Dynamische Zellstrukturen
Theorie und Anwendung eines KNN-Modells

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Jörg Bruske

Kiel
1998

Zusammenfassung

In dieser Arbeit wird eine neuartige KNN-Architektur vorgestellt, die *Dynamische Zellstruktur (DCS)*, die das Approximationsverhalten von Netzwerken radialer Basisfunktionen (RBF-Netzen) mit der Repräsentationskraft topologieerhaltender Karten vereint. Die topologieerhaltenden Karten werden dabei zu einer im Vergleich zu herkömmlichen RBF-Netzwerken effizienteren Ausgabeberechnung und Adaption genutzt. Daneben tragen sie zu einer Regularisierung des Lernproblems bei und bewirken durch effektive Projektion auf die Eingabemannigfaltigkeit eine verminderte Rauschsensitivität. Durch inkrementelle Adaption sowohl der Parameter als auch der Modellordnung und der topologieerhaltenden Repräsentation der Eingabemannigfaltigkeit weist die DCS ein Höchstmaß an Flexibilität auf. Hierfür werden eine Vielzahl von Lernregeln angegeben, die im Kontext der stochastischen Approximation und der statistischen Mustererkennung diskutiert werden.

Die experimentelle Evaluierung der DCS erfolgt im Kontext sämtlicher relevanter Lernparadigmen, d.h. dem unüberwachten und überwachten Lernen, dem Fehlerrückkopplungs-Lernen und dem Reinforcement-Lernen, wobei im Rahmen des Reinforcement-Lernens das Q-Lernen und das reellwertige Reinforcement-Lernen untersucht werden. Das vorgestellte Anwendungsspektrum reicht von der topologieerhaltenden Repräsentation von Eingabemannigfaltigkeiten über die überwachte Konstruktion diverser Klassifikatoren und das überwachte Lernen der Regelungsfunktion eines invertierten Pendels bis zum Erlernen der Sakkadensteuerung eines Stereokamerakopfes und der visuomotorischen Koordination eines Industrieroboterarmes durch Fehlerrückkopplung sowie dem Erlernen einer Hindernisvermeidungsstrategie für einen autonomen mobilen Roboter durch reellwertiges, verzögertes Reinforcement.

Danksagung

Diese Arbeit entstand an dem von Prof. Dr. Gerald Sommer gegründeten und geleiteten Lehrstuhl für Kognitive Systeme des Instituts für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel. Ihm als meinem Doktorvater möchte ich an erster Stelle meinen Dank für seine Begleitung und Unterstützung der vorliegenden Arbeit aussprechen. Besonders möchte ich mich für das große Vertrauen, das er mir von Anfang an entgegenbrachte, seine ständig offene Tür und seine zahlreichen Ratschläge und Ermutigungen bedanken.

Als nächstes gilt mein Dank meinen Kollegen Udo Mahlmeister, Michael Hansen, Josef Pauli und Konstantinos Daniilidis. Durch den gemeinsamen Aufbau des Lehrstuhls zusammengeschweißt, verband uns eine weit über die fachliche Zusammenarbeit hinausgehende Freundschaft, die sich nicht zuletzt in einer hervorragenden Arbeitsatmosphäre äußerte. Auch Henrik Schmidt und Gerd Diesner möchte ich für gute Zusammenarbeit und die technische Unterstützung bei der Überwindung so mancher Hard- und Software-Hürde danken.

Für fruchtbare Diskussionen und überdurchschnittliches Engagement sei an dieser Stelle meinen DiplomandInnen namentlich gedankt. Im Rahmen dieser Arbeit fühle ich mich dabei Ingo Ahrns, Lars Riehn und Erika Ábrahám-Mumm zu besonderem Dank verpflichtet, deren Ausdauer und Experimentierfreudigkeit insbesondere die experimentellen Ergebnisse zur lernenden Hindernisvermeidung und Online-Stabbalance (Ingo [2]), Sakkadensteuerung (Lars [107]) und Hand-Auge-Koordination (Erika [1]) zu verdanken sind. Aber auch Bertil Schmidt, Harald Stark, Tim Haase, Oliver Harwart, Jan Tröster und Anja Ahlfeld möchte ich für ihre Unterstützung bei der Entwicklung eines Attentionsmechanismus (Bertil), der Entwicklung eines auf probabilistischen Besetztheitsgittern beruhenden autonomen Serviceroboters (Harald und Tim), den Untersuchungen zur landmarkengestützten Roboternavigation (Oliver), der Entwicklung eines Ansichten-basierten Greifsystems (Jan) und der Implementierung einer Vision-basierten Hindernisvermeidung (Anja) danken, selbst wenn ihre Beiträge nicht in diese Arbeit einfließen konnten.

In tiefer Schuld stehe ich bei meiner Frau Petra. Für ihr Verständnis und ihre Unterstützung, die sie mir während dieser schwierigen Zeit zukommen ließ, danke ich ihr von ganzem Herzen.

Schließlich möchte ich mich bei meinen Eltern bedanken. Ohne ihren Rückhalt und Beistand wäre mein Leben anders verlaufen und nicht nur diese Arbeit nicht zustande gekommen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Betrachtet man die boomartige Entwicklung der Neuroinformatik in der letzten Dekade, so wurzelt die Faszination dieses recht jungen und interdisziplinär angelegten Forschungsgebietes zum einen in dem uralten Menschheitstraum von der Erschaffung künstlicher intelligenter Wesen, zum anderen in der ernüchternden Beobachtung, daß heute existierende Computersysteme zwar hervorragende, die menschlichen Fähigkeiten in mancher Hinsicht um Dimensionen übersteigende Leistungen auf dem Gebiet der Symbolverarbeitung und des Numbercrunchings erbringen, andererseits aber auf Gebieten der Kognitiven Intelligenz, wie der robusten Erkennung "natürlicher" Objekte und Signale, selbst einfacheren biologischen Systemen hoffnungslos unterlegen sind. Während lange Zeit dieses Versagen existierender technischer Systeme fehlender Rechengeschwindigkeit zugeschrieben wurde, setzt sich zunehmend die Erkenntnis durch, daß kognitive Intelligenz eines anderen, biologienäheren Berechnungsparadigmas als dem der Symbolverarbeitung bedarf. Die Neuroinformatik mit ihrem expliziten Bezug zum biologischen Vorbild verspricht diesen Paradigmenwechsel.

Innerhalb der Neuroinformatik sind dabei zwei Hauptforschungsrichtungen auszumachen: Zum einen die *Modellierung* (neuro-) biologischer und psychologischer Phänomene, also die am biologischen Vorbild orientierte Modellentwicklung, zum anderen die Nutzung solcher Modelle zur *Problemlösung* in Form der sogenannten *Künstlichen Neuronalen Netze (KNN)*, in die sich auch die vorliegende Arbeit einordnet. Kennzeichen der KNN ist dabei eine weitestgehende Abstraktion vom biologischen Substrat, mit dem sie aber wesentliche Merkmale der biologischen Informationsverarbeitung teilen. Diese umfassen massive Parallelität, Lokalität , Redundanz, Adaptivität und stochastische Eigenschaften, auch wenn KNN heute meist auf sequentiell, deterministisch und redundanzfrei arbeitenden von-Neumann-Architekturen simuliert werden. Verbindendes Element der KNN ist der Grundgedanke, durch eine Vielzahl vergleichsweise einfacher Grundbausteine, die künstlichen Neuronen, und deren massive Vernetzung komplexe, problemadaptive Berechnungsmodelle zu konstruieren.

Heute kommen KNN hauptsächlich als Funktionsapproximatoren für hochdimensionale Regressions- oder Klassifikationsaufgaben zur Anwendung. Das Training von KNN wird damit zu einem Modell- und Parameteroptimierungsproblem, für das die Statistik und stochastische Approximationstheorie den derzeit besten Beschreibungsrahmen bilden. KNN zeichnen sich hier vor allem dadurch aus, daß sie durch ihre Nichtlinearität universale Approximatoren darstellen und herkömmlichen Funktionsapproximationsansätzen hinsichtlich Generalisierungsfähigkeit und Skalierungsverhalten in höherdimensionalen Räumen überlegen sind. So konnte Barron [7] zeigen, daß der Approximationsfehler für KNN mit sigmoiden Basisfunktionen unabhängig von der Dimension d des Eingaberaumes im allgemeinen mit $O(1/N)$ (N die Anzahl der Neuronen) abnimmt, während dieser bei Verwendung herkömmlicher Verfahren (z.B. Polynomapproximatoren) nur mit $O(1/N^{2/d})$ abnimmt und daher eine exponentiell mit der Dimension wachsende Anzahl von Basisfunktionen erfordert. Dieses vorteilhafte Skalierungsverhalten und das aus einer kleineren Anzahl von Basisfunktionen resultierende bessere Generalisierungsverhalten verdanken KNN der datenabhängigen Adaption der Basisfunktionen selbst, wodurch ihre Anzahl im wesentlichen von der Komplexität des Problems, nicht aber von der Dimension des Eingaberaumes bestimmt wird.

Während klassische, "symbolische" Systeme der *Künstlichen Intelligenz* (KI) sich innerhalb eines fest vorgegebenen Repräsentationsrahmens bewegen und Lernen sich oft nur auf eine Umordnung bekannten Wissens beschränkt (Bsp.: Explanation Based Learning), kommt KNN bei der Lösung des grundlegenden Problems symbolischer KI-Systeme eine entscheidende Bedeutung zu, der sogenannten Symbolverankerung. Dieses Problem beschreibt die Schwierigkeit, den Elementen eines Symbolsystems eine Bindung zur Welt zu geben. Mit KNN können erstmals Systeme gebaut werden, die lernen, Symbole mit Repräsentationen zu verbinden, die in den eigenen Erfahrungen verankert sind. Die Bedeutung der Symbole wird hierbei in Bezug auf die eigene Erfahrungsmöglichkeiten erfaßt. Dieses Potential beruht auf der Fähigkeit von KNN, sensorische Eingangssignale mit nicht-sensorischen Aktivierungen zu verbinden, Aktivierungsmuster zu kategorisieren und Parameter in Abhängigkeit von der Erfahrung des Systems adaptieren zu können [34]. Wissen wird in KNN nicht als objektiv festlegbare Struktur angesehen, sondern als Struktur, die sich ein System zum Zwecke einer optimalen Anpassung an seine Umgebung aufbaut. Damit sind KNN prädestiniert, eine zentrale Rolle im *verhaltensbasierten Entwurf technischer Systeme* [17, 118] einzunehmen, dem aus heutiger Sicht meistversprechenden Ansatz zur Konstruktion kompetenter, robuster Systeme. Im Mittelpunkt dieses Entwurfsprinzips steht die Erkenntnis, daß Beobachter und Umwelt eine Einheit bilden müssen. Hiernach zeichnen sich Systeme, die dem verhaltensbasierten Entwurfsprinzip genügen, durch folgende Eigenschaften aus [119]: *Situiertheit* (die Interaktion mit der Umwelt erfolgt durch konkrete und nicht simulierte Aktionen), *Körperlichkeit* (Wissenserwerb und Repräsentationsbildung erfolgen auf Basis des sensorischen Inputs und der situierten Wechselwirkung mit der Umwelt), *Kompetenz* (als Ausdruck der Fähigkeit, Kategorien und Konzepte bezüglich der Umwelt bilden und nach diesen kompetent handeln zu können) und *Emergenz* (Entstehen qualtitativ neuer Verhaltensweisen durch Wechselwirkung primitiverer Verhalten untereinander und mit der Umwelt).

Zum erfolgreichen Einsatz als adaptive Module in verhaltensbasierten Systemen müssen KNN zwei Voraussetzungen erfüllen: Zum einen erfordern sie eine *höchst flexible Struktur* (Architektur des Netzwerkes), die die Ausbildung einer der Aufgabenstellung angepaßten optimalen Repräsentation erlaubt, zum anderen müssen sie in der Lage sein, auf eine Vielzahl von Lernparadigmen zurückzugreifen, um möglichst alle relevanten, in der

Wechselwirkung mit der Umwelt auftretenden Informationen zu nutzen. Die vorliegende Arbeit versucht, hierzu einen Beitrag zu leisten. So wird im ersten Teil der Arbeit eine KNN-Architektur vorgestellt, die *Dynamische Zellstruktur (DCS)*, die erstmals das gute Approximationsverhalten von *Netzwerken radialer Basisfunktionen (RBF-Netzen)* mit der Repräsentationskraft optimal topologieerhaltender Karten vereint. Die DCS besitzt ein Höchstmaß an Flexibilität, denn sie erlaubt eine inkrementelle Adaption sowohl der Parameter als auch der Ordnung des Modelles (Anzahl der Neuronen) und der topologieerhaltenden Repräsentation der Eingabemannigfaltigkeit. Hierfür wird eine Vielzahl von teils speziell auf die DCS zugeschnittenen Lernregeln angegeben, die im Kontext der stochastischen Approximation und der statistischen Mustererkennung diskutiert werden. Besonderes Gewicht liegt dabei auf die Entwicklung von Online-Lernregeln. Die DCS weist insbesondere folgende, nicht nur für verhaltensbasierte Systeme wichtige Eigenschaften auf: i) Fähigkeit zur inkrementellen Online-Adaption der Parameter, der Modellordnung und der Topologie, ii) Fähigkeit zum One-shot-Lernen, d.h. Speicherung von Ein-/ Ausgabepaaren nach einmaliger Präsentation, und iii) Fähigkeit zur Integration von aufgabenspezifischem Vorwissen. Die topologieerhaltenden Karten werden zu einer im Vergleich zu herkömmlichen RBF-Netzwerken effizienteren Ausgabeberechnung und Adaption genutzt, die jeweils nur topologisch benachbarte Berechnungsknoten umfaßt. Daneben tragen sie zu einer Regularisierung des Lernproblems bei und bewirken durch effektive Projektion auf die Eingabemannigfaltigkeit eine verminderte Rauschsensitivität.

Im zweiten Teil der Arbeit erfolgt die experimentelle Evaluierung der DCS und der vorgestellten Lernregeln im Zusammenhang mit sämtlichen relevanten Lernparadigmen, d.h. dem unüberwachten und überwachten Lernen, dem Fehlerrückkopplungs-Lernen und dem Reinforcement-Lernen, wobei im Rahmen des Reinforcement-Lernens das Q-Lernen und das reellwertige Reinforcement-Lernen untersucht werden. Es kommen ausschließlich Online-Lernregeln zum Einsatz, die jeweils im Kontext der spezifischen Anwendung und des jeweiligen Lernparadigmas abgeleitet werden. Dabei werden, insbesondere im Rahmen des Reinforcement-Lernens, auch neue Lernverfahren entwickelt. Als exemplarische Anwendungen werden zum einen Benchmarks herangezogen (diese umfassen CMU-Klassifikations-Benchmarks, das Stabbalance-Problem in zwei Varianten und den Ritterschen Versuchsaufbau zur visuomotorischen Koordination eines Roboterarmes), zum anderen werden originär im Rahmen dieser Arbeit entstandene Anwendungen und Lösungsansätze dargestellt. Letztere umfassen

- die kalibrierungsfreie adaptive Sakkadensteuerung eines TRC-Bisight Stereokamerasystems nach dem Fehlerrückkopplungsprinzip,
- das Erlernen eines ultraschallbasierten hindernisvermeidenden Navigationsverhaltens für einen (simulierten) mobilen Roboter, den TRC-Labmate, durch reellwertiges verzögertes Reinforcement und
- die Verifikation von Schlüsselpositionen in Gesichtsbildern[1] nach dem Satz von Bayes.

Diese Zweiteilung reflektierend gliedert sich die Arbeit wie folgt: In Kapitel 2 wird nach einigen generellen Bemerkungen zum Training von KNN ein Überblick über existierende KNN-

[1]in Zusammenarbeit mit der GSF-Medis.

Modelle mit lokal rezeptiven Feldern gegeben. Diese sind insbesondere die von T. Kohonen entwickelten Kohonenkarten, das Neuronengas und die perfekt topologieerhaltenden Karten von T. Martinetz sowie die RBF-Netze. Auf Grundlage der topologieerhaltenden Karten und der RBF-Netze werden dann in Kapitel 3 die Dynamische Zellstruktur entwickelt und deren Eigenschaften untersucht. Ein großer Teil dieses Kapitels ist dem Entwurf von Lernregeln zur Adaption der Parameter, der Modellordnung und der topologieerhaltenden Karten gewidmet. Mit Kapitel 4 beginnt der experimentelle Teil der Arbeit, der Einsatz von DCS im Rahmen der verschiedenen Lernparadigmen und unterschiedlichster Anwendung, beginnend mit dem unüberwachten und überwachten Lernen. Anwendungsbeispiele sind hier die selbstorganisierende Kartierung von Mannigfaltigkeiten, die CMU-Klassifikations-Benchmarks, überwachte Stabbalance und die Verifikation von Schlüsselpositionen in Gesichtsbildern. Kapitel 5 behandelt das Fehlerrückkopplungs-Lernen. Nach einer kurzen Einführung in dieses Lernparadigma wird es zur Ableitung entsprechender Lernverfahren für eine DCS-basierte adaptive Sakkadensteuerung und Hand-Auge-Koordination genutzt. Kapitel 6 bietet eine Einführung in das allgemeine Reinforcement-Lernproblem und untergliedert sich danach in das Reinforcement-Lernen als diskretes Markovsches Entscheidungsproblem und das reellwertige Reinforcement-Lernen. Beide Lernparadigmen werden ausführlich diskutiert und zur Entwicklung neuer Lernverfahren genutzt. Insbesondere wird ein Weg zur Verschmelzung von Watkins' Q-Lernen mit Suttons TD(λ)-Lernen aufgezeigt. Anwendungen sind das Stabbalancieren und die hindernisvermeidende Navigation eines mobilen Roboters. Den Abschluß bildet Kapitel 7 mit einem Fazit und Ausblick.

Die vorliegende Arbeit bildet auch die Klammer um eine Reihe von Zeitschriftenartikeln ([29, 26, 24] u. [57][2]) Konferenzbeiträgen ([30, 19, 3, 23, 25, 22, 56, 20]) und Technischen Berichten ([28, 27, 21]) des Autors, indem die dort präsentierten Ergebnisse in einen Gesamtzusammenhang gestellt werden. Für die Gelegenheit zur schritthaltenden Veröffentlichung und die Förderung der Teilnahme an wichtigen Konferenzen über die letzten Jahre sei an dieser Stelle dem Institut für Informatik und Praktische Mathematik der Christian-Albrechts-Universität zu Kiel ausdrücklich gedankt. So gaben zahlreiche anonyme Gutachter ein für die Einschätzung der eigenen Arbeit unerläßliches reichhaltiges Feedback, und so mancher wichtige Denkanstoß erfolgte auf einer Konferenz. Dieser Politik ist es auch zu verdanken, daß bereits vor Abschluß dieser Arbeit viele der hier präsentierten Gedanken und Ergebnisse von anderen Forschungsgruppen als Anregung und Grundlage für weiterführende Arbeiten aufgegriffen werden konnten, worüber ich mich besonders freue. Beispiele sind hier [116, 36, 49].

[2] als Koautor

Kapitel 2

Existierende Neuronale Modelle mit lokal rezeptiven Feldern

Dieses Kapitel bietet eine kurze Einführung in das Training von KNN und einen Überblick über die bekanntesten KNN-Modelle mit lokal rezeptiven Feldern. Im Rahmen der Kurzeinführung werden dabei insbesondere diejenigen Konzepte und mathematischen Beschreibungsansätze betont, die als Grundlage für das Verständnis der folgenden Kapitel dienen. Für eine ausführliche Einführung in das Training von KNN unter dem Zugang der statistischen Mustererkennung ist [15] zu empfehlen. Der Überblick über die existierenden KNN-Modelle umfaßt das Modell von Kohonen [71], das Neuronengas und die perfekt topologieerhaltenden Karten von Martinetz [80] sowie die Netzwerke radialer Basisfunktionen (RBF-Netze). Sie bilden den Ausgangspunkt für die Entwicklung der Dynamischen Zellstruktur in Kapitel 3.

2.1 Generelle Bemerkungen zum Training von KNN

2.1.1 Überwachtes versus Unüberwachtes Lernen

Überwachtes Lernen

Ein Lernverfahren wird als *überwacht* bezeichnet, wenn zu jeder Trainingseingabe x aus dem Eingaberaum I eine zugehörige Sollausgabe t des Netzwerkes aus dem Ausgaberaum A geliefert und zur Adaption des Modells genutzt wird.

Im allgemeinen[1] wird ein überwachtes Lernverfahren durch Angabe

- einer Fehlerfunktion $Err : T \subseteq I \times A \rightarrow \mathbb{R}$, die den Fehler auf der Trainingsmenge mißt, und
- eines Optimierungsverfahrens zur Minimierung von Err

[1]Es gibt auch überwachte Lernverfahren, für die keine Fehlerfunktion angegeben werden kann und die allein durch Angabe einer Adaptionsregel beschrieben werden. Ein Beispiel ist die erweiterte Kohonenregel (Seite 20).

spezifiziert. Die Trainingsmenge T kann dabei entweder als eine endliche Menge von Trainingsdaten $T = \{(x,t)_1, \ldots, (x,t)_m\}$, $m = |T|$, vorliegen oder aber eine unendliche, nur implizit über die Verbundwahrscheinlichkeit $p(x,t) = p(t|x)\,p(x)$ definierte Menge sein. Übliche Fehlerfunktionen sind die *Minkowski-R* Fehlerfunktionen

$$Err^R = \sum_{(x,t)\in T} \sum_{i=1}^{m} |y_i(x) - t_i|^R \quad \text{bzw.} \quad Err^R = \int\int \sum_{i=1}^{m} |y_i(x) - t_i|^R\, p(t|x)\, p(x) dt\, dx \quad (2.1)$$

oder die *Kreuzentropie*, falls die Soll- und Istausgaben Wahrscheinlichkeiten repräsentieren:

$$\begin{aligned} Err^{cross} &= -\sum_{(x,t)\in T} (t \ln y(x) + (1-t)\,(1 - \ln y(x)) \quad \text{bzw.} \qquad (2.2) \\ Err^{cross} &= \int\int (t \ln y(x) + (1-t)\,(1 - \ln y(x))\, p(t|x)\, p(x)\, dt\, dx. \end{aligned}$$

Als wichtige Spezialfälle der Minkowski-R Fehlerfunktionen erhält man für $R = 2$ (L_2-Norm) die *quadratische Fehlerfunktion* Err^2 und für $R = 1$ (L_1-Norm) die *City-Block-Metrik*. Die Minkowski-R Fehlerfunktionen weisen eine wahrscheinlichkeitstheoretische Motivation auf (siehe z.B. [16]), denn es läßt sich zeigen, daß ihre Minimierung äquivalent mit der Maximierung der Verbundwahrscheinlichkeit (Likelihood) der Trainingsmenge $\mathcal{L} = \Pi_{(x,t)\in T}\, p(t,x)$ ist, wenn die Dichteverteilung des Rauschens auf den Sollausgaben die Form

$$p(\epsilon) = \frac{R\beta^{1/R}}{2\Gamma(1/R)} \exp(-\beta|\epsilon|^R). \qquad (2.3)$$

annimmt. Im Falle der quadratischen Fehlerfunktion ergibt sich die Gaußverteilung, im Falle der City-Block-Metrik die Laplaceverteilung.

Die quadratische Fehlerfunktion hat zwei weitere Eigenschaften, die sie zum Einsatz in Regressions- und Klassifikationsproblemen prädestinieren und die im Rahmen dieser Arbeit ausgenutzt werden. Zum einen läßt sich zeigen (siehe z.B. [16]), daß der quadratische Fehler genau dann minimal wird, wenn die Istausgabe dem Erwartungswert der Sollausgabe entspricht, d.h.

$$y(x) = \int t\, p(t|x)\, dt = < t|x >, \qquad (2.4)$$

das Ziel der allgemeinen *Regressionsaufgabe*. Für *Klassifikationsaufgaben* mit einer 1-aus-m Kodierung ($t_i = 1$ gdw. $x \in C_i$) folgt aus (2.4), daß die Istausgabe, die $Err^2(T)$ minimiert, die posteriori Klassenzugehörigkeitswahrscheinlichkeiten widerspiegelt, also

$$y_i(x) = P(C_i|x), \qquad (2.5)$$

und damit eine Bayes-optimale Klassifikation erlaubt. Voraussetzung zum Erlernen der optimalen Istausgaben ist allerdings, daß die Trainingsmenge genügend groß ist, daß das Approximationsschema genügend flexibel ist, um $E[t|x]$ zu modellieren, und daß ein geeignetes Trainingsverfahren zur Minimierung von $Err^2(T)$ zur Verfügung steht.

