Vorlesung Maschinelles Lernen

R. Der
Universität Leipzig
Institut für Informatik

16. November 1998

Inhaltsverzeichnis

Kapitel 1

Einführung.

Lernen ist ein zentraler Faktor der Intelligenz. Ein Blick in die Natur lehrt uns, daß gerade die Lernfähigkeit eine grundlegende Komponente der Überlebensleistungen der Lebewesen darstellt. In der Evolution wird Lernen lange vor dem Entstehen von Intelligenz beobachtet, die Fähigkeit zu Lernen ist so gesehen nicht nur eine konstitutive Komponente sondern auch eine unabdingbare Voraussetzung für Intelligenz.

Lebewesen sind lernfähige Strukturen weil sie zur Verbesserung ihrer Überlebensfähigkeiten ihr Verhalten selbst modifizieren können. Betrachten wir Lebewesen als programmgesteuerte Agenten, dann bedeutet metaphorisch gesprochen Lernfähigkeit die selbstorganisierte Adaption dieses Programms. Diese Leistung ist für Lebewesen unverzichtbar, da sie sich in einer komplexen, sich ständig ändernden Umwelt behaupten müssen, die sie immer wieder vor neue Aufgaben stellt.

Die Realisierung intelligenter Systeme durch Computer, die nicht programmiert sondern angelernt werden, wurde als Fernziel der KI schon sehr zeitig formuliert. Eine moderne Sichtweise der KI betrachtet intelligente Systeme als rationale Agenten, die mit der Welt interagieren. Ein solcher Agent, sei es ein autonomer Roboter oder ein Softwareagent im Internet, sieht sich in ähnlicher Weise wie ein Lebewesen einer komplexen dynamischen Umgebung ausgesetzt. Das Ideal des perfekten Programms für einen solchen Agenten, das also eine algorithmische Struktur realisiert, die alle möglichen Situationen antizipiert und entsprechende Lösungsalgorithmen bereithält, wird mit zunehmender Komplexität des Agenten und seiner Umgebung immer unerreichbarer. Stattdessen muß der Agent lernfähig sein, um mit qualitativ neuen Situationen zurechtzukommen und seine Funktionsfähigkeit auch unter außergewöhnlichen Umständen aufrechterhalten zu können.

Die Vorlesung will einen breiten Überblick über das Gebiet des Lernens geben. Wir schließen uns dem Paradigma des (rationalen) Agenten an, wenn dessen Lernfähigkeit im Vordergrund steht werden wir aber, wie in der Psychologie üblich, diesen als den Lerner bezeichnen. Ziel ist, mehr oder weniger vollständig das Repertoire bereitzustellen, mit dem ein Agent zum Lernen befähigt werden kann. Die Betonung liegt dabei auf konkreten Algorithmen wobei die Praxistauglichkeit ein wichtiger Aspekt der Auswahl war.

1.1 Definition des Lernens. Lernziele.

Lernen wird in den unterschiedlichen Wissensgebieten durchaus verschieden definiert. Wir wollen uns der allgemeinen Definition des Lernens von H. Simon anschließen:

- Lernen ist jeder Prozeß bei dem ein System selbstorganisiert seine Leistung verbessert. Das kann durch Erwerb neuer bzw. Aktualisierung alter Kenntnisse oder Methoden erzielt werden.

Aus mehr psychologischer Sicht können wir

- Lernen als einen Prozess definieren, der zu relativ stabilen Veränderungen im Verhalten oder Verhaltenspotential führt und auf Erfahrungen aufbaut. Lernen ist nicht direkt zu beobachten. Es muß aus den Veränderungen des beobachteten Verhaltens erschlossen werden.

(Zitat aus [57]).

Mehr eingeschränkte Definitionen orientieren sich an den Lernzielen:

- Lernen ist Erwerb von Wissen (hauptsächlich im Zusammenhang mit Expertensystemen)

- Lernen ist Erwerb von Geschicklichkeit. Psychologische Erkenntnisse (Norman 80) besagen: Fertigkeiten wachsen durch Übung auch nach verbaler Instruktion noch weiter. Dieser Lernvorgang ist bisher wenig verstanden.

- Lernen als Verstehen: Bildung von Theorien und induktive Inferenz (Ableitung allgemeiner Gesetze oder Regeln aus Beispielen)

Die hier ohne Anspruch auf Vollständigkeit formulierten Lernziele – Leistungsverbesserung, Erwerb von Wissen oder Geschicklichkeit, Verstehen – können durch Interaktion mit der (gegebenenfalls um eine Lernumgebung = Lehrer erweiterten) Welt erreicht werden.

1.2 Formen des Lernens.

Die Art der Interaktion mit der Welt und die Spezifik der Lernumgebung definiert die Form des Lernens. Wir unterscheiden prinzipiell die folgenden Fälle:

- Überwachtes Lernen oder **Lernen mit Lehrer:** Hier besteht die Funktion des Agenten meist darin, richtige Antworten zu geben, d. h. das adaptive System soll lernen, ganz bestimmte Input-Output-Relationen zu reproduzieren und zu verallgemeinern, die ihm von einem Lehrer in Form von Beispielen vorgelegt wurden, vgl. s. Abbildungen 1.1 und 1.2.

- Nichtüberwachtes Lernen: Hier lernt der Agent völlig selbständig gewisse Zusammenhänge zwischen den Input-Daten zu erkennen und entsprechende Output-Informationen zu erzeugen, vgl. Abb. 1.3.

- Wenn die Reaktion des Agenten eine Aktion ist (z. B. Regelimpuls, Ausführen einer Handlung), der die Welt beeinflußt, dann kann die zu erlernende Funktion darin bestehen, die in der aktuellen Situation beste Aktion auszugeben. Wenn es keinen Lehrer gibt, der diese Aktionen kennt, dann erfolgt das **Lernen durch Tun**, d. h. es werden immer neue Aktionsstrategien ausprobiert, bis die beste gefunden ist. In letzterem Falle kann das adaptive System noch weiter in einen informationsverarbeitenden Anteil und einen Aktionsteil untergliedert werden.

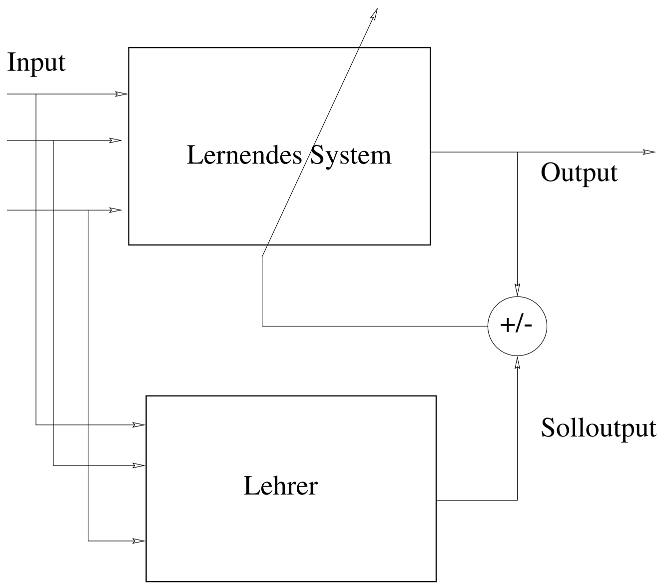

Abbildung 1.1: Beim überwachten Lernen wird dem Lernenden System (Lerner) von einem Lehrer zu jedem Input der gewünschte Solloutput vorgegeben. Im vorliegenden Fall kommen die Inputs von einer beliebigen externen Quelle, die Lehrer und Lerner gemeinsam sehen.

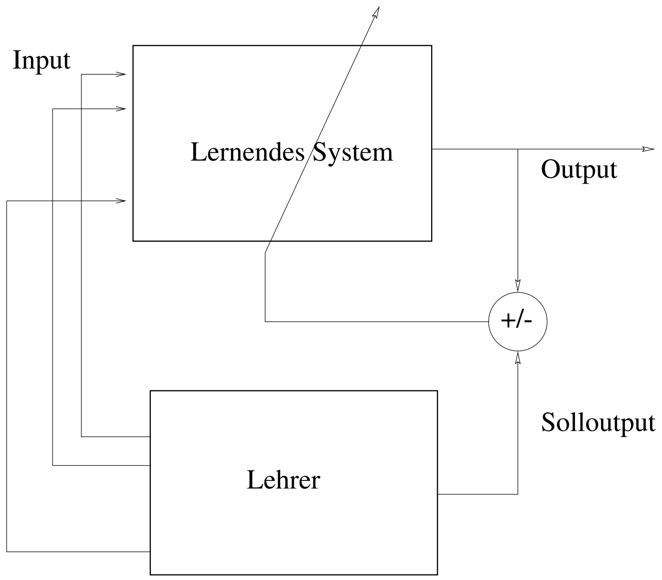

Abbildung 1.2: Wie Abbildung 1.1 aber für den Fall, daß der Lehrer die Inputs für das lernende System selbst erzeugt, wie es z. B. der Fall ist, wenn der Lehrer Beispielinstanzen für einen nur ihm bekannten Funktionszusammenhang generiert.

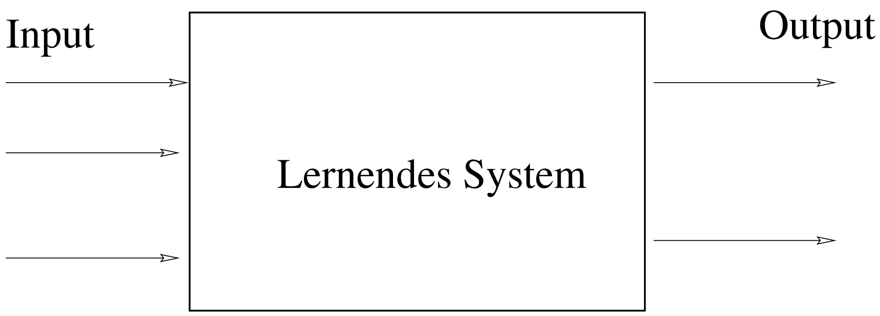

Abbildung 1.3: Nichtüberwachtes Lernen (unsupervised learning). Der Lerner erkennt selbständig – meist aus den statistischen Eigenschaften der Datenverteilung – Strukturen oder innere Zusammenhänge der Daten. Typisches Beispiel ist die Clusterung von Daten.

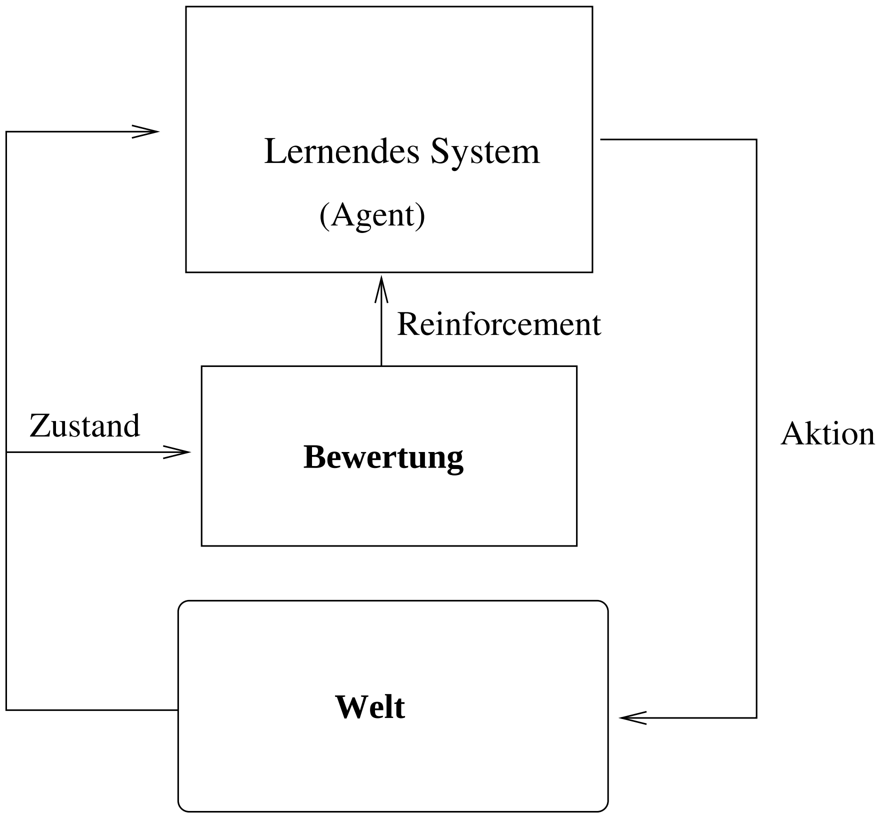

Abbildung 1.4: Reinforcement Lernen. Der Lerner übt in Abhängigkeit vom aktuellen Weltzustand eine Aktion aus. Dadurch verändert sich die Welt. Ist ein Zielzustand erreicht dann gibt die Bewertungssinstanz eine definierte Menge von Reinforcment an den Agenten aus.

1.3 Arten des Lernens.

Der Lernmodus (Form des Lernens) ergibt sich aus der Art der Wechselwirkung des Agenten mit der Welt. Diese Wechselwirkung - genauer der Konflikt zwischen den aktuellen und den optimalen Leistungen des Systems - ist die Triebkraft für die adaptive Veränderung des lernenden Systems. Die gewünschte Veränderung der Eigenschaften des Agenten gelingt durch die gezielte, vom Agenten selbst organisierte Beeinflussung seiner adaptiven Komponenten. Je nach Art der variablen Elemente unterscheidet man beispielsweise die folgenden Fälle:

- **Parameterlernen**: Die in technischen Systemen häufigste Form. Beispiel: Ein Regler, dessen Regelfunktion von einer Reihe von Parametern abhängt, die zielorientiert durch einen geeigneten Lernalgorithmus verändert werden.
- **Strukturlernen**: Hierbei wird die innere Struktur des lernenden Systems verändert.
- **Adaptive Programmierung**: Für programmgesteuerte Systeme ist das Programm die variierbare Komponente des lernenden Systems.

Der Lernmechanismus bestimmt die Art der Einflußnahme auf die Eigenschaften des adaptiven Systems zur Verbesserung seiner Funktion. Diese hängt natürlich in entscheidender Weise von der Struktur des adaptiven Systems ab. Ein entscheidendes Problem für die praktische Realisierung des Lernens ist die Frage, wie die optimale Einstellung der variablen Elemente des lernenden Systems gefunden werden soll.

1.4 Verfahren des Lernens.

Prinzipiell ist Lernen ein Optimierungsproblem, d. h. es gilt, über die variablen Elemente die Eigenschaften des lernenden Systems so lange zu ändern, bis die optimale Performance erreicht ist. Allerdings muß **Lernen** genauer **als Optimierungsproblem bei unvollständiger Apriori-Information** betrachtet werden, da meist nur eine begrenzte Anzahl von (oft verrauschten) Trainingsinstanzen vorliegt, die der Lerner dann generalisieren muß. Außerdem erfolgt Lernen meist im On-line-Betrieb, so daß die Kostenfunktion nicht statisch sondern datengetrieben variabel ist. Deswegen und wegen der Gefahr der kombinatorischen Explosion beruht ein praktikabler Lernalgorithmus auf speziellen Optimierungsstrategien und insbesondere auf einer zielorientierten Variation der Eigenschaften des Lerners. Die Konstruktion und Analyse effektiver Verfahren des Lernens (Lernalgorithmen) stellt die eigentliche Herausforderung des Maschinellen Lernens dar. Beispiele:

- **Gradientenverfahren** sind vor allem im Fall des Lernens mit Lehrer anwendbar, da die Abweichung der Antwort des Systems von der Sollantwort eine Art Gütefunktion für ein Optimierungsproblem darstellt. Verrauschte bzw. unvollständige Trainingsinstanzen führen auf stochastische Gradientenverfahren.

- **Statistische Lernverfahren** kommen zur Anwendung, wenn die zu erlernende Funktion probabilistisch formuliert ist. Beispiel: Adaptive Konstruktion eines optimalen Bayesklassifikators.

- **Heuristische Suchverfahren** sind Methoden, um insbesondere bei Baumsuchverfahren nur einen kleinen Teil des gesamten Suchraumes absuchen zu müssen.

- **Genetische Algorithmen** sind Suchverfahren, die als kollektive Problemlösestrategien oder kollektive Optimierungsverfahren betrachtet werden können. In jüngster Zeit sind sie für lernende Systeme von besonderem Interesse, da mit der Methode der genetischen Programmierung Computerprogramme als lernende Systeme betrachtet und entsprechend adaptiv variiert werden. Diese Verfahren realisieren weitgehend die Idee des sich selbst programmierenden Computers.

1.5 Metrische vs. nichtmetrische Daten.

Die konkrete Ausführung eines jeden Lernverfahrens orientiert sich im allgemeinen stark an der Natur der Daten für die es konzipiert wurde. Eine Einteilung (von vielen möglichen) der Lernverfahren orientiert sich an einer Unterscheidung in metrische und nichtmetrische Daten.

Generell erhält der Lerner Informationen über die Objekte der Welt in Form des Merkmalsvektors x. Die Komponenten x_i des Merkmalsvektors $x = (x_1, ..., x_n)$ sind die Attribute des Objektes. Beispiel: Steckbrief einer Person

$$x = (< \text{Größe} >, < \text{Augenfarbe} >, \ldots)$$

Die Attributwerte können je nach Domäne rellwertig, ordinal, nominal oder beliebige symbolische Ausdrücke sein.

Die mathematische Natur dieser Vektoren hängt stark vom Datentyp ab. Beschreibt x beispielweise den Zustand eines physikalischen oder technischen Sytems, dann ist es sinnvoll, x als einen "echten" Vektor im R^n zu betrachten, d.h. der Abstand zwischen zwei Objekten mit Datenvektoren x und x' ist durch die Euklidische Metrik gegeben

$$d\,(x, x') = \sqrt{\sum_{1} (x_1 - x'_1)^2} \tag{1.1}$$

Damit gilt für drei beliebige Vektoren x, y, z die Dreiecksrelation

$$d(x, y) + d(y, z) \geq d(x, z) \tag{1.2}$$

Diese fundamentale Relation ist nicht nur für die Euklidische Metrik erfüllt sondern gilt allgemein für eine Vielzahl anderer Abstandsdefinitionen. Wir wollen Daten, für die die Dreiecksrelation gilt, metrische Daten nennen.

Allerdings gilt die Dreiecksrelation gerade für viele in der Informatik wichtigen Datentypen nicht. Das ist insbesondere der Fall, wenn die Komponenten des Merkmalsvektors nominale oder symbolische Ausdrücke sind. Dann gibt es keine algebraische Definition des Abstandes wie etwa Gl. (1.1). Stattdessen sind die Abstände paarweise beliebig vorgegeben, so daß das Erfülltsein der Dreiecksrelation eher die Ausnahme darstellt. Daten dieses Typs heißen nichtmetrische Daten.

In den ersten drei Kapiteln der Vorlesung werden zunächst Lernverfahren behandelt, die für metrische Daten besonders gut geeignet sind. Diese leiten sich von geometrischen Vorstellungen über die Verteilung der Datenvektoren im Raum ab und setzen damit stillschweigend die Gültigkeit der Dreiecksrelation voraus.

Danach folgt in Kap. 5 und in folgenden Kapiteln die Darstellung der Verfahren für nichtmetrische Daten.

Kapitel 2

Parameteradaptive Lernverfahren

2.1 Anliegen

Wir beschäftigen uns in diesem Kapitel mit lernenden Systemen, deren Eigenschaften durch einen Satz von Parametern, zusammengefaßt in einem Parametervektor $w \in \mathcal{R}^p$, kontrolliert werden. Mathematisch kann das lernende System

$$
\begin{array}{cc|c|cc}
\hline
x_1 & \rightarrow & \text{Lern.–System} & \rightarrow & y_1 \\
\vdots & \vdots & \text{mit} & \vdots & \vdots \\
x_n & \rightarrow & \text{Par.—Vektor} & \rightarrow & y_m \\
 & & w & & \\
\hline
\end{array}
$$

durch seine Transferfunktion

$$y = S\,(x; w) \tag{2.1}$$

definiert werden, wobei $x = (x_1, \ldots, x_n)$ der Input in das System, $y = (y_1, \ldots, y_m)$ der assoziierte Output und $w = (w_1, \ldots, w_p)$ der Vektor der Systemparameter ist.

Ein parameteradaptives Lernverfahren leistet die Anpassung dieser Parameter dergestalt, daß das lernende System eine vorgegebene Aufgabe möglichst gut erfüllt.

2.2 Lernen aus Beispielen

2.2.1 Aufgabenstellung

Beim Lernen aus Beispielen geht es um die Inferenz eines dem Lerner unbekannten Funktionszusammenhanges

$$y = f(x), \qquad x \in \mathcal{R}^n, \quad y \in \mathcal{R}^m$$

anhand von Beispielen für diesen Funktionszusammenhang. Jedes Beispiel ist eine Instanz, also ein Paar (x, y) mit $y = f(x)$, des Funktionszusammenhanges. Im allgemeinen sieht der Lerner nur eine beschränkte Menge solcher Instanzen, d. h. es existiert eine Menge Ω von Inputs x für die der Lerner die zugehörigen Outputs y sieht. Diese Instanzen, also die kombinierten Vektoren z

$$z = \left(x, y^{soll}\right) = \left(x_1, \ldots, x_n, y_1^{soll}, \ldots, y_m^{soll}\right), \quad x \in \Omega \tag{2.2}$$

bilden die Trainings– oder Beispielmenge für den Lerner, wobei $y^{soll} = f(x)$ der vom Lehrer vorgegebene Sollwert zum Input $x \in \Omega$ ist.

Im konkreten Fall der Funktionsanpassung ist zu beachten, daß die Funktionswerte auch verrauscht (durch Meßfehler oder andere Störgrößen verfälscht) sein können, d. h. die Sollwerte sind

$$y^{soll} = f(x) + \xi \tag{2.3}$$

wobei die Störgröße ξ durch eine Zufallszahl modelliert werden kann, die potentiell ebenfalls von x abhängt.

In Gleichung (2.3) ist stillschweigend angenommen, daß die Zufallszahl nicht von x selbst abhängt. Das entspricht dem Fall des sog. additiven Rauschens. Viel komplizierter ist i.a. der Fall des multiplikativen Rauschens

$$y^{soll} = f(x) + g(x, \xi) \tag{2.4}$$

mit einer beliebigen Funktion g. Beispiel: $g(x, \xi) = \xi x$.

Ein wichtiger Anwendungsfall ist die Systemidentifikation bzw. –Modellierung. Dabei soll der Lerner das Input–Output Verhalten eines unbekannten Systems modellieren, wobei die Sollvorgaben y_i^{soll} der Trainingsbeispiele durch den Output des Zielsystems für den gemeinsam anliegenden Input x gegeben sind, das ist die in Abb. 1.1 dargestellte Situation.

2.2.2 Allgemeines zum Lernen aus Beispielen

Das Lernen aus Beispielen zählt auch in psychologischer Hinsicht zu den am besten verstandenen Lernmodi. Die zu erlernende Aufgabe besteht hier darin, anhand einer Reihe von Trainingsbeispielen das Soll-Verhalten (z. B. die Funktion $f(x)$) zu erlernen. Die Trainingsbeispiele bestehen dabei meist aus einer Menge von Input-Output-Relationen, das Ziel-Verhalten in der Reproduktion und Generalisierung dieser Relationen. Generalisierung heißt dabei, die richtige Zuordnung auch für Input-Output-Paare zu produzieren, die nicht Elemente des Trainingssatzes bildeten aber der gleichen Bildungsvorschrift genügen. Die richtige Generalisierung entspricht also der Inferenz dieser Bildungsvorschrift, d. h. der Regel, die den Input-Output-Relationen zugrundeliegt.

2.2.3 Die Kostenfunktion

Ziel des Parameterlernens ist, die Parameter so einzustellen, daß für die Elemente des Trainingssets, d. h. für Inputs $x \in \Omega$ die Output-Werte gemäß 2.1 möglichst wenig von den entsprechenden Sollwerten y^{soll} abweichen. Die Abweichungen werden üblicherweise mittels einer **Kostenfunktion** gemessen, wie z. B. die Summe der quadratischen Abweichungen der Soll- von den Ist-Werten. Für eine einzelne Instanz $z = (x, y^{soll})$ schreiben wir

$$\begin{aligned} E\left(x|w\right) &= \frac{1}{2}\sum_{i=1}^{m}\left(y_i - y_i^{soll}\right)^2 \qquad (2.5) \\ &= \sum_{i=1}^{m}\left(S_i\left(x;w\right) - y_i^{soll}\left(x\right)\right)^2 \end{aligned}$$

Ziel ist natürlich die Mimimierung der über alle Instanzen gemittelten Kosten

$$E\left(w\right) = \frac{1}{2N_\Omega}\sum_{x\in\Omega} E\left(x|w\right) \qquad (2.6)$$

wobei N_Ω die Zahl der Trainingsinstanzen ist. Die Lernaufgabe ist im Idealfall genau dann gelöst, wenn die Kostenfunktion für nichtverrauschte Daten den Wert Null erreicht. Dieser Fall impliziert die folgenden Voraussetzungen:

- Das Erfülltsein der sog. *closed world assumption*, d.h. daß die vorhandenen Daten die Welt vollständig erfassen, so daß kein Generalisierungsproblem auftreten kann.

- Der Lerner hat eine ausreichende Kapazität, d. h. er ist hinreichend flexibel, so daß er die Funktiion auch wirklich realisieren kann.

Die Quadratsumme der Fehler ist nicht immer die beste Kostenfunktion. Will man z. B. erreichen, daß Ausreißer (das sind Instanzen bei denen die Abweichung zwischen Ist– und Sollwert besonders groß ist) nicht so stark in die Kostenfunktion eingehen, dann kann man z. B. die Kostenfunktion

$$E(w) = \frac{1}{2N_\Omega} \sum_{x\in\Omega} \sum_{i=1}^{m} \left| y_i - y_i^{soll} \right|$$

verwenden. Allgemein kann man den Grad der Bewertung der Ausreißer durch eine Kostenfunktion der Form

$$E(w) = \frac{1}{2N_\Omega} \sum_{x\in\Omega} \sum_{i=1}^{m} \left| y_i - y_i^{soll} \right|^p \tag{2.7}$$

über den Parameter p einstellen.

2.2.4 Kostenfunktionen mit Nebenbedingungen

Unabhängig von der Form der gewählten Kostenfunktion muß es nicht immer günstig sein, unter allen Umständen das niedrigste Kostenniveau anzustreben, z.B. wenn den technisch realisierbaren Werten der Parameter Grenzen gesetzt sind oder wenn die Zahl der Parameter so groß ist, daß die Gefahr der Überanpassung (*overfitting*) an verrauschte Meßgrößen besteht.

Zusatzforderungen an die Parameter können leicht durch entsprechende Terme in der Kostenfunktion Berücksichtigung finden. Wenn beispielsweise die Nebenbedingung in der Forderung besteht, daß die Norm des Vektors w nicht zu groß werden soll, dann ist ein geeigneter Ansatz eine
Kostenfunktion mit Nebenbedingung:

$$E(w) = \frac{1}{2N_\Omega} \sum_{x\in\Omega} \sum_{i=1}^{m} \left(y_i - y_i^{soll} \right)^2 + \lambda \sum_{k=1}^{p} w_k^2 \tag{2.8}$$

wobei der "Strafterm" für $\lambda \geq 0$ die Kosten vergrößert, sobald die Norm des Parametervektors wächst. Die Größe von λ bestimmt dabei, wie stark diese Nebenbedingung Berücksichtigung findet.

2.2.5 Gewichtete Kostenfunktionen

Treten Inputs x mit unterschiedlicher relativer Häufigkeit auf oder sollen die Trainingsinstanzen unterschiedlich stark beim Lernen berücksichtigt werden, dann bietet sich die Einführung von Gewichten in der Kostenfunktion für die verschiedenen Trainingsinstanzen an

$$E\left(w\right) = \frac{1}{2N_\Omega} \sum_{x \in \Omega} \sum_{i=1}^{m} Q(x) \left(y_i - y_i^{soll}\right)^2 \tag{2.9}$$

Sind die x mit Verteilungsdichte $p(x)$ kontinuierlich verteilt, dann liefert $Q(x) = 1/p(x)$ eine von der statistischen Verteilung der Daten unabhängige Kostenfunktion. Treten andererseits im diskreten Fall die x mit unterschiedlicher relativer Häufigkeit als Inputs auf, ist $Q(x)$ diese Häufigkeit und soll der Lerner häufiger anfallende Beispiele stärker beachten, dann ist 2.9 gerade die richtige Kostenfunktkion.

2.2.6 Allgemeine Formulierung der Kostenfunktion

Allgemein fallen die Beispiele $z = (x, y^{soll})$ gemäß einer Verteilung $P(z)$ an, die allerdings meist unbekannt ist. Die allgemeinste Kostenfunktion lautet dann

$$E(\mathbf{w}) = \int dP(z)\, E(z, \mathbf{w}) \tag{2.10}$$

wobei $E(z, \mathbf{w})$ der bei gegebenem Papramtervektor w für ein bestimmtes Trainingsbeispiel $z = (x, y^{soll})$ beobachtete Fehler ist. Beispielsweise gilt für den quadratischen Ansatz

$$E(z, \mathbf{w}) = \sum_{i=1}^{m} \left(S_i(x, w) - y_i^{soll}\right)^2 \tag{2.11}$$

Nach Vapnik [51] ergeben sich die zentralen Fragen der Lerntheorie aus der Tatsache, daß die Verteilung $P(z)$ im allgemeinen nicht bekannt ist sondern daß eben nur eine gemäß der Verteilung P zufällig gezogene endliche Menge Ω von N Trainingsbeispielen vorliegt. Anstelle von Gl. (2.10) ist die Kostenfunktion

$$E(\mathbf{w}) = \frac{1}{N} \sum_{z \in \Omega} E(z, \mathbf{w}) \tag{2.12}$$

In Termen dieser sog. empirischen Kostenfunktion können die Fragen nach der Konsistenz, der Konvergenzgeschwindigkeit und der Verallgemeinerungsfähigkeit allgemein untersucht werden, vgl. [51].

2.3 Gradientenverfahren

Die Kostenfunktion $E\left(w\right)$ ist eine Funktion der variablen Parameter w des lernenden Systems. Wir können uns E als ein Gebirge über dem Raum der Parameter vorstellen.

2.3.1 Minimierung der Kostenfunktion durch Gradientenabstieg

Die einfachste Methode zur schrittweisen Minimierung der Kostenfunktion ist ein **Gradientenverfahren**, d. h. wir wählen in jedem Schritt das Inkrement Δw von w proportional zum Gradienten von $E\left(w\right)$

Gradientenverfahren:

$$\Delta w = -\epsilon \frac{\partial}{\partial w} E\left(w\right) \tag{2.13}$$

Die Lernregel bedeutet für die i–te Komponente w_i des Parametervektors w eine Verschiebung um Δw_i, die mit Faktor ϵ proportional zum Anstieg der Kostenfunktion längs der i–ten Achse des Koordinatensystems (das den Parameterraum $\mathcal{R}^p$ aufspannt) ist.

Der Vektor

$$G\left(w\right) = \frac{\partial}{\partial w} E = \left(\frac{\partial}{\partial w_1} E, \ldots, \frac{\partial}{\partial w_p} E \right) \tag{2.14}$$

ist der Gradient von E im p–dimensionalen Raum. Seine Komponenten sind gerade die Ableitungen der Funktion $E(w)$ längs der Koordinatenachsen. Da die Ableitung den Anstieg S von E längs der jeweiligen Achse bemißt, ist der Gradient ein Vektor S in Richtung des steilsten Anstieges dessen Länge zum Anstieg längs dieser Richtung proportional ist.

Die Lernregel 2.13 bedeutet offensichtlich

$$\Delta w = -\epsilon G(w) \tag{2.15}$$

Der Abstieg in der Fehlerlandschaft erfolgt in Richtung des negativen Gradienten mit einer Schrittweite, die mit Faktor ϵ proportional zur lokalen Steilheit des Gebirges in Richtung des Gradienten ist.

Damit leistet dieses Verfahren einen lokal optimalen **Abstieg** in der Landschaft der Kosten, wobei der Lernparameter ϵ die Größe des Lernschrittes skaliert. Mit einem Schrittzähler (Zeit) $t = 0, 1, 2, \ldots$ wird der Parametervektor eine Funktion der diskreten Zeit t und 2.13 kann auch als

$$w\,(t+1) = w\,(t) - \epsilon G\,(w\,(t)) \tag{2.16}$$

geschrieben werden. Für hinreichend kleine ϵ konvergiert der Vektor $w\,(t)$ gegen das nächstgelegene Minimum der Fehlerfunktion.

In C^{++}-Notierung schreiben wir die Lernregel (2.15) bzw. (2.16) als

$$w- = \epsilon G(w) \tag{2.17}$$

2.3.2 Konvergenzbetrachtungen

Zur Diskussion des Konvergenzverhaltens für beliebigen Lernparameter ϵ betrachten wir den trivialen Fall eines skalaren Parameters $w \in \mathbf{R}^1$ und einer einfachen quadratischen Kostenfunktion

$$E = \frac{1}{2} a\,(w-b)^2 \tag{2.18}$$

so daß

$$\Delta w = -\eta\;(w-b)\,, \qquad \eta = a\epsilon \tag{2.19}$$

Mit dem Ansatz

$$w\,(t) - b = e^{-\hat{\eta}\,t}\;[w\,(0) - b] \tag{2.20}$$

und Umschreiben von 2.19 in

$$\frac{w\,(t+1) - b}{w\,(t) - b} = 1 - \eta \tag{2.21}$$

finden wir, daß mit

$$\hat{\eta} = -\ \ln\left(1 - \eta\right), \qquad 0 < \eta < 1 \tag{2.22}$$

der Ansatz 2.20 tatsächlich die Differenzengleichung 2.19 löst.

Für $\eta < 1$ konvergiert $w\left(t\right)$ offensichtlich exponentiell gegen b. Ähnlich liest man aus Gleichung (2.21) ab, daß w für $1 < \eta < 2$ ebenfalls exponentiell aber mit alternierendem Vorzeichen konvergiert während $w\left(t\right)$ für $\eta > 2$ divergiert, denn dann ist $|1 - \eta| > 1$ in (2.21). Das ist bei der Parameterwahl zu berücksichtigen, sonst kann man offensichtlich böse Überraschungen erleben. Der Fall $\eta = 1$ ist getrennt zu behandeln, man sieht unmittelbar, daß in diesem Fall die Konvergenz in einem einzigen Schritt erfolgt. Für $\eta = 2$ schließlich ergibt sich $w(t+1) = -w(t)$, d.h. w ändert nur das Vorzeichen. Für $\eta << 1$ findet man durch Reihenentwicklung des Logarithmus

$$\hat{\eta} \approx \eta = \epsilon a \tag{2.23}$$

Die Konvergenzgeschwindigkeit ist also direkt proportional zur Schrittweite ϵ und zur Krümmung (s.u.) a der Fehlerfunktion.

Die obigen Überlegungen haben allgemeinere Bedeutung. Für eine hinreichend glatte Fehlerfunktion gilt in einer gewissen Umgebung um ein Minimum $w = w_{\min}$ die Näherung (w sei der Einfachkeit halber weiter ein Skalar)

$$E\left(w\right) = E\left(w_{\min}\right) + \frac{1}{2}\left(w - w_{\min}\right)^2 E''\left(w_{\min}\right) \tag{2.24}$$

(Taylorentwicklung unter Berücksichtigung von $E'\left(w_{\min}\right) = 0$). Liegt der Startwert $w\left(0\right)$ von w in diesem Bereich dann gilt in gleicher Näherung

$$w\left(t\right) - w_{\min} = e^{-\hat{\eta}t}\left[w\left(0\right) - w_{\min}\right] \tag{2.25}$$

mit

$$\hat{\eta} = -\ln\left(1 - \epsilon E''\left(w_{\min}\right)\right)$$

wobei $E''\left(w_{\min}\right)$ gerade die Krümmung der Fehlerfunktion am Minimum ist[1].

[1] Die Krümmung K einer Kurve $y = f\left(w\right)$ im Punkt w ist durch $K = \frac{f''}{(1+f'^2)^{3/2}}$ definiert, wobei $f' = \frac{d}{dw}f\left(w\right)$ und $f'' = \frac{d^2}{dw^2}f\left(w\right)$. In einem Extremum gilt wegen $f' = 0$ somit $K = f''\left(w_{extr}\right)$.

