

9

9 Anhang: Grundlegende Theorien

9.1 Einleitung

Information Engineering basiert auf einer Reihe von grundlegenden Theorien. Diese Theorien sind im Rahmen einer Vortragsreihe von Studentinnen und Studenten der Hochschule für Technik erarbeitet worden und liegen in diesem Kapitel aufbereitet als Sammlung ohne Anspruch auf Vollständigkeit vor.

In diesem Kapitel wird eine Reihe von grundlegenden Konzepten zur Realisierung von abstrakten Maschinen, auch Automaten genannt, vorgestellt. Der systematische Umgang mit Informationen erfolgt durch die konkrete Realisierung solcher Abstrakten Maschinen. Sehr viele Software- und Hardwarekomponenten basieren auf diesen Maschinen.

Dieses Kapitel umfasst eine Sammlung von grundlegenden Konzepten, die für die Interpretation von Informationen von zentraler Bedeutung sind.

9.2 Abstrakte Maschinen

In diesem Kapitel wird eine Reihe von grundlegenden Konzepten zur Realisierung von abstrakten Maschinen, auch Automaten genannt, vorgestellt. Der systematische Umgang mit Informationen erfolgt durch die konkrete Realisierung solcher Abstrakten Maschinen. Sehr viele Software- und Hardwarekomponenten basieren auf diesen Maschinen.

Folgende Maschinen werden vorgestellt:

- Turingmaschine (Beitrag von Roman Bruggisser)
- Finite State Machine (Beitrag von Joachim Huber)
- Deterministische & nicht deterministische Automaten (Beitrag von Roland Ulrich)
- Markov Automaten (Beitrag von Ranjan Benz)
- Stochastische Automaten (Arsim Gashi)
- Lattice-Fuzzy Automaten (Beitrag von Michal Kritzner)
- Zelluläre Automaten (Beitrag von Manuela Lehmann)

9.2.1 Formale Definition eines Automaten

Der Automat (griechisch: *automatos* = sich selbst, aus eigenem Antrieb bewegend) ist ein System, welches Informationen aufnimmt, verarbeitet und abgibt.



Der abstrakte Automat

Ein abstrakter Automat mit Ausgabe ist ein 5-Tupel $A(I, O, S, f_s, f_o)$

I : Nichtleere Menge der Eingabevariablen (Eingabealphabet - Input Symbols)

O : Nichtleere Menge der Ausgabevariablen (Ausgabealphabet - Output Symbols)

S : Nichtleere Menge der Zustände (Set of States)

f_s : Die Zustandsfunktion $f_s: X \times Z \rightarrow Z$ (State Transition Function)

f_o : Die Ausgabefunktion $f_o: X \times Z \rightarrow Y$ (Output Function)

Der Automat heisst endlich, wenn die Mengen I , O und S endlich sind.

Die drei Ausprägungsmengen in der Definition eines Automaten können folgendermaßen interpretiert werden:

- *Eingabe*: Ein Automat muss von außen bedient werden können.
- *Interne Zustände*: Jeder Automat befindet sich immer in einem bestimmten Zustand. Unter Einwirkung der Eingaben kann der Automat eine Reihe von Zuständen durchlaufen.
- *Ausgabe*: Im Laufe seiner Arbeit produziert der Automat Informationen, d.h. er gibt Ausgabedaten aus.

9.2.2 Die Turingmaschine

Will man die Lösbarkeit eines Problems aufzeigen, so entwickelt man eine Methode, mit welcher sich das Problem lösen lässt. Für den Anwender der Methode genügt es zu wissen, wie diese anzuwenden ist. Dass das Problem mit der Methode gelöst wurde, lässt sich in der Regel am korrekten Ergebnis erkennen oder aber mit anderen Methoden nachprüfen [Schneider, Werner 2004].

Schwieriger wird es jedoch, wenn man beweisen möchte, dass ein Problem unlösbar ist. Ein falsches Ergebnis zeigt uns, dass die gewählte Methode für das Problem ungeeignet ist, eine Aussage darüber, ob das Problem überhaupt lösbar ist lässt sich damit aber nicht anstellen.

Unter anderen hat sich in den 30er Jahren des 20. Jahrhunderts auch der britische Mathematiker Alan Turing (23.06.1912 – 07.06.1954) mit dieser Thematik auseinandergesetzt. 1936 stellte er schliesslich eine Methode vor, mit deren Hilfe eine Aussage über die Lösbarkeit eines Problems erfolgen kann, die Turingmaschine.

Definition

Eine Turingmaschine ist keine reale Maschine, vielmehr handelt es sich um eine (mathematische) Methode, welche jedoch einen starken Maschinenorientierten Ansatz verfolgt. Mit Hilfe von Simulatoren lässt sich diese Methode dennoch sehr gut veranschaulichen. Die Turingmaschine ist nicht die einzige Methode, welche sich mit der Lösbarkeit von Problemen beschäftigt, doch sind alle vernünftigen Modelle zur Turingmaschine äquivalent, was zur so genannten Church'schen These [Church 1936] führt (vereinfacht formuliert):

Church'sche These:

Alles was berechenbar ist, ist durch eine Turingmaschine berechenbar!

Oder umgekehrt: Alles was nicht durch eine Turingmaschine berechnet werden kann, ist überhaupt nicht berechenbar!

Aufbau und Funktionsweise

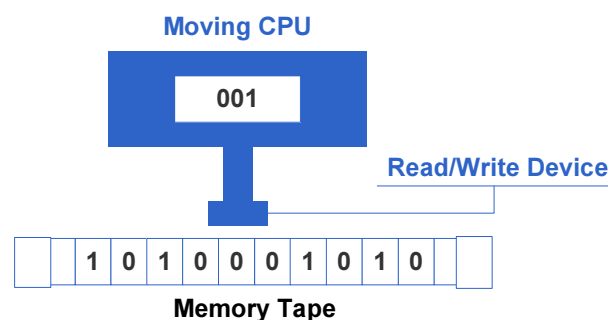


Abbildung 9.1 Turing Maschine

Eine Turingmaschine besteht aus drei Komponenten:

- *Memory Tape*: Das unendliche Arbeitsband dient als Eingabe- wie auch als Ausgabeband. Es ist in einzelne Felder aufgeteilt, welche je ein Zeichen aufnehmen können.
- *Read/Write Device*: Der Lese/Schreibkopf kann zu jedem Zeitpunkt genau ein Zeichen des unendlichen Arbeitsbandes lesen oder überschreiben. Nach jeder Verarbeitung eines Zeichens kann sich der Kopf entweder nach rechts oder links um ein Feld (Zeichen) bewegen.
- *Moving CPU*: Die Steuereinheit enthält das so genannte Turingprogramm, welches aus einer endlichen Zahl von elementaren Operationen besteht. Während der Abarbeitung des Turingprogramms kann die Steuereinheit eine endliche Zahl von verschiedenen Zuständen annehmen. Da die Zahl der Operationen und Zustände der Steuereinheit endlich ist, spricht man auch von einer endlichen Steuerung.

Je nach Zustand und dem aktuellen Inhalt des Arbeitsbandes kann jeweils eine der folgenden Aktionen ausgeführt werden:

- Schreiben eines neuen Zeichens an die aktuelle Bandposition
- Verschieben des Lese/Schreibkopfes um eine Position nach rechts oder links
- Verändern des internen Zustands

Mehrere solcher Befehle lassen sich zu einfachen Programmen zusammenführen. Benötigt man komplexere Programme können diese aus einfacheren zusammengesetzt werden.

Die universelle Turingmaschine

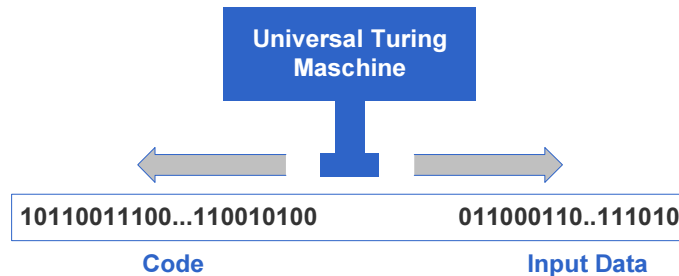


Abbildung 9.2 Universelle Turing Maschine

Unter einer universellen Turingmaschine versteht man eine Maschine, welche durch die Kombination vieler kleiner Turingmaschinen in der Lage ist, jede beliebige Aufgabe zu lösen. Dazu schreibt man die Kodierung (Turingprogramm) der zu simulierenden Turingmaschine auf die linke und die Bandinschrift auf die rechte Seite des Bandes der universellen Turingmaschine. Die universelle Maschine liest nun abwechselnd Kodierung und Bandinschrift und ermittelt dadurch den Zustand der zu simulierenden Turingmaschine.

Mehrband-Turingmaschinen

Da auch ein moderner Rechner immer eine endlich Zahl innerer Zustände und Anweisungen aufweist, kann im Prinzip jeder Rechner als Turingmaschine betrachtet werden. Das Modell der Turingmaschine weicht jedoch von der klassischen Rechnerarchitektur (Von-Neumann) ab, da sie Eingabemedium und Speicher nicht unterscheidet (Arbeitsband). Ausserdem lässt die Linearität des Speichers einer Turingmaschine zu wünschen übrig. Um zwei Feldinhalte miteinander zu vergleichen sind immer mindestens so viele Operationen auszuführen, wie die beiden Felder voneinander entfernt sind. Das heisst, je grösser die Distanz zweier Felder desto grösser die Anzahl Schritte für einen Vergleich der beiden Feldinhalte. Dieses Problem führte zur Idee der Mehrband-Turingmaschine, welche der Von-Neumann-Architektur schon wesentlich näher kommt. Eine solche Maschine unterscheidet nun ein reines Eingabeband und mehrere Arbeitsbänder. Dadurch bleibt nun einerseits der Inhalt des Eingabebandes während der gesamten Berechnung gleich und andererseits lässt sich durch die Verwendung mehrerer Arbeitsbänder die Anzahl Operationen verringern, da man auf mehrere Werte gleichzeitig (parallel) zugreifen kann.

9.2.3 Finite State Machine

Eine Finite State Machine (FSM) oder auch ein endlicher Automat ist ein Modell eines Systems mit einer begrenzten Anzahl von definierten Stati wobei die Statusänderungen vom Input in das System abhängen [Planxton 2005]. Ein FSM ist ein in sich geschlossenes System, welches durch die Menge der Zustände und deren Statusübergänge definiert wird.

Eine FSM besteht aus:

Der Menge der Inputs: $I = \{i_1, i_2, \dots, i_n\}$

Der Menge der Outputs: $O = \{o_1, o_2, \dots, o_n\}$

Der Menge der Stati: $S = \{s_1, s_2, \dots, s_n\}$

Dem Initialstatus: $s_1 \in S$

Der Übergangsfunktionen: $f_s(S, I)$

Der Outputfunktionen $f_o(S)$

Darstellungsformen

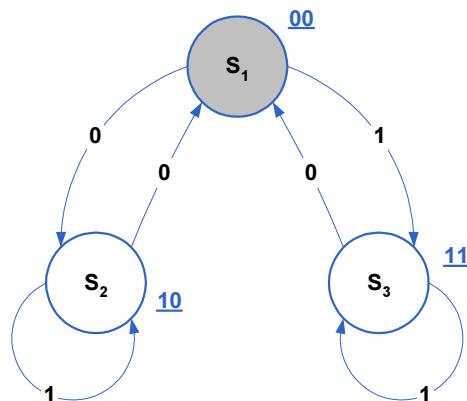


Abbildung 9.3 Zustandsdiagramm einer FSM

Eine Finite-State-Machine kann mit einem Zustandsdiagramm beschrieben werden.

- *Kreis*: Symbolisiert einen Status innerhalb des Systems
- *Kanten*: Repräsentieren Statusübergänge
- *Beschriftungen*: Beschreiben den Input und den (hier unterstrichenen) Output. Nicht jeder Status muss einen Output generieren.

Tabelle 9.1 Die Zustände der FSM

$s_1 = \text{Initial}$	t	$t+1$	$t+2$
------------------------	-----	-------	-------

Inputs	0	1	0
Outputs	00	11	00

Tabelle 9.2 State Transition Table

State Transition Table $f_s(S, I)$		
0	S_1	S_2
1	S_1	S_3
0	S_2	S_1
1	S_2	S_2
0	S_3	S_1
1	S_3	S_3

Beispiel: $f_s(S_1, 0) = S_2$:

Wenn Status 1 den Input 0 erhält, wird in den Status 2 gewechselt.

$f_o(S)$	
S_1	00
S_2	10
S_3	11

Beispiel: $f_o(S_1, 0) = S_2$:

Wird in den Status s_2 gewechselt, wird der Output 10 generiert.

Anwendungen

Finite State Machines sind im Alltag sehr verbreitet. Beispiele für Alltagsgegenstände deren Verhalten sich leicht durch eine FSM beschreiben lassen sind Getränkeautomaten, Lichtschalter oder Liftsteuerungen. Auch ein PC oder ein beliebiges Programm kann als State Machine beschrieben werden. Diese können beliebig komplex sein, wenn jedoch nur sehr rudimentäre Dinge interessieren, zum Beispiel PC läuft/läuft nicht oder Programm erwartet Input/erwartet keinen Input, können auch in sich komplexe Systeme vereinfacht dargestellt werden. Anwendungen finden State Machines immer dann, wenn ein System abhängig vom aktuellen Status auf ein Input reagieren muss.