Ist die Fehlerfunktion definiert, so steht zur Adaption der Modellparameter im Falle einer endlichen Trainingsmenge (batch Training) ein Repertoir von (nichtlinearen) numerischen Optimierungsverfahren zur Verfügung (siehe z.B. [103], [139]). Im Falle zufällig online generierter, potentiell unendlich großer Trainingsmengen (Online-Training) kommen Verfahren der stochastischen Approximation zum Einsatz (siehe hierzu auch Abschnitt 2.1.2). Im Zusammenhang mit KNN wie MLP-Netzen, RBF- und DCS-Netzen ist von wesentlicher Bedeutung, daß die Netzausgabe $y(x)$ nach den Parametern des Netzes differenzierbar ist und die Berechnung der partiellen Ableitungen nach der Kettenregel linear mit der Verbindungszahl skaliert. Damit sind auch Fehlerfunktionen wie der Minkowski-R Fehler und die Kreuzentropie nach den Parametern differenzierbar und erlauben den Einsatz gradientenbasierter Optimierungsverfahren. Die beiden am meisten verbreiteten gradientenbasierten Verfahren sind Konjugierter Gradientenabstieg und der Levenberg-Marquard-Algorithmus [15]. Einfacher Gradientenabstieg ist im Zusammenhang mit MLP-Netzen auch als *Backpropagation* bekannt.

Unüberwachtes Lernen

Ein Lernverfahren wird als *unüberwacht* oder *selbstorganisierend* bezeichnet, wenn es ein Modell ohne Rückkoppelung einer Sollausgabe oder eines Fehlersignals adaptiert.

Damit ist unüberwachtes Lernen allein von den Trainingseingaben $x \in E$ abhängig. Die Kohonenregel (2.12), die Adaptionsregel des Neuronengases (2.19) und die Fritzke-Lernregel (3.29) sind Beispiele unüberwachter (online) Lernregeln. In einigen Fällen, wie z.B. beim Neuronalen Gas, läßt sich zeigen, daß die Lernregel eine auf den Eingaben definierte Energiefunktion minimiert. Für andere Lernregeln, wie z.B. die Kohonenregel, scheint eine solche Energiefunktion nicht formulierbar, bzw. konnte trotz intensiver Bemühungen bislang nicht gefunden werden [70]. Dennoch konvergieren auch diese Regeln typischerweise zu einer charakteristischen Lösung, allein die mathematische Beschreibung des Ergebnisses des Selbstorganisationsprozesses gelingt nur unzureichend.

Im Rahmen von KNN-Modellen mit lokal rezeptiven Feldern ist unüberwachtes Lernen vor allem bei der Verteilung einer Menge $S = \{c_1, \ldots, c_N\}$ von *Zentren* $c_i \in \mathbb{R}^n$ (den Zentren der rezeptiven Feldern) im Eingaberaum $\mathbb{R}^n$ von Bedeutung. Ziel ist dabei die Modellierung der Eingangswahrscheinlichkeitsdichte $p(x)$ durch die Dichte der Zentren $P(c)$ im Eingaberaum oder die Minimierung des quadratischen Vektorquantisierungsfehlers

$$Err^{VQ} = \sum_{c_i \in S} \int_{V_i} (x - c_i)^2 \, p(x|V_i) \, dx \, P(V_i), \tag{2.6}$$

wobei $V_i \subseteq \mathbb{R}^n$ die Voronoizelle von Zentrum c_i bezeichnet[2]. Die Minimierung von Err^{VQ} hängt dabei mit der Modellierung der Wahrscheinlichkeitsdichte der Eingaben insofern zusammen, als im Grenzfall unendlich vieler Zentren für die Zentrenverteilung, die Err^{VQ} minimiert,

$$P(c) \sim p(x)^{\zeta} \tag{2.7}$$

mit $\zeta = \frac{n}{n+2}$ gilt[3]. Die Bedeutung von (2.7) besteht darin, daß Vektorquantisierungsverfahren zur Modellierung der Wahrscheinlichkeitsdichte genutzt werden können und im Falle

[2] Das zentrale Anliegen der Vektorquantisierung besteht darin, eine gemäß einer Dichtefunktion $p(x)$ verteilte Menge von Vektoren $x \in \mathbb{R}^n$ so mit einem endlichen Codebuch $S \subset \mathbb{R}^n$ zu repräsentieren (zu quantisieren), daß der Quantisierungsfehler Err^{VQ} möglichst minimal wird.

[3] Dies folgt direkt aus einem allgemeineren Resultat in [138], wonach der Minkowski-R Vektorquantisierungsfehler $\sum_{c_i \in S} \int_{V_i} |x - c_i|^R \, p(x|V_i) \, dx \, P(V_i)$ durch eine Zentrenverteilung mit $\zeta = \frac{n}{n+R}$ minimiert wird.

von $n \to \infty$ das informationstheoretische Optimum ($\zeta = 1$) mit dem optimalen Vektorquantisierer zusammenfällt. Der Koeffizient ζ wird als *Auflösungsfaktor* bezeichnet und kann zur Charakterisierung von unterschiedlichen Selbstorganisationsprozessen herangezogen werden, für die er unterschiedliche Werte annimmt (siehe Abschnitte 2.2.1 und 2.3). Die Steuerung des Auflösungsfaktors durch gezielte Beeinflussung der Lernparameter in Selbstorganisationsprozessen [11] bietet in diesem Zusammenhang einen interessanten Ansatz, auf den hier aber nicht näher eingegangen werden kann.

2.1.2 Batch- versus Online-Lernen

Batch-Lernen zeichnet sich dadurch aus, daß die Generierung der Trainingsmenge $T \subseteq I \times A$ und die Adaption auf Grundlage von T in zwei getrennten, aufeinanderfolgenden Phasen erfolgt. Beim *Online-* oder *Schritthaltenden-Lernen* hingegen sind Datengenerierungs- und Adaptionsphase nicht getrennt, es erfolgt ein Adaptionsschritt stets auf Basis eines einzelnen Trainingsdatums oder weniger aktueller Trainingsdaten. Online-Lernen kann im Rahmen adaptiver Echtzeitsysteme erforderlich werden, es eignet sich aber auch zur Parameteranpassung an zeitlich langsam variierende Prozesse. Darüberhinaus ist es speicherplatzsparend, da nur das letzte, aktuelle Trainingsdatum gespeichert werden muß.

Während Batch-Lernen mit endlichen Trainingsmengen arbeitet, eignet sich Online-Lernen zur Verarbeitung potentiell unendlicher Trainingsmengen, wie sie durch Observierung eines Zufallsprozesses entstehen. Für Online-Lernen ergibt sich ein *Stabilität/Plastizität-Dilemma*: Das Lernverfahren kann im allgemeinen nicht gleichzeitig maximal plastisch sein, d.h. alle Trainingseingaben gleichermaßen bei der Adaption berücksichtigen, und Stabilität, d.h. ein definiertes Verhalten, garantieren. Im Rahmen der stochastischen Approximation (siehe 2.1.3) wird dieses Dilemma zu Ungunsten der Plastizität durch eine geeignete Verkleinerung der Lernraten mit der Zeit gelöst.

Batch-Lernen hingegen ist aufgrund der endlichen Trainingsmengen mit dem *Generalisierungsproblem* bzw. *Overfittingproblem* konfrontiert: Mit einem geeigneten Modell und genügend hoher Modellordnung (Zahl der freien Parameter) ist es immer möglich, die Trainingsmenge und damit das Rauschen auf den Trainingsdaten "auswendig" zu lernen. Das Antwortverhalten für neue Eingaben (Generalisierung) hingegen bleibt unzufriedenstellend. Das Generalisierungsproblem steht in engem Zusammenhang mit dem *Bias/Variance-Dilemma* [39], das sich im Falle der L_2-Norm (vgl. [15]) wie folgt darstellt: Betrachtet man die über alle möglichen Trainingsmengen T gemittelte quadratische Abweichung der Istausgabe vom Erwartungswert der Sollausgabe $< t|x >$, so läßt sich diese als Summe eines Bias- und Varianz-Terms schreiben, d.h..

$$E_T[\{y(x) - < t|x >\}^2] = \underbrace{\{E_T[y(x)] - < t|x >\}^2}_{\text{Bias}} + \underbrace{E_T[\{y(x) - E_T[y(x)]\}^2]}_{\text{Varianz}}.$$

Der Bias gibt hierbei an, wie stark die über alle Trainingsmengen gemittelte Abweichung der Istausgabe vom Erwartungswert der Sollausgabe ist, die Varianz, wie sensibel die Istausgabe auf unterschiedliche Trainingsmengen reagiert. Für den Extremfall einer konstanten Istausgabe (unabhängig von der Trainingsmenge) ergibt sich der Varianzanteil zu null, der Bias hingegen wird im allgemeinen beträchtlich sein. Dagegen ergibt sich im Fall des

Auswendiglernens von Trainingsmengen der Beitrag des Bias-Terms zu null, während die Varianz gerade der Rauschvarianz auf den Trainingsmengen entspricht.

Für eine gegebene (potentiell zu hohe) Modellordnung läßt sich das Generalisierungsproblem durch *Regularisierung* lösen. Weitere Lösungsansätze für das Generalisierungsproblem bilden das Training mit künstlich verrauschten Daten und der frühzeitige Abbruch des Lernvorgangs [16].

Eine Anpassung der Modellordnung an die zu lernende Funktion zur Vermeidung des Generalisierungsproblems erfolgt im Rahmen dieser Arbeit durch Beobachtung des Ausgabeverhaltens auf von der Trainingsmenge unabhängigen Testmengen (*Kreuzvalidierung*). Die optimale Modellordnung ist dann erreicht, wenn das Ausgabeverhalten auf den Testmengen optimal ist. Ein anderer Ansatz ist, einen mit der Modellordnung monoton steigenden Bestrafungsterm einzuführen. Für lineare Modelle ist Akaike's *final prediction error (FPE)* [4] ein solcher Ansatz. Im Zusammenhang mit KNN entwickelte Moody [91] das *generalized prediction error* (GPE) Kriterium, das auf die effektive Anzahl von Freiheitsgraden des Modells Bezug nimmt und hierdurch auch eventuell vorhandene Regularisierungsterme berücksichtigt.

2.1.3 Stochastische Approximation

Gegenstand der von Robbins und Monro [111] begründeten *Stochastischen Approximation* sind Verfahren, um die Nullstellen von stochastisch definierten Funktionen zu finden. Sie eignen sich insbesondere zum Online-Training von KNN, und tatsächlich stellen die meisten existierenden (und die in dieser Arbeit diskutierten) Verfahren zum Online-Training stochastische Approximationsverfahren dar.

Sei g eine vom Parameter θ abhängige Zufallsvariable (Zufallsfunktion), wobei θ selber eine Zufallsvariable ist. Sei ferner $f(\theta) = E[g|\theta]$ der Erwartungswert von g bei gegebenem θ. Dann ist das Robbins-Monro Verfahren[4] zur Bestimmung der Nullstelle $\theta^\star$ von f mit $f(\theta^\star) = 0$ gegeben durch die rekursive Folge von Schätzern

$$\theta_{t+1} = \theta_t + \alpha_t\, g(\theta_t), \tag{2.8}$$

wobei die Koeffizienten α_t den folgenden zwei Bedingungen genügen müssen:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \quad \text{und} \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty. \tag{2.9}$$

Die Relevanz dieser Verfahren für die Online-Optimierung von KNN folgt aus der Tatsache, daß die meisten Fehlerfunktionen Err die Form $Err(T) = E[err(x,t)]$ aufweisen, wobei

[4] Unter Voraussetzung der Eindeutigkeit der Nullstelle und beschränkter Varianz von $g(.)$ zeigten Robbins und Monro, daß das Verfahren unter der Bedingung $(\theta - \theta^\star) f(\theta) > 0$ mit Wahrscheinlichkeit 1 zu $\theta^\star$ konvergiert. In [64] wird gezeigt, daß das Verfahren auch für den vektorwertigen Fall $\theta, g(.), f(.) \in \mathbb{R}^n$ mit Wahrscheinlichkeit 1 zu $\theta^\star \in \mathbb{R}^n$ konvergiert, falls $f(.)$ der Linearitätsbedingung

$$\inf_{\epsilon < ||(\theta - \theta^\star)|| < \epsilon^{-1}} \{(\theta - \theta^\star)^T f(\theta)\} > 0 \quad \forall \epsilon > 0$$

genügt.

$err : I \times A \to \mathbb{R}$ den Fehler auf dem Trainingsbeispiel (x, t) mißt (so auch Err^R und Err^{cross} nach (2.1) bzw. (2.2)). Ist err nach den Parametern W des KNN differenzierbar, so ist $E[\nabla_{W^\star} err] = 0$ notwendige Bedingung für ein lokales Maximum von Err in $W^\star$. Mit der Robbins-Monro-Prozedur

$$W_{t+1} = W_t + \alpha_t \nabla_{W_t} err_t \tag{2.10}$$

und α_t entsprechend (2.9) kann $W^\star$ dann online bestimmt werden. Dieses Verfahren, das auch unter dem Namen *stochastischer Gradientenabstieg* bekannt ist, bildet die Grundlage der meisten Online-Trainingsverfahren für KNN[5].

Die Robbins-Monro-Prozedur zur Online-Optimierung löst das Stabilität/Plastizität-Dilemma zugunsten der Stabilität (Konvergenz mit Wahrscheinlichkeit 1) durch "Einfrieren" der Lernrate α_t in (2.9). Dabei sorgt die erste Bedingung dafür, daß die Korrekturen groß genug sind, um die Nullstelle zu finden, während die zweite Bedingung garantiert, daß das aufakkumulierte Rauschen beschränkte Varianz aufweist und den Konvergenzprozeß nicht stören kann.

2.2 Selbstorganisierende Merkmalskarten (SOM)

In diesem Abschnitt wird das wohl bekannteste[6] KNN-Modell mit lokal rezeptiven Feldern vorgestellt, die *Selbstorganisierenden Merkmalskarten* (SOM) von Kohonen [71], die auch unter dem Namen *Kohonenkarten* bekannt sind. Die Kohonenkarten zeichnen sich durch ein Gitter aus, auf dem die Berechnungsknoten angeordnet sind und über das eine Wechselwirkung mit bezüglich des Gitters benachbarten Knoten stattfindet. Je nach Dimensionalität des Gitter spricht man von d-dimensionalen Kohonenkarten.

Während Kohonenkarten eine stark neurophysiologische Motivation aufweisen (die kortikalen Karten)[7], beschränkt sich Abschnitt 2.2.1 auf die Darstellung der algorithmischen Aspekte des Modells von Kohonen. Das Modell von Kohonen, das ursprünglich ein rein unüberwachtes Lernverfahren im Sinne von Abschnitt 2.1.1 beschreibt, wurde von Ritter et al. auf ein überwachtes Lernmodell erweitert, das in Abschnitt 2.2.2 erläutert wird. Als Kombination von unüberwachtem und überwachtem Lernen weist diese *Erweiterte Kohonenkarte* allerdings spezifische Probleme auf, die im Rahmen von Klassifikationsaufgaben durch die Einführung der *Lernenden Vektorquantisierer* (LVQ) zu beheben versucht wurden. Letztere werden in Abschnitt 2.2.3 näher beleuchtet.

2.2.1 Kohonenkarten

Die d-dimensionale Kohonenkarte besteht aus N auf einem d-dimensionalen Gitter L angeordneten Knoten, wobei jedem Knoten ein n-dimensionaler Zentrumsvektor $c_i \in \mathbb{R}^n$

[5] Im Falle nichtlinearer Berechnungsfunktionen wird $E[\nabla_W err]$ im allgemeinen mehrere Nullstellen aufweisen. Der Robbins-Monro-Prozeß konvergiert dann zu einem lokalen Maximum von Err [70].

[6] Eine fortlaufend von der Helsinki KNN-Gruppe um T. Kohonen aktualisierte Referenzenzliste zählt bereits mehr als 3000 Einträge.

[7] Siehe hierzu [70] und [109].

zugeordnet ist. Eine Eingabe $x \in \mathbb{R}^n$ wird auf den Knoten abgebildet, dessen Zentrumsvektor den geringsten euklidischen Abstand zu x hat, die *best matching unit*, bmu_x. Damit beschreibt die Kohonenkarte eine Abbildung $f^{Kohonen} : \mathbb{R}^n \to \{1, \ldots, N\}$ mit

$$bmu_x = f^{Kohonen}(x) = \arg\min_j \{\parallel x - c_j \parallel\}. \tag{2.11}$$

Nach Bestimmung der bmu_x werden dann in der Lernphase die Zentrumsvektoren online nach dem Selbstorganisationsprozeß

$$c_i(t+1) = c_i(t) + \epsilon(t) g^{Kohonen}_{bmu_x,i}(t)\,(x - c_i(t)) \tag{2.12}$$

adaptiert, wobei die Funktion

$$g^{Kohonen}_{bmu_x,i}(t) = \exp(-\frac{d_L(i, bmu_x)}{2\sigma^2(t)}) \tag{2.13}$$

als *Nachbarschaftsfunktion* dafür sorgt, daß nicht nur das Zentrum der bmu_x, sondern auch die Zentren bezüglich des Gitters benachbarter Knoten in Richtung der Eingabe adaptiert werden. Die Abstandsfunktion $d_L(i, j)$ mißt dabei den Abstand von Knoten i und j bezüglich des Gitters L. Üblicherweise wird sowohl der Kooperationsparameter $\sigma(t)$ als auch die Lernrate $\epsilon(t)$ als monoton fallende Funktion der Zeit t gewählt und damit das Stabilität/Plastizität-Dilemma zugunsten der Stabilität gelöst.

Was leistet die Kohonenkarte? Zuerst einmal fällt auf, daß für $\sigma \to 0$ in (2.13) nur noch die bmu_x am Selbstorganisationsprozeß (2.12) beteiligt ist und damit (2.12) identisch mit der *Nearest Neighbor Regel* wird

$$c_i(t+1) = \begin{cases} c_i(t) + \epsilon(t)(x - c_i(t)) : & \text{falls } i = bmu_x \\ 0 : & \text{sonst} \end{cases}, \tag{2.14}$$

die wiederum nichts anderes als die Robbins-Monro-Prozedur zur Minimierung des Vektorquantisierungsfehlers Err^{VQ} nach (2.6) darstellt. Der Auflösungsfaktor ergibt sich in diesem Fall zu $\zeta = \frac{n}{n+2}$. Für $\sigma \neq 0$ ließ sich bislang keine Fehlerfunktion aufstellen, die durch (2.12) minimiert wird, genauso wie der Auflösungsfaktor bislang nur für den Spezialfall 1-dimensionaler Eingaberäume und Kohonengitter zu $\zeta = \frac{2}{3}$ berechnet werden konnte [110]. Die Kohonenkarte mit Nachbarschaftskooperation scheint sich der mathematischen Analyse zu entziehen [70]. Auch wenn sich die Auflösungsfaktoren nicht berechnen lassen, gilt doch offensichtlich, daß die Verteilung der Zentren der Kohonenkarte der Wahrscheinlichkeitsdichte der Eingaben folgt. Verglichen mit der Nearest Neighbor Regel bewirkt die Nachbarschaftskooperation eine Beschleunigung der Konvergenz (gemessen als Anzahl der Iterationsschritte), da nach (2.14) in einem Adaptionsschritt stets alle Knoten adaptiert werden. Wichtiger aber ist, daß (2.14) einen Zwang derart auf das System ausübt, daß bezüglich des Gitters benachbarte Knoten Zentrumsvektoren annehmen, die im Eingaberaum benachbart sind. Damit eignen sich Kohonenkarten insbesondere zur Darstellung (Visualisierung) hochdimensionaler Daten auf niedrigdimensionalen Karten. Übersteigt die intrinsische (topologische) Dimensionalität der Daten jedoch die Dimensionalität der Kohonenkarte, so wird es zu Faltungen der Karte innerhalb der Datenmannigfaltigkeit kommen.

In diesem Fall ist zwar immer noch gewährleistet, daß bezüglich des Gitters benachbarte Knoten benachbarte Stützstellen im Eingaberaum aufweisen, umgekehrt wird es aber im Eingaberaum benachbarte Zentren geben, die bezüglich des Gitters nicht benachbarten Knoten zugeordnet werden. Die Kohonenkarte ist damit im allgemeinen nicht topologieerhaltend.

2.2.2 Erweiterte Kohonenkarten

Ritter et al. [108] erweiterten die Kohonenkarte für überwachte Lernaufgaben durch zusätzliche Speicherung eines Ausgangsvektors $o_i \in \mathbb{R}^n$ an jedem Knoten des Gitters. Die erweiterte Kohonenkarte beschreibt eine Abbildung $f^{EKK} : \mathbb{R}^n \to \mathbb{R}^m$,

$$f^{EKK}(x) = o_{bmu_x}. \tag{2.15}$$

Die Abbildung stellt ein reines Table-Lookup-Verfahren dar und ist als solches zur universellen Approximation stetiger Funktionen auf kompakten Teilmengen des $\mathbb{R}^n$ befähigt. Die Adaption der erweiterten Kohonenkarte erfolgt online, wobei bei Präsentation eines Trainingsbeispiels $(x, t) \in T$ die Zentren nach der Kohonenregel (2.12) adaptiert werden und die Ausgabevektoren nach

$$o_i(t+1) = o_i(t) + \alpha(t) g^{EKK}_{bmu_x,i}(t)\,(t - o_i(t)). \tag{2.16}$$

Dabei wird $g^{EKK}_{bmu_x,i}(t)$ analog zur Nachbarschaftsfunktion $g^{Kohonen}_{bmu_x,i}(t)$ gewählt, allerdings mit gegebenenfalls unterschiedlichem Nachbarschaftsradius $\sigma'^2(t)$

$$g^{EKK}_{bmu_x,i}(t) = \exp(-\frac{d_L(i, bmu_x)}{2\sigma'^2(t)}). \tag{2.17}$$

Für eine gegebene Zentrenverteilung und $\sigma' \to 0$ stellt (2.16) offensichtlich gerade die Robbins-Monro-Prozedur zur Minimierung des quadratischen Fehlers $Err^2(T)$ dar, womit die Ausgabevektoren o_i gegen den Erwartungswert der Sollwerte der Trainingsbeispiele in den entsprechenden Voronoizellen c_i konvergieren. Für $\sigma' \neq 0$ wirkt der Nachbarschaftskooperationseffekt wie im Falle der einfachen Kohonenkarte konvergenzbeschleunigend, eine Fehlerfunktion aber läßt sich nicht mehr angeben. Dabei regularisiert die Nachbarschaftskooperation den Adaptionsprozeß derart, daß bezüglich des Gitters benachbarte Knoten im Ausgaberaum benachbarte Ausgabevektoren annehmen. Dieser Regularisierungseffekt ist insbesondere beim Erlernen mehrdeutiger Abbildungen (wie in Abschnitt 5.3) von Bedeutung.

Kritisch ist zu den erweiterten Kohonenkarten anzumerken, daß die Adaption der Zentren und damit die Stützstellenverteilung unabhängig vom Approximationsfehler erfolgt. Aufgrund der Kohonenregel (2.12) folgt die Stützstellenverteilung nämlich stets der Wahrscheinlichkeitsverteilung und erzeugt so gegebenenfalls eine hohe Auflösung in Regionen, in denen die Approximation trivial ist (z.B. konstante Funktion), und umgekehrt. Auch wirkt sich der Nachbarschaftskooperationseffekt negativ auf die Approximationseigenschaften aus, wenn die intrinsische (topologische) Dimensionalität der Daten nicht mit der a priori festgelegten Dimensionalität der Kohonenkarte übereinstimmt. Schließlich fehlt den erweiterten Kohonenkarten als Approximationsverfahren 0.ter Ordnung (Table-Lookup-Verfahren) jede Interpolationsfähigkeit. Dies führt verglichen mit Approximationsverfahren höherer Ordnung im allgemeinen zu einer weitaus höheren Anzahl von Stützstellen für dieselbe Approximationsgüte[8].

2.2.3 LVQ

Dem Hauptnachteil der (erweiterten) Kohonenkarten, der fehlenden Sensitivität der Zentrenadaptionsregel bezüglich des Ausgabefehlers, wurde mit der Entwicklung der *Lernenden Vektorquantisierer* [70] entgegenzusteuern versucht. Diese für Klassifikationsaufgaben entwickelten statistischen Approximationsverfahren nutzen zu einer am Klassifikationsfehler orientierten Zentrenadaption die mit den Knoten assoziierten Ausgabevektoren (d.h. die Klassenzugehörigkeiten). So lautet die LVQ1-Lernregel[9] zur Zentrenadaption

$$c_i(t+1) = c_i(t) + \begin{cases} \epsilon(t)\,(x - c_i(t)) : & \text{falls } i = bmu_x \text{ und } x \text{ zur Klasse } o_i \text{ gehört} \\ \epsilon(t)\,(c_i(t) - x) : & \text{falls } i = bmu_x \text{ und } x \text{ nicht zur Klasse } o_i \text{ gehört} \\ 0 \qquad : & \text{sonst} \end{cases} . \tag{2.18}$$

Ziel der LVQ1-Regel ist es, eine Zentrenverteilung zu generieren, die nicht mehr die Wahrscheinlichkeitsdichte der Eingaben $p(x)$ sondern die a posteriori Wahrscheinlichkeiten der Klassenzugehörigkeit $p(o_i|x)$ modelliert, um so eine Bayes-optimale Klassifikation zu erlauben, [70]. Man beachte, daß die LVQ-Regeln ohne Nachbarschaftskooperation arbeiten.