Die Bedeutung der Parameter wird besonders klar, wenn wir die Update–Regel mit $\tau = 1/\epsilon$ als

$$\tau \Delta w = -\frac{\partial E}{\partial w} \tag{2.26}$$

schreiben und $\epsilon E'' << 1$ annehmen. Dann wird 2.25 (mit $\ln(1-x) \approx -x$ für $x << 1$)

$$w(t) - w_{\min} = e^{-t/t_c}\left[w(0) - w_{\min}\right] \tag{2.27}$$

wobei

$$t_c = \tau \rho \tag{2.28}$$

Formel 2.27 besagt, daß die Annäherung an das Minimum exponentiell schnell mit der Zeitkonstanten t_c erfolgt, die das Produkt der absoluten Zeitkonstanten τ des Lerners mit dem Krümmungsradius $\rho = 1/E''(w_{\min})$ der Fehlerfunktion am Minimum ist. Eine monotone Konvergenz des Verfahrens ist nur gesichert, wenn $t_c \leq 1$ ist.

2.3.3 Übergang zur Differentialgleichung

Für hinreichend kleine ϵ approximiert der Differentialquotient den Differenzenquotient beliebig genau (unter Vorraussetzung der Stetigkeit des Gradienten), so daß 2.13 bzw. 2.15 in

$$\dot{w} = -\epsilon\, G(w) \tag{2.29}$$

übergeht, wobei der Punkt die Ableitung nach der Zeit bedeutet und $w = w(t)$. (2.29) ist damit ein Satz von p Differentialgleichungen für $w \in \mathbf{R}^p$.

In der Umgebung eines Minimums w_{min} der Fehlerfunktion reproduziert sich (wegen des verschwindend kleinen ϵ) automatisch (2.27), d. h.

$$w(t) - w_{\min} = e^{-t/t_c}\left[w(0) - w_{\min}\right] \tag{2.30}$$

wobei $w(0)$ der Startwert für den Parameter w ist. Für die Kosten gilt

$$E(w(t)) = a\, e^{-\frac{2t}{\tau\rho}} \left(w(0) - w_0\right)^2$$

Das Gradientenverfahren erzielt also eine exponentielle Annäherung an den optimalen Kostenwert $E = 0$.

2.4 Gütekriterien

Das obige allgemeinen Verfahren treibt den Lerner in das nächstgelegene Minimum. Das muss nicht immer das beste Ergebnis sein, sogar wenn das globale Minimum erreicht wurde. Insbesondere droht bei korrumpierten (verrauschten) Daten immer die Gefahr der Überanpassung (Overfitting), d. h. daß es der Lerner nicht oder nur unvollständig schafft, die unter dem Rauschen liegende Gesetzmäßigkeit richtig zu erfassen. Das äußert sich z. B. darin, daß der Lerner neu hinzukommende Daten in der Regel nur sehr schlecht repräsentieren kann.

Aufteilung in Trainings– und Testmenge

Liegen ausreichend viele Daten vor, so empfiehlt sich die Aufspaltung der gesamten Trainingsmenge in eine aktive Trainingsmenge $\mathbf{T}_{aktiv}$ und eine (kleinere) Testmenge $\mathbf{T}_{test}$, s. Abbildung 2.1.

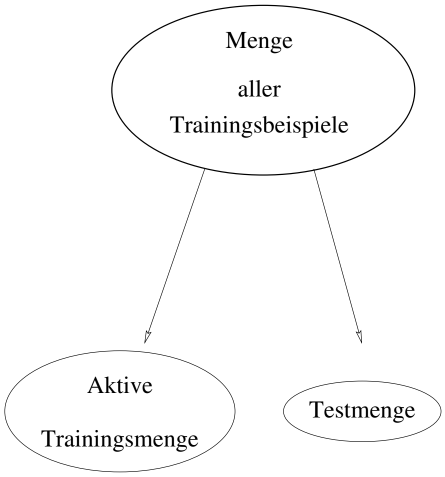

Abbildung 2.1: Zur Bewertung bzw. Verbesserung der Verallgemeinerungsfähigkeit des Lerners wird die Gesamtmenge der Trainingsbeispiele aufgespalten. Der Lerner sieht während des Lernens nur die Beispiele aus der aktiven Trainingsmenge $\mathbf{T}_{aktiv}$, seine Verallgemeinerungsfähigkeit erweist sich auf der (i. a. kleineren) Menge $\mathbf{T}_{test}$ der Testbeispiele.

Der Verlauf des Fehlers über den Lernschritten ist dann für die aktive Trainingsmenge und Testmenge unterschiedlich. Mit der Entwicklung eines

"Verständnisses" der Daten durch Lernen auf der aktiven Trainingsmenge wird sich auch der Fehler auf der Testmenge, deren Daten ja dem gleichen Bildungsgesetz unterliegen wie die aktiven Trainingsdaten, verringern. Das geht so lange bis das Overfitting der Daten der aktiven Trainingsmenge einsetzt. Damit ist der Wiederanstieg des Fehlers auf der Testmenge das geeignete Signal für den Abbruch des Lernens.

Kreuzvalidierung (cross validation)

Sind nur wenige Daten vorhanden, so kann die Testmenge nicht hinreichend groß gewählt werden, um eine verläßliche Aussage über das mittlere Verhalten des Lerners zu gewinnen. Die Aufteilung der Daten $x \in T$ auf das Mengenpaar $\mathbf{T}_{aktiv}, \mathbf{T}_{test}$ kann aber – etwa in einem Stichprobenverfahren – wiederholt ausgeführt werden. Auf jedem $\mathbf{T}_{aktiv}$ wird dann ein Lerner trainiert und wie oben auf $\mathbf{T}_{test}$ evaluiert. Das so geschaffene Ensemble von Lernern bietet eine größere statistische Sicherheit für die Performanzbeurteilung.

Besteht speziell die Testmenge nur aus einem Datum, so können wir N Mengenpaare

$$\mathbf{T}^{(k)} = \{\mathbf{T} \backslash z_k, z_k\} \tag{2.31}$$

bilden, wobei $z_k = (x_k, y_k^{soll}), x_k \in \mathbf{R}^n, y_k \in \mathbf{R}^m$ das $k-$te Datum aus $\mathbf{T}$, $\mathbf{T} \backslash z_i$ die Trainingsmenge ohne dieses spezielle Datum und N die Gesamtzahl der Daten in $\mathbf{T}$ ist. Jeder Lerner k lernt individuell auf seiner Menge der aktiven Trainingsdaten $\mathbf{T}_{aktiv} = \mathbf{T} \backslash z_k$, sein Fehler $E_k(w)$ wird bezüglich seines einen Testdatums z_k bestimmt

$$E_k = \frac{1}{2}(S(w|x_k) - y^{soll})^2 \tag{2.32}$$

Der Fehler über das Ensemble aus N Lernern ist dann

$$E = \frac{1}{N} \sum_{k=1}^{N} E_k \tag{2.33}$$

Dieser kann wie im Fall einer hinreichend großen Testmenge als Kriterium für den vorzeitigen Abbruch des Lernens zur Vermeidung des Overfittings eingesetzt werden.

Hat man verschiedene Lerner, so kann man zu jedem eine solche Ensembleaussage gewinnen auf deren Basis eine Auswahl unter den Lernern getroffen werden kann. Unterschiedlich strukturierte Lerner sind beispielsweise durch Funktionsansätze mit unterschiedlich großer Parameterzahl gegeben. Wird z. B. ein Polynomansatz für den Lerner gemacht

$$S(w|x) = \sum_{l=0}^{M} w_l x^l, \quad x \in \mathbf{R}^1 \tag{2.34}$$

so ist die Zahl der Parameter und damit die Flexibilität des Lerners durch M gegeben.

2.5 Simulated Annealing

Das Gradientenabstiegsverfahren ist ein sog. *greedy* Algorithmus, d. h. seine Strategie besteht darin, den Fehler in jedem Schritt um den maximal möglichen Betrag zu vermindern. Damit steuert er auf schnellstem Wege das nächstgelegene Minimum an, ohne Rücksicht darauf, daß es vielleicht noch andere, tiefere Minima gibt. Er erreicht folglich immer nur das dem Startpunkt nächsterreichbare, lokale Minimum. In der Praxis kann man diesen Nachteil durch vielfach wiederholte Suche mit immer anderen Startpunkten teilweise ausräumen.

Oft sind aber die Fehlerflächen durch eine Vielzahl lokaler Minima charakterisiert. Ist zudem der Suchraum hochdimensional, dann kann die Zahl der erforderlichen Startpunkte schnell explodieren. Eine sehr erfolgreiche Strategie ist in diesen Fällen das "simulated annealing", das auf der Einführung einer stochastischen Update–Regel beruht. In der Terminologie des stochastischen Gradienten bzw. der Langevin-Gleichung (s. Kap. 2.9) kann man auch von der Einführung eines künstlichen Rauschterms sprechen.

2.5.1 Stochastische Update–Regel

Algemein ergibt sich eine stochastische Update–Regel aus einer deterministischen immer so, daß der Vorgabe der deterministischen Regel nur mit einer gewissen Wahrscheinlichkeit P_{akz} (der sog. Akzeptanzwahrscheinlickkeit) gefolgt wird bzw. daß mit der dazu komplementären Wahrscheinlichkeit $P_{abl} = 1 - P_{akz}$ das Gegenteil ausgeführt wird. Im Falle des Gradientenabstiegsverfahrens

$$\Delta w_i = -\epsilon \frac{\partial E(w)}{\partial w_i} \tag{2.35}$$

heißt das, daß

$$\text{Update } w \rightarrow w + \Delta w \quad \text{mit Wahrscheinlichkeit } P_{akz} \tag{2.36}$$

ausgeführt wird, ansonsten

$$\text{Update } w \rightarrow w - \Delta w \tag{2.37}$$

Es wird also mit der Rückweisungswahrscheinlichkeit $P_{abl} = 1 - P_{akz}$ genau das Gegenteil des von der Gradientenregel geforderten Updates und damit ein Schritt **aufwärts** in der Fehlerlandschaft ausgeführt[2].

Einen geeigneten Ansatz für diese Wahrscheinlichkeit findet man durch folgende Überlegung. Sicher sollte die Wahrscheinlichkeit eine Funktion von ΔE sein und zwar so, daß große ΔE (weite Schritte) eher akzeptiert werden als kleine, wobei die Stärke der Abhängigkeit durch einen Parameter β regulierbar sein sollte. Eine geeignete Funktion ist die logistische Funktion (eine tiefergehende Begründung findet sich in Kap. 2.5.3), d. h. wir schreiben

$$P_{akz} = \frac{1}{1 + e^{\beta \Delta E}} \tag{2.38}$$

(beachte daß $\Delta E < 0$). In der Physik beschreibt die dort Fermifunktion genannte Funktion in 2.38 die Übergangswahrscheinlickeit zwischen verschiedenen Energieniveaus unter dem Einfluß eines Umgebungsmediums der Temperatur $T = 1/\beta$. Im aktuellen Kontext ist es deshalb üblich, den Parameter β als Inverses einer *fiktiven* Temperatur T zu interpretieren

$$\beta = \frac{1}{T}$$

Zum Verständnis betrachten wir die folgenden Fälle:

1. $\beta|\Delta E| \ll 1$: Es ist also $|\Delta E| \ll 1/\beta$ bzw. $|\Delta E| \ll T$. Das ist der Limes sehr hoher Temperaturen. Man sieht unmittelbar, daß in der Grenze $T \rightarrow \infty$ $P_{akz} = P_{abl} = 1/2$. Die deterministische Vorschrift wird also nur in der Hälfte der Fälle befolgt, d. h. sie spielt gar keine Rolle.

2. $\beta|\Delta E| \gg 1$: Es ist also $|\Delta E| \gg 1/\beta$ bzw. $|\Delta E| \gg T$. Das ist der Limes sehr niedriger Temperaturen. Im Limes $T \rightarrow 0$ folgt $P_{akz} = 1$, d. h. man erhält wieder die derministische Vorschrift (greedy Strategie).

[2] Das kann auch für den Udate Δw_i jeder Komponente w_i individuell ausgeführt werden

Wie man aus 2.38 unmittelbar ableitet gilt auch

$$\frac{P_{akz}}{P_{abl}} = e^{-\beta \Delta E} = e^{-\Delta E/T} \tag{2.39}$$

2.5.2 Simulated Annealing

Wie man sieht, wird für endliche Temperaturen in Abhängigkeit von T die greedy Strategie nur mehr oder weniger "sklavisch" befolgt. Mit anderen Worten, in Abhängigkeit von der Temperatur kommt man aus einem Minimum, in das man einmal geraten ist, mit gewisser Wahrscheinlichkeit (Entweichwahrscheinlichkeit, s. a. Kapitel 2.6) auch wieder heraus.

Die Strategie des Simulated Annealing ist dann, den Lernprozeß mit einer recht hohen Temperatur zu beginnen, so daß man relativ unabhängig von der genauen Form der Fehlerlandschaft große Bereiche des Parameterraumes abläuft. Dann wird die Temperatur und damit diese Entweichwahrscheinlichkeit nach einem geeigneten Regime langsam abgesenkt. Da diese Entweichwahrscheinlichkeit um so kleiner ist, je tiefer das Minimum, kann man hoffen, daß man mit fortschreitender Abkühlung mit hoher Sicherheit in einem besonders tiefen Minimum steckenbleibt, vgl. [17, 19, 18].

Bemerkung: Das "echte" Simulated Annealing verwendet keine Gradienteninformation sondern erzeugt die Verschiebung Δw rein zufällig. Die folgenden Überlegungen beziehen sich streng genommen auf diesen "echten" Fall. Die hier vorgestellte Variante ist sozusagen eine Hybridversion zwischen einem echten SA und einem "echten" Gradientenabstieg.

2.5.3 Die Natur des stochastischen Lernprozesses

Durch den Übergang zur stochastischen Update–Regel entspricht der Zeitverlauf des Parametervektors einer konkreten Realisierung eines (Markovschen) stochastischen Prozesses. Bei festgehaltener Temperatur erreicht dieser Prozeß einen stationären Zustand mit besonders einfachen Eigenschaften, was eben durch die spezielle Wahl von P_{akz}, s. Gleichung 2.38 garantiert ist. Insbesondere ist die Wahrscheinlichkeitsverteilung $P(w)$ der Werte des Parametervektors w gerade die aus der Statistischen Mechanik gut bekannte Boltzmann–Gibbs– Verteilung

[3]

$$P(w) = \frac{1}{Z}\exp(-\beta E(w)) \tag{2.40}$$

wobei Z die Normierung garantieren soll.

Für die Übergangswahrscheinlichkeit $P_{tr}(w \to w')$ gilt das Prinzip der detaillierten Bilanz, d. h. daß die mittlere Zahl der Übergänge von einem Parametervektor w nach w' gleich der Zahl der Übergänge von w' nach w ist

$$P(w)P_{tr}(w \to w') = P(w')P_{tr}(w' \to w) \tag{2.41}$$

Des weiteren gilt für die Entweichwahrscheinlichkeit aus einem Minimum die bekannte Kramer's Regel, die im Kapitel 2.6 behandelt wird. Weitere interessante Anregungen findet man in [3].

2.6 **Entweichen aus einem lokalen Minimum – Die Kramers'sche Formel

Die Formulierung der Lerndynamik als stochastischer Prozeß insbesondere die explizite Darstellung durch eine Langevin– oder Fokker–Planck–Gleichung (s. Kap. 2.9), setzt die lernenden Systeme in Beziehung zu bestimmten, gut untersuchten physikalischen Systemen, für die eine Vielzahl von interessanten Ergebnissen vorliegt, s. z. B. [50, 40, 14, 15] Von besonderem Interesse für den Umgang mit lernenden Systemen sind etwa Aussagen über die mittlere Zeit, die der Lerner braucht, um aus einem Minimum zu entweichen.

Angenommen wir führen ein stochastisches Updaten wie im Kap. 2.5 beschrieben aus oder unser Lernprozeß ist durch die Langevingleichung (2.55) beschrieben (s. u.). Sei w der Einfachkeit halber ein Skalar. [4] Als Beispiel betrachten wir eine Fehlerfunktion $E\,(w) = x^2 e^{-x}$. Diese hat ein Minimum an $w = w_{\min} = 0$ und ein Maximum an $w = w_{\max} = 2$. Wir nennen den Bereich $w < 2$ das Bassin des Minimums an $w = 0$ und können nun fragen, wie lange es im Mittel dauert, bis der

[3] Ausführungen zur Statistischen Analyse des Lernprozesses finden sich in Kap. 2.10

[4] Außerdem nehmen wir ein weißes Rauschen der Temperatur T in der Langevingleichung (2.55) an und setzen $E(w) = \langle E(w; x)\rangle$

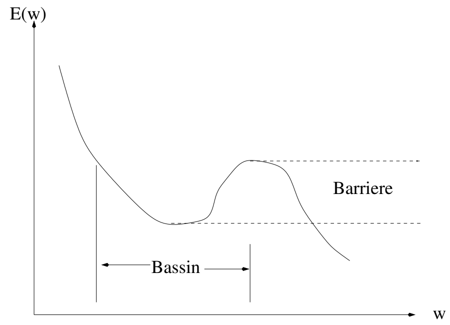

Abbildung 2.2: Ein lokales Minimum, begrenzt durch eine Barriere und das dazugehörige Bassin.

Lerner – charakterisiert durch seinen Parametervektor w – dieses Bassin verläßt. Das kann genauer gefaßt werden, indem wir nach der sog. *mean first passage time* t_{mfp} fragen, d. h. also nach der Zeit, die der Parameter unter der Lerndynamik (im Mittel über viele solche Versuche) braucht, bis er – bei $w = w_{\min}$ startend – zum ersten Mal die Bassingrenze, d. h. den Punkt $w = w_{\max}$ überschreitet.

Man wird nun erwarten, daß t_{mfp} natürlich von der Temperatur des Rauschens und von der genauen Form der Fehlerbarriere abhängt, die der Lerner überwinden muß. Erstaunlicherweise gestaltet sich dieser Zusammenhang aber außerordentlich einfach, die schon in den 40-er Jahren von dem Physikochemiker *Kramers* gefundene Formel lautet[5]

$$\begin{aligned} t_{mfp} &= \frac{2\pi}{|E''(w_{\min})\, E''(w_{\max})|}\; exp(\frac{\delta E}{T}) \\ \delta E &= E(w_{\max}) - E(w_{\min}) \end{aligned} \tag{2.42}$$

wobei δE also die Höhe der Barriere [6], T die Temperatur des Rauschens und E'' die zweite Ableitung am Minimum $w_{\min}$ bzw. am Maximum $w_{\max}$ der das Bassin begrenzenden Barriere ist (im Beispiel ist $w_{\min} = 0$ und $w_{\max} = 2$). Der Vorfaktor drückt die Proportionalität zum Krümmungsradien $\rho \sim 1/|E''|$ der Fehlerfunktion an den beiden Extrema aus. Bei gleicher Barrierenhöhe ΔE entweicht der

[5] Die Formel gilt näherungsweise und zwar um so besser, je glatter die Fehlerlandschaft ist.

[6] δE hier also nicht ein Update sondern die Barrierenhöhe.

Lerner um so schneller aus dem Bassin, je weiter das Bassin bzw. je breiter die Schwelle (großer Krümmungsradius an $w = w_{\min}$ bzw. $w = w_{\max}$) ist.

In ihrem wesentlichen Teil – der Exponentialfunktion – hängt die *mean first passage time* offensichtlich ausschließlich von der Höhe und nicht von der genauen Gestalt der zu überwindenden Fehlerbarriere ab. Insbesondere geht die Stärke (Temperatur) des Rauschens nur hier ein. Bei gegebener Barrierenhöhe δE geht die *mean first passage time* für wachsende (abnehmende) Stärke des Rauschens exponentiell gegen Null (Unendlich).

Dieser Zusammenhang zwischen Barrierenhöhe ΔE und Temperatur des Umgebungsmediums ist in der physikalischen Chemie als *Arrhenius*'sches Gesetz der Zerfallszeit von Molekülen bekannt[7].

Eine Vielzahl weiterer interessanter Resultate findet sich vor allem in [14, 15]

2.7 Stochastische Gradientenverfahren. On–line Lernen.

Bisher haben wir angenommen, daß die Beispielmenge *a priori* bekannt ist, so daß die Kostenfunktion (2.6), die ja die Summation über alle $x \in \Omega$ erfordert, explizit berechnet werden kann. Das entspricht in vielen Fällen nicht den Gegebenheiten des Lernvorganges.

Außerdem mußten wir, um der Gefahr des "Versackens" in den lokalen Minima zu entgehen, künstliche Störterme einführen, s. Kap. 2.5. Bei den im folgenden besprochenen *on–line* Verfahren ergeben sich ähnlich hilfreiche Rauschterme automatisch.

2.7.1 *On-line* Lernen

Praxistypisch ist ein *on-line* Lernen, bei dem jedes dem Lerner präsentierte Trainingsbeispiel einen Lernschritt auslöst. Damit ist die Kostenfunktion vom aktuellen Trainingsbeispiel abhängig. Werden die Beispiele zufällig (mit einer gegebenen Wahrscheinlichkeitsverteilung) aus der Menge Ω der Trainingsbeispiele

[7] In den physikalischen Formeln erscheint T mit der Boltzmannkonstanten k multipliziert. Formel 2.42 bleibt gültig, wenn wir die Maßeinheit für die Temperatur so wählen, daß $k = 1$.

ausgewählt, so ergibt sich ein stochastisches Gradientenverfahren. Die Kostenfunktion hängt jetzt explizit von $\vec{z} = \left(x, \vec{y}^{soll}\right), \quad x \in \Omega$ ab, d. h. wir erhalten eine

Stochastische Kostenfunktion:

$$E\left(w,x\right) = \frac{1}{2}\sum_{i=1}^{m}\left(y_i - y_i^{soll}\right)^2 = \frac{1}{2}\sum_{i=1}^{m}\left[S_i\left(w,x\right) - y_i^{soll}\right]^2, \quad x \in \Omega \qquad (2.43)$$

Die bisher betrachtete Kostenfunktion E ist der Mittelwert der stochastischen Kostenfunktion. Der Lernalgorithmus ist nun ein

Stochastisches Gradientenverfahren:

$$\Delta w_i = -\epsilon \frac{\partial}{\partial w_i} E\left(w,x\right) \qquad (2.44)$$

Das ist ein **datengetriebener** Algorithmus, da bei jeder Präsentation eines Datums x ein Lernschritt vollzogen wird.

Gleichung (2.44) erzeugt mit $t = 0, 1, \ldots$ einen zeitdiskreten stochastischen Prozeß $w\left(t\right)$. Dieser Prozeß ist speziell ein Markow-Prozeß, wenn die Vektoren x zu jedem Zeitpunkt unabhängig voneinander aus der Menge Ω gezogen werden. In jedem Falle beschreibt $w\left(t\right)$ eine Zufallswanderung im Raum der Parametervektoren und die Frage ist, unter welchen Bedingungen dieses Verfahren gegen das Minimum der **mittleren** Kosten $E\left(w\right)$ konvergiert.

2.7.2 Stochastische Approximation

Der in Gleichung (2.44) eingeführte Gradient

$$G\left(w,x\right) = \bigtriangledown_w E\left(w,x\right) = \frac{\partial}{\partial w} E\left(w,x\right) \qquad (2.45)$$

heißt wegen seiner Abhängigkeit vom Zufallsvektor x stochastischer Gradient. Wegen dieser Abhängigkeit kann im allgemeinen $w\left(t\right)$ nicht gegen einen festen Wert konvergieren, weil der Gradient in Abhängigkeit von x immer wieder andere Werte annehmen wird. Eine Konvergenz kann aber erzwungen werden, wenn die Schrittweite ϵ des Verfahrens zunehmend reduziert ("eingefroren") wird, d. h. wir wählen den Lernparameter ϵ zeitabhängig

$$\Delta w_i = -\epsilon_t \frac{\partial}{\partial w_i} E\left(w,x\right) \qquad (2.46)$$

Unter geeigneten Annahmen für die stochastische Folge der Inputs x gilt folgender

Satz: Für eine stetig von w abhängige Funktion $E(w, x)$ konvergiert der stochastische Prozeß $w(t)$ mit Wahrscheinlichkeit 1 im quadratischen Mittel gegen einen festen Wert w_∞

$$P\left[\lim_{t\to\infty} \left\langle \|w(t) - w_\infty\|^2 \right\rangle \ < \ \eta\right] = 1$$

falls

$$\sum_{t=1}^{\infty} \epsilon_t = \infty, \qquad \sum_{t=1}^{\infty} \epsilon_t^2 < \infty \tag{2.47}$$

Der erreichte Parametervektor w_∞ entspricht dabei einem (nicht unbedingt dem globalen) Minimum der Funktion $E(w)$. Eine detaillierte Darstellung findet sich in [52, 51].
Obwohl wir mit dem stochastischen Gradienten und damit in einem *on–line* Lernverfahren arbeiten, erreichen wir durch das Abkühlen des Lernparameters letztendlich doch ein Minimum der gemittelten Kostenfunktion (2.6) bzw. (2.10),(2.12). Das ist der Sinn der stochastischen Approximation.

Da die Reihe $\sum_{t=1}^{\infty} \frac{1}{t^\alpha}$ für $\alpha \leq 1$ divergent ist, erfüllt

$$\epsilon_t = \epsilon_0 \frac{1}{t^\alpha} \qquad \text{mit} \qquad \frac{1}{2} < \alpha < 1 \tag{2.48}$$

das Kriterium 2.47. Etwas günstiger ist der Ansatz

$$\epsilon_t = \frac{1}{(t + t_0)^\alpha}, \qquad \frac{1}{2} < \alpha < 1 \tag{2.49}$$

bei dem über einen Zeitraum t_0 der Wert des Lernparameters relativ konstant bleibt und erst dann die Abkühlung voll greift. Der t_0 Term verhindert also die Überbewertung der anfänglich ($t \leq t_0$) präsentierten Trainingsbeispiele. Für $t >> t_0$ ist 2.49 dem Ansatz 2.48 äquivalent.

In der Praxis wird oft ein über einen Zeitraum T lineares Abkühlen von einem Anfangswert ϵ_0 auf einen danach konstant gehaltenen Endwert ϵ_0

$$\epsilon_t = \begin{cases} \epsilon_0 + \frac{t}{T}(\epsilon_1 - \epsilon_0) & \text{für} \quad 0 \leq t \leq T \\ \\ \epsilon_1 & \text{für} \quad T \leq t \end{cases} \tag{2.50}$$

bevorzugt. Eine andere Möglichkeit ist ein exponentielles Abkühlen. Dieses kann durch eine Update–Regel für das ϵ_t realisiert werden, d. h. wir inkrementieren ϵ_t

im Zeitschritt t durch

$$\Delta\epsilon_t = -\frac{1}{t_0}\epsilon_t \tag{2.51}$$

wobei t_0 die Zeitkonstante für das Abkühlen von ϵ ist.

Vorteile der stochastischen Approximation beim Lernen:

- Ermöglicht On-line-Betrieb, d. h. die Daten werden sequentiell verarbeitet.
- Neu anfallende Trainingsbeispiele können leicht "nachgelernt" werden.

2.8 Effektivierung der Lernverfahren. Stochastisches Manhattan–Lernen

Trotz ihrer allgemeinen Vorzüge können die stochastischen Gradientenverfahren oft auch in Schwierigkeiten geraten. Zu diesen gehören z. B. die langsame Konvergenzgeschwindigkeit in Bereichen wo die Fehlerfunktion einen sehr flachen Verlauf nimmt. Dazu kommt die Unsicherheit in der Wahl der Lernrate bzw. in der Abkühlungsstrategie die ja eigentlich die Kenntnis der Krümmung der Fehlerfläche vorraussetzt. Natürlich sind eine Vielzahl von Methoden zur Überwindung dieser und ähnlicher Schwierigkeiten bekannt, insbesondere wurden in den letzten Jahren in der Neuroinformatik interessante Lösungen gefunden. Besonders zu nennen sind Quickprop, Lernen mit Massetermen, *gradient reuse* Verfahren und vor allem Verfahren zur automatischen Anpassung des Lernparameters, vgl. die Vorlesung Neuroinformatik [5] und die einschlägige Literatur wie z. B. [41, 55].

Hier soll noch eine andere Möglichkeit, das sog. Manhattan–Lernen vorgestellt werden. Dabei wird der Gradient einfach durch sein Vorzeichen ersetzt

$$\Delta w_i = -\epsilon sign(\frac{\partial E(w,x)}{\partial w_i}) \tag{2.52}$$

Mit einer geeigneten Abkühlungsstrategie kann man eine stark beschleunigte Konvergenz des Lernens erreichen. Besonders interessant vom Standpunkt der theoretischen Analyse von Lernverfahren ist die Tatsache, daß kürzlich für das Manhattan–Lernen ein exakter Ausdruck für den Zeitevolutionsoperator und damit für die Dynamik des Systems gefunden werden konnte, s. [29].

2.9 **Stochastisches Gradientenverfahren als Langevin–Dynamik

Die stochastische Approximation kann durch die Umwandlung der Update–Regel in eine *Langevin*–Gleichung besonders deutlich veranschaulicht werden. Die Langevin–Gleichung ist aus der Physik bekannt und beschreibt die Bewegung eines Teilchens in einem Medium unter dem Einfluß der stochastischen Kräfte, die durch die Stöße mit Umgebungsmolekülen verursacht werden (thermische Fluktuationen — Brownsche Molekularbewegung). Sehr gute Darstellungen findet man in den Monografien [50, 40, 14, 15] u. a.

Wir betrachten die allgemeine Update–Regel 2.13

$$\Delta w_i = -\epsilon \frac{\partial E}{\partial w_i} \tag{2.53}$$

und beachten, daß $E = E(w, x)$, d. h. daß E eine stochastische Größe ist, da sie ja von dem Wert der stochastischen Variablen x abhängt. Die x werden mit einer Wahrscheinlichkeit $V(x)$ aus der Menge X der N Datenpunkte gezogen und für jedes x wird ein Lernschritt gemäß 2.53 durchgeführt.

Die Idee ist, den stochastischen Gradienten (2.45) als Summe seines systematischen und eines rein stochastischen Anteiles zu schreiben, d. h.

$$G_i = \frac{\partial}{\partial w_i} E(w, x) = -\phi_i(w) + \xi_i(w, x) \tag{2.54}$$

wobei

$$\begin{aligned}
\phi_i(w) &= -\left\langle \frac{\partial}{\partial w_i} E(w, x) \right\rangle \\
&\text{und} \\
\xi_i(w, x) &= \frac{\partial}{\partial w_i} E(w, x) - \left\langle \frac{\partial}{\partial w_i} E(w, x) \right\rangle = \frac{\partial}{\partial w_i} [E - \langle E \rangle]
\end{aligned}$$

so daß ϕ_i die systematisch wirkende Verschiebung des Parameters w_i darstellt.

Die Langevindarstellung der stochastischen Updateregel lautet damit

$$\Delta w_i = \epsilon \phi_i(w) + \xi_i \tag{2.55}$$

Der Sinn der Zerlegung folgt aus

$$\langle \xi_i(w, x) \rangle = 0 \tag{2.56}$$

d. h. daß die mittlere Wirkung von ξ_i auf den Update Δw_i verschwindet. Werden die x rein zufällig aus der Menge der Inputs gezogen, dann ist ξ immer ein weißes Rauschen, da ja die einzelnen Instanzen nicht voneinander abhängen, d. h. wennn x und x' Inputs zu zwei verschiedenen Zeiten t und t' sind, dann gilt

$$\langle \xi_i (w, x) \, \xi_j (w, x') \rangle = 0 \quad \forall i, j \tag{2.57}$$

wobei $\langle ... \rangle$ das Mittel über alle x und x' bedeutet.

Der Zusammenhang mit der Physik ergibt sich, indem wir $w \in R^n$ als den Ort eines **massefreien** Teilchens im n–dimensionalen Raum interpretieren, das sich in einem viskosen Medium unter dem Einfluß der Kraft $\phi = (\phi_1, ..., \phi_n)$ bewegt. Wegen

$$\left\langle \frac{\partial}{\partial w_i} E (w, x) \right\rangle = \frac{\partial}{\partial w_i} \langle E (w, x) \rangle \tag{2.58}$$

ist offensichtlich die mittlere Fehlerfunktion $E (w)$ das Potential der Kraft ϕ in der Langevingleichung.

Der Satz 2.7.2 über die Konvergenz des stochastischen Gradientenfolgeverfahrens besagt dann nichts anderes als daß durch die Abkühlung der Lernrate die w_i asymptotisch nur noch durch die mittlere Fehlerfunktion gesteuert werden, so daß auf diese Weise ein Minimum von $E (w)$ angelaufen wird.

Das "Rauschen" kann mit der Temperatur des fiktiven Umgebungsmediums in Beziehung gesetzt werden. Dazu betrachten wir die Korrelationsfunktion

$$\langle \xi_i (w, x) \, \xi_j (w, x) \rangle = D_{ij} \tag{2.59}$$

und setzen etwa $\frac{1}{n} \sum D_{ii} = T$.

Für hinreichend kleine ϵ kann die Differenzengleichung 2.55 in eine Differentialgleichung überführt werden, folglich stehen anstelle von (2.55) die stochastischen Differentialgleichung

$$\dot{w}_i (t) = \epsilon \phi_i (w (t)) + \xi_i (t) \tag{2.60}$$

In wichtigen Anwendungsfällen [9] gilt, daß das Rauschen in niedrigster Ordnung von ϵ nicht von den Parametern w_k abhängt. In dieser Näherung ist 2.60 dann eine stochastische Differentialgleichung mit **additivem** weißem Rauschen.

Dann kann die Differentialgleichung leicht in eine Fokker-Planck-Gleichung für die Wahrscheinlichkeit $P (w, t)$ umgeschrieben werden. $P (w, t) \, dw$ sei wie bisher die Wahrscheinlichkeit, daß zum Zeitpunkt t die Parameter $w_i, \quad i = 1, ..., n$ Werte im Intervall $[w_i, w_i + dw_i]$ annehmen. Die Fokker-Planck-Gleichung für den Fall einer einzigen Variablen w lautet

$$\frac{\partial}{\partial t} P(w,t) = -\frac{\partial}{\partial w}\phi(w) P(w,t) + T\frac{\partial^2}{\partial w^2} P(w,t) \tag{2.61}$$

wenn wir standardmäßig das (weiße) Rauschen so normieren, daß

$$\langle \xi(t)\, \xi(t') \rangle = 2T\delta(t - t')$$

Die rechte Seite der Fokker-Planck-Gleichung besteht aus dem systematischen oder Driftterm (erster Term) und dem Diffusionsterm, der die Rolle der stochastischen Kräfte widerspiegelt. Der Diffusionskoeffizient wurde in die Temperatur inkorporiert. Das ist in dem vorliegenden Fall einer fiktiven Temperatur angebracht.

Für große Zeiten t konvergiert die Wahrscheinlichkeitsverteilung $P(w,t)$

$$\lim_{t\to\infty} P(w,t) = P_{stat}(w) \tag{2.62}$$

gegen die stationäre Verteilung

$$P_{stat}(w) = \frac{1}{Z} e^{-\frac{E(w)}{T}} \tag{2.63}$$

mit Normierung

$$Z = \int dw\, e^{-\frac{E(w)}{T}}$$

Für eine hinreichend glatte Fehlerfunktion liefert die Taylorentwicklung um ein Minimum $w = w_{\min}$ die Näherung

$$E(w) = E(w_{\min}) + \frac{1}{2}(w - w_{\min})^2 E''(w_{\min})$$

(da $E'(w_{\min}) = 0$). Im Gültigkeitsbereich der Näherung ist P_{stat} durch die Gaußsche Normalverteilung gegeben

$$P_{stat} = \frac{1}{Z} e^{-\left(\frac{w - w_{\min}}{\rho}\right)^2} \tag{2.64}$$

wobei $\rho = 1/E''(w_{\min})$ der Krümmungsradius der Fehlerfunktion im Minimum ist. Offensichtlich ist w in einem Bereich der Ausdehnung ρ um $w_{\min}$ konzentriert, da ja $P_{stat}(w) \approx 0$ für $|w - w_{\min}| >> \rho$.

2.10 ***Statistische Analyse

Die Lernregel (2.44) definiert einen zeitdiskreten stochastischen Prozeß (und zwar einen Markov—Prozeß) dessen Eigenschaften durch Wahrscheinlichkeitsaussagen festgelegt sind. Beispielsweise können wir Aussagen über das mittlere Verhalten des Parametervektors w gewinnen. Für die Analyse des Lernprozesses wurden besonders erfolgreich Methoden der Statistischen Mechanik herangezogen, vgl. [22, 28, 3]. Wir wollen uns hier mit einigen Anfangsgründen der statistischen Analyse lernender Systeme begnügen.

