 zur Berechnung des Informationsgewinns eines Attributs A liest**

$$\text{InfoGain}(D,A) = H(D) - \sum_{i=1}^{N} \frac{|Di|}{|D|} H(Di),$$

**Dabei ist n die Anzahl der Werte des aktuell beobachteten Attributs. Wenn das Attribut A irgendwo im Teilbaum als Nachfolger des Wertes aj** getestet wird, dann **kommt nur der Wert A = aj** im Datensatz **Dj und jeder seiner Teilmengen Dÿ** vor . **Also Dÿ = Dÿ** . **und für alle k = j , |Dÿ j** $_k$ **| = 0 und wir erhalten**

$$\text{InfoGain}(D\ddot{y},A) = H(D\ddot{y}) - \sum_{i=1}^{N} \frac{|D\ddot{y}_i|}{|D\ddot{y}|} H(D\ddot{y}_i) = H(D\ddot{y}_J) - \frac{|D\ddot{y}|_J}{|D\ddot{y}|_J} H(D\ddot{y}_J) = 0.$$

**Das wiederholte Attribut hat somit einen Informationsgewinn von Null, weshalb es nicht mehr**

**verwendet wird. (b) Aus jedem kontinuierlichen Attribut A wird ein binäres Attribut A>ÿD,A generiert. Wenn weiter unten im Baum das Attribut A erneut mit einem anderen Schwellenwert ÿDÿ ,A diskretisiert wird, dann unterscheidet sich** das Attribut A>ÿDÿ ,A von **A>ÿD,A. Hat es dann einen höheren Informationsgewinn als alle anderen Attribute, wird es im Baum verwendet. Dies bedeutet jedoch auch, dass bei mehreren Vorkommen eines kontinuierlichen Attributs die Schwellenwerte unterschiedlich sein müssen.**

**Übung 8.15**

**(a) P *(Himmel = klar)* = 0,65 P *(Balken = steigend)* = 0,67**

| *Himmel* | *Bar* | P *(Prec = dry\|Sky,Bar)* |
|---|---|---|
| Klar | Steigend | 0,85 |
| Klar | Fallend | 0,44 |
| Bewölkt | Fallend | 0,45 |
| Bewölkt | Fallend | 0,2 |

**(b) P *(Himmel = klar)* = 0,65**

| *Bar* | P *(Prec = dry\|Bar)* |
|---|---|
| Steigend | 0,73 |
| Fallend | 0,33 |

| *Himmel* | P *(Balken = steigend\|Himmel)* |
|---|---|
| Klar | 0,72 |
| Bewölkt | 0,57 |

**(c) Die notwendigen CPTs für P *(Prec|Sky,Bar)* und P *(Bar|Sky)* sowie P *(Sky)* sind bereits bekannt.**

**(d) Pa = (0,37, 0,065, 0,095, 0,12, 0,11, 0,13, 0,023, 0,092)**
**Pb = (0,34, 0,13, 0,06, 0,12, 0,15, 0,054, 0,05, 0,1)**
**P = (0,4, 0,07, 0,08, 0,1, 0,09, 0,11, 0,03, 0,12) (ursprüngliche Verteilung) Quadratisch**

**Abstand: dq (Pa,P) = 0,0029, dq (Pb,P) = 0,014 Kullback-Leibler-Abstand: dk(Pa,P) = 0,017, dk(Pb,P) = 0,09 Beide Abstandsmetriken zeigen, dass das Netzwerk ( a) nähert sich der Verteilung besser an als Netzwerk (b). Dies bedeutet, dass die Annahme, dass *Prec* und *Sky*** bei gegebenem ***Bar* bedingt unabhängig sind, weniger wahrscheinlich ist als die Annahme, dass *Sky* und *Bar* unabhängig sind.**

**(e) Pc = (0,4, 0,07, 0,08, 0,1, 0,09, 0,11, 0,03, 0,12). Diese Verteilung ist genau gleich der ursprünglichen Verteilung P. Dies ist nicht überraschend, da im Netzwerk keine Kanten fehlen. Dies bedeutet, dass keine Unabhängigkeiten angenommen wurden.**

**Übung 8.16 Wir können sofort erkennen, dass Scores und Perzeptrone äquivalent sind, indem wir ihre Definitionen vergleichen. Nun zur Äquivalenz zu naivem Bayes: Zuerst stellen wir fest, dass P (K|S1,...,Sn) > 1/2 äquivalent zu P (K|S1,...,Sn) > P (¬K|S1)** ist ,...,Sn) , **weil P (¬K|S1,...,Sn) > 1 ÿ P (K|S1,...,Sn). Tatsächlich haben wir es hier mit einem binären naiven Bayes-Klassifikator zu tun.**

**Wir wenden den Logarithmus auf die naive Bayes-Formel an**

$$\mathbf{P(K|S1,...,Sn) = P\frac{P(S1|K)\cdots P(Sn|K)\cdot P(K)}{(S1,...,Sn)}},$$

**und erhalten**

$$\mathbf{\log P(K|S1,...,Sn) = \log P(S1|K) + \cdots + \log P(Sn|K) + \log P(K) \ddot{y} \log P(S1,...,Sn).} \quad \mathbf{(11.18)}$$

Um einen Score zu erhalten, müssen wir die Variablen **S1,...,Sn als numerische Variablen mit den Werten 1 und 0 interpretieren.** Das können wir leicht erkennen

$$\mathbf{\log P(Si|K) = (\log P(Si = 1|K) \ddot{y} \log P(Si = 0|K))Si + \log P(Si = 0|K).}$$

**Es folgt dem**

$$\mathbf{\sum_{i=1}^{N} \log P(Si|K) = \sum_{i=1}^{N} (\log P(Si = 1|K) \ddot{y} \log P(Si = 0|K))Si + \sum_{i=1}^{N} \log P(Si = 0|K).}$$

**Nun definieren wir $w_i = \log P(Si = 1|K) \ddot{y} \log P(Si = 0|K)$ und $c = \sum_{i=1}^{n} \log P(Si = 0|K)$ und vereinfachen**

$$\mathbf{\sum_{i=1}^{N} \log P(Si|K) = \sum_{i=1}^{N} wiSi + c.}$$

**Eingesetzt in (11.18) erhalten wir**

$$\mathbf{\log P(K|S1,...,Sn) = \sum_{i=1}^{N} wiSi + c + \log P(K) \ddot{y} \log P(S1,...,Sn).}$$