2.3 Neuronengas und Topologieerhaltende Netze

Das von Martinetz im Rahmen seiner Dissertation [80] entwickelte *Neuronengas* stellt ein weiteres populäres KNN-Modell lokal rezeptiver Felder dar. Es zeichnet sich dadurch aus, daß die Aktivierung von Berechnungsknoten gemäß einer Funktion der Sortierreihenfolge bezüglich des euklidischen Abstandes von der Eingabe x erfolgt, und nicht wie beim Kohonengitter als Funktion des Gitterabstandes von der bmu_x. Es existiert keine Verbindungsstruktur zwischen den Berechnungsknoten. Ähnlich wie das Modell von Kohonen wurde das Neuronengas ursprünglich zum unüberwachten Lernen entwickelt und später für überwachte Lernaufgaben erweitert.

Neben dem Neuronengas führte Martinetz die "Perfekt Topologieerhaltenden Merkmalskarten" und die Klasse der "Topologieerhaltenden Netze" ein, die für die Dynamische Zellstruktur von zentraler Bedeutung sind.

2.3.1 Neuronengas

Wie die Kohonenkarte besteht das Neuronengas aus N Berechnungsknoten, wobei jedem Knoten ein n-dimensionaler Zentrumsvektor $c_i \in \mathbb{R}^n$ zugeordnet ist. Eine Eingabe $x \in \mathbb{R}^n$

[8]Dieser Nachteil wird durch Ritters *Lokal Lineare Karten* (LLM) als lokal lineare Approximationsverfahren behoben.

[9]Es existieren mehrere LVQ-Varianten (siehe [70] für einen Überblick).

wird wie bei der Kohonenkarte nach (2.11) auf die bmu_x abgebildet. Nach Bestimmung der bmu_x werden dann in der Lernphase die Zentrumsvektoren online nach dem Selbstorganisationsprozeß

$$c_i(t+1) = c_i(t) + \epsilon(t) g^{NG}_{ord(i)}(t)(x - c_i(t)) \tag{2.19}$$

adaptiert, mit

$$g^{NG}_{ord(i)}(t) = \exp(-\frac{ord(i)}{\sigma(t)}). \tag{2.20}$$

Dabei gibt $ord(i)$ die euklidische *Nachbarschaftsrangfolge* von Zentrum c_i bezüglich der Eingabe x an: Ist $\{c_{i_1}, \ldots, c_{i_N}\}$ die bezüglich des euklidischen Abstandes zu x aufsteigend sortierte Folge von Zentrumsvektoren, so ist $ord(i_j) \stackrel{def}{=} j - 1$. Ähnlich der Kohonenkarte werden sowohl die Lernkonstante ϵ als auch der Kooperationsparameter σ mit der Zeit auf null abklingen, um Konvergenz zu erzwingen.

Im Gegensatz zur Kohonenkarte konnte Martinetz in [80] für das Neuronengas eine Kostenfunktion angeben, auf der (2.19) einen stochastischen Gradientenabstieg und damit einen Robbins-Monro-Prozeß implementiert:

$$Err^{NG} = \frac{1}{2\sum_{i=1}^{N} g^{NG}_{ord(i)}} \sum_{i=1}^{N} \int g^{NG}_{ord(i)} (x - c_i)^2 \, p(x) \, dx \tag{2.21}$$

Interessant ist hierbei, daß für $\sigma \to \infty$ das Fehlermaß Err^{NG} parabolische Gestalt annimmt, für $\sigma \to 0$ hingegen Err^{NG} in den quadratischen Fehler Err^{VQ} übergeht, dessen Minimierung durch lokale Minima erschwert wird. Dies bildet die Motivation für eine dem *simulated annealing* ähnliche "Abkühlung" von σ zur Minimierung von Err^{VQ}, siehe [80]. Des weiteren konnte Martinetz zeigen, daß sein Neuronengas einen Auflösungsfaktor von $\zeta = \frac{n}{n+2}$ aufweist, welcher mit dem Auflösungsfaktor für optimale Vektorquantisierer übereinstimmt. Auch dies unterstreicht die hervorragende Eignung des Neuronengases als Vektorquantisierer.

Als Nachteil des Neuronengases muß die Zeitkomplexität der Berechnung der Nachbarschaftsrangfolge $ord(i)$ genannt werden. Diese ist für jede Eingabe neu zu berechnen und weist serielle Zeitkomplexität $O(N \log N)$ auf. Im Gegensatz hierzu ist die Nachbarschaftsrangfolge in der Kohonenkarte durch das starre d-dimensionale Gitter fest vorgegeben, und die Berechnung der Nachbarschaftsfunktion kann in linearer Zeit erfolgen. Allerdings erfordern die Kohonenkarten Vorwissen zur Anpassung von d an die intrinsische Dimensionalität der Daten, was beim Neuronengas nicht erforderlich ist.

2.3.2 Erweitertes Neuronengas

In Analogie zur erweiterten Kohonenkarte läßt sich auch das Neuronengas für überwachte Lernaufgaben durch zusätzliche Speicherung eines Ausgangsvektors $o_i \in \mathbb{R}^m$ an jedem

Knoten erweitern. Als Evaluierungsfunktion $f^{NG}: \mathbb{R}^n \to \mathbb{R}^m$ ergibt sich dann[10]

$$f^{ENG}(x) = o_{bmu_x}. \tag{2.22}$$

Die Adaption des erweiterten Neuronengases erfolgt ebenfalls analog zur erweiterten Kohonenkarte, wobei bei Präsentation eines Trainingsbeispiels $(x, t) \in T$ die Zentren nach der Lernregel (2.19) und die Ausgabevektoren nach

$$o_i(t+1) = o_i(t) + \alpha(t)(t - o_i(t)) g^{ENG}_{ord(i)} \tag{2.23}$$

mit eventuell von $\sigma(t)$ abweichendem Kooperationsparameter $\sigma'(t)$ adaptiert werden. Für eine gegebene Zentrenverteilung und $\sigma' \to 0$ stellt (2.23) die Robbins-Monro-Prozedur zur Minimierung des quadratischen Fehlers $Err^2(T)$ dar, für $\sigma' \neq 0$ wirkt der Nachbarschaftskooperationseffekt wie auch bei der Kohonenkarte zugleich konvergenzbeschleunigend und regularisierend.

Hauptkritikpunkt des erweiterten Neuronengases ist wie bei den erweiterten Kohonenkarten, daß die Adaption der Zentren und damit die Stützstellenverteilung vollkommen unabhängig vom Approximationsfehler erfolgt.

2.3.3 Topologieerhaltende Netze

Der im Zusammenhang mit KNN-Modellen zentrale Begriff der Topologieerhaltung wurde erstmals von Martinetz [81] exakt definiert: Martinetz bezeichnet einen Graphen genau dann als perfekt topologieerhaltend, wenn er der induzierten Delaunay-Triangulation entspricht. Mit einer einfachen Hebbschen Lernregel konnten nun, wenigstens theoretisch[11], perfekt topologieerhaltende Karten konstruiert werden. Die zentralen Definitionen und Ergebnisse zu perfekt topologieerhaltenden Karten seien an dieser Stelle sinngemäß rekapituliert.

Es bezeichnen

- $P = (G, S)$ ein parametrisierter Graph $G = (V, E)$, in dem jedem Knoten $i \in V$ eineindeutig ein Zentrumsvektor $c_i \in S$ mit $S \subset \mathbb{R}^n$ und $|S| = |V| = N$ zugeordnet ist,

- $M \subseteq \mathbb{R}^n$ eine Mannigfaltigkeit von Merkmalsvektoren,

[10] Anstelle von (2.22) und (2.23) schlägt Martinetz in [80] die Verwendung von

$$f^{ENG}(x) = \frac{1}{\sum_{i=1}^{N} g^{ENG}_{ord(i)}} \sum_{i=1}^{N} o_i g^{ENG}_{ord(i)} \quad \text{und} \quad o_i(t+1) = o_i(t) + \alpha(t) \frac{1}{\sum_{i=1}^{N} g^{ENG}_{ord(i)}} (t - o_i(t)) g^{ENG}_{ord(i)}$$

vor. Dies bietet gegenüber dem reinen Table-Lookup-Verfahren den Vorteil, daß die Anzahl der unterschiedlichen Sortierreihenfolgen in Abhängigkeit von der Eingabe x für eine gegebene Zentrenverteilung die Anzahl der Zentren bei weitem übersteigt und so eine feinere Diskretisierung bei gleicher Stützstellenanzahl ermöglicht. Aufgrund der fest vorgegebenen Anordnung der Zentren wird die Zahl der möglichen Sortierreihenfolgen allerdings *nicht* $N!$ betragen, wie dies in [80] impliziert wird.

[11] Hierauf wird in Abschnitt 3.1 zurückgekommen.

- $V_i = \{x \in \mathbb{R}^n | \forall j \in V, j \neq i : \| x - c_i \| \leq \| x - c_j \| \}$ die Voronoizelle zu Zentrum c_i und
- $V_i^{(M)} = V_i \cap M$ die *maskierte Voronoizelle* zu c_i.

Definition 2.1 (Nachbarschaft in M) *Zwei Zentren $c_i, c_j \in S$ sind* benachbart *in M, falls sie eine gemeinsame Grenze aufweisen, d.h. $V_i^{(M)} \cap V_j^{(M)} \neq \emptyset$.*

Definition 2.2 (Perfekt Topologieerhaltende Merkmalskarte) *Der Graph G heißt* Perfekt Topologieerhaltende Merkmalskarte *von S gegeben M, falls er die* induzierte Delaunay-Triangulation *$D_M(S)$ von S darstellt, d.h.*

$$(i,j) \in E \Leftrightarrow V_i^{(M)} \cap V_j^{(M)} \neq \emptyset.$$

Definition 2.3 (dichte Zentrenmenge) *Eine Zentrenmenge $S = \{c_1, \ldots, c_N\}$ heißt* dicht in M, *falls für jeden Punkt $x \in M$ das Dreieck $\Delta(x, c_{bmu_x}, c_{smu_x})$, das von x sowie seinem nächst und zweitnächst liegenden Zentrum c_{bmu_x} und c_{smu_x} gebildet wird, vollständig in M liegt, d.h. $\Delta(x, c_{bmu_x}, c_{smu_x}) \subseteq M$.*

Satz 2.1 *Wenn die Zentrenmenge S von $P = (G, S)$ dicht in M ist, dann konvergiert $G = (V, E)$ durch die Lernregel*

$$E(t+1) = E(t) \cup \{(bmu_{x(t)}, smu_{x(t)}), (smu_{x(t)}, bmu_{x(t)})\},$$

mit $E(0) = \emptyset$ gegen $D_M(S)$ mit Wahrscheinlichkeit 1.

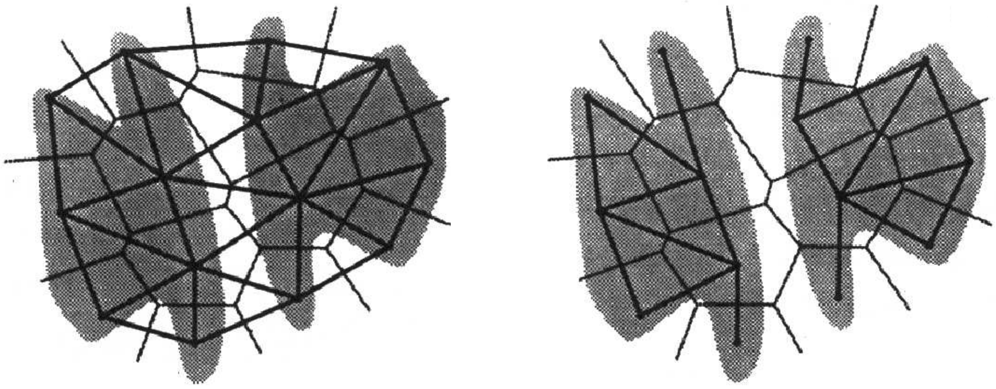
Beweis: Analog zu Beweis von Satz 3 in [81].

Während der perfekt topologieerhaltende Graph also gerade als die induzierte Delaunay-Triangulation definiert wird (eine Illustration findet sich in Abbildung 2.1), zeigt Satz 2.1, wie diese unter der Voraussetzung der Dichtheit von S durch einfaches Verbinden des nächst und zweitnächst liegenden Knotens für jeden gemäß $p(x)$ erzeugten Stimulus konstruiert werden kann. Im Grenzfall unendlich vieler Stimuli konvergiert G dann gegen $D_M(S)$ mit Wahrscheinlichkeit eins. Martinetz führte diese Lernregel als *Hebbsches Wettbewerbslernen* ein. Seine Formulierung

$$\Delta C_{ij} \propto \begin{cases} y_i y_j : & y_i y_j \geq y_k y_l, \quad k, l = 1, \ldots, |V| \\ 0 \quad : & \text{sonst} \end{cases} . \tag{2.24}$$

Hierbei bezeichnet C die Adjazenzmatrix von G (mit $C_{ij} \neq 0 \Leftrightarrow (i,j) \in E$) und y_i die Aktivierung des i.ten Neurons, wobei die Aktivierung eine streng monoton mit dem Abstand $\| c_i - x \|$ im Eingaberaum fallende Funktion sein muß. Ein KNN, das perfekt topologieerhaltende Karten mittels Hebbschen Wettbewerbslernens aufbaut, nennt Martinetz *Topology Representing Network (TRN)*.

Es sei an dieser Stelle noch auf eine interessante Erweiterung des Begriffs der topologieerhaltenden Karten im Rahmen von [2] hingewiesen. In dieser Arbeit konnten sowohl der Begriff als auch die Hebbsche Lernregel für den Fall verallgemeinert werden, daß die Zentren c_i nicht mehr nur Punkte sondern auch affine Teilräume des euklidschen Raums repräsentieren können und das Abstandsmaß als Infimum des euklidschen Abstands zweier Punkte der affinen Teilräume definiert wird.



a) b)

Abbildung 2.1: Perfekt topologieerhaltende Karten: a) Voronoizellen (dünne Striche) und Delaunay Triangulation (dicke Striche) für eine durch die schattierten Regionen gegebene Mannigfaltigkeit im $\mathbb{R}^2$. b) Voronoizellen und induzierte Delaunay Triangulation (nach [81]).

Eigenschaften Perfekt Topologieerhaltender Karten

Martinetz konnte zeigen, daß bezüglich der Lernkomplexität perfekt topologiererhaltender Karten bei einer Gleichverteilung der Stimuli im Eingaberaum

$$|E| + |E| \ln |E| + (\frac{1}{\alpha} - 1)|E|^2 \tag{2.25}$$

eine obere Schranke für die im Mittel zu präsentierende Anzahl von Trainingsbeispielen ist, wenn $|E|$ Kanten zu lernen sind. Der Faktor α, $1 \leq \alpha < \infty$ ist von der Homogenität der Zentrenverteilung abhängig, mit $\alpha = 1$ bei gleich großen Voronoizellen (homogene Verteilung). Ist die intrinsische Dimensionalität der Mannigfaltigkeit $d \leq 2$ und damit G planar und $|E| \in O(N)$, so ist die Musterkomplexität O(N log N) für $\alpha \approx 1$. Sie entspricht damit der optimalen seriellen Zeitkomplexität zur Konstruktion der Delaunay Triangulation [101].

Unter Ausnutzung der Delaunay Triangulation[12] kann eine ganze Reihe von Nachbarschaftsproblemen sehr viel effizienter gelöst werden, siehe [101]. So lassen sich beispielsweise das *Nächste-Nachbar-* und *k-Nächste-Nachbar-Problem* in $O(logN)$ anstelle von $O(N)$ Zeit lösen, und minimale euklidische Spannbäume lassen sich in $O(N)$ anstatt in $O(NlogN)$ Zeit konstruieren.

Ein mögliches Anwendungsgebiet perfekt topologieerhaltender Karten und damit der induzierten Delaunay Triangulation sieht Martinetz im Zusammenhang mit KNN, wo auf Nachbarschaftskooperation beruhende Lern- und Klassifikationsregeln nur dann optimale Resultate liefern, wenn die Topologie perfekt repräsentiert ist. Des weiteren zeigt das in

[12]Die Delaunay Triangulation entspricht der induzierten Delaunay Triangulation nur dann, falls M eine konvexe Teilmenge des $\mathbb{R}^n$ ist.

[80] angeführte Beispiel zur *Pfad-erhaltenden Repräsentation von Mannigfaltigkeiten*, wie perfekt topologieerhaltende Karten zur Navigation genutzt werden können[13].

2.4 Netzwerke radialer Basisfunktionen

Zusammen mit Martinetz' perfekt topologieerhaltenden Karten bilden Netzwerke radialer Basisfunktionen (RBF-Netze) die Grundlage der in Kapitel 3 einzuführenden Dynamischen Zellstrukturen, weshalb dieser Netzwerktyp etwas ausführlicher vorgestellt wird.

RBF-Netze sind Feed-Forward-Netze, die ihre Ausgabe gemäß der Evaluierungsfunktion $f : \mathbb{R}^n \to \mathbb{R}^m$ mit

$$f^{rbf}(x) = \sum_{i=1}^{N} o_i h(\| x - c_i \| / \sigma_i) \quad \text{bzw.} \tag{2.26}$$

$$f^{nrbf}(x) = \frac{\sum_{i=1}^{N} o_i h(\| x - c_i \| / \sigma_i)}{\sum_{j=1}^{N} h(\| x - c_j \| / \sigma_j)} \tag{2.27}$$

berechnen, wobei im zweiten Fall von *Normalisierten RBF-Netzen* gesprochen wird. Die monoton fallende, stetig differenzierbare Aktivierungsfunktion $h : \mathbb{R}^+ \to [0,1]$ mit $h(0) = 1$ und $h(\infty) = 0$ realisiert zusammen mit der Norm $\| \, . \, \|$ die N radialen Basisfunktionen[14]. Die Vektoren $c_i \in \mathbb{R}^n$ werden als *Zentren* der Basisfunktionen bezeichnet, die Skalare σ_i als *Radien* der Basisfunktionen. Die Vektoren $o_i \in \mathbb{R}^m$ stellen die mit jedem Knoten des Netzwerkes assoziierten *Ausgabevektoren* dar.

Wie in Abschnitt 2.4.1 näher ausgeführt, sind RBF-Netze für eine große Klasse von Aktivierungsfunktionen universale Approximatoren. Sie lassen sich als stetig differenzierbare Verallgemeinerung von tabularen Funktionsrepräsentationen auffassen, vgl. [99]. Letztere erhält man im Falle normalisierter RBF-Netze für $\sigma \to 0$. Dies ist aber bei weitem nicht die einzige Interpretation von RBF-Netzen. Es gehört zu den Besonderheiten dieses Netzwerktypes, daß er das Kondensat einer ganzen Reihe von Konzepten bildet, angefangen von der Theorie der exakten Interpolation bis zur Fuzzy-Logic. Dieser universelle Charakter von RBF-Netzen wird in Abschnitt 2.4.2 resümiert.

Neben ihrer Lokalität ist die Existenz schneller (batch) Trainingsverfahren der wesentliche Pluspunkt von RBF- gegenüber MLP-Netzen. Dabei werden in einer ersten Trainingsphase die Zentren c_i und die Radien σ_i mittels eines unüberwachten Trainingsverfahrens gelernt, während in einer zweiten Phase die (lineare) Optimierung der Ausgabeschicht erfolgt, z.B. [90]. Für eine detaillierte Darstellung von Offline- und Online-Lernverfahren sei auf Abschnitt 3.4 verwiesen. Dort werden Lernregeln für DCS-Netze angegeben, die sich, da RBF-Netze spezielle DCS-Netze darstellen, meist direkt auf RBF-Netze übertragen lassen.

[13] Dieses kleine Navigationsbeispiel einer umherwandernden Maus, die während eines "random walks" eine Karte von ihrer Umgebung aufbaut, geht freilich nicht auf das Hauptproblem des Kartenaufbaus bei realen Robotern ein, die fehlerhafte Odometrie.

[14] Falls nicht explizit erwähnt, wird im folgenden stets die L_2-Norm verwendet.

2.4.1 Approximationseigenschaften

RBF-Netze sind ähnlich wie Multilayer-Sigmoid-Netze universale Approximatoren. Für RBF-Netze mit Gaußschen Basisfunktionen und individuell variierenden Radien σ_i läßt sich die universelle Approximationseigenschaft direkt nach dem Satz von Stone-Weierstraß zeigen. Siehe hierzu [50] für den Fall nicht-normaliserter RBF-Netze und [47] für den Fall normalisierter RBF-Netze mit Gaußschen Basisfunktionen.

Den allgemeinsten Beweis der universellen Approximationseigenschaft im Falle nicht-normalisierter RBF-Netze führten Park und Sandberg in [95]:

Satz 2.2 (Park und Sandberg 91) *Sei S_h die Familie von Funktionen*

$$q(x) = \sum_{i=1}^{N} o_i h(\frac{x - c_i}{\sigma}),$$

mit $N \in \mathbb{N}$, $\sigma > 0$, $o_i \in \mathbb{R}$, $c_i \in \mathbb{R}^n$ und $h : \mathbb{R}^n \to \mathbb{R}$ eine integrierbar beschränkte Funktion derart, daß h fast überall stetig ist und $\int_{\mathbb{R}^n} h(x)dx \neq 0$. Dann ist die Familie S_h dicht in $L_p(\mathbb{R}^n)$ für alle $p \in [1, \infty)$.

Der Satz ist insofern von Bedeutung, als die universelle Approximationseigenschaft hier auch für den Fall konstanter Radien gezeigt wird und als Kerne K alle sinnvollen Basisfunktionen in Betracht kommen. Diese müssen nicht einmal radialsymmetrisch sein.

Für normalisierte RBF-Netze stammt die allgemeinste Aussage von M. Benaim [12]:

Satz 2.3 (Benaim 94) *Sei $S(A)$ die auf $A \subset \mathbb{R}^n$ definierte Familie von reellwertigen Funktionen, die durch*

$$q(x) = \sum_{i=1}^{N} o_i \frac{h_\sigma(\| x - c_i \|)}{\sum_{j=1}^{N} h_\sigma(\| x - c_j \|)} \quad \mathit{mit} \quad h_\sigma = \exp(-\frac{x^2}{2\sigma^2}),$$

mit $N \in \mathbb{N}$, $\sigma > 0$, $o_i \in \mathbb{R}$, $c_i \in \mathbb{R}^n$ implementiert wird. Bezeichne ferner $C(K)$ den Raum der reellwertigen stetigen Funktionen auf einer kompakten Teilmenge K des $\mathbb{R}^n$. Dann ist

1. *für eine kompakte Menge $A \subseteq \mathbb{R}^n$ $S(A)$ dicht in $C(A)$ und*
2. *für $p \geq 1$ $S(\mathbb{R}^n)$ dicht in $L_p(\mathbb{R}^n)$.*

Dieser (im Original etwas allgemeiner gefaßte) Satz zeigt auch für den Fall normalisierter RBF-Netze die universelle Approximationseigenschaft bei konstanten Radien. Leider bezieht er sich nur auf Gaußsche Basisfunktionen. Es wäre allerdings angesichts Satz 2.2 erstaunlich, wenn ähnliche Resultate nicht auch für rationale Basisfunktionen ableitbar wären.

Wie Poggio und Girosi bereits in [41] feststellten, ist die Fähigkeit zur universellen Approximation (die sie als "Weierstraß"-Eigenschaft bezeichnen) jedoch keine besondere Auszeichnung von KNN und RBF-Netzen im speziellen. Die Autoren weisen darauf hin, daß die Resultate der Approximationstheorie dafür sprechen, daß fast jede Nichtlinearität in den Knoten der verdeckten Schicht die Weierstraß-Eigenschaft zusichert. Es gilt daher, Kriterien zu finden, die spezielle KNN vor anderen Approximationsschemata auszeichnen. Für RBF-Netze ist ein solches Kriterium die Bestapproximations-Eigenschaft[15]. Während sich für RBF-Netze die Existenz und Eindeutigkeit der Bestapproximation zeigen läßt, erfüllen z.B. MLP-Netze die Bestapproximationseigenschaft nicht [41].

Für MLP-Netze mit sigmoiden Aktivierungsfunktionen ist eine herausragende Eigenschaft die von Barron gezeigte Unabhängigkeit der Konvergenzrate von der Dimensionalität des Eingaberaumes. Girosi und Anzellotti [40] postulieren, daß ähnliche Resultate auch für RBF-Netze gelten. In diese Richtung weisen auch ihre in [40] veröffentlichten Ergebnisse, die zeigen, daß für bestimmte Funktionsräume die Konvergenzrate für RBF-Netze $O(\frac{1}{\sqrt{n}})$ ist, n die Anzahl der Basisfunktionen, unabhängig von der Dimension des Eingaberaumes.

2.4.2 RBF-Netze - ein universales Berechnungsparadigma

RBF-Netze sind keine Erfindung der KNN-Forschung (auch wenn diese ihnen zum Durchbruch verholfen hat), sondern ergeben sich als Lösung einer Vielzahl unterschiedlicher Problemstellungen und Betrachtungsweisen. Einige dieser Problemstellungen, die die *Universalität* des RBF-Ansatzes unterstreichen, seien hier aufgeführt. Die Darstellung lehnt sich dabei eng an den Überblick in [15] an.