2.10.1 Lerndynamik im Wahrscheinlichkeitsbild

Zur Gewinnung von Wahrscheinlichkeitsaussagen muß im Prinzip der Lernvorgang jeweils bei $t = 0$ beginnend hinreichend (unendlich) oft wiederholt ausgeführt werden. Die Startwerte für den Parametervektor w sind dabei durch eine Wahrscheinlichkeitsverteilung $P_0(w)$ vorgegeben zu denken. Für jeden konkreten Startwert $w = w_0$ erzeugt die wiederholte Anwendung der Lernregel eine konkrete Realisierung des stochastischen Prozesses. Die Beobachtung hinreichend vieler Realisierungen erlaubt die Ermittlung des zeitlichen Verhaltens der Wahrscheinlichkeitsverteilung

$$P(w;t) \quad \text{mit} \quad \int_{-\infty}^{+\infty} dw\, P(w;t) = 1 \tag{2.65}$$

und $P(w;0) = P_0(w)$. Wir können sagen, daß $P(w;t)$ die Lerndynamik im Wahrscheinlichkeitsbild beschreibt.

Statt den Lernvorgang wiederholt auszuführen kann man sich in Analogie zum Vorgehen in der Statistischen Mechanik auch ein **Ensemble** von hinreichend (unendlich) vielen Lernern vorstellen, die alle gleichzeitig mit einem individuell gemäß $P_0(w)$ ausgewähltem w gestartet und in jedem Zeitschritt nach der gemeinsamen Lernregel mit für jeden Lerner individuellem Input x upgedatet werden. Die Inputs, die die einzelnen Lerner in einem Zeitschritt sehen, sind also gemäß $V(x)$ verteilt.

Die so ermittelte Wahrscheinlichkeit ist in beiden Fällen gleich. Mit Hilfe dieser Wahrscheinlichkeitsverteilung können wir beliebige Mittelwerte bilden, z.

B. ist das mittlere Verhalten von w selbst durch

$$\overline{w}(t) = \int dw\, w\, P(w;t) \tag{2.66}$$

gegeben.

Die Definition der Anfangswerte über die Verteilung $P_0(w)$ schließt den Fall des immer gleichen Startwertes w_0 für den Parametervektor w mit ein. Dazu wählen wir

$$P_0(w) = \delta(w - w_0) \tag{2.67}$$

wobei δ die sog. δ—Funktion (genauer gesagt δ–Distribution) ist, die normiert ist

$$\int_{-\infty}^{+\infty} dw\, \delta(w - w_0) = 1 \tag{2.68}$$

und die unendlich scharfe Verteilung realisiert, d. h. es gilt

$$\int_{a-\frac{\epsilon}{2}}^{a+\frac{\epsilon}{2}} dw\, \delta(w - a) = 1 \tag{2.69}$$

für beliebig kleine ϵ. Als eine näherungsweise Realisierung der δ–Funktion kann man sich eine Stufe der (infinitesimal kleinen) Breite η und Höhe $1/\eta$ vorstellen

$$\delta(w-a) = \begin{cases} 0 & \text{für} \quad |w-a| > \frac{\eta}{2} \\ \\ \frac{1}{\eta} & \text{sonst} \end{cases} \tag{2.70}$$

2.10.2 Übergangswahrscheinlichkeit

Die Werte des Parametervektors w sind zu jedem Zeitpunkt (d. h. im t—ten Lernschritt) gemäß $P(w,t)$ verteilt. Durch Ausführen des Lernschrittes im Ensemble verändert sich die Wahrscheinlichkeit von $P(w;t)$ in $P(w;t+1)$. Beide Wahrscheinlichkeiten sind durch die sog. Übergangswahrscheinlichkeit miteinander verbunden.

Die Übergangswahrscheinlichkeit $P(w' \mid w)$ ist die Wahrscheinlichkeit für den Übergang $w \to w'$ im t–ten Lernschritt. Genaugenommen ist das wieder eine Wahrscheinlichkeitsdichte, d. h. hat der Parametervektor im Zeitschritt t den Wert w, so ist die Wahrscheinlichkeit, daß nach Ausführen des Lernschrittes der neue Wert $w' = w + \Delta w$ im Intervall $[w', w' + dw']$ liegt, durch

$$P(w' \mid w)\, dw' \tag{2.71}$$

gegeben. Die Wahrscheinlichkeiten summieren sich über alle Intervalle zu eins,

$$\int dw' \, P\left(w' \mid w\right) = 1 \tag{2.72}$$

(Normierung).

Die Übergangswahrscheinlichkeiten können aus der Lernregel (2.44), d. h.

$$\Delta w = -\epsilon \frac{\partial E}{\partial w} = -\epsilon G\left(w, x\right)$$

explizit bestimmt werden. Das System führt den Übergang $w \rightarrow w'$ im Lernschritt ja genau dann aus, wenn $w' = w + \Delta w$ bzw. wenn $\Delta w = -\epsilon G\left(w, x\right) = w' - w$, genauer wieder, wenn Δw im Intervall $|w' - w, w' - w + dw'|$ liegt. Das ist nur für ganz bestimmte x der Fall. Wir müssen also genau die relative Zahl der Fälle finden, für die das gilt. Mit der approximativen Definition 2.69 der δ–Funktion verifiziert man leicht, daß

$$P\left(w' \mid w\right) = \int_{-\infty}^{+\infty} dx \, P\left(x\right) \, \delta\left(w' - w + \epsilon G\left(w, x\right)\right) \tag{2.73}$$

die richtige Formel zur Bestimmung der Übergangswahrscheinlichkeit ist.

Mit 2.68 und

$$\begin{aligned} & \int_{-\infty}^{+\infty} dw' \int_{-\infty}^{+\infty} dx \, P\left(x\right) \, \delta\left(w' - w + \epsilon G\left(w, x\right)\right) \\ = & \int_{-\infty}^{+\infty} dx P\left(x\right) \int_{-\infty}^{+\infty} dw' \, \delta\left(w' - w + \epsilon G\left(w, x\right)\right) \end{aligned}$$

verifiziert man auch unmittelbar, daß

$$\int_{-\infty}^{+\infty} dw' P\left(w' \mid w\right) = 1$$

so daß die so ermittelte Übergangswahrscheinlichkeit tatsächlich normiert ist.

Mit der Übergangswahrscheinlichkeit finden wir nun den Zusammenhang zwischen den Wahrscheinlichkeiten zu den verschiedenen Zeitpunkten und damit eine iterative Formel für die Dynamik (zeitliche Veränderung) der Wahrscheinlichkeit:

$$P\left(w'; t+1\right) = \int_{-\infty}^{+\infty} dw P\left(w' \mid w\right) P\left(w; t\right) \tag{2.74}$$

In der Tat, $P\left(w'; t+1\right)$ ist die Wahrscheinlichkeitsdichte dafür, daß der neue Wert des Parametervektors w' ist. Der Ausdruck 2.74 ist die gewichtete Summe über alle die Fälle in denen w' erreicht wurde, wobei das Gewicht die relative Häufigkeit der Fälle ist, bei denen der Übergang von einem bestimmten w ausging.

Mit 2.72 verifiziert man unmittelbar die Erhaltung der Normierung, d. h.

$$\int_{-\infty}^{+\infty} dw P\left(w; t+1\right) = \int_{-\infty}^{+\infty} dw P\left(w; t\right) = 1 \tag{2.75}$$

wie es sein muß.

2.10.3 Vergessen der Anfangsbedingungen

Durch Iteration von Gleichung 2.74 mit der Startbedingung

$$P\left(w; 0\right) = P_0\left(w\right) \tag{2.76}$$

kann der zeitliche Verlauf der Wahrscheinlichkeiten und damit auch der aus diesen bestimmten Mittelwerte, vgl. 2.66, berechnet werden. Das sieht allerdings recht kompliziert aus und ist für praxisrelevante Systeme sicher nur schwer auszuführen. Insbesondere hängen die Wahrscheinlichkeiten und damit das Verhalten des Systems zu allen Zeiten von der Startverteilung $P_0\left(w\right)$ ab.

In den meisten praktischen Fällen ergibt sich allerdings eine starke Vereinfachung: Die Anfangsverteilung wird relativ schnell vergessen (ist transient). Danach hängt der Verlauf des Lernprozesses wesentlich nur vom Verhalten des Parametervektors $w\left(t\right)$ innerhalb eines begrenzten Zeithorizontes in der Vergangenheit ab. In vielen Fällen ist dabei nur das Verhalten des Mittelwertes selbst innerhalb dieses Zeithorizontes relevant. Dazu kommt, daß sich für das Verhalten des Mittelwertes zumindest näherungsweise eine geschlossene Gleichung finden läßt (s. unten). In diesem Falle kann man dann also die vergangenen Mittelwerte innerhalb des Zeithorizontes aus den aktuellen Mittelwerten bestimmen. Eine tiefergehende Diskussion dieser Fragestellungen im Rahmen der statistischen Mechanik findet sich in den Publikationen [7, 8].

Unter diesen Umständen kann man annehmen, daß $P\left(w; t\right)$ nur von diesen Mittelwerten und nicht mehr explizit von der Zeit abhängen, d.h.

$$P\left(w; t\right) \Rightarrow P_{<w(t)>}\left(w\right) \tag{2.77}$$

wobei der Index die parametrische Abhängigkeit der Wahrscheinlichkeit von den Mittelwerten ausdrücken soll. Beispiel ist eine um den Mittelwert zentrierte

Gaussverteilung mit eventuell ebenfalls vom Mittelwert abhängiger Breite σ, d. h. für den eindimensionalen Fall

$$P_{<w(t)>}(w) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(w- < w(t) >)^2}{2\sigma^2}\right) \tag{2.78}$$

Damit kann man nun eine selbstkonsistente Gleichung für die Mittelwerte selbst ableiten. Mit der allgemeinen Lernregel

$$\Delta w = -\epsilon \frac{\partial E}{\partial w} = -\epsilon G(w, x) \tag{2.79}$$

wobei $w = w(t)$ und $w(t+1) = w(t) + \Delta w(t) = w(t) - \epsilon G(w, x)$ folgt durch Mittelung

$$\begin{aligned} \langle w(t+1)\rangle &= \langle w(t)\rangle + \langle \Delta w(t)\rangle & (2.80) \\ &= \int dw\, w P(w;t) - \epsilon \int dw\, dx\, G(w,x)\, P(x)\, P(w;t) & (2.81) \\ &= \langle w(t)\rangle - \epsilon \int dw\, \Phi(w)\, P(w;t) & (2.82) \end{aligned}$$

wobei

$$\Phi(w) = \int dx\, P(x)\, G(w,x) \tag{2.83}$$

der mittlere Update bei gegebenem w als aktuellem Wert ist.

Mit 2.77 folgt dann eben, daß sich diese Größen alle als Funktion des Mittelwertes ausdrücken lassen, d.h. wir schreiben

$$\int dw\, dx\, G(w,x)\, P(x)\, P(w;t) = \psi(\langle w(t)\rangle) \tag{2.84}$$

Die Update–Regel

$$\Delta w = -\epsilon \psi(w) \tag{2.85}$$

bestimmt dann (nach Abklingen der Transienten) exakt das Verhaltens des Mittelwertes des Parametervektors über die Zeit. Man beachte, daß Φ und ψ genau dann gleich sind, wenn wir 2.84, d.h. $\psi(\langle w(t)\rangle) = \int dw\, \Phi(w)\, P(w;t)$ schreiben dürfen

$$\int dw\, \Phi(w)\, P(w;t) = \Phi\left(\int dw\, w\, P(w;t)\right) = \Phi(\langle w\rangle)$$

Das ist genau dann exakt richtig, wenn die Verteilung P unendlich scharf ist, wenn also z. B. in der Gaussverteilung 2.78 $\sigma \to 0$ gilt. In der Näherung ist es gut, wenn die Fluktuationen nicht zu groß sind. Allgemein kann man schreiben

$$\psi(w) = \Phi(w) + O(\epsilon) \tag{2.86}$$

weil die Größe der Fluktuationen um den Mittelwert durch den Lernparameter ϵ gegeben ist. Das ist allerdings noch keine interessante Aussage, weil das Inkrement des Mittelwertes auch mit ϵ skaliert. Über einen festen Zeitraum betrachtet geht der Einfluß der Korrekturterme der Ordnung ϵ damit nicht notwendigerweise gegen Null. Es bedarf folglich einer besonderen Untersuchung der durch die Inputs x verursachten Stärke der Fluktuationen, um das Gleichsetzen von ψ und Φ zu rechtfertigen.

Kapitel 3

Lernende Klassifikatoren

Die im Kapitel 2 betrachteten Verfahren dienen dem Erlernen einer reellwertigen Funktion aus Beispielen. In diesem Kapitel wollen wir uns ebenfalls mit dem Lernen aus Beispielen beschäftigen. Der Lerner soll jetzt aber die Fähigkeit erwerben, Objekte oder Situationen auf Grund verschiedener Merkmale in Klassen einzuordnen, wobei gegebenenfalls diesen Klassen bestimmte Aussagen (Bedeutungen) zugeordnet werden können. Anwendung finden Klassifikatoren z.B. bei der Diagnose in der Medizin, der automatischen Sprachverarbeitung [44], der Datenkompression [45], der Signalaverarbeitung [16] u. a.

Neben ihrer eigenständigen Bedeutung als Werkzeug zur direkten Einordnung von Objekten oder Sachverhalten in die vorgegebenen Klassen spielen Klassifikatoren für das Maschinelle Lernen eine wichtige Rolle zur Datenaufbereitung. Die "klassischen" Verfahren des Maschinellen Lernens sind regel- oder logikbasiert (vgl. etwa das Konzeptlernen in Kap. 5) und setzen auf einer symbolischen Objektbeschreibung auf. Stärker noch als andere Teilgebiete der künstlichen Intelligenz ist aber das Maschinelle Lernen in der Praxis mit "schmutzigen" Daten konfrontiert, d. h. die anfallenden Daten sind reellwertig, verrauscht und oft unvollständig oder unzuverlässig. Dann kann die Vorverarbeitung im Sinne einer Einteilung der Daten in bestimmte Klassen eine wesentliche Verbesserung der Entscheidungsfähigkeit etwa eines Konzeptlerners bringen.

Das Erlernen eines Klassifikators anhand vorgegebener Beispiele ist nicht nur Gegenstand des Maschinellen Lernens sondern auch der klassischen Statistik und in jüngerer Zeit der Methode der neuronalen Netze. Eine vergleichende Bewertung verschiedener Klassifikationsverfahren ist im Abschlußbericht [33] des ESPRIT Projektes "StatLog" zu finden. Daneben gibt es eine reichhaltige Literatur, ein Klassiker für die mehr statistisch orientierten Zugänge ist das Buch von Duda und Hart [11]. In jüngster Zeit gibt es dazu vor allem auf dem Gebiet der künstlichen neuronalen Netze eine Vielzahl neuer Ideen zur Klassifikation. Wir wollen uns im folgenden zunächst mit Verfahren beschäftigen, die besonders für metrische Daten geeignet sind. Diese Verfahren stehen denen der statistischen Methode oder der neuronalen Netze nahe.

3.1 Definition des Klassifikators

Mathematisch ist der Klassifikator eine Funktion $K : x \to \kappa$, die also jedem Merkmalsvektor $x \in D$ aus der betrachteten Objektdomäne D ein Klassenlabel oder einen Klassennamen $\kappa \in A$ zuweist. Erlernen einer Klassifikation heißt Induktion dieser Funktion aus einer (kleinen) Menge T von Beispielen

$$T = \left\{ \left(x, \kappa^{soll}\right) \mid \quad x \in \Omega \subseteq D, \quad x \in R^n, \kappa^{soll} \in A \right\} \tag{3.1}$$

wobei Ω wieder die Domäne der Beispielobjekte (Trainingsmenge) ist. Die Sollvorgaben werden vom Lehrer geliefert

$$\text{Lehrer:} \qquad \kappa^{soll} = K\left(x\right) \tag{3.2}$$

Der lernende Klassifikator (Lerner) ist wie bisher durch seine Transferfunktion

$$\text{Lerner:} \qquad \kappa = S\left(x \mid w\right) \tag{3.3}$$

beschrieben, wobei w wieder das Verhalten des Lerners festlegt. In den unten behandelten Beispielen sind das (i) für den Fall des lernenden Vektorquantisierer und des k–nearest–neighbours Algorithmus ein Satz von klassenbeschreibenden Vergleichsvektoren oder (ii) im Fall des lernenden Bayes–Klassifikators die Parameter der Wahrscheinlichkeitsverteilungen und (iii) ein Satz von Regeln oder ein Entscheidungsbaum. Entsprechend unterschiedlich gestalten sich die zugehörigen Lernalgorithmen.

In (3.2) bzw. (3.3) weisen K bzw. S jedem x in genau definierter Weise je eine Klasse zu. Diese deterministische Klassifikationsregel entspricht oft nicht den

Erfordernissen des Umgangs mit *real world* (schmutzigen) Daten. Bei diesen ist oft selbst der Lehrer nicht in der Lage, eine sichere Klassifikation vorzunehmen. Das kann in das obige Schema durch Hinzunahme einer Rückweisungsklasse mit aufgenommen werden, in die alle nicht sicher klassifizierbaren Daten abgebildet werden.

Allgemeiner tragen Klassenzuweisungen bei korrumpierten (verrauschten) Daten oft nur Wahrscheinlichkeitscharakter. Ein geeigneter Ansatz zur Definition des optimalen Klassifikators findet sich im Abschnitt 3.4 über den lernenden Bayesklassifikator.

3.2 Der k–nearest–neighbours Algorithmus

Der k–nearest–neighbours Algorithmus ist eigentlich kein richtiger Lernalgorithmus (s. u.), er ist aber ein wichtiges Verfahren der Statistik zur Klassifikation mit sehr guter Leistungsfähigkeit und soll deshalb in die Darstellung mit aufgenommen werden.

Der k–nearest–neighbours Algorithmus geht davon aus, daß wir alle Beispiele der Trainingsmenge T im Speicher halten und daß ein Abstandsmaß $d(x, x')$ zwischen beliebigen Vektoren $x, x', \ldots$ des Datenraumes zur Verfügung steht. Die Klassifikation eines beliebigen Datums $x \in D$ erfolgt in zwei Schritten

1. Bestimmung der (bezüglich des Abstandsmaßes d) k nächstgelegenen Vektoren aus der Trainingsmenge T, d.h. wir bestimmen für alle Datenvektoren aus der Objektdomäne Ω der Trainingsmenge die Abstände

$$\delta_x(y) = d(x, y) \quad \forall y \in \Omega$$

   und suchen die k Vektoren y mit den kleinsten Werten für $\delta_x(y)$ heraus.

2. Bestimmung der Klassenzugehörigkeit durch ein "Mehrheitsvotum", d. h. ausgegeben wird die Klasse, die unter den k Vektoren am häufigsten vertreten ist (für die Vektoren der Trainingsmenge ist die Klasse ja bekannt).

Das ist die deterministische und damit einfachste Version. Hat man hinreichend viele Trainingsdaten und ausreichend Speicherplatz und Rechenpower, so kann

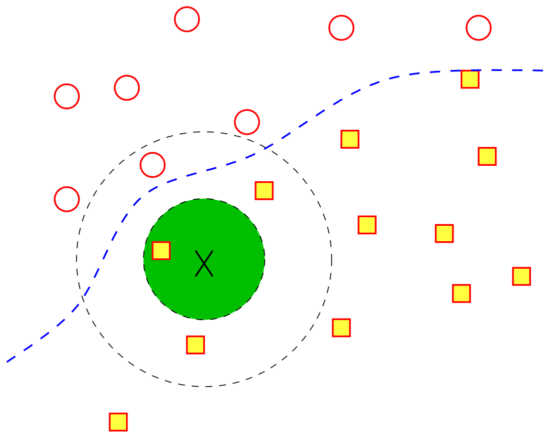

Abbildung 3.1: Klassifikation mit dem *k-nearest-neighbour* Algorithmus für $k = 1$ (innerer Kreis) und $k = 4$. Die Kreise bzw. Quadrate stellen die Datenvektoren der beiden Klassen dar. Gestrichelt ist die Grenze zwischen den beiden Klassendomänen eingezeichnet.

k sehr groß gemacht und aus den Klassenfrequenzen in den ausgewählten Vergleichsvektoren auf Wahrscheinlichkeiten bzw. Sicherheit der Zuweisung rückgeschlossen werden. Das soll hier aber nicht behandelt werden.

Bei der Anwendung des k–nearest–neighbours Algorithmus ist die richtige Wahl von k gütebestimmend. Diese entscheidet bei verrauschten Daten über die Balance zwischen Sicherheit der Klassifikation auf der Domäne der Trainingsdaten und Verallgemeinerungsfähigkeit der Klassifikation auf noch nicht gesehene Daten. In der Praxis erfolgt die richtige Wahl von k durch Ausprobieren unter Einsatz bestimmter Heuristiken wie etwa im im Kap. **??** über Gütekriterien beschrieben. Häufig wird die Methode des cross validation eingesetzt.

3.3 Erlernen einer Klassenrepräsentation

Für metrische Daten findet die Klassifikationsaufgabe eine einfache geometrische Interpretation. Die Merkmalsvektoren $x \in \mathcal{R}^n$ werden als Punkte im n - dimensionalen Raum gedacht. Die Klassen sind dann in eindeutiger Weise durch eine Aufteilung des Raumes in disjunkte Gebiete beschrieben, wobei jede Klasse auch mehrere solche Gebiete belegen kann, vgl. Abb. 3.1. Die Klassifikation entspricht dann der Identifikation des Gebietes bzw. der Gebietsgrenzen.

3.3.1 Repräsentanten

Ein gängiges Verfahren repräsentiert diese Gebiete durch einen oder einige wenige prototypische Merkmalsvektoren, die sog. (Klassen-) Repräsentanten, deren optimale Positionen im Raum der Merkmalsvektoren anhand von Beispielen gelernt werden. Jede Klasse $k \in A$ wird also durch eine Menge von n_k Vektoren

$$\text{Repräsentanten der Klasse } k: \quad \{w_{k_i} | i = 1, ..., n_k; \quad w_{k_i} \in R^n\} \tag{3.4}$$

repräsentiert. Die Klassifikation eines beliebigen Datenvektors x erfolgt nach dem Prinzip des kleinsten Abstandes, d. h. x wird demjenigen Repräsentanten zugeordnet, zu dem es den kleinsten Abstand hat, der Klassenlabel dieses Repräsentanten wird ausgegeben.

$$x \in \text{ Klasse } k \qquad \text{falls} \quad \min_i d\,(x, w_{k_i}) \leq \min_i d\,(x, w_{l_i}) \quad \forall\, l \neq k \tag{3.5}$$

Mit der Argumentfunktion arg können wir nun die Funktion $S(x|w)$ aus Gleichung (3.3) explizit schreiben

$$k = S\,(x|w) = \arg \min_l (\min_i d\,(x, w_{l_i})) \tag{3.6}$$

oder

$$k = S\,(x|w) = \arg \min_l \delta_x(l) \tag{3.7}$$

wobei $\delta_x(l) = \min_i d\,(x, w_{l_i})$ der Abstand von x zum nächstgelegenen Repräsentanten der Klasse l ist. $d\,(a, b)$ ist eine beliebige Abstandsfunktion zwischen den beiden Vektoren a, b, z. B. der Euklidische Abstand

$$d\,(a, b) = \sqrt{\sum_{\alpha=1}^{n} (a_\alpha - b_\alpha)^2} \tag{3.8}$$

oder auch ein beliebig anderer abstrakter Abstand.

Dieses Prinzip partitioniert den Datenraum in Zellen von Punkten, die zum jeweiligen Repräsentanten den kleinsten Abstand haben, d. h. es wird eine Voronoi–Parkettierung generiert, s. Kap 4.1, auf der der Klassifikator lebt. Die Klassengrenzen sind folglich durch Polygonzüge approximiert.

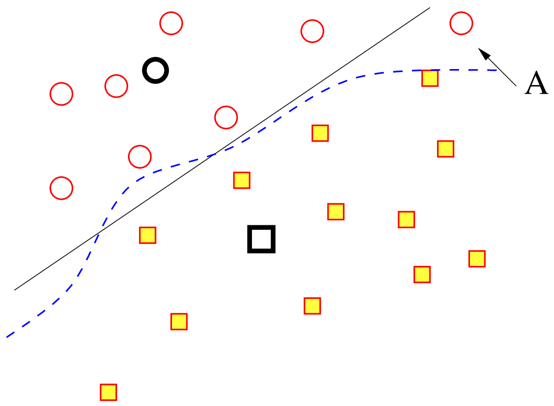

Abbildung 3.2: Datenvektoren zweier verschiedener Klassen mit hypothetischen Repräsentanten. Die eingezeichnete Gerade ist die Grenzlinie zwischen den Voronoizellen der beiden Repräsentanten. Die so induzierte Domänenstruktur der beiden Klassen ist linear und produziert eine Fehlklassifikation des mit A gekennzeichneten Datums. Das kann nur durch Hinzunahme eines weiteren Repräsentanten korrigiert werden.

3.3.2 Erlernen der Klassenrepräsentation – Der lernende Vektorklassifikator

Für eine möglichst variable Grenzziehung zwischen den Gebieten der verschiedenen Klassen erweist es sich als günstig, die Klassen jeweils durch mehrere Prototypvektoren (Repräsentanten) zu identifizieren, auch wenn die Klassendomäne zusammenhängend ist. Wie oben dargestellt, gehört zu jeder Klasse k eine Menge von Repräsentanten w_{k_l} , $l = 1, \ldots, n_k$ und x wird genau dann der Klasse zugeordnet, wenn es **einem** dieser Repräsentanten am nächsten ist.

Ziel des Lernverfahrens ist es, die Repräsentanten w_{k_l} zu lernen. Ein gut geeignetes Verfahren wurde von *Kohonen* [24, 25] angegeben und (etwas unglücklich) Lernender Vektorquantisierer (LVQ) genannt. Wir wollen ihn hier als **lernenden Vektorklassifikator (LVK)** bezeichnen, weil das Ziel eben nicht wie in Kap. 4.1 einfach die Quantisierung (Diskretisierung) des Datenraumes ist, sondern weil eine **Klassifikation** der Datenvektoren erlernt werden soll.

In der Grundversion lautet der Lernalgorithmus des LVK für diskrete Zeiten

$t = 0, 1, \ldots$

$$\Delta w_s(t) = \epsilon_t \left[x - w_s(t) \right] \quad \text{bei richtiger Klassifikation}$$

$$\text{und} \tag{3.9}$$

$$\Delta w_s(t) = -\epsilon_t \left[x - w_s(t) \right] \quad \text{bei falscher Klassifikation}$$

wobei

$$|x - w_s| \leq |x - w_{l_i}| \quad \forall l, i \tag{3.10}$$

Es lernt also immer nur der Repräsentant mit dem geringsten Abstand zum Input-Vektor $x \in \Re^n$. Dieser Repräsentant kann als Sieger im Wettbewerb um die Re-

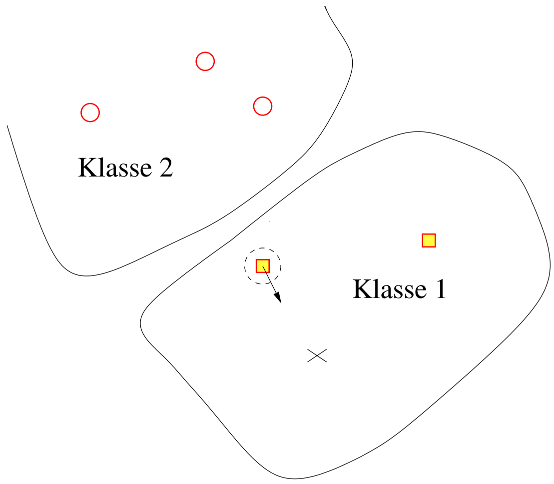

Abbildung 3.3: Erlernen einer Klassenrepräsentation mit dem Algorithmus (3.9) des LVK1: Die Abbildung zeigt die Domänen zweier Klassen mit zwei bzw. drei Klassenrepräsentanten. Die Datenvektoren (nicht dargestellt) liegen beliebig innerhalb ihrer jeweiligen Domänen verteilt. Im Lernschritt des LVK1 wird der zum aktuellen Datenvektor (Kreuz) nächstgelegene Repräsentant ermittelt. Falls dieser wie im Bild klassentreu ist, wird er um den Bruchteil ϵ seiner Entfernung zum Datenvektor auf diesen zubewegt. Im Bild ist $\epsilon = 0.3$

präsentation des aktuellen Inputs betrachtet werden. Zur "Belohnung" darf der Sieger lernen (*Wettbewerbslernen*). Ergebnis des Lernschrittes ist eine Annäherung des Repräsentanten an einen zu seiner Klasse gehörigen Merkmalsvektor,

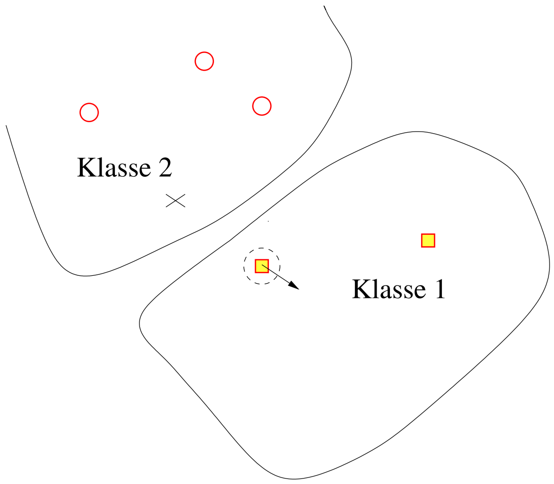

Abbildung 3.4: Erlernen einer Klassenrepräsentation mit dem Algorithmus (3.9) des LVK1: Im Unterschied zur Abbildung 3.3 sind jetzt anliegender Datenvektor (Kreuz) und nächstgelegener Repräsentant klassenfremd. Gemäß 3.9 wird nun der Repräsentant vom Datenvektor wegbewegt.

während ein klassenfremder Prototypvektor w_s vom Datenpunkt x wegbewegt wird, s. Abb. 3.3 und 3.4. Über viele Lernschritte erfolgt somit eine langsame "Diffusion" der Repräsentanten einer bestimmten Klasse in das für diese Klasse relevante Gebiet des Datenraumes. Die Gebiete zwischen den einzelnen Klassen werden durch einen sich aus der *Voronoi-Parkettierung* (s. Kap. 4.1) ergebenden Polygonzug begrenzt.

Wie durch den Index t an der Lernrate ϵ explizit ersichtlich ist die Lernrate wieder geeignet "abzukühlen", s. Kap. **??**.

Probleme bei diesem Lernverfahren sind sog. *dead units*, d. h. Repräsentanten w_{k_l} die auf der Trainingsmenge niemals Sieger werden und demzufolg überflüssig sind (Speicherplatz!) und die Tatsache, daß man die Zahl der Repräsentanten pro Klasse (die die Struktur der Gebietsgrenze bestimmt) von außen vorgeben muß.

3.3.3 Varianten des LVK – der LVK3

Der LVK3, eine Weiterentwicklung nach Kohonen [25] des LVK1, lernt nach folgendem Algorithmus:

1. Bestimme für den aktuellen Datenvektor x die beiden nächstgelegenen Repräsentanten $R_1 : w_i$ und $R_2 : w_j$, wobei

$$|x - w_i| \leq |x - w_j| \tag{3.11}$$

2. Falls genau einer dieser Repräsentanten klassentreu ist und mit $d_l = |x - w_l|$ gilt

$$\min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > \frac{1-u}{1+u} \tag{3.12}$$

dann wird folgender Lernschritt ausgeführt:

$$\Delta w_i = \epsilon\,(x - w_i) \tag{3.13}$$

$$und \tag{3.14}$$

$$\Delta w_j = -\epsilon\,(x - w_j) \tag{3.15}$$

wenn w_i der klassentreue Repräsentant ist, und umgekehrt. Sind beide Repräsentanten klassentreu, so wird für beide ein "normaler" Lernschritt ausgeführt:

$$\Delta w_k = \epsilon\,(x - w_k)\,, k = i, j \tag{3.16}$$

Diese Variante des LVK positioniert die Repräsentanten so, daß die Grenzen zwischen den Gebieten möglichst detailgetreu modelliert werden. Mit diesem Algorithmus wurden eine Reihe von bemerkenswerten Erfolgen z. B. bei der Spracherkennung erzielt. [25] Der Algorithmus steht mit einigen Anwendungsbeispielen als *public domain* Programm zur Verfügung. Eine andere sehr effektive Variante ist der DLVQ (dynamically learning vector quantizer), der die Zahl der Repräsentanten pro Klasse adaptiert, d.h. daß der Algorithmus so lange neue Ressourcen (Repräsentanten) allokiert, bis ein bestimmtes Gütekriterium erfüllt ist, vgl. [55].

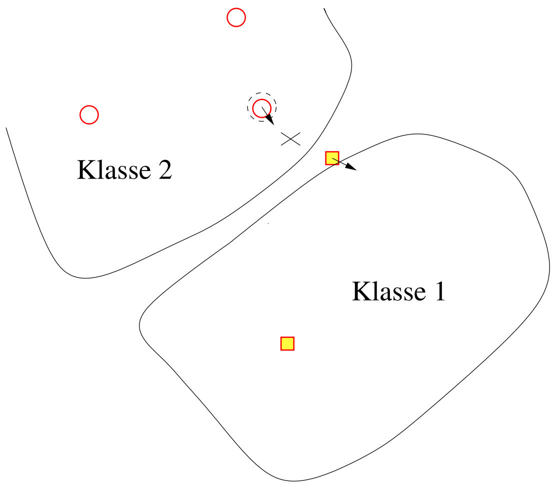

Abbildung 3.5: Der LVK3 lernt, die Repräsentanten bevorzugt entlang der Grenzen ihrer jeweiligen Klasse anzuordnen. Eine Verschiebung eines Repräsentanten findet statt, wenn (i) von den beiden, dem aktuellen Datenvektor nächstgelegenen Repräsentanten (erster und zweiter Sieger), genau einer klassentreu ist. Liegt der Datenvektor noch zusätzlich in einem gewissen Fenster zwischen den beiden Repräsentanten, dann erfolgt eine Verschiebung des klassentreuen bzw. des klassenfremden Repräsentanten auf den Datenvektor zu bzw. von diesem weg. Sind (ii) beide Repräsentanten klassentreu, so führen beide einen normalen Lernschritt wie in Abb. 3.3 aus.

3.4 On-line lernender Bayes-Klassifikator

In der Praxis ist die Klassifikation aufgrund einer Vielzahl störender Einflußfaktoren oft eine Wahrscheinlichkeitsaussage. Die Definition des optimalen Klassifikators ist dann nicht eindeutig. Wir wollen im folgenden die in der statistischen Entscheidungstheorie übliche Definition betrachten und danach ein Lernverfahren zum Erlernen eines in diesem Sinne optimalen Klassifikators angeben.

3.4.1 Definition des optimalen Klassifikators

Sei $P(k \mid x)$ die Wahrscheinlichkeit, daß x zur Klasse k gehört und $S(l \mid k)$ die durch die Zuordnung zu l bei Vorliegen von k verursachten Kosten. Das mit der Entscheidung für die Klasse l verbundene Risiko $R(l \mid x)$ ist für K Klassen

$$R(l \mid x) = \sum_{k=1}^{K} S(l \mid k)\; P(k \mid x)$$

Ziel ist die Minimierung des Risikos.

Falls die Kosten für eine Fehlklassifikation alle gleich groß sind

$$S(k \mid l) = 1 - \delta_{kl} = \left\{ \begin{array}{ccc} 0 & f\ddot{u}r & k = l \\ 1 & sonst & \end{array} \right\}$$

(δ_{kl} *Kroneckersymbol*), folgt wegen

$$R(l \mid x) = \sum_{k \neq l} P(k \mid x) = 1 - P(l \mid x)$$

als Regel für die optimale Klassifikation, daß x der Klasse mit der höchsten *a posteriori* (s.u.) Wahrscheinlichkeit zugeordnet werden muß:

$$x \in Klasse\, l \qquad falls \qquad P(l \mid x) = \max_k P(k \mid x)$$

Formal können wir $\kappa(x)$ definieren durch

$$\kappa(x) = \arg_{(l)} \max_l P(l \mid x)$$

wobei $\arg_{(l)}$ die Argumentfunktion bezüglich l bezeichnet, die hier also aus allen l mit $(1 \leq l \leq K)$ dasjenige heraussucht, für das $P(l \mid x)$ sein Maximum annimmt.