Für die Entscheidung K muss nach der Definition des Bayes-Klassifikators gelten, dass logP (K|S1,...,Sn) > log(1/2). Es muss also entweder so sein

$$\sum_{i=1}^{N} wiSi + c + \log P(K) \ddot{y} \log P(S1,...,Sn) > \log(1/2)$$

oder das

$$\sum_{i=1}^{N} wiSi > \log 1/2 \ddot{y} c \ddot{y} \log P(K) + \log P(S1,...,Sn),$$

womit wir einen Score mit dem Schwellenwert ÿ = log 1/2 ÿ c ÿ logP (K) + logP (S1,...,Sn) definiert haben. Da alle Transformationen rückgängig gemacht werden können, können wir auch jeden beliebigen Score in einen Bayes'schen Klassifikator umwandeln. Damit ist die Äquivalenz nachgewiesen.

Übung 8.17 Die Logarithmierung von (8.10) auf Seite 206 ergibt

$$\log P(l|s1,...,sn) = \log c + \log P(l) + \sum_{i=1}^{l} ni \log P(wi|l).$$

Dadurch werden aus sehr kleinen negativen Zahlen moderate negative Zahlen. Da die Logarithmusfunktion monoton wächst, maximieren wir zur Bestimmung der Klasse gemäß der Regel

$$\text{INaive-Bayes} = \underset{l \ddot{y} \{w,f\}}{\operatorname{argmax}} \log P(l|s1,...,sn).$$

Der Nachteil dieser Methode ist die etwas längere Rechenzeit in der Lernphase bei großen Texten. Beim Klassifizieren erhöht sich die Zeit nicht, da die Werte logP (l | s1,...,sn) beim Lernen gespeichert werden können.

Aufgabe 8.19 Sei f streng monoton wachsend, also ÿx,yx < y ÿ f (x) < f (y). Wenn nun d1(s, t) < d1(u, v), dann ist eindeutig d2(s, t) = f (d1(s, t)) < f (d1(u, v)) = d2(u, v ). Da auch die Umkehrung von f streng monoton ist, gilt das Gegenteil, d. h. d2(s, t) < d2(u, v) ÿ d1(s, t) < d1(u, v). Somit wurde gezeigt, dass d2(s, t) < d2(u, v) ÿ d1(s, t) < d1(u, v).

Übung 8.20

23·26 x1x2 = 4, und somit ds(x1, x2) = = ~~6,11 4~~ ÿ 26·20

= 2, und somit ds(x2, x3) = = 11,4 x2x3

23·20 x1x3 = 1, und somit ds(x1, x3) = = ~~21,4 1~~ 2 ÿ

Die Sätze 1 und 2 sind hinsichtlich der Distanzmetrik ds am ähnlichsten .

**Übung 8.21 Hilfe bei Problemen mit KNIME: www.knime.org/forum**

## 11.9 Neuronale Netze

**Aufgabe 9.1 Wir wollen zeigen, dass f (ÿ + x) + f (ÿ ÿ x) = 1.**

$$f(\ddot{y} + x) = 1 + \frac{1}{e^{\frac{x}{T}}} = \frac{T e^{\frac{x}{T}}}{1 + e^{\frac{x}{T}}}, \qquad f(\ddot{y} \ddot{y} x) = 1 + \frac{1}{e^{\frac{x}{T}}},$$

$$f(\ddot{y} + x) + f(\ddot{y} \ddot{y} x) = 1.$$

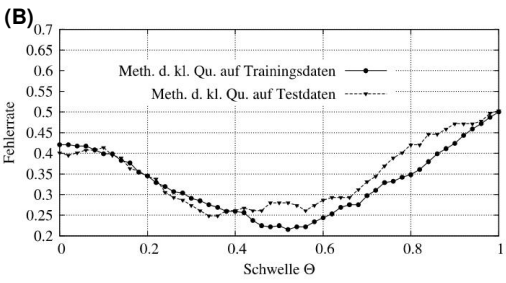
**Aufgabe 9.3 Jedes im Netzwerk gespeicherte Muster hat eine Größe von n Bits. Das Netzwerk hat insgesamt n(n ÿ 1)/2 Gewichte. Wenn wir 16 Bits pro Gewicht reservieren und den Binärspeicher der Größe 16n(nÿ1)/2 als gleich groß definieren, dann können damit eindeutig N = 8n(nÿ1)/n = 4(n ÿ 1) Muster n Bits gespeichert werden in Größe. Für große n erhalten wir N = Bildet man wie in (9.11) auf Seite 237 den Quotienten ÿ aus der Anzahl der speicherbaren Bits und der Anzahl der verfügbaren Speicherzellen, so erhält man für den Listenspeicher 2 den Wert 1 und den Wert ÿ ≈ 0,146n (= 16n(n ÿ 1)/2) ÿ 0,018 für das Hopfield-Netzwerk. Der klassische Speicher hat somit (bei 16 Bit pro Gewicht) eine etwa 55-mal höhere Kapazität.**

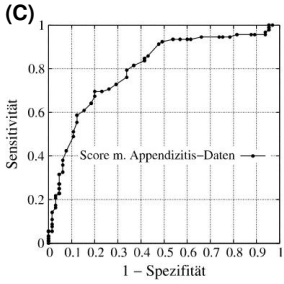
**Übung 9.4**

**(a) Mathematica-Programm für die Methode der kleinsten Quadrate.**

```
LeastSq[q_,a_] := Module[{Nq,Na,m,A,b,w}, Nq = Länge[q]; m
        = Länge[q[[1]]]; Na = Länge[a]; If[Nq != Na, Print["Length[q] != Länge[a]"];
        Ausgang, 0]; A = Tabelle[N[Summe[q[[p,i]] q[[p,j]], {p,1,Nq}]], {i,1,m}, {j,1,m} ]; b =
        Table[N[Sum[a[[p]] q[[p,j]], {p,1,Nq}]], {j,1,m}]; w = LinearSolve[A,b]

]
LeastSq::usage = "LeastSq[x,y,f] berechnet aus den Abfragevektoren q[[1]],..., q[[m]] eine Tabelle mit Koeffizienten
    w[[i]] für eine lineare Abbildung f[x] = Summe[w[[i]] x[[i]], {i,1,m}] mit f[q[[p]]] = a[[p]].“
```