9.2.4 Deterministische & nicht deterministische Automaten

Definition:

Ein endlicher Automat beschreibt ein Modell eines Systems, welches sich in einer endlichen Anzahl Zuständen befindet. Es besteht aus:

Dem Eingabealphabet: I

Dem Startzustand: $S_1 \in S$

Der Übergangsfunktion: f_s

Der endlichen Menge von Zuständen: S

Der Menge von Endzuständen: O

Alle Ereignisse laufen nach vorher festgelegten Gesetzen ab. Ein deterministischer Automat liest aus einer endlichen Menge von Symbolen eine Folge von Symbolen schrittweise ein und stoppt anschliessend in einem seiner Zustände. Das Interessante an nicht deterministischen Automaten ist, dass sie in mehreren Zuständen gleichzeitig sein können. Für beide Typen gilt: Zustände können entweder akzeptierend oder nicht akzeptierend sein.

Arbeitsweise der Automaten

Der Automat liest in jedem Schritt ein Eingabesymbol und geht abhängig von seinem momentanen Zustand in einen neuen Zustand über. Beim deterministischen Automat gibt es genau eine Möglichkeit wie der Automat vom einen State in den nächsten wechseln kann. Der nicht deterministische Automat kennt dagegen mehrere Wege. Er ist also nicht ganz so strikt an einen genau spezifizierten Ablauf gebunden.

Einsatzmöglichkeiten und Beispiele

Das parsen eines Strings ist eine typische Aufgabe für einen nicht deterministischen Automaten. Der String wird Zeichenweise in den Automaten eingespeist und dieser bleibt dann in einem akzeptierenden oder nichtakzeptierenden Zustand stehen. Akzeptiert heisst in diesem Fall, dass der String in der durch den Automaten gegebene Sprache existiert [Friedl 2007].



Abbildung 9.4 Ein deterministischer Automat, welcher den String "die" akzeptiert

Der Automat liest den String "die" nun Schrittweise ein. Wenn er ein "d" liest wechselt er vom (Anfangs-) Zustand 0 in Zustand 1, folgt nun ein "i" geht er in Zustand 2 über und wenn als nächstes ein "e" gelesen wird wechselt er in Zustand 3 (in diesem Fall ein Akzeptierer). Der Automat hätte nun "die" als gültigen String erkannt, da er im akzeptierenden Zustand 3 stehen geblieben ist.

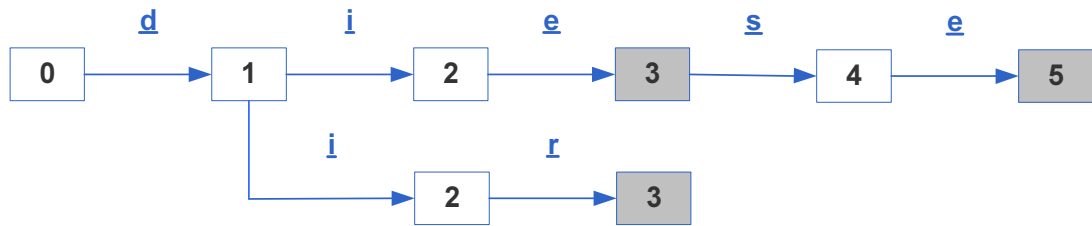


Abbildung 9.5 Ein nichtdeterministischer Automat, welcher die drei Wörter “die”, “diese” und “dir” akzeptiert

Dieser nicht deterministische Automat erkennt die drei Wörter “die”, “diese” und “dir”. Beim Zustandsübergang von 1 nach 2 ist der Automat nach dem Zeichen “i” in beiden Zuständen gleichzeitig. Erst beim einlesen des nächsten Zeichens wird entschieden welcher Weg nun eingeschlagen wird.

9.2.5 Markov Automaten

A. A. Markov

A. A. Markov (1856 – 1922), russischer Mathematiker, hat 1913 ein statistisches Verfahren entwickelt, mit dem er die Häufigkeitsverteilung im Auftreten russischer Vokale und Konsonanten berechnen konnte. Praktisch hat er es an der Novelle Puschkins „Eugen Onegin“ angewendet.

Markov-Ketten

Das Markov-Modell beruht auf einem stochastischen Prozess (Folge von elementaren Zufallseignissen, die nicht unabhängig voneinander sind) [Rabiner, Juang 1986]. Dabei ist jeder Zustand nur vom vorgehenden Zustand abhängig. Diese Abhängigkeit wird mit einer Wahrscheinlichkeit angegeben. Dieser stochastische Prozess ohne Gedächtnis mit endlich vielen Zuständen entspricht den Eigenschaften eines endlichen Automaten und wird Markov-Kette genannt.

Definition:

Eine Markov-Kette benötigt folgende Daten:

Ω : Anfangswahrscheinlichkeit eines Zustandes

A : Übergangswahrscheinlichkeit zum nächsten Zustand

S : Elementarereignismenge

Beispiel

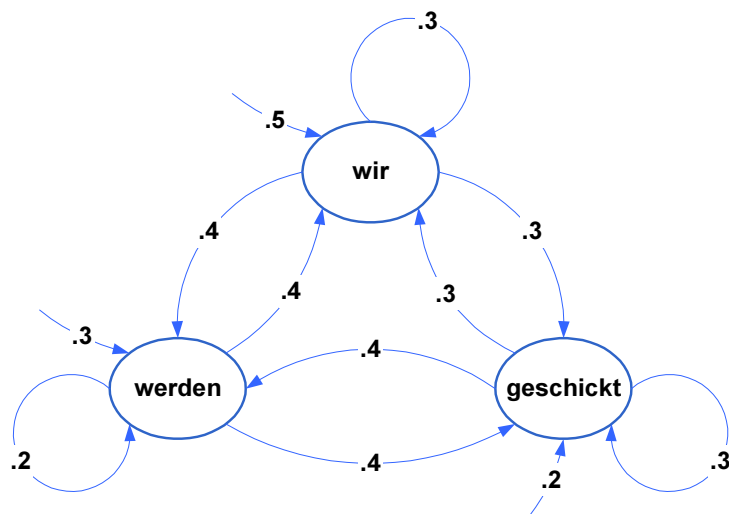


Abbildung 9.6 Markov-Kette

Die Kette besteht aus einer Elementarereignismenge, einer Anfangswahrscheinlichkeit und einer Zustandsübergangswahrscheinlichkeit.

Elementarereignismenge: $\Omega(S) = \{ \text{geschickt, werden, wir} \}$

Tabelle 9.3 Anfangswahrscheinlichkeit

X_t	Ω
geschickt	0.2
werden	0.3
wir	0.5

Tabelle 9.4 Zustandsübergangswahrscheinlichkeit

$X_t = s_i$	$X_{t+1} = s_j$		
	geschickt	werden	wir
geschickt	0.3	0.4	0.3
werden	0.4	0.2	0.4
wir	0.3	0.4	0.3

Kommentar zum Beispiel

Es braucht eine Anfangswahrscheinlichkeit eines Zustandes, die angibt mit welcher Wahrscheinlichkeit dieser Zustand den Startzustand bildet, also z.B. Ein Satz mit <wir> anfängt. $P(X_1 = x_1)$, also $P(X_1 = \text{wir})$. Dazu wird von jedem Zustand die Übergangswahrscheinlichkeit in den nächsten Zustand definiert, z.B. wie wahrscheinlich ist es, dass nach einem

<wir> ein <werden> kommt. $P(X_{t+1} = x_{t+1} \mid X_1 = x_1, X_2 = x_2, \dots, X_t = x_t) = P(X_{t+1} = x_{t+1} \mid X_t = x_t)$, also $P(X_{t+1} = \text{werden} \mid X_t = \text{wir})$.

Die Wahrscheinlichkeit einer Markovkette kann so beschrieben werden:

Für die Zustandssequenz (X_1, X_2, \dots, X_t) gilt $P(X_1, \dots, X_t) = P(X_1) * P(X_2 \mid X_1) * P(X_3 \mid X_2) * \dots * P(X_t \mid X_{t-1})$.

Markov-Modell

Im Markov-Modell wird jedem Zustand (oder Übergang) ein Output zugeordnet (State / Arc Emission Model) zugeordnet.

- *Zustandssequenz sichtbar*: Visible Markov Model (VMM)
- *Zustandssequenz unsichtbar*: Hidden Markov Model (HMM)



Definition Markov-Modell:

Die Bestandteile eines Markov-Modells sind:

S: Menge der Zustände (Elementarereignismenge)

Ω: Anfangswahrscheinlichkeit

A: Zustandsübergangswahrscheinlichkeit

B: Wahrscheinlichkeit des Outputs

O: Ausgabe Alphabet (Output)

Diese Daten werden für die übersichtliche Darstellung in Matrizen abgebildet

Anwendungen

Tabelle 9.5 Evaluation

Gegeben:	Set of HMM (Ω, A, B), Outputsequenz
Gesucht:	Das wahrscheinlichste HMM
Algorithmus:	Forward
Anwendung:	Spracherkennung

Tabelle 9.6 Dekodierung

Gegeben:	Set of HMM (Ω, A, B), Outputsequenz
Gesucht:	Die wahrscheinlichste Hidden Sequenz
Algorithmus:	Viterbi
Anwendung:	Wordprocessing

Tabelle 9.7 Training, Modellgenerierung

Gegeben:	Hidden- und Outputsequenz
Gesucht:	HMM (Ω , A, B)
Algorithmus:	forward/backward, Baum-Welch

9.2.6 Stochastische Automaten

Stochastische Automaten sind eine Erweiterung der nichtdeterministischen Automaten. Ein stochastischer Automat ist ein Automat, dessen Verhalten bei gleichem Input variieren kann. Die Übergänge zwischen den einzelnen Stati (Transitions) werden als Wahrscheinlichkeitsfunktionen dargestellt.

Definition Stochastischer Automat:

Ein stochastischer Automat ist ein 6-Tupel (S, s_0, I, V, f_s, f_o)

S : Eine nichtleere Menge der Zustände

$s_0 \in S$: Der Initialzustand

I : Nichtleere Menge der Eingabevariablen

$V [0,1]$: Der Wertebereich

f_s : Die Übergangsfunktion $f_s: X \times Z \rightarrow Z$

f_o : Die Ausgabefunktion $f_o: X \times Z \rightarrow Y$

Ein stochastischer Automat kann jede Folge von Zuständen durchlaufen, die mit einem Zustand s_0 beginnt. Solange der Automat mit einer Wahrscheinlichkeit > 0 annimmt, dass ein nächster Zustand erreicht werden kann und solange die Abfolge der Übergangswahrscheinlichkeiten nicht verschwindend klein ist, funktioniert der Automat weiter. Eine Abfolge von Zuständen wird auch Trajektorie genannt.

Beispiel

Stochastische Automaten eignen sich sehr gut zur Modellierung komplexer Verhaltensstrukturen von Systemen. Sie werden in der Automobilindustrie für die Steuerung von Bremssystemen und in der Robotik für die Nachbildung "menschlicher" Verhaltensweisen eingesetzt.

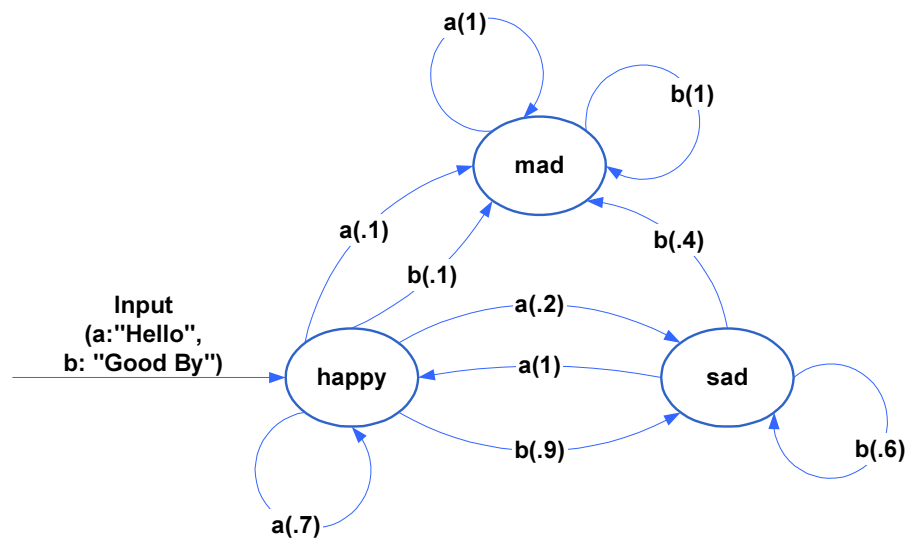


Abbildung 9.7 The Emotion Robot

Der Roboter kann in drei Zuständen sein (happy, sad, mad). Zwei Input Signale sind möglich (a: "Hello" und b: "Good By"). Das Modell bildet die Wahrscheinlichkeit des Eintreffens eines bestimmten Zustandübergangs bei einem bestimmten Eingangssignal ab [Dougherty, Giardina 1988].

9.2.7 Lattice-Fuzzy Automaten

Die klassische Mengenlehre (Boolesche Algebra) benannt nach George Bool definiert, dass ein Zeichen einer Menge angehört oder eben nicht (true, false oder 1, 0). Man teilt die Variablen einem der zwei Zustände (0,1 oder false, true) zu.

Kann eine Zuordnung eines Zeichens mit dem Wertepaar z.B. Null oder Eins nicht stattfinden, so spricht man von unscharfen Mengen (fuzzy sets). Die gegebene Variable kann eher Null oder eher Eins (bzw. false, true) sein. Es existieren gleitende Übergänge der Zugehörigkeit eines Zeichens im Intervall $[0, 1]$. Die Fuzzy-Mengen Theorie weist Zeichen von unscharfen Mengen ihre Zugehörigkeit im Intervall $[0, 1]$ zu.