Wegen der allgemeinen Unsicherheit der Klassifikation in diesem Wahrscheinlichkeitsszenario sollte konsistenterweise die Möglichkeit einer Rückweisungsentscheidung bestehen. Mit Einführung einer Rückweisungsklasse r gilt bei wieder gleichen Kosten für jede Fehlklassifikation jetzt:

$$S\left(l \mid k\right) = \begin{cases} 0 \quad f\ddot{u}r\ l = k \quad (richtige\, Klassifikation) \\ c_r \quad f\ddot{u}r \quad l = r \quad (R\ddot{u}ckweisung) \\ c_f \quad f\ddot{u}r \quad l \neq k, r \quad (Fehlklassifikation) \end{cases}$$

wobei c_r die Kosten für eine nichtgetroffene Entscheidung mißt und c_f entsprechend die für die falschen Klassifikationen. Damit können wir für die **optimale Entscheidungsregel** z. B. folgenden Ansatz wählen:

$$\begin{array}{lll} x \in Klasse\, k & wenn & P\left(k \mid x\right) = \max_l P\left(l \mid x\right) \\ & und & P\left(k \mid x\right) \geq 1 - \frac{c_r}{c_f} \end{array}$$

$$R\ddot{u}ckweisung \quad wenn \quad P\left(k \mid x\right) < 1 - \frac{c_r}{c_f}$$

Die Klassifikation erfolgt also nur, wenn die *a posteriori* Wahrscheinlichkeit einen gewissen Schwellwert überschreitet, d. h. bei $c_r \geq c_f$ erfolgt niemals eine Rückweisung. Wie erwartet hängt die Entscheidungsfindung nur vom Verhältnis der Kosten für Rückweisung und Fehlklassifikation ab. Die Regel generiert eine eindeutige Aufteilung des gesamten Input-Raumes in Klassen- und Rückweisungsgebiete.

3.4.2 Bayessche Regel und Rückführung auf die a priori Wahrscheinlichkeitsverteilung

Soweit die Theorie, die davon ausgehen darf, daß die *a posteriori* Wahrscheinlichkeiten $P(k|x)$ bekannt sind. Die Bestimmung dieser Wahrscheinlichkeiten ist das zentrale Problem einer praktischen Anwendung für unbekannte Verteilungen, vor allem dann, wenn die Menge der Datenvektoren x (Stichprobenumfang) beschränkt ist. Die *a posteriori* Wahrscheinlichkeiten können aus den *a priori* Wahrscheinlichkeiten mittels der Bayesschen Regel bestimmt werden.

Seien $P\left(k\right)$ die *a priori* Wahrscheinlichkeiten für das Vorliegen der Klasse k und $P\left(x \mid k\right)$ die bedingte Wahrscheinlichkeit, den Datenvektor x bei Vorliegen der Klasse k zu beobachten[1]. **Beispiel**: Jeder Datenvektor $x = (x_1, \ldots, x_n)$ sei der Merkmalsvektor eines Patienten einer bestimmten medizinischen Einrichtung, wobei die x_i etwa die physiologischen Werte (Blutdruck, Körpertemperatur, ...) des jeweiligen Patienten darstellen. Die Klassen $k \in A$ sind dann die verschiedenen Krankheiten, an denen die Patienten leiden und $P(k)$ ihre Auftretenswahrscheinlichkeit, näherungsweise gegeben durch die relative Häufigkeit, mit der diese Krankheit bei den Patienten diagnostiziert wurde. [2] Die $P(k)$ sind also a priori bekannt. Die bedingte Wahrscheinlichkeit $P(x \mid k)$ gibt dann an, mit welcher Wahrscheinlichkeit ein bestimmter Merkmalsvektor x bei einem Patienten beobachtet wird, der an der bestimmten Krankheit k erkrankt ist. Diese bedingten Wahrscheinlichkeiten sind als relative Häufigkeiten aus der Datenmenge zu entnehmen und damit a priori bekannt.

Mit dem Satz von der totalen Wahrscheinlichkeit

$$P(x) = \sum_{l=1}^{K} P(x \mid l)\ P(l)$$

gilt die Bayes-Beziehung

$$P(k \mid x) = \frac{P(x \mid k)\, P(k)}{P(x)} \tag{3.17}$$

die in der Form $P(k \mid x)\, P(x) = P(x \mid k)\, P(k)$ geschrieben plausibel ist, weil die linke wie die rechte Seite jeweils die Wahrscheinlichkeit für das Vorliegen eines k, x - Paares zum Ausdruck bringt. Mit der Bayesschen Regel kann also die *a posteriori* zur *a priori* Wahrscheinlichkeit in Beziehung gesetzt werden.

Mit (3.17) lautet die Klassifikationsregel ohne Rückweisung und bei wieder gleichen Kosten für die Fehlklassifikationen, d. h. $S(l \mid k) = 1 - \delta_{l,k}$

$$x \in \text{Klasse } k, \qquad \text{falls} \qquad P(x \mid k)\, P(k) = \max_l P(x \mid l)\, P(l) \tag{3.18}$$

[1] Wir nehmen den Vektor x als diskretisiert an, so daß wir es mit einer endlichen Menge von Inputvektoren x zu tun haben.

[2] Wir betrachten die Menge der Patienten als eine Stichprobe endlichen Umfangs N. Diese relative Häufigkeit ist eine Schätzung der Wahrscheinlichkeit selbst, also eine Zufallszahl, die um so besser mit der Wahrscheinlichkeit übereinstimmt, ja größer die Stichprobe ist.

Mit Rückweisung gilt entsprechend

$$x \in \text{Klasse } k, \qquad \text{falls} \qquad P(x \mid k)\, P(k) = \max_l P(x \mid l)\, P(l)$$

und

$$P(x \mid k)\, P(k) \geq \left(1 - \tfrac{c_f}{c_r}\right) \sum_l P(x \mid l)\, P(l)$$

3.4.3 Parameterlernen der a priori Wahrscheinlichkeit

Die *a priori* Wahrscheinlichkeit ist allerdings in den meisten Fällen ebenfalls unbekannt. Das Ziel eines lernfähigen Klassifikators liegt in der inkrementellen Bestimmung dieser Wahrscheinlichkeiten aus der gegebenen Datenmenge, preferentiell im *on-line* Betrieb. Das läuft auf ein Parameterlernverfahren hinaus, falls für diese Wahrscheinlichkeiten ein Ansatz mit unbekannten Parametern gemacht werden darf, die dann gelernt werden.

Ein naheliegender Ansatz für $p(x \mid l)$ ist eine Gaussverteilung. Zur Demonstration des Verfahrens betrachten wir den eindimensionalen Fall $x \in \mathcal{R}^1$ und setzen

$$p(x \mid l) = \frac{1}{\sqrt{2\pi\sigma_l}}\, e^{-\frac{\left(x - m^{(l)}\right)^2}{2\sigma^{(l)}}} \tag{3.19}$$

mit den unbekannten Parametern $m^{(l)}$ und $\sigma^{(l)}$, die Mittelwert und Varianz der Verteilung darstellen, d. h.

$$m^{(l)} = \langle x \rangle^{(l)}, \sigma^{(l)} \qquad = \left\langle \left(x - m^{(l)}\right)^2 \right\rangle^{(l)}$$

wobei $\langle ... \rangle^{(l)}$ die Mittelung über die zur Klasse l gehörigen x bezeichnet

$$\langle g(x) \rangle^{(l)} = \int dx\, g(x)\; p(x \mid l)$$

Diese Mittelung kann im Sinne eines Lernalgorithmus inkrementell im *on line* Betrieb durchgeführt werden. Allgemein ist der Mittelwert

$$m = \frac{1}{N} \sum_{t=1}^{N} x_i$$

mit N Größen $x_t,\ t = 1, \ldots, N$ als Endwert $m = m(N)$ einer Zeitreihe $m(t)$ gegeben, die durch den Algorithmus

$$\begin{aligned} \Delta m(t-1) &= \tfrac{1}{t}\,(x_t - m(t-1)), \qquad t = 1, 2, \ldots, N \\ \textit{oder} \quad m(t) &= \tfrac{t-1}{t} m(t-1) + \tfrac{1}{t} x_t \end{aligned} \tag{3.20}$$

erzeugt wird, wobei $\Delta m(t-1) = m(t) - m(t-1)$ und $m(0) = 0$.

Bemerkung*:* $m(t)$ konvergiert auch gegen den Mittelwert m, für $t \to \infty$, wenn zu jedem Zeitpunkt t das x_t als ein zufällig aber gleichwahrscheinlich aus der Menge der x ausgewähltes Element verstanden wird (sonst konvergiert $m(t)$ gegen das gewichtetes Mittel).

Mit dieser Vorschrift können die Parameter der Verteilung (3.19) mittels

$$\Delta m^{(l)} = \frac{1}{t}\,\left(x - m^{(l)}\right)$$

und

$$\Delta\sigma^{(l)} = \frac{1}{t}\,\left(\left(x - m^{(l)}\right)^2 - \sigma^{(l)}\right)$$

gelernt werden, wobei $m^{(l)} = m^{(l)}(t-1)\,, \quad \sigma^{(l)} = \sigma^{(l)}(t-1)$. Für die mehrdimensionale Normalverteilung (Index l hier weggelassen)

$$p(x \mid l) = \frac{1}{\sqrt{(2\pi)^n\,|\hat{\sigma}|}}\, e^{-\frac{1}{2}\left[(x-a)^T\,\hat{\sigma}^{-1}(x-a)\right]} \tag{3.21}$$

mit $\left|\hat{\sigma}^{(l)}\right|$ - Determinante der Kovarianzmatrix $\hat{\sigma}^{(l)}$

$$\sigma_{ij}^{(l)} = \left\langle \left(x_i - m_i^{(l)}\right)\left(x_j - m_j^{(l)}\right)\right\rangle^{(l)}$$

und u^T - Zeilenvektor gilt analog

$$\Delta m^{(l)} = \frac{1}{t}\,\left(x - m^{(l)}\right)$$

und

$$\Delta\hat{\sigma}^{(l)} = \frac{1}{t}\,\left[\left(x - m^{(l)}\right)\left(x - m^{(l)}\right)^T - \hat{\sigma}^{(l)}\right]$$

(*Beachte:* $u\,v^T$ ist das äußere Produkt der Vektoren $\vec{u}$,und v und damit eine Matrix).

Die Klassenzuweisung wird vom Lehrer während des Lernvorganges geleistet, d. h. für jedes x kann die Klasse k als gegeben angenommen werden, so daß die obige Vorschrift bei Vorliegen hinreichend vieler Trainingsbeispiele gegen die richtigen Parameter der *a priori* Wahrscheinlichkeitsverteilung der einzelnen Klassen konvergiert, vorausgesetzt, der Ansatz einer Gaussverteilung ist gerechtfertigt.

Kapitel 4

Lernen ohne Lehrer: Clusterung und Quantisierung von Daten

Die in Kapitel 2. und 3. dargestellten Lernverfahren beruhen auf dem Vorhandensein eines Lehrers, der die Solloutputs für den Lerner bei gegebenem Input vorgibt. Der Lerner erlernt also eine Funktion. Aber auch ohne einen Lehrer kann man eine Menge lernen! Alltäglich lernen wir beispielsweise, die auf uns einströmenden Datenmengen zu strukturieren, beispielsweise anhand von Merkmalen die Daten in Gruppen einzuordnen. Im Unterschied zum im Kap. 3 besprochenen lernenden Klassifikator, gelingt uns das Schaffen von Ordnung auch ohne einen Lehrer. Stattdessen können wir offensichtlich selbst Relationen zwischen den Daten erkennen, die sich als Kriterium für die Eingruppierung der Daten eignen. Im folgenden Abschnitt 4.1 beschäftigen wir uns zunächst mit dem Erlernen einer problemangepaßten Diskretisiertung der Daten und danach im Abschnitt 4.2 mit nichtüberwachten Lernverfahren zur Clusterung von Daten.

4.1 Adaptive Vektorquantisierer

VQ werden allgemein zur **Datenreduktion** oder als **Kodierer** eingesetzt, d. h. zur problemangepaßten Diskretisierung (=Quantisierung) oder Partitionierung von hochdimensionalen Datenräumen, für die bei der Kodierung ein gewisser Informationsverlust hingenommen werden kann. (Unterschied zur **Datenkompression** bei der eine vollständige Rekonstruktion der Ausgangsdaten möglich sein soll). Einen guten Überblick über die einschlägigen Methoden findet man in [16]. Anwendungen in der Datenkompression sind in [45] zusammengestellt. Ziel ist, Datenvektoren x in Gruppen zusammenzufassen, die jeweils durch einen einzigen Vektor, ihren Codebuchvektor (CV) oder Repräsentanten, repräsentiert werden. Das entspricht der Aufteilung der Menge der Datenvektoren in Teilmengen entsprechend den Gruppen $i = 1, 2, \ldots, N$, wobei allen $x \in$ Gruppe i der Repräsentant w_i nach dem Prinzip des kleinsten Abstandes zugeordnet wird (**Kodierungsschritt**). Sei im $\mathcal{R}^n$ ein Abstand $d\,(x, y)$ zwischen zwei beliebigen Vektoren x, y definiert. Das kann z. B. der Euklidische Abstand

$$d\,(x, y) = \sqrt{\sum_i \left(x_i - y_i\right)^2}$$

sein. Der Kodierungsschritt ist

$$i = \arg\min_j d(x, w_j) \tag{4.1}$$

Geometrisch läßt sich diese Gruppierung als die Voronoiparkettierung des Inputraumes interpretieren. Darunter verstehen wir die Zerlegung des Raumes in die Menge $V = \{V_i \mid i = 1, ..., n\}$ der Voronoizellen der einzelnen Vektoren w_i, so daß

$$\mathcal{R}^n = V_1 \cup V_2 \cup ... \cup V_N$$

Die zum Vektor w_i gehörige Voronoizelle V_i ist definiert als das Gebiet aller x, die zu w_i einen kleineren (oder höchstens den gleichen) Abstand als zu allen restlichen Kodebuchvektoren haben

$$V_i = \{x \mid d\,(x, w_i) \leq d\,(x, w_j) \quad \forall j \neq i\} \tag{4.2}$$

Die Grenzen der einzelnen Voronoizellen sind folglich Hyperebenen, die die Verbindungslinien zwischen benachbarten Repräsentanten auf halbem Wege senkrecht schneiden.

Der Vorteil der Vektorquantisierung wird offensichtlich wenn wir ein Informationsübertragungsystem betrachten, bei dem ein gewisser Verlust an Übertragungstreue in Kauf genommen werden kann. Als Information wird dann nur der

Index i übertragen und im Decodierungsschritt dann w_i ausgegeben [1]

$$\text{Input} \quad x \quad \Rightarrow \quad \boxed{\begin{array}{c}\text{Kodierer}\\ x \;\rightarrow\; i\end{array}} \quad \rightarrow \quad i \quad \rightarrow \quad \boxed{\begin{array}{c}\text{Dekodierer}\\ i \rightarrow\; w_i\end{array}} \quad \Rightarrow w_i \quad \text{Output} \tag{4.3}$$

Der Kodierer ist also durch die Funktion $i: \; x \rightarrow i(x)$ gegeben, wobei (s. 4.1)

$$i = \arg\min_j d(x, w_j) \tag{4.4}$$

d. h. die Datenvektoren zum Repräsentaten w_i liegen alle in dessen Voronoizelle.

Für das Maschinelle Lernen kann der Vektorquantisierer vor allem als Instrument zur problemangepaßten Diskretisierung von Datenräumen eingesetzt werden, d. h. als eine Vorverarbeitungseinheit, die reellwertige und eventuell verrauschte Daten in diskrete Kategorien einordnet, die dann von regelbasierten Systemen des Maschinellen Lernens unmittelbar weiterverarbeitet werden können.

Frage ist nun, wie eine möglichst effektive Vektorquantisierung aussieht. Dazu gibt es im wesentlichen zwei Kriterien – die übertragene Transinformation und den Rekonstruktionsfehler.

Die Transinformation als Gütemaß

Der VQ kann als ein Kodierer betrachtet werden, der Inputs x Gruppenlabel $l, \quad l = 1, \ldots, L$ zuordnet. Wir stellen nun zunächst die Frage nach der im Sinne der Informationstheorie optimalen Form der Kodierung. Diese Frage kann ganz allgemein ohne Bezug auf eine spezielle Realisierung eines VQ beantwortet werden. Das entsprechende Qualitätsmaß ist die Transinformation, die die durch den Kodierer übertragene Information mißt. Wenn die Inputvektoren $x \in \Re^n$ stochastische Vektoren mit einer Verteilung $p(x)$ sind, so ist die Inputinformation

$$I^{(in)} = -\sum_x p(x) \ln p(x)$$

Wegen der Zuordnung $x \rightarrow w_l$ bzw. $x \rightarrow$ Klassenlabel $l, \quad l = 1, \ldots, L$ definiert die Verteilung $p(x)$ auch eine Outputverteilung $P(l)$ mit Information

$$I^{(out)} = -\sum_{l=1}^{L} P(l) \ln P(l)$$

[1] Die Vektoren w_i brauchen nur einmalig übertragen zu werden.

Die bedingte Outputinformation $I^{(out|x)} = -\sum_{l=1}^{L} P(l \mid x) \ln P(l \mid x)$ bzw. ihr über die Inputs gemittelter Wert

$$I^{(out|in)} = \left\langle I^{(out|x)} \right\rangle = \sum_x p(x) \sum_{l=1}^{L} P(l \mid x) \ln P(l \mid x)$$

ist ein Maß für die Outputinformation bei festgehaltenem Input x. Bei einer wie im betrachteten Fall deterministischen Zuordnung (sog. harte Kodierung) ist $P(l \mid x)$ entweder Null oder Eins und damit ist die bedingte Information gleich Null, wie es sein muß, denn die bedingte Information mißt die durch das Rauschen des Kodierers erzeugte "falsche" Information.. Die am Ausgang anliegende nutzbare oder unverfälschte Information ist gerade die Transinformation, d. h. die durch den Kodierer übertragene Information

$$I^{(trans)} = I^{(out)} - I^{(out|in)}$$

Im betrachteten Fall der harten Kodierung gilt also $I^{(trans)} = I^{(out)}$ und der optimale Kodierer ist durch $I^{(out)} = \max$ gegeben, was durch

$$P(l) = \frac{1}{L}$$

erfüllt wird, d., h. **der Kodierer arbeitet optimal, wenn alle Repräsentanten** im Mittel über die Inputs **gleich oft angesprochen werden**, bzw. wenn alle Zellen im Mittel gleich viele Datenpunkte enthalten. Für $p(x) = const$ heißt das, daß alle Domänen (Voronoizellen) gleich groß sind.

Der Rekonstruktionsfehler

Der Rekonstruktionsfehler drückt den durch die Ersetzung der x durch die w gemachten Fehler in Termen der mittleren quadratischen Abweichung aus

$$E(\mathbf{w}) = E(\{w_i \mid i = 1, \ldots, N\}) = \langle E(\mathbf{w}, x) \rangle = \sum_x p(x)\; E(\mathbf{w}, x)$$

mit

$$E(\mathbf{w}, x) = d\left(w_{i(x)}, x\right)^2$$

wobei $i(x)$ der Index des Siegerrepräsentanten zu x ist, d. h. $d\left(w_{i(x)}, x\right) = \min_l d(w_l, x)$. Speziell gilt für den Fall des Euklidischen Abstandes

$$E(\mathbf{w}, x) = \sum_\alpha (x_\alpha - w_{i(x)\alpha})^2 \tag{4.5}$$

Bei gleichverteilten Daten ist der Rekonstruktionsfehler am kleinsten, wenn alle Zellen gleich groß sind und eine dichteste Kugelpackung des Raumes realisieren. Das enspricht auch dem Maximum der Transinformation. Bei inhomogenen Datenverteilungen liefern beide Maße aber durchaus unterschiedliche optimale VQ.

4.1.1 Der Lernalgorithmus

Die Bestimmung des besten Satzes von Prototypvektoren erfordert die Minimierung von E bzw. die Maximierung von I^{trans} als Funktion der w. Das kann durch ein Gradientenfolgeverfahren erreicht werden. Für die Minimierung des Rekonstruktionsfehlers E wird dabei jeder Vektor w_i schrittweise gemäß

$$\Delta w_i = -\epsilon \frac{\partial E}{\partial w_i} \tag{4.6}$$

verändert.

Dazu muß aber der mittlere Fehler E bekannt sein, d. h. man muß schon alle Daten vorliegen haben. Ein inkrementelles Verfahren, bei dem mit jedem neu einkommenden Datenvektror x eine "infinitesimal" kleine Veränderung der Codebuchvektoren vorgenommen wird, ergibt sich durch die stochastische Approximation des Gradientenabstieges (4.6). Das wird mit folgendem Algorithmus realisiert: Bei Eingabe von x bestimmen wir zunächst den zuständigen (*best match*) Repräsentanten w_i und führen für diesen den Lernschritt gemäß

$$\Delta w_i = -\epsilon \, (w_i - x) \qquad i = i\,(x) \tag{4.7}$$

aus, wobei die Fehlerfunktion (4.5) zugrundegelegt wurde[31]. Der Lernparameter ϵ ist im Sinne eines simulated annealing wieder geeignet abzukühlen.

Der Algorithmus erzeugt einen zeitdiskreten stochastischen Prozeß $w\,(t)$. Im stationären Zustand gilt $\langle \Delta w \rangle = 0$ und folglich

$$\langle w_l \rangle^{(l)} = \langle x \rangle^{(l)}$$

wobei die $\langle x \rangle^{(l)}$ die Mittelung über die Domäne des Repräsentanten w_l ist. Der Repräsentant ist also bei sorgfältigem "Einfrieren" gleich dem Mittelwert der in seiner Domäne gelegenen Inputvektoren x.

Das Verfahren heißt auch Wettbewerbslernen, weil die Repräsentanten sozusagen im Wettbewerb um die beste Repräsentation der Inputs stehen.

4.1.2 Topografische Vektorquantisierer

Bei den bisher dargestellten und auch bei den meisten der in der Praxis gebräuchlichen Verfahren spiegelt sich die Metrik des Datenraumes in den Repräsentanten nicht wieder. Im Inputraum benachbarte Datenvektoren werden im allgemeinen nicht auf benachbarte Indizes abgebildet, die Indexfunktion generiert keine nachbarschaftserhaltende (topgrafische) Abbildung. Für viele Anwendungen ist das ein echter Nachteil. Als Alternative ist aus der Neuroinformatik der Algorithmus von Kohonen bekannt, der unter geeigneten Voraussetzungen einen topografischen Vektorquantisierer generiert, vgl. [25, 42, 55, 6].

4.2 Clusterung von Daten

Daten bilden oft in natürlicher Weise Gruppen. Das ist beispielsweise der Fall, wenn die Daten gestörte Versionen einer kleinen Anzahl von Prototypen sind. Beispiele finden sich wieder (i) in der Spracherkennung, wo die Prototypen die Phoneme in der (wie auch immer definierten) Standardaussprache sind oder (ii) in der Bildverarbeitung, wo verschiedene Bilder ein und derselben Person eine Gruppe bilden. Eine besondere Schwierigkeit in der Clusterung von Daten liegt dann vor, wenn es keine äußeren Hinweise für die Zuordnung der Daten zu den einzelnen Clustern gibt. In vielen Fällen ist auch die Zahl der Cluster unbekannt. Dann kann die Clusterung nur aus den statistischen Eigenschaften der Daten gewonnen werden, was fast immer ein nichttriviales und oft nicht eindeutig lösbares Problem darstellt.

Wir wollen zunächst Clusterungsverfahren darstellen, die für metrische Daten besonders gut geeignet sind. Diese leiten sich von geometrischen Vorstellungen über die Verteilung der Datenvektoren im Raum ab und setzen damit stillschweigend die Gültigkeit der Dreiecksrelation voraus, vgl. Abschnitt 5.

Nichtmetrische Daten erfordern im allgemeinen andere Verfahren, die sich mehr in Richtung kombinatorische Optimierung orientieren. Ein Beispiel soll in Abschnitt 4.2.4 gegeben werden. Eine andere Möglichkeit ist die Einbettung der Daten in einen metrischen Raum vgl. 4.2.4.

4.2.1 Problemstellung

Ziel der Clusterung ist die Einordnung der Menge aller Datenvektoren $x_k \in D$, $x_k \in R^n$ in eine Anzahl K von Clustern, wobei sich die Datenvektoren eines Clusters grob gesprochen durch eine besonders starke Verwandtschaft auszeichnen. Formal definieren wir eine Zugehörigkeitsfunktion

$$\chi_v(x) = \begin{cases} 1 & \text{falls} \quad x \in \text{Cluster } v \\ \\ 0 & \text{falls} \quad x \notin \text{Cluster } v \end{cases} \tag{4.8}$$

die die Zugehörigkeit des Datums $x \in R^n$ zum Cluster v bestimmt. Genausogut können wir mit der Indexfunktion $i(x)$ operieren, zum Datum x den Index v des zugehörigen Clusters ausgibt.

Falls jeder Cluster durch einen Prototypen oder Repräsentaten w_v dargestellt werden kann, dann ist die Indexfunktion in natürlicher Weise durch das Prinzip des kleinsten Abstandes gegeben, d. h. wie im Fall des Klassifikators gilt

$$v = \arg\min_v d(x, w_v) \tag{4.9}$$

4.2.2 Der *k–means* Algorithmus

Eines der bekanntesten Verfahren zur Bestimmung der Prototypen ist der *k–means* Algorithmus. Dies ist ein iteratives Verfahren, das von einer geeigneten Initialisierung ausgehend die Repräsentanten (Prototypen) schrittweise verbessert. Ziel ist dabei, daß die Prototypen mit den Zentroiden (Schwerpunkten) des Clusters den sie definieren, zusammenfallen. Die Clusterzugehörigkeit ist durch den nächstgelegenen Repräsentanten, der Schwerpunkt eines Clusters v durch

$$z_v = \frac{1}{Z} \sum_{x \in D} \chi_v(x)\, x = \frac{1}{Z} \sum_{x \in D_v} x \tag{4.10}$$

bestimmt, wobei D_v die Menge der Datenvektoren, die zum Cluster v gehören und $Z = \sum_{x \in D} \chi_v(x)$ den Normierungsfaktor, also die Zahl $\#D_v$ der zum Cluster v gehörigen Datenvektoren bezeichnet.

Ziel des Algorithmus ist die schrittweise Verbesserung der Übereinstimmung zwischen Schwerpunkt und Repräsentant eines jeden Clusters[2]. Dazu werden in jedem Schritt der Iteration die Schwerpunkte aller Cluster neu berechnet. Die so ermittelten Schwerpunkte bilden die neuen Prototypen. Konvergenz ist erreicht, wenn sich die Schwerpunkte nicht mehr verschieben.

4.2.3 Adaptive Vektorquantisierer zur Clusterung

Wie in Kap. 4.1 ausgeführt, generiert auch der adaptive VQ eine Konfiguration, in der die Repräsentaten im Zentrum ihrer Domäne liegen und leistet damit im Prinzip das gleiche wie der *k-means* Algorithmus. Man startet mit einer geeignet erzeugten Menge von Prototypvektoren $w_k \in R^n, k = 1, 2, \ldots$ und verschiebt in jedem Lernschritt den Siegervektor gemäß

$$\Delta w_s = \epsilon(x - w_s) \tag{4.11}$$

wobei $x \in R^n$ ein beliebiges Datum ist.

Die Clusterung der Daten kann dann anschließend durch Inspektion der Prototypvektoren erhalten werden. Sind die Cluster gut getrennt und lassen sich diese bei bekannter Anzahl durch einen Prototyp je Cluster modellieren, wird dieses Verfahren im allgemeinen die hypthetischen Prototypvektoren w_k in die Schwerpunkte der Cluster verteilen.

Der Unterschied zum *k-means* Algorithmus liegt hier wieder in dem *on-line* Charakter des Lernens in (4.11) und auch in Verlauf und Ergebnis der Clusterung. Sind nämlich die Daten zahlreich und die Clusterstruktur nichttrivial, dann ist das Auffinden der besten Clusterung ein komplexes nichtlineares Optimierungsproblem für das die Algorithmen meist nur eine suboptimale Lösung finden. Das Ergebnis des *k-means* Algorithmus hängt dann stark von der Initialisierung und das des adaptiven VQs von der Abkühlungsstrategie ab.

4.2.4 Nichtmetrische Daten

Für nichtmetrische Daten, für die nur die paarweisen Abstände d_{kl} bekannt sind, kann der Ansatz (4.10) Verwendung finden, wenn die Daten vorher in einen me-

[2]Wir nehmen der Einfachkeit halber an, daß jeder Cluster durch genau einen Prototypvektor repräsentiert wird.

trischen Raum eingebettet wurden, s. unten. Ansonsten bietet sich der Weg über eine Kostenfunktion an, deren Minimierung, wie unten dargestellt, die richtige Clusterung liefert.

Einbettung in metrische Räume

Grundprinzip der Einbettung ist die nachbarschaftserhaltende Abbildung der Daten in einen Raum, in dem eine (Euklidische) Metrik definiert ist. Ein modernes Verfahren ist das *multidimensional scaling* vgl. [4, 20]. Dazu wird $\forall k = 1, ..., N$ dem k–ten Element aus der Menge der Daten D ein fiktiver Datenvektor $x_k \in R^n$ zugeordnet, wobei $n < N$. Das definiert eine Abbildung der Menge D in den n–dimensionalen Euklidischen Raum, so daß also x_k das Bild (die Projektion) des k–ten Elementes der Menge D ist. Von D braucht nur die Matrix **d** bekannt zu sein, deren Elemente d_{ik} den (verallgemeinerten) Abstand zwischen den Elementen i und k von D quantifizieren. Die Abbildung (Einbettung) soll nun möglichst abstandstreu sein, d. h. die Abstände d_{ik} werden möglichst genau durch die (Euklidischen) Abstände im Zielraum R^n reproduziert.

Die beste Einbettung minimiert folglich die sog. Einbettungskosten

$$H = \frac{1}{2} \sum_{i,k=1}^{N} \left(d\left(x_i, x_k\right) - d_{ik}\right)^2 \tag{4.12}$$

(O. B. d. A. können wir annehmen, daß der Ursprung des KS mit dem Schwerpunkt $\sum x_i/N$ zusammenfällt) Die Minimierung von H ist ein nicht-konvexes Optimierungsproblem, d. h. daß im allgemeinen sehr viele lokale Minima existieren. (Die früher besprochene Methode des simulated annealing ist deshalb eine beliebte Methode zur Lösung des Einbeittungsproblems.)

Kostenfunktion

Falls man den Umweg {uber die Einbettung in einen metrischen Raum nicht gehen möchte, bietet sich die Formulierung als Optimierungsproblem über eine Kostenfunktion an. Eine geeignete Funktion ist

$$E_C\left(\chi\right) = \sum_{v=1}^{N_C} \frac{1}{Z_v N} \sum_{k=1}^{N} \sum_{l=1}^{N} \chi_{kv} \chi_{lv} d_{kl} \tag{4.13}$$

wobei $N = \#D$ (Gesamtzahl der Datenvektoren), Z_v die Zahl der Datenvektoren im Cluster v und

$$\chi_{kv} = \begin{cases} 1 \quad \text{falls} \quad \text{Datum } k \in \text{Cluster } v \\ \\ 0 \qquad\qquad\qquad\qquad \text{sonst} \end{cases} \tag{4.14}$$

legt die (zu bestimmenden) Clusterzugehörigkeiten fest. Die im Sinne dieser Kostenfunktion optimale Clusterung ergibt sich durch die Minimierung von E_C nach den Clusterzugehörigkeiten $\{\chi_{lv}\}$. Der Hauptterm in der Kostenfunktion (4.13) ist klein, wenn in jedem Cluster alle Daten paarweise einander besonders ähnlich sind. Andererseits bestraft der Vorfaktor das Entstehen kleiner Cluster, er befördert also eine möglichst gleichmäßige Verteilung der Daten auf die Cluster.

Minimierung der Kostenfunktion

Die im Sinne der Kostenfunktion optimale Clusterung ergibt sich aus der Minimierung der Kostenfunktion (4.13) durch Variation der χ_{kv}. Da diese binärwertig sind, ist das ein diskretes Optimierungsproblem, das wieder nichtkonvex, d. h. mit lokalen Minima "gesegnet" ist. Das diskrete Optimierungsproblem kann nicht durch Gradientenabstieg gelöst werden. Besonders geeignet sind die später zu besprechenden genetischen Algorithmen (s. Kapitel 8) oder die dem *simulated annealing* eng verwandten Monte Carlo Methoden. Ziel ist der schrittweise Abstieg auf der Kostenlandschaft durch tentative Veränderungen der Clusterzugehörigkeiten, wobei wieder ein stochastisches Element (Rauschen) das Steckenbleiben in den Nebenminima verhindert. Die Stärke der stochastischen Einflüsse regelt eine fiktive Temperatur T, die im Laufe des Verfahrens abgesenkt wird.

Der Algorithmus ist durch den folgenden Pseudocode gegeben:

1. Initialisiere die $\chi_{kv} \in \{0,1\}$ $\forall k, l = 1, \ldots, N undv \in 1, \ldots, N_C$ geeignet, z. B. $\chi_{kl} = random \in \{0,1\}$.

2. Bestimme einen Interimssatz $\chi' = \{\chi'_{kl}\}$ durch Modifikation des Satzes $\chi = \{\chi_{kl}\}$ der Clusterzugehörigkeiten. (Das kann auf verschiedene Weise geschehen, z. B. durch Zufallsauswahl eines Paares (i, j) und "Flippen" der Clusterzugehörigkeit, d. h. $\chi_{ij} = 1 \Rightarrow \chi'_{ij} = 0$ bzw. $\chi_{ij} = 0 \Rightarrow \chi'_{ij} = 1$. Alle anderen Clusterzugehörigkeiten bleiben erhalten).

3. Bestimme die damit erzielte Änderung $\Delta E_C = E_C(\chi') - E_C(\chi)$ der Kostenfunktion E_C

4. Bestimme die Akzeptanzwahrscheinlichkeit für diese Änderung

$$P_{akz} = \frac{1}{1 + e^{\Delta E_C / T}}$$

5. Führe mit mit Wahrscheinlichkeit P_{akz} die Ersetzung $\chi \Leftarrow \chi'$ durch.

6. GOTO 2

Kapitel 5

Konzeptlernen

5.1 Einführung

Bisher behandelte Methoden des Lernens aus Beispielen beruhten hauptsächlich auf der Variation reellwertiger Parameter. Diese sind besonders für metrische Daten geeignet. Typische Anwendungen finden diese Verfahrren bei der Realisierung lernfähiger technischer Systeme (Spracherkennung, Controlling, Robotik, adaptive Filter zur Signalverarbeitung, u.a.) Die Verfahren arbeiten weitgehend modellfrei, d. h. sie setzen kein Verständnis der Daten voraus, die sie verarbeiten. Man bewegt sich dabei also auf der präkognitiven Ebene.

Auf der kognitiven Ebene arbeiten wir im allgemeinen mit einer symbolischen Repräsentation der Welt, wobei die Funktion des Lerners durch ein Regelsystem gegeben ist, das auch eine **Erklärung** des erlernten Verhaltens liefert. Dies ist insbesondere beim Erlernen eines Konzeptes der Fall.

Wir betrachten das Kozept (von Lat. *conceptus* = Gedanke, Begriff) als ein Ordnungselement, das es erlaubt, in sinnvoller Weise die Objekte der Welt in Klassen einzuteilen (positive und negative Instanzen des Konzeptes). Die Darstellung der Konzepte kann dabei sehr unterschiedlich sein, es kann durch einen Satz von Regeln beschrieben sein oder durch ein Prädikat, das genau dann wahr ist, wenn das Objekt eine positive Instanz des Konzeptes ist. **Beispiele:**

- Das Konzept "GERADE_ZAHL" ist in Termen des Prädikates "n ist eine gerade Zahl" = GERADE_SEIN(n) bezüglich der Instanzen (jede natürliche

Zahl n) mathematisch eindeutig definiert.

- Das Konzept "GRÜN" mit Prädikat GRÜN_SEIN($< Objekt >$) kann nicht eindeutig auf die Angabe physikalischer Eigenschaften des Objektes reduziert werden. Das ist der allgemeine Fall. Die Bedeutung des Konzeptes liegt in seiner offensichtlichen Brauchbarkeit als Ordnungselement im Umgang mit den Dingen der uns umgebenden Welt.

Wir stellen in den Mittelpunkt unserer Überlegungen des Konzeptlernens den Lehrer oder besser die Welt als Quelle der Daten. Die Welt verstehen wir als eine Ansammlung verallgemeinerter Objekte, das sind beliebige dingliche Objekte aber auch Sachverhalte, Situationen, Handlungsabläufe oder ähnliches. Über diese Objekte liegen uns empirische Daten vor, die Aussagen über die Eigenschaften der Objekte machen. Wir dürfen folglich voraussetzen, daß jedes Objekt durch einen verallgemeinerten Merkmalsvektor $\vec{x}$ beschrieben ist. Außerdem ist eine Klasseneinteilung für diese Objekte gegeben, die in eindeutiger Weise für jedes Objekt festlegt, ob das Objekt als zur Klasse gehörig akzeptiert oder verworfen wird. Formal nimmt diese Klassenzuweisung der Lehrer vor, dem damit eine Kenntnis des Konzeptes schon unterstellt wird. In Wirklichkeit liegen eher die folgenden zwei Situationen vor:

- Der Lehrer (in dem Fall besser der Experte) nimmt die Einteilung eher intuitiv vor, ohne sich über die logische Grundlage seiner Entscheidungen wirklich im Klaren zu sein. Die Identifikation der Farbe eines Objektes fällt in diese Kategorie.