Exakte Interpolation

Eine erste Motivation zur Verwendung von RBF-Netzen ergibt sich aus dem Problem der exakten Interpolation einer Menge von mehrdimensionalen Datenpunkten $T = \{(x_i, t_i)\}$, $x_i \in \mathbb{R}^n$, $t_i \in \mathbb{R}^m$, wobei alle x_i und t_i verschieden sein müssen. Das Ziel ist, eine Funktion $g(x)$ zu finden, so daß

$$g(x_i) = t_i, \quad i = 1, \ldots, |T|.$$

Michelli ([85]) konnte zeigen, daß für eine große Klasse radialer Basisfunktionen $h(.)$ das exakte Interpolationsproblem durch eine gewichtete Summe der Basisfunktionen

$$g(x) = \sum_{(x_i,t_i)\in T} o_i h(\| x - x_i \|), \tag{2.28}$$

also durch ein RBF-Netz mit $|T|$ Knoten, gelöst wird. Die Klasse $h(.)$ umfaßt alle praktisch relevanten Basisfunktionen, insbesondere die Gaußsche Basisfunktion.

Regularisierungsansatz

Eine weitere Motivation zum Einsatz von RBF-Netzen bietet der Regularisierungsansatz von Poggio und Girosi. Sie konnten in [99] zeigen, daß die um einen Regularisierungsterm

[15] Diese bezeichnet die Eigenschaft einer Menge von Approximationsfunktionen $\mathcal{F}$, daß es für jede Funktion g aus einer Funktionenmenge Φ eine Funktion f aus $\mathcal{F}$ gibt, die minimalen Abstand zu g hat (siehe [41]).

$|Py|^2$ erweiterte quadratische Fehlerfunktion Err^2 auf einer Trainingsmenge T,

$$E = \frac{1}{2} \sum_{(x,t)\in T} (y(x) - t)^2 + \frac{\gamma}{2} \int |Py|^2 dx, \tag{2.29}$$

eindeutig durch

$$y(x) = \sum_{(x_i,t_i)\in T} o_i G(x - x_i), \tag{2.30}$$

gelöst wird, wobei G die Greensche Funktion des Operators $\hat{P}P$ ist[16] und γ der Regularisierungsparameter. Der Differentialoperator P regularisiert dabei das schlecht gestellte Interpolationsproblem derart, daß eine Funktion mit durch P bestimmten Glattheitseigenschaften (glatte Interpolation der Datenpunkte) gesucht wird, wobei aus dem Interpolationsein Approximationsproblem wird. Ist P translations- und rotationsinvariant, so ergibt sich als Greensche Funktion eine radiale Funktion $G(\parallel x - x' \parallel)$, womit (2.30) die Form der RBF-Evaluierungsfunktion (2.26) eines RBF-Netzes annimmt. Beispielsweise erhält man [15] mit dem Differentialoperator P nach

$$\int |Py|^2 dx = \sum_{l=0}^{\infty} \frac{\sigma^{2l}}{l!2^l} \int |D^l y(x)|^2 dx \quad \text{mit} \quad D^{2l} = (\nabla^2)^l \text{ und } D^{2l+1} = \nabla(\nabla^2)^l \tag{2.31}$$

als Greensche Funktion gerade die Gaußfunktion,

$$G(\parallel x - x' \parallel) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp(-\frac{\parallel x - x' \parallel)}{2\sigma^2}), \tag{2.32}$$

und damit ein RBF-Netz mit Gaußschen Basisfunktionen. Man beachte in diesem Zusammenhang, daß der Regularisierungsterm die Größe der rezeptiven Felder (und damit die Glattheit) bestimmt.

Interpolation Verrauschter Daten

Das Problem der Interpolation verrauschter Daten liefert ein weiteres Argument zur Verwendung von RBF-Netzen [135]. Hierbei wird angenommen, daß die Trainingsdaten unverrauschte Realisierungen einer Funktion $t(x)$ darstellen, die Eingaben x aber in verrauschter Form vorliegen. Als quadratische Fehlerfunktion ergibt sich

$$E = \int \int (y(x + \xi) - t(x))^2 \, \tilde{p}(\xi) \, p(x) \, d\xi \, dx \tag{2.33}$$

wobei $\tilde{p}(\xi)$ die Rauschverteilung darstellt. Die Lösung von (2.33) ergibt sich zu (vgl. [15])

$$y(z) = \frac{\int t(x) \, \tilde{p}(z - x) \, p(x) \, dx}{\int \tilde{p}(z - x) \, p(x) \, dx} \tag{2.34}$$

Im Falle einer endlichen Anzahl von Datenpunkten läßt sich (2.34) durch

$$y(z) = \frac{\sum_{(x,t)\in T} t \, p(z - x)}{\sum_{(x,t)\in T} p(z - x)}, \tag{2.35}$$

approximieren. Unter Annahme einer radialen Dichtefunktion $\tilde{p}(\xi)$ nimmt (2.35) dann wieder die Form der normalisierten RBF-Evaluierungsfunktion (2.27) an.

---

[16] Die Greensche Funktion des Operators $\hat{P}P$ ist über $\hat{P}PG(x, x') = \delta(x - x')$ definiert.

Regression mit Gaußschen Kernen

Eine Verbindung von RBF-Netzen zur Regression mit Gaußschen Kernen wurde in [117] aufgezeigt. Ausgangspunkt ist hier, daß die optimale Approximation der bedingte Erwartungswert der Sollausgabe ist (Seite 14), d.h.

$$y^{opt}(x) = < t|x > = \int t\, p(t|x)\, dt = \frac{\int t\, p(t,x)\, dt}{\int p(x,t)\, dt} \tag{2.36}$$

Wird nun die Verbundwahrscheinlichkeit $p(t,x)$ durch einen Parzen-Schätzer mit Gaußschen Fensterfunktionen geschätzt,

$$\hat{p}(x,t) = \frac{1}{|T|} \sum_{(x',t') \in T} \frac{1}{(2\pi\sigma^2)^{(n+m)/2}} \exp\left(-\frac{\| x - x' \|^2}{2\sigma^2} - \frac{\| t - t' \|^2}{2\sigma^2}\right), \tag{2.37}$$

so ergibt sich

$$y(x) = \frac{\sum_{(x',t') \in T} t' \exp(-\frac{\|x-x'\|^2}{2\sigma^2})}{\sum_{(x',t') \in T} \exp(-\frac{\|x-x'\|^2}{2\sigma^2})} \tag{2.38}$$

und damit ein normalisiertes RBF-Netzwerk mit Gaußschen Basisfunktionen, dessen Parameter direkt den Trainingsdaten entsprechen.

Bayes-optimale Klassifikation

Eine Motivation zur Verwendung von RBF-Netzen im Zusammenhang mit Klassifikationsaufgaben lieferte u.a. [78]. Nach der Bayes-Regel gilt für die a posteriori Wahrscheinlichkeit der Klasse C_k

$$P(C_k|x) = \frac{p(x|C_k)P(C_k)}{p(x)} = \frac{p(x|C_k)P(C_k)}{\sum_{k'} p(x|C_{k'})P(C_{k'})}. \tag{2.39}$$

Werden nun die klassenbedingten Wahrscheinlichkeiten $p(x|C_k)$ über Gaußsche Mischverteilungen modelliert, d.h.

$$p(x|C_k) = \sum_{j=1}^{N} p(x|j)P(j|C_k), \tag{2.40}$$

so ergibt sich durch Einsetzen von (2.40) in (2.39) wiederum ein normalisiertes RBF-Netzwerk mit radialen Basisfunktionen:

$$P(C_k|x) = \sum_{j=1}^{N} \frac{P(j|C_k)P(C_k)}{P(j)} \frac{p(x|j)P(j)}{\sum_{i=1}^{N} p(x|i)P(i)} = \sum_{j=1}^{N} o_j \frac{h_j(\| x - c_j \|)}{\sum_{i=1}^{N} h_i(\| x - c_i \|)} \tag{2.41}$$

Verbindung zur Fuzzy-Logic

Schließlich sei noch auf die funktionale Äquivalenz von RBF-Netzen und Fuzzy-Reglern vom Sugeno-Typ hingewiesen, wie sie erstmals in [132] aufgezeigt wurde. Sie läßt sich zur Vorstrukturierung und Interpretation von RBF-Netzen nutzen, so z.B. in [60]). Eine ausführliche Darstellung der Verbindung zur Fuzzy-Logic erfolgt im Zusammenhang mit der Einführung der DCS-Netzwerke in Abschnitt 3.3.5.

Kapitel 3

Dynamische Zellstrukturen (DCS)

In diesem Kapitel wird das für die vorliegende Arbeit zentrale KNN-Modell, die *Dynamische Zellstruktur* (DCS), eingeführt. Die hierbei wesentliche Idee ist, die in Abschnitt 2.4 aufgeführten Vorzüge von RBF-Netzen mit den von Martinetz entwickelten und in Abschnitt 2.3.3 erläuterten perfekt topologieerhaltenden Karten zu verbinden, wobei letztere zur Erzielung von Nachbarschaftskooperationseffekten nach dem Vorbild der in Abschnitt 2.2 dargestellten Kohonenkarten (SOM) eingesetzt werden. Die neue Qualität von DCS-Netzen besteht darin, die topologische Struktur der Eingabedaten zu modellieren und direkt in einem konnektionistischen Approximationsverfahren zu nutzen. Die erweiterten Kohonenkarten mit ihrer starren Gitterstruktur sind zu diesem Zweck im allgemeinen ungeeignet, während das erweiterte Neuronengas wie auch RBF-Netze nur metrische Information zur Approximation nutzen. Letztere ist im Fall konvexer Eingabemannigfaltigkeiten vollkommen ausreichend, bei komplizierter gestalteten, nicht konvexen Mannigfaltigkeiten hingegen läßt die Hinzunahme von topologischer Information eine qualitative Verbesserung des Approximationsverhaltens erwarten. Darüber hinaus nutzt die DCS topologieerhaltende Karten sowohl zu einer gegenüber RBF-Netzen wesentlich beschleunigten Ausgabeberechnung als auch in einer Reihe von Lernregeln, die die Kenntnis der topologisch nächsten Nachbarn von Stützstellen voraussetzen.

Zunächst werden in Abschnitt 3.1 *Optimal topologieerhaltende Karten* (OTPMs) eingeführt, die sich als Approximation perfekt topologieerhaltender Karten interpretieren lassen. Über OTPMs werden dann in Abschnitt 3.2 die Dynamischen Zellstrukturen als konnektionistische Berechnungsstrukturen eingeführt, welche OTPMs erlernen und zur Approximation nutzen. Die Eigenschaften der DCS, und hier insbesondere Fragen der Approximationseigenschaft und der Auswertungskomplexität, werden in Abschnitt 3.3 erörtert. Die Diskussion der Lernregeln in Abschnitt 3.4 schließlich nimmt den größten Teil dieses Kapitels ein. Neben Verfahren zur Parameteradaption werden hier insbesondere auch Lernregeln zur Adaption der Modellordnung (Wachstum der Netze) und zur Adaption der topologieerhaltenden Karten diskutiert.

Dieses Kapitel bietet insbesondere eine erweiterte Zusammenfassung der über [29, 27, 22, 24] verteilten Diskussion von Eigenschaften und Lernregeln Dynamischer Zellstrukturen.

3.1 Optimal Topologieerhaltende Karten (OTPMs)

Die im folgenden zu definierenden *Optimal Topologieerhaltenden Karten (OTPMs)* stehen in enger Beziehung zu Martinetz' perfekt topologieerhaltenden Karten und damit der induzierten Delaunay Triangulation $D_M(S)$, vgl. Abschnitt 2.3.3. Sie stellen eine Approximation an perfekt topologieerhaltende Karten für den praktisch relevanten Fall dar, daß anstelle der Mannigfaltigkeit M nur eine Trainingsmenge T zur Verfügung steht. Wird dennoch Hebbsches Wettbewerbslernen zum Aufbau einer lateralen Verbindungsstruktur nach (2.24) angewendet, so entsteht eine Karte, die zwar im allgemeinen nicht perfekt topologieerhaltend, aber im Sinne von Villmanns *topographischer Funktion* [129, 128] optimal topologieerhaltend ist[1].

Definition 3.1 (Optimal Topologieerhaltende Karte) *Sei $T \subset \mathbb{R}^n$ eine Trainingsmenge von Merkmalsvektoren und $P = (G, S)$ ein parametrisierter Graph $G = (V, E)$ mit $S \subset \mathbb{R}^n$, $|S| = |V|$, in dem jedem Knoten $i \in V$ eineindeutig ein Zentrum $c_i \in S$ zugeordnet ist.*
Dann heißt G Optimal Topologieerhaltende Karte *von S für die Trainingsmenge* T*, $OTPM_T(S)$, falls*

$$(i,j) \in E \Leftrightarrow \exists x \in T \; \forall k \in V \backslash \{i,j\} : \max\{\| c_i - x \|, \| c_j - x \|\} \leq \| c_k - x \|$$

Derart definiert, kann $OTPM_T(S)$ als Approximation von $D_M(S)$ auf Grundlage der Trainingsmenge T aufgefaßt werden, denn nach Satz 2.1 gilt:

Bemerkung 3.1 *Falls $T = M$ und S dicht in M ist, dann ist $OTPM_T(S) = D_M(S)$.*

Man beachte, daß analog zu Definition 2.2 die Definition der OTPM konstruktiv ist: Um $OTPM_T(S)$ zu berechnen, muß lediglich ein $x \in T$ gemäß $p_T(x)$ ausgewählt werden, der nächste und zweitnächste Zentrumsvektor c_{bmu_x} und c_{smu_x} aus S ermittelt und Knoten bmu_x mit smu_x verbunden werden. Im Falle einer unendlichen Trainingsmenge konvergiert der so konstruierte Graph gegen $OTPM_T(S)$ mit Wahrscheinlichkeit eins. Für eine endliche Trainingsmenge genügt ein Durchlauf durch die Trainingsmenge.

Im Hinblick auf eine topologieerhaltende Repräsentation der Trainingsmenge T weist $OTPM_T(S)$ zwei wichtige Eigenschaften auf: Erstens hängt der Graph nur von der intrinsischen (topologischen) Dimensionalität von T ab, ist also unabhängig von der Dimensionalität n des Eingaberaumes. Eine Einbettung von T in einen höher dimensionalen Raum ändert den Graphen nicht. Zweitens ist er invariant gegenüber Skalierung und starren Transformationen (Translation und Rotation) der Daten. $OTPM_T(S)$ stellt die Repräsentation der Trainingsmenge T auf Grundlage der Zentrenmenge S dar, die die Topologie von T optimal widerspiegelt.

[1] Grundgedanke der "topographischen Funktion" zur Messung des Grades der Topologieerhaltung eines gegebenen parametrisierten Graphens (G, S) ist es, zu S mittels der Hebb-Regel (2.24) einen zweiten Graphen G' zu konstruieren und den topologischen Defekt als Abweichung zwischen G und G' zu messen. Dieser Defekt wird per Konstruktion der topographischen Funktion gerade null wenn $G = G'$. Damit ist G' im Sinne der topographischen Funktion der optimal topologieerhaltende Graph.

Bezüglich der Zeitkomplexität des Erlernens von OTPMs sind im Falle einer endlichen Trainingsmenge offensichtlich $O(|T|)$ Präsentationen zur Generierung von $OTPM_T(S)$ nötig, im Falle einer unendlichen Trainingsmenge lassen sich die von Martinetz gezeigten Resultate für das Erlernen von $D_M(S)$ übertragen. Nach Gleichung (2.25) sind für das Erlernen eines optimal topologieerhaltenden Graphens $G = (V, E)$ mit Hebbschem Online-Lernen im Durchschnitt $O(|E| \log |E|)$ Präsentationen von Trainingsdaten notwendig, falls die Zentrenverteilung homogen ist. Für hochgradig inhomogene Zentrenverteilungen hingegen erreicht die Komplexität $O(|E|^2)$ Präsentationen. Falls die Dimensionalität der Mannigfaltigkeit kleiner als drei und die Zentrenverteilung dicht ist, so hängt dabei $|E|$ linear von $|V|$ ab. Für größere Dimensionen sind keine Resultate bekannt, es gilt jedoch stets $|E| \in O(|V|^2)$. Dies beantwortet auch die Frage nach der Platz- und Musterkomplexität von OTPMs.

Eine weitere Frage ist, wie die Anzahl der Kanten und Knoten in einer OTPM mit der intrinsischen Dimension d einer Mannigfaltigkeit skaliert, wenn diese perfekt topologieerhaltend zu repräsentieren ist. Eine untere Schranke ergibt sich als die Anzahl der Kanten und Knoten, die der einfachste d-dimensionale Polyeder aufweist. Dies ist gerade der d-dimensionale Hypertetraeder mit $d + 1$ Knoten und $\frac{d(d+1)}{2}$ Kanten, womit die Zahl der Knoten mindestens linear, die der Kanten mindestens quadratisch in d wachsen muß. Des weiteren ergibt sich im Fall einer degenerierter Zentrenverteilung (Zentren liegen auf einer Kugelschale), daß die Anzahl der von einem Knoten ausgehenden Kanten für $d > 1$ lokal immer den Maximalwert von $|V| - 1$ annehmen kann.

3.2 DCS - Kombination von RBF-Netzen mit OTPMs

Nach Einführung der optimal topologieerhaltenden Karten sollen nun die Dynamischen Zellstrukturen definiert werden. Hierzu wird zuerst die *Allgemeine Dynamische Zellstruktur (ADCS)* definiert, die eine sehr allgemeine Berechnungsstruktur darstellt und dadurch charakterisiert ist, daß sie eine OTPM erlernt, welche sie zur Ausgabeberechnung und Adaption nutzen kann. Anschließend wird die *Dynamische Zellstruktur (DCS)* als Spezialfall einer ADCS einführt, bei der die Evaluierungsfunktion durch radiale Basisfunktionen realisiert wird.

Sei im folgenden $I \subset \mathbb{R}^n$ der Eingaberaum (Definitionsbereich), $A \subset \mathbb{R}^m$ der Ausgaberaum (Wertebereich).

Definition 3.2 *(Allgemeine Dynamische Zellstruktur) Eine* Allgemeine Dynamische Zellstruktur (ADCS) *ist ein Tripel $(P, f_P, \mathcal{L})$. Hierbei ist $P = (G, O, R, S, C)$ ein parametrisierter Graph $G = (V, E)$, in dem jedem Knoten $i \in V$ eineindeutig ein* Zentrum $c_i \in S$ *und ein Tupel $(o_i, \sigma_i) \in O \times R$ zugeordnet ist und jeder Kante $(i, j) \in E$ ein Wert $C_{ij} \in C$. Dabei heißt $O \subset A$ mit $|O| = |V|$ die* Menge der Ausgabevektoren*, R mit $|R| = |V|$ die* Menge der rezeptiven Felder*, $S \subset I$ mit $|S| = |V|$ die* Menge der Zentrumsvektoren *und C mit $|C| = |E|$ die* Menge der Kantengewichtungen. *Die* Evaluierungsfunktion *$f_P : I \times P \to A$ bildet eine Eingabe aus dem Eingaberaum auf eine Ausgabe aus dem Aus-*

gaberaum ab. Die Lernfunktion $\mathcal{L}$ *bildet ein Tupel aus einem parametrisierten Graphen* P *und einer Trainingsmenge* T *auf einen parametrisierten Graphen* P' *ab.* P' *muß der Bedingung*

$$(i,j) \in E' \Rightarrow (i,j) \in E_{OTPM_T(S')} \vee [V_i^{(M)} \cap V_j^{(M)} \neq \emptyset]$$

genügen.

Eine ADCS ist damit definiert als 3-Tupel, bestehend aus einem parametrisierten Graphen P, der die Architektur des Netzwerkes beschreibt, einer Evaluierungsfunktion f_P, die auf P Bezug nehmen kann, und einer Lernfunktion $\mathcal{L}$. Die Parameter der ADCS sind durch die Parameter des parametrisierten Graphen gegeben. Dieser wird von der Lernfunktion unter Vorgabe einer Trainingsmenge adaptiert, wobei gefordert wird, daß die Verbindungsstruktur nach Ende der Adaption nur Kanten enthält, die in $OTPM_T(S')$ vorkommen oder in der induzierten Delaunay-Triangulation von S enthalten sind. Diese Definition erlaubt explizit das Einbringen von a priori Wissen bezüglich der Topologie von M, indem "falsche" Kanten in $OTPM_T(S')$ verworfen und zusätzlich Kanten, die in der induzierten Delaunay-Triangulation aber in Ermangelung reichhaltiger Trainingsdaten nicht in $OTPM_T(S')$ enthalten sind, hinzugenommen werden können. Man beachte, daß die Lernfunktion sämtliche Komponenten des parametrisierten Graphen umfaßt. Insbesondere wird neben der Adaption der Modellparameter (Zentren, Ausgaben, Radien und Kantengewichtungen) auch eine Adaption der Modellordnung (Anzahl der Knoten und Kanten) zugelassen. Dies erklärt den Namen *Dynamische* Zellstruktur.

Mit den bisher eingeführten Berechnungsmodellen steht die ADCS in folgender Beziehung, wobei $\mathcal{F}_{NN}$ die von einem bestimmten Netzwerktyp realisierbare Funktionenklasse bezeichnet:

Bemerkung 3.2 $\mathcal{F}_{ADCS} \cap \mathcal{F}_{Kohonen} \neq \emptyset$, $\mathcal{F}_{NG} \subset \mathcal{F}_{ADCS}$, $\mathcal{F}_{RBF} \subset \mathcal{F}_{ADCS}$.

Die nicht leere Schnittmenge mit der Funkionenklasse der (erweiterten) Kohonenkarten ergibt sich dabei aus deren starrer, im allgemeinen nicht optimal topologieerhaltenden Verbindungsstruktur. Zum Nachweis der Teilmengenbeziehungen ist jeweils die Berechnungsfunktion entsprechend zu wählen.

Während herkömmliche RBF-Netze keine OTPM lernen und zur Ausgabeberechnung nutzen, ist dies der Grundgedanke der im folgenden zu definierenden Klasse der RBF-basierten Dynamischen Zellstrukturen, die kurz als *Dynamische Zellstruktur* (DCS) bezeichnet werden. Sie kombinieren RBF-Netze mit OTPMs und bilden den Kern der vorliegenden Arbeit. Eine Illustration findet sich in Abbildung 3.1.

Definition 3.3 *(Dynamische Zellstruktur) Eine* Dynamische Zellstruktur *(DCS) ist eine ADCS* $(P, f_P, \mathcal{L})$, $P = (G, O, R, S, C)$, *mit* $R \subset \mathbb{R}$ *der Menge der Radien der Basisfunktionen und einer durch*

$$f_P^{rbf}(x) = \sum_{i \in \mathcal{A}_P(x)} o_i h(\| x - c_i \| / \sigma_i) \quad bzw. \tag{3.1}$$

$$f_P^{nrbf}(x) = \frac{\sum_{i \in \mathcal{A}_P(x)} o_i h(\| x - c_i \| / \sigma_i)}{\sum_{j \in \mathcal{A}_P(x)} h(\| x - c_j \| / \sigma_j)} \tag{3.2}$$

gegebenen Evaluierungsfunktion f_P. Dabei ist $c_i \in S$, $\sigma_i \in R$ und $\mathcal{A}_P(x) \subseteq V$ eine von x und P abhängige Menge aktiver Knoten*. Die Aktivierungsfunktion $h : \mathbb{R}^+ \rightarrow [0,1]$ realisiert zusammen mit der Norm $\| \, . \, \|$ die radialen Basisfunktionen*[2]*. Im Falle von $f_P = f_P^{nrbf}$ wird von einer* Normalisierten DCS *gesprochen.*

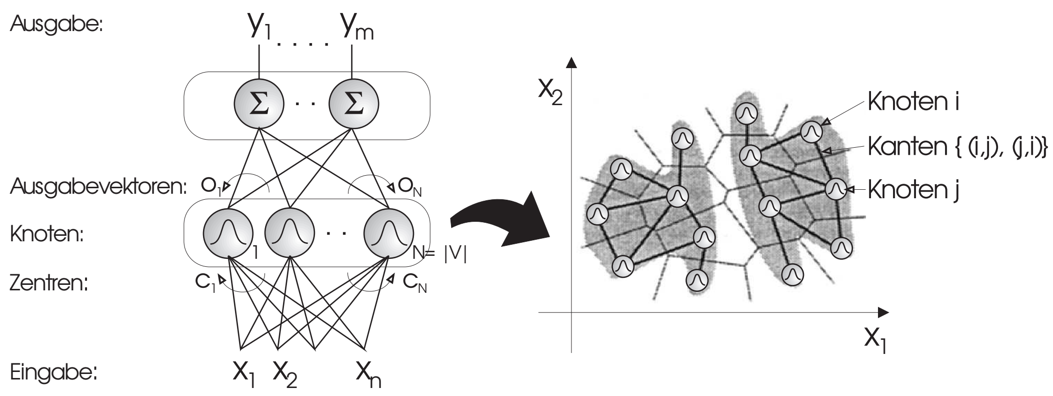


Abbildung 3.1: Dynamische Zellstrukturen kombinieren RBF-Netze und OTPMs.

Für den Fall $\mathcal{A}_P(x) = V$ ergeben sich gerade die Evaluierungsfunktionen für RBF-Netze (2.26) und (2.27). Daher gilt offensichtlich

Bemerkung 3.3 $\mathcal{F}_{RBF} \subset \mathcal{F}_{DCS}$.