Zu den bekannten Mengen, welche algebraischen Strukturen in der Mathematik unterliegen, bestehen noch weitere Gruppen, Verbände, Grafen oder Gitter. Das Gitter nennt sich englisch *Lattice*. Die Lattice Theorie entstand im 19. Jahrhundert (zum Teil unter anderem Namen) und beruht auf der Booleschen Algebra (Evariste Galois, Niels Henrik, Abel, Sophus Lee) [Birkhoff 1979].

Lattice Theorie

Die Lattice Theorie wird mit Gittern (Lattice) oder Grafen dargestellt. Man spricht von Lattice, wenn ein Gitter ein einzigartiges Element zuoberst und zuunterst (1, 0 oder true,

false oder andere) definiert hat. Das Lattice ordnet also anhand von unteren und oberen Grenzen (supremum, infimum oder min, max) die Objekte (Variablen) in einem Vektorfeld (Grafen) ein. So entstehen so genannte Halbgruppen, die nicht leer sind und eine Verknüpfung haben, wie es in der Übersicht zu sehen ist. Auf natürliche Weise ist es so möglich eine Formalisierung, Einteilung von Objekten durchzuführen. Wobei die Booleschen Gesetzmässigkeiten und Axiome weitgehend Gültigkeit haben.

Definition Fuzzy Automaten:

Ein Fuzzy Automat ist ein 6-Tupel (S, s_0, I, V, f_s, f_o)

S : Eine nichtleere Menge der Zustände

$s_0 \in S$: Der Initialzustand

I : Nichtleere Menge der Eingabevariablen

$V [0,1]$: Der Wertebereich

f_s : Die Delta oder auch Übergangsfunktion $f_s: X \times Z \rightarrow Z$

f_o : Die Funktion, die den Endzustand des Automaten darstellt $f_o: X \times Z \rightarrow Y$

Im Prinzip sind Lattice und Fuzzy Automaten eine Erweiterung der stochastischen Automaten. Die Summe der Wahrscheinlichkeiten aller Übergangsfunktionen in einem stochastischen Automaten muss 1 sein. In einem Fuzzy und in einem Lattice Automaten sind die Wahrscheinlichkeiten der Verteilung gewissen Bereichen zugeordnet, sie sind also variabel. Dies hat zur Folge, dass auch nicht klar abgegrenzte Eingabevariablen verarbeitet werden können. Typischerweise finden diese Automaten dann Anwendung, wenn der Wert eines Eingangssignals (z.b. Temperatur) variieren kann und je nach Variation oder nach Kombination der Werte der verschiedenen Eingangssignale ein anderes Verhalten des Automaten gewünscht wird.

Lattice Automaten

Lattice Automaten sind von Fuzzy Automaten abgeleitet. Lediglich der Wertebereich $V [0,1]$ ist in einzelne Gitter aufgeteilt.

Anwendungen

Es gibt in der Wissenschaft etliche Modelle von Lattice Gittern. Für Modelle von Molekülen in der Chemie und deren Rotationen, Änderungen oder Transformationen wird die Lattice Theorie oft eingesetzt. Andere Anwendungen finden in der Gasausbreitung, der Luftströmungstechnik, der Geometrie, den Karten (Topologie) sowie in der Elektrotechnik (Filter) Platz.

9.2.8 Zelluläre Automaten

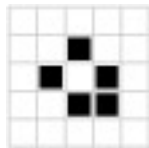
Zelluläre (auch zellulare) Automaten [Sennbill et al. 2004] dienen der Modellierung von diskreten dynamischen Systemen. Ein dynamisches System beschreibt die zeitliche Veränderung von Größen, zum Beispiel die zeitliche Veränderung von Populationszahlen konkurrierender Spezies (z. Bsp. Räuber und Beute). Zelluläre Automaten sind fähig, nicht numerische Simulationen von physischen, chemischen und biologischen Vorgängen durchzuführen und darzustellen.

Die Theorie der zellulären Automaten wurde in den 40-iger Jahren des letzten Jahrhunderts von John von Neumann (dem Erfinder der von Neumann Architektur/Maschine) entwickelt. Er hat jedoch auch Aspekte und Ideen von Stanislaw Ulam und Alan Turing einfließen lassen. Von Neumann hat die Theorie erarbeitet während er sich mit der Erforschung von Selbst-Reproduktion in der Biologie beschäftigte.

Im Wesentlichen geht es um eine Ansammlung von so genannten „Zellen“ auf einem Raster. Diese Zellen evolvieren aufgrund von Regeln, die auf den Zuständen der benachbarten Zellen beruhen. Das Raster kann sowohl in seiner einfachsten Form als einzelne Linie vorkommen, als auch in verschiedensten Ausprägungen über mehrere Dimensionen. Im Normalfall existieren nur wenige Zustände der Zellen. Trotz der Einfachheit dieser Automaten sind sie jedoch in der Lage, komplizierteste dynamische Systeme darzustellen.

9.2.9 Game of Life

Einer der bekanntesten zellulären Automaten im zweidimensionalen Raum ist das so genannte „Game of Life“ von J. H. Conway. Conway entdeckte und entwickelte die Regeln zum Game of Life im Jahr 1970. Mit jeder Generation der Zellen auf dem 2-dimensionalen Rastermuster des Game of Life, werden Zellen entweder aktiviert, deaktiviert oder in ihrem vorherigen Zustand belassen. Die Regeln im Game of Life sind einfach, aber effizient:



Wird der Zustand einer einzelnen Zelle geprüft, wird der Zustand aller 8 Nachbarzellen berücksichtigt. Alle Zellen die aktiviert sind werden summiert. Je nachdem wie viele das sind, kommen die folgenden Regeln zur Anwendung:

- *Tod*: Wenn nur 2 oder weniger Zellen rund um die getestete Zelle aktiv sind, „stirbt“ unsere Zelle, das heisst, sie wird deaktiviert.
- *Überleben*: Falls die Anzahl aktiver Zellen um unsere Zelle genau 2 oder genau 3 ist, bleibt der Zustand unserer Zelle unverändert.
- *Geburt*: Falls unsere Zelle deaktiviert ist und die Anzahl lebender Nachbarzellen genau 3 ist, wird unsere Zelle aktiviert; eine neue lebende Zelle ist geboren.

Einsatz und Sinn von zellulären Automaten

Heutzutage werden mit Hilfe von zellulären Automaten verschiedenste Abläufe im menschlichen Hirn erforscht. Zelluläre Automaten werden hierbei als eine vereinfachte

Vorstufe von neuronalen Netzwerken angesehen. Weiter wurden in letzter Zeit Forschungen zum Verbrauch und der Nutzung von Landfläche (Urban Evolution) von zellulären Automaten unterstützt. Auch in der Erforschung von Verkehrssituationen wie zum Beispiel Staus und des „Stop and go“-Phänomens werden zelluläre Automaten als Hilfsmittel eingesetzt.

9.3 Grundlegende Konzepte

Dieses Kapitel umfasst eine Sammlung von grundlegenden Konzepten, die für die Interpretation von Informationen von zentraler Bedeutung sind.

Folgende Konzepte und Theorien werden vorgestellt:

- Rekursivität (Beitrag von Micha Häusermann)
- Backtracking (Beitrag von Anja Schicker)
- Bayes'sche Filter (Beitrag von Erkan Arslan)
- Fast Fourier Transformation (Beitrag von Fabian Boller)
- Blackboard Systems (Beitrag von Marius Lang)

9.3.1 Rekursivität

Rekursivität kommt von Rekursion was Selbstbezüglichkeit bedeutet (lat. Recurrere = zurücklaufen). Aus der mathematischen Sicht beschreibt die Rekursion Prozesse die sich so lange selbst wieder aufrufen bis eine vordefinierte Randbedingung erfüllt ist. Aus sprachlicher Sicht bedeutet Rekursion das Zurückführen von Aussagen bis zu einer klaren Definition. Die Mengenlehre sagt aus mit Rekursion ist es möglich aus endlichen Elemente eine unendliche Menge zu beschreiben.

Definition Rekursion:

Rekursion ist ein Aus der Mathematik übernommener Begriff, der in der Linguistik die formalen Eigenschaft von Grammatiken bezeichnet, mit einem endlichen Inventar von Elementen und einer endlichen Menge von Regeln eine unendliche Menge von Sätzen zu erzeugen [Bussmann 2002].

Beispiel: Fibonacci-Reihe:

$$f(0) = 1$$

$$f(1) = 1$$

$$f(n+2) = f(n) + f(n+1)$$



Beispiel: Rekursive Aussage:

„Der folgende Satz ist wahr“- „Der vorhergegangene Satz ist nicht wahr“

Listing 9.1 Fakultätsberechnung mit Java

```
public class faculty {  
    public static long faculty(int n) {  
        return n==0?1:n*facultyRec(Math.abs(n)-1);  
    }  
    public static void main(String[] args) {  
        System.out.println("Fakultät von 7 ist: " + faculty(7));  
    }  
}
```

Vergleich Iteration / Rekursion

In den meisten Fällen lassen sich Probleme sowohl rekursiv als auch iterativ lösen. Auf die Programmierung bezogen bedeutet aber ein rekursiver Ansatz meist mehr Zeit und Speicherbedarf, da bei jedem Rekursionsschritt die Variablen auf den Stack gelegt werden müssen und ein zusätzlicher Funktionsaufruf durchgeführt werden muss. Meist ist jedoch die rekursive Lösung eleganter in der Implementation.

Infinite Rekursion

Lässt man bei der Rekursionsdefinition die Abbruchbedingung weg, so erhält man eine infinite Rekursion (Endlos-schleife). Um diesen in der Informatik meist unerwünschten Effekt zu beseitigen, verwendet man ein „inSchleufeMethode“ mit eigener Abbruchbedingung.

Anwendungen

- *Informatik:* Suchalgorithmen
- *Informatik:* Sortieralgorithmen
- *Informatik:* Compilerbau (recursive descent)
- *Informatik:* Grundlage der Definition von funktionalen Sprachen
- *Mathematik:* Fraktale / Mandelbrotmenge
- *Mathematik:* Sierpinski Dreieck
- *Mathematik:* Grundlage effizienter Algorithmen

9.3.2 Backtracking

Definition Backtracking:

Backtracking (backtrack – zurückverfolgen) ist eine systematische Art der Suche in einem vorgegebenen Lösungsraum. Backtracking ist eine Tiefensuche [Kondrak, van Beeck 1995].

Tiefensuche ist ein Fachbegriff, welcher ein Verfahren zum Durchsuchen bzw. Durchlaufen der Knoten und Blätter einer baumartigen Datenstruktur bezeichnet. Tiefensuche steht im Gegensatz zur Breitensuche, wobei letztere im Allgemeinen speicheraufwändiger ist. Dafür terminiert die Tiefensuche nicht, falls mindestens ein Pfad des Baumes unendliche Länge hat. Backtracking ist eine Heuristik-Methode. Heuristiken sind „Strategien, die mit höherer Wahrscheinlichkeit (jedoch ohne Garantie) das Auffinden einer Lösung beschleunigen sollen.“

Funktionsweise

Beim Backtracking startet man an der Wurzel und geht so weit wie möglich entlang der bestehenden Kanten in die Tiefe, ehe man zurückgeht (englisch: backtracking) und dann in bislang unbesuchte Teilbäumen absteigt.

Formell betrachtet ist Tiefensuche eine uninformierte Suche, die voranschreitet durch expandieren des ersten auftretenden Kind-Knotens des Suchbaumes und so tiefer und tiefer geht bis sie einen Zielzustand gefunden hat oder einen Knoten, der keine Kinder hat. Dann geht die Suche zurück um von einem höher gelegenen Knoten wieder abzusteigen.

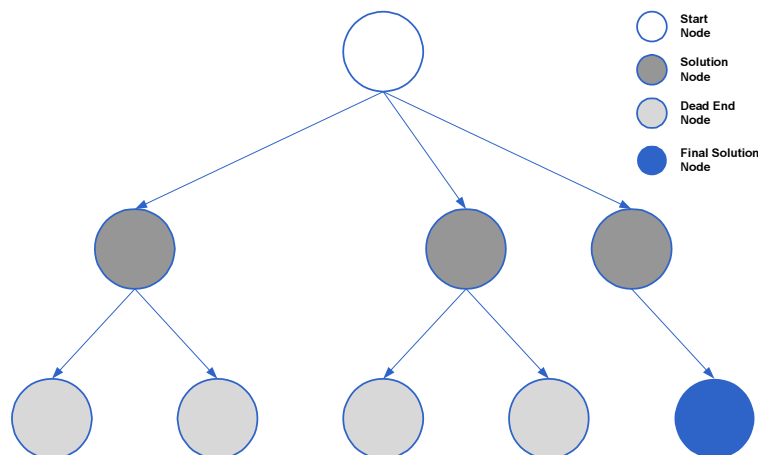


Abbildung 9.8 Backtracking

Algorithmisch betrachtet werden alle gerade expandierten Knoten vorne in die Suchwarteschlange (Datenstruktur) zur weiteren Expansion gestellt.

- Tiefensuche ist *vollständig*: wenn der Baum endlich ist, wird eine Lösung gefunden, falls sie existiert

- Tiefensuche ist *optimal*, falls der Baum endlich ist und alle Pfadkosten nicht-negativ sind.