- Die Klasseneinteilung kann schon in der Natur der anfallenden Daten begründet sein. Beispielweise können in Abhängigkeit von den Startbedingungen Prozesse zu einem bestimmten Ergebnis führen oder eben nicht. Krankheiten als Beispiel.

Die Herausforderung an den maschinellen Lerner besteht in beiden Fällen im Entdecken der logischen Entscheidungsprinzipien. Somit ist der Lerner am Ende seines Lernprozesses i. a. klüger als der Lehrer, denn er kann ein klares logisches Schema für seine Entscheidungen vorweisen. So wollen wir als Ergebnis des Lernens beispielsweise eine bestimmte Regel haben, die es auch dem Farbenblinden erlaubt, anhand der physikalischen Merkmale (Spektrum) beliebige Objekte in Grüne und Nichtgrüne zu klassifizieren. Ebenso kann der Lerner ein logisches Schema erlernen, mit dem er aus den physiologischen Merkmalen der Patienten auf das Vorliegen einer bestimmten Krankheit schließen kann.

Schematisch können wir also unsere allgemeine Formulierung der Transferfunktion des Lerners

$$y = S\,(x \mid w)$$

oder

$$\vec{x} \;\rightarrow \boxed{\begin{array}{c}\text{Lerner}\\ -\,-\,-\,-\,-\\ \{w\}\end{array}} \rightarrow y$$

jetzt so interpetieren, daß $\mathbf{w}$ einen Satz von Verhaltensregeln darstellt. Beispiel:

$$\vec{x} \;\rightarrow \boxed{\begin{array}{l}\qquad\qquad\text{Lerner}\\ \qquad\quad -\,-\,-\,-\,-\\ IF\ x_1 < 1\ THEN\ y = x_1 x_2\\ IF\ \ldots\\ \vdots\end{array}} \rightarrow y$$

5.1.1 Darstellung der Objekte (Instanzen des Konzeptes)

Konkrete Lernverfahren des Konzeptlernens setzen eine adäquate Darstellung der Welt durch eine geeignete Beschreibungssprache voraus.

Darstellung der Objekte durch Merkmalsvektor $\vec{x} = (x_1, ..., x_n)$

- Komponenten des Merkmalsvektors sind Attribute

$$\vec{x} = (<\text{Attribut}\,1>, <\text{Attribut}\,2>, \ldots)$$

- **Beispiel**: Steckbrief

$$\vec{x} = (<\text{Größe}>, <\text{Augenfarbe}>, \ldots)$$

- Attributwerte rellwertig, ordinal, nominal oder bel. symbolische Ausdrücke

$$\begin{array}{cccc} \vec{x} = & (<\text{Größe}>, & & \ldots) \\ & \swarrow \quad \downarrow \quad \searrow & & \\ & 199\,cm \quad \text{groß} \quad Kat.\,3 & & \end{array}$$

- **Beispiel:** Eine Welt verschiedenfarbiger geometrischer Objekte – Kreise, Rechtecke, Dreiecke,...die die Farben Rot, Grün, Blau, ... annehmen können. Objekt durch einen Merkmalsvektor $\vec{x} = (x_1, x_2)$ (geordnetes Paar bzw. Liste) mit den Attributen $x_1 =< Farbe >$ und $x_2 =< Gestalt >$ beschrieben, z. B.

  $$\vec{x} = (rot, kreis)\,, \quad \vec{x} = (grün, rechteck)\,, \ldots$$

  Jeder dieser Merkmalsvektoren beschreibt eine Instanz des Konzeptes GRÜN, das Prädikat ist "GRÜN_SEIN". Die positiven Instanzen sind durch die Menge $\{(grün, y) \mid y = kreis, dreieck, \ldots\}$ gegeben.

Darstellung der Objekte durch prädikative Ausdrücke

Das Prädikat dient der Beschreibung von Klassen konkreter Objekte. Darstellung durch unäre oder binäre Prädikate. KRANK($peter$) beschreibt das einstellige Prädikat "krank sein". Mehrstellige Prädikate dienen der Darstellung von Relationen $R(x_1, x_2, .., x_n)$ zwischen den Objekten z.B. BESITZ($emil, fahrrad$), bzw. der Formulierung von Aussagen, d. h. deklarativen Sätzen mit Wahrheitswert "wahr" oder "falsch". Ein konkretes Objekt o ist durch einen Satz $D\,(o)$ beschrieben, wobei D ein logischer Ausdruck von einem Argument ist.

Beipiel: Das Objekt "roter Kreis" ist mittels unärer oder binärer Prädikate durch Ausdrücke der Form

- $ROT\,(o) \wedge KREIS\,(o)$

bzw.

- $GESTALT(o, kreis) \wedge FARBE(o, rot)$

beschrieben.

5.1.2 Darstellung der Konzepte

In einer konkreten Darstellung wird das Konzept durch einen Satz von Regeln beschrieben, der in eindeutiger Weise die negativen von den positiven Instanzen unterscheidet. In der prädikatenlogischen Formulierung beispielsweise wird das Konzept durch ein Prädikat dargestellt. Aus der Sicht des Lerners ist das das Zielprädikat **Z**. Aufgabe des Lerners ist das Auffinden eines äquivalenten logischen Ausdrucks, der die eindeutige Klassifikation der Objekte in positive und negative Instanzen des Konzeptes erlaubt.

- Ergebnis des Lernens ist Aussage der Form

$$\boxed{\forall o \quad \mathbf{Z}(o) \Leftrightarrow C(o)}$$

wobei C ein logischer Ausdruck. Beispiel: Konzept $ECKIG$:

$$\forall o \quad \mathrm{ECKIG}(o) \Longleftrightarrow \mathrm{GESTALT}(o, dreieck) \vee \mathrm{GESTALT}(o, viereck) \tag{5.1}$$

5.1.3 Instanzen– und Regelraum

Instanzenraum

- Geeigneter abstrakter Raum zur Darstellung der möglichen Instanzen des Problems.
- Jede Instanz und damit auch jedes Trainingsbeispiel ist ein Punkt in diesem Raum. Allgemein nichtmetrischer Raum.

Regelraum

- Geeigneter abstrakter Raum zur Darstellung der Regeln.
- Beispiel: Welt farbiger geometrischer Objekte. Beispiel für eine Regel in prädikatenlogischer Formulierung

$$\forall o \; FARBE(o, grün) \wedge GESTALT(o, dreieck) \Leftrightarrow GRÜN(o) \tag{5.2}$$

- Eine beliebige Regel kann als eine Hypothese über das Konzept aufgefaßt werden.

5.1.4 Lernaufgabe

Gegeben:

- Eine Menge von Trainigsbeispielen,

$$\mathcal{T} = \{(\vec{x}, y) \mid \vec{x} \in \Omega, \quad y \in \{T, F\}\}$$

  d. h. einer Menge von positiven und negativen Instanzen des Konzeptes + Ein Lehrer (Trainer), der die Trainingsbeispiele (in positive ($y = T$) und negative ($y = F$)) klassifiziert.

- Eine geeignete Beschreibungssprache für diese Instanzen.

- Der Regelraum mit einer geeigneten Darstellung (d. h. einer Sprache zur Formulierung) der möglichen oder denkbaren Konzepte.

Gesucht:

- Regel zur eindeutigen Identifikation der positiven Instanzen des Konzeptes.

Lernen ist hier das zielgerichtete Absuchen des Regelraumes nach der am besten passenden Regel. Grundlegend für die meisten Lösungsverfahren ist die **Annahme einer geschlossenen Welt (Closed world Assumption), d. h. daß der Regelraum das gesuchte Konzept enthält.** Die gefundene Regel ist dem Lehrer i. a. nicht bekannt. Die Klassifikation der Instanzen kann vom Lehrer nach einem beliebigen Prinzip vorgenommen worden sein. Der Lerner hat dann das selbst dem Lehrer unbekannte Konzept zur Klassifikation der Daten gewonnen. Damit leistet das Erlernen des Konzeptes einen Beitrag zum Verststehen der durch die Daten repräsentierten Sachverhalte.

5.1.5 Einsatzmöglichkeiten des Konzeptlernens

1. **Klassifikation**: Unbekannte Inputs werden als positive oder negative Instanzen des Konzeptes klassifiziert.

2. **Vorhersage**: Wenn Trainingsinstanzen sukzessive Elemente einer Folge sind, können neue Elemente der Folge vorhergesagt werden.

3. **Datenkompression**: Suche Konzept zur kompakten Beschreibung eines großen Instanzenraumes.

4. Und andere.

5.2 Inferenzprozeduren

5.2.1 Allgemeines

- Lernen = Absuchen des Regelraumes nach dem *best match* Konzept.

- *Exhaustive search* $\Rightarrow$ Kombinatorische Explosion.

- Heuristiken zur Exploration des Regelraumes und Aufstellung von Hypothesen.

  $$\boxed{\text{Regel}\, i} \rightarrow \boxed{\begin{array}{c}\text{Heuristischer}\\ \text{Operator}\end{array}} \rightarrow \boxed{\text{Regel}\, i'}$$

  analog

  $$\boxed{\text{Hypothese}\, i} \rightarrow \boxed{\begin{array}{c}\text{Heuristischer}\\ \text{Operator}\end{array}} \rightarrow \boxed{\text{Hypothese}\, i'}$$

- Häufigste Inferenzprozedur: Verallgemeinerung einer bestehenden Regelbasis.

  $$\begin{array}{c} \boxed{\text{Regel}\, i_1} \searrow \\ \vdots \\ \boxed{\text{Regel}\, i_s} \nearrow \end{array} \boxed{\begin{array}{c}\text{Verallg.}\\ \text{Operator}\end{array}} \rightarrow \boxed{\text{Regel}\, i}$$

- Die Darstellung der Regeln im Prädikatenkalkül impliziert i. a. eine vergleichsweise einfache Syntax für die Verallgemeinerung der Regeln.

5.2.2 Syntaktische Regeln der Inferenz

Die gezielte Absuche des Regelraumes velangt, sich in diesem definiert bewegen zu können. Das kann auf der syntaktischen Ebene organisiert werden, was sich insbesondere für modellfreie Verfahren anbietet, da diese ohne Heuristiken für die Bewegung im Regelraum auskommen sollen. Beispiele:

- *Verwandlung von Konstanten in Variable*: Die Aufgabe sei die Entdeckung des Konzeptes "PAAR" in einer Menge aus N Objekten. Jede Instanz ist eine Menge $\hat{o}$ aus N Elementen mit den Attributen FARBE und GESTALT . Eine positive Instanz des Konzeptes PAAR (in der Menge befinden sich zwei Objekte der gleichen Gestalt) ist z. B. beschrieben durch

$$\text{Instanz 1: } \exists o, o' \quad GESTALT\,(o, kreis) \wedge GESTALT\,(o', kreis) \tag{5.3}$$

$$\text{Instanz 2: } \exists o, o' \quad GESTALT\,(o, dreieck) \wedge GESTALT\,(o', dreieck) \tag{5.4}$$

1. Die durch Ersetzen der Variablen durch eine Konstante x verallgemeinerte hypothetische Regel ist dann

$$Regel : \forall \emptyset, o' \quad GESTALT\,(o, x) \wedge GESTALT\,(o', x) \Leftrightarrow PAAR\,(o, o') \tag{5.5}$$

- *Weglassen von Bedingungen*: Für das gleiche Problem seien die positiven Trainingsinstanzen

$$\begin{aligned} \text{Instanz 1: } \exists o, o' \quad & GESTALT\,(o, kreis) \wedge GESTALT\,(o', kreis) \wedge \\ & FARBE\,(o, rot) \\ \text{Instanz 2: } \exists o, o' \quad & GESTALT\,(o, dreieck) \wedge GESTALT\,(o\prime, dreieck) \wedge \\ & FARBE\,(o, rot) \end{aligned} \tag{5.6}$$

  gegeben. Die naheliegende Verallgemeinerung ergibt sich dann aus dem Weglassen der irrelevanten Bedingung $FARBE\,(o, rot)$.

- *Hinzufügen von Optionen*: Die Aufgabe sei nun, das Konzept $ECKIG$ zu lernen. Die positive Instanz

$$I1 : \quad GESTALT\,(o, dreieck)$$

könnte zu der Regel

$$Regel: \quad \forall o \quad GESTALT\,(o, dreieck) \Leftrightarrow ECKIG\,(o)$$

gedeutet werden. Eine weitere Trainingsinstanz

$$I2: \quad GESTALT\,(o, viereck)$$

führt dann mit der Hinzunahme der durch diese Instanz nahegelegten Option auf die Regel

$$Regel': \quad \forall o \quad GESTALT\,(o, dreieck) \vee GESTALT\,(o, viereck) \Leftrightarrow ECKIG\,(o)$$

- *Kurvenanpassung*: Die Aufgabe sei nun die Entdeckung eines funktionellen Zusammenhanges zwischen zwei Inputs x_1, x_2 und einem Output y. Die Instanzen sind die Tripel (x_1, x_2, y), z. B.

$$\begin{array}{ll} I1: & (1,1,1) \\ I2: & (2,1,2) \\ I3: & (0,0,1) \end{array}$$

Durch lineare Regression folgt als hypothetischer allgemeiner Zusammenhang

$$Regel\,(\vec{x}, y): \quad y = x_1 - x_2 + 1 \tag{5.7}$$

was gegen neue Instanzen getestet werden kann. Widersprechen beispielsweise die neuen Instanzen der Regel (5.7), so kann eine weitere Verallgemeinerung dieser Regel nötig werden, versuchsweise etwa durch **Nullsetzen des Koeffizienten** von x_2

$$Regel'\,(\vec{x}, y): \quad y = \alpha\, x_1 + \beta$$

wobei die Koeffizienten α, β geeignet gewählt werden müssen, um die bisher gesehenen Trainingsinstanzen optimal zu fitten.

5.2.3 Absuchen des Regelraumes. Aktives Lernen.

- Lernen = Entdecken des auf alle Instanzen passenden Konzeptes.

- Trainingsinstanzen dirigieren Richtung der Suche im Regelraum.

- Für auseinanderfallende Darstellungen der Räume: Kompliziertes Wechselspiel zwischen Bewegungen im Regel- und im Instanzenraum

$$\begin{tikzcd}
 & \boxed{\text{Exploration-Planung}} \arrow[d] & \\
 & \boxed{\text{Instanzen-Wahl}} \arrow[dl] & \\
\boxed{\text{Instanzen-Raum}} \arrow[dr] & & \boxed{\text{Regel-Raum}} \arrow[ul] \\
 & \boxed{\text{Interpretation}} \arrow[ur] &
\end{tikzcd}$$

- **Beispiel** für auseinanderfallende Beschreibungssprachen: Welt farbiger geometrischer Objekte mit Darstellungen, wobei die

  – Trainingsinstanzen durch Merkmalsvektoren $\vec{x} = (< Farbe >, < Gestalt >)$ und die

  – Regeln durch Aussagen der Art

  $$\forall o \quad GESTALT\,(o, dreieck) \vee GESTALT\,(o, viereck) \Leftrightarrow ECKIG\,(o)$$

  beschrieben sind.

- Lernen geschieht in den Schritten Präsentation einer Instanz, **Interpretation derselben in der Sprache der Regeln**, Test der Übereinstimmung mit den vorhandenen Hypothesen (als gültig angenommene Regeln), Aktualisierung der Hypothesen, Planung der weiteren Schritte z. B. durch gezielte Auswahl der nächsten Instanz.

- Der Lerner kann neue Instanzen gezielt abrufen, um den maximalen Lernfortschritt zu erzielen $\quad\Rightarrow\quad$ **Aktives Lernen**.

5.3 Der Versionsraum

5.3.1 Idee

- Einheitliche Darstellung von Instanzen und Regeln (Instanzen als spezielle Regeln formuliert).
- Feststellung einer Ordnung im gemeinsamen Darstellungsraum. Ordnungsparameter = Grad der Allgemeinheit der Regeln. Generell kann so jeder Regelraum gemäß dem Allgemeinheitsgrad der Regeln geordnet werden.
- Eingrenzung der Menge $\mathcal{H}$ der plausiblen Hypothesen durch die Menge $\mathcal{G}$ ihrer allgemeinsten und $\mathcal{S}$ ihrer spezifischsten Elemente.

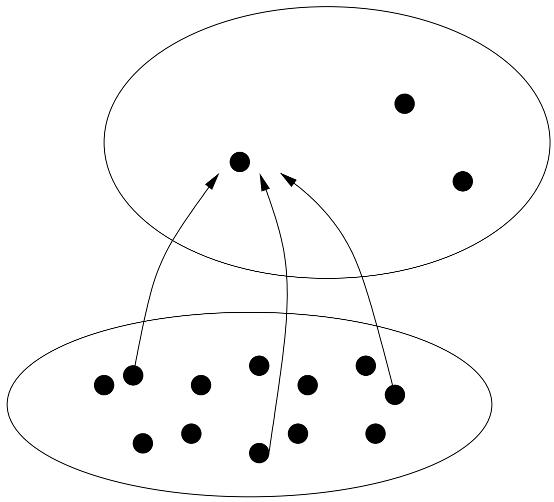

Abbildung 5.1: Zwei Ebenen eines hiearchisch geordneten Regelraumes. Ein elementarer Verallgemeinerungsschritt projiziert von der unteren auf die nächst höhere Ebene

5.3.2 Organisation des Versionsraumes

Definition

- Darstellung des Konzeptes durch einen prädikativen Ausdruck, der wahr für alle positiven und keine der negativen Instanzen des Konzeptes ist.

- Hypothesen sind tentative Formulierungen des Konzeptes. Jede Hypothese $\mathbf{H}_i$ ist Satz
$$\forall o \quad \mathbf{Z}(o) \Longleftrightarrow C_i(o)$$
wobei C_i ein beliebiger in der gültigen Beschreibungssprache formulierter logischer Ausdruck ist.

- Alle Hypothesen durch die Menge aller prädikativen Ausdrücke gegeben, die auf der konkreten Objektdomäne denkbar sind.

- Der Regelraum ist durch die Menge dieser Hypothesen gegeben.

- Der Versionsraum: Menge aller Hypothesen, die mit den schon gesehenen Trainingsinstanzen noch verträglich sind.

$$\boxed{\text{Versionsraum} = \text{Menge } \mathbf{H} \text{ der } \textbf{plausiblen} \text{ Hypothesen}}$$

Partielle Ordnung im Versionsraum

- Ordnung von Spezifisch nach Allgemein.

  1. Spezifischste Punkte (Regeln) im Regelraum $\rightarrow$ Trainingsinstanzen.
  2. Allgemeinste Regel $\longrightarrow$ Allbeschreibung ("Alle Objekte sind positive Instanzen des Konzeptes")

- Eingrenzung der Menge **H** der plausiblen Hypothesen durch die Menge **G** ihrer allgemeinsten und **S** ihrer spezifischsten Elemente.

5.3.3 Das Lernverfahren

1. **Initialisierung**: Setze Versionsraum gleich dem vollen Regelraum, d. h. $\mathbf{G} = \{true\}$ (entsprechend der Aussage "Alle Objekte sind positive Instanzen des Konzeptes") und $\mathbf{S} = \{false\}$ ("Kein Objekt ist positive Instanz des Konzeptes"). Damit liegen alle Hypothesen tatsächlich im Raum zwischen diesen Grenzmengen.

2. **Präsentation einer Trainingsinstanz und Aktualisierung von G bzw. S:**

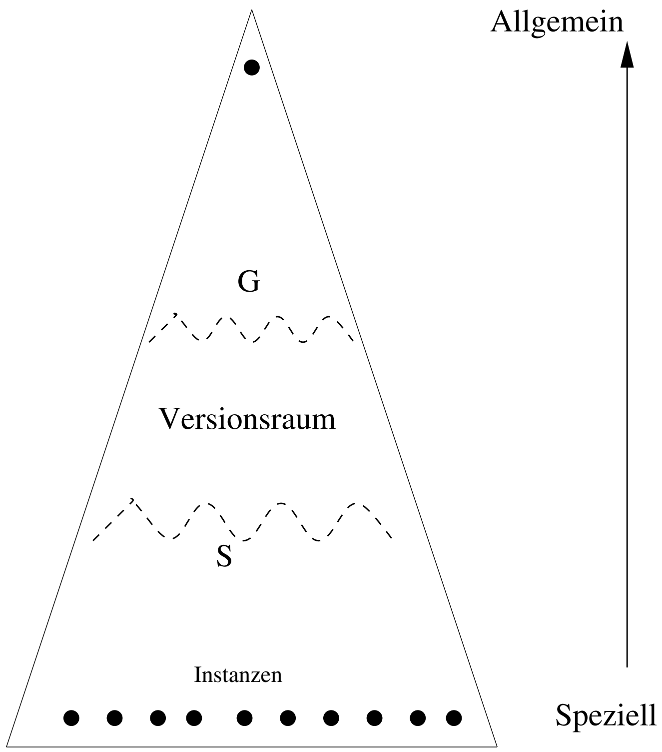

Abbildung 5.2: Hierachisch nach Grad der Allgemeinheit der Regeln geordneter Regelraum. Die spezifischsten Regeln sind die Instanzen des Konzeptes selbst, die allgemeinste Regel ist die Allbeschreibung. Den Versionsraum **H** bilden die beim aktuellen Stand des Lernens noch plausiblen Regeln bzw. Hypothesen. Die Mengen **S** (unten) bzw. **G** (oben) sind die Grenzmengen von **H** gegeben durch die Menge der spezifischsten bzw. allgemeinsten Hypothesen in **H**.

- Bei Präsentation einer **positiven Trainingsinstanz** (Update-S Prozedur): Elimination aller Konzepte aus **G**, die TI fehlklassifizieren. Ersetzen der mit TI inkonsistenten Konzepte in **S** durch ihre kleinstmöglichen Verallgemeinerungen.
- Bei Präsentation einer **negativen Trainingsinstanz** (Update-G Prozedur): Elimination aller Konzepte aus **S**, die TI fehlklassifizieren. Ersetzen der mit TI inkonsistenten Konzepte in **G** durch ihre kleinstmöglichen Spezialisierungen.

Verallgemeinerung bzw. Spezialisierung so, daß TI mit erfaßt wird!
Der Algorithmus ist ein **Kandidaten – Eliminations – Algorithmus.**

Beispiel

- Welt aus Objekten mit Farbe = rot bzw. grün und Gestalt = Dreieck, Viereck bzw. Kreis.
- Notation (r, D) für $ROT\,(o) \wedge DREIECK\,(o)$ usw.
- Das Programm soll das Konzept "KREIS" erkennen.
- Nach der Allgemeinheit der Hypothesen geordnet ist der anfängliche Versionsraum in Abb. 5.3 dargestellt.

- Initialisierung: Die Menge **H** ist gleich dem gesamten Regelraum:
- **G** enspricht Hypothese "Alle Objekte sind Kreise".
- **S** enspricht Hypothese "Alle Objekte sind keine Kreise".
- Präsentation der Trainingsinstanz (r, K) = positive Instanz des Konzeptes $\Rightarrow$ Update-S Prozedur mit Ergebnis

$$\begin{aligned} \mathbf{G} &= \{true\} = (x, y) \\ \mathbf{S} &= (r, K) \end{aligned} \tag{5.8}$$

  Der Versionsraum ist in Abb. 5.4 dargestellt.

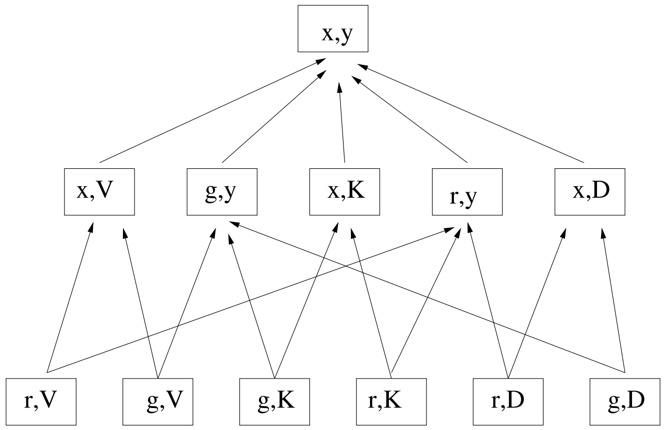

Abbildung 5.3: Der nach der Allgemeinheit der Hypothesen geordnete Regelraum einer Welt aus farbigen geometrischen Objekten. Notation: Das Objekt (r, V) bezeichnet ein rotes Viereck usw.

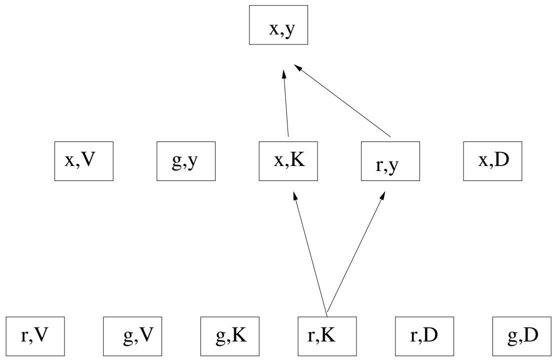

Abbildung 5.4: Der Versionsraum nach der Präsentation der ersten Instanz "roter Kreis".Die Trainingsinstanz selbst ist wegen der Initialisierung $\mathbf{S} = \{false\}$ die kleinstmögliche Verallgemeinerung. Die Pfeile markieren die potentiellen Pfade durch den Versionsraum.

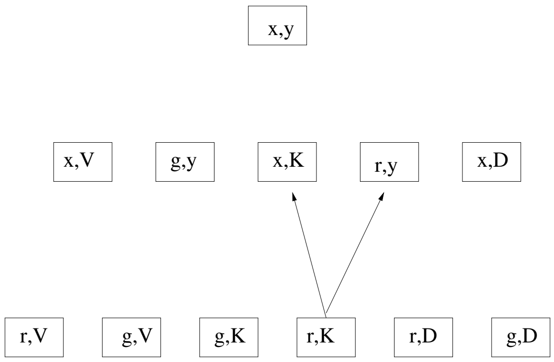

Abbildung 5.5: Der Versionsraum nach der Präsentation der negativen Instanz "grünes Dreieck".

- Neue Trainingsinstanz: (g, D) als negative Instanz des Konzeptes $\Rightarrow$ Update-G Prozedur führt auf Verkleinerung des G- Raumes, d. h. der Versionsraum ist jetzt begrenzt durch

$$\mathbf{G} = \{(x, K), (r, y), (x, V)\}$$
$$\mathbf{S} = \{(r, K)\} \tag{5.9}$$

  s. Abb. 5.5.

- Dritte Trainingsinstanz: (g, K) als weitere positive Trainingsinstanz $\Rightarrow$ Update-S führt auf Elimination von (r, y) und (x, V) aus G da inkonsistent mit der Trainingsinstanz. Verallgemeinerung in $\mathbf{S}$ führt auf (x, K). Folglich

$$G = (x, K)$$

$$S = (x, K)$$

- Anhalten des Algorithmus, da $G = S$. Ausgabe des Konzeptes "KREIS"

$$\forall o \quad GESTALT\,(o, K) \quad \Leftrightarrow \quad KREIS$$

- Das Beispiel zeigt, daß die Mengen S bzw. G gerade die Mengen der hinreichenden bzw. notwendigen Bedingungen für die Zugehörigkeit zu den positiven Instanzen des Konzeptes darstellen. Z. B. enthält S nach der zweiten

Trainingsinstanz die hinreichenden Bedingungen dafür, (r, K) als positive Instanz des Konzeptes $KREIS$ zu klassifizieren.

Bemerkung: Der Kandidaten-Eliminationsalgorithmus ist ein

- **Breadth-First Algorithmus**. Kombinatorische Explosion des Aufwandes bei großen Regelräumen.
- Der Algorithmus ist **datengetrieben**. Empfindlich gegen zufällige Störungen (Rauschen) in den Trainingsinstanzen.
- Der Algorithmus ist ein Least-Commitment-Algorithmus, d. h. er verallgemeinert nur, wenn er dazu durch eine Trainingsinstanz gezwungen wird und nimmt diese Verallgemeinerung nur in der kleinstmöglichen Form vor.
- Unfähigkeit disjunktive Konzepte zu entdecken. Ein Ausweg ist die **Vermeidung trivialer Disjunktionen** durch geeignete Vorschriften.

5.4 Klassifikationslernen

- Gegeben sei eine Menge von Objekten mit bekannten Klassenzugehörigkeiten.
- **Ziel**: Erlerne Regel zur Klassifikation dieser Objekte.
- Jede Klasse k durch ein Konzept C_k beschreibbar, d. h.

$$\boxed{\begin{array}{c} \boxed{\text{Objekt x} \in \text{Klasse k}} \\ \Updownarrow \\ \boxed{\text{x positive Instanz des Konzeptes } C_k} \end{array}}$$

- Lernen (Absuche des Regelraumes) mit Algorithmus A^q (Michalski): Auffinden des allgemeinsten Konzeptes zur jeweiligen Klasse durch Anwenden des Kandidaten–Eliminationsalgorithmus wie beim Erlernen eines *single* Konzeptes.
- Versionsraummethode. *Single–representation* Trick.

- Folge: Konzeptbeschreibungen überlappen in nicht mit Trainingsinstanzen belegten Bereichen des Regelraumes.

- Ausweg: Verwende zum Erlernen eindeutiger Klassifikationsregeln die Konzepte der schon gelernten Klassen als negative Instanzen zum Erlernen der aktuellen Klasse.

- Nebeneffekt: Dominanzhierarchie der Konzepte gemäß ihrer Reihenfolge im Lernprozeß.

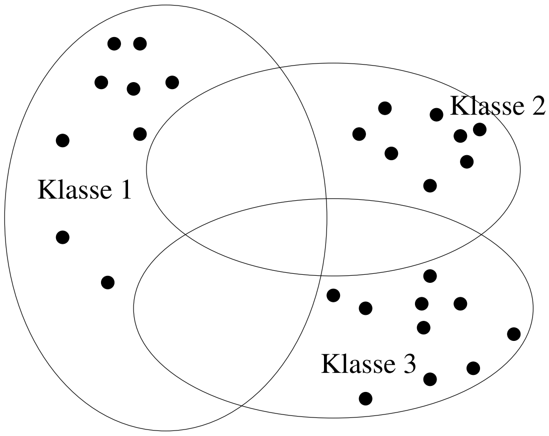

Abbildung 5.6: Instanzen– bzw. Regelraum mit Instanzen dreier Klassen. Die durch das Kozeptlernen gefundenen Regeln überlappen in den nicht instanziierten Gebieten des Regelraumes.

5.5 Generate-And-Test Verfahren (G & T)

Eine Alternative zu den datengetriebenen Verfahren stellen modellgetriebene Verfahren dar, die bezüglich Unempfindlichkeit gegenüber Rauschen den datengetriebenen Verfahren weit überlegen, allerdings andererseits nicht wie diese in der Lage sind, durch neue Trainingsinstanzen hinzukommende Zusatzinformation inkrementell in die Konzeptbeschreibung zu integrieren.

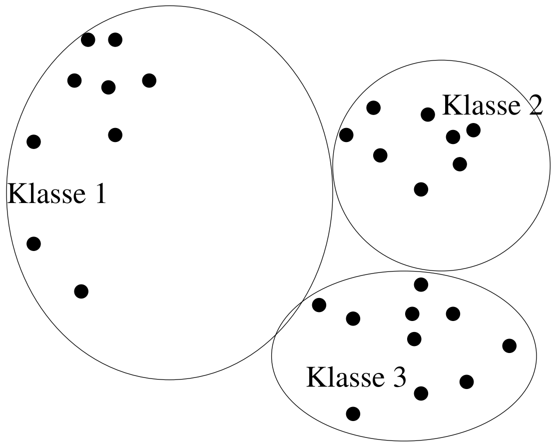

Abbildung 5.7: Die Konzepte früher gelernter Klassen können als negative Instanzen für das Erlernen der aktuellen Klasse dienen. Ergebnis sind nicht überlappende Klassendomänen. Allerdings induziert die Reihenfolge des Lernens eine Dominanzhierarchie der Klassen. Im Bild wurde Klasse 1 vor Klasse 2 und diese vor Klasse 3 gelernt.

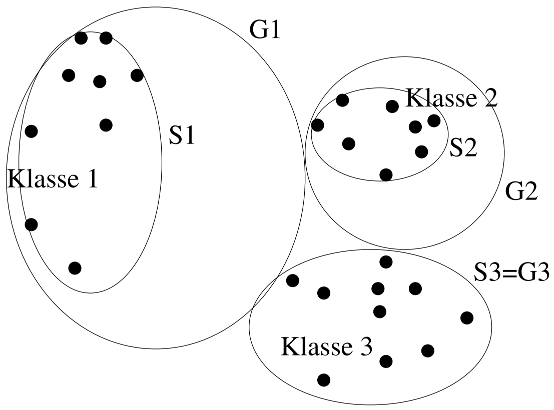

Abbildung 5.8: Erlernen der Mengen G und S für jede Klasse.

Beispiel: INDUCE1.2-Algorithmus (Dietterich und Michalski 1981). Unter Benutzung des Tricks der einheitlichen Repräsentation des Instanzen- und Regelraumes (s. o.) bilden die den zufällig ausgewählten Trainingsinstanzen entsprechenden Punkte im Regelraum die Ausgangspunkte für eine aufwärtsgerichtete Strahlsuche (*Beam Search)* im Regelraum. Das entspricht dem Update-S Algorithmus in der Versionsraummethode. Hier werden allerdings modellbasiert Heuristiken aktiviert, die die Verallgemeinerung von S steuern, so daß nur einige wenige Verallgemeinerungen wirklich entdeckt werden. Der Basismechanismus der Verallgemeinerung ist dabei das **Weglassen von konjunktiven Bedingungen und Hinzufügen innerer disjunktiver Optionen.**

Allgemein folgen die über eine *Beam-Search-Prozedur* realisierten G & T Verfahren etwa folgendem Schema:

1. Initialisierung: Wähle zufällig oder gezielt τ Trainingsinstanzen aus der Menge $\mathcal{T}$ aller Trainingsinstanzen aus. Als Punkte im Regelraum interpretiert bilden diese die Menge H der Hypothesen.

2. Generierung neuer Hypothesen durch minimale Verallgemeinerung von H.

3. Pruning: Auswahl der τ besten (=plausibelsten) Hypothesen. Bewertungsfunktionen: Länge der Konzeptbeschreibungen, Grad der Übereinstimmung mit allen Trainingsinstanzen.

4. Test: Überprüfe Übereinstimmung mit allen Trainingsinstanzen. Übergabe der Konzepte mit vollständiger Übereinstimmung an Ausgabemenge C.

5. Abbruch bei $H ='' leer''$ oder $C ='' voll''$, sonst Wiederbeginn bei 1.

Der Algorithmus benötigt nur positive Trainingsinstanzen.

Eine Anwendung – Erlernen von Strukturkonzepten

Wir betrachten Strukturen, die aus einer Menge geometrischer Objekte (Bausteine) konstruiert werden können. Erlernt werden sollen Konzepte, die rein strukturelle Information enthalten (Beispiel: "BOGEN", "LIEGEN_AUF",...).

Für die Beschreibung der Szene ist es günstig, unäre und binäre (oder höhere) Prädikate zu verwenden, die unären dienen der Beschreibung der Eigenschaften wie Größe oder Gestalt der Elemente (Bausteine), die binären enthalten die strukturellen Informationen, d. h. sie beschreiben die Relationen zwischen den Elementen in der betrachteten Struktur. Die Trainingsinstanz

$$I1 : \exists u, v : \quad G(u) \wedge K(u) \wedge G(v) \wedge K(v) \wedge AUF(u, v)$$

beschreibt zwei aufeinanderliegende große Kreise. (Abkürzungen: $G \rightarrow GROSS$; $K = KREIS$). Analog beschreibt

$$I2 : \exists u, v, w : \quad KL(u) \wedge K(u) \wedge G(v) \wedge Q(v) \wedge G(w) \wedge Q(w)$$

$$\wedge AUF(u, v) \wedge AUF(v, w)$$

einen kleinen Kreis $(KL(u) = KLEIN(u))$ auf zwei übereinanderliegenden großen Quadraten.