Hinsichtlich der Evaluierungsfunktion ist der einzige Unterschied zwischen RBF- und DCS-Netzen, daß in DCS-Netzen nur eine von der topologischen Karte abhängige Teilmenge von RBF-Knoten an der Approximation beteiligt ist. Für den Rest dieser Arbeit werden Teilmengen betrachtet, die sich aus den bzgl. der topologischen Karte nächsten Nachbarn der bmu_x zusammensetzen, d.h.

$$\mathcal{A}_P(x) = Nh^{(i)}(bmu_x), \ i \in \mathbb{N}_0 \quad \text{mit} \tag{3.3}$$

$$Nh^{(i)}(bmu_x) = \{ j \in V \,|\, \exists \, \text{Pfad}(bmu_x - j) \subseteq E : |\text{Pfad}(bmu_x - j)| \leq i \}. \tag{3.4}$$

Der topologische Abstand zweier Knoten ist hierbei als minimale Pfadlänge zwischen zwei Knoten bzgl. der OTPM definiert. Diese Menge kann in $O(|E|)$ Zeit berechnet werden.

3.3 Eigenschaften von DCS-Netzwerken

Es sollen nun die Eigenschaften der im vorhergehenden Abschnitt eingeführten (RBF-basierten) Dynamischen Zellstrukturen untersucht werden. Zunächst wird dabei in 3.3.1

[2]Wenn nicht explizit erwähnt, wird im folgenden stets die L_2-Norm verwendet.

die universale Approximationseigenschaft der DCS nachgewiesen. Anschließend wird gezeigt, daß DCS-Netze die Berechnungskomplexität von tabularen Funktionsapproximatoren (Abschnitt 3.3.2) mit der Approximationsgüte von RBF-Netzen kombinieren (Abschnitt 3.3.3). Hierbei wird als weiterer interessanter Aspekt der DCS die regularisierende Wirkung beschränkter Nachbarschaftsfunktionen hervortreten, die einen positiven Einfluß auf die Generalisierungsfähigkeit und Rauschsensitivität des Netzes hat. Des weiteren wird in Abschnitt 3.3.4 gezeigt, daß DCS-Netze eine effektive Projektion auf die Eingabemannigfaltigkeit berechnen. Schließlich wird in Abschnitt 3.3.5 auf die Verbindung von DCS-Netzen zur Fuzzy-Logic hingewiesen und in Abschnitt 3.3.6 auf ihre Wirkungsweise als Linerisierer nichtlinearer Probleme.

3.3.1 Universale Approximationseigenschaft

Im folgenden wird die universale Approximationseigenschaft von DCS-Netzen nachgewiesen, indem sie für alle möglichen Nachbarschaftsfunktionen $Nh^{(i)}(bmu_x)$ gezeigt wird. Es werden drei Fälle unterschieden: Table-Lookup-Verfahren für $i = 0$, Approximation über alle Knoten ($i = \infty$) und beschränkte topologische Nachbarschaften ($0 \leq i \leq \infty$).

Table-Lookup-Verfahren ($i = 0$) Für $i = 0$ ist $Nh^{(i)}(bmu_x) = bmu_x$. Im Falle des normalisierten Approximationsschemas wird die Ausgabe gerade zu $y(x) = o_{bmu_x}$ und damit die DCS als Lookup-Table "mißbraucht". Für den Fall der nicht-normalisierten Evaluierungsfunktion gilt diese Aussage natürlich auch, denn mit $\sigma \to \infty$ läßt sich das Table-Lookup-Verfahren beliebig genau approximieren. Da eine auf einer kompakten Teilmenge des $\mathbb{R}^n$ definierte stetige Funktion f von einem Table-Lookup-Verfahren beliebig gut approximiert werden kann [51], gilt:

Lemma 3.1 *Sei $C(K)$ der Raum der stetigen Funktionen auf einer kompakten Teilmenge $K \subseteq \mathbb{R}^n$. Dann sind DCS-Netze mit Nachbarschaftsfunktion $Nh^{(0)}(bmu_x) = bmu_x$ dicht in $C(K)$.*

Beteiligung aller Knoten ($i = \infty$) Bei Beteiligung aller Knoten an der Approximation ergeben sich gerade die RBF-Netze[3]. Von diesen ist nach Abschnitt 2.4.1 selbst für den Fall gleicher Radien σ die universelle Approximationseigenschaft gezeigt. Es gilt also

Lemma 3.2 *Für $p \geq 1$ sind DCS-Netze mit Nachbarschaftsfunktion $Nh^{(\infty)}(bmu_x)$ dicht in $L_p(\mathbb{R}^n)$.*

Beschränkte Topologische Nachbarschaft ($0 \leq i \leq \infty$) Der Fall der beschränkten topologischen Nachbarschaftsfunktion $Nh^{(i)}(bmu_x)$ mit $0 \leq i \leq \infty$ ist der eigentlich interessante. Bezüglich der Approximationseigenschaft stetiger Funktionen auf kompakten Teilmengen des $\mathbb{R}^n$ gilt auch hier

[3] Voraussetzung ist hier, daß die topologische Karte zusammenhängend ist

Lemma 3.3 *Sei $C(K)$ der Raum der stetigen Funktionen auf einer kompakten Teilmenge $K \subseteq \mathbb{R}^n$. Dann sind DCS-Netze mit Nachbarschaftsfunktion $Nh^{(i)}(bmu_x)$ mit $0 \leq i \leq \infty$ dicht in $C(K)$.*

Dies läßt sich durch Rückführung auf den Fall der tabularen Funktionsrepräsentation leicht zeigen: Zu einer gegebenen Approximationsgüte ϵ konstruiere man zuerst die Lookup-Table für $\epsilon/2$ und ersetze die Lookup-Table-Einträge dann durch Nachbarschaftsstrukturen mit Pfadlänge $\geq i$ und gleicher Ausgabe wie der ursprüngliche Eintrag. Im normalisierten Fall wird dann für $\sigma \to 0$ die Lookup-Table beliebig genau approximiert. Im Falle nicht-normaliserter RBF-Netze läßt sich durch eine Verkleinerung der Radien in Verbindung mit einer Erhöhung der Knotenanzahl die Lookup-Table ebenfalls beliebig genau approximieren. Man beachte, daß für beschränkte Nachbarschaften die von einem DCS-Netz realisierte Funktion nicht mehr stetig ist (im Gegensatz zu RBF-Netzen) und daher insbesondere der Satz von Stone-Weierstraß zum Beweis der Approximationseigenschaften nicht mehr angewendet werden kann.

3.3.2 Berechnungskomplexität

Betrachtet man die Anzahl der Berechnungsschritte in Abhängigkeit von der Knotenanzahl $|V|$, die zur Auswertung der Evaluierungsfunktion für ein DCS-Netz nötig ist, so ergibt sich zunächst zur Bestimmung der bmu_x (Lösung des Nächsten-Nachbar-Problemes) ein linearer serieller Zeitaufwand, bzw. bei einer parallelen Implementation mit $O(|V|)$ Prozessoren ein Zeitaufwand von $O(logN)$.

Auch wenn der serielle Berechnungsaufwand sich damit asymptotisch nicht vom Berechnungsaufwand für RBF-Netze unterscheidet, ist doch gerade bei Verwendung kleiner Nachbarschaftsfunktionen, insbesondere der Einer-Nachbarschaft, eine signifikante Beschleunigung zu verzeichnen. Dieser rührt daher, daß $Nh^{(1)}(j)$ für $j \in V$ ähnlich einem Table-Lookupverfahren typischerweise nur einen Bruchteil der Knoten in V umfaßt. Nur für diese ist die Aktivierungsfunktion zu berechnen und sind ihre m-dimensionalen Ausgabevektoren gewichtet aufzuaddieren bzw. zu adaptieren. Für planare OTPMs ist die durchschnittliche Anzahl der direkten Nachbarn sogar konstant, so daß nur O(1) anstelle von O(N) Knoten ausgewertet werden müssen. Die Beschleunigung ist freilich durch die Vorleistung zum Aufbau und Speichern der OTPM (siehe Abschnitt 3.1) erkauft.

Das Nächste-Nachbar-Problem läßt sich durch Aufbau entsprechender Datenstrukturen (z.B. k-d Bäume oder Delaunay-Triangulation) erheblich beschleunigen. Es ergibt sich dann eine serielle Zeitkomplexität von $O(log|V|)$ zur Bestimmung der bmu_x und unter Ausnutzung der Delaunay-Triangulation bei einem planaren Graphen sogar eine parallele Zeitkomplexität von O(1). Nachteil der genannten Datenstrukturen ist ihre fehlende Dynamik, d.h. sie gehen von einer statischen Zentrenanordnung aus, für die dann in einer Vorbearbeitungsphase die Datenstruktur aufgebaut wird.

3.3.3 Regularisierung durch Nachbarschaftskooperation

DCS-Netze mit beschränkter Nachbarschaftsfunktion sind ein effizientes Approximationsverfahren, das immer dann ein qualitativ ähnliches Approximationsverhalten wie ein ansonsten identisches RBF-Netz (unbeschränkte Nachbarschaft) aufweist, wenn die aktive Knotenmenge diejenigen Knoten umfaßt, die im RBF-Netz eine wesentlich von null verschiedene Aktivierung aufweisen. Für ein DCS-Netz mit Einer-Nachbarschaft ist hierfür die *Orthonormalitätsbedingung*

$$h\left(\frac{\| c_i - c_j \|}{\sigma_j}\right) \approx \delta_{ij} \tag{3.5}$$

hinreichend. Dies soll anhand eines kleinen Beispiels demonstriert werden. Es werden hierzu die Knoten normalisierter DCS- und RBF-Netze mit Gaußschen Aktivierungsfunktionen äquidistant über eine U-förmige Eingabemannigfaltigkeit verteilt und durch Variation der Radien der Basisfunktionen ihr Überlapp gesteuert. Die Ausgabevektoren der Netze werden mittels Singulärwertzerlegung (siehe Abschnitt 3.4.1) bestimmt, wobei als Trainingsmenge 1000 Paare (x, ρ) dienen, die als Eingabe die kartesischen Koordinaten x eines Punktes auf dem "U" und als Sollwert die Weglänge ρ (innerhalb des "U") bis zu diesem Punkt repräsentieren. Abbildung 3.2, links, illustriert die U-förmige Eingabemannigfaltigkeit und die äquidistante Anordnung der Zentren eines Netzes mit 15 Knoten. Der Abstand der Schenkel ist d, die Länge der Schenkel $L_S = 5d$ und die Gesamtlänge $L_{tot} = 10d + \pi\frac{d}{2}$. Sei nun $\Delta^2 = (c_i - c_{i+1})^2 \approx (\frac{L_{tot}}{N-1})^2$ der quadratische Abstand benachbarter Zentren, so

Abbildung 3.2: U-förmige Eingabemannigfaltigkeit. Links: Unverrauschte Trainingsdaten und Lage der Zentren für ein Netz mit 15 Knoten. Rechts: Verrauschte Testdaten.

werden drei unterschiedliche Überlappungsfälle (klein, optimal[4], groß) durch folgende Wahl der Radien σ erzeugt:

$$\sigma^2_{small} = \frac{\Delta^2}{30}, \qquad \sigma^2_{opt} = \frac{\Delta^2}{3} \quad \text{und} \quad \sigma^2_{big} = \frac{\Delta^2}{0.3} \tag{3.6}$$

Die Abbildungen 3.3 a)-c) zeigen das Approximationsverhalten (mittlerer quadratischer Fehler, *mse*) für Netzgrößen von 1 bis 50 Knoten. Untersucht wird jeweils ein RBF-Netz (rbf), ein DCS-Netz mit einer OTPM (otpm), ein DCS-Netz mit vorinitialisierter topologischer Karte (exact_topo) und eine tabulare Funktionsrepräsentation (bmu) als Referenzkurve. Für σ_{small} und σ_{opt} (Abb. 3.3 a und b)) weist das normalisierte DCS-Netz mit

[4]Eine Überlappung wird hier als optimal bezeichnet, wenn für alle $x \in [c_i, c_j]$ die *Interpolationsbedingung* $h(\frac{\|(x-c_i)\|}{\sigma}) + h(\frac{\|(x-c_j)\|}{\sigma}) \approx 1$ mit $h(z) = \exp(-\frac{z^2}{\sigma^2})$ gilt.

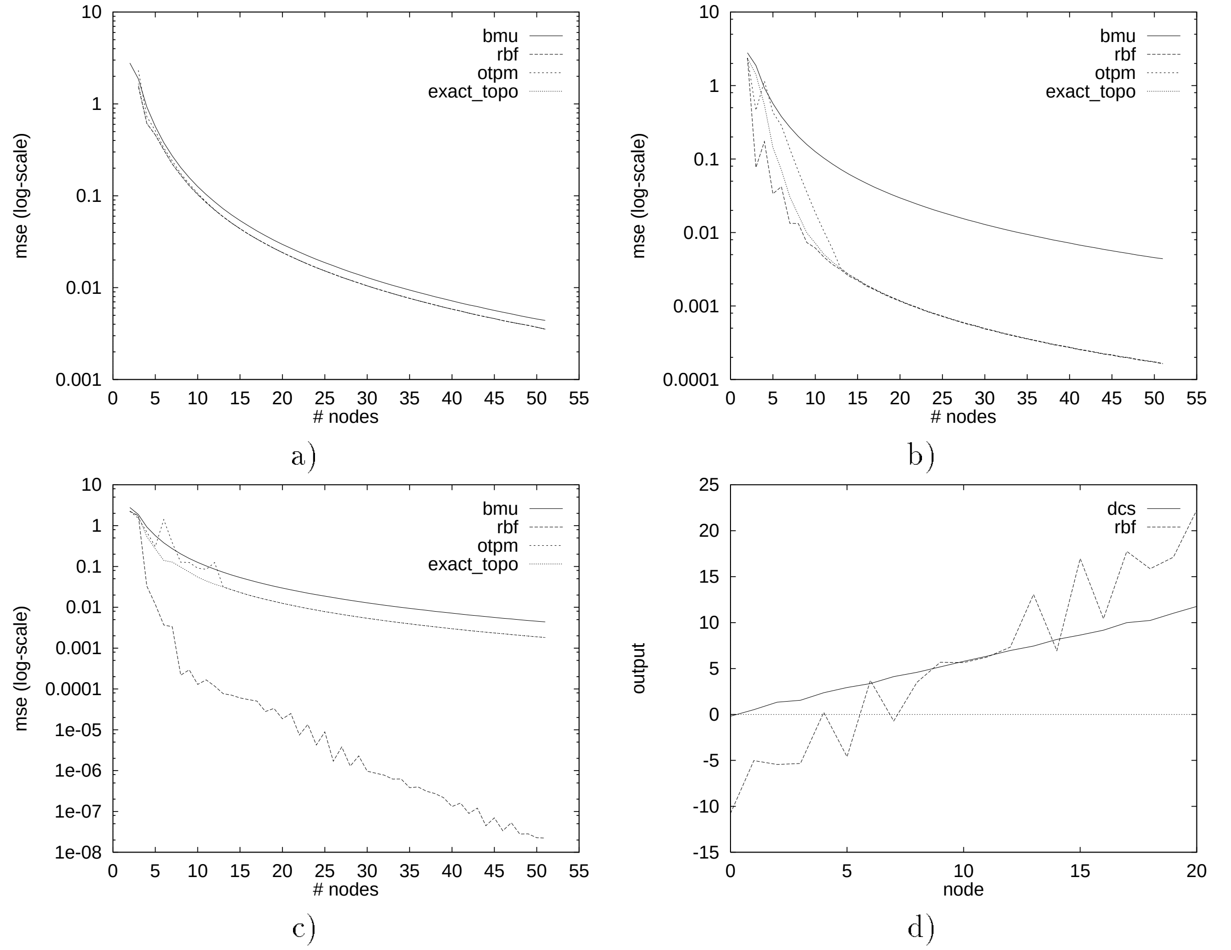


Abbildung 3.3: Approximationsverhalten von DCS-Netzen mit unterschiedlichem Überlapp: a) kleiner Überlapp; b) optimaler Überlapp; c) großer Überlapp; d) Ausgabe in topologischer Reihenfolge

Einer-Nachbarschaft wie erwartet (Gültigkeit von (3.5)) ein qualitativ ähnliches Approximationsverhalten wie das normalisierte RBF-Netz auf. Für sehr kleine Radien σ_{small} (Abb. a) imitieren die Netze dabei ein Table-Lookup-Verfahren. Für große Radien σ_{big} (Abb. c) und damit einen großen Überlapp verletzt das DCS-Netz Bedingung (3.5) und zeigt ein im Vergleich zum Fall optimaler Radien (Abb. b) verschlechtertes Approximationsverhalten, das aber immer noch besser als ein Table-Lookup-Verfahren ist. Dies ist aufgrund der verschlechterten Interpolation zwischen benachbarten Knoten nicht erstaunlich. Das RBF-Netz hingegen zeigt eine signifikante Verbesserung des Approximationsfehlers. Die Ursache für dieses unterschiedliche Verhalten bei großem Überlapp der Basisfunktionen wird in Abbildung 3.3 d) deutlich. Hier ist die Ausgabe o_i über der topologischen Reihenfolge i der Knoten im trainierten Zustand für großen Überlapp (σ_{big}) aufgetragen. Beim RBF-Netz weisen benachbarte Knoten sehr unterschiedliche Ausgaben auf, die darüber hinaus den Wertebereich der Abbildung verlassen. Dies zeigt, daß mit zunehmendem Überlapp der Basisfunktionen die RBF-Netze ihre Lokalitätseigenschaft verlieren, es wird eine globale Repräsentation der Daten erzeugt. Die RBF-Netze verlieren ihre Interpretation als erweiterte Table-Lookup-Verfahren und damit auch ihre Verbindung zur Fuzzy-Logik, erreichen aber eine hohe Approximationsgüte. Das DCS-Netz hingegen zeigt den erwarteten linearen Anstieg der Ausgaben o_i mit dem Knotenindex. Hier wirkt die beschränkte Nachbarschaftsfunktion in dem Sinne regularisierend, daß benachbarte Knoten ähnliche Ausgaben aufweisen, zwischen denen lokal interpoliert wird. Die Interpretation als erweitertes Table-Lookup-Verfahren bleibt um den Preis einer im Vergleich zu σ_{opt} schlechteren Approximationsgüte erhalten.

Neben der regularisierenden Wirkung beschränkter Nachbarschaftsfunktionen bleibt damit festzuhalten, daß die Wahl der Radien der Basisfunktionen entscheidenden Einfluß auf die Approximationsgüte hat. Ein Verfahren zur automatischen Anpassung der Radien unter Ausnutzung der topologischen Karte wird in Abschnitt 3.4.4 diskutiert.

Die Beobachtung, daß mit steigendem Überlapp der Approximationsfehler auf den Trainingsdaten abnimmt, ist in Einklang mit den in [46] präsentierten Ergebnissen. Dort wird auch gezeigt, daß in RBF-Netzen die Abnahme des Approximationsfehlers mit steigendem Überlapp mit einer dramatischen (exponentiellen) Verschlechterung der Kondition der *Designmatrix*[5] A,

$$A^T = [a(x_1), \ldots, a(x_k)],\ k = |T|$$

mit

$$a_i(x) = \begin{cases} h(\| x - c_i \| / \sigma_i) & : \text{ rbf} \\ \frac{h(\|x-c_i\|/\sigma_i)}{\sum_{j \in \mathcal{A}_P(x)} h(\|x-c_j\|/\sigma_j)} & : \text{ nrbf} \end{cases}, \tag{3.7}$$

einhergeht (siehe auch Abschnitt 3.4.1). Die Verschlechterung der Kondition von A wirkt sich neben einer Verschlechterung der numerischen Stabilität insbesondere in einer Verlangsamung der Konvergenzgeschwindigkeit beim Einsatz von gradientenbasierten Verfahren und einer gesteigerten Sensitiviät gegenüber Störungen der optimalen Lösung (der optimalen Ausgabevektoren) aus [46]. In Abschnitt 3.3.4 wird deutlich, daß der größere Überlapp und damit die verschlechterte Kondition der Designmatrix bei RBF-Netzen auch eine gesteigerte Sensitivität gegenüber Rauschen in den Eingabevektoren bewirkt und damit eine Verschlechterung der Generalisierungsfähigkeit bedeutet. Am Beispiel der U-förmigen Eingabemannigfaltigkeit zeigt Abbildung 3.4, daß für σ_{sml} und σ_{opt} die Designmatrizen A von RBF- und DCS-Netzen mit Einer-Nachbarschaft gut konditioniert sind, während sie für einen großen Überlapp A im Falle des RBF-Netzes schlecht konditioniert sind. Für denselben Überlapp ist die Kondition des DCS-Netzes um einen Faktor 10^2 besser. Die regularisierende Wirkung der Einer-Nachbarschaft kann daher auch über eine Verbesserung der Kondition der Designmatrix erklärt werden.

[5]Tatsächlich wird in [46] die Kondition c_E der *Eingangs-Korrelationsmatrix* $A^T A$ betrachtet, während hier die Kondition c_D der Designmatrix A als Verhältnis aus größtem zu kleinstem Singulärwert von A gemessen wird. Dabei gilt $c_E = c_D^2$.

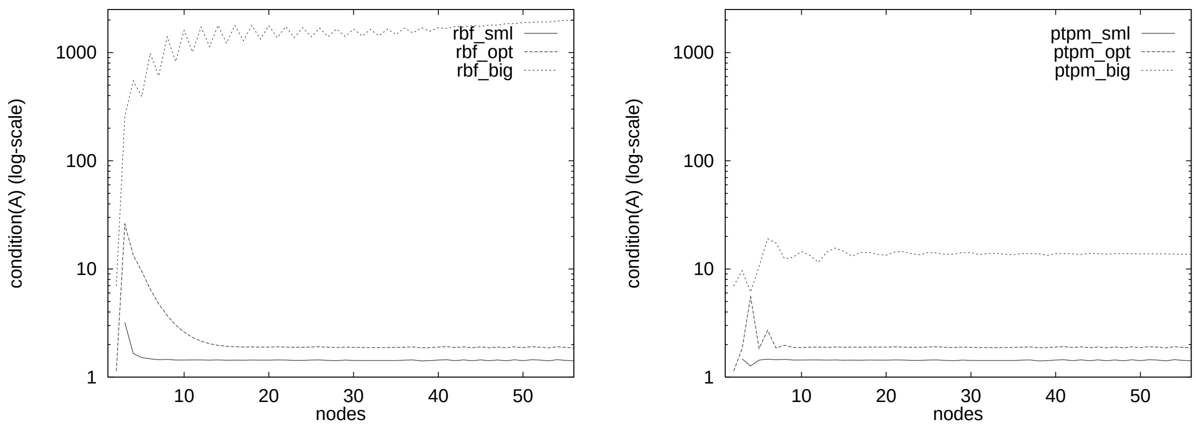


Abbildung 3.4: Kondition der Designmatrix A. Links: Für RBF-Netze mit kleinem (sml), optimalem (opt) und großem (big) Überlapp. Rechts: für DCS-Netz mit Einer-Nachbarschaft.

3.3.4 Effektive Projektion und Interpolation verrauschter Daten

Während in den meisten Anwendungen kaum ein Unterschied hinsichtlich der Approximationsgüte zwischen der Verwendung rationaler und Gaußscher Aktivierungsfunktionen festzustellen sein wird, besitzen letztere doch eine Reihe theoretischer Vorzüge. So kann, siehe Abschnitt 2.4.2, im Falle der Gaußschen Aktivierungsfunktion die Verbindung zwischen RBF-Approximation und probabilistischen Ansätzen hergestellt, wie auch eine multivariate Gaußsche Aktivierungsfunktion als Fuzzy AND verstanden werden. Letzteres geht auf die lineare Separabilität der Gaußfunktion zurück, die u.a. auch die direkte Anwendbarkeit des Satzes von Stone-Weierstraß zum Beweis der universalen Approximationsfähigkeit zur Folge hat.

In Zusammenhang mit dem normalisierten Approximationsschema besitzt die Gaußsche Aktivierungsfunktion eine weitere nützliche Eigenschaft, die als "Rauschunempfindlichkeit" bezeichnet werden soll und sich in folgendem Satz zusammenfassen läßt:

Satz 3.1 (Rauschunempfindlichkeit) *Sei $y(x) \in \mathbb{R}^m$ die Ausgabe eines normalisierten DCS-Netzes mit der aktiven Knotenmenge $\mathcal{A}_P(x)$ und Gaußschen Aktivierungsfunktionen mit gleichen Weiten σ zur Eingabe $x \in M \subseteq \mathbb{R}^n$. Sei ferner η "orthogonales Rauschen" mit $<\eta, (x - c_i)>= 0$ f.a. $c_i \in S$. Dann ist das DCS-Netz unempfindlich gegenüber dem orthogonalen Rauschen, d.h.*

$$y(x) = y(x + \eta)$$

Beweis

$$y(x+\eta) = \frac{\sum_{i\in\mathcal{A}_P(x)} o_i \, \exp(\frac{(x+\eta-c_i)^2}{\sigma})}{\sum_{j\in\mathcal{A}_P(x)} \exp(\frac{(x+\eta-c_j)^2}{\sigma})} = \frac{\sum_{i\in\mathcal{A}_P(x)} o_i \, \exp(\frac{\eta^2}{\sigma}) \exp(\frac{(x-c_i)^2}{\sigma})}{\sum_{j\in\mathcal{A}_P(x)} \exp(\frac{\eta^2}{\sigma}) \exp(\frac{(x-c_j)^2}{\sigma})} = y(x)$$

q.e.d.