Listing 9.2 Backtracking Algorithmus

```
boolean FindeLoesung(int index, Lsg loesung, ...) {
    // index ist die aktuelle Schrittzahl
    // Teillösungen loesung werden als Referenz übergeben.
    Solange es noch neue Teil-Lösungsschritte gibt:
        a) Wähle einen neuen Teil-Lösungsschritt schritt; // Heuristik
        b) Falls schritt gültig ist:
            I)Erweitere loesung um schritt;
            II) Falls loesung vollständig ist, return true, sonst:
                if (FindeLoesung(index+1,loesung)) { // rekursiv
                    return true; // Lösung gefunden
                } else { // Wir sind in einer Sackgasse
                    Mache schritt rückgängig; // Backtracking
                }
            }
        Gibt es keine neuen Teil-Lösungsschritte mehr, so: return false
    }
```

Anwendungen

Es gibt verschiedenste Anwendungsgebiete, welche die Heuristik des Backtracking nutzen:

- Die Programmiersprache Prolog, auf welcher die verschiedensten Wissensbasierter Systeme aufgebaut sind arbeitet mit Backtracking
- Suchmaschinen
- Browserfunktionalität ‚Back‘
- Künstliche Intelligenz

9.3.3 Bayes'sche Filter

Die gigantische Vielfalt und das rasante Wachstum in der modernen Informationsgesellschaft führt zu Problem bei der Informations- Beschaffung. Um diesen Missstand Abhilfe zu schaffen wird versucht Computerprogramme durch die Nachbildung menschlicher Problemlösefähigkeiten(Kognitionswissenschaften) die Systeme „intelligenter“ zu machen. Als Model werden der Mensch und dessen Aneignung von Wissen verwendet. Der Mensch erlangt einen Teil des Wissens in der Schule, Lehre, Studium oder sonstigen Bildungsstätten aber der grosse Teil wird durch Erfahrung gesammelt. Es treten oftmals Situationen auf die nicht behandelt wurden.

Hier kommen zwei verschieden Möglichkeiten zum Zug:

- Wenn die Situation in ähnlicher Form schon einmal aufgetreten ist kann der Mensch daraus Schlüsse für diese Situation ziehen um sie zu lösen.
- Falls noch keine Erfahrung gesammelt wurde, wird eine Entscheidung getroffen und aus dieser neu erworbenen Erfahrung kann zu einem späteren Zeitpunkt Gewinn gezogen werden.

Das Verfahren von Bayes bietet dazu die Möglichkeit vorhandenes und neues Wissen miteinander zu verbinden. Es gibt die Möglichkeit bestehendes Wissen zu nutzen und immer weiter zu vervollständigen. Hierzu sind Beispiele als eine Wissensbasis vorhanden und es werden Hypothesen aufgestellt, deren Wahrscheinlichkeit anhand der Beispiele nachgewiesen wird. Neue Beispiele beeinflussen die aufgestellte Hypothese. Das bestehende Wissen und diese neuen Fakten fließen dann zusammen und das Wissen vervollständigt sich.

Bayes Theorem

Im Bereich des maschinellen Lernens gilt das Interesse oftmals der Bestimmung der Hypothese, die auf beobachtete Daten am meisten zutrifft. Die beste Hypothese ist jene, die nach hinzufügen von neuem Wissen die grösste Wahrscheinlichkeit besitzt. Das Bayes Theorem liefert diese Wahrscheinlichkeit. Das Bayes-Theorem (oder auch Satz von Bayes) ist ein Ergebnis der Wahrscheinlichkeitstheorie benannt nach seinem Erfinder dem Mathematiker Thomas Bayes (1702-1761) der in London geboren wurde und Pfarrer der Gemeinde „Little Mount Sion“ war. Als Student in London, eignete er sich die mathematischen Kenntnisse an. Bayes erbrachte wichtige Beiträge zur Wahrscheinlichkeitstheorie und sein „Essay towards solving a problem in the doctrine of chances“ enthielt unter anderem einen Spezialfall der unter seinen Namen berühmt gewordenen Formel [Bayes 1763].

Formel von Bayes:



$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

$P(A|B)$ die (bedingte) Wahrscheinlichkeit des Ereignisses A unter der Bedingung, dass B eingetreten ist

$P(B|A)$ die (bedingte) Wahrscheinlichkeit des Ereignisses B unter der Bedingung, dass A eingetreten ist

$P(A)$ die A-priori-Wahrscheinlichkeit des Ereignisses A

$P(B)$ die A-priori-Wahrscheinlichkeit des Ereignisses B

Die grosse Bedeutung des Satzes von Bayes liegt in der Verarbeitung von Informationen, was zu einer Verbesserung der Wahrscheinlichkeitsaussage für den Eintritt eines Ereignisses führt. Vor der Durchführung des Zufall Vorgangs beträgt die Eintrittswahrscheinlichkeit für das Ereignis A gleich $P(A)$. diese Wahrscheinlichkeit wird daher auch als a-priori-Wahrscheinlichkeit bezeichnet. Ist Ereignis B eingetreten, so kann diese Information durch Anwendung des Satzes von Bayes zu einer verbesserten oder korrigierten Wahrscheinlich-

keit $P(A | B)$ für den Eintritt von Ereignis A verarbeitet werden. Diese Wahrscheinlichkeit wird daher auch als a posteriori Wahrscheinlichkeit bezeichnet. Diese korrigierte Grösse liefert eine bessere Basis für die Entscheidungsfindung

Formel von Bayes für endlich viele Ereignisse:

$$P(A_i|B) = \frac{P(B|A_i) \cdot P(A_i)}{\sum_{j=1}^n P(B|A_j) \cdot P(A_j)}$$

Wenn $A_i; i = 1..n$ eine Zerlegung der Ergebnismenge in disjunkte Ereignisse ist, gilt für die A-posteriori-Wahrscheinlichkeit $P(A|B)$

Umsetzung der Bayes'schen Eigenschaften

Die Methode der Bayes'schen Filter ist zur Zeit sehr populär und eine der erfolgreichsten in der Wahrscheinlichkeit von Hypothesen.

Sie findet Anwendung in:

- Moderne Spamblocker für Email oder SMS geben einen kleinen Vorgeschmack auf das Potenzial der Bayes basierenden Erkennungstechniken [Keinhörster, Sandhaus 2013]. Weit treffsicherer als jeder regelbasierte Filter, entscheidet der Bayes-Filter anhand von Worthäufigkeiten im Posteingang ob ein Mail Spam(unerwünschte Nachricht) oder Ham (erwünschte Nachricht) ist. Bevor Nachrichten mit Hilfe der Bayes'schen Methode gefiltert werden können, muss eine Datenbank mit Wörtern erstellt werden die aus Spam-Beispielen und gültigen Mitteilungen stammen. Da sich der Bayes'sche Anti Spam Filter automatisch an die Benutzerkorrespondenz anpasst kann er für jede Unternehmung oder Sprache verwendet werden, z.B. würde das Wort „Hypothek“ bei einer Bank einen andere Wertigkeit erhalten als bei einem Fahrrad Händler.
- Intel soll eine Gruppe von Bayesian- Softwarebibliotheken freigeben die den Softwareentwicklern helfen soll bessere Maschinenlernmöglichkeiten zu ihren Programmen zu entwickeln.
- Microsoft investiert grosse Forschungsarbeiten in den Einsatz von entscheidungstheoretischen Nutzermodellen, das von der Wahrscheinlichkeit des Eintretens bestimmter Ereignisse handelt anhand des Bayes'schen Theorem. Intelligente Bedienerführung mit Attention User Interface(AUI) sollen die Kommunikation mit Computern künftig erleichtern. Der Computer reagiert dabei auf nutzer- und kontext- abhängige Interessen. Eines von vielen AUI-Projekten ist z.B. Lumiere, mit dem die Technologie für den Assistenten bei Office XP entwickelt wurde.

Neben der Biometrik und Chemie ist ein weiterer Anwendungsbereich das Wissensmanagement. So hat die Firma Autonomy, die als erste Kommerzielle Firma im Jahr 1996 ein mit dem Bayes Theorem der bedingten Wahrscheinlichkeiten ausgestattete Software auf den Markt gebracht. Die Funktion des entwickelten Recherchesystem für Computer-

Anwendungen analysiert sämtliche eingaben eines Benutzer, gruppiert (Klassen) diese nach Kennwörter und erstellt durch die Verwendung der Formel für die bedingte Wahrscheinlichkeit einen inneren Zusammenhang zwischen den Kernbegriffen des Textes. Mit diesem Wissen ausgestattet kann eine Suchmaschine dem Anwender Webseiten oder Dokumente empfehlen, die für den oder die eingegebenen Texte inhaltlich relevant sind. Es ist in diesem Zusammenhang auch zu erwähnen das solche Systeme sich sehr gut für die Charakterisierung von Personen eignen.

9.3.4 Fast Fourier Transformation

1755 fand der Mathematiker Bernoulli eine andere Lösungsform für die schwingende Saite, indem er stehende Wellen verwendete. Als Fourier seine Gleichung der Wärmeleitung herleitete, entdeckte er, dass sie ebenfalls spezielle Lösungen hat, die sich in eine Funktion des Orts multipliziert mit einer Funktion der Zeit zerlegen lassen. Er verkündete, dass jede Temperaturverteilung, also jeder Graph, unabhängig davon, aus wie vielen Einzelstücken er besteht, durch eine Reihe von Sinus- und Kosinusfunktionen dargestellt wird.

Im Gegensatz zur eindimensionalen Logarithmus-Transformation, die einen einzelnen Wert X in einen einzelnen Wert $\log(X)$ transformiert, transformiert die Fourier-Transformation eine Funktion einer Variable in eine Funktion einer anderen Variable, jeweils definiert von $-\infty$ bis $+\infty$. Die Hauptaufgabe der Fourier-Transformation besteht darin, ein beliebiges Signal in eine Summe von Sinusfunktionen unterschiedlicher Frequenz, Amplitude und Phase zu zerlegen.

Entstehungsgeschichte

Während einer Sitzung des wissenschaftlichen Beratungskomitees des US-Präsidenten zu Beginn der 60er Jahre des letzten Jahrhunderts stellte Richard Garwin fest, dass John Tukey sich mit der Erstellung von Programmen für die Fourier Transformation beschäftigte. Garwin war für seine Forschungsarbeit auf der Suche nach einer schnellen Methode zur Berechnung der Fourier-Transformation. Tukey erklärte Garwin im Wesentlichen das, was später zu dem berühmten Cooley-Tukey-Algorithmus führte.

Garwin ließ das Verfahren im IBM Forschungszentrum programmieren. Dort wurde James Cooley mit der Bearbeitung der Aufgabe beauftragt. Nachdem er das Programm für Garwin fertiggestellt hatte, häuften sich die Nachfragen für Kopien für das Programm und Cooley wurde zu einer Veröffentlichung darüber gebeten. 1965 publizierte Cooley und Tukey den jetzt weltberühmten Aufsatz "An Algorithm for the Machine Calculation of Complex Fourier Series" [Cooley, Tukey 1965]. Nach der Publikation wurde bekannt, dass sich auch andere Personen ähnlicher Verfahren bedienten.

Anwendungsgebiete

Die Fourier-Transformation hat sich zur Problemvereinfachung in vielen wissenschaftlichen Bereichen als sehr effektiv erwiesen. Eine sehr einfache Anwendung ist die Frequenzfilterung. Da man eine Funktion, bzw. ein Signal vom Zeitbereich in den Frequenzbereich transformiert, kann man nun leicht Frequenzen verändern, löschen oder hinzufügen, bevor man das Signal wieder rücktransformiert. Weitere Anwendungsgebiete sind Lineare Systeme, Antennen, Optik, Stochastische Prozesse, Wahrscheinlichkeitstheorie, Quantenphysik oder Randwert-Probleme.

9.3.5 Blackboard Systems

Blackboard Systeme sind keine neuen Technologien. Das erste System mit dieser Architektur war ‚Hearsay 2‘ und wurde in den Jahren 1971-1976 von Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy entwickelt (Thematik Sprachverstehen) [Erman et al. 1980]. Hauptsächlich wird diese Technologie bei komplexen Systemen eingesetzt.

Das Blackboard-Modell besteht aus drei Komponenten:

- Die *Blackboard* ist eine Datenbank mit diversen Daten wie: Problembeschreibende Daten., Teillösungen, Themenbeiträge, allg. Informationen etc.
- *Knowledge Sources* sind unabhängige Module, die Wissen beinhalten und selbst aus unterschiedlichen Architekturen aufgebaut sein können.
- Die *Control Component* fällt Laufzeit-Entscheide betreffend der Aktivitäten zur Lösung eines Problems. Ausserdem kontrolliert die Control Component auch die Zugriffe der Knowledge Sources auf die Blackboard Datenbank.

Charakteristiken eines Blackboard Systems sind:

- Independence of Expertise (die Knowledge Sources sind absolut unabhängig voneinander)
- Diversity in Problem Solving Techniques (unterschiedlichste Techniken, Architekturen, Algorithmen)
- Flexible Representation of Blackboard Information
- Common Interaction Language
- Positioning Metrics
- Event Based Activation
- Need for Control

Anwendungsgebiete

Einige Bereiche die Blackboard Systems eingesetzten sind:

- Knowledge-Based Simulation, Symbolic Learning

- Process Controlling, Planning and Scheduling

Das System besteht aus den drei folgenden Hauptkomponenten:

- The Blackboard
- Knowledge Sources
- Control Component

Tabelle 9.8 Kommerzielle Systeme

Hersteller	Kommentar
CRYSTALIS	A blackboard based project (1976-1983) to build a system for determining the location of a protein's atoms in space relative to each other using X-ray data.
KSAR	Knowledge Source Activation Record representing a specific knowledge source action that is initiated by a specific blackboard event.
HASP	Uses blackboard technology to process real-time sonar signals (1982).
HEARSAY-II	The first blackboard system evolving between 1971-1976 for speech understanding of single spoken words out of a list of 1,000.
OPM	This blackboard application planned multiple-task actions given conflicting goals and constraints.