Entsprechend der Dichotomie der Prädikate ist es angezeigt, den Regelraum R in zwei Unterräume R_1 (nur-Struktur-Raum=**Masterraum**) und R_2 (nur Eigenschaften der Elemente) zu zerlegen, was eine entsprechende Zerlegung der Konzeptbeschreibung induziert. Für das Erlernen eines Strukturkonzeptes erscheint es sinnvoll, zunächst eine Projektion (durch Weglassen aller durch unäre Prädikate formulierten Bedingungen) der vollen Beschreibung der Trainingsinstanzen in den Masterraum R_1 vorzunehmen, das dort gültige Konzept zu ermitteln und danach im R_2 eine an der neuen Struktur des R_1 orientierte Beam-Search durchzuführen.

Diese Zerlegung des Regelraumes definiert gleichsam das Modell, das der Suche zugrundeliegt. Das Modell integriert in geeigneter Weise das Ziel - Auffinden eines reinen *Struktur*konzeptes - in die Strategie für das Absuchen des Regelraumes.

Im obigen Beispiel sind die projizierten Regeln

$$\begin{aligned} R1' : &\quad \exists u, v \quad AUF(u, v) \\ R2' : &\quad \exists u, v, w : AUF(u, v) \wedge AUF(v, w) \end{aligned} \tag{5.10}$$

Die *Beam Search* in dem so definierten R_1 liefert $\quad \exists u, v \quad AUF\,(u, v) \quad$ als neue Beschreibung einer positiven Instanz des Konzepts. Von diesem Ergebnis geleitet wird nun ein neuer Merkmalsvektor

$$(GROESSE\,(u)\,, GESTALT\,(u)\,, GROESSE\,(v)\,, GESTALT\,(v))$$

zur Beschreibung der Instanzen des R_2 gebildet, und die anstehenden Objekte werden in dieser Sprache beschrieben. (Beispiel: Die der Instanz $I2$ entsprechende Situation ist alternativ durch die Merkmalsvektoren $(klein, kreis, gro\ss, quadrat)$ oder $(gro\ss, quadrat, gro\ss, quadrat)$ beschrieben). Die anschließende *Beam Search* im R_2 führt zur Verallgemeinerung dieser Trainingsinstanzen. Die Kombination mit der im R_1 gefundenen Beschreibung ergibt schließlich die zwei gültigen Konzeptbeschreibungen

$$C1: \quad \exists u, v: \quad GROSS\,(u) \wedge GROSS\,(v) \wedge AUF\,(u, v)$$

$$C2: \quad \exists u, v: \quad KREIS\,(u) \wedge GROSS\,(v) \wedge AUF\,(u, v)$$

was den Aussagen "ein Kreis liegt auf einem großen Objekt" oder "zwei große Objekte liegen übereinander" entspricht.

Kapitel 6

Erlernen von Entscheidungsbäumen

6.1 Entscheidungsbäume

Entscheidungsbäume können wie die Ausdrücke der Prädikatenlogik ebenfalls zur Darstellung beliebiger Konzepte oder Regeln Verwendung finden. Entscheidungsbäume sind gerichtete Graphen, die mit einem Wurzelknoten beginnen und in einer nicht festgelegten Anzahl von Blattknoten enden. Dazwischen liegt eine ebenfalls offene Anzahl von Entscheidungsknoten. Wir stellen die Objekte wieder durch Merkmalsvektoren dar. Beispiel: Eine Welt geometrischer Objekte, die durch drei Merkmale charakterisiert seien

$$\vec{x} = (farbe, gestalt, größe)$$

Die Blattknoten sind mit Entscheidung JA oder $NEIN$ über die Zugehörigkeit der präsentierten Instanz zu den positiven Instanzen des Konzeptes belegt. [1] Der Wurzelknoten kann ein Entscheidungs– oder ein Blattknoten sein. Ein Baum kann aus einem einzigen Blattknoten bestehen. Der Wurzelknoten dient als Input, die Blattknoten als Output des Entscheidungsbaumes. Jeder Entscheidungsknoten ist einem bestimmten Merkmal $\langle m \rangle$ zugeordnet und ist Ausgangspunkt für N_{m_i} gerichtete Kanten (Zeiger) auf weitere Knoten, wobei N_m die Zahl der

[1] Ist die Menge der einem Blattnoten zugeordneten Instanzen leer ist, dann erfolgt die Belegung mit einem *default* Wert.

verschiedenen Attributwerte des Merkmals $\langle m \rangle$ ist (jeweils ein Zeiger für jede Ausprägung des Merkmales).

Der Entscheidungsprozeß entspricht einer Bewegung durch den Baum, die am Wurzelknoten startet und an genau einem der Blattknoten endet. An jedem der angelaufenen Entscheidungsknoten $< m_i >$ wird der Wert des Merkmals m_i abgefragt und dem zugeordneten Zeiger zum nächsten Knoten gefolgt. Am Blattknoten angekommen wird die Belegung des Blattknotens als Output des Entscheidungsprozesses ausgegeben.

Der Entscheidungsbaum splittet die Menge $\mathbf{T}$ der Trainingsinstanzen in Teilmengen auf

$$\mathbf{T} = \cup_{\alpha=1}^{N_B} \mathbf{T}_\alpha = \mathbf{T}_1 \cup \mathbf{T}_2 \ldots \cup \mathbf{T}_{N_B} \tag{6.1}$$

wobei$\mathbf{T}_\alpha$ die Menge der Instanzen ist, die am Blattknoten α ankommt. **Beispiele:**

- Entscheidungsbaum als Beschreibung des Konzeptes $GRÜNES\_DREIECK$:

$$\begin{array}{ccccc}
 & & \langle gestalt \rangle & & \\
quadr. \swarrow & & \swarrow krs. & \searrow dr. & \\
[N] & [N] & & \langle farbe \rangle & \\
 & & rot \swarrow & & \searrow grün \\
 & & [N] & & [J]
\end{array}$$

- Das disjunktive Konzept $GRÜNES\ DREIECK \vee GROSSES\ QUADRAT$ ist durch folgenden Entscheidungsbaum repräsentiert

$$\langle gestalt \rangle$$

$$quadr. \swarrow \qquad \swarrow krs. \qquad \searrow dr.$$

$$\langle größe \rangle \qquad [N] \qquad \langle farbe \rangle$$

$$groß \swarrow \quad \downarrow klein \qquad rot \swarrow \qquad \searrow grün$$

$$[J] \qquad [N] \qquad [N] \qquad [J]$$

Die Darstellung eines Konzeptes (einer Regel) durch einen Entscheidungsbaum ist nicht eindeutig. Als anerkanntes heuristisches Kriterium für die Auswahlt des besten Entscheidungsbaumes gilt das *Ockham's razor* Prinzip, das grob gesprochen den Entscheidungsbaum mit der besten Generalisierungsleistung favorisiert [2].

6.2 Der Lernalgorithmus ID3/C4.5

Das bekannteste Verfahren zum Erlernen eines Entscheidungsbaumes aus einer Menge $\mathcal{T}$ von Trainingsinstanzen ist Quinlans ID3/C4.5 Algorithmus[36], s. auch [35, 37]. Das Erlernen eines Entscheidungsbaumes au einer beliebigen Menge $\mathcal{T}$ von Trainingsinstanzen findet in folgenden Schritten statt:

1. Initialisierung mit leerem Entscheidungsbaum

2. Aktualisierung des Entscheidungsbaumes in den Schritten

   (a) Falls alle Instanzen in $\mathcal{T}$ (Inputmenge) positive Instanzen des Konzeptes sind $\Rightarrow$ Schaffe einen $[J]$ Knoten

[2] Wilhelm von Ockham bzw. William of Occam (ca. 1285 – 1350): Scholastischer Philosoph und Theologe. Franziskaner. Gründer des spätmittelalterlichen Nominalismus. Sein (angeblicher) Ausspruch *entia non sunt multiplicanda praeter necessitatem* wird als allgemeines Prinzip, das Einfache dem Kompliziertem vorzuziehen, interpretiert.

(b) Falls alle Instanzen in $\mathcal{T}$ negative Instanzen des Konzeptes sind $\Rightarrow$ Schaffe einen $[N]$ Knoten

(c) Sonst:

i. (Heuristische) Auswahl des "besten" Merkmals $\langle m \rangle$. Einrichtung eines Entscheidungsknotens mit je einer gerichteten Kante für jeden der möglichen Attributwerte $m_1, m_2, \ldots m_{N_m}$.

$$\langle m \rangle$$

$$m_1 \swarrow \quad m_2 \swarrow \quad \ldots\ldots \qquad \searrow m_{N_m}$$

Beispiel: $m = farbe, \quad m_1 = rot,\ m_2 = grün, \ldots$.

ii. Jede der gerichteten Kanten $i = 1, 2, \ldots, N_m$ ist wieder Wurzel eines Entscheidungsbaumes. Konstruktion dieses Entscheidungsbaumes durch Rekursion beginnend bei (2) mit $\mathcal{T} \leftarrow \mathcal{T}_i$ wobei $\mathcal{T}_i =$ Menge aller Trainingsinstanzen die Kante i zugewiesen wurden.

Eine etwas erweiterte Darstellung des Algorithmus findet sich in Abb. 6.1.

6.2.1 Lernen mit Fenstermengen

Der obige Algorithmus erfordert die ständige direkte Verfügbarkeit aller Trainingsinstanzen. Das ist bei großen Datensätzen von erheblichem Nachteil. Eine entscheidende Verbesserung bringt hier der Algorithmus ID3 bzw. C4.5 in der neueren Version. Dieser wählt durch aktive Experimentplanung eine gute Untermenge der Trainingsinstanzen aus.

Der Algorithmus:

1. Wähle eine Fenstermenge genannte Stichprobe $\mathcal{F} \subset \mathcal{T}$.

2. Lerne Entscheidungsbaum, der die Trainingsinstanzen in $\mathcal{F}$ beschreibt.

3. Durchsuche $\mathcal{T}$ nach Gegenbeispielen.

4. Bilde eine neue Fenstermenge durch Kombination einer ausgewählten Untermenge der aktuellen Fenstermenge mit den Gegenbeispielen.

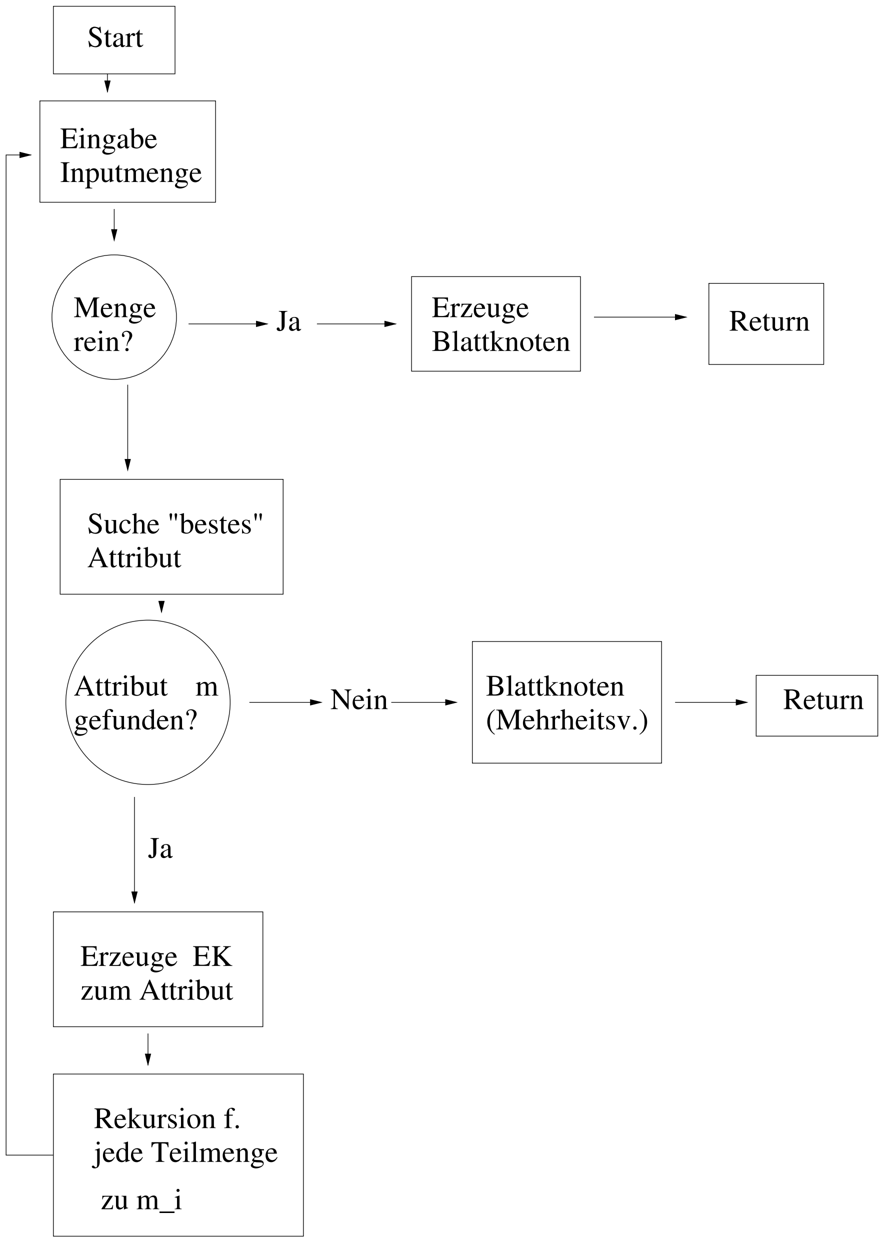

Abbildung 6.1: Der Algorithmus zum Erlernen von Entscheidungsbäumen. Gegenüber der Darstellung im Text enthält der Algorithmus hier einen zusätzlichen Zweig, der bei verrauschten Daten ein Pruning bewirkt, s. u.

5. Abbruch oder Neubeginn bei 2.

Das Verfahren ist ein Beispiel einer sog. erwartungsgeleiteten Filterung (*expectation-based filtering*) der Datenbasis, d. h. das Programm richtet seine Aufmerksamkeit auf die Trainigsbeispiele, die seinen Erwartungen am meisten widersprechen.

6.3 Heuristiken für die Merkmalsauswahl

Das Ergebnis des Lernens ist nicht eindeutig festgelegt. Die Struktur des erhaltenen Baumes ist entscheidend durch die Folge der gewählten Entscheidungsmerkmale bestimmt. Die Struktur des Baumes bestimmt aber die Generalisierungsfähigkeit und Rauschtoleranz des gefundenen Konzeptes. Für die Einrichtung eines Entscheidungsknotens kommen in der Praxis verschiedene Heuristiken zur Auswahl des besten Merkmals zum Einsatz.

Ein Ansatz besteht in der Auswahl des Merkmals mit der jeweils größten diskriminativen Potenz, d. h. des Merkmals, welches die Inputmenge $\mathbf{T}^{(in)}$ so splittet, daß die entstehenden Teilmengen möglichst rein sind. Ein quantitatives Maß liefert die Informationstheorie. Ziehen wir zufällig (mit Zurücklegen) immer wieder Exemplare aus einer beliebigen Menge, hier $\mathbf{T}^{(in)}$, aus positiven und negativen Instanzen des Konzeptes, dann können wir die Folge der Lehrerentscheidungen mit J oder N als eine Zeichenkette deuten. Die Wahrscheinlichkeit für das Auftreten eines der beiden Zeichen des Alphabetes ist durch

$$p_J = \frac{n_J}{n}, \quad p_N = \frac{n_N}{n} \tag{6.2}$$

gegeben, wobei n_J bzw. n_N die Zahl der positiven bzw. negativen Instanzen des Konzeptes in der Menge $\mathbf{T}^{(in)}$ aus $n = n_J + n_N$ Instanzen darstellt.

Die bekannte Formel für die Shannonsche Entropie der Zeichenkette liefert unmittelbar den folge nden Ausdruck für die Information (pro Zeichen) dieser Zeichenkette [3]:

$$I^{(in)} = -\sum_{\alpha\in\{J,N\}} p_\alpha \log_2 p_\alpha = -p_J \log_2 p_J - p_N \log_2 p_N \tag{6.3}$$

[3] Wir nehmen den Logarithmus zur Basis 2, damit sich die Information in Bit pro Zeichen ergibt.

Wir interpretieren $I^{(in)}$ als die Inputinformation in unseren Knoten. Analog finden wir für jede der ausgegebenen Teilmengen $\mathbf{T}_i$

$$I_i^{(out)} = -\sum_{\alpha\in\{J,N\}} p_{i\alpha} \log_2 p_{i\alpha} \tag{6.4}$$

wobei $p_{i\alpha} = \frac{n_{i\alpha}}{n_i}$ mit $n_{i\alpha}$ = Zahl der positiven ($\alpha = J$) bzw. negativen ($\alpha = N$) Instanzen des Konzeptes in der Teilmenge und $n_i = n_{iJ} + n_{iN}$. Im Mittel über die Informationen der Teilmengen finden wir

$$\begin{aligned} I^{(out)} &= \sum_i \frac{n_i}{n} I_i^{(out)} \qquad (6.5)\\ &= -\sum_i \frac{n_i}{n} \sum_{\alpha\in\{J,N\}} p_{i\alpha} \log_2 p_{i\alpha} \\ &= -\frac{1}{n}\sum_i \sum_{\alpha\in\{J,N\}} n_{i\alpha} \log_2 p_{i\alpha} \end{aligned}$$

Das ist die im Mittel vom Lehrer einzubringende Information zur Klassifikation der Outputs des Knotens.

Fälle: Sind alle Teilmengen rein, d. h. alle $p_{i\alpha}$ sind entweder 0 oder 1, dann ist $I^{(out)} = 0$. Damit wird also am Output des Knotens keine Lehrerinformation mehr benötigt und alle Outputs können in einen Blattknoten münden. Im anderen Extremfall bewirkt der Knoten überhaupt keine Verbesserung der Separation von positiven und negativen Beispielen, d. h. $p_{i\alpha} = p_\alpha \ \forall i$ so daß sich durch Vergleich von (6.5) und (6.3) mit $\sum_i n_i = n$ sofort $I^{(in)} = I^{(out)}$ ergibt. Daraus können wir als Kriterium für das beste Merkmal (im Mittel reinste Menge) die Forderung

$$I^{(out)} = Min.$$

ableiten.

Der durch Einführung des Knotens erzielte Informationsgewinn (d. i. die Menge von Information, die der Lehrer weniger zur Verfügung stellen muß) ist

$$I^{(gain)} = I^{(in)} - I^{(out)}$$

und für das beste Merkmal gilt

$$I^{(gain)} = Max.$$

Das abgeleitete Kriterium ist aber keineswegs zwingend. Heuristische Verfahren zur besten Wahl des Tests (Merkmals) an jedem Knoten sind Gegenstand intensiver Forschungen, vgl. [56].

6.4 Stetige Attribute

Entscheidungsbäume arbeiten primär nur mit diskreten Attributen, genaugenommen sollte die Zahl der möglichen Attributwerte nicht zu groß sein, damit die Breite des Baumes sich in Grenzen hält. Stetige Attributwerte müssen folglich zunächst diskretisiert werden.

6.4.1 Die Intervallteilungsmethode

Das gelingt durch eine Intervallteilung des Wertebereiches des Attributes und Abbildung des Attributwertes auf die Nummer des Intervalles, in das der Wert fällt. Da die Zahl der Intervalle möglichst klein sein soll, ist die richtige Wahl der Intervallteilung von entscheidender Bedeutung für das praktische Funktionieren des Verfahrens.

Die beste Intervallteilung kann u. a. mit Hilfe der Informationstheorie gewonnen werden. Die Intervallteilung bestimmt ja wie die Inputmenge des Knotens zum betrachteten Attribut in Teilmengen aufgegliedert wird. Die Intervallteilung beeinflußt folglich unmittelbar den Informationsgewinn $I^{(gain)}$ des Knotens. Die beste Intervallteilung (im Sinne dieses informationstheoretischen Maßes) kann folglich durch Maximierung von $I^{(gain)}$ bezüglich der Intervallgrenzen gewonnen werden. Dazu kann ein beliebiges Optimierungsverfahren dienen. Zu beachten ist allerdings, daß $I^{(gain)}$ eine unstetige Funktion der Intervallgrenzen ist, denn $I^{(gain)}$ ändert sich nur genau dann, wenn sich die Zahl der vom Intervall überdeckten Instanzen verändert.

Die gefundene Aufteilung beeinflußt die Leistungsfähigkeit des induzierten Baumes oft in entscheidender Weise. Eine Diskussion des Problems und interessante Lösungen wurden kürzlich von Quinlan gegeben, s. [37].

6.4.2 Kombinationsattribute

Diese Methode modelliert die Grenzen zwischen den Domänen ausschließlich durch achsenparallele (parallel zu den Koordinatenachsen des Instanzenraumes verlaufende) Geradenstücke. Das ist besonders ineffektiv bei linear separablen Klassen, die sich durch eine schräg zu den Achsen verlaufende Gerade bzw. Hyperebene trennen lassen.

Dieser Nachteil der Intervallteilungsmethode läßt sich durch Bildung von Kombinationsattributen teilweise aufheben. Sei etwa für zwei stetige Merkmale x_1 und x_2 die Klassengrenze durch die Gerade $x_2 = ax_1 + b$ gegeben. Für die Kombinationsattribute

$$x'_1 = A_{11}x_1 + A_{12}x_2 \tag{6.6}$$

$$x'_2 = A_{21}x_1 + A_{22}x_2$$

kann man die Koeffizienten A_{ij} immer so wählen, daß die Klassengrenze im neuen Koordinatensystem parallel zu einer der beiden neuen Koordinatenachsen ist. Dann reicht die Unterteilung der relevanten Achse in zwei geeignete Teilintervalle zur Einrichtung eines exakten Entscheidungsbaumes aus.

Durch nichtlineare Kombinationsattribute etwa der Gestalt

$$\begin{aligned} x'_1 &= A_{11}x_1 + B_{11}x_1^2 + B_{12}x_1x_2 + ... + A_{12}x_2 + ... \\ &. \\ &. \\ &. \end{aligned} \tag{6.7}$$

lassen sich auch krummlinige Klassengrenzen durch einfache Intervallteilung der neuen Koordinatenachsen in den Entscheidungsbaum-Algorithmus effektiv integrieren.

Die Bildung von Kombinationsattributen ist nicht nur für reellwertige Attribute von Interesse, sondern kann auch als ein allgemeines Mittel zur Vereinfachung der Entscheidungsbäume eingesetzt werden. Das hat vor allem auch Konsequenzen für die **verständliche** Interpretation der erhaltenen Entscheidungsbäume als Regelsatz für die Klassifikation.

6.4.3 Hybride Verfahren

In der Praxis bewähren sich vor allem hybride Verfahren, die durch Kombination verschiedener Methoden entstehen. Für die Diskretisierung bietet sich die Verwendung eines **Vektorquantisierers** an, der vor allem den Vorteil hat, die Intervalle bzw. Voronoizellen so zu positionieren, daß die Instanzen optimal überdeckt werden, vgl. Kap. 4.1. Das ist vor allem für hochdimensionale Räume (viele Attribute) entscheidend wichtig.

6.5 Rauschen

Real World Daten sind fast immer verrauscht. Die ungenügende Beherrschung dieses Problems ist ein entscheidender Hemmfaktor für den praktischen Einsatz der Verfahren des maschinellen Lernens.

6.5.1 Arten des Rauschens

Wir unterscheiden zwei Arten von Rauschen, die in der Praxis meist in Kombination auftreten:

1. Beschreibungsrauschen (*description noise*): Das Zielprädikat weist jeder Klasse (positive oder negative Instanzen) im Instanzenraum jeweils eine Domäne zu. Durch fehlerhafte Attributwerte falsch beschriebene Objekte können fälschlich in die komplementäre Domäne geraten, so daß Inseln klassenfremder Instanzen in den Domänen entstehen. Rauschen verkompliziert folglich die Domänenstruktur und damit auch die aus den Beispielen gelernten Konzeptbeschreibungen. Im Extremfall stimmt ein falsch beschriebenes Objekt mit einem Objekt der Komplementärklasse überein, so daß ein konsistentes Konzept aus den Trainingsbeispielen nicht gewonnen werden kann.

2. Klassifikationsrauschen (*classification noise*): Ist die Beschreibung korrekt, aber die Klassenzuweisung (pos./neg. Instanz) durch den Lehrer fehlerhaft, dann kann der Entscheidungsbaum ebenfalls sehr kompliziert werden, da der Lernalgorithmus versucht, die fehlerhaften Beispiele mit zu integrie-

ren. Zudem verhindern durch Fehlklassifikation erzeugte widersprüchliche Beispielpaare oder –Gruppen das Erlernen eines konsistenten Konzeptes.

Scheinklassifikationen (*spurious classifications*)

Überwiegt die Rolle des Rauschens, dann reflektiert der gelernte Entscheidungsbaum nicht das unterliegende Konzept. Der Extremfall ist das Erlernen eines Kozeptes aus rein zufälligen Daten — der erlernte Entscheidungsbaum liefert dann **scheinbar** einen Satz von Regeln zur Klassifikation der Daten.

Beispiel: Erlernen des Konzeptes $SECHS$ beim Würfeln (Zahl 6 wird geworfen). Haben wir eine **endlich** große Anzahl N von Trainingsinstanzen (Würfen) und eine **hinreichend** große Anzahl von Attributen wie Farbe, Form, Material des Würfels (bei mehreren Würfeln), Name und Alter des Werfers, Vorgeschichte, ... so kann man scheinbar gültige Regeln dafür aufstellen, wann eine 6 gewürfelt wird. Der Entscheidungsbaum macht dann eben Aussagen der Qualität ”wenn Egon mit dem roten Würfel, nachdem er am Vorabend Bier getrunken hat,...”. Diese stellen sich aber schnell als unhaltbar heraus, wenn neue Traininingsinstanzen dazu kommen – das gefundene Konzept hat eine schlechte **Generalisierungsfähigkeit**.

6.5.2 Overfitting

Im allgemeinen werden sich das Rauschen und das eigentliche Konzept unterschiedlich stark in den Trainingsinstanzen abbilden. Die Forderung nach größtmöglicher Genauigkeit der Klassifikation verrauschter Daten generiert einen Entscheidungsbaum, der neben dem eigentlichen Konzept auch das Rauschen mit beschreibt. Das entspricht einer Überanpassung des Entscheidungsbaumes an die Daten, was mit einer übermäßig komplizierten Konzeptbeschreibung (schlecht interpretierbarerer Regelsatz) und verminderter Generalisierungsfähigkeit des gelernten Konzeptes bezahlt wird. Allgemeine Untersuchungen zum Problem des Overfittings finden sich in einer neueren Arbeit von Schaffer [46].

6.5.3 Pruning

Pruning ist eine Technik zum sinnvollen Umgang mit dem Rauschen, d. h. zur Vermeidung des Overfittings. Dieses kann durch die Favorisierung einfacher Konzeptbeschreibungen reduziert werden, die Kunst besteht in der richtigen Balance zwischen Einfachkeit der gelernten Konzeptbeschreibung und ausreichender Genauigkeit der Klassifikation. Das kann durch die Generalisierungsfähigkeit der gelernten Konzeptbeschreibung kontrolliert werden. Diese wird durch Aufteilung der Menge der Trainingsinstanzen in eine eigentliche Trainings– und eine Testmenge und Bestimmung des Klassifikationsfehlers auf der Testmenge gemessen. Bei kleinen Datensätzen kommt die Methode der Kreuzvalidierung zum Einsatz, vgl. Kap. 2.4.

Für das Pruning wurde eine Vielzahl von Techniken entwickelt. Beispiele:

1. Bei den manchmal Prepruning genannten Verfahren wird schon während des Lernens entschieden, ob für eine noch nicht reine Teilmenge (als Output eines vorhandenen Knotens) noch ein neuer Knoten eingerichtet werden soll oder nicht. Kriterium kann wieder der durch die Einrichtung des Knotens zu erwartende Informationsgewinn $I^{(gain)}$ sein. Liegt dieser für alle Merkmale unterhalb einer gewissen Schwelle, dann kann man auf einen starken Einfluß des Rauschens schließen, die Merkmale können als irrelevant für die Klassifikation gelten. Man richtet dann für diese Teilmenge einen Blattknoten ein, dessen Label (J/N) sich etwa nach der mehrheitlich vertretenen Klasse (pos./neg. Instanz) in der Teilmenge richtet (Majoritätsregel = Mehrheitsvotum), vgl. Abb. 6.1.

2. Beim sog. Postpruning wird der Entscheidungsbaum zuerst vollständig gelernt und danach ein Pruning, d. h. ein Entfernen bestimmter Knoten einschließlich der an ihnen hängenden Teilbäume durchgeführt. Kriterien für die Knotenauswahl liefert wieder die Informationstheorie.

3. Weitere Verfahren: Ein kostensensitives Pruningverfahren wurde in [23] eingeführt. Andere Verfahren finden sich in [32, 13, 12].

Für den Umgang mit real world Daten ist vor allem die Unsicherheit in der

Klassifikation geeignet zu berücksichtigen. Ein Zugang bietet sich über eine fuzzy Beschreibung an, s. hierzu [54].

6.6 Anwendungen

Verschiedene Anwendungen haben die Entscheidungsbaumverfahren in der **Molekularbiologie** gefunden, vgl. [2]. Zur automatischen Auswertung von **Sonar**daten wurden diese zuerst einer der aus der Signalverarbeitung bekannten Transformationen (Fourier–, Wigner-Ville bzw. Wavelet–Transformation) unterworfen und dann mittels eines Entscheidungsbaumes klassifiziert. Dieses Vorgehen unterstreicht die Wichtigkeit einer geeigneten Vorverarbeitung der Daten bevor der Entscheidungsbaum effektiv gelernt werden kann.

Entscheidungsbäume können auch mit der Methode der gentischen Programmierung gelernt werden, vgl. [26]. Eine Anwendung zur Inferenz linguistischer Regeln findet sich in [49].

In der Robotik ist die Erfassung der lokalen Umgebung eines mobilen Roboters eine wesentliche Grundlage für die Entscheidungsfindung über die nächsten Aktionen. Eine Klassifikation der Umgebungsobjekte anhand der Sonardaten mit Hilfe eines gelernten (C4.5) Entscheidungsbaumes ist in [1] dargestellt.

6.7 Vergleich mit anderen Methoden

Einen Methodenvergleich bietet die Arbeit [30], ein allgemeiner Überblick über verschiedene Methoden zur Klassifikation mit Entscheidungsbäumen findet sich in [43].

Kapitel 7

Reinforcement–Lernen

Die bisher behandelten Lernsituationen des überwachten Lernens gingen alle davon aus, daß der Lehrer zu jeder Traininsinstanz die *Soll*-Werte für den Output des lernenden Systems zur Verfügung stellt. Dies entspricht in keiner Weise der Situation in der sich z. B. ein Lebewesen befindet, das lernen muß, sich in einer ihm mehr oder weniger unbekannten Umgebung zu behaupten. In dieser Situation gibt es selten einen Lehrer, der dem lernenden System Beispiele für das Soll-Verhalten vorgeben kann. Die einzige Information die dem Lerner zur Verfügung steht ist die oft erst nach langer Zeit erfolgende Reaktion der Umgebung in Form von Belohnung oder Bestrafung für richtiges oder falsches Verhalten. Der Lerner deutet die Belohnung als Bestätigung (reinforcement) seines Verhaltens, und entwickelt Strategien, um sein *reinforcement* zu vergrößern.

Eine erste Formulierung des RLs wurde von Thorndike (1911) unter der Bezeichnung "*the law of effects*" aus dem Studium des Lernverhaltens von Tieren abgeleitet: Animals responses to environment in a given situation are adapted such that the probability for finding satisfaction if this situation occurrs again is enhanced. Die "*satisfaction*" kann als die Belohnung oder *Bestätigung* (*reinforcement*) des Lebewesens durch seine Umgebung aufgrund eines sinnvollen Verhaltens gedeutet werden. Ziel der Adaption des Lebewesens ist also die Verbesserung seiner Chancen, in jeder Situation in Zukunft ein möglichst großes Reinforcement zu realisieren.

Diese Form des Lernens, kann als eine Art des überwachten Lernens verstanden werden, allerdings bei sehr unspezifischer Information durch den Lehrer;

statt einer Gradienteninformation über Richtung und Stärke der vorzunehmenden Änderungen am System, wie sie etwa bei den in Kapitel 2 besprochenen Parameterlernverfahren zur Verfügung steht, besteht hier die Information oft nur in einem einzigen Bit (Erfolg oder Mißerfolg der Aktionen des Agenten). Ein typisches Beispiel für RL ist etwa das Schachspiel, bei dem der Spieler erst nach einer Vielzahl von Zügen eine Information - Sieg oder Niederlage - über den Wert seiner Spielstrategien bekommt. Sind Teilziele formuliert, so kann das Reinforcement auch abgestuft erteilt werden. Möglich ist auch negatives Reinforcement (Strafe) $r < 0$, falls der Lerner eine Handlung mit einem verbotenen oder unerwünschten Ergebnis ausgeführt hat.

Entsprechend der geringen Information, die das lernende System zur Verbesserung seiner Strategie aus der Umgebung zur Verfügung hat, sind effektive Lernverfahren für das RL wesentlich schwieriger zu konzipieren als im Fall des Lernens mit Lehrer, der das Sollverhalten detailliert vorgibt. Ein wesentlicher Unterschied besteht auch zwischen Lernsituationen bei denen das *reinforcement* Signal sofort zur Verfügung steht von denen eines zeitverzögerten Signals (Beispiel: Schachspiel).

Wir konzentrieren uns nach der formalen Definition und der Einführung wichtiger Hilfsfunktionen im folgenden vor allem auf das Q-Lernen, das ein allgemeines, systematisches und modellfreies Verfahren des reinforcement Lernens darstellt.

7.1 Allgemeines

Das RL ist vor allem für reaktive Agenten (Lerner) von Interesse, die also in Reaktion auf den aktuellen Zustand x, $x \in \mathbf{R}^n$ der Welt eine Aktion a, $a \in \mathbf{R}^m$ auslösen, wodurch sich der Zustand der Welt verändert. Das Gesamtsystem Welt + Agent ist formal durch die Überführungsfunktion Φ beschrieben

$$x' = \Phi\left(x, a\right) \tag{7.1}$$

Wir nehmen dabei eine Markov-Welt an, d. h. der neue Weltzustand x' allein durch den aktuellen Weltzustand x und die aktuell anliegenden Aktionen $a = (a_1, \ldots, a_m)$ gegeben. Wichtige Verallgemeinerungen sind:

- Verallgemeinerung (1): Abhängigkeit von früheren Zuständen und evtl. explizite Abhängigkeit von der Zeit (dynamische Umgebung)

$$\boxed{x'=\Phi\left(x, x^{(-1)}, \ldots, a; t\right)} \tag{7.2}$$

mit $x = x(t)$ und $x^{(-k)} = x(t - k\tau)$ ist der um k Zeitschritte zurückliegende Weltzustand.

- Verallgemeinerung (2): Die Überführungsfunktion Φ ist eine stochastische Funktion

- Verallgemeinerung (3): Die Welt enthält verborgene Parameter.

- Weltmodell i. a. nicht bekannt. Der Agent muß dann das Weltmodell mit lernen. Dieses Wissen kann durch zufallsgeleitetes Ausprobieren der Aktionen gewonnen werden.

Wir wollen im fogenden von einer diskreten Menge von Zuständen der Welt ausgehen, d.h. wir ersetzen den allgemeinen Weltzustand x durch eine diskrete Variable j.

7.2 Policy Funktion

Das Verhalten des Agenten ist durch die policy Funktion ψ definiert. Diese weist dem Perzept (Weltzustand j) die auszuführende Aktion $a \in \mathcal{A}(j)$, $\psi : j \to a$ oder

$$a = \psi(j) \tag{7.3}$$

zu. [1] Das Ausführen der Aktion a überführt den Zustand j der Welt in einen neuen Zustand j'. Damit definiert die Policy eine Trajektorie im Zustandsraum. Aufgabe ist das Auffinden einer mögichst optimalen policy Funktion ψ.

Eine wichtige Verallgemeinerung ist eine stochastische Policy, die nicht eine definierte Trajektorie sondern eine Zufallswanderung durch den Zustands- Aktionsraum erzeugt. Formal ist die Policy eine stochastische Funktion, die nur über

[1] Die policy Funktion ist gleich der Transferfunktion des Lerners in unserer bisherigen Terminologie.

die Wahrscheinlichkeiten für das Ausführen einer Aktion definiert ist:

$$p_j(a) = P(\text{Aktion}\, a \text{ im Zustand}\, j \text{ ausgewählt}) \tag{7.4}$$

Im Weltzustand j wird die Aktion a mit Wahrscheinlichkeit $p_j(a)$ ausgeführt. Entsprechend ist die Dynamik des Gesamtsystems jetzt durch die

$$\text{Übergangswahrscheinlichkeiten } P(j \to j')$$

mit j' ein möglicher Folgezustand von j, beschrieben.