**Aufgabe 9.6 (a) Lernen funktioniert fehlerfrei. (b) Lernen funktioniert nicht ohne Fehler!**

**Aufgabe 9.7**

**(a) Eine Abbildung f heißt linear, wenn für alle x,y,k gilt: f (x + y) = f (x) + f (y) und f (kx) = kf (x) . Seien nun f und g lineare Abbildungen.**
**Dann ist f (g(x + y)) = f (g(x) + g(y)) = f (g(x)) + f (g(y)) und f (g(kx)) = f ( kg(x)) = kf (g(x)). Somit handelt es sich bei aufeinanderfolgenden Ausführungen linearer Abbildungen um eine lineare Abbildung.**

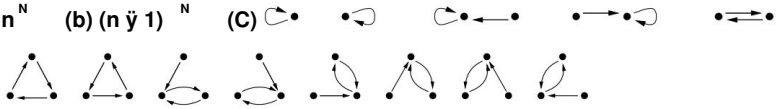
**(b) Wir beobachten zwei beliebige Ausgabeneuronen j und k. Jeder der beiden repräsentiert eine Klasse. Die Klassifizierung erfolgt durch Bildung des sei die Maximums der beiden Aktivierungen. i wj ixi und *netk* = i wkixi gewichtete Summe der Werte ar. Es sei *netj* = riving an den Neuronen j und k. Sei außerdem *netj* > *netk* . Ohne Aktivierungsfunktion wird Klasse j ausgegeben. Wenn nun eine streng monotone Aktivierungsfunktion f angewendet wird, ändert sich am Ergebnis nichts, da aufgrund der streng monotonen Funktion f ( netj ) > *f(netk) gilt.***

**Aufgabe 9.8 Der transformierte** $\overset{2}{i}$ **, f2(x1, x2) =** $x_{2}^{2}$ **. Dann ist die Trennlinie in der Raum f1(x1, x2)** = x hat die Gleichung **y1 + y2 = 1.**

## 11.10 Verstärkungslernen

**Aufgabe 10.1**

**(a)** $n^{N}$ **(b)** $(n \ddot{y} 1)^{N}$ **(C)**



**(D)**

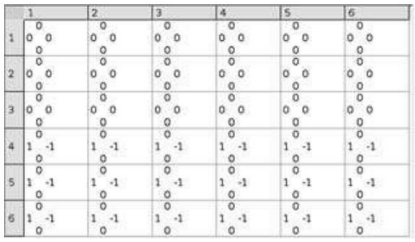


**Aufgabe 10.2 Werteiteration mit zeilenweiser Aktualisierung der V^-Werte von links oben nach rechts unten ergibt**

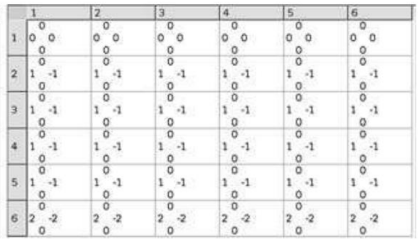
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **0** | **0** | ÿ | **0,81 0,9** | ÿ | **1,35 1,49** | ÿ···ÿ | **2,36 2,62** |
| **0** | **0** | | **0,73 1** | | **1,21 1,66** | | **2,12 2,91** |

**Übung 10.3**

**(C)**



6 × 6 feedback for a completely smooth surface



6 × 6 feedback for a very heavy survace (like thick snow)

(d) Wir sehen, dass der Wert ÿ umso näher an 1 liegen muss, je länger eine Richtlinie wird (d. h. je mehr Schritte beispielsweise ein Zyklus einer zyklischen Strategie hat), da ein höherer Wert für ÿ zu einem längeren Gedächtnis führt möglich. Die Werteiteration konvergiert jedoch viel langsamer.

Aufgabe 10.4 Der Wert V folgt: $^{ÿ}$ (3, 3) unten rechts in der Zustandsmatrix wird geändert als

$$V^{ÿ}(3,1) = 0{,}9\,V^{ÿ}(2,1) = 0{,}9\,2V\,ÿ\,(2,2) = 0{,}9\,3V\,ÿ\,(2,3) = 0{,}9\,4V\,ÿ\,(3,3). \quad (11.19)$$

Diese Gleichungskette folgt aus (10.6), weil für alle gegebenen Zustandsübergänge die maximale unmittelbare Belohnung r(s, a) = 0 ist, und das ist auch der Fall

$$V^{ÿ}(s) = \max_{A}[r(s,a) + ÿ\,V^{ÿ}(ÿ(s,a))] = ÿ\,V^{ÿ}(ÿ(s,a)) = 0{,}9V^{ÿ}(ÿ(s,a)).$$

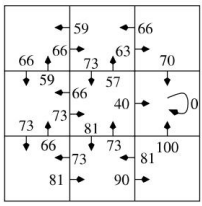
Aus (10.6) folgt auch, dass V maximal $^{ÿ}$ (3, 2) = 1 + 0,9 V $^{ÿ}$ (3, 1), weil r(s, a) = 1 ist. Analog gilt, dass V $^{ÿ}$ (3, 3) = 1 + 0,9 V $^{ÿ}$ (3, 2) und der Kreis schließt. Die beiden letzten Gleichungen zusammen $^{ÿ}$ (3, 3) = 1 + 0,9(1 + 0,9V $^{ÿ}$ (3, 1)). Aus (11.19) folgt, dass V folgt $^{ÿ}$ ergeben V (3, 1) = 0,9 4V ÿ (3, 3). Ersetzt in V $^{ÿ}$ (3, 3), dies

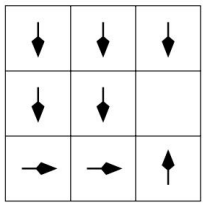
$$V^{ÿ}(3,3) = 1 + 0{,}9(1 + 0{,}9\,5V\,ÿ\,(3,3)),$$

woraus sich die Behauptung ergibt.