Die Rauschunempfindlichkeit besagt, daß eine zu dem durch die Differenzvektoren $(x - c_i)$ aufgespannten linearen Unterraum U orthogonale Rauschkomponente in der Eingabe die Approximation nicht beeinflußt. Es findet damit eine Projektion der Eingabe auf U statt, ohne daß diese Projektion explizit berechnet werden muß. Approximiert U lokal die Mannigfaltigkeit M, wie das bei Verwendung der Einer-Nachbarschaft und im Verhältnis zur Krümmung von M ausreichend vielen Zentren $c_i \in M$ in guter Näherung immer erreichbar ist, so findet effektiv eine Projektion auf M statt.

Die Unempfindlichkeit eines DCS-Netzes gegenüber orthogonalem Rauschen in den Eingabedaten verdeutlicht folgendes Experiment, bei dem DCS-Netze mit Einer-Nachbarschaft und stark überlappenden Basisfunktionen auf unverrauschten Daten trainiert und auf verrauschten Daten getestet werden. Das Training der Netze geschieht dabei analog zu dem Experiment in Abschnitt 3.3.3, diesmal allerdings mit konstantem, nicht dem Abstand benachbarter Zentren angepaßtem Radius von $\sigma^2 = \frac{1}{0.3}$. Die Testdaten wurden analog den Trainingsdaten (Abbildung 3.2, links) generiert, jedoch durch orthogonales, uniform verteiltes Rauschen mit einer Amplitude von $\pm 0.3d$ verrauscht (Abbildung 3.2, rechts).

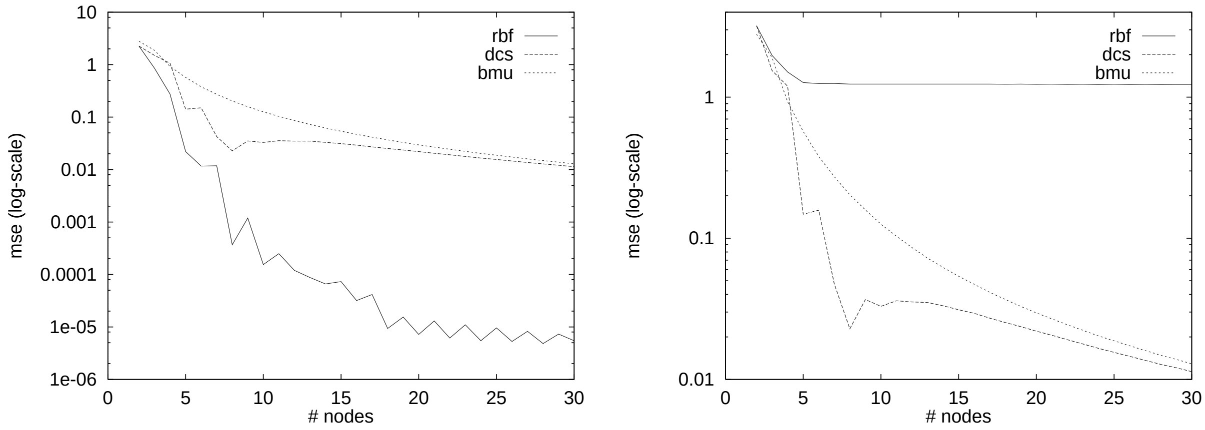


Abbildung 3.5: Approximationsverhalten von RBF- und DCS-Netzen mit Einer-Nachbarschaft. Links: Trainingsfehler auf unverrauschten Trainingsdaten. Rechts: Testfehler auf orthogonal verrauschten Testdaten.

Wie der Verlauf der Fehlerkurven für die Trainings- und Testdaten in Abbildung 3.5 zeigt, ist das DCS-Netz fast völlig unempfindlich für das Rauschen auf den Testdaten, nicht dagegen ein bis auf die unbeschränkte Nachbarschaftsfunktion identisches RBF-Netz. Hier zeigt sich wie bei dem Versuch in Abschnitt 3.3.3, daß die unverrauschten Trainingsdaten aufgrund der überlappenden Basisfunktionen mit einer sehr hohen Genauigkeit approximiert werden können, der Fehler auf den verrauschten Testdaten aber um einen Faktor 10^5 höher liegt. Dies liegt daran, daß beim RBF-Netz zur Approximation von Daten auf dem einen Schenkel des "U" auch die Knoten auf dem gegenüberliegenden Schenkel beitragen, und zwar teilweise stärker als bezüglich des "U" nähere Knoten auf demselben Schenkel.

Es können so beim RBF-Netz Stützstellen mit kleinem *Inter-Mannigfaltigkeits-Abstand* (kleinem euklidischen Abstand) aber großem Intra-Mannigfaltigkeits-Abstand[6] die Approximation negativ beeinflussen. Idealerweise sollte aber die Aktivierung von Knoten nur vom Intra-Mannigfaltigkeits-Abstand abhängen. Hier bilden DCS-Netze mit beschränkter topologischer Nachbarschaft einen Kompromiß, indem sie nur die bzgl. der OTPM nächsten Nachbarn der bmu_x in die Approximation einbeziehen. Liegen diese in Relation zur lokalen Krümmung der Mannigfaltigkeit hinreichend dicht, so ist der euklidische Abstand dieser Knoten zum Stimulus x eine gute Näherung des Intra-Mannigfaltigkeits-Abstandes. Knoten mit großem Intra- und damit potentiell kleinem Inter-Mannigfaltigkeits-Abstand werden von der Approximation ausgeschlossen. DCS-Netze mit beschränkter Nachbarschaftsfunktion beruhen dabei auf einer *doppelten Nichtlinearität*: Zusätzlich zur Verwendung nichtlinearer Basisfunktionen ist die (nichtlineare) Bestimmung der bmu_x Voraussetzung zur Bestimmung der topologischen Nachbarn.

3.3.5 Verbindung zur Fuzzy-Logic

Von den RBF-Netzen erben DCS-Netze die konzeptionelle Nähe zur Fuzzy-Logic. In diesem Abschnitt wird insbesondere auf die strukturelle Identität zwischen Sugeno Fuzzy-Reglern und DCS-Netzen eingegangen. Der Vorteil dieser Analogie ist, daß sich DCS-Netze durch Vorwissen in Form von Fuzzy-Regeln *vorinitialisieren* lassen und daß sich das Trainingsergebnis als eine Menge von Fuzzy *If-Then*-Regeln *interpretieren* läßt. Lernalgorithmen für DCS-Netze (siehe 3.4) lassen sich so direkt als Lernregeln für Fuzzy-Regler verstehen, während umgekehrt adaptive Fuzzy-Regler Anregungen für Trainingsverfahren von DCS-Netzen liefern können. Voraussetzung für eine Interpretation als Fuzzy-Regler ist, daß zur Ausgabeberechnung zwischen den Ausgabewerten benachbarter Knoten interpoliert wird, d.h. eine lokale Funktionsrepräsentation vorliegt. Wie in Abschnitt 3.3.3 aufgezeigt, ist dies bei RBF-Netzen mit überlappenden Basisfunktionen nicht der Fall.

Ein Sugeno Fuzzy-Regler, vgl. [94], berechnet eine Funktion $y : \mathbb{R}^n \to \mathbb{R}^m$ und beruht auf einer Menge von N linguistischen Regeln der Form

$$R_r : \textbf{if} \quad antecedent_r(x) \quad \textbf{then} \quad f_r(x).$$

Der Wert der Nachbedingung der r.ten Regel wird als der Wert der mit dieser Regel assoziierten Ausgabefunktion $f_r : \mathbb{R}^n \to \mathbb{R}^m$ berechnet. Er trägt zu der Ausgabe $y(x)$ des Fuzzy-Reglers, gewichtet mit dem Erfüllungsgrad $\tau_r(x) = antecedent_r(x)$, $0 \leq \tau_r(x) \leq 1$, bei, und zwar gemäß der *Defuzzyfizierungsformel*

$$y(x) = \frac{\sum_{r=1}^{N} \tau_r(x) f_r(x)}{\sum_{r=1}^{N} \tau_r(x)}. \tag{3.8}$$

Offensichtlich werden (3.8) und (3.2), die Defuzzyfizierungsformel und die Evaluierungsfunktion für normalisierte RBF-Netze, identisch, falls

[6]In einer differenzierbaren Mannigfaltigkeit $M \subseteq \mathbb{R}^n$ korrespondiert jeder Weg w zwischen zwei Punkten $x_1, x_2 \in M$ mit einer differenzierbaren Ortsfunktion r_w und einem Intervall $I_w = [u_1, u_2]$ mit $r_w : I \to M$, $r_w(u_1) = x_1$ und $r_w(u_2) = x_2$. Der Intra-Mannigfaltigkeits-Abstand zwischen x_1 und x_2 läßt sich dann als Infimum über alle Weglängen, d.h. $L(x_1, x_2) = \inf_w \int_{I_w} |\dot{r}_w| du$, definieren.

- die Funktionen in den Nachbedingungen auf konstante Funktionen beschränkt werden, d.h. $f_r(x) = const_r$
- die Vorbedingungen auf Fuzzy-Propositionen der Form

$$antecedent_r : \text{“}X \text{ is } T_X^{i_r}\text{”}$$

  festgelegt werden, wobei $T(X) = \{T_X^1, \ldots, T_X^{m_X}\}$ die Menge der linguistischen Terme der linguistischen Variablen X mit den Zugehörigkeitsfunktionen $M^{i_X} : \mathbb{R}^n \rightarrow [0,1]$ ist und
- die Zugehörigkeitsfunktionen M^{i_X} auf radiale Basisfunktionen festgelegt werden.

DCS-Netze mit beschränkter Nachbarschaftsfunktion $Nh^{(i)}(bmu_x)$ lassen sich damit als Sugeno Fuzzy-Regler auffassen, bei denen nur eine Teilmenge von Regeln zum Fuzzy-Inferenzprozeß beiträgt. Interessant sind hier der Winner-take-all-Fall für $i = 0$, der nur die bezüglich der euklidischen Norm nächstliegende Regel selektiert und damit ein Table-Lookup-Verfahren realisiert, sowie der Fall $i = 1$, bei dem zusätzlich zur nächstliegenden Regel ihre direkten topologischen Nachbarn in die Approximation einfließen. Dies kann als Auswahl der “wichtigsten” Regeln interpretiert werden, was wie im Falle der DCS-Netze zu einer erheblichen Beschleunigung des Fuzzy-Inferenzprozesses führt.

Es sei noch angemerkt, daß sich bei einer Implementierung der Zugehörigkeitsfunktionen M^{i_X} als multivariate Gaußfunktionen und einer Implementierung der Fuzzy-Konjunktion als algebraisches Produkt die Vorbedingungen alternativ als Konjunktion von n univariaten Propositionen darstellen lassen:

$$antecedent_r : \text{“}X_1 \text{ is } T_{X_1}^{i_{1,r}}\text{”} and \ldots and \text{“}X_1 \text{ is } T_{X_n}^{i_{n,r}}\text{”}.$$

Hierbei bezeichnen die X_i eindimensionale linguistische Variablen mit univariaten Gaußfunktionen $M^{i_{j,r}}(x_i)$ als Zugehörigkeitsfunktionen, die gerade die Randverteilungen von $M^{i_r}(x)$ sind.

3.3.6 Linearisierung Nichtlinearer Probleme

Einen weiteren Einblick in die Wirkungsweise von DCS-Netzen liefert ihre Interpretation als Linearisierer nichtlinearer Probleme. Offensichtlich läßt sich die Evaluierungsfunktion eines DCS-Netzes darstellen als

$$y(x) = Wa(x) \tag{3.9}$$

mit $W = [o_1, \ldots, o_N] \in \mathbb{R}^{m \times N}$ und $a(x) \in \mathbb{R}^N$ gemäß (3.7). Es erfolgt zuerst eine nichtlineare Transformation (3.7) der Eingabe $x \in \mathbb{R}^n$ in einen Aktivierungsvektor $a(x) \in \mathbb{R}^N$ und damit ein Wechsel der Repräsentation vom Eingaberaum $\mathbb{R}^n$ in den Aktivierungsraum $\mathbb{R}^N$. Auf letzterem wird das Approximationsproblem durch die lineare Abbildung (3.9) zu lösen versucht. Ein Beispiel, wie durch Linearisierung mit radialen Basisfunktionen nicht linear separierbare Funktionen linear separabel im Aktivierungsraum werden, liefert das 2-Spiralen-Problem in Abschnitt 4.2.2.

3.4 Lernverfahren

Nachdem im letzten Abschnitt die generellen Approximationseigenschaften von DCS-Netzen erörtert wurden, sollen in den Abschnitten 3.4.1 - 3.4.4 Verfahren zur Parameteradaption beschrieben werden. Die Lernverfahren werden dabei in überwachte und unüberwachte Lernverfahren gegliedert (vgl. Abschnitt 2.1.1). Für endliche Trainingsmengen werden Batch-Lernverfahren angegeben, für potentiell unendliche, nur durch (im allgemeinen unbekannte) Dichteverteilungen charakterisierte Trainingsmengen (stochastische) Online-Adaptionsverfahren auf der Basis einzelner Trainingsbeispiele.

Neben Lernverfahren zur Parameteradaption werden in Abschnitt 3.4.5 Lernverfahren zur Konstruktion von OTPMs vorgestellt. Schließlich werden in Abschnitt 3.4.6 Lernverfahren zur Adaption der Knotenzahl und damit der Modellordnung angegeben. Jede zulässige Kombination von Lernverfahren zur Parameteradaption, Adaption der lateralen Verbindungsstruktur und der Modellordnung beschreibt eine mögliche Lernfunktion $\mathcal{L}$ (vgl. Def. 3.2).

3.4.1 Adaption der Ausgabeschicht

Wie in Abschnitt 3.3.6 dargestellt, ist die Ausgabe $y(x)$ eines DCS-Netzes eine lineare Funktion des Aktivierungsvektors $a(x)$

$$y(x) = Wa(x) \tag{3.10}$$

mit $W = [o_1, \ldots, o_N] \in \mathrm{IR}^{m \times N}$ und $a(x) \in \mathrm{IR}^N$. Damit ist die Netzausgabe nach den Ausgabevektoren o_i differenzierbar, was bei einer nach den Netzausgaben differenzierbaren Fehlerfunktion $Err(T)$ den Einsatz gradientenbasierter Optimierungsverfahren, insbesondere auch stochastischen Gradientenabstiegs, zur Optimierung von W erlaubt.

Im Zusammenhang mit Batch-Lernverfahren ist insbesondere die quadratische Fehlerfunktion $Err^2(T)$ interessant, da sich das Problem der Optimierung von W so zum Problem der *linearen Regression* reduziert, für das analytische, nicht-iterative Lösungen angegeben werden können. Es ist

$$Err^2(T) = \frac{1}{2} \sum_{(x,t)\in T} (t - Wa(x))^2, \tag{3.11}$$

so daß für einen optimalen Parametervektor W

$$\nabla_W Err^2(T) = A^T\Theta - A^T AW^T = 0 \tag{3.12}$$

mit $\Theta^T = [t_1, \ldots, t_k]$, $k = |T|$, und $A^T = [a(x_1), \ldots, a(x_k)]$, gelten muß. Unter der Annahme, daß $A^T A$ nicht singulär ist, wird die *Normalgleichung* (3.12) eindeutig durch

$$W^T = A^\dagger \Theta \tag{3.13}$$

mit $A^\dagger = (A^T A)^{-1} A^T$, der *Pseudoinversen* von A, gelöst [104]. Falls $A^T A$ singulär ist, ist (3.12) nicht eindeutig lösbar. Jede Lösung ist dann eine bezüglich der Minimierung von (3.11) optimale Lösung. Eine eindeutige Lösung läßt sich durch die zusätzliche Forderung nach Minimalität von $\| W \|_2$ gewinnen [104].

In der Praxis ist von der Lösung über die Pseudoinverse aufgrund numerischer Instabilitäten abzuraten, stattdessen ist die *Singulärwertzerlegung (SVD)* von A zu bevorzugen [103]: Ist $A = U\, diag\{w_1, \ldots, w_N\}\, V^T$, $U \in \mathbb{R}^{|T| \times N}$, $V \in \mathbb{R}^{N \times N}$ die Singulärwertzerlegung der Designmatrix A, so ergibt

$$W^T = V\, diag\{\frac{1}{w_1}, \ldots, \frac{1}{w_N}\}\, U^T \Theta \tag{3.14}$$

eine bezüglich $Err^2(T)$ beste Lösung. Existieren mehrere Lösungen ($A^T A$ singulär), so liefert SVD die Lösung mit minimaler L_2-Norm $\| W \|_2$, falls vorher $\frac{1}{w_i}$ auf 0 für $w_i = 0$ gesetzt wurde. Diese ist auch die zur numerischen Weiterverarbeitung am besten geeignete Lösung (beste Kondition). Im Falle von DCS-Netzen mit beschränkter Nachbarschaftsfunktion, insbesondere der Einer-Nachbarschaft, ist A sehr dünn besetzt und ergibt bei entsprechender Anordnung der Trainingsdaten eine Quasi-Bandmatrix. Für diese Matrizen existieren besonders effiziente Verfahren zur SVD [103]. Des weiteren ist die SVD dadurch interessant, daß durch Weglassen kleiner Diagonalelemente (Nullsetzen von $\frac{1}{w_i}$ falls $w_i \approx 0$) der Fehler auf den Daten nicht wesentlich größer, aber die Anzahl der freien Parameter reduziert wird. Dadurch wird die Generalisierungsfähigkeit verbessert und die Rauschsensitivität vermindert.

Die Möglichkeit, die Ausgabevektoren bei gegebenen Basisfunktionen optimal mit SVD zu bestimmen, legt beim Batch-Lernen eine 2-Phasen-Strategie nahe, in der abwechselnd die Ausgabeschicht (analytisch mit SVD) und die Basisfunktionen (iterativ, z.B. mit Gradientenabstieg) optimiert werden.

Im Falle des Online-Lernens der Ausgabevektoren ergibt sich als stochastischer Gradientenabstieg auf $Err^2(T)$ nach (3.11)

$$W_{t+1} = W_t + \alpha_t (t - y(x)) a^T(x) \tag{3.15}$$

und mit α_t gemäß (2.9) gerade die Robbins-Monro-Prozedur zur Minimierung von $Err^2(T)$. Die Lernregel (3.15) ist auch unter dem Namen *Least Mean Square (LMS) Regel*, *Widrow-Hoff-Regel* oder *Delta-Regel* bekannt. Eine Regularisierung der Minimierungsaufgabe läßt sich durch Addition des Bestrafungsterms $\Omega = W^2$ für große Gewichte (engl. *weight decay term*) zur rechten Seite von (3.11) erreichen, was zu einer Addition von νW zur rechten Seite von (3.15) führt. Dies wirkt sich in einer Bevorzugung kleinerer Lösungsvektoren und einer Verringerung der effektiven Zahl freier Parameter um den Preis einer Verzerrung der Lösung aus, ähnlich dem Nullsetzen der $\frac{1}{w_i}$ falls $w_i \approx 0$ bei der SVD.

Die Matrixschreibweise von (3.15) sollte nicht darüber hinwegtäuschen, daß sich mit DCS-Netzen mit beschränkter Nachbarschaft gegenüber RBF-Netzen eine durchschnittliche serielle Beschleunigung von $\frac{N}{l}$ für einen Adaptionsschritt der Ausgabeschicht ergibt, wenn l die durchschnittliche Größe der aktiven Knotenmenge angibt. Dieser fällt insbesondere bei großen Ausgabedimensionen ins Gewicht.

3.4.2 Adaption lokaler Fehlervariablen zur Varianzprädiktion

Neben der Approximation der (unbekannten) Sollfunktion durch die Netzausgabe $y(x)$ ist aus zwei Gründen auch die Modellierung des Erwartungswertes des Ausgabefehlers (der

Varianz) als Funktion von x,

$$\sigma^2(x) = \int err(x,t)p(t|x)dt, \tag{3.16}$$

im trainierten Zustand des Netzes sinnvoll. Zum einen kann die Varianz als Vertrauensmaß für die Netzausgabe (Fehlerbalken) dienen, zum anderen für eine fehlergesteuerte Verfeinerung von DCS-Netzen (siehe Abschnitt 3.4.6).

Prädiktion der Varianz

Mit einer zusätzlichen *Fehlervariablen* $r_i \in \mathbb{R}$ als weiterem Parameter für jeden Knoten im DCS-Netz läßt sich die Varianz als

$$\hat{\sigma}^2(x) = r^T a(x) \tag{3.17}$$

mit $r^T = [r_1, \ldots, r_N]$ und $a(x) \in \mathbb{R}^N$ nach (3.7) als zusätzliche Ausgabe eines DCS-Netzes modellieren. Zum Training dieser Funktion wird eine zusätzliche Trainingsmenge T' durch

$$T' = \{(x, err(x,t))|(x,t) \in T\} \tag{3.18}$$

definiert, mittels derer die Fehlervariablen wie in Abschnitt 3.4.1 beschrieben durch lineare Ausgleichsrechnung (d.h. durch Minimierung der quadratischen Fehlerfunktion) adaptiert werden können. Als Online-Lernregel ergibt sich so in Analogie zu (3.15)

$$r_{t+1} = r_t + \beta_t(err(x,t) - \hat{\sigma}^2(x))a(x). \tag{3.19}$$

Prädiktion der durchschnittlichen Varianz pro Voronoizelle

Im Rahmen der fehlergesteuerten Verfeinerung von DCS-Netzen (Abschnitt 3.4.6) interessiert weniger der genaue Verlauf von $\sigma^2(x)$ als Funktion von x als vielmehr die durchschnittliche Varianz σ_i^2 in der Voronoizelle des i.ten Zentrums c_i,

$$\sigma_i^2 = \int_{V_i} \sigma^2(x)\, p(x|V_i)\, dx. \tag{3.20}$$

Diese wird durch eine Lookup-Tabelle dargestellt, die wiederum durch Fehlervariablen r_i realisiert wird. Zum Erlernen der Einträge r_i gilt es in diesem Fall, die quadratische Fehlerfunktion

$$\begin{aligned} E_i &= \int_{V_i} (\sigma^2(x) - r_i)^2\, p(x|V_i)\, dx & (3.21) \\ &= \int_{V_i} (\int err(x,t)\, p(t|x)\, dt - r_i)^2\, p(x|V_i)\, dx & (3.22) \end{aligned}$$

für alle Zentren zu minimieren. Durch Nullsetzen der Ableitung nach r_i ergibt sich aus (3.22) die (Batch-) Lösung

$$r_i = \int_{V_i} \int err(x,t)\, p(t|x)\, dt\, p(x|V_i)\, dx. \tag{3.23}$$

Als Online-Lernregel zur Adaption der r_i mittels stochastischen Gradientenabstieges auf (3.22) erhält man

$$\Delta r_i = \begin{cases} \beta(t)\,(err(x,t) - r_i): & i = bmu_x \\ 0 & : sonst \end{cases} \quad . \tag{3.24}$$

Man beachte, daß (3.23) und (3.24) unabhängig von $p(V_i)$ und damit der Wahrscheinlichkeitsdichte der Eingaben sind.

Einfluß von Rauschen auf die Varianzschätzung

Die Varianz $\sigma^2(x)$ nach (3.16) beinhaltet nicht nur den eigentlichen Approximationsfehler sondern auch das Rauschen auf den Sollausgaben, wie man leicht am Beispiel des quadratischen Fehlers sieht. Mit den bedingten Erwartungswerten

$$< t_i|x > \; = \int t_i\, p(t_i|x)\, dt_i \tag{3.25}$$

$$< t_i^2|x > \; = \int t_i^2\, p(t_i|x)\, dt_i \tag{3.26}$$

läßt sich (3.16) mit $err(x,t) = (y(x) - t)^2$ als

$$\sigma^2(x) = \sum_i (y_i(x) - < t_i|x >)^2 + \sum_i (< t_i^2|x > - < t_i|x >^2) \tag{3.27}$$

umschreiben. Der erste Term gibt dabei den Approximationsfehler an (und ist durch ein verbessertes Modell beliebig minimierbar), der zweite die Rauschvarianz der Sollwerte. Letztere stellt damit eine untere Schranke für die Varianz $\sigma^2(x)$ dar.

3.4.3 Adaption der Zentren

Nach den Lernregeln für die Ausgabevektoren $o_i \in \mathbb{R}^m$ werden nun Lernregeln zur Adaption der Zentren $c_i \in \mathbb{R}^n$ angegeben. Wiederum werden zuerst unüberwachte, dann überwachte Lernregeln betrachtet, die sich jeweils in Batch- und Online- Verfahren gliedern.