9.4 Grundlegende Theorien

Dieses Kapitel umfasst folgende Basistheorien:

- Shannon's Informationstheorie (Beitrag von Roberto Nibali)
- Komplexitätstheorie (Beiträge von Nicolas Magro und Flurin Capaul)
- Queuing Theory (Beitrag von Dominik Hof)
- Game Theory (Beitrag von Christoph Pfyffer)
- Inference (Beitrag von Andreas Büchler)
- Proportional and Predicate Logic (Beitrag von Manuel Permuy)
- Fuzzy Logic (Beitrag von Ferat Jakupi)

9.4.1 Die Informationstheorie von Shannon

Nach Claude Elwood Shannon ist Information definiert als "*a measure of one's freedom of choice when one selects a message*" [Shannon 1948]. Information und Ungewissheit sind in der Informationstheorie eng verwandt. Die Information bezieht sich auf den Grad der Unsicherheit in der momentanen Situation. Je mehr Ungewissheit man von einer Nachricht entzieht, desto grösser ist die Beziehung zwischen Input und Output eines Informationskanals. Mit der Ungewissheit bezieht man sich auch aufs Konzept der Vorhersagbarkeit. Wenn etwas komplett vorhersagbar ist, dann ist es bestimmt und enthält in diesem Fall (wenn überhaupt) sehr wenige Informationen.

Ein verwandter Begriff, die Entropie gehört auch zu den wichtigen Konzepten der Informationstheorie. Mit Entropie bezieht man sich auf den Grad der Wahllosigkeit, dem Mangel an Organisation oder einer Unordnung in einer Situation. Die Informationstheorie gibt alle Informationsquantitäten in bits an.

Redundanz ist ein weiteres Konzept, welches seine Wurzeln in der Informationstheorie hat. Redundanz ist das Gegenstück zur Information. Etwas, das redundant ist, ergänzt den Informationsgehalt der Nachricht (wenn überhaupt) nur gering. Redundanz ist jedoch wichtig, damit man dem Rauschen in einem Kommunikationssystem entgegenkommen kann.

Shannon's Formel für diskrete Informationsübertragung:

$$C = \lim_{T \rightarrow \infty} \frac{\log_2 T}{T}, H = -\sum_{i=1}^M P_i \cdot \log_2 P_i, R = H_{\text{before}} - H_{\text{after}}$$

C: Kapazität des Kanals

N(T): Anzahl der erlaubten Signale während der Zeit T

H: Entropie (Überraschungsfaktor)

M: Die Anzahl unterscheidbarer Symbole

P: Wahrscheinlichkeit des Auftretens eines Symbols

R: Information oder die Redundanzfreiheit

Shannon's Formel für kontinuierliche Informationsübertragung:

$$C = \lim_{T \rightarrow \infty} \max_{P(x)} \frac{1}{T} \iint P(x, y) \cdot \log \frac{P(x, y)}{P(x) \cdot P(y)} dx dy$$

Shannon's Ziel war einfach; Die Verbesserung der Übertragung von Informationen \über den Telegraphen oder eine Telefonleitung bei elektrischer Interferenz oder Rauschen. Die Formeln, welche das Problem mathematisch beschreiben gehören zu den wichtigsten Errungenschaften der Informationstheorie:

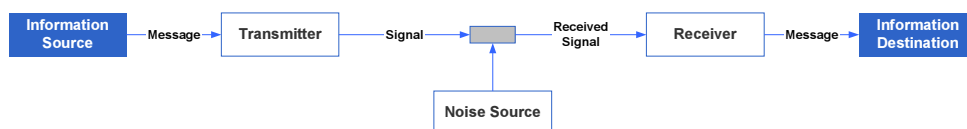


Abbildung 9.9 Das grundlegende Modell für die Kommunikation

Das (mathematische) Modell für die Kommunikation ist in **Abbildung 9.9** dargestellt.

- Die *Information Source* produziert eine Nachricht, welche übertragen werden soll, die Quelle zu Deutsch.
- Der *Transmitter* sorgt dafür, dass die Nachricht kodiert und auf das Medium übertragen wird.
- Der *Channel* ist lediglich das Medium, auf welchem das Signal übertragen wird.
- Der *Receiver* führt die inverse Operation des Transmitters aus, er rekonstruiert die originale Nachricht aus dem Signal.
- Die *Destination* schliesslich ist die Person (oder das Ding), für welche die Nachricht bestimmt ist, die Senke zu Deutsch.
- Die *Noise Source* ist der Rauschgenerator.

Beispiel: 7-Segment Anzeige:

Alle Ziffern haben dieselbe Wahrscheinlichkeit: $\Pr\{X_v\} = 0.1$

Der Informationsgehalt pro Ziffer: $I(X_v) = -\log_2(\Pr\{X_v\}) = \log_2(10) = 3.32 \text{ bit}$

Der durchschnittliche Informationsgehalt: $H(X_v) = \Pr\{X_v\} * I(X_v) = 3.332 \text{ bit}$

Die Entropie des Alphabets: $H(X_v) = \sum_v H(X_v) = 3.32 \text{ bit}$

Die absolute Redundanz: $R = m - H(X) = 7 \text{ bit} - 3.32 \text{ bit} = 3.68 \text{ bit}$

Die relative Redundanz: $r = R/m = 3.68 \text{ bit}/7 \text{ bit} = 52.54 \%$

Anwendungsgebiete

Die Theorie von Shannon hat weitreichende Konsequenzen und Erkenntnisse unter anderem in folgenden Gebieten der Forschung gebracht: Computer Technologie, Physik, Molekularbiologie, Biotechnik, Psychologie, Linguistik und Kommunikationstheorie.

Eine interessante Anwendung ist zum Beispiel das Erkennen von Informationen in der Bildverarbeitungstheorie im medizinischen Bereich. Des Weiteren basiert die differentielle Kryptoanalyse teilweise auf den Theoremen von Shannon.

9.4.2 Komplexitätstheorie

Die Komplexitätstheorie ist ein Kerngebiet der theoretischen Informatik. Sie ist sehr stark mit dem Gebiet „Effiziente Algorithmen und Datenstrukturen“ verbunden, die sich mit der Entwicklung und Analyse von Algorithmen zur Problemlösung befasst. Die Komplexitätstheorie lotet im Prinzip die Grenzen des mathematisch machbaren aus. Dies in dem sie versucht, den Mindestressourcenbedarf zur Lösung eines Problems oder einer Problemklasse zu bestimmen [Hartmanis, Stearns 1965]. Dabei muss der lösende Algorithmus nicht zwingend schon bekannt sein.

Die Komplexitätstheorie hat einen völlig anderen Ansatz als die Algorithmen. Sie untersucht, welche Probleme „schwer“ zu lösen sind. „Schwer“ bedeutet hier sehr viele und/oder teure Ressourcen (z.B. Rechenzeit, Speicher, Hardware,...). Sie versucht also nachzuweisen, dass schwierige Probleme nicht mit geringem Ressourceneinsatz zu lösen sind.

Die Komplexitätstheorie hat dabei folgende Auswirkungen auf den Entwurf von Algorithmen:

- Da der minimale Ressourcenbedarf bestimmt wird, wird die Suche nach einer Lösung mit geringerem Ressourcenbedarf überflüssig
- Sie kann die Möglichkeit einer billigeren approximativen Lösung oder einer eingeschränkten Speziallösung begünstigen
- Sie kann Anregungen zu neuen algorithmischen Ansätzen und Verfahren liefern.

Beim Entwurf von Algorithmen wird also die obere Schranke gesucht, d.h. es kann eine Lösung geben, die besser ist. Die Komplexitätstheorie liefert die untere Schranke, also den minimalen Ressourcenbedarf. Bei einem optimalen Algorithmus fallen beide Schranken zusammen. Dies ist z.B. bei Suchalgorithmen der Fall. Wichtig dabei ist, dass die Aussagen über die Mindestressourcen alle Algorithmen betreffen, also auch die noch unbekannten!

Zeit- und Platzkomplexität

Es werden zwei Komplexitätsarten unterschieden:

- *Zeitkomplexität*: Die Ausführungsdauer eines Algorithmus.
- *Platzkomplexität*: Der Speicherverbrauch eines Algorithmus

Definition Komplexität:

Die Komplexität wird als Funktion der Anzahl der Eingabeelemente formuliert.

Die so genannten Landau Symbole beschreiben Mengen von Funktionen mit ähnlichen Wachstumsverhalten.

$O(n)$: obere Schranke

$\Omega(n)$: untere Schranke

$\Theta(n)$: Mittlere Schranke

Definition obere Schranke:

Für zwei Funktionen $f(n)$, $g(n)$ gilt, wenn $g(n)$ bis auf einen konstanten Faktor c höchstens so stark wie $f(n)$ wächst.

Tabelle 9.9 Aufwandsklassen

$g(n)$	Bezeichnung	$g(10)$	$g(1'000)$	$g(1'000'000)$
1	Konstant	1	1	1
$\log n$	Logarithmisch	3	10	20
n	Linear	10	1'000	1'000'000
$n * \log n$	Log – linear	30	10'000	$2 * 10^7$
n^2	Quadratische	100	1'000'000	10 ¹²
n^3	Kubisch	1'000	10 ⁹	10 ¹⁸
$2n$	Exponentiell	1'000	10300	10300'000

P und NP Computing

Alle Probleme mit der Laufzeit von $O(n^k)$ werden als in Polynomialzeit lösbar bezeichnet. Die Klasse aller Probleme, die sich auf einer deterministischen sequentiellen Maschine (z.b. Von Neumann Maschine) in Polynomialzeit lösen lassen, wird als **P** bezeichnet.

Die Klasse aller Probleme, die sich von einer (hypothetischen) nichtdeterministischen Maschine lösen lassen, wird als **NP** (Nondeterministic-Polynomial) bezeichnet.

Die zentrale Frage der theoretischen Informatik ist, ob die Klasse **NP** mit der Klasse **P** übereinstimmt. Heute wird angenommen, dass dies nicht der Fall ist. Das bedeutet, dass es eine Reihe von Problemen gibt, die nicht in endlicher Zeit von einer Maschine gelöst werden können [Vardi 2010].

Beispiele solcher Probleme sind:

- Erfüllbarkeit einer Aussage
- Verpackungsprobleme
- Zahlentheoretische Probleme
- Rundreiseproblem
- Optimierung von Flüssen

9.4.3 Queuing Theory

Der Begriff „Queuing Theory“ ins Deutsche übersetzt heisst Warteschlangentheorie. Diese Theorie ist ein Teilbereich der Wahrscheinlichkeitstheorie und ein Beispiel für die Angewandte Mathematik. Dabei geht es hauptsächlich um die Frage, wie die zu erwartende mittlere Wartezeit ist. Warteschlangen wurden seit 1909 systematisch vor allem für die Telekommunikation erforscht; diese lässt sich aber auf viele andere Bereiche übertragen [Gross et al. 2013].

Das Phänomen des Wartens wird seit fast einem Jahrhundert wissenschaftlich erforscht. Bereits 1917 publizierte der dänische Ingenieur und Mathematiker A.K. Erlang, der bei einer Kopenhagener Telefongesellschaft beschäftigt war, eine mathematische Formel, mit deren Hilfe man Fernsprechvermittlungsstellen dimensionieren kann. Inzwischen sind mehr als 10 000 wissenschaftliche Publikationen über Warteschlangenprobleme erschienen, die sich auf die unterschiedlichsten Bereiche unseres täglichen Lebens erstrecken.

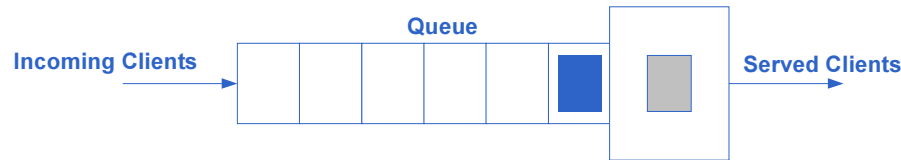


Abbildung 9.10 Das Grundmodell der Queuing Theory

Die Warteschlangentheorie bedient sich zur Beschreibung von Bedienungssystemen eines einfachen Grundmodells. Es besteht aus dem so genannten Bedienungsschalter, der über ein oder mehrere parallel arbeitende gleichartige Maschinen bzw. Arbeitsplätze verfügt, und aus einem Warteraum. Die Kunden treffen einzeln und zu zufälligen Zeitpunkten vor dem Bedienungsgesamt ein. Ein neu ankommender Kunde wird bedient, sofern mindestens eines der Bedienungsgesamte frei ist, andernfalls muss er sich in die Warteschlange einreihen.

Ziel der Warteschlangentheorie ist es nun, Grundlagen für die Dimensionierung von Bedienstationen bereitzustellen, bei denen die Gesamtkosten minimal werden.

Zu diesem Zweck werden die Bedienstationen mit Hilfe bestimmter Kenngrößen charakterisiert, die mittels spezieller Warteschlangenmodelle abgeleitet werden.