7.3 Die Wertfunktion (Utility)

Wie schon erwähnt, definiert die Policy **Trajektorien** im Zustandsraum. Darunter verstehen wir eine Abfolge von Zuständen

$$j \to j' \to j'' \to \ldots \tag{7.5}$$

die durch Ausführen der Aktionen $a, a', \ldots$ entsteht. Die inkrementelle Optimierung der Policy, die in Kap. 7.5 besprochen wird, gründet sich auf die Einführung eines Maßes für den Wert eines Zustandes j. Wir definieren zunächst den Wert eines Zustandes für den Fall einer deterministischen policy Funktion $a = \psi(j)$, s. Kap. 7.3.1 und in Kap. 7.3.2 den Fall der stochastischen Policy.

7.3.1 Wertfunktion und deterministische Policy

Unter einer deterministichen Policy ist die Aktion a eine eindeutige Funktion des Zustandes j. Folglich gibt es zu jedem Startzustand genau eine Trajektorie. Die Zukunft des Systems ist (bei fester Policy) für alle Zeiten festgelegt. Der Wert eines jeden Zustandes j wird an der Menge von Reinforcement gemessen, das der Lerner unter der gegebenen Policy ψ in der Zukunft zu erwarten hat, wenn er von j aus startet und der Policy ψ folgt. Befindet sich der Lerner im Zustand j und sind $j', j'', ..., j^{(n)}, ...$ die Folgezustände, d. h. die Zustände die das System unter der Policy ψ in den folgenden Zeitschritten durchläuft, so ist der Wert V des Zustandes j durch

$$V(j) = r(j') + \gamma r(j'') + \ldots = \sum_{k=0}^{\infty} \gamma^k r\left(j^{(k+1)}\right) \tag{7.6}$$

definiert, wobei $r\left(j^{(k)}\right)$ = Reinforcement im k–ten Folgezustand.[2]. Die Größe γ in (7.6) ist die sog. discount Rate, die wegen $\gamma < 1$ zukünftiges Reinforcement geringer wichtet als das unmittelbar erhältliche. Deshalb wird ein Zustand um so höher bewertet, je schneller der Agent von j zum Ziel (Quelle für Reinforcement) gelangen kann, Beispiel s. u.

Bemerkungen:

- Die disccount Rate $\gamma < 1$ bewirkt eine Abwertung späterer Reinforcement–Erträge gegenüber früheren. Mit dem Ansatz $\gamma = e^{-\tau}$ und $\gamma^k = e^{-k\tau}$ ist ersichtlich, daß
$$\tau = -\log\gamma, \quad 0 < \gamma < 1$$
den Zeithorizont definiert, innerhalb dessen zukünftige Reinforcement–Erträge wesentlich zum Wert des Zustandes j beitragen.

Interpretation: $V(j)$ ist die Summe aller (diskontierten) R–Beiträge, die der Agent in der Zukunft "einsammeln" wird, wenn er im Zustand j startet und strikt der policy ψ folgt, Kurzformel: $V(j)$ = Wert im Zustand j. Aus (7.6) folgt

$$V(j) = r(j') + \gamma V(j') \tag{7.7}$$

oder

$$\text{Wert}(j) = \quad r(j') \quad + \quad \gamma\ \text{Wert}(j') \tag{7.8}$$

Der Wert des Folgezustandes j' kann folglich wie ein internes Reinforcement–Signal für den Ausgangszustand j gedeutet werden. Dieses erscheint aber gegenüber dem unmittelbaren Reinforcement–Signal $r(j')$ um den Faktor γ abgewertet (diskontiert).

Beispiel:

1. Sei in einem System nur ein Zielzustand j_{goal} vorhanden, d. h. $r(j) = 1$ für $j = j_{goal}$ und $r(j) = 0$ für alle andere Zustände. Wenn der Agent unter der gegebenen Policy ψ von einem Zustand j aus gerade K_j Schritte bis zum

[2] Unter einer stochastischen Policy oder in einer nichtdeterministischen Welt ist dieser Wert nur im Mittel über viele Trajektorien des Agenten definiert. Experimentell ist $V(j)$ dann so zu bestimmen, daß man immer wieder den Agenten im Zustand j startet und die Menge von Reinforcement registriert, die er unter der Policy ψ in der Folge akkumuliert.

Erreichen des Zielzustandes braucht, dann ist offensichtlich gemäß 7.6 der Wert des jeweiligen Zustandes[3] j

$$V\left(j\right) = \gamma^{\left(K_j - 1\right)} \tag{7.9}$$

Der Wert eines Zustandes nimmt offensichtlich mit der Zahl der zum Erreichen des Zielzustandes erforderlichen Schritte exponentiell ab.

2. In dem im folgenden dargestellten einfachen Labyrinth soll sich ein Agent bewegen, der in jedem Zustand (Gitterpunkt) j über vier Aktionen $\uparrow, \downarrow, \rightarrow, \leftarrow$ verfügt, d. h. gehe einen Schritt in eine der vier Richtungen N, S, O, W. Dargestellt ist immer die Aktion, die er im jeweiligen Zustand j ausführt. Die Aktion ($\leftarrow$) in der Startzelle ist nicht explizit dargestellt.

$$\begin{array}{cccccccccc}
\cdot & \cdot & \times & \times & \times & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \times & \times & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \times & \rightarrow & \rightarrow & \boxed{\mathrm{Z}} & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \rightarrow & \uparrow & \times & \times & \times & \times & \cdot \\
\times & \times & \times & \uparrow & \times & \times & \times & \times & \times & \cdot \\
\times & \times & \times & \uparrow & \times & \times & \times & \times & \cdot & \cdot \\
\cdot & \cdot & \cdot & \uparrow & \leftarrow & \boxed{\mathrm{S}} & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{array} \tag{7.10}$$

Der Agent braucht 9 Schritte vom Startzustand $\boxed{\mathrm{S}}$ zum Zielzustand $\boxed{\mathrm{Z}}$. Der Wert des Startzustandes ist folglich gleich γ^8. Im betrachteten Beispiel ist der kürzeste Weg eindeutig. In Gitterwelten wie der dargestellten gibt es aber im allgemeinen mehrere gleichlange Wege, die Wertfunktion hat dann für verschiedene, aber gleichweit vom Ziel entfernte Zustände den gleichen Wert.

[3]Wir nehmen dazu an, daß der Agent nach Erreichen des Zieles "aus dem Rennen" genommen wird, ansonsten müsste in 7.9 auch das weitere Schicksal des Agenten mit Eingang finden.

7.3.2 Wertfunktion und stochastische Policy

Zur Diskussion der Wertfunktion unter einer stochastischen Policy verwenden wir die Wahrscheinlichkeiten $P(j \to j')$ für den Übergang vom Zustand j nach j'. Die Wahrscheinlichkeit, eine bestimmte Trajektorie $j \to j' \to j'' \to \ldots$ zu beobachten ist dann durch

$$P\left(j \to j'\right) P\left(j' \to j''\right) \ldots \tag{7.11}$$

gegeben. Der Wert $V(j)$ eines Zustandes ist jetzt als Mittel über alle Trajektorien gegeben, die von j aus starten:

$$\begin{aligned} V(j) &= \sum_{j'} P\left(j \to j'\right) r\left(j'\right) + \qquad (7.12) \\ &\quad \gamma \sum_{j'j''} P\left(j \to j'\right) P\left(j' \to j''\right) r\left(j''\right) + \ldots \end{aligned}$$

und damit

$$V(j) = \sum_{j'} P\left(j \to j'\right)\left(r\left(j'\right) + \gamma V\left(j'\right)\right) \tag{7.13}$$

oder

$$V(j) = \left\langle r\left(j'\right) + \gamma V\left(j'\right)\right\rangle_{j \to j'} \tag{7.14}$$

wobei $\langle \ldots \rangle_{j \to j'}$ ist Mittel über alle Folgezustände j', die von j aus erreicht werden können.

7.4 Passives Erlernen der Wertfunktion

Das Gebirge der Werte V über den Zuständen j ist offensichtlich chrakteristisch für jede Policy. Die optimale Policy ist dadurch ausgezeichnet, daß von allen Zuständen aus der kürzeste Weg zum Ziel der des steilsten Aufstieges ist. Die Wertfunktion ist allerdings i. a. nicht explizit bekannt. Ein Lernalgorithmus für $V(j)$ für das inkrementelle Erlernen dieses Wertgebirges bei **vorgegebener** Policy ergibt sich logisch aus den folgenden Schritten:

- Initialisierung: $V(j) = 0 \quad \forall j$. Zufällige Auswahl eines Startzustandes $j = \mathit{random}$.

- Idee: Gleichung (7.7) nur für die exakte Wertfunktion richtig. Fehlerfunktion für das Lernen (deterministische Policy):

$$E = \left(r\left(j'\right) + \gamma V\left(j'\right) - V\left(j\right)\right)^2 \tag{7.15}$$

- E ist Funktion des Wertes $V(j)$. Gradientenabstieg ergibt die Lernregel

$$\Delta V\left(j\right) = \varepsilon\left[r\left(j'\right) + \gamma V\left(j'\right) - V\left(j\right)\right] \tag{7.16}$$

  Der Update von $V(j)$ wird ausgeführt, wenn der Weltzustand j durch Ausüben einer Aktion a in den neuen Zustand j' überführt wurde. Nach Erreichen eines Zielzustandes[4] erfolgt Neuinitialisierung (Neustart in einem zufällig gewählten Zustand j)

- Bei geeigneter Abkühlung der Lernrate ε gilt die Lernregel (7.16) auch für eine stochastische Policy (s. Kap. "Stochastische Approximation")

Mit der Update-Regel (7.16) kann die Wertfunktion zu einer vorgegebenen Policy durch wiederholtes Ablaufen aller Trajektorien inkrementell gelernt werden.

7.5 Optimierung der Policy durch Wertiteration

Die Wertfunktion $V(j)$ ist wie schon erwähnt bei fester Policy eindeutig festgelegt. Umgekehrt das Ziel auf kürzestem Wege, wenn er in jedem Zustand j genau die Aktion $a^* \in \mathcal{A}(j)$ auswählt, die ihn am weitesten an das Ziel heran– und somit in den besten der ihm erreichbaren Zustände überführt[5]. Sei j'_a der Folgezustand von j

$$j \quad {}_a\!\longrightarrow \quad j'_a \tag{7.17}$$

[4] Wenn der Agent nach Erreichen des Zielzustandes neu gestartet wird, dann gibt es für j^{goal} keinen Folgezustand und damit auch keinen Update für $V\left(j^{goal}\right)$, so daß immer $V\left(j^{goal}\right) = 0$ bleibt.

[5] Falls zwei oder mehrere gleichgute Zustände von j aus erreichbar sind, ist 7.18 so zu interpretieren, daß es die gleichguten Aktionen zur Auswahl stellt.

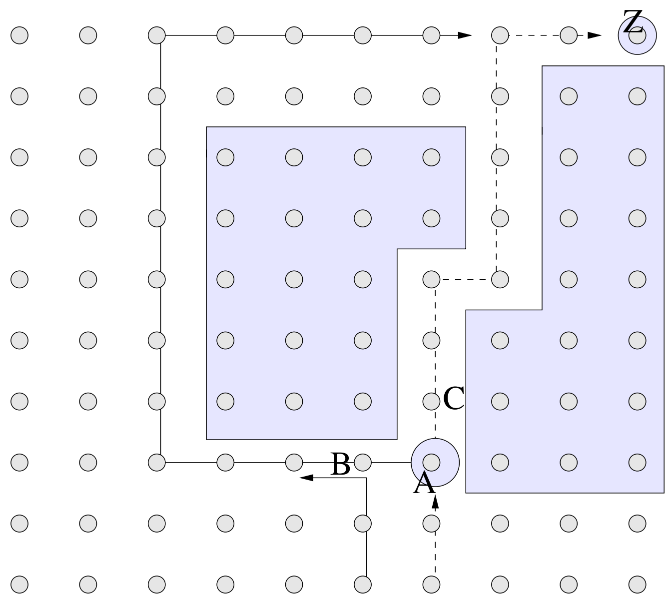

Abbildung 7.1: Beispiel für passives und aktives Erlernen eines Wertgebirges. Wird unter einer vorgegebenen (stochastischen) Policy der Weg durch den Korridor sehr viel seltener begangen als der Weg um das Hindernis herum, so geht wegen $V(A) = \gamma p_{A \to B} V(B) + \gamma p_{A \to C} V(C)$ in den Wert von Zustand A der Wert von Zustand C praktisch nicht ein. Damit ist $V(A) \approx \gamma V(B)$, also $V(A) < V(B)$ obwohl der Weg über C zum Ziel Z von A aus kürzer und somit $V(C)$ wesentlich größer als $V(B)$ ist. Das unter der aktuellen Policy **passiv** erlernte Wertgebirge ist folglich suboptimal. Die Verhältnisse ändern sich, wenn die Policy durch die Regel 7.19 bestimmt wird. Dann werden Übergänge in Zustand C häufiger ausgeführt, was wegen $V(C) > V(B)$ unter der Lernregel (7.16) eine allmähliche Wersteigerung von A und in Folge auch seiner Nachbarn nach sich zieht. Nach Abkühlung (T=0) und Erreichen des Gleichgewichtszustandes spiegelt dann das Wertgebirge die Entfernungen zum Zielzustand richtig wider.

der durch Ausführen einer beliebigen der Aktionen $a \in \mathcal{A}(j)$ erreicht. wird. Dann ist folglich (vgl. 7.3)

$$\psi(j) = \arg\max_a V(j'_a) \tag{7.18}$$

Die Policy (7.18) entspricht dem Prinzip des steilsten Aufstieges auf dem Wertgebirge über den Zuständen. Wird nun gleichzeitig die Wertfunktion nach der Update-Regel 7.16 weiter upgedatet, so kann sich eine zunehmende Verbesserung der Policy ergeben, wobeidie Konvergenz des Verfahrens gegen die optimale Policy allerdings nicht garantiert werden kann. Außerdem vor allem in der Früphase der Optimierung die Policy stochastisch sein, um die erforderliche Exploration des Zustands– Aktionsraumes sicherzustellen.

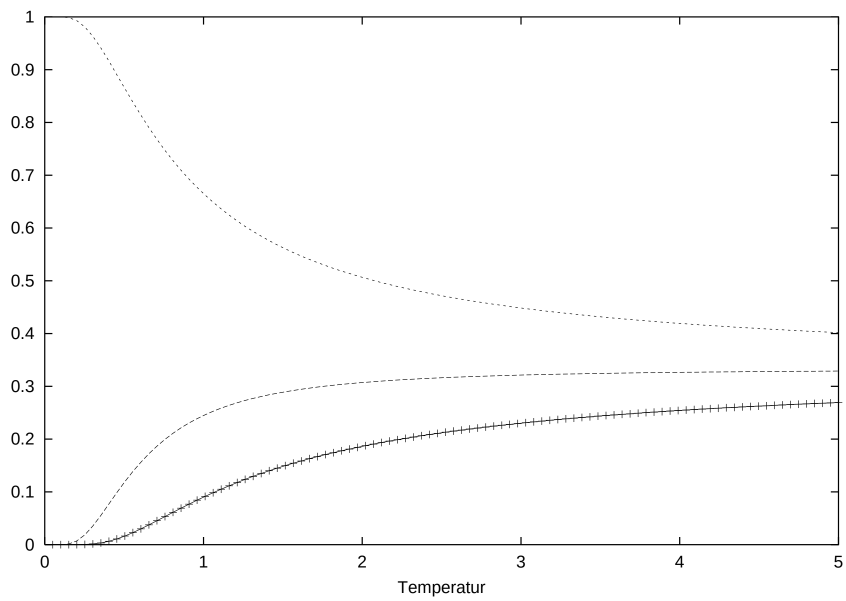

Abbildung 7.2: Boltzmannverteilung der Wahrscheinlichkeiten (7.19) als Funktion der Temperatur für drei Werte $V = 1, 2, 3$ der Zielzustände. Für hohe Temperaturen konvergieren alle Wahrscheinlichkeiten gegen den gleichen Wert ($p = 1/3$). Für $T \to 0$ setzt sich die zu $V = 3$ gehörige Aktion durch.

Eine auf der Wertfunktion beruhende Definition der stochastischen Policy ist in eleganter Weise durch die Boltzmannverteilung der Wahrscheinlichkeiten gegeben.

$$p_j\left(a\right) = \frac{e^{V(j_a')/T}}{\sum_{j'} e^{V(j')/T}} \tag{7.19}$$

Diese Policy ist das stochastische *Pendant* des Prinzips “steilster Aufstieg”, bei dem immer die beste der verfügbaren Aktionen ausgeführt wird. Jetzt gilt für die Aktionswahl das Wahrscheinlichkeitsprinzip wobei die fiktive Temperatur T die Schärfe der Auswahl bestimmt, s. Abb. 7.2. Für $T \to 0$ gewinnt man die *greedy* Strategie (steilster Aufstieg) zurück.

Der Optimierungsprozeß besteht aus den Schritten

1. Initialisierung: $V(j) = 0 \quad \forall j$. Bestimme Startzustand j des Agenten.

2. Wähle eine Aktion a gemäß (7.19) und führe diese aus.

3. Update

   $$\Delta V\left(j\right) = \varepsilon\left[r\left(j'\right) + \gamma V\left(j'\right) - V\left(j\right)\right] \tag{7.20}$$

   wobei j' der erreichte Zustand ist.

4. $j \leftarrow j'$. GOTO 2.

Dieser Prozeß wird mit einer ausreichend hohen Temperatur (Explorationsrate) begonnen und im Sinne des simulated annealing hinreichend langsam abgekühlt.

Durch die Abkühlung bei gleichzeitigem Lernen der Wertfunktion verändert sich nach (7.19) die Policy und damit das Wertgebirge ständig. Es ist ja

$$V\left(j\right) = \left\langle V\left(r(j') + \gamma j'\right)\right\rangle \tag{7.21}$$

Bei hoher Temperatur tragen die Werte aller von j aus erreichbaren Zustände j' etwa gleich stark zum Wert von j bei während für kleinere Temperaturen $V(j)$ zunehmend durch den besten dieser Zustände bestimmt ist. Die Favorisierung der Übergänge in besser bewertete Zustände erhöht wegen (7.21) $V\left(j\right) = \gamma\left\langle V\left(j'\right)\right\rangle$ den Wert des Zustandes j. Das passiert für alle Zustände bis sich ein neues (Quasi–) Gleichgewicht des Wertgebirges zur Temperatur T einstellt. Hinreichend langsames Abkühlen produziert in der Grenze ein Wertgebirge, das der optimalen Policy entspricht. Bei zu schneller Abkühlung kann es allerdings leicht passieren, daß der steilste Aufstieg im so erhaltenen Wertgebirge nicht immer den kürzesten

Weg zum Ziel ergibt. Wurden kürzere Pfade nicht hineichend oft begangen, dann werden sie im Update (7.20) nur selten berücksichtigt und folglich im Wertgebirge unterrepräsentiert sein, vgl. Abb. 7.1. **Bemerkung:** Die Aktionswahl nach Wert der Folgezustände setzt ein Weltmodell voraus, d. h. Kenntnis der Überführungsfunktion

$$j' = \Phi(j, a)$$

für die Weltzustände.

7.6 Q–Lernen

Die Bestimmung der richtigen Aktion aus der Wertfunktion nach (7.18) bzw. (7.19) setzt die Kenntnis der Werte V der Folgezustände j'_a für alle im Zustand j verfügbaren Aktionen $a \in \mathcal{A}(j)$ voraus. Diese Information über die Wirkung einer jeden Aktion $a \in \mathcal{A}(j)$ steht im allgemeinen nicht zur Verfügung sondern muß im Laufe des Lernens durch Ausprobieren erst erworben werden. Zur Speicherung dieser Information wurde von Watkins die sog. Q–Funktion eingeführt, die im Gegensatz zur Wertfunktion $V(j)$ nicht nur den Zustand j sondern ein Zustands–Aktionspaar (ZAP) (j, a) bewertet. Dieser Wert $Q(j, a)$ ist ein Maß für das (diskontierte) Reinforcement, das der Agent in Zukunft aufsammeln wird, wenn er im Zustand j gerade $a \in \mathcal{A}(j)$ ausführt und danach der Policy ψ folgt. Bis auf den ersten Term ist das zukünftige (diskontierte) Reinforcement durch (7.6) gegeben, das erste Glied der Summe in (7.6) muß allerdings durch das Reinforcement ersetzt werden, das der Agent im durch die Aktion a erreichten Folgezustand j'_a erhält, also

$$Q(j, a) = r(j'_a) + \sum_{k=1}^{\infty} \gamma^k r\left(j^{(k+1)}\right) \tag{7.22}$$

Mit 7.18 folgt die Beziehung

$$V(j) = \max_a Q(j, a) \tag{7.23}$$

und

$$\psi(j) = \arg\max_a Q(j, a) \tag{7.24}$$

Im folgenden **Beispiel** ist im Unterschied zu 7.10 nicht nur die Trajektorie dargestellt, die sich bei der Wahl der Aktion $\leftarrow$ im Startzustand $\boxed{\mathrm{S}}$ erghibt, sondern auch die aus der Wahl $\downarrow$ folgende Trajektorie (Ausschnitt aus dem Labyrinth)

$$\begin{array}{cccccccccc} \times & \times & \times & \uparrow & \times & \times & \times & \times & \cdot & \cdot \\ \cdot & \cdot & \cdot & \uparrow & \leftarrow & \boxed{\mathrm{S}} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \uparrow & \leftarrow & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array} \tag{7.25}$$

Diese überführt den Agent in den südlich von $\boxed{\mathrm{S}}$ gelegenen Zustand. Die Trajektorie ist folglich zwei Schritte länger, und damit ist $Q\left(\boxed{\mathrm{S}}, \downarrow\right) = \gamma^2 Q\left(\boxed{\mathrm{S}}, \leftarrow\right) = \gamma^2 V\left(\boxed{\mathrm{S}}\right)$. Wegen der Gleichberechtigung verschiedener Trajektorien (s.o.) unter der Policy ψ hätte der Agent auch in dem "südlich" von $\boxed{\mathrm{S}}$ gelegenen Zustand wieder umkehren können,

$$\begin{array}{cccccccccc} \times & \times & \times & \uparrow & \times & \times & \times & \times & \cdot & \cdot \\ \cdot & \cdot & \cdot & \uparrow & \leftarrow & \boxed{\mathrm{S}} & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \uparrow\downarrow & \cdot & \cdot & \cdot & \cdot \end{array}$$

mit dem gleichen Ergebnis $Q\left(\boxed{\mathrm{S}}, \downarrow\right) = \gamma^2 Q\left(\boxed{\mathrm{S}}, \leftarrow\right)$ für das Verhältnis der Q–Werte.

Die Regel 7.24 bestimmt die auszuführende Aktion in jedem Zustand eindeutig. Die vor allem in der Frühphase des Lernens erforderliche Exploration des Zustands–Aktionsraumes (ZAR) wird dagegen durch eine stochastische Policy realisiert, wobei die Aktion a mit einer Wahrscheinlichkeit $p_a(j)$ ausgeführt wird. Die Wahrscheinlichkeiten lassen sich etwa analog zu (7.19) durch eine Boltzmannverteilung in Termen der Q–Funktion definieren

$$p_a(j) = \frac{e^{Q(j,a)/T}}{\sum_{a'} e^{Q(j,a')/T}} \tag{7.26}$$

Die fiktive Temperatur T bestimmt die Explorationsrate, die wieder im Laufe des Lernens "abgekühlt" wird, für $T \to 0$ wird schließlich der deterministische Fall erreicht, d. h.

$$T \to \infty: \quad p_a(j) \to \begin{cases} 1 & f\ddot{u}r \quad a = \arg\max Q(j,a) \\ \\ 0 & sonst \end{cases} \tag{7.27}$$

Die Q–Funktion legt offensichtlich des Verhalten des Systems vollständig fest. Das Q–Lernen, d. h. die schrittweise inkrementelle Verbesserung einer Anfangsschätzung der Q–Funktion wird durch den Algorithmus von Watkins [53] realisiert.

Der Algorithmus des Q–Lernens

1. Initialisierung der Q–Funktion, z. B. $Q(j,a) = 0\,\forall j,a$ oder $Q(j,a) = random\,\forall j,a$. Start des AR in einem beliebigen Zustand j.

2. Im aktuellen Zustand j: Auwahl einer Aktion a gemäß der Wahrscheinlichkeitsverteilung $\{p_a(j)\}$.

3. Ausführung von a: AR führt Übergang von j in den Folgezustand j' durch[6].

4. **Update** $Q(j,a)$ für das ZAP (j,a) gemäß[7]

$$\Delta Q(j,a) = \epsilon\left[r(j') + \gamma V(j') - Q(j,a)\right] \tag{7.28}$$

wobei $r(j')$ das im Folgezustand j' erhaltene Reinforcement und

$$V(j') = \max_{a'} Q(j',a') \tag{7.29}$$

der Wert des Folgezustandes ist.

5. j' wird neuer aktueller Zustand

6. Abbruch? GOTO 2.

Die Konvergenz des Algorithmus (unter einer geeigneten Abkühlungsstrategie für ϵ und T) gegen die optimale Strategie kann exakt gezeigt werden, s. [53].

Bemerkungen:

[6]Der Fall $j' = j$ bedarf in den praktischen Anwendungen besonderer Aufmerksamkeit.

[7]ΔQ ist die Änderung (Inkrement) von Q im Updateschritt (Lernschritt).

- Der Algorithmus kann als ein Gradientenverfahren verstanden werden. Ausgangspunkt ist die Überlegung, daß $Q\left(j,a\right)$ die Menge von (diskontiertem) Reinforcement ist, das der Agent einsammeln wird, wenn er in j startend zunächst die Aktion a ausführt und dann der Policy ψ folgt:

$$\begin{aligned} Q\left(j,a\right) &= r\left(j'_a\right) + \left\langle \sum_{k=1}^{\infty} \gamma^k r\left(j^{(k+1)}\right) \right\rangle && (7.30) \\ &= r\left(j'_a\right) + \gamma \left\langle Q\left(j',a'\right)\right\rangle && (7.31) \\ &= r\left(j'_a\right) + \gamma V\left(j'\right) && (7.32) \end{aligned}$$

(Mittelung über alle Trajektorien bzw. über alle Folgezustände j' von j und Aktionen a'. j'_a ist Folgezustand von j nach Ausführung von a.) Im Algorithmus verbessern wir inkrementell eine Schätzung dieser Funktionswerte. Für das aktuelle Zustands–Aktionspaar (j,a) ist der Fehler für unsere Schätzung der zugehörigen Q–Werte

$$E = \frac{1}{2}\left(r\left(j'_a\right) + \gamma \left\langle Q\left(j',a'\right)\right\rangle - Q\left(j,a\right)\right)^2$$

Daraus ergibt sich eine Lernregel durch Gradientenabstieg, d. h. Minimierung des Fehlers bezüglich des Wertes von $Q(j,a)$. Eine Beschleunigung des Lernens erzilent man durchdie Ersetzung

$$\left\langle Q\left(j',a'\right)\right\rangle \rightarrow \max_{a'} Q\left(j',a'\right)$$

Daraus folgt unmittelbar der Update in (7.28).

- In der Praxis erfordert die optimale Abkühlungsstrategie zu viel Zeit. Aus praktischen Gründen wird deshalb oft nicht bis $T = 0$ abgekühlt. Trotz stochastischer Aktionswahl liefert die Definition

$$a = \arg\max_{a'} Q\left(j,a'\right) \qquad (7.33)$$

(auszuführende Aktion ist die mit dem höchsten Q–Wert) immer eine deterministische Policy. Diese ist oft schon die optimale Policy auch wenn noch $T > 0$.

7.7 Ein neues Verfahren zur direkten Wertiteration.

Einfacher zu implemetieren und in verschiedener Hinsicht dem Q–Lernen überlegen ist ein von uns entwickeltes Verfahren zur direkten Wertiteration [10]. Dieses verzichtet auf die Einführung einer Bewertung von Zustands-Aktionspaaren sondern verbessert stattdessen inkrementell direkt eine Schätzung des Wertes $V(j)$ der Zustände und der Wahrscheinlichkeiten $p_a(j)$ für die Aktionen selbst. Wesentlicher Unterschied ist die Update–Regel, s. Punkt (7.34).

Der Algorithmus der direkten Wertiteration

1. Initialisierung der V–Funktion und aller Wahrscheinlichkeiten z. B. setze man $V(j) = 0\,\forall j$ und alle Wahrscheinlichkeiten einander gleich. Start des AR in einem beliebigen Zustand j.

2. Im aktuellen Zustand j: Auwahl einer Aktion a gemäß der Wahrscheinlichkeitsverteilung $\{p_a(j)\}$.

3. Ausführung von a: AR führt Übergang von j in den Folgezustand[8] j' durch.

4. **Update**
$$\begin{aligned} \tau\Delta V(j) &= r(j') + \gamma V(j') - V(j) \\ \tau_p \Delta p_a(j) &= r(j') + \gamma V(j') - V(j) \end{aligned} \tag{7.34}$$

5. j' wird neuer aktueller Zustand

6. Abbruch? GOTO 2

Die Parameter sind τ die Zeitkonstante ($1/\tau$ die Lernrate) für das Lernen und $\tau_p = K\tau$ wobei K in der Größenordnung $K = 10$ fest gewählt werden kann. Typische Werte für die Zeitkonstante liegen zwischen 5 und 100.
Bemerkung: In der Praxis (also mit den angegebenen Zeitskalen) muß "von außen" gesichert werden, daß immer $0 < p_\alpha < 1$ gilt.

[8] Der Fall $j' = j$ bedarf in den praktischen Anwendungen besonderer Aufmerksamkeit.

Das Verfahren zeichnet sich gegenüber dem Q–Lernen durch eine Reihe von Vorteilen aus. Diese betreffen (i) die Selbsteinstellung der Explorationsrate, d. h. der Übergang zur deterministischen Strategie wird nicht global geregelt sondern stellt sich anhand der lokalen Unterschiede in den Werten der angesteuerten Folgezustände individuell von selbst ein. Damit ist die Abkühlungsstrategie des Q-Lernens eingespart worden. (ii) Die Übernahme von Vorwissen erfolgt durch die Vorgabe von Werten und Wahrscheinlichkeiten der einzelnen Zustände bzw. Aktionen und kann damit unmittelbar etwa aus vorhandenem Regelwissen übernommen werden.

Alle bisherigen Tests (Labyrintproblem, cart-centering und die Lösung eines Targeting-Problems der Chaoskontrolle) belegten die funktionelle Äquivalenz der beiden Verfahren, insbesondere erzeugten beide die gleiche Wertfunktion und die gleiche deterministische Strategie.

Der Ursprung des neuen Lernverfahrens

Die Gleichungen sind eine zeitdiskrete, stochastische Approximation der Fisher-Eigen Gleichungen der präbiotischen Evolution, die eine Selektionsdynamik $\dot{p}_i = \epsilon\,(F_i - F)\,p_i$ einer Population von N Spezies $i \in \{1, \ldots, N\}$ beschreiben, die mit relativer Häufigkeit p_i auftreten, wobei F_i die Fitneß der Spezies i und $F = \sum_i p_i F_i$ die mittlere Fitneß der Population bedeuten. Bei beliebiger Initialisierung der p_i wird asymptotisch die Spezies mit der größten Fitneß selektiert, d. h. $p_i \to 1$ für $i = \arg\max p_i$. Die Konvergenzgeschwindigkeit nimmt dabei mit steigender Variationsbreite der Fitneßwerte logarithmisch – und damit nur schwach – zu.

Für die Wertiteration interpretieren wir die in jedem Zustand j verfügbaren Aktionen $a \in \mathbf{A}$ als die Individuen einer Population, die Wahrscheinlichkeiten als deren relative Häufigkeiten und $r\,(j') + \gamma V\,(j')$ als die Fitneß der Aktion a im Zustand j. Diese Bewertung der Fitneß führt zu einer Verkoppelung der Populationen und bewirkt letztendlich die zeitliche Rückdiffusion des Reinforcement-Signals längs der optimalen Trajektorien.

Kapitel 8

Evolutionsstrategien und Genetische Programmierung

8.1 Allgemeines

Bisher haben wir immer einzelne Agenten (Lerner) betrachtet und für diese geeignete Lernverfahren gesucht, die die Leistung des Agenten in der konkreten Aufgabenstellung optimieren. Im folgenden wollen wir uns mit Lernverfahren befassen, die mit einem Ensemble bzw. einer Population von Agenten operieren, die sich nach den Prinzipien der biologischen Evolution entwickeln. Dabei entstehen durch Wechselwirkungen zwischen den Individuen kollektive Effekte, die einen oft entscheidenden Effektivitätsvorteil bringen, s. z.B B. [48].

8.1.1 Problemstellung

Wir betrachten zunächst wieder einen Agenten der mit der Welt wechselwirkt,

$$\boxed{\begin{array}{c} Agent \\ \\ \vec{w} \end{array}} \quad \begin{array}{c} \rightarrow \\ \leftarrow \end{array} \quad \boxed{Welt} \tag{8.1}$$

wobei die Eigenschaften des Agenten wie bisher durch $w = (w_1, w_2, \ldots, w_p)$ festgelegt seien. In Abhängigkeit von der konkreten Anwendung (in Klammern dazugefügt) kann w u. a. folgende Bedeutungen haben:

- w ist der reellwertig oder binär kodierte Parametervektor eines technischen Systems
- w ist ein Satz von Produktionsregeln (*classifier systems*)
- w ist ein Computerprogramm, das das Verhalten des Agenten programmiert (s. u. Genetische Programmierung)

In biologischer Sprechweise ist w das Chromosom, das den Genotypen des Agenten vollständig beschreibt. Der Phänotyp ist der Agent und sein Verhalten wird durch den Genotyp in eindeutiger Weise spezifiziert (bei einer technischen Anwendung ist der Phänotyp die Maschine).

Ein Beispiel: Evolution eines abstrakten Automaten

In Abb. 8.1 sehen wir einen abstrakten Automaten, der etwa als Controller einer virtuellen Ameise dienen kann, die sich in der in Abb. 8.2 gezeigten Gitterwelt bewegen und dabei so viel wie möglich von den food pellets einsammeln soll, vgl. [27].

Der in Abbildung 8.1 gezeigte Automat kann ebenso durch die folgende Tabelle der Übergänge zwischen seinen Zuständen bei gegebenen Inputs definiert werden:

Nr.	Aktueller Zustand	Input	Neuer Zustand	Operation
1	00	0	01	10=Right
2	00	1	00	11=Move
3	01	0	10	01=Left
4	01	1	00	11=Move
5	10	0	11	01=Left
6	10	1	00	11=Move
7	11	0	00	10=Right
8	11	1	00	11=Move

(8.2)

Wir können diese Tabelle und damit das Verhalten in folgender Weise in einem Bitstring kodieren:

$$w = (00\,0110\,0011\,1001\,0011\,1101\,0011\,0010\,0011)$$

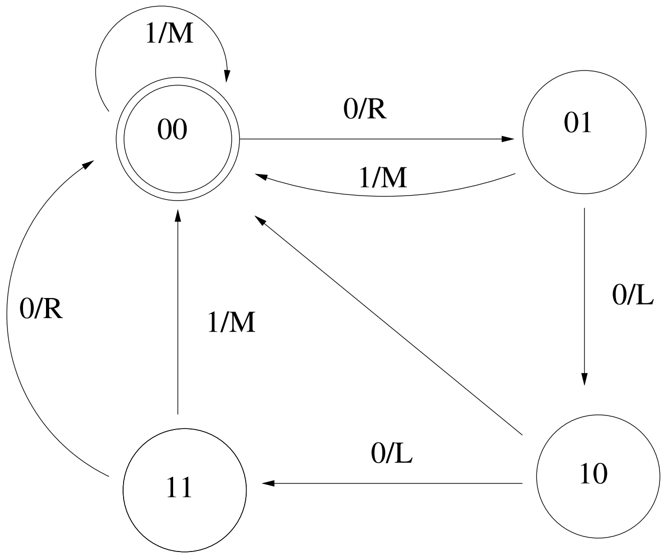

Abbildung 8.1: Der gezeigte Automat besitzt vier innere Zustände, mit Zustand 00 als Startzustand. Er sieht zwei mögliche Inputsituationen "no-food-ahead" kodiert mit 0 und "food-ahead" kodiert mit 1. In Abhängigkeit vom jeweiligen Input und seinem eigenen inneren Zustand führt der Automat eine der möglichen Aktionen $Move$ = Ein Schritt vorwärts, $Right$=Einen Schritt nach rechts oder $Left$ = Ein Schritt nach links aus. Die Übergänge zwischen den einzelnen Zuständen tragen Label, die den aktuellen Input und die zugehörige Aktion kodieren.