Unüberwachtes Lernen der Zentren

Obwohl unüberwachtes Training der Zentrumsvektoren nur bedingt sinnvoll ist (schließlich ist man an der Minimierung einer auf der Netzausgabe definierten Fehlerfunktion und damit an einem überwachten Training interessiert), ist es doch im Zusammenhang mit RBF-Netzen weit verbreitet. Ziel des unüberwachten Trainings ist die Erzeugung einer der Wahrscheinlichkeitsdichte der Eingaben entsprechenden Zentrenverteilung. Wie in Abschnitt 2.1.1 diskutiert, sind hierzu Vektorquantisierungsverfahren besonders geeignet. Als Lernverfahren bietet sich daher jeder Vektorquantisierungsalgorithmus an, der N Stützstellen unter Minimierung des Quantisierungsfehlers $Err^{VQ}(T)$ im Eingaberaum verteilt. Ein Beispiel ist der LBG-Algorithmus [77].

Als Online-Vektorquantisierer kommt die stochastische Variante des LBG-Algorithmus in Betracht [79] oder auch das in 2.3 diskutierte Neuronengas. Diese Verfahren nutzen freilich die bei DCS-Netzen in Form der OTPM gespeicherte topologische Struktur der Eingabemannigfaltigkeit nicht aus. Ein Vektorquantisierungsverfahren, das die topologische Verbindungsstruktur ausnutzt, ergibt sich nach Vorbild der Kohonenregel (2.12), wenn an die Stelle der Gitter-Metrik $d_L(i, bmu_x)$ die allgemeine Graph-Metrik $d_G(i, bmu_x)$ tritt, die den Abstand zweier Knoten als minimale Pfadlänge in G mißt. Man erhält

$$\Delta c_i = \epsilon(t)\,(x - c_i)\,h(\frac{d_G(i, bmu_x)}{\sigma_t}). \tag{3.28}$$

Wie die Kohonenregel nutzt (3.28) den Nachbarschaftskooperationseffekt zu einer Beschleunigung des Lernvorganges, nun allerdings mit der optimal topologieerhaltenden Verbindungsstruktur anstelle des starren Kohonengitters. Wie bei der Kohonenregel ist die Ableitung einer Energiefunktion oder des Auflösungsfaktors für (3.28) ein bislang ungelöstes Problem. Als Spezialfall ergibt sich für die Einer-Nachbarschaft die *Fritzke-Lernregel* [7]

$$\Delta c_i = \left\{ \begin{array}{ll} \epsilon_{bmu}\,(x - c_i): & i = bmu_x \\ \epsilon_{Nh}\,(x - c_i): & i \in Nh^{(1)}(bmu_x) \backslash \{bmu_x\} \\ 0 & : \; sonst \end{array} \right. , \tag{3.29}$$

auf die im Rahmen dieser Arbeit noch häufiger zurückgegriffen wird. Die Verwendung einer kleinen Nachbarschaft bietet nicht nur Berechnungsvorteile, sondern hilft auch zu verhindern, daß Zentren mit größerem Intra-Mannigfaltigkeitsabstand eine konkave Mannigfaltigkeit verlassen, indem sie durch (3.28) in Richtung der Eingabe x adaptiert werden. Für konvexe Eingabemannigfaltigkeiten hingegen gilt offensichtlich:

Bemerkung 3.4 *Liegen die Eingaben x und die initialen Zentrumsvektoren c_i in einer konvexen Teilmenge M des Eingaberaumes $\mathbb{R}^n$, so verlassen die Zentrumsvektoren M durch Anwendung von (3.28) oder (3.29) diese nicht, falls nur stets $\epsilon(t) \leq 1$.*

Überwachtes Lernen

Da die Netzausgabe von DCS-Netzen (3.1), (3.2) von den Zentrumsvektoren c_i zwar nichtlinear aber differenzierbar abhängt, können gradientenbasierte Verfahren zur nichtlinearen Optimierung der Zentrumsvektoren eingesetzt werden[8]. Dabei ist in der Praxis oft zu beobachten, daß die Zentrumsvektoren die Eingabemannigfaltigkeit verlassen, selbst wenn diese konvex ist. Um solche lokalen Minima zu verhindern, kann zur Fehlerfunktion $Err(T)$ der Vektorquantisierungsfehler Err^{VQ} nach (2.6) als Regularisierungsterm addiert werden.

---

[7] Vom Autor dieser Arbeit nach Bernd Fritzke benannt, der diese Lernregel zur Zentrenadaption seiner Wachsenden Zellstrukturen [37] nutzt.

[8] Insbesondere beim Batch-Training ist darauf zu achten, daß die Evaluierungsfunktion bei beschränkten Nachbarschaften nur stückweise stetig und differenzierbar ist und Verschiebungen der Zentren, die eine Verschiebung der Voronoizellen bewirken, eine Adaption der OTPM erforderlich machen können. Dies erschwert den Einsatz konventioneller Optimierungsverfahren wie des Levenberg-Marquardt-Algorithmus oder Konjugierten Gradientenabstieges.

Stochastischer Gradientenabstieg Für das Online-Training bietet sich stochastischer Gradientenabstieg an. Ist $err(x,t)$ eine nach der Netzausgabe differenzierbare Fehlerfunktion, so ergibt sich nach der Kettenregel

$$\nabla_{c_i} err = \sum_{j=1}^{m} \left(\frac{\partial err}{\partial y_j} \frac{\partial y_j}{\partial h_i}\right) \frac{\partial h_i}{\partial net_i} \frac{\partial net_i}{\partial dist_i} \nabla_{c_i} dist_i. \tag{3.30}$$

Mit $net_i = -\frac{(x-c_i)^2}{\sigma^2}$ und der Gaußschen Aktivierungsfunktion $h_i = \exp(net_i)$ ist dabei

$$\frac{\partial y_j}{\partial h_i} = \begin{cases} o_{ij} & : \text{ rbf} \\ \frac{o_{ij} - y_j}{\sum_{j \in A_P(x)} h(net_i)} & : \text{ nrbf} \end{cases}, \tag{3.31}$$

$$\frac{\partial h_i}{\partial net_i} = h_i \,, \; \frac{\partial net_i}{\partial dist_i} = -\frac{1}{\sigma^2} \text{ und } \nabla_{c_i} dist_i \;\; = \;\; -(x - c_i). \tag{3.32}$$

Betrachtet man die obigen Gleichungen, so fällt auf, daß der stochastische Gradient $\nabla_{c_i} err$ die Richtung des Differenzvektors zwischen Stimulus und Zentrum, $(x - c_i)$, aufweist. Dies bietet die Möglichkeit, seine Amplitude (die Schrittweite) zu normieren, deren Betrag jedoch keinesfalls $\| x - c_i \|$ übersteigen sollte. Unter Beibehaltung der Richtung und des Vorzeichens des Gradienten erhält man hiermit die Lernregel

$$\Delta c_i = \epsilon(t) \operatorname{sgn}\left(\sum_{j=1}^{m} \frac{\partial err}{\partial y_j} \frac{\partial y_j}{\partial h_i}\right) h_i \, (x - c_i). \tag{3.33}$$

Es fällt die Ähnlichkeit von (3.33) zu (3.28) bzw. für die Einer-Nachbarschaft zu (3.29) auf. Der Unterschied liegt zum einen darin, daß (3.33) den Fehler in der Netzausgabe berücksichtigt (Vorzeichen der Adaption), zum anderen, daß anstelle der topologischen Abstandsfunktion $d_G(.)$ der euklidische Abstand tritt.

Wenn der stochastische Gradient nach den Zentren gemäß (3.30) die Voraussetzungen des Robbins-Monro-Prozesses erfüllt (Eindeutigkeit der Nullstelle etc.), konvergiert der Robbins-Monro-Prozeß mit (3.33) und (3.30) zur selben Nullstelle.

Auch stochastischer Gradientenabstieg ist mit dem Problem konfrontiert, daß Zentrumsvektoren die Eingabemannigfaltigkeit verlassen können. Addition des Regularisierungsterms $\Omega = \nu Err^{VQ}(T)$ nach (2.6) zum Erwartungswert $Err(T) = E[err(x,t)]$ resultiert hier in einer Addition von $\nu(x - c_i)$ zum stochastischen Gradienten, so daß eine Mischform aus unüberwachtem und überwachtem Lernen entsteht. Eine andere Möglichkeit, das Ausreißen von Zentrumsvektoren zu verhindern, besteht darin, Adaptionsschritte nur dann durchzuführen, wenn das Vorzeichen $\operatorname{sgn}(\sum_{j=1}^{m} \frac{\partial err}{\partial y_j})$ in 3.33 positiv ist. Es ergibt sich die *restringierte Gradientenabstiegsregel*

$$\Delta c_i = \begin{cases} \epsilon(t)\,(x - c_i) : & \sum_{j=1}^{m} \frac{\partial err}{\partial y_j} \frac{\partial y_j}{\partial h_i} > 0 \\ 0 & : \; sonst \end{cases} \tag{3.34}$$

Offensichtlich gilt:

Bemerkung 3.5 *Liegen die Eingaben x und die initiale Zentrumsvektoren c_i in einer konvexen Teilmenge M des Eingaberaumes $\mathbb{R}^n$, so verlassen die Zentrumsvektoren M durch Anwendung von (3.34) diese nicht, falls nur stets $\epsilon(t) \leq 1$.*

Die Verzerrung durch Berücksichtigung nur der Gradienten mit positivem Vorzeichen wird um so kleiner, je besser sich $E[err(x,t)]$ durch eine quadratische Form auf den Voronoizellen approximieren läßt und je symmetrischer die Voronoizellen sind. Die quadratische Form bewirkt, daß die Ableitung linear wird und damit $\nabla_{c_i} err(x) \approx -\nabla_{c_i} err(x + 2(c_i - x))$ gilt. Die Symmetrie sorgt dafür, daß $x + 2(c_i - x)$ in der Voronoizelle von c_i liegt.

Wie auch bei der Ausgabeberechnung und der Adaption der Ausgabegewichte, erzielen DCS-Netze mit beschränkter Nachbarschaft gegenüber RBF-Netzen eine durchschnittliche serielle Beschleunigung von $\frac{N}{l}$ für einen Zentrenadaptionsschritt, wenn l die durchschnittliche Größe der aktiven Knotenmenge angibt. Durch die in (3.30) enthaltene Summe über alle Ausgabegewichte wirkt sich die Beschleunigung auch hier besonders bei großen Ausgabedimensionen aus.

Abschließend sei noch auf die Ähnlichkeit von (3.33) zur LVQ1-Regel (2.18) von Kohonen et al. verwiesen. Tatsächlich liest sich (3.33) als "Fuzzy"-Version der LVQ1-Regel, wobei das Vorzeichen durch die Ableitung des Fehlers nach der Aktivierung anstatt durch die Klassenzugehörigkeit bestimmt wird. Unter Voraussetzung einer 1-von-m Kodierung der Klassenzugehörigkeit und der quadratischen Fehlerfunktion versucht das DCS-Netz, die posteriori Wahrscheinlichkeiten der Klassenzugehörigskeit $p(C_i|x)$ zu approximieren, vgl. 2.1.1, so daß durch (3.33) ein bessere Approximation der posteriori Wahrscheinlichkeiten angestrebt wird. Die Approximation dieser Wahrscheinlichkeiten ist auch Intention der LVQ1-Regel, siehe [70].

Fehlermodulierte Lernregel zur Zentrenadaption Um eine am Ausgabefehler orientierte Lernregel zur Zentrenadaption zu erhalten, müssen nicht unbedingt Gradienten berechnet werden. Es kann auch die in den Fehlervariablen r_i akkumulierte Information über den Verlauf des Ausgabefehlers genutzt werden. Der im folgenden vorzustellenden *fehlermodulierten Lernregel* liegen dabei zwei Überlegungen zugrunde: Erstens sollte der mittlere Ausgabefehler (die Varianz) pro Voronoizelle (3.20) über dem gesamten Eingaberaum im Gleichgewichtszustand *gleich* klein sein. Zweitens kann der Ausgabefehler in einem Gebiet des Eingaberaumes dadurch verkleinert werden, daß die Anzahl der Zentren (Stützstellen) in diesem Gebiet vergrößert wird, d.h. die Auflösung erhöht wird. Daraus folgt, daß ein Zentrum c_i nur in dem Maß in Richtung des Eingabestimulus x verschoben werden sollte, wie der Ausgabefehler r_{bmu_x} in der Voronoizelle, in der x liegt, größer ist als r_i. Für $r_i = r_{bmu_x}$ sollte keine Verschiebung in Richtung von x stattfinden. Dieses Prinzip wird durch folgende Lernregel für eine Einer-Nachbarschaft realisiert, wobei $g : \mathbb{R}_{\geq 0} \rightarrow [0,1]$ mit $g(0) = 1$ und $g(1) = 0$ eine monoton fallende Funktion ist[9]:

$$\Delta c_i = \begin{cases} \epsilon_{Nh}(t)\,(x - c_i)\,g(\frac{r_i}{r_{bmu_x}}) : & i \in Nh^{(1)}(bmu_x) \backslash bmu_x \\ \epsilon_{bmu}(t)\,(x - c_i)\,g(\frac{\bar{r}}{r_{bmu_x}}) : & i = bmu_x \\ 0 & : \; sonst \end{cases} . \tag{3.35}$$

Bei dieser Lernregel werden zusätzlich Bewegungen von c_{bmu_x} erlaubt, falls der Ausgabefehler r_{bmu_x} der betreffenden Voronoizelle kleiner als der Fehler in den angrenzenden Voronoizellen, $\bar{r} = \frac{1}{|Nh^{(1)}(bmu_x)|-1} \sum_{i \in Nh^{(1)}(bmu_x) \backslash bmu_x}$, ist. Die Bezeichnung *fehlermodulierte*

[9]Es wird im folgenden stets $g(z) = 2z^2 - 3z + 1 \; : \; z \in [0,1]$ und $g(z) = 0 \; : \; z > 1$ benutzt.

Lernregel bezieht sich auf die Ähnlichkeit von (3.35) mit (3.4.3), der Fritzke-Lernregel. Der einzige Unterschied ist die fehlerabhängige Modulation. Man beachte, daß im Zustand einer Gleichverteilung der mittleren Ausgabefehler, d.h. $r_i = r_j$ für $1 \leq i \neq j \leq N$, tatsächlich keine Zentrenbewegung mehr stattfindet, die Zentrenanordnung also *stabil* ist.

Die fehlermodulierte Lernregel wurde entwickelt, um das Problem nicht-stationärer Wahrscheinlichkeitsverteilungen beim Erlernen eines Regelungsverhaltens für typische dynamische Systeme zu entschärfen. Diese Systeme bewegen sich nach Erlernen einer korrekten Regelungsstrategie nur noch auf einer bestimmten Soll-Trajektorie im Zustandsraum. Auf dieser Trajektorie wird die Eingabedichte daher mit der Zeit am größten. Eine allein von der Wahrscheinlichkeitsdichte im Eingaberaum abhängige Lernregel wie die Kohonenregel oder die Fritzkeregel würde mit der Zeit sämtliche Knoten auf dieser Trajektorie plazieren. Damit würden die für die Regelung während der Einschwingphase relevanten Regionen des Zustandsraumes nicht mehr überdeckt und das Regelungsverhalten auf diesen Regionen verlernt. Die fehlermodulierte Lernregel in Verbindung mit einer von der Wahrscheinlichkeitsdichte unabhängigen Lernregel zur Varianzprädiktion, z.B. (3.24), hingegen stoppt diese Magnetwirkung der Trajektorie, wenn der dort erreichte Fehler kleiner wird als der Fehler in den sonstigen überdeckten Regionen des Zustandsraumes. Die Zentrenverteilung bleibt stabil. Man beachte, daß stochastischer Gradientenabstieg diesen Stabilisierungseffekt im allgemeinen nicht aufweist: Er minimiert den Erwartungswert des Fehlers, d.h. den Fehler gewichtet mit der Wahrscheinlichkeitsdichte. Der Fehler akkumuliert sich in den Gebieten hoher Eingabedichte und damit auf der Soll-Trajektorie, was auch in diesem Falle zu einem (wenn auch abgeschwächten) Magneteffekt der Trajektorie führt. Allein ein komplettes Verschwinden des Fehlers auf der Solltrajektorie kann diesen Effekt stoppen.

Probleme im Zusammenhang mit der fehlermodulierten Lernregel sind bei inhomogenen Rauschverteilungen zu erwarten: Wie in 3.4.2 diskutiert, setzt sich der Erwartungswert des Ausgabefehlers aus dem tatsächlichen Approximationsfehler und einem vom Rauschen abhängigen Term (der Rauschvarianz) zusammen. Starkes Rauschen in einer bestimmten Region des Eingaberaumes führt dann solange zu einer Anziehung von Knoten, bis der Ausgabefehler in den übrigen Regionen gleich (schlecht) ist.

Die fehlermodulierte Lernregel kommt in den Abschnitten 4.4 und 5.2 zur Anwendung. Sie wurde auch von anderen Forschungsgruppen erfolgreich eingesetzt, so in [116] zum inkrementellen Lernen verhaltensbasierter Kategorien eines autonomen mobilen KHEPERA^{TM}-Roboters.

3.4.4 Adaption der Radien der Basisfunktionen

Anders als die Überschrift dieses Abschnitts impliziert, wurden die Radien der Basisfunktionen im Rahmen der in dieser Arbeit beschriebenen Versuche *nicht* adaptiert. Dies hat im wesentlichen zwei Gründe: Erstens kann mit variablen Radien nicht mehr garantiert werden, daß für Knoten i, j mit $\| c_i - x \| > \| c_j - x \|$ auch $h(\| c_i - x \| / \sigma_i) > h(\| c_j - x \| / \sigma_j)$ gilt. So müßte beispielsweise $Nh^{(1)}(bmu_x)$ nicht notwendigerweise die am stärksten aktivierten Knoten umfassen, was zu Inkonsistenzen mit dem auf dem euklidischen Abstand beruhenden Erlernen von OTPMs führt. Selbst die bmu_x müßte nicht zwangsläufig der

Knoten mit der höchsten Aktivität sein[10]. Zweitens haben konstante, gleich große Radien beim Einsatz in normalisierten DCS-Netzen die in Abschnitt 3.3.4 diskutierte Eigenschaft der Rauschunterdrückung durch effektive Projektion auf die Eingabemannigfaltigkeit. Diese geht bei variablen, unerschiedlich großen rezeptiven Feldern verloren. Schließlich ist der Effekt einer Variabilität der Radien gerade im Falle des Batch-Lernens nur schwer kontrollierbar: So kann für $\sigma^2 \to 0$ durchaus der Fehler auf der Trainingsmenge ein Minimum annehmen, aber jede Generalisierungsfähigkeit verloren gehen. Aus diesem Grund muß, wie in Abschnitt 2.4.2 diskutiert, eine Regularisierung des Problems durch einen Glättungsterm erfolgen. Da ein ortsinvarianter Regularisierungsterm aber die Größe von σ eindeutig bestimmt (siehe (2.31) und (2.32)) , können anstelle des Austestens verschiedener Regularisierer gleich verschiedene Werte von σ getestet werden.

Nearest Neighbor Regel zur Radienadaption

Sollen die Größen der rezeptiven Felder dennoch adaptiv gestaltet werden, so bietet sich als unüberwachte Adaptionsregel an, die Größe σ^2 proportional zur (Eingabe-) Varianz der Trainingsdaten in der Voronoizelle des entsprechenden Zentrums zu setzen [90]. Diese kann im Rahmen eines Batch-Verfahrens in einem Durchlauf durch die Trainingsmenge berechnet werden, online läßt sie sich als gleitender Mittelwert der quadratischen Einzelabweichungen berechnen. In DCS-Netzen mit Einer-Nachbarschaft kann zur Erzielung eines ähnlichen Effekts σ^2 proportional zum mittleren quadratischen Abstand des Zentrums und seiner direkten topologischen Nachbarn gesetzt werden,

$$\sigma_i^2 \approx \sum_{j \in Nh^{(1)}(bmu_x)} (c_j - c_i)^2. \tag{3.36}$$

Diese Adaptionsregel nutzt die OTPM und ist für Online-Lernverfahren geeignet. Sie kann auch in Verbindung mit RBF-Netzen eingesetzt werden und ist ein Beispiel dafür, daß sich das Lernen topologischer Beziehungen auch dann lohnt, wenn diese nicht unmittelbar in die Evaluierungsfunktion einfließen. Man beachte, daß die nach (3.36) bestimmten Größen keine freien Parameter sind, da sie in eindeutiger Weise von den Zentren abhängen. Eine Adaption nach (3.36) hat das Ziel, die Aktivierungsfunktionen im Mittel so überlappen zu lassen, daß in Analogie zu den Zugehörigkeitsfunktionen in der Fuzzy-Logic zwischen den entsprechenden Ausgabevektoren gut interpoliert aber der Überlapp an den Zentren möglichst gering wird.

Stochastischer Gradientenabstieg zur Radienadaption

Schließlich kann zur überwachten Adaption der Größen der rezeptiven Felder natürlich auch der Gradientenabstieg (batch oder stochastisch) eingesetzt werden, wobei jedoch den

[10] Es wäre allerdings denkbar, daß die Topologie mit einer Hebbschen Lernregel aufgrund der Antworten der Aktivierungsfunktionen (mit unterschiedlichen Radien) und nicht des euklidischen Abstandes erlernt wird. Letzteres würde eine *Filtermetrik* induzieren. Während diese Verallgemeinerung noch sinnvoll erscheint, wäre aber die Wechselwirkung von Adaption der Topologie und gleichzeitiger Adaption der Radien nur schwer interpretierbar.

eingangs erwähnten negativen Effekten der überwachten Adaption Rechnung getragen werden muß. Es ist

$$\nabla_{\sigma_i} err = \sum_{j=1}^{m} \left(\frac{\partial err}{\partial y_j} \frac{\partial y_j}{\partial h_i}\right) \frac{\partial h_i}{\partial net_i} \frac{\partial net_i}{\partial \sigma_i}, \tag{3.37}$$

mit $\frac{\partial y_j}{\partial h_i}$ und $\frac{\partial h_i}{\partial net_i}$ für Gaußsche Aktivierungsfunktionen nach (3.31), respektive (3.32), und

$$\frac{\partial net_i}{\partial \sigma_i} = \frac{(x - c_i)^2}{\sigma_i^3}. \tag{3.38}$$

Bei Anwendung dieser Lernregel ist zur Vermeidung von Instabilitäten unbedingt darauf zu achten, daß keine Vorzeichenwechsel von σ eintreten.

3.4.5 Adaption der Topologie

In diesem Abschnitt werden Lernregeln zur Adaption der lateralen Verbindungsstruktur von DCS-Netzen in Richtung einer OTPM diskutiert.

Einfache Hebb-Regel

Für eine feste Menge von Zentrumsvektoren S ergibt sich die Lernregel direkt aus Definition 3.1: Es müssen bei Präsentation einer Eingabe x nur das euklidisch nächste und zweitnächste Zentrum, c_{bmu_x} und c_{smu_x}, bestimmt und daraufhin die beiden Knoten bmu_x und smu_x verbunden werden. Damit ergibt sich die Lernregel

$$E(t+1) = E(t) \cup \{(bmu_x, smu_x), (smu_x, bmu_x)\}, \tag{3.39}$$

die einen bidirektionalen, ungewichteten Graphen aufbaut. Im Falle einer endlichen Trainingsmenge ergibt sich die $OTPM_T(S)$ nach einem Durchlauf durch die Trainingsmenge, im Falle des Online-Lernens mit einer unendlichen Trainingsmenge konvergiert die Lernregel mit Wahrscheinlichkeit 1. Wie von Martinetz in [81] ausgeführt und in Abschnitt 2.3.3 erläutert, kann diese Lernregel als Hebbsches Wettbewerbslernen verstanden werden.

Einfache Hebb-Regel mit Abklingkonstante

Falls die Zentrumsvektoren in S durch Adaption ihre Position ändern können, gleichzeitig aber online $OTPM_T(S)$ zu approximieren ist, so müssen nach (3.39) eingefügte, aber aufgrund einer Verschiebung von Zentrumsvektoren obsolete (falsche) Kanten abgebaut werden. Dies kann über einen zeitlichen Abbauprozeß der Kanten geschehen, wobei einer Kante (i, j) beim Einfügen oder Erneuern nach (3.39) ein Gewicht von $C_{ij} = 1$ zugeordnet wird, das exponentiell abnimmt, falls Knoten i und j nicht bmu_x und smu_x sind. Eine Kante wird dann gelöscht, wenn ihr Gewicht einen Schwellwert ξ unterschreitet. Zusammengefaßt ergibt sich

$$C_{ij} = \begin{cases} 1 & : \{i, j\} = \{bmu_x, smu_x\} \\ 0 & : C_{ij} < \xi \\ \kappa\, C_{ij} & : \text{sonst} \end{cases}, \tag{3.40}$$

wobei $0 \leq \kappa \leq 1$ die Abklingkonstante angibt und ein Nullsetzen von C_{ij} zu einer Entfernung von (i, j) aus E führt. Die Abklingkonstante kann dabei nicht unabhängig vom Schwellwert ξ gewählt werden. Lernregel (3.40) baut einen bidirektionalen Graphen mit symmetrischer Adjazenzmatrix C auf. Sie betrifft durch den Abbauprozeß in jedem Schritt alle Kanten und hat daher Zeitkomplexität $O(N^2)$. Bereits Martinetz wies in [80] im Zusammenhang mit einer ähnlichen Lernregel darauf hin, daß ein Abbau nur der zur momentanen bmu_x adjazenten Verbindungen ausreichend ist und in einer linearen Zeitkomplexität resultiert.