Übung 10.5

Stabile Q-Werte und eine optimale Politik:



Aufgabe 10.6

(a) Anzahl der Zustände = ÿ $^{N}$ . Anzahl der Aktionen pro Zustand = ÿ $^{N}$ ÿ 1. Anzahl der Pol cies = (ÿn ) $^{ÿ^{N}}$ nÿn = ÿ .

(B)

| | l = 1 | l = 2 | l = 3 | l = 4 | l = 10 |
|---|---|---|---|---|---|
| n = 1 | 1 | 4 | 27 | 256 | 1010 |
| n = 2 1 | | 256 | 3,9 × 108 | 1,8 × 1019 | 10200 |
| n = 31 1,7 × 107 n = 41 1,8 × | | | 4,4 × 1038 | 3,9 × 10115 | 103000 |
| 1019 3,9 × 10154 n = 81 3,2 × 10616 1,4 × | | | | 3,2 × 10616 | 1040000 |
| 1025043 6,7 × 10315652 | | | | | 10800000000 |

(c) Pro Staat gibt es nun 2n mögliche Aktionen. Es gibt also $(2n)^N$ Richtlinien.

| | l = 1 | l = 2 | l = 3 | l = 4 | l = 10 |
|---|---|---|---|---|---|
| n = 1 | 2 | 4 | 8 | 16 | 1024 |
| n = 2 | 4 | 256 | | | |
| n = 3 | 6 | 1,7 × 1021 | 2,6 × 1049 | 4,3 × 1073 | 1,6 × 10778 |
| n = 4 | 8 | 2,8 × 1031 | 1,8 × 10231 | 1,0 × 107900 | 1,6 × 109030 |
| n = 8 | 16 | 1,7 × 1078913 | | | 1,8 × 10120411998 |

(d) 10120411998 verschiedene Richtlinien können niemals kombinatorisch untersucht werden, selbst wenn alle der verfügbaren Computer der Welt sollten parallel darauf laufen.
Daher sind „intelligente“ Algorithmen erforderlich, um ein optimales oder nahezu optimales Ergebnis zu finden Politik.

# Verweise


[Ada75] EW Adams. *Die Logik der Bedingungen.* Synthese Library, Band 86. Reidel, Dordrecht, 1975.

[Alp04] E. Alpaydin. *Einführung in maschinelles Lernen.* MIT Press, Cambridge, 2004.

[APR90] J. Anderson, A. Pelionisz und E. Rosenfeld. *Neurocomputing (Band 2): Wegbeschreibung für die Forschung.* MIT Press, Cambridge, 1990.

[AR88] J. Anderson und E. Rosenfeld. *Neurocomputing: Grundlagen der Forschung.* MIT Press, Cambridge, 1988. Sammlung grundlegender Originalarbeiten.

[Bar98] R. Bartak. Online-Leitfaden zur Constraint-Programmierung, 1998. http://kti.ms.mff.cuni.cz/~bartak/einschränkungen.

[BCDS08] A. Billard, S. Calinon, R. Dillmann und S. Schaal. Roboterprogrammierung durch Demonstration. In B. Siciliano und O. Khatib, Herausgeber, *Handbook of Robotics,* Seiten 1371–1394. Springer, Berlin, 2008.

[Bel57] RE Bellman. *Dynamische Programmierung.* Princeton University Press, Princeton, 1957.

[Ber89] M. Berrondo. Fallgruben für Kopffüssler. Fischer Taschenbuch, Nr. 8703, 1989.

[BFOS84] L. Breiman, J. Friedman, RA Olshen und CJ Stone. *Klassifikations- und Regressionsbäume.* Wadsworth, Belmont, 1984.

[Bib82] W. Bibel. *Automatisierte Beweisführung von Theoremen.* Vieweg, Wiesbaden, 1982.

[Bis05] CM Bischof. *Neuronale Netze zur Mustererkennung.* Oxford University Press, London, 2005.

[Bis06] CM Bischof. *Mustererkennung und maschinelles Lernen.* Springer, New York, 2006.

[BM03] AG Barto und S. Mahadevan. Jüngste Fortschritte beim hierarchischen Verstärkungslernen. *Discrete Event Syst., Sonderausgabe zum Reinforcement Learning,* 13:41–77, 2003.

[Bra84] V. Braitenberg. *Fahrzeuge – Experimente in der Synthetischen Psychologie.* MIT Press, Cam Bridge, 1984.

[Bra01] B. Brabec. Computergestützte regionale Lawinenprognose. Doktorarbeit, ETH Zürich, 2001.

[Bra11] I. Bratko. *PROLOG-Programmierung für künstliche Intelligenz.* Addison-Wesley, Reading, 4. Auflage, 2011.

[Bri91] *Enzyklopädie Britannica.* Encyclopedia Britannica, London, 1991.

[Bur98] CJ Burges. Ein Tutorial zu Support-Vektor-Maschinen zur Mustererkennung. *Daten Mindest. Wissen. Discov.,* 2(2):121–167, 1998.

[CAD] CADE: Konferenz zum Thema Automatisierte Deduktion. www.cadeconference.org.

[Che83] P. Cheeseman. Eine Methode zur Berechnung verallgemeinerter Bayes'scher Wahrscheinlichkeitswerte für Expertensysteme. In *Proc. des 8. Intl. Gemeinsame Konferenz. über künstliche Intelligenz* (IJCAI 83), 1983.

[Che85] P. Cheeseman. Zur Verteidigung der Wahrscheinlichkeit. In *Proc. des 9. Intl. Gemeinsame Konferenz. An Künstliche Intelligenz* (IJCAI-85), 1985.

[CL73] CL Chang und RC Lee. *Beweisen symbolischer Logik und mechanischer Theoreme.* Academic Press, Orlando, 1973.




W. Ertel, *Einführung in die Künstliche Intelligenz,*
Bachelor-Themen in Informatik,
DOI 10.1007/978-0-85729-299-5, 

[Cle79] WS Cleveland. Robuste lokal gewichtete Regression und glättende Streudiagramme. *Marmelade. Stat. Assoc.,* 74(368):829–836, 1979.