Solche Kenngrößen sind beispielsweise:

- Durchschnittliche Auslastung
- Durchschnittliche Länge der Warteschlange
- Durchschnittliche Wartezeit
- Durchschnittliche Bedienzeit
- Durchschnittliche Anzahl Kunden im System
- Durchschnittliche Anzahl Kunden in der Bedienstation

Tabelle 9.10 Bedingungsregeln für die Abfertigung in Warteschlangen

Regel	Erklärung
FIFO	First In, First Out. Die Bedienung erfolgt in der Reihenfolge der Ankünfte.
LIFO	Last In, First Out. Die Bedienung erfolgt in umgekehrter Reihenfolge der Ankünfte.
SIRO	Selection In Random Order. Der nächste Kunde wird zufällig ausgewählt.
Non-preemptive	Relative Priorität. Manche Kunden werden gegenüber anderen Kunden vorran-

priority	gig behandelt. Der laufende Bedienungsprozess wird jedoch nicht unterbrochen.
Preemptive priority	Absolute Priorität. Besitzt der neu ankommende Kunde gegenüber den anderen Kunden im System eine höhere Priorität, so wird der laufende Bedienungsprozess unterbrochen und mit der neuen Forderung fortgesetzt. Die alte Forderung wird zurückgestellt.
RR	Round Robin. Jeder Kunde kann den Bediener jeweils nur für ein bestimmtes Zeitintervall in Anspruch nehmen. Kunden, deren Abfertigung mehr Zeit benötigt, müssen sich deshalb mehrmals hintereinander in die Warteschlange einreihen

Anwendungsgebiete

Beispiele aus unserem täglichen Leben, bei welchem wir mit Warteschlangen konfrontiert werden sind in der nachfolgenden Tabelle aufgelistet.

Tabelle 9.11 Bedingungsregeln für die Abfertigung in Warteschlangen

Kunden	Bedienstation	Abfertigung
Patienten	Arzt	Behandlung
Bankkunde	Kassierer	Geldauszahlung
Programm	CPU	Programmschritt ausführen
Anfrager	Helpdesk	Antwort
Maschinist	Servicemann	Reparatur

9.4.4 Game Theory

Die Game Theory (Spieltheorie) untersucht das rationale Verhalten von Interagierenden Personen in verschiedenen Lebenssituationen (Games). Die Spieltheorie wurde 1944 von John von Neumann und Oskar Morgenstern in ihrem Artikel "The Theory of Games and Economic Behavior" formuliert [von Neumann, Morgenstern 2004].

Grundsatz der Game Theory:

Es existieren Parallelen zwischen Gesellschaftsspielen wie Schach, Poker, Mensch ärgere dich nicht, etc. und den Entscheidungsproblemen in der Ökonomie und Politik.

Diese Gemeinsamkeiten sind:

1. Es sind nur wenige Entscheidungsträger involviert (nicht einer, nicht viele).
2. Der Nutzen (Kosten, Gewinn) eines Entscheidungsträgers ist abhängig von den Handlungen anderer. Dieser Effekt wird auch Interdependenz genannt.

Überall, wo Konkurrenz von "Individuen" um Ressourcen zu untersuchen ist, können spieltheoretische Untersuchungen eingesetzt werden, um entweder vorhandenes Verhalten zu erklären, oder aber verbesserte Strategien zu entwerfen. Im Forschungsbereich der Künstlichen Intelligenz und "Artificial Life" werden Agenten so programmiert, dass sie in

der Umwelt überleben. Dieses Überleben wird als Konkurrenzsituationen umgesetzt, in denen die Agenten mit anderen Agenten oder realen Objekten um Ressourcen streiten. Die Spieltheorie kommt sehr oft in Strategie-Games zur Anwendung.

Anwendungsgebiete

Tabelle 9.12 Beispiel Gefangenendilemma: Die beiden Gefangenen Albert und Bob stehen vor einer Entscheidung. Sollen Sie gestehen oder schweigen?

		Albert	
		Gestehen	Schweigen
Bob	Gestehen	10,10	0,20
	Schweigen	20,0	1,1

- *Dominante Strategie:* Die Strategie, mit der ein Spieler immer besser fährt, egal was sein Mitspieler macht.
- *Dominantes Strategie-Equilibrium:* Ergebnis, wenn alle Spieler die dominante Strategie wählen.

Tabelle 9.13 Beispiel Nullsummenspiel: Die beiden Spieler Albert und Bob müssen sich entscheiden. Sollen Sie Schere, Stein oder Papier spielen?

		Albert		
		Schere	Stein	Papier
Bob	Schere	0,0	-1,1	1,-1
	Stein	1,-1	0,0	-1,1
	Papier	-1,1	1,-1	0,0

- *Maximin-Kriterium:* In einem Nullsummenspiel muss die rationale Strategie den minimalen Gewinn maximieren.
- *Gemischte Strategie:* Diese Strategie wird zufällig aus einem Set von möglichen Strategien gewählt.

Tabelle 9.14 Beispiel Noconstant Sum Games aus der Wirtschaft

		Perrier		
		Price = 1\$	Price = 2\$	Price = 3\$
Evian	Price = 1\$	0,0	50,-10	40,20
	Price = 2\$	-10,20	20,20	90,10
	Price = 3\$	-20,40	10,90	50,50

- *Nash-Equilibrium:* Existiert eine Menge von Strategien, so dass kein Spieler profitiert, wenn er seine Strategie ändert während der andere seine beibehält. Die Menge dieser Strategien und der dazugehörige Profit ist das Nash-Equilibrium. Ein Dominantes Strategie-Equilibrium ist ein Nash-Equilibrium [Nash 1950].

9.4.5 Inference

Für einen Computer ist es relativ einfach z.B. ein Stück Source-Code auf die syntaktische Gültigkeit zu überprüfen. Bekanntlich schwieriger wird es, wenn die Semantik überprüft werden soll, oder eine Aussage evaluiert werden muss (z.B. eine Supportanfrage). Eine Möglichkeit dieses Problem anzugehen, besteht darin aus vorhandenem Wissen, z.B. aus einer Knowledge Database, Schlüsse auf das zu lösende Problem zu ziehen. Eine solche Database besteht aus verschiedenen inference rules die dann von einer inference engine verarbeitet werden um aus den Regeln Schlüsse zu ziehen [MacKay 2003].

Definition: Inference:

Als Schlussfolgerung, Konklusion oder Schlussatz bezeichnet man in der Logik die sich im Rahmen eines Beweises ergebende hergeleitete Aussage.

Im allgemeinen Sprachgebrauch versteht man unter einer Schlussfolgerung auch das Durchführen eines Beweises, also das Schlussfolgern.

Für die automatisierte, Computergestützte Schlussfolgerung wird auch häufig der Begriff Inferenz (engl. inference oder reasoning) verwendet.

Inference Rules

Ein Beispiel für eine solche Regel könnte lauten: *“Falls der Käufer ein interner Kunde ist, ist der Rabatt 20% höher,..”*. Die Struktur einer inference rule besteht also aus einer if-clause und einer then-clause.

Eine Sammlung von Inference Rules ist eine Sammlung von Aussagen oder Annahmen auf die die Regeln angewendet werden (auch das „working memory“ genannt).

Rule chaining

Es gibt zwei Hauptmethoden um Inference Rules zu verbinden, die des Forward-Chaining und die des Backward-Chainings [Sharma et al. 2012].

- *Forward Chaining:* Wird auch datengesteuerte Inferenz genannt und ist eine Vorgehensweise, bei der man von einer Anfangssituation auf die Endsituation schliesst.
- *Backward Chaining:* Wird auch zielgesteuerte Inferenz genannt und ist eine Vorgehensweise, bei der man mit dem Endziel beginnt (d.h. mit dem Sachverhalt, den man aufgrund der Problemstellung erreichen möchte); dieses Ziel wird in Unterziele aufgeteilt, diese werden ebenfalls wieder aufgeteilt etc., bis die Ziele elementare Fakten sind, von denen man weiß, ob sie zutreffen oder nicht.

Listing 9.3 Algorithmus für das forward-chaining

Wiederhole folgende Operationen bis es keine Regel mehr gibt, deren if-clause erfüllt wird.

Finde eine Regel deren if-clause die Aussagen aus dem “working memory” erfüllt.

Füge die Aussagen, die in der then-clause stehen, dem "working memory" hinzu

Anwendungen

- Semantic Web
- Expert Systems

9.4.6 Proportional and Predicate Logic

Die *Logik* beschäftigt sich mit den Normen des korrekten Schlussfolgerns. Sie untersucht, unter welchen Bedingungen das Folgern einer aus einer Menge anderer Aussagen korrekt ist und entwickelt hierzu zur exakten Beschreibung der untersuchten. Sie ist ein Teilgebiet der und teilweise auch der, hat Querbezüge u.a. zur und ist letztlich Grundlage für alle Wissenschaften. Wenn logisches Schließen zu Widersprüchen führt, so kann ein entstehen.

Aussagenlogik (proportional logic) – Logik der Sätze

Geschichte:

- Aristoteles (Griechenland 384 – 322 v.Chr):
- Aussagen sind entweder wahr oder falsch (law of non-contradiction).
- George Boole (England 1815 – 1864):
- Entwicklung der Algebra (Mathematical Analysis of Logic [Boole 2011])

Theorie:

- Die kleinste Logik Einheit ist ein Satz (und zwar nur Aussagesätze,
- z.B. "Gras ist grün", " $1 + 1 = 3$ " und nicht "Wie ist los?"
- Durch Verbindungselemente können Sätze verbunden werden:

IMPLIES Beispiel: "Wenn ich im Lotto gewinne, gebe ich dir 1000 Franken"

Prädikatenlogik (predicate logic) – Logik von Objekten

Geschichte

- Gottlob Frege (1848-1925): Entwicklung der modernen Prädikatenlogik [von Kutschera 1989].

Theorie:

- Erweiterung der Aussagenlogik um "Prädikate", Variablen und Quantifizierungen. Prädikate sind Satz-Templates (Funktionen), welche eine Eigenschaft von Objekten oder deren Beziehung ausdrückt, z.B. $give(x,y,z)$, $loves(x,y)$
- Variablen sind Stellvertreter für eine Objektgruppe.
- Qualifizierungen definieren die Variablen: Alle \forall , einige \exists und keine $\neg\forall$.

Tabelle 9.15 Beispiele für Prädikatenlogik

Aussage	Prädikat
Everybody loves somebody	$\forall x, \exists y : \text{loves}(x, y)$
Nobody loves everybody	$\neg(\exists x, \forall y : \text{loves}(x, y))$

9.4.7 Fuzzy Logic

In der Mitte der 60er Jahre erkannte Professor Lotfi Zadeh von der Berkeley Universität in Kalifornien, dass mit der 'true' oder 'false' Logik der Booleschen Algebra die Grauwerte, die bei vielen Anwendungen auftreten, nur ungenügend beschrieben werden können [Zadeh 1996].

Das Wort Fuzzy bedeutet 'unscharf, verwischt'. Ist Fuzzy-Logik dann also eine unscharf schliessende Logik? - Ja, aber das nicht alleine. Fuzzy-Logik ist zunächst einmal vor allem das 'Handling' unscharfer Werte(-mengen) mit einer hierzu geeigneten Verknüpfungslogik.

Diese Logik ist zwar streng mathematisch genommen als Verknüpfung unscharfer Wertemengen mit Hilfe unscharfer Relationen unter Anwendung unterschiedlichster logischer Verknüpfungsoperatoren aufzufassen, die verwendeten Operatoren sind aber letztlich aus der klassischen Mengentheoretischen Mathematik bekannt.

Einfache Beispiele für derartige Operatoren sind: Minimum-, Maximum-, ODER-, und justierbare UND- und ODER-Operatoren (Um die Vielfalt aufzuzeigen, einige Namen von Unterklassen der genannten Operatoren: Algebraic-Product-, Bounded-Differenz-, Einstein-Product-, Hamacher-Product-, Yager-Intersection- und Fuzzy-AND-Operator. Lassen Sie sich hierdurch nicht abschrecken, die 'Alltagsarbeit' beschränkt sich meistens auf einige wenige!)

Anwendungen

Die Fuzzy Logik findet vor allem in der Regelungstechnik viele Anwendungen. Nun es gibt auch weitere Anwendungen:

- Regelung von Ottomotoren
- Automatikgetriebe
- Fotokameras
- ABS

9.5 Neuronale Netze

Die Realisierung von künstlichen neuronalen Netzen ist ein Versuch, die Struktur von Hirnzellen nachzubilden. Das Ziel dieses Bereiches der Informatik ist, Selbstlernende und intelligente Systeme zu bauen. In diesem Kapitel werden die wichtigsten Ansätze vorgestellt:

- Petri-Netzwerke (Beitrag von Oliver Locher)
- Neuronale Netze (Beitrag von Lorenzo Pepe)
- Self Organizing Maps (Beitrag von Thomas Kuhn)
- Backpropagation Networks (Beitrag von Peter Zimmermann)
- Hopfield Networks ((Beitrag von Marco Reichmuth)

9.5.1 Petri-Netze

Petri-Netze wurden 1962 von Carl Adam Petri zur Beschreibung nebenläufiger kommunizierender Prozesse erfunden. Dynamische Systeme mit fester (Kommunikations-) Struktur lassen sich damit beschreiben [Petri 1962].

Petri-Netze sind ein häufig benutztes Modellierungswerkzeug zur Darstellung von Systemen und den Abläufen innerhalb dieser Systeme. Allen Petri-Netzen gemeinsam ist ihr grundsätzlicher Aufbau.

Petri-Netze bestehen aus:

- *Plätzen (Stellen)*: Repräsentieren Zustandsparameter (Speicherzustände); Puffer für Daten (eventuell mit Kapazitätsangabe); passive Elemente; durch Kreise dargestellt.
- *Transitionen (Ereignisse)*: Repräsentieren Verarbeitungsschritte; bilden dynamisches Systemverhalten ab; beim Schalten (Statfinden) des Ereignisses werden Systemzustände lokal verändert (Zeitpunkt dafür nicht festgelegt); aktive Elemente; durch Kästen oder Striche dargestellt.
- *Marken (Token) auf Plätzen*: Repräsentieren Systemzustände; das Vorliegen von Daten bzw. Erfülltsein von Bedingungen.
- *Kanten (Bögen)* verbinden Plätze mit Transitionen oder umgekehrt; beschreiben Abhängigkeiten zwischen Speicherzuständen (Plätzen) und Ereignissen (Transitionen), eventuell durch Gewichte bewertet.
- *Markierung*: Belegung der Plätze mit Marken; Festlegung des Gesamtzustands des Systems; werden durch das Schalten von Transitionen verändert (Änderung des Systemzustandes).