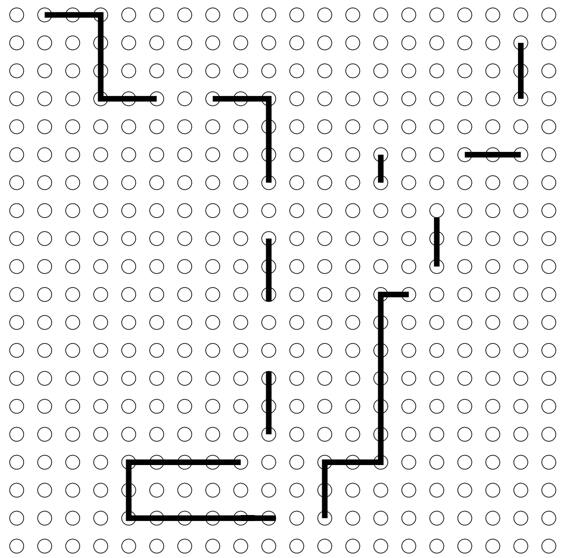

Abbildung 8.2: Eine Gitterwelt mit einer mehrfach unterbrochenen Futterspur

wobei die ersten beiden Bitpositionen den Inputzustand kodieren (hier fest der Zustand 00) und die weiteren jeweils 4 Bit langen Blöcke für jeden der insgesamt 8 Zustands/Input Situationen der Tabelle den neuen Zustand und die zugehörige Aktion aufführen, d. h.

$$\underbrace{00}_{Input} \quad \underbrace{0110}_{1} \quad \underbrace{0011}_{2} \quad \underbrace{1001}_{3} \quad \underbrace{0011}_{4} \quad \ldots$$

Die Bitkette legt also das Verhalten des Automaten in eindeutiger Weise fest, die Anwendung des genetischen Algorithmus erzeugt eine Evolution des Automaten.

8.2 Evolution

Wie schon erwähnt, erfolgt die Evolution in einer Population P von Individuen, deren jedes durch seinen jeweiligen Genotyp w definiert ist. Die Evolution kann als eine Abfolge von Generationen verstanden werden, wobei die Population in jeder Generation einer Reihe von Operationen unterworfen wird. Wir betrachten explizit die folgenden

8.2.1 Elemente der Evolution

Fitneßbestimmung:

In jeder Generation muß die Fitneß aller Individuen bestimmt werden. Dazu wird jeder Genotyp $w \in P$ in seinen Phänotyp übersetzt und dessen Leistung in der konkreten Aufgabe ermittelt. In einem geeigneten Wertsystem ausgedrückt ist das die Fitneß $Q(w)$ des Individuums $w \in P$.

Reproduktion:

Beim Generationenwechsel erfolgt die Vermehrung der Individuen entweder durch Kopie der elterlichen Genotypen oder durch Kreuzung (cross over) je zweier oder mehrerer Eltern (s. u.).

Selektion

Die fitneßgewichtete Bevorzugung "besserer" Individuen bei der Reproduktion bewirkt einen Selektionsdruck in Richtung höherer Fitneßwerte.

Genetische Operationen

Genetische Operatoren verändern das Erbgut und treiben damit die Evolution voran.

1. **Mutation** Die Mutation nimmt zufällige Veränderungen der Genotypen vor. Bei der Kodierung des Genotypen durch eine Bitkette (eigentliche genetische Algorithmen) wird jedes Bit mit einer Wahrscheinlichkeit p geflippt. Das kann z. B. fogendermaßen formalisiert werden:

$$w_i := z(w_i - \frac{1}{2}) + \frac{1}{2} \tag{8.3}$$

wobei $w_i \in 0, 1$ und $z \in \{+1, -1\}$ ist eine Zufallszahl

$$P\,(z = 1) = p \quad \text{und} \quad P\,(z = -1) = 1 - p \tag{8.4}$$

Für $p = 1$ findet keine Mutation statt, für $p = 0$ wird im Mittel in jedem zweiten Fall das Allel geflippt ($0 \rightarrow 1$ bzw. $1 \rightarrow 0$). Die Größe $m = 1 - p$ heiß deshalb die Mutationsrate.

Bei reellwertigen Komponenten von w erfolgt die Mutation durch Addition eines Inkrements Δw_i

$$w_i := w_i + \Delta w_1 \tag{8.5}$$

wobei generisch das Inkrement eine normalverteilte Zufallszahl mit Verteilung

$$p(\Delta w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\Delta w)^2}{2\sigma^2}} \tag{8.6}$$

ist. Der Parameter σ regelt die Schrittweite der Veränderung und damit die Mutationsrate.

2. **Crossing**: Das Crossing spielt in der Philosophie der genetischen Algorithmen eine besondere Rolle. Insbesondere J. Holland unterstreicht, daß erst mit der Einführung des Crossings ein der reinen Zufallssuche qualitativ überlegenes Verfahren gefunden war. Die Essenz des Crossing-Effekts kommt im sog. Schema–Theorem zum Ausdruck, das die genetischen Algorithmen als Verfahren zum Erkennen und Absuchen günstiger Hyperebenen (Teilmengen) des Suchraumes ausweist, s. [27, 21].

   Einfachster Fall ist das Ein-Punkt–Crossing

$$\begin{array}{c} w_1 = \begin{array}{|c|c|c|c|c|c|c|}\hline a_1 & a_2 & \dots & a_k & a_{k+1} & \dots & a_L \\ \hline\end{array} \\ \updownarrow \\ w_2 = \begin{array}{|c|c|c|c|c|c|c|}\hline b_1 & b_2 & \dots & b_k & b_{k+1} & \dots & b_L \\ \hline\end{array} \end{array} \Rightarrow \begin{array}{c} w_1' = \begin{array}{|c|c|c|c|c|c|c|}\hline a_1 & a_2 & \dots & a_k & b_{k+1} & \dots & b_L \\ \hline\end{array} \\ \\ w_2' = \begin{array}{|c|c|c|c|c|c|c|}\hline b_1 & b_2 & \dots & b_k & a_{k+1} & \dots & a_L \\ \hline\end{array} \end{array} \tag{8.7}$$

   wobei a_i und b_i die Komponenten des Vektors w_1 bzw. w_2 sind. Diese können beliebig binär– oder reellwertig sein.

   Analoge Darstellungen können auch für die Operatoren eines Mehrpunktcrossings angegeben werden.

3. **Inversion**: Ein zufällig ausgewählter Abschnitt des Vektors w wird invertiert.

4. **Gen-Duplikation**: Als eine neue, besonders erfolgversprechende genetische Operation wird derzeit die Gen-Duplikation diskutiert. Diese Überlegungen fußen auf der provokativen Hypothese von Sagan, der schon seit den siebziger Jahren dieses seltene Ereignis in der Evolution als ihr eigentlich progressives Element propagiert.

5. In der einschlägigen Literatur wird ein Vielzahl weiterer Operationen diskutiert, die sich auf veschiedenen Domänen unterschiedlich bewährt haben.

8.3 Strategien der Evolution

Der Gang der Evolution wird wesentlich durch den Reproduktionsprozeß und das Selektionsverfahren bestimmt. In den zuerst von Rechenberg [38, 39] eingeführten sog. mehrgliedrigen Evolutionsstrategien erzeugen in jeder Generation μ Eltern λ Nachkommen. Grundlegend unterscheiden wir bei der Selektion zwischen den sog. Komma– und Plus– Strategien. Bei "Plus-" Strategien erfolgt die Selektion aus der Menge der Eltern **und** Kinder, so daß die Eltern überleben können, solange sie nicht von ihren Kindern übertroffen werden. In den (μ, λ) Evolutionsstrategien rekrutieren sich hingegen die Eltern der nächsten Generation ausschließlich aus den μ besten ihrer λ Nachkommen ($\lambda > \mu$). Damit haben die Eltern nie eine Chance, ihr vielleicht besseres Erbgut weiterzugeben. Die Komma–Strategie hat folglich einen mehr explorativen Charakter während die Plus–Strategie eher konservativ ist und schon erworbenes Wissen über die "besseren" Gebiete des Suchraumes nicht preisgibt.

Wir stellen im folgenden verschiedene Evolutionsstrategien für den Fall reellwertiger w_i schematisch dar.

8.3.1 $(1+1)$–Evolutionsstrategie

$\boxed{1.}$ $t = 0:$

$Initialisierung \quad \mathcal{P}_t = \left(w_t^E\right)$ (8.8)

$\Downarrow$

$\boxed{2.}$ $\boxed{\text{Mutiere} \quad w_t^N = w_t^E + \sigma \, \Delta w_t}$

$\Downarrow$

Selektiere :

$w_{t+1}^E = w_t^N, \text{ falls } \quad F\left(w_t^N\right) > F\left(w_t^E\right)$

$w_{t+1}^E = w_t^E \quad \text{sonst}$

$\Downarrow$

$\boxed{\text{Neue Population } \mathcal{P}_{t+1}}$

$\Downarrow$

$\boxed{\text{t=t+1}}$

$\Downarrow$

$\boxed{\text{Abbruch?}}$

$\Downarrow$

$\boxed{\text{GOTO 2.}}$

Die Mutation erfolgt mit der Wahrscheinlichkeitsverteilung der Δw_i

$$p\left(\Delta w_i\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\Delta w_i\right)^2\right)$$

d. h. die Δw_i sind normalverteilt und die Stärke der Mutation wird durch den Parameter σ bestimmt, der geeignet zu wählen ist.

8.3.2 Die $(\mu + \lambda)$ –Strategie

$\boxed{1.}$ $t = 0$:

Initialisierung $\mathcal{P}_t = \left(w_t^{E1}, ..., w_t^{E\mu}\right)$

$\Downarrow$

$\boxed{2.}$

Elterngeneration

$\mathcal{P}_t = \left(w_t^{E1}, ..., w_t^{E\mu}\right)$

$\#\mathcal{P}_t = \mu$

Rekombination&

Reproduktion $\Rightarrow$

Population $\mathcal{P}_t^{\lambda N}$; $\#\mathcal{P}_t^{\lambda N} = \lambda$

"Nachkommen"

$\Downarrow$

Mutation: $w_t^{Nj} = w_t^{Nj} + \sigma\,\Delta w_t$

für $w_t^{Nj} \in \mathcal{P}_t^{\lambda N}$, $j = 1, ..., \lambda$

$\Downarrow$ $\Downarrow$

Intermediäre Population

$\mathcal{P}_t^{E\&N} = \mathcal{P}_t \cup \mathcal{P}_t^{\lambda N}$, $\#\mathcal{P}_t^{E\&N} = \mu + \lambda$

$\Downarrow$

Bewertung $\Rightarrow$ Rangfolge in der Gruppe der Eltern und Nachkommen $\mathcal{P}_t^{E\&N}$ gemäß Fitneß $F(w)$

$\Downarrow$

Selektiere:

Auswahl der μ besten aus den

$\mu + \lambda$ Individuen aus $\mathcal{P}_t^{E\&N}$

$\Downarrow$

Neue Population $\mathcal{P}_{t+1}$

$\Downarrow$

t=t+1

$\Downarrow$

Abbruch?

$\Downarrow$

GOTO 2.

8.3.3 (μ, λ)-Strategien

$\boxed{1.}$ $t = 0:$

Initialisierung $\mathcal{P}_t = \left(w_t^{E1}, ...w_t^{E\mu}\right)$

$\Downarrow$

$\boxed{2.}$

Rekombination&Reproduktion $\Rightarrow$

Population $\mathcal{P}_t^{\lambda N}, \quad \#\mathcal{P}_t^{\lambda N} = \lambda$

$\lambda > \mu$ "Nachkommen"

$\Downarrow$

Mutation: $w_t^{Nj} = w_t^{Nj} + \sigma\, \Delta w_t$

für $w_t^{Nj} \in \mathcal{P}_t^{\lambda N}, \quad j = 1, ..., \lambda$

$\Downarrow$

Intermediäre Population

$\mathcal{P}_t^{\lambda N}, \quad \#\mathcal{P}_t^{\lambda N} = \lambda$

$\Downarrow$

Bewertung $\Rightarrow$ Rangfolgegemäß Fitneß $F(w)$

$\Downarrow$

Selektiere:

Auswahl der μ besten aus den λ Nachkommen$\mathcal{P}_t^{\lambda N}$

$\Downarrow$

Neue Population $\mathcal{P}_{t+1}, \quad \#\mathcal{P}_{t+1} = \mu$

$\Downarrow$

t=t+1

$\Downarrow$

Abbruch?

$\Downarrow$

GOTO 2.

Die (μ, λ)-Strategien führen die Selektion nur bezüglich der Nachkommen aus, die Eltern der neuen Generation rekrutieren sich nur aus den besten der Nachkommen.

8.3.4 Allgemeine Notation

Wir wollen nun noch eine differenziertere Schreibweise einführen, die die Einzelheiten des Reproduktionsvorganges mit erfaßt. Insbesondere geht aus der bisherigen Notation noch nicht die genaue Form der Rekombination hervor. Das Crossing kann ja nicht nur zwischen zwei Individuen erfolgen, in der Praxis hat sich durchaus auch die Rekombination mehrer Individuen bewährt. Werden aus μ Eltern λ Nachkommen generiert und diese in Gruppen zu je ρ Individuen rekombiniert, dann schreiben wir

$$(\mu/\rho + \lambda) \tag{8.9}$$

Zusätzlich kann man noch den Satz der Parameter Ω (Mutationsraten, Selektionsmodus,...) und die Zahl m der Generationen mit aufführen, über die die Evolution erfolgen soll [21]:

$$(\mu/\rho + \lambda \mid \Omega)^m \tag{8.10}$$

8.3.5 Selbstbestimmte Mutationsraten

Der Erfolg der Evolutionsstrategien hängt entscheidend von der richtigen Wahl der Strategieparameter (Mutationsrate, ...) ab. Die Situation ist durch die Existenz eines sog. Evolutionsfensters gekennzeichnet, in das die Parameter fallen müssen, damit die Evolution in endlicher Zeit ein Ergebnis erzielt. Die Lage des Fensters ist i. a. unbekannt, so daß man auf Probieren angewiesen ist. Eine interessante Variante ist die Adaption der Parameter oder die unten besprochene Methode der Metaevolution.

Für die Adaption der Mutationsrate, speziell der Variationsbreite der mutativen Veränderungen σ, vgl. Gleichung 8.6, kann nach Rechenberg [38, 39] folgendes Adaptionsverfahren dienen:

Sei h_e die Erfolgshäufigkeit der auftretenden Mutationen, d. h. das Verhältnis von erfolgreichen zur Gesamtzahl der Mutanden. Nach der $\frac{1}{5}$-Regel von Rechenberg gilt

$$\sigma_t = \begin{cases} \sigma_{t-1}/c \quad falls \quad h_e > \frac{1}{5} \\ \\ \sigma_{t-1} \cdot c \quad falls \quad h_e < \frac{1}{5} \\ \\ \sigma_{t-1} \quad falls \quad h_e = \frac{1}{5} \end{cases} \tag{8.11}$$

Der empirische Wert von c ist $c = \sqrt[L]{0.85}$, wobei L die Zahl der Parameter ist.

8.3.6 Metaevolution

Die Strategie "Evolution der Evolution" oder Metaevolution wird auf zwei verschiedene Weisen verwirklicht. Einerseits als ein Verfahren zur automatischen Bestimmung der Strategieparameter (wie etwa die Mutationsrate) und andererseits als eine Evolution von Populationen, die selbst eine interne Evolution durchlaufen. Dabei werden also Subpopulationen wie Individuen behandelt.

Evolution über Subpopulationen

Vor allem bei multimodalen Fitneßlandschaften hat es sich als günstig erwiesen, mit einer Population zu arbeiten, die sich aus mehreren Subpopulationen zusammensetzt. In einer Fitneßlandschaft mit mehreren Hauptgipfeln kann sich dann jede der Subpopulationen auf einen der Hauptgipfel konzentrieren, falls die Subpopulationen die Chance erhalten, sich über einen begrenzten Zeitraum ungestört zu entwickeln, d. h. sich nicht gegen die Individuen der anderen Subpopulationen behaupten zu müssen. Das kann durch eine Isolation der Subpopulationen errreicht werden. Insbesondere von Mühlenbein wurde darauf hingewiesen, daß schon Darwin die ungestörte Evolution in isolierten Subpopulationen als essentielles Element der Evolution herausgestellt hatte, vgl. [34, 47]. In der oben eingeführten Notation schreiben wir beipielsweise

$$(\mu/2 + \lambda(\mu_0/3 + \lambda_0 \mid \Omega_0)^{m_0} \mid \Omega)^m \tag{8.12}$$

Das beschreibt eine Population aus μ Individuen, die jeweils Subpopulationen aus μ_0 Individuen darstellen. Jede Subpopulation erzeugt qua Kopie λ Kindpopulationen, die sich über m_0 Generationen ungestört entwickeln können. Die Rekombination erfolgt auf Gruppen von je 2 bzw. 3 Individuen.

Evolution der Strategieparameter

Avancierte Algorithmen unterwerfen die Strategieparameter ebenfalls der Evolution. Diese evolutive Einstellung der Parameter der Evolution zielt auf eine Evolution der Evolutionsstrategie selbst und wird deshalb ebenfalls als Metaevolution bezeichnet. Diese Strategien zweiter Ordnung finden i. a. in den oben dargestellten mehrgliedrigen Evolutionsstrategien Anwendung. Die Metaevolution wird in eleganter Weise dadurch verwirklicht, daß die Strategieparameter wie etwa σ formal mit in den Satz der Objektparameter (Stellgrößen) w aufgenommen werden, d. h. wir erweitern

- $w := (w, \sigma, ...)$ mit $\sigma = (\sigma_1, ..., \sigma_L)$ ein Satz komponentenspezifischer Schrittweiten.
- Die Fitneßfunktion hängt von $\sigma, ...$ nicht ab, $F(w) = F(w_1, ..., w_L)$ und ist so auch für den erweiterten Parametervektor definiert.
- Die Mutation ergibt sich aus dem Vektor Δw normalverteilter Zufallszahlen mit Verteilungsdichte

$$p(\Delta w) = \sqrt{\frac{|A|}{(2\pi)^L}} \exp\left(-(\Delta w)^T A \, \Delta w / 2\right)$$

  wobei A die Matrix der Kovarianzen $|A|$ ihre Determinante und $(\Delta w)^T$ ein Zeilenvektor (transponiertes von Δw) ist. Die Matrix A ist geeignet zu parametrisieren, damit $|A| > 0$ auch bei der Adaption von A (Metaevolution) gesichert bleibt.

8.4 Genetische Programmierung

Ziel der genetischen Programmierung ist die Evolution von Computerprogrammen. Im Gebiet des Maschinellen Lernens kann die Methode der genetischen Programmierung unmittelbar Anwendung finden. Wir stellen dabei die allgmeine Transferfunktion des Lerners

$$y = S(x \mid w) \tag{8.13}$$

direkt durch ein Computerprogramm dar. In der von Koza [27] realisierten Form ist das ein LISP- Ausdruck, also

$$x \;\rightarrow \boxed{\begin{array}{c} \text{Lerner} \\ ----- \\ \{w\} \end{array}} \rightarrow y \tag{8.14}$$

mit $w =$ LISP–Ausdruck. Beispiel

$$x \;\rightarrow \boxed{\begin{array}{c} \text{Lerner} \\ ----- \\ (* \quad x_1 \quad x_2 \quad (IF \quad (< \quad x_1 \quad 1) \quad 1 \quad 0)) \end{array}} \rightarrow y \tag{8.15}$$

Der Ouptut des Lerners ist folglich $y = x_1 x_2$, falls $x_1 < 1$ und $y = 0$ sonst.

Die genetische Programmierung kann im Kontext des Maschinellen Lernens als ein kollektives, evolutionäres Lernverfahren über einer Population von N Computerprogrammen betrachtet werden. Im konkreten Fall ist jedes Individuum der Population durch einen LISP–Ausdruck beschrieben. Dieser stellt den Genotyp des entsprechenden Agenten (Lerners) dar. Der Phänotyp ist der reale Lerner in seinem durch das Programm beschriebenen Verhalten, dessen Fitness in der Anwendung auf die konkrete Lernaufgabe bestimmt werden kann.

Die Population wird in der 0–ten Generation durch die zufallsgeleitete Erzeugung von N Programmen initialisiert und dann der Evolution unterworfen. Dazu wird in jeder Generation die Fitness eines jeden Individuums (Programms) bestimmt, die Population fitneßgewichtet reproduziert und durch genetische Operatoren modifiziert. Dieses Lernen über Generationen erzeugt im Idealfalle ein oder mehrere Programme, die die vorgegebene Lernaufgabe optimal lösen.

Ein typischer Anwendungsfall ist etwa das Lernen aus Beispielen, also hier das Auffinden eines mathematischen (oder logischen) Ausdrucks, der einen durch eine Anzahl von Beispielen $\eta^{(i)} = \left(x^{(i)}, y^{(i)}_{soll}\right), \quad i = 1, ..., n$ gegebenen (unbekannten) funktionalen Zusammenhang $y = F(x)$ am besten repräsentiert. Jedes der Computerprogramme realisiert eine Funktion $F^{(app)}(x)$ als (approximative) Darstellung von $F(x)$. Die Güte der Approximation kann durch die mittlere quadratische Abweichung

$$E = \frac{1}{2} \sum_{x^{(i)}} \left(F^{(app)}\left(x^{(i)}\right) - y^{(i)}_{soll}\right)^2 = \frac{1}{2} \sum_{x^{(i)}} \left(F^{(app)}\left(x^{(i)}\right) - F\left(x^{(i)}\right)\right)^2 \tag{8.16}$$

quantifiziert werden. $Q = -E$ definiert für jedes Programm die Fitneß die in jeder Generation für alle Individuen der Population jeweils auf der Beispielmenge bestimmt werden muß.

Die effektive Realisierung der genetischen Programmierung ist wesentlich von der Wahl einer geeigneten Programmiersprache zur Implementierung der konkurrierenden Computerprogrammme abhängig. Von J. Koza wird hierbei LISP als besonders geeignet dargestellt, von C. Jacob wurde alternativ eine erfolgreiche Implementierung in Mathematica realisiert, vgl. [21].

8.4.1 Realisierung der genetischen Programmierung

Für die GP ist insbesonder die Eigenschaft von LISP nützlich, daß Daten und Programme den gleichen operationalen Status haben, d. h. Daten können als Programme interpretiert werden und umgekehrt. Die S-Ausdrücke werden aus

$$\Phi = \{\phi_1, ..., \phi_{N_F}\} \qquad \text{(Menge der Funktionen)}$$

und

$$T = \{a_1, ..., a_{N_T}\} \qquad \text{(Menge der Terminals=Inputs)}$$

bzw. der entsprechenden Vereinigungsmenge

$$\Pi = \{\Phi, T\} = \{\phi_1, ..., \phi_{N_F}, a_1, ..., a_{N_T}\} \tag{8.17}$$

gebildet. Die Menge der Funktionen umfaßt beispielsweise

- Arithmetische Operationen, $+, -, *, /, ...$
- Mathematische Funktionen, $\sin, \quad \cos, \quad \log, ...$
- Boolsche Operatoren, $and, or, not, ...$
- Bedingte Operatoren, $if \ \ , ...$
- Rekursionen
- ...
- Weitere dommänenspezifische Funktionen.

Die Terminalmenge enthält

- Variable Atome als Inputs
- Numerische und/oder logische Konstanten
- ...

In der Vereinigungsmenge (8.17) werden die Terminals als argumentfreie Funktionen interpretiert und gleichberechtigt mit den anderen Funktionen behandelt. Der Funktionssatz Π muß nun eine Reihe offenkundiger **Bedingungen** erfüllen:

- **Abgeschlossenheit:** Die Funktionen müssen alle auftretenden Argumente verarbeiten können. Das ist trivialerweise erfüllt, wenn Φ und T nur Boolsche Funktionen bzw. Variable enthalten. Für reellwertige Argumente empfiehlt sich etwa die Einführung einer geschützten Division

$$a\%b = \begin{cases} 1 \text{ falls } b = 0 \\ \\ \frac{a}{b} \text{ sonst} \end{cases} \tag{8.18}$$

  was durch

$$\text{(defun} \quad \% \quad \text{(Z} \quad \text{N)} \quad (if \quad = \quad 0 \quad \text{N)} \quad 1 \quad \text{(/} \quad \text{Z} \quad \text{N))} \tag{8.19}$$

  definiert werden kann.

- **Vollständigkeit:** Aus der Grundmenge Π der Funktionen muß eine Lösung des Problems konstruierbar sein.

Beispiel: Für die Realisierung einfacher Boolscher Funktionen mit zwei Argumenten setzen wir

$$\Phi = \{and, or, not\}$$

und für die Terminalmenge

$$T = \{d_0, d_1\}$$

mit den Boolschen Variablen d_0, d_1. Die Vereinigungsmenge ist dann

$$\Pi = \{and, or, not, d_0, d_1\}$$

Eine spezieller aus diesen Funktionen erzeugter Ausdruck ist z. B. die Paritätsfunktion

$$(or \quad (and\ (not\ d_0)\ (not\ d_1)) \quad (and\ d_0\ d_1))$$

die den Wert $true$ liefert, wenn d_0, d_1 beide gleichzeitig $true$ oder $false$ sind und $false$ sonst.

Ein Trick, um die Abgeschlossenheit des Funktionssatzes zu erreichen, besteht in der Einführung einer numerisch-wertigen Logik, d. h. wenn ein logischer Ausdruck zu $true$ (oder $false$) evaluiert, wird dieser Wert als 1 (oder 0) interpretiert, sobald er als Argument in einer numerischen Funktion auftritt.

8.4.2 Initialisierung

Die genetische Programmierung startet mit einer biasfreien (zufallserzeugten) Anfangspopulation. Die Zufallserzeugung eines S-Ausdruckes geht folgendermaßen vor sich:

1. Wähle aus dem Satz Φ der Funktionen ein Element zufällig aus. Diese Funktion bildet den Wurzelknoten [1]. Bestimme die relevante Zahl der Argumente der ausgewählten Funktion.

2. Für jedes dieser freien Argumente: Wähle aus Π ein Element zufällig aus und besetze das freie Argument mit diesem Element (Funktion oder Terminal).

3. Falls keine freien Argumente mehr existieren (weil alle mit Terminals besetzt sind) $\rightarrow$ $Halt$. Falls die maximal erlaubte Tiefe des Baumes erreicht ist: Besetze alle verbleibenden freien Argumente mit Terminals, $\rightarrow$ $Halt$. Ansonsten gehe zu 2.

Die initiale Population besteht aus N solcher zufällig generierter Ausdrücke.

8.4.3 Genetische Operatoren

Die so gewonnene Population wird nun der Evolution unterworfen. Die relevanten genetischen Operatoren sind hier in folgender Weise realisiert:

[1] Wir interpretieren die LISP–Ausdrücke als Baumgraphen.

1. Crossover: Hier werden zwei S-Ausdrücke aus der Population selektiert (mit Wahrscheinlichkeit proportional zur Fitneß) und zufällig ausgewählte Subausdrücke zwischen den beiden Eltern ausgetauscht.

   Elter 1 | Elter 2

   $$(and \quad (or\, A\, B) \quad (and\, A\, B)) \qquad (or \quad (not\, B) \quad (and\ (not\, A\, not\, B)))$$

   gilt nach dem Crossing

   Elter 1 | Elter 2

   $$(and \quad (and\ (not\, A \quad not\, B)) \quad (and\, A\, B)) \qquad (or \quad (not\, B) \quad (or\, A\, B))$$

2. Mutation: Hier werden ähnlich wie bei der zufälligen Initialisierung Subausdrücke zufällig geändert. Beipiel: Der Ausdruck

   $$(and \quad (or\, A\, B) \quad not\, A)$$

   wird im Ergebnis der Mutation

   $$(and \quad (or\ A\ (not\, B)) \quad not\, A)$$

3. Permutation: Die Permutation kann als ein Spezialfall der Mutation aufgefasst werden. Beipiel: Der Ausdruck

   $$(+\ (if\ (<\ A\, B)\ C\, D)\ (-\, E\, F))$$

   geht durch Permutation der beiden Terminals A, B über in

   $$(+\ (if\ (<\ B\, A)\ C\, D)\ (-\, E\, F))$$

4. Einkapselung: Für die Bewahrung einmal gefundener Teillösungen hat sich die Einkapselungsoperation bewährt. Dabei werden Subausdrücke als Terminals definiert, die dann nicht mehr verändert werden können, d. h. der S-Ausdruck

   $$(* \quad A \quad (+\, B\, C))$$

wird mit

$$(\text{defun} \quad E0 \quad (B\,C) \quad (+\,B\,C))$$

definiert und kann dann wie in

$$(* \quad A \quad (E0\,B\,C))$$

verwendet werden.

5. Gen-Duplikation: Als eine neue, besonders erfolgversprechende genetische Operation wird derzeit die Gen-Duplikation diskutiert. In der genetischen Programmierung wird diese Option hauptsächlich durch die Methode der automatischen Funktionsdefinition realisierbar, s. [27].

Literaturverzeichnis

[1] Learning decision trees for mapping the local environment in mobile robot navigation. Technical report.

[2] S. Arikawa, S. Kuhara, S. Miyano, Y. Mukouchi, A. Shinohara, and T. Shinohara. A machine discovery from amino acid sequences by decision trees over regular patterns. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 618–625, ICOT, Japan, 1992. Association for Computing Machinery.

[3] K. Binder and D. W. Heerman. *Monte Carlo Simulation in Statistical Mechanics*. Springer-Verlag, Berlin, 1988.

[4] T. Cox and M. A. A. Cox. *Multidimensional scaling*. Chapman & Hill, 1994.

[5] R. Der. Vorlesung Neuroinformatik. Kapitel 2. UniversitätLeipzig, Institut für Informatik. http://www.informatik.uni-leipzig.de/ der/Vorlesungen/el-proz.ps.Z.

[6] R. Der. Vorlesung Neuroinformatik. UniversitätLeipzig, Institut für Informatik. http://www.informatik.uni-leipzig.de/ der/Vorlesungen/skrpt-ni.html.

[7] R. Der. The time local view of nonequilibrium statistical mechanics I. *J. Stat. Phys.*, 46:349–390, 1987.

[8] R. Der. The time local view of nonequilibrium statistical mechanics II. *J. Stat. Phys.*, 46:391–425, 1987.

[9] R. Der and M. Herrmann. Critical phenomena in self-organized feature maps: A Ginzburg-Landau approach. *Phys. Rev. E*, 49(5):5840–5848, 1994.

[10] R. Der and M. Herrmann. Self-adjusting reinforcement learning. In *Nonlinear Theory and Applications - NOLTA 96*, pages 441 – 444, 1996. Internet: http://www.informatik.uni-leipzig.de/ der/Veroeff/nolta96.ps.gz.

[11] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1972.

[12] F. Esposito, D. Malerba, and G. Semeraro. Simplifying decision trees by pruning and grafting: New results. *Lecture Notes in Computer Science*, 912:287–??, 1995.

[13] J. Fürnkranz. Pruning algorithms for rule learning. *Machine Learning*, 27:139, 1997.

[14] C. W. Gardiner. *Handbook of Stochastic Methods.* Springer, 1985.

[15] C. W. Gardiner. *Handbook of Stochastic Methods.* Springer, 1990.

[16] A. Gersho and R. M. Gray. *Vector quantization and signal processing.* Kluwer Academic Publishers, Boston/Dordrecht/London, 1992.

[17] T. Heskes, E. Slijpen, and B. Kappen. Learning in neural networks with local minima. *Physical Review A*, 46:5221–5231, 1992.

[18] T. Heskes and W. Wiegerinck. A theoretical comparison of batch-mode, on-line, cyclic, and almost-cyclic learning. *IEEE Transactions on Neural Networks*, 7(4):919–925, July 1996.

[19] T. M. Heskes, E. T. P. Slijpen, and B. Kappen. Cooling schedules for learning in neural networks. *Physical Review E*, 47:4457–4464, 1993.

[20] T. Hofmann and J. Buhmann. Multidimensional scaling and data clustering. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 459–466. The MIT Press, 1995.

[21] C. Jacob. *Principia Evolvica - Simulierte Evolution mit Mathematica.* dpunkt-Verlag, Heidelberg, 1997.

[22] W. Kinzel and M. Opper. Dynamics of learning. In E. Domany, J. L. van Hemmen, and K. Schulten, editors, *Physics of Neural Networks*, volume 1. Springer-Verlag, Berlin, 1990.

[23] U. Knoll, G. Nakhaeizadeh, and B. Tausend. Cost-sensitive pruning of decision trees. *Lecture Notes in Computer Science*, 784:383–??, 1994.

[24] T. Kohonen. *Self–Organization and associative memory*, volume 8 of *Springer Series in Information Science*. Springer, 1984.

[25] T. Kohonen. *The self—organizing map*. Springer, 1995.

[26] J. R. Koza. Concept formation and decision tree induction using the genetic programming paradigm. *Lecture Notes in Computer Science*, 496:124–??, 1991.

[27] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection. 2nd edition.* MIT Press, 1993.

[28] A. Krogh and P. Sollich. Statistical mechanics of ensemble learning. *Physical Review*, E 55:811, 1997.

[29] T. K. Leen. Stochastic Manhattan learning: An exact time-evolution operator for the ensemble dynamics. ftp://speech.cse.ogi.edu/pub/neural/papers/LeenMoody96.StochMan.ps.Z.

[30] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. An empirical comparison of decision trees and other classification methods. Technical Report 979, Department of Statistics, University of Wisconsin-Madison, Madison, WI, June 30 1997.

[31] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions Comm.*, 28:84–95, 1980.

[32] M. Mehta, J. Rissanen, and R. Agrawal. MDL-based decision tree pruning. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 216–221, Aug. 1995.

[33] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine learning, neural and statistical classification.* Ellis Hoorwood, 1994. STATLOG-Report.

[34] H. Mühlenbein. Darwin's continent cycle theory and its simulation by the prisoner's dilemma. *Complex Systems*, 5:459–478, 1991.

[35] J. R. QUINLAN. *Inductive inference as a tool for the construction of efficient classification programs.* Tioga, Palo Alto, CA, 1983.

[36] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1992.

[37] J. R. Quinlan. Improved Use of Continuous Attributes in C4.5. *Journal of Artificial Intelligence*, 4:77–90, 1996.

[38] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, 1973.

[39] I. Rechenberg. *Evolutionsstrategie'94.* Frommann-Holzboog, Stuttgart, 1994.

[40] H. Risken. *The Fokker-Planck equation.* Springer, 1989.

[41] R. Rojas. *Theorie der neuronalen Netze.* Springer-Verlag, 1993.

[42] R. Rojas. *Theorie der neuronalen Netze.* Springer-Verlag, 1993.

[43] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 21:660–674, 1991.

[44] G. Salton. *Automatic Text Processing.* Addison Wesley, Massachusetts, 1989.

[45] K. Sayood. *Introduction to data compression.* Morgan Kaufmann, San Francisco, 1996.

[46] C. Schaffer. When does overfitting decrease prediction accuracy in induced decision trees and rule sets? In Y. Kodratoff, editor, *Proceedings of the European Working Session on Learning : Machine Learning (EWSL-91)*, volume 482 of *LNAI*, pages 192–205, Porto, Portugal, Mar. 1991. Springer Verlag.

[47] D. Schlierkamp-Voosen and H. Mühlenbein. Strategy adaptation by competing subpopulations. *Lecture Notes in Computer Science*, 866:199–??, 1994.

[48] H. P. Schwefel. *Parallel problem solving from nature*. Springer, Berlin [u.a.], 1990.

[49] E. V. Siegel and K. R. McKeown. Emergent linguistic rules from inducing decision trees: disambiguating discourse clue words. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Menlo Park, CA, USA, July 1994. AAAI Press.

[50] N. G. van Kampen. *Stochastic Processes in Physics and Chemistry*. Elsevier, 1981.

[51] V. N. Vapnik. *The nature of statistical learning theory*. Springer, 1995.

[52] W. N. Wapnik and A. J. Tscherwonenkis. *Theorie der Zeichenerkennung*. Akademie–Verlag, 1979.

[53] C. J. Watkins and P. Dayan. Technical note Q-learning. *Machine Learning*, 8:279, 1992.

[54] Y. Yuan and M. J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, 69(2):125–139, 1995.

[55] A. Zell. *Simulation neuronaler Netze*. Addison–Wesley, 1994.

[56] X. J. Zhou and T. S. Dillon. A statistical-heuristic feature selection criterion for decision tree induction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(8):834–841, Aug. 1991.

[57] P. G. Zimbardo. *Psychologie*. Springer, 1995.