[CLR90] T. Cormen, Ch. Leiserson und R. Rivest. *Einführung in Algorithmen.* MIT Press, Cambridge, 1990.

[CM94] WF Clocksin und CS Mellish. *Programmieren in Prolog.* Springer, Berlin, 4 Auflage, 1994.

[Coz98] FG Cozman. Javabayes, Bayesianische Netzwerke in Java, 1998. www.cs.cmu.edu/~javabayes.

[dD91] FT de Dombal. *Diagnose akuter Bauchschmerzen.* Churchill Livingstone, London, 1991.

[dDLS+72] FT de Dombal, DJ Leaper, JR Staniland, AP McCann und JC Horrocks. Computergestützte Diagnose akuter Bauchschmerzen. *Br. Med. J.,* 2:9–13, 1972.

[Dee11] Das DeepQA-Projekt, 2011. http://www.research.ibm.com/deepqa/deepqa.shtml.

[DHS01] RO Duda, PE Hart und DG Stork. *Musterklassifizierung.* Wiley, New York, 2001.

[Dia04] D. Diaz. GNU-PROLOG. Universität Paris, 2004. Aufl. 1.7, für GNU Prolog-Version 1.2.18, http://gnu-prolog.inria.fr.

[DNM98] CL Blake, DJ Newman, S. Hettich und CJ Merz. UCI-Repository der Maschine Lerndatenbanken, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[Ede91] E. Eder. *Relative Komplexität von Kalkülen erster Ordnung.* Vieweg, Wiesbaden, 1991.

[Elk93] C. Elkan. Der paradoxe Erfolg der Fuzzy-Logik. In *Proceedings of the Eleventh National Conference on Artificial Intelligence* (AAAI-93), Seiten 698–703. MIT Press, Cambridge, 1993.

[Ert93] W. Ertel. *Parallele Suche mit randomisiertem Wettbewerb in Inferenzsystemen.* DISKI, Band 25. Infix, St. Augustin, 1993. Dissertation, Technische Universität München.

[Ert11] W. Ertel. *Künstliche Intelligenz,* 2011. www.hs-weingarten.de/~ertel/aibook. Homepage zu diesem Buch mit Materialien, Demoprogrammen, Links, Literatur, Errata usw.

[ES99] W. Ertel und M. Schramm. Kombination von Daten und Wissen durch MaxEnt-Optimierung von Wahrscheinlichkeitsverteilungen. In PKDD'99 *(3. Europäische Grundsatzkonferenz und Praxis der Wissensentdeckung in Datenbanken).* LNCS, Band 1704, Seiten 323–328. Springer, Prag, 1999.

[ESCT09] W. Ertel, M. Schneider, R. Cubek und M. Tokic. Der Lehrkasten: ein Universelles Roboter-Lernrahmen. In *Proceedings der 14. Internationalen Konferenz on Advanced Robotics* (ICAR 2009), 2009. www.servicerobotik.hs-weingarten.de/Lehrbox.

[ESS89] W. Ertel, J. Schumann und Ch. Suttner. Lernheuristik für einen Theorembeweis unter Verwendung der Rückausbreitung. In J. Retti und K. Leidlmair, Herausgeber, *5. Österreichische Artificial-Intelligence-Tagung.* Informatik-Fachberichte, Band 208, Seiten 87–95. Springer, Berlin, 1989.

[Fit96] M. Passend. *Logik erster Ordnung und automatisierter Theorembeweis.* Springer, Berlin, 1996.

[FNA+09] D. Ferrucci, E. Nyberg, J. Allan, K. Barker, E. Brown, J. Chu-Carroll, A. Ciccolo, P. Duboue, J. Fan, D. Gondek et al. Zur offenen Weiterentwicklung der Frage Antwortsysteme. IBM Technischer Bericht RC24789, Yorktown Heights, NY, 2009. http://www.research.ibm.com/deepqa/question_answering.shtml.

[FPP07] D. Freedman, R. Pisani und R. Purves. *Statistiken.* Norton, New York, 4. Auflage, 2007.

[Fra05] C. Frayn. Theorie der Computerschachprogrammierung, 2005. www.frayn.net/beowulf/theory.html.

[Fre97] E. Freuder. Auf der Suche nach dem Heiligen Gral. *Einschränkungen,* 2(1):57–61, 1997.

[FS97] B. Fischer und J. Schumann. Setheo geht in die Softwareentwicklung: Anwendung von ATP zur Wiederverwendung von Software. In *der Konferenz zum automatisierten Abzug* (CADE-14). LNCS, Band 1249, Seiten 65–68. Springer, Berlin, 1997. http://ase.arc.nasa.gov/people/schumann/publications/papers/cade97-reuse.html.

[GW08] RC González und RE Woods. *Digitale Bildverarbeitung.* Pearson/Prentice Hall, Upper Saddle River/Princeton, 2008.

[Göd31a] K. Gödel. Diskussion zur Grundlegung der Mathematik, Erkenntnis 2. *Monatshefte Mathematik. Phys.,* 32(1):147–148, 1931.

[Göd31b] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und veränderter Systeme I. *Monatshefte Math. Phys.,* 38(1):173–198, 1931.

[HKP91] J. Hertz, A. Krogh und R. Palmer. *Einführung in die Theorie der neuronalen Berechnung.* Addison-Wesley, Reading, 1991.

[Hon94] B. Hontschik. *Theorie und Praxis der Appendektomie.* Mabuse, Frankfurt am Main, 1994.

[Hop82] JJ Hopfield. Neuronale Netze und physikalische Systeme mit entstehenden kollektiven Rechenfähigkeiten. *Proz. Natl. Acad. Wissenschaft. USA,* 79:2554–2558, 1982. Nachdruck in [AR88], Seiten 460–464.

[HT85] JJ Hopfield und DW Tank. „Neuronale“ Berechnung von Entscheidungen in der Optimierung Probleme. *Biol. Cybern.,* 52(3):141–152, 1985. Springer.