Beispiele

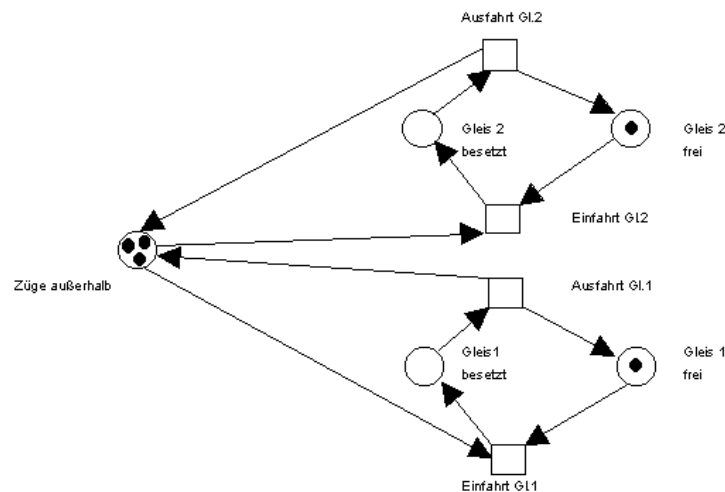


Abbildung 9.11 Petrinetz eines Sackbahnhofs

Das Beispiel zeigt ein einfaches Petri-Netz, welches einen Sackbahnhof mit zwei Gleisen und nur einer Ein-Ausfahrt simuliert. Ankommende Züge (dargestellt in der Stelle "Züge außerhalb" links) erreichen den Bahnhof über die Transition "Einfahrt G.1" bzw. "Einfahrt G.2" und verlassen ihn durch die entsprechende Ausfahrts-Transition. Die Stellen "Gleis 1 frei" und "Gleis 2 frei" werden benötigt, um "Zusammenstöße" zu vermeiden, indem die "Einfahrt" in einen bestimmten Abschnitt nur ermöglicht wird, wenn die entsprechende Stelle markiert ist und die Marke entfernt wird, während ein Zug das jeweilige Gebiet "durchfährt".

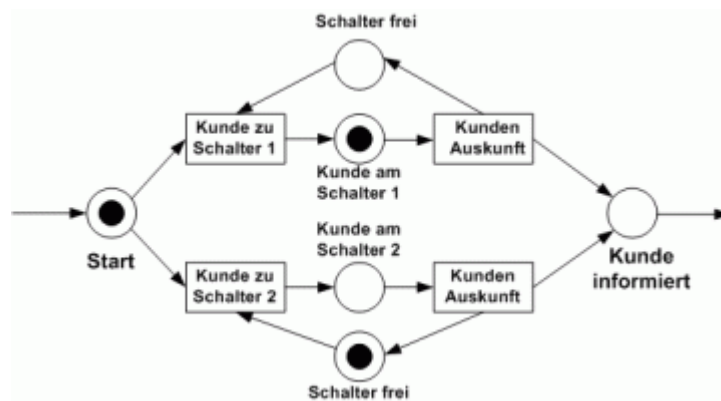


Abbildung 9.12 Petrinetz einer Schalterhalle mit zwei Schaltern

Coloured Petri-Netze

Das Problem bei Petri Netzen ist, sie werden schon bei sehr einfachen Abläufen sehr umfangreich. Um dieses Problem der Netzgröße zu lösen, wurden so genannte Colored-Petri-Netze entwickelt, die unterschiedliche Arten von Markierungen erlauben [Jensen, Kristensen 2009].

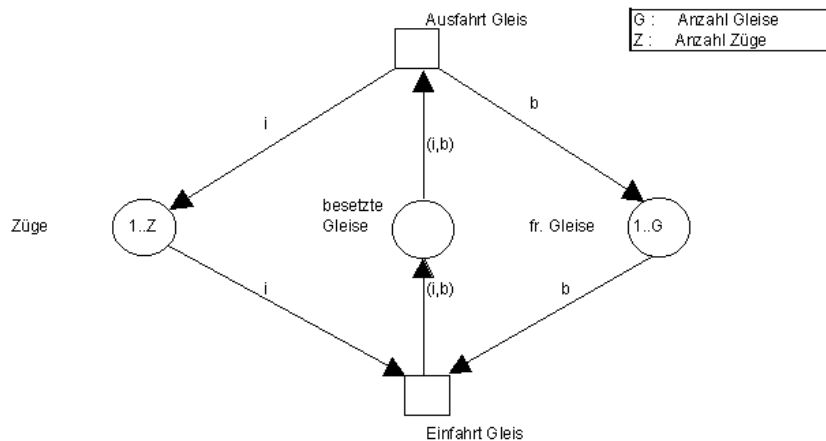


Abbildung 9.13 Coloured Petrinetz eines Sackbahnhofs

Ein System mit den Zügen als Coloured Petri Net. Die Markierungen in CPNs werden durch eine Zahl genauer spezifiziert. So befinden sich z.B. in der Stelle "freie Gleise" G Markierungen, die von 1 bis G durchnummeriert sind. G bezeichnet hierbei die Anzahl der Gleise, die in diesem Netz bei Simulationsbeginn bestimmt werden kann. Ähnlich verhält es sich mit der Stelle "Züge". Die Stelle "besetzte Gleise" enthält sogar eine "zweidimensionale" Markierung, die die Markierung für Gleis mit der für Zug kombiniert, um anzuzeigen, welcher Zug sich auf welchem Gleis befindet. Die Variablen an den Kanten bezeichnen die Markierungen, die bei Schalten der jeweiligen Transition entfernt/hinzugefügt werden, z.B. würde bei Schalten von "Einfahrt Gleis" dieselbe Markierung, die aus "freie Gleise" entnommen wurde, in "besetzte Gleise" abgelegt, zusammen mit der Markierung, die der Stelle "Züge" entnommen wurde.

9.5.2 Neuronale Netze

Künstliche neuronale Netze bestehen aus einer großen Anzahl einfacher Verarbeitungseinheiten, den Neuronen. Information wird verarbeitet, indem sich die Neuronen mit Hilfe von gerichteten Verbindungen untereinander Signale zusenden [Haykin 2008]. Jedes Neuron innerhalb des Netzes kennt nur die Signale, die er in regelmäßigen Abständen empfängt, und das Signal, das er regelmäßig an andere Neuronen aussendet. Und doch sind solche einfache Einheiten in der Lage, komplexe Aufgaben durchzuführen, wenn sie in-

nerhalb eines Netzes koordiniert zusammenarbeiten. Dies geschieht im Prinzip analog zu den Vorgängen im Gehirn.

Neuronale Netze zeichnen sich durch ihre Lernfähigkeit aus. Sie können eine Aufgabe anhand von Trainingsbeispielen erlernen, ohne dazu explizit programmiert werden zu müssen. Weitere Vorteile sind die hohe Parallelität bei der Informationsverarbeitung, die hohe Fehlertoleranz und die verteilte Wissensrepräsentation, wodurch ein zerstörtes Neuron nur einen relativ kleinen Wissensausfall bedeutet. Gewisse Netzmodelle sind auch in der Lage neue Strukturen und Neuronen zu entwickeln.

Basiskomponenten

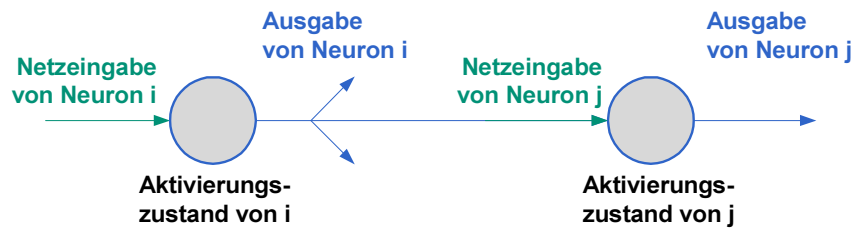


Abbildung 9.14 Basiskomponenten eines Neuronen Netzes

- *Neuronen* oder Knoten sind einfache Prozessoren
- Die *Netzeingabe* eines Neurons ist ein Eingangssignal oder Stimulus.
- Die *Ausgabe* wird durch Kombination der Eingaben eines Neurons mit der *Aktivierungsfunktion* erzeugt.
- Die *Verbindungen* zwischen Neuronen sind oft gewichtet, um das übertragende Signal zu verstärken oder zu vermindern

Typische Aktivierungsfunktionen

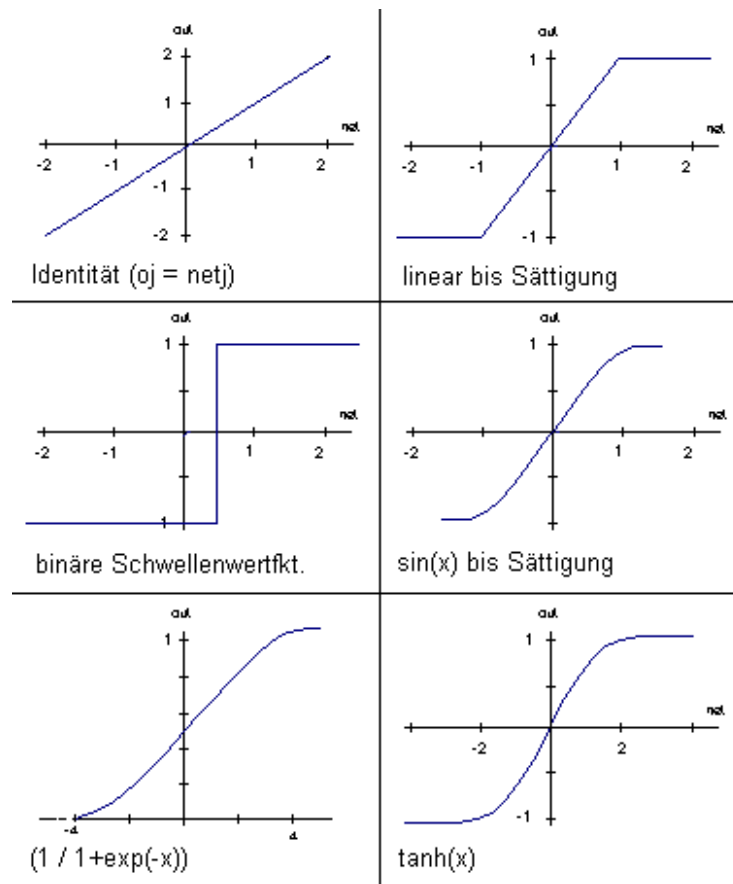


Abbildung 9.15 Typische Aktivierungsfunktionen

- *Identitätsfunktion:* Das Eingangs- und das Ausgangssignal sind gleich.
- *Binäre Schwellenfunktion:* Die Aktivierung ist auf 1 oder 0 beschränkt.
- *Sigmoide Funktion:* Sigmoide Funktionen werden als kontinuierliche Funktionen zwischen 0 und 1 abgebildet.

Modelle Neuronaler Netze

Es wird zwischen Netzen ohne Rückkoppelung (Feedforward Neuronal Networks) und Netzen mit Rückkoppelung (Recurrent Neuronal Networks) unterschieden.

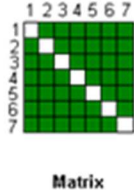
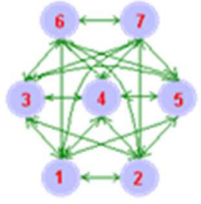
Bei Netzen ohne Rückkopplungen werden Daten nur in eine Richtung weitergegeben, d.h. es gibt kein Weg der wieder zum Ursprung zurückführt. Man unterscheidet außerdem zwischen ebenenweise verbundene und allgemeine feedforward-Netze.

Tabelle 9.16 Feedforward Nets - Netze ohne Rückkoppelung

Name	Erklärung	Beispiel
Ebenenweise verbundene Netze	Sie sind in mehrere Schichten eingeteilt, wobei es nur Verbindungen von einer Schicht zur nächsten gibt [Svozil et Al. 1997].	<p>Matrix</p>
Allgemeine Feedforward-Netze	Sie besitzen zusätzlich auch sogenannte shortcut connections, Verbindungen zwischen Neuronen aus verschiedenen Ebenen [Parker 2007].	<p>Matrix</p>

Tabelle 9.17 Recurrents Nets - Netze mit Rückkoppelung

Name	Erklärung	Beispiel
Netze mit direkten Rückkopplungen (Direct Feedback)	Die Neuronen haben eine Verbindung von ihrer Ausgabe zurück zur Eingabe und können dadurch ihre eigene Aktivierung verstärken oder abschwächen. Diese Verbindungen bewirken oft, dass Neuronen die Grenzzustände ihrer Aktivierungen annehmen, weil sie sich selbst verstärken oder hemmen.	<p>Matrix</p>
Netze mit indirekten Rückkopplungen (Indirect Feedback)	Diese Netze besitzen Rückkopplungen von Neuronen höherer Ebenen zu Neuronen niedrigerer Ebenen. Dadurch erreicht man eine Aufmerksamkeitssteuerung auf bestimmte Bereiche von Eingabeneuronen oder auf bestimmte Eingabemerkmale durch das Netz.	<p>Matrix</p>
Netze mit Rückkopplungen innerhalb einer Schicht (Lateral Feedback)	Netze mit Rückkopplungen innerhalb derselben Schicht werden oft für Aufgaben eingesetzt, bei denen nur ein Neuron einer Gruppe aktiv werden soll. Jedes Neuron hat dann hemmende Verbindungen zu den anderen Neuronen und oft noch eine aktivierende direkte Rückkopplung zu sich	<p>Matrix</p>

	selbst. Das Neuron mit der stärksten Aktivierung, der Gewinner, hemmt dann die anderen Neuronen. Daher heißt eine solche Topologie auch <i>winner-takes-all-Netzwerk</i> .	
Vollständig verbundene Netze	Vollständig verbundene Netze haben Verbindungen zwischen allen Neuronen. Sie sind insbesondere als Hopfield- Netze bekannt geworden. Bei diesen muss allerdings auch die Verbindungsmatrix symmetrisch sein und die Diagonale darf nur Nullen enthalten.	 <p>Matrix</p> 

9.5.3 Self Organizing Maps

Der Begriff Self Organizing Maps (SOM) steht für einen Algorithmus, welcher mehrdimensionale Daten in einer Datenstruktur mit geringerer Tiefe abbildet. Ausserdem sorgt dieser Algorithmus dafür, dass in der Projektion ähnliche Daten zusammenfinden. Der Erfinder von SOM ist Teuvo Kohonen [Kohonen 1990].