[HTF09] T. Hastie, R. Tibshirani und J. Friedman. *Die Elemente des statistischen Lernens: Daten Bergbau, Schlussfolgerung und Vorhersage.* Springer, Berlin, 3. Auflage, 2009. Online-Version: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[Jay57] ET Jaynes. Informationstheorie und statistische Mechanik. *Physik. Rev.,* 1957.

[Jay03] ET Jaynes. *Wahrscheinlichkeitstheorie: Die Logik der Wissenschaft.* Cambridge University Press, Cambridge, 2003.

[Jen01] FV Jensen. *Bayesianische Netzwerke und Entscheidungsgraphen.* Springer, Berlin, 2001.

[Jor99] MI Jordan, Herausgeber. *Lernen in grafischen Modellen.* MIT Press, Cambridge, 1999.

[Kal01] JA Kalman. *Automatisiertes Denken mit OTTER.* Rinton, Paramus, 2001. www-unix.mcs.anl.gov/AR/otter/index.html.

[Kan89] Th. Kane. Maximale Entropie in Nilssons probabilistischer Logik. In *Proc. vom 11 International Gemeinsame Konferenz. über künstliche Intelligenz* (IJCAI-89), 1989.

[KK92] JN Kapur und HK Kesavan. *Prinzipien der Entropieoptimierung mit Anwendungen.* Academic Press, San Diego, 1992.

[KLM96] LP Kaelbling, ML Littman und AP Moore. Reinforcement Learning: eine Umfrage. *J. Artif. Intel. Res.,* 4:237–285, 1996. www-2.cs.cmu.edu/afs/cs/project/jair/pub/Band 4/kaelbling96a.pdf.

[KMK97] H. Kimura, K. Miyazaki und S. Kobayashi. Verstärkungslernen in POMDPs mit Funktionsnäherung. In *14. International Conference on Machine Learning,* Seiten 152–160. Morgan Kaufmann, San Mateo, 1997. http://sysplan.nams.kyushu-u.ac.jp/gen/papers/JavaDemoML97/robodemo.html.

[Koh72] T. Kohonen. Korrelationsmatrixspeicher. *IEEE Trans. Comput.,* C-21(4):353–359, 1972. Nachdruck in [AR88], Seiten 171–174.

[Lar00] FD Laramée. Schachprogrammierung, Teile 1–6, 2000. www.gamedev.net/reference/Programmierung/Funktionen/Schach1.

[Le999] LEXMED – ein lernendes Expertensystem für die medizinische Diagnose, 1999. www.lexmed.de.

[Lov78] DW Loveland. *Automatisierte Theoremprüfung: Eine logische Grundlage.* Nordholland, Amsterdam, 1978.

[LSBB92] R. Letz, J. Schumann, S. Bayerl und W. Bibel. SETHEO: ein leistungsstarker Theorembeweiser. *J. Autom. Reason.,* 8(2):183–212, 1992. www4.informatik.tu-muenchen.de/~letz/setheo.

[McC] W. McCune. Automatisierte Abzugssysteme und -gruppen. www-unix.mcs.anl.gov/AR/andere.html. Siehe auch www-formal.stanford.edu/clt/ARS/systems.html.

[McD82] J. McDermott. R1: ein regelbasierter Konfigurator von Computersystemen. *Artif. Intellektuell,* 19:39–88, 1982.

[MDBM00] G. Melancon, I. Dutour und G. Bousque-Melou. Zufällige Generierung von Dags für Diagrammzeichnung. Technischer Bericht INS-R0005, Niederländisches Forschungszentrum für Mathematik und Informatik (CWI), 2000. http://ftp.cwi.nl/CWIreports/INS/INS-R0005.pdf.

[Mit97] T. Mitchell. *Maschinelles Lernen.* McGraw-Hill, New York, 1997. www-2.cs.cmu.edu/~tom/mlbook.html.

[MP69] M. Minsky und S. Papert. *Perzeptrone.* MIT Press, Cambridge, 1969.

[Neu00] M. Neugeborenes. *Automatisierte Beweisführung von Theoremen: Theorie und Praxis.* Springer, Berlin, 2000.

[Nil86] NJ Nilsson. Wahrscheinlichkeitslogik. *Artif. Intell.,* 28(1):71–87, 1986.

[Nil98] N. Nilsson. *Künstliche Intelligenz – eine neue Synthese.* Morgan Kaufmann, San Mateo, 1998.

[NPW02] T. Nipkow, LC Paulson und M. Wenzel. *Isabelle/ HOL – Ein Beweisassistent für Logik höherer Ordnung.* LNCS, Band 2283. Springer, Berlin, 2002. www.cl.cam.ac.uk/Research/HVG/Isabelle.

[NS61] A. Newell und HA Simon. GPS, ein Programm, das menschliches Denken simuliert. In H. Billing, Herausgeber, *Lernende Automaten,* Seiten 109–124. Oldenburg, München, 1961.

[NSS83] A. Newell, JC Shaw und HA Simon. Empirische Untersuchungen mit der Logik Theoriemaschine: eine Fallstudie zur Heuristik. In J. Siekmann und G. Wrightson, Herausgeber, *Automatisierung des Denkens 1: Klassische Arbeiten zur Computerlogik 1957–1966,* Seiten 49–73. Springer, Berlin, 1983. Erstveröffentlichung: 1957.

[OFY+95] C. Ohmann, C. Franke, Q. Yang, M. Margulies, M. Chan, van PJ Elk, FT de Dombal und HD Röher. Diagnosescore für akute Appendizitis. *Chirurg,* 66:135–141, 1995.

[OMYL96] C. Ohmann, V. Moustakis, Q. Yang und K. Lang. Evaluierung automatischer Wissenserwerbstechniken bei der Diagnose akuter Bauchschmerzen. *Artif. Intel. Med.,* 8:23–36, 1996.

[OPB94] C. Ohmann, C. Platen und G. Belenky. Computerunterstütze Diagnose bei akuten Bauchschmerzen. *Chirurg,* 63:113–123, 1994.

[Pal80] G. Palm. Zum assoziativen Gedächtnis. *Biol. Cybern.,* 36:19–31, 1980.

[Pal91] G. Palm. Speicherkapazitäten lokaler Regeln für synaptische Modifikation. *Concepts Neurosci.,* 2(1):97–128, 1991. MPI Tübingen.

[Pea84] J. Perle. *Heuristiken, intelligente Suchstrategien zur Lösung von Computerproblemen.* Addison-Wesley, Reading, 1984.

[Pea88] J. Perle. *Probabilistisches Denken in intelligenten Systemen. Netzwerke plausibler Schlussfolgerungen.* Morgan Kaufmann, San Mateo, 1988.

[PL05] L. Panait und S. Luke. Kooperatives Lernen mit mehreren Agenten: der Stand der Technik. *Auton. Agents Multi-Agent Syst.,* 11(3):387–434, 2005.

[PS08] J. Peters und S. Schaal. Verstärkendes Erlernen motorischer Fähigkeiten mit Politikgradienten. *Neural Netw.,* 21(4):682–697, 2008. http://www-clmc.usc.edu/publications/P/peters-NN2008.pdf.

[PS09] G. Pólya und S. Sloan. *Wie man es löst: Ein neuer Aspekt der mathematischen Methode.* Ishi, Mountain View, 2009.

[Qui93] J. Ross Quinlan. *C4.5: Programme für maschinelles Lernen.* Morgan Kaufmann, San Mateo, 1993. C4.5-Download: http://www.rulequest.com/Personal, C5.0-Download: http://www.rulequest.com/download.html.

[RB93] M. Riedmiller und H. Braun. Eine direkte adaptive Methode für eine schnellere Backpropagation Lernen: der Rprop-Algorithmus. In *Proceedings der IEEE International Conference on Neural Networks,* Seiten 586–591, 1993.

[RGH+06] M. Riedmiller, T. Gabel, R. Hafner, S. Lange und M. Lauer. Die Brainstormer: Entwurfsprinzipien lernfähiger autonomer Roboter. *Inform.-Spektrum,* 29(3):175–190, 2006.

[RHR86] DE Rumelhart, GE Hinton und RJ Williams. Interne Repräsentationen lernen durch Fehlerfortpflanzung. In [RM86].

[Ric83] E. Rich. *Künstliche Intelligenz.* McGraw-Hill, New York, 1983.

[RM86] D. Rumelhart und J. McClelland. *Parallele verteilte Verarbeitung,* Band 1. MIT Press, Cambridge, 1986.

[RM96] W. Rödder und C.-H. Meyer. Kohärente Wissensverarbeitung bei maximaler Entropie von SPIRIT. In KI-96 *(Bundeskonferenz* KI), Dresden, 1996.

[RMD07] M. Riedmiller, M. Montemerlo und H. Dahlkamp. Lernen, ein echtes Auto zu fahren 20 Minuten. In FBIT '07: *Proceedings of the 2007 Frontiers in the Convergence of Biowissenschaften und Informationstechnologien,* Seiten 645–650. IEEE Computer Society, Washington, D.C., 2007.

[RMS92] H. Ritter, T. Martinez und K. Schulten. *Neuronale Berechnung und Selbstorganisation Karten.* Addison-Wesley, Reading, 1992.

[RN10] S. Russell und P. Norvig. *Künstliche Intelligenz: Ein moderner Ansatz.* Prentice Hall, New York, 3. Auflage, 2010. 1. Auflage: 1995. http://aima.cs.berkeley.edu.

[Roba] Offizielle Robocup-Website. www.robocup.org.

[Robb] Der Robocup-Fußballsimulator. http://sserver.sourceforge.net.

[Rob65] JA Robinson. Eine maschinenorientierte Logik basierend auf dem Auflösungsprinzip. *J. ACM,* 12(1):23–41, 1965.

[Roj96] R. Rojas. *Neuronale Netze: Eine systematische Einführung.* Springer, Berlin, 1996.

[Ros58] F. Rosenblatt. Das Perzeptron: ein probabilistisches Modell zur Informationsspeicherung und Organisation im Gehirn. *Psychol. Rev.,* 65:386–408, 1958. Nachdruck in [AR88], Seiten 92–114.

[Ros09] SM Ross. *Einführung in die Wahrscheinlichkeitstheorie und Statistik für Ingenieure und Wissenschaftler.* Academic Press, San Diego, 2009.

[RW06] CE Rasmussen und CKI Williams. *Gaußsche Prozesse für maschinelles Lernen ing.* MIT Press, Cambridge, 2006. Online-Version: http://www.gaussianprocess.org/gpml/chapters/.

[SA94] S. Schaal und CG Atkeson. Roboter-Jonglage: speicherbasierte Implementierung Lernen. *IEEE-Steuerungssystem. Mag.,* 14(1):57–71, 1994.

[Sam59] AL Samuel. Einige Studien zum maschinellen Lernen mithilfe des Damespiels. *IBM J.,* 1(3):210–229, 1959.

[SB98] R. Sutton und A. Barto. *Verstärkungslernen.* MIT Press, Cambridge, 1998. www.cs.ualberta.ca/~sutton/book/the-book.html.

[SB04] J. Siekmann und Ch. Benzmüller. Omega: Computergestützte Mathematik. In KI 2004: *Fortschritte in der künstlichen Intelligenz.* LNAI, Band 3238, Seiten 3–28. Springer, Berlin, 2004. www.ags.uni-sb.de/~omega.

[Sch96] M. Schramm. *Indifferenz, Unabhängigkeit und maximale Entropie: Eine Wahrscheinlichkeitstheoretische Semantik für Nicht-Monotones Schließen.* Dissertation zur Informatik, Band 4. CS, München, 1996.

[Sch01] J. Schumann. *Automatisierte Beweisführung von Theoremen in der Softwareentwicklung.* Springer, Berlin, 2001.

[Sch02] S. Schulz. E – ein intelligenter Theorembeweiser. *J. AI Commun.,* 15(2/3):111–126, 2002. www4.informatik.tu-muenchen.de/~schulz/WORK/eprover.html.

[Sch04] A. Schwartz. *SpamAssassin.* O'Reilly, Cambridge, 2004. SpamAssassin-Homepage: http://spamassassin.apache.org.