Funktionsweise

- *Gegeben:* Mehrdimensionale Daten (z.B. 60 3-dimensionale Vektoren). Dieses Set von Daten wird Input genannt.
- *Gesucht:* Verteilung dieser Daten auf eine Struktur mit weniger Dimensionen (auf z.B. ein 10x10 Array, also ein 2-dimensionales Array). Diese Struktur nennen wir den Output.
 1. *Die Inputdaten auf den Output verteilen:* SOM ist ein iterativer Algorithmus. Zu Beginn wird ein zufälliger Output produziert, indem irgendwelche Daten in der Output-Struktur gespeichert werden.
 2. *Nehme ein Sample:* Es wird ein beliebiges Input-Element als Sample angenommen
 3. *Bestimme den „Best Match“:* Untersuche alle Datenvektoren auf deren Nähe zum Sample. Die üblichste Formel zur Bestimmung der Nähe zwischen zwei Vektoren ist die Euklidische Entfernung (3-dimensional)
 4. *Skaliere den „Best Match“ und dessen nächste Nachbarn:* Nehme die Nachbarn des „Best Match“. Das Verfahren, wie diese bestimmt werden, ist nicht festgelegt und kann frei gewählt werden. Einzige Anforderung an das Selektionsverfahren ist, dass im Laufe der Iterationen weniger Nachbarn bestimmt werden. Diese Nachbarn werden dem Sample ähnlicher gemacht, indem deren Datenvektoren in Richtung Sample-Vektor skaliert werden. Dabei soll der „Best Match“ am meisten dem Sample gleich gemacht werden, die Entferntesten Nachbarn am wenigsten

ten. Eine Möglichkeit dazu ist folgende Formel zu verwenden: Zuerst wird der „Best Match“ eingegeben, dann schrittweise die nächsten Nachbarn und zuletzt den Entferntesten. läuft dabei von 1 nach 0.

5. Falls genügend oft Iteriert, brich ab. Falls nicht, kehre zu 2. zurück

Beispiel: Farben in einem zweidimensionalen Array

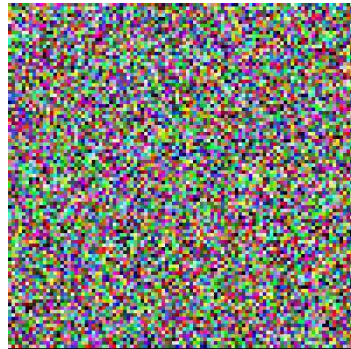


Abbildung 9.16 Array mit zufälligen Farbwerten

Das Standardbeispiel zu SOMs ist das Problem „Wie speichere ich Farben in einem Array“. Die Daten sind Farbpunkte, dreidimensionale Inputvektoren mit den Komponenten [Rot, Grün, Blau], welche in einem zwei-dimensionalen Array abgelegt werden sollen.

Belegt man das Array mit zufälligen Farbwerten zufällig ab, ergibt das, wenn man die Farbpunkte des Arrays ausgibt, eine ungeordnete Menge.



Abbildung 9.17 Array mit gruppierten Farbwerten

Lässt man den SOM-Algorithmus einige Male über dieses ungeordnete Array laufen, entstehen grössere Farbflächen, da ähnliche Farben gruppiert werden.

Anwendungen

- SOMs werden in der Erkennung von Sprache verwendet. Anstelle von Bildpunkten sind die Inputdaten Bruchstücke von Sprache. Diese Bruchstücke werden durch ablegen in einer SOM zu ähnlichen Lautgruppen organisiert.
- Bibliothekssysteme können mit Hilfe von SOMs Publikationen gruppieren. Neue Publikationen können schnell klassifiziert werden.
- SOMs können als neuronales Netzwerk betrachtet werden.

9.5.4 Backpropagation

Bei allen neuronalen Netzen stellt sich immer die Frage, wie das Netz lernen soll. Je nach Netztyp wurden unterschiedliche Regeln und Algorithmen entwickelt, welche aber nur für relativ einfache Architekturen geeignet waren. Für mehrschichtige Netze existierte viele Jahre lang keine Regel zur Aktualisierung der Gewichte.

In den 70er-Jahren entwickelte Werbos eine Technik zum Anpassen der Gewichte [Werbos 1974], doch es waren Rumelhart et al., die 1986 mit dem Backpropagation-Algorithmus den neuronalen Netzen zum Durchbruch verhelfen [Rumelhart et al. 1986].

Die Idee hinter dem Algorithmus ist, den festgestellten Fehler zwischen erwarteter und tatsächlicher Ausgabe beim Rückwärtsdurchlauf auf die verborgenen Einheiten zu verteilen, also die Gewichte entsprechend dem gemachten Fehler anzupassen. Zu diesem Zweck kommt die generalisierte Delta-Regel zum Einsatz. Sie erlaubt es, den Fehler proportional zu den aktuellen Gewichtungen zurückzupropagieren.

Konsequenz:

Die Knoten welche einen grossen Anteil am Fehler haben werden am stärksten angepasst.

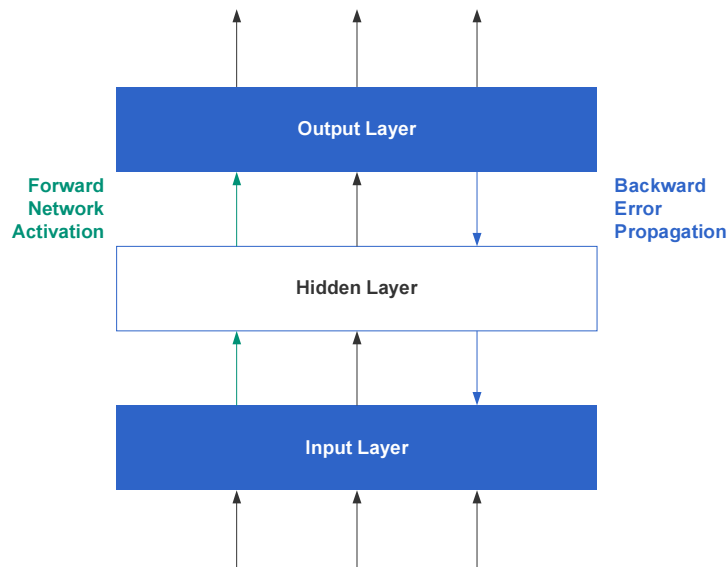


Abbildung 9.18 Prinzip Backpropagation

Der Backpropagation-Algorithmus definiert zwei Netzdurchläufe:

- Einen *Vorwärtsthroughlauf* von der Eingabeschicht zur Ausgabeschicht.
- Einen *Rückwärtsthroughlauf* von der Ausgabeschicht zur Eingabeschicht.

Ablauf:

- Schritt 1: Verbindungsgewichte zufällig initialisieren
- Schritt 2: Eingabemuster und gewünschte Ausgabemuster präsentieren
- Schritt 3: Aktuelle Ausgaben berechnen und Fehler zurückleiten
- Schritt 4: Verbindungsgewichte adaptieren

Schritt 2 bis 4 so lange wiederholen, bis Fehlerwert E kleiner als vorgegebene Schwelle oder eine bestimmte Anzahl von Trainingsdurchläufen erfolgt ist.

Anwendungen

Der Backpropagation-Algorithmus ist heute weit verbreitet und wird in den meisten mehrschichtigen neuronalen Netzen angewendet. Insbesondere im Bereich der Mustererkennung, wo einem Eingabemuster in der Trainingsphase eindeutig ein Ausgabemuster zugeordnet werden kann, ist dieser Algorithmus nicht mehr wegzudenken.

- *NETtalk: Der Klassiker*: Bereits im Jahr 1987 entwickeltes Netz, das lernt englischen Text auszusprechen [Sejnowski, Rosenberg 1987]. Zu diesem Zweck wird jeweils ein Textfenster von sieben Zeichen eingelesen. Die Trainingsdaten konnten am Ende mit 95% Genauigkeit wiedergegeben werden. Bei den Testdaten fiel dieser Wert auf 78%.

- **Handschrift:** Ein im Jahr 1989 entwickeltes neuronales Netz zur Erkennung von Postleitzahlen auf handgeschriebenen Briefumschlägen [Le Cun et Al. 1989]. Segmentierung der Zeichen erledigte ein Präprozessor, das Netz musste nur einzelne Ziffern erkennen können. Bei sehr schwierigen Fällen konnte das Netz bei mehreren Ausgabeknoten einen hohen Wert ausgeben und somit eine Wahrscheinlichkeit für bestimmte Ausgaben ausdrücken. In den restlichen Fällen erreichte das Netz eine Genauigkeit von 99%.
- **ALVINN:** Das „Autonomous Land Vehicle In a Neuronal Network“ ist ein 1993 entwickeltes Netz, um ein Auto selbständig fahren zu lassen [Pomerleau 1992]. Es lernte durch Beobachtung eines menschlichen Fahrers. Als Sensoren wurden Farb-Stereo-Videokamera, Laser und Radar verwendet. Nach Aufzeichnung von 5 Minuten Fahrt und einem 10-minütigen Training konnte ALVINN bereits allein fahren. ALVINN fuhr mit 70 mph über eine Entfernung von 90 Meilen auf öffentlichen Landstrassen. Allerdings kam ALVINN nicht besonders gut mit wechselnden Lichtverhältnissen und anderen Fahrzeugen klar.

9.5.5 Hopfield-Netze

Ein Hopfield Netzwerk ist ein sich selbst organisierendes, neuronales Netz, welches ohne kontrollierten Lernvorgang lernt. Hopfield Netze werden insbesondere bei der Mustererkennung (Sprache und Gesichter) eingesetzt. Bei diesen Netzen geht es darum, dass Gewichtungen gefunden werden, die dafür sorgen, dass topologische Strukturen erhalten bleiben. Die Lernregel legt hier nicht mehr die Multiplikation der Gewichtungen zugrunde, sondern die Subtraktion.

Das asynchrone neuronale Netzwerk wurde 1982 vom amerikanischen Physiker John Hopfield entwickelt. Spezielle Eigenschaften sind, dass sie keine direkten Rückkoppelungen haben und alle Verbindungen symmetrisch sind [Hopfield 1982].

Definition: Hopfield-Netz

Das Netz besteht aus einer Schicht aus n Neuronen

Die Neuronen sind untereinander total vernetzt

Kein Neuron ist direkt mit sich selbst vernetzt (direkte Rückkoppelung, $\Delta w_{ii} = 0$)

Das Netz ist symmetrisch gewichtet

Den einzelnen Neuronen ist jeweils eine lineare Schwellenwertfunktion $f(x) = 1$ falls $x \geq 0$, sonst -1 als Aktivierungsfunktion zugeordnet

Eingabe ist die übliche gewichtete Summe Die Knoten welche einen grossen Anteil am Fehler haben werden am stärksten angepasst.

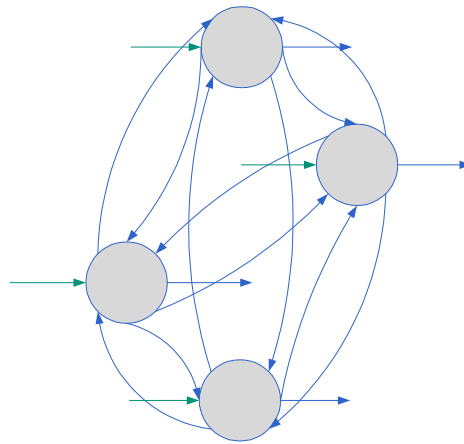


Abbildung 9.19 Hopfield-Netz mit 4 Neuronen

Die Gewichtung Θ oder auch das Lernen der Neuronen erfolgt meist mit Hilfe der Hebb-sche- oder Delta-Widrow-Hoff-Lernregel.

Definition: Hebbsche Regel:

Die Hebbsche Lernregel besagt, dass wenn Zelle j eine Eingabe von Zelle i erhält und dabei gleich-ermassen aktiv sind, wird das dazwischenliegende Gewicht erhöht. Die Lernrate η liegt zwischen 0,01 und 0,5, ein guter Wert ist $\eta = 0,1$.

Gewichtung: $\Delta w_{ij} = \eta \cdot x_i \cdot x_j$

Anwendungen

John Hopfield hatte das Netz für die Modellierung von Materialien die sich wie magnetische Dipole verhalten (Spingläser) verwendet. Sie werden auch noch verwendet für:

- Boolesche Funktionen
- Fehlerkorrektur bei Datenübertragung
- Einfache Schriftenerkennung (OCR)
- Einfache Reiseroutenberechnung
- Gesicht- und Spracherkennung