[SE90] CH. Suttner und W. Ertel. Automatische Erfassung suchführender Heuristiken. Im *10 Int. Konf. zum automatischen Abzug.* LNAI, Band 449, Seiten 470–484. Springer, Berlin, 1990.

[SE00] M. Schramm und W. Ertel. Argumentation mit Wahrscheinlichkeiten und maximaler Entropie: das System PIT und seine Anwendung in LEXMED. In K. Inderfurth et al., Herausgeber, *Operations Research Proceedings* (SOR'99), Seiten 274–280. Springer, Berlin, 2000.

[SE10] M. Schneider und W. Ertel. Roboterlernen durch Demonstration mit lokalem Gaussian Prozessregression. In *Proceedings of the IEEE/ RSJ International Conference on Intelligente Roboter und Systeme* (IROS'10), 2010.

[SET09] T. Segaran, C. Evans und J. Taylor. *Programmierung des Semantic Web.* O'Reilly, Cam Bridge, 2009.

[Sho76] EH Shortliffe. *Computergestützte medizinische Beratung, MYCIN.* Nordholland, New York, 1976.

[Spe97] *Geheimnisse des Geistes.* Sonderausgabe. Scientific American Inc., 1997.

[Spe98] *Intelligenz erforschen.* Scientific American Presents, Band 9. Scientific American Inc., 1998.

[SR86] TJ Sejnowski und CR Rosenberg. NETtalk: ein paralleles Netzwerk, das lesen lernt laut. Technischer Bericht JHU/EECS-86/01. The John Hopkins University Electri cal Engineering and Computer Science Technical Report, 1986. Nachdruck in [AR88], Seiten 661–672.

[SS02] S. Schölkopf und A. Smola. *Lernen mit Kerneln: Support Vector Machines, Regularisierung, Optimierung und mehr.* MIT Press, Cambridge, 2002.

[SS06] G. Sutcliffe und C. Suttner. Der Stand von CASC. *AI Commun.,* 19(1):35–48, 2006. CASC-Homepage: www.cs.miami.edu/~tptp/CASC.

[SSK05] P. Stone, RS Sutton und G. Kuhlmann. Verstärkungslernen für Robocup-Fußball-Keepaway. *Adaptives Verhalten,* 2005. www.cs.utexas.edu/~pstone/Papers/bib2html-links/AB05.pdf.

[Ste07] J. Stewart. *Multivariable Infinitesimalrechnung.* Brooks Cole, Florenz, 2007.

[SW76] CE Shannon und W. Weaver. *Mathematische Grundlagen der Informationstheorie.* Oldenbourg, München, 1976.

[Größe10] C. Szepesvari. *Algorithmen für Reinforcement Learning.* Morgan & Claypool, San Rafael, 2010. Entwurf online verfügbar: http://www.ualberta.ca/~szepesva/RLBook.html.

[Ted08] R. Tedrake. Lernkontrolle bei mittleren Reynolds-Zahlen. Im *Workshop zu: Robotics Challenges for Machine Learning II, Internationale Konferenz über intelligente Roboter und Systeme* (IROS 2008), Nizza, Frankreich, 2008.

[TEF09] M. Tokic, W. Ertel und J. Fessler. Der Crawler, ein Klassenzimmer-Demonstrator für verstärkendes Lernen. In *Proceedings of the 22nd International Florida Artificial Konferenz der Intelligence Research Society* (FLAIRS 09). AAAI Press, Menlo Park, 2009.

[Tes95] G. Tesauro. Zeitdifferenzlernen und TD-Gammon. *Komm. ACM,* 38(3), 1995. www.research.ibm.com/massive/tdl.html.

[Tok06] M. Tokic. Entwicklung eines lernfähigen Laufroboters. Diplomarbeit Hochschule Ravensburg-Weingarten, 2006. Inklusive Simulationssoftware verfügbar auf www.hs-weingarten.de/~ertel/kibuch.

[Tur37] AM Turing. Über berechenbare Zahlen, mit einer Anwendung auf das Entschei dungsproblem. *Proz. London. Mathematik. Soc.,* 42(2), 1937.

[Tur50] AM Turing. Computermaschinen und Intelligenz. *Mind,* 59:433–460, 1950.

[vA06] L. v. Ahn. Spiele mit einem Zweck. *IEEE Comput. Mag.,* 96–98, Juni 2006. http://www.cs.cmu.edu/~biglou/ieee-gwap.pdf.

[Wei66] J. Weizenbaum. ELIZA – ein Computerprogramm zur Untersuchung der natürlichsprachlichen Kommunikation zwischen Mensch und Maschine. *Komm. ACM,* 9(1):36–45, 1966.

[WF01] I. Witten und E. Frank. *Data Mining.* Hanser, München, 2001. DataMining Java Li brary WEKA: www.cs.waikato.ac.nz/~ml/weka.

[Whi96] J. Whittaker. *Grafische Modelle in der angewandten multivariaten Statistik.* Wiley, New York, 1996.

[Wie] U. Wiedemann. PhilLex, Lexikon der Philosophie. www.phillex.de/paradoxa.htm.

[Wie04] J. Wielemaker. SWI-PROLOG 5.4. Universität van Amsterdam, 2004. www.swi-prolog.org.

[Wik10] Wikipedia, die freie Enzyklopädie, 2010. http://en.wikipedia.org.

[Gewinnen] P. Winston. Spieldemonstration. http://www.ai.mit.edu/courses/6.034f/gamepair.html. Java-Applet für Minimax- und Alpha-Beta-Suche.

[Zel94] A. Zell. *Simulation neuronaler Netze.* Addison-Wesley, Reading, 1994. Beschreibung von SNNS und JNNS: www-ra.informatik.uni-tuebingen.de/SNNS.

[ZSR+99] A. Zielke, H. Sitter, TA Rampp, E. Schäfer, C. Hasse, W. Lorenz und M. Rothmund. Überprüfung eines diagnostischen Scoresystems (Ohmann-Score) für die akute Blinddarmentzündung. *Chirurg,* 70:777–783, 1999.

# Index



W. Ertel, *Einführung in die künstliche Intelligenz,*
Bachelor-Themen in Informatik, DOI
10.1007/978-0-85729-299-5, 