Unsymmetrische Hebb-Regel

Zur Vermeidung *toter Knoten*, d.h. von Knoten, die mit keinem anderen mehr verbunden sind und auch nicht mehr *bmu* werden, wurde in [3] die Lernregel (3.40) derart modifiziert, daß a) nur noch die von der bmu_x ausgehenden Verbindungen dem Abbauprozeß unterworfen werden und b) die Verbindungen (bmu_x, j) und (j, bmu_x) nur dann gekappt werden, wenn beide einen Schwellwert unterschritten haben. Es ergibt sich

$$C_{ij} = \begin{cases} 1 & : \{i,j\} = \{bmu_x, smu_x\} \\ 0 & : i = bmu_x \wedge j \neq smu_x \wedge (\kappa C_{ij} < \xi \wedge C_{ji} < \xi) \\ 0 & : j = bmu_x \wedge i \neq smu_x \wedge (\kappa C_{ji} < \xi \wedge C_{ij} < \xi) \\ \kappa C_{ij} & : i = bmu_x \wedge j \neq smu_x \wedge (\kappa C_{ij} \geq \xi \wedge C_{ji} \geq \xi) \\ C_{ij} & : \text{sonst} \end{cases} . \tag{3.41}$$

Bedingung a) resultiert in einer unsymmetrischen Adjazenzmatrix C, Bedingung b) zusammen mit a) verringert das Risiko, die gegenseitigen Verbindungen zu einem Knoten zu trennen, der nicht mehr *bmu* wird. Die Lernregel kommt in den Abschnitten 4.4 und 6.3.4 zur Anwendung, sie wurde auch in [116] genutzt.

Modifizierte Hebb-Regel

Ein Nachteil beider bislang vorgestellten Lernregeln mit Abklingkonstante ist, daß die Kombination aus Abklingkonstante und Schwellwert nicht unabhängig von der (im allgemeinen unbekannten) Kanten- und Knotenzahl der OTPM gewählt werden kann. Dies ergibt sich daraus, daß mit einer steigenden Zahl von Kanten die Wahrscheinlichkeit, eine Kante (i, j) aufzufrischen, sinkt. Die folgende Lernregel behebt dieses Problem durch die Forderung nach Konstanz der Summe über die Gewichte aller von einem Knoten emanierenden Kanten, d.h. $\sum_j C_{ij} = 1 \; \forall i \in V$, und einem Löschkriterium, das die Anzahl der von Knoten i ausgehenden Kanten m_i berücksichtigt. Sie läßt sich wie folgt zusammenfassen:

- *Erzeugung einer neuen Verbindung:* Falls bmu_x und smu_x nicht bereits verbunden sind, werden sie nach

$$C_{ij} = \frac{1}{m_i + 1} \quad \text{und} \quad C_{ik} := C_{ik}(1 - C_{ij}) \; \forall k \in Nh^{(1)}(i) \backslash \{j\} \tag{3.42}$$

  mit $(i, j) \in \{(smu_x, bmu_x), (bmu_x, smu_x)\}$ gegenseitig verbunden.

- *Adaption bestehender Verbindungen:* Falls bmu_x und smu_x bereits verbunden sind, so sind ihre gegenseitigen Verbindungen zu verstärken, und zwar auf Kosten der restlichen Verbindungen, die hierdurch abgeschwächt werden. Dies wird durch

$$C_{ij} := C_{ij} + \sum_{k \in Nh^{(1)}(i) \setminus \{j\}} \kappa\, C_{ik} \quad \text{und} \quad C_{ik} := C_{ik}(1-\kappa)\; \forall k \in Nh^{(1)}(i) \setminus \{j\} \tag{3.43}$$

mit $(i,j) \in \{(smu_x, bmu_x), (bmu_x, smu_x)\}$ erreicht.

- *Löschen einer Verbindung:* Wie auch bei Lernregel (3.41) werden zur Vermeidung toter Knoten die gegenseitigen Verbindungen zwischen der bmu_x und einem ihrer Nachbarn nur dann gelöscht, wenn beide den Schwellwert unterschreiten:

$$C_{bmu_x,j} = C_{j,bmu_x} = 0 \quad \text{falls} \quad C_{bmu_x,j} \leq \frac{\xi}{m_{bmu_x}} \wedge C_{j,bmu_x} \leq \frac{\xi}{m_j}. \tag{3.44}$$

Nach dem Löschen einer Verbindung sind die von der bmu_x und dem Nachbarn ausgehenden Verbindungen auf 1 zu renormalisieren.

Offensichtlich bleibt die Summe über die Gewichte aller von einem Knoten emanierenden Kanten bei Anwendung der Regel konstant 1. In Verbindung mit dem von m_i abhängigen Löschkriterium bleibt unter der Voraussetzung, daß alle Kanten eines Knotens die gleiche Trefferwahrscheinlichkeit haben, die Lernregel unabhängig von der Zahl der Kanten und damit der absoluten Größe der Trefferwahrscheinlichkeit. Die Zeitkomplexität eines Adaptionsschrittes bleibt linear in der Anzahl der Knoten.

3.4.6 Adaption der Modellordnung - Wachstum des Netzes

Nachdem in den letzten Abschnitten Verfahren zur Adaption der Parameter und der topologischen Karte diskutiert wurden, sollen nun Lernverfahren zur Adaption der Modellordnung für DCS-Netze vorgestellt werden. Dabei ist die Grundidee eine *fehlergetriebene Verfeinerung* der durch das DCS-Netz beschriebenen Abbildung: In den Regionen des Eingaberaumes, in denen die Approximation schlecht ist, wird durch Einfügen zusätzlicher Knoten die Auflösung erhöht und so die Approximation lokal verbessert[11]. Hierbei wird die Lokalität der rezeptiven Felder von DCS-Netzen in doppelter Weise ausgenutzt. Zum einen ermöglichen erst die lokalen rezeptiven Felder, den über die Voronoizellen gemittelten Ausgabefehler wie in Abschnitt 3.4.2 beschrieben lokal zu schätzen und zu "lokalisieren", zum anderen ist die Auswirkung eines neu eingefügten Knotens, d.h. der Erhöhung der Modellordnung, auf die Abbildung nur von lokaler Wirkung. Letzteres erübrigt ein komplettes Neutraining der Parameter der restlichen Knoten, das Training kann *inkrementell* stattfinden.

Bei Adaption der Modellordnung durch Einfügen neuer Neuronen anhand einer endlichen Trainingsmenge (Batch-Training) gelten die in Abschnitt 2.1.2 im Zusammenhang

[11]Dieses Prinzip der fehlergetriebenen Verfeinerung ist durchaus verbreitet. Es findet sich z.B. in Multiskalen- und Mehrgitteransätzen, wo es zu einer (regulären) Verfeinerung der Skalen bzw. Gitter herangezogen wird.

mit dem Bias/Variance Dilemma formulierten generellen Aussagen zum Generalisierungs- und Overfittingproblem: Der Fehler auf einer Trainingsmenge kann durch Erhöhung der Modellordnung beliebig verringert werden, was jedoch nicht mit einer Abnahme des Erwartungswertes des Fehlers korrespondiert, diesen im Gegenteil beim Auswendiglernen erhöht. Die im Rahmen dieser Arbeit gewählte Methode zur Vermeidung von Overfitting ist Kreuzvalidierung, so daß der Einfügeprozeß gestoppt wird, wenn der Fehler auf einer Testmenge sein Minimum erreicht hat.

Beim Online-Training tritt das Overfittingproblem nur in abgeschwächter Form auf: Eine zu hohe Modellordnung führt hier aufgrund der unendlichen Trainingsmenge nicht zu einer Vergrößerung des Erwartungswertes des Fehlers, Rauschen wird nicht approximiert. Allerdings wird der Fehler nach Erreichen einer optimalen Modellordnung auch nicht mehr abnehmen, so daß das Wachstum theoretisch dann zu stoppen ist, wenn bei Erhöhung der Modellordnung keine Abnahme des Ausgabefehlers mehr zu beobachten ist. Praktisch wird im Rahmen dieser Arbeit das Wachstum beim Online-Training dann gestoppt, wenn entweder eine geforderte Präzision δ_{err} unterschritten oder eine vorgegebene Maximalzahl von Knoten N_{max} überschritten wird.

Fehlergetriebene Adaption der Modellordnung

Eine fehlergetriebene Verfeinerungsstrategie erfordert die Spezifikation dreier Punkte. Diese sind erstens die Berechnung des "Fehlers", zweitens das Einfügungskritererium (wann eine Adaption der Modellordnung ausgelöst wird), und drittens die Initialisierung der neu eingefügten Berechnungsknoten. Im folgenden werden die im Rahmen dieser Arbeit zum Einsatz kommenden Varianten zusammenhängend diskutiert.

Lokale Fehlerberechnung Als lokales Fehlermaß dient stets die durchschnittliche Varianz σ_i^2 pro Voronoizelle nach (3.20). Im Falle einer endlichen Trainingsmenge kann diese nach (3.23) durch (Batch-) *Fehlerakkumulation* explizit berechnet werden:

$$r_i = \frac{1}{|T_i|} \sum_{T_i = \{x | (x,t) \in T \wedge x \in V_i\}} err(x,t). \tag{3.45}$$

Bei der Berechnung der lokalen Fehler nach (3.45) fließt die a priori Wahrscheinlichkeit $p(x \in V_i)$ und damit die Wahrscheinlichkeitsdichte der Eingaben nicht mit ein. Soll diese berücksichtigt werden und damit als lokales Fehlermaß die anteilige durchschnittliche Varianz pro Voronoizelle $\tilde{\sigma}_i^2 = \sigma_i^2\, p(V_i)$ dienen, so ist in (3.45) statt durch $|T_i|$ durch $|T|$ zu dividieren.

Für eine Online-Adaption der Modellordnung kann anstelle der Batch-Fehlerakkumulation nach (3.45) eine Online-Variante, die *Online-Fehlerakkumulation*, benutzt werden, indem der Fehler nur über die nächsten λ Schritte akkumuliert wird. Es ergibt sich

$$r_i = \frac{1}{|\lambda|} \sum_{\{x | (x,t) \in T_\lambda \wedge x \in V_i\}} err(x,t), \tag{3.46}$$

wobei (3.46) die Wahrscheinlichkeitsdichte der Eingaben berücksichtigt. Nach einer Adaption der Modellordnung sind sämtliche Fehlervariablen r_i auf null zu setzen.

Für eine reine Online-Adaption ohne Berücksichtigung der Wahrscheinlichkeitsdichte der Eingaben kann der lokale Erwartungswert des Ausgabefehlers durch *gleitende Mittelwertberechnung* nach (3.24) mit einer konstanten Adaptionsrate $\beta(t) = \beta$ berechnet werden,

$$\Delta r_i = \begin{cases} \beta\,(err(x,t) - r_i): & i = bmu_x \\ 0 & : sonst \end{cases} \quad . \tag{3.47}$$

Möchte man die anteilige durchschnittliche Varianz $\tilde{\sigma}_i^2$ schätzen (Berücksichtigung der Wahrscheinlichkeitsdichte der Eingaben), so sind die nach (3.47) geschätzten Fehlerterme mit den a priori Wahrscheinlichkeiten $p(V_i)$ zu multiplizieren, die beim Online-Training nicht unmittelbar zur Verfügung stehen. Letztere lassen sich aber durch Einführung eines zusätzlichen Zählers (Integrators) pro Knoten schätzen.

Einfügungskriterien Während beim Batch-Training die Modellordnung im allgemeinen nach einem kompletten Durchlauf durch die Trainingsmenge (Berechnung der lokalen Varianz nach (3.45)) angepaßt wird, ergeben sich beim Online-Training zwei mögliche Vorgehensweisen. Die erste besteht darin, eine Adaption der Modellordnung stets nach einer vorgegebenen Anzahl von λ_{ins} Trainingsschritten vorzunehmen, und hier wie beim Batch-Training als Region zur Verfeinerung durch Neueinfügen eines Knotens die Voronoizelle mit der größten Varianz auszuwählen, d.h. V_i mit $i = \arg\max_j r_j$. Wenn a priori ein vom DCS-Netz zu realisierender (und erreichbarer!) minimaler Ausgabefehler δ_{err} angegeben werden kann, bietet sich die zweite Möglichkeit an, nämlich einen neuen Knoten immer dann einzufügen, wenn die Fehlervariable r_{bmu_x} der bmu_x und der momentane Fehler $err(x,t)$ die geforderte Präzision δ_{err} überschreiten:

$$r_{bmu_x} > \delta_{err} \quad \text{und} \quad err(x,t) > \delta_{err}. \tag{3.48}$$

Diese präzisionsgesteuerte Online-Wachstumsstrategie ist rein lokal und rein fehlergetrieben. Dadurch, daß das Einfügekriterium nicht nur vom momentanen Fehler sondern auch von der Fehlervariable der bmu_x abhängt, ist die Regel weniger rauschempfindlich als z.B. die Plattsche Regel [98]. Durch das Nullsetzen der Fehlervariablen nach dem Einfügen und die Wahl einer kleinen Integrationskonstante β wird zu häufiges Einfügen in unmittelbarer Nähe ein und derselben Stützstelle vermieden.

Initialisierung neuer Knoten Ein Batch-Ansatz zur Initialisierung eines neu einzufügenden Knotens besteht darin, das Zentrum c_i und den Ausgabevektor o_i des neuen Knotens mit der Trainingsein- und ausgabe zu initialisieren, die den größten Fehler erzeugt,

$$c_i = x \quad \text{und} \quad o_i = t \quad \text{mit} \quad [(x,t) \in T_i] \wedge [err(x,t) > err(x',t')\ \forall (x',t') \in T_i], \tag{3.49}$$

wodurch der Fehler maximal verkleinert werden sollte. Dieses Verfahren, das als *Einfügen von Prototypen* aufgefaßt werden kann, garantiert, daß das neue Zentrum in der Eingabemannigfaltigkeit liegt. Beim Online-Lernen bedeutet Einfügung von Prototypen, daß der neue Berechnungsknoten mit der Ein- und Ausgabe des aktuellen Trainingsdatums initialisiert wird,

$$c_i = x \quad \text{und} \quad o_i = t. \tag{3.50}$$

Dies führt in DCS-Netzen zu sogenannten *One-shot-Lernen*.

Eine Alternative zum Einfügen von Prototypen ist, unter den direkten Nachbarzellen der zu verfeinernden Zelle V_i diejenige mit dem größten Fehler auszusuchen und zwischen den entsprechenden Zentren ein neues Zentrum einzufügen. Hierbei werden Zentren und Ausgabevektoren mit einem fehlerabhängigen Interpolationsfaktor $\gamma = \frac{r_j}{r_i + r_j}$ nach

$$c_{new} = c_i + \gamma(c_j - c_i) \quad \text{und} \quad o_{new} = o_i + \gamma(o_j - o_i) \tag{3.51}$$

initialisiert. Diese *Interpolationsmethode* ist robuster gegen Ausreißer als das Einfügen von Prototypen, da sie nur die gemittelten Fehler r_i berücksichtigt. Ihr Nachteil ist, daß ein neues Zentrum nach (3.51) nur in dem von den existierenden Zentren aufgespannten Unterraum eingefügt wird. Es ist daher eine zusätzliche Verteilung der Zentren durch die in Abschnitt 3.4.3 beschriebenen Verfahren zwingend erforderlich. Ansonsten würden, wenn man mit nur zwei Zentren c_1 und c_2 startet, alle neu eingefügten Zentren auf der Linie $[c_1, c_2]$ zu liegen kommen, der aufgespannte Unterraum wäre dann stets eindimensional.

Distanzgetriebene Adaption der Modellordnung

Parallel oder alternativ zum bislang diskutierten fehlergetriebenen Wachstum kommt insbesondere beim Online-Training *distanzgetriebenes Wachstum* in Betracht. Hierbei wird ein Prototyp (x, t) immer dann eingefügt, wenn der Abstand der Eingabe x zu seiner bmu_x eine Schwelle δ_{dist} überschreitet,

$$(x - c_{bmu_x})^2 > \delta_{dist}^2. \tag{3.52}$$

Als unüberwachtes Verfahren ist distanzgetriebenes dem fehlergetriebenen Wachstum im Sinne der Fehlerminimierung unterlegen. Um eine möglichst schnelle Überdeckung des Eingaberaumes zu erzielen, bietet sich dieses Verfahren aber dennoch an. Dabei sollte die Schwelle δ_{dist} in funktionalem Zusammenhang mit der Größe der rezeptiven Felder σ stehen. Man beachte, daß distanzgetriebene Einfügestrategien Spezialfälle der fehlergetriebenen darstellen, in denen der auf dem Eingaberaum definierte Quantisierungsfehler als Fehlermaß dient.

3.5 Alternative Ansätze

Es sei an dieser Stelle noch eine kurze Einordnung und Abgrenzung zu Arbeiten anderer Autoren vorgenommen, die sich mit ähnlichen Architekturen und Lernverfahren wie die der DCS beschäftigen.

Außer von Martinetz' Arbeiten zu topologieerhaltenden Netzen [80, 81] wurde die Entwicklung der DCS vor allem von Fritzkes *Wachsenden Zellstrukturen* (GCS) [37] inspiriert, an die sich auch die Namensgebung des Autors der vorliegenden Arbeit (Dynamische Zellstruktur) anlehnt. Bei GCS handelt es sich um ein wachsendes RBF-Netz, das ähnlich wie DCS-Netze eine zusätzliche laterale Verbindungsstruktur zwischen den RBF-Knoten aufweist. Diese ist aber nicht adaptiv sondern ähnlich dem Kohonengitter von vorgegebener Struktur (k-dimensionale Hypertetraeder). In einer Weiterentwicklung seiner GCS tauschte später auch Fritzke seine starre Verbindungsstruktur gegen adaptive, perfekt topologieerhaltende Karten aus und nennt diese neue Struktur *Growing Neural Gas* (GNG). Die Publikation des GNG und seine Anwendung für unüberwachtes Lernen von Topologien [38] erfolgte im November 1994 auf der NIPS' 94, zeitgleich mit der Veröffentlichung der DCS [30]. Die erste Präsentation der DCS fand bereits im Mai 1994 auf der ICANN'94 (Deadline: Dezember 1993) als Poster statt[12]. In der folgenden Zeit wurden vom Autor die in diesem Kapitel dargestellten Grundlagen und Lernregeln erarbeitet und DCS konsequent im Rahmen aller wesentlichen Lernparadigmen getestet und eingesetzt (Kapitel 4 - 6). In diesem Kontext stellt Fritzkes GNG eine spezielle (hinsichtlich der Lernregeln und des verwendeten Lernparadigmas) DCS dar.

Wachsende RBF-Netze wurden außer von Fritzke u.a. von Platt [98] und Hakala [48] verwendet. Mit der in Abschnitt 3.4.6 eingeführten Terminologie berechnet Fritzke den Ausgabefehler durch Online-Fehlerakkumulation und fügt neue Knoten fehlergetrieben nach einer vorgegebenen Anzahl von Adaptionsschritten mittels der Interpolationsmethode ein. Platt und Hakala arbeiten mit einer distanzgetriebenen Wachstumsstrategie, wobei allerdings ein neuer Knoten nur dann eingefügt wird, wenn der momentane Fehler $err(x,t)$ einen Schwellwert überschreitet. Ihre Wachstumsstrategien verzichten auf Varianzprädiktion und sind durch Berücksichtigung nur des momentanen Fehlers sehr empfindlich gegenüber Rauschen. Im Gegensatz hierzu wird in der vorliegenden Arbeit neben Diskussion und Anwendung einer ganzen Familie von Wachstumsstrategien die Adaption der Modellordnung von RBF-Netzen im Zusammenhang mit der Varianzprädiktion betrachtet und so auch der Einfluß von Rauschen erfaßt.

DCS-Netze stehen mit einem weiteren Approximationsverfahren, den *Parametrisierten Selbstorganisierenden Merkmalskarten* (PSOM) von Walter [130], in engem Zusammenhang. Deren Grundidee ist, eine Mannigfaltigkeit M *bekannter* topologischer Struktur und intrinsischer Dimensionalität d auf einem d-dimensionalen Gitter L abzutasten. In den Gitterpunkten $i \in L$ werden dann Basisfunktionen $H(i,s)$ mit Referenzvektoren $c_i \in M$ so konstruiert, daß sich mit der Abbildung

$$x' = \sum_i H(i,s)c_i \quad s \in S \subseteq \mathbb{R}^d,\, L \subset S \tag{3.53}$$

eine stetige und differenzierbare Approximation von M ergibt. Sie stellen damit die unmittelbare Erweiterung von Kohonens Merkmalskarten dar, bei denen an die Stelle der stetigen Approximation durch Superposition von Basisfunktionen (3.53) der einfache Table-Lookup tritt. Von den Basisfunktionen wird gefordert, daß sie die Orthonormalitätseigenschaft ($H(i,j) = \delta_{ij}$) und die Interpolationseigenschaft ($\sum_i H(i,s) = 1 \ \forall s \in S$) erfüllen. In [130] werden Lagrange- und Chebyshev-Polynome zur Realisierung der Basisfunktionen diskutiert. Werden die Basisfunktionen über radiale Basisfunktionen realisiert und die Knoten gemäß der Gitterstruktur verbunden, so ergibt sich ein DCS-Netz, für das Orthonormalitäts- und Interpolationseigenschaft im allgemeinen nur näherungsweise erfüllt werden können (abhängig von der Wahl der radialen Basisfunktion und der Anordnung der Zentren). Ein DCS-Netz mit beschränkter Nachbarschaftsfunktion stellt in diesem Zusammenhang eine effiziente aber unstetige Approximation der Mannigfaltigkeit dar, eine *lokale*

[12] Die Aufnahme in den Konferenzband wurde trotz durchweg positiver Kritik wegen Überlänge (6 statt 5 Seiten) abgelehnt.

PSOM [130]. Der Vorteil der DCS-Netze ist freilich, daß die Topologie der Mannigfaltigkeit nicht notwendigerweise wie bei der PSOM oder Kohonenkarte a priori bekannt sein muß, sondern selbständig in Form einer OTPM gelernt werden kann. Abbildung 3.6 liefert

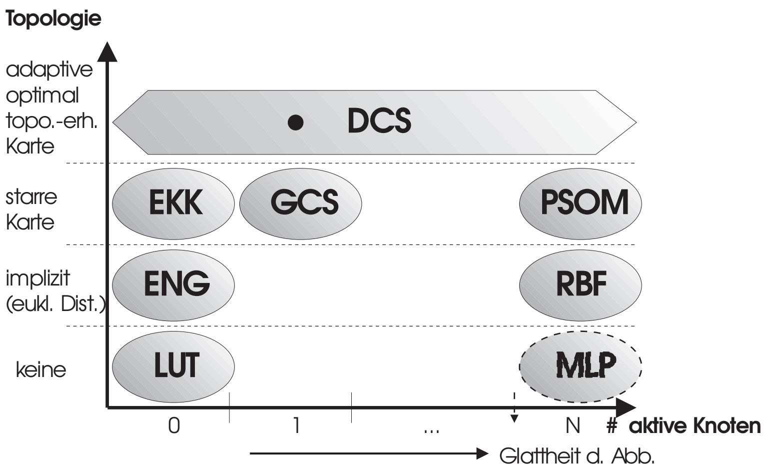


Abbildung 3.6: Einordnung der KNN-Modelle lokal rezeptiver Felder.

eine abschließende Einordnung der diskutierten KNN-Modelle lokal rezeptiver Felder zur Funktionsapproximation bezüglich der Repräsentation topologischer Beziehungen und des Approximationsverhaltens. Die erweiterte Kohonenkarte (EKK), GCS und PSOM nutzen topologische Karten fest vorgegebener, lokal nicht variabler Dimensionalität. Das erweiterte Neuronengas (ENG) sowie die RBF-Netze nutzen eine implizite topologische Kodierung über die Nachbarschaftsrangfolge bzw. Aktivierungsfunktion, die jeweils vom euklidischen Abstand zum Stimulus abhängt. Diese Kodierung ist nur für konvexe Mannigfaltigkeiten korrekt. Die einfache Lookup-Table (LUT) schließlich nutzt keine topologischen Beziehungen, genauso wie die MLP-Netze[13], die aufgrund der fehlenden Lokalität der Basisfunktionen hier nur gestrichelt eingezeichnet sind. Bezüglich ihres Approximationsverhaltens verhalten sich EKK, ENG und LUT identisch[14], sie realisieren jeweils eine Lookup-Table (ein aktiver Knoten). Allerdings nutzen ENG und EKK topologische Information zur Adaption aller Knoten. PSOM, RBF- und MLP-Netze beteiligen sämtliche Knoten an der Ausgabeberechnung (und Adaption). Sie realisieren stetig differenzierbare Abbildungen. Im GCS-Netz hingegen ist nur eine Einer-Nachbarschaft aktiv, die Abbildung ist unstetig. Mit abnehmendem Überlapp benachbarter rezeptiver Felder verliert diese Unstetigkeit allerdings ihre praktische Relevanz. Gleiches gilt für die in dieser Arbeit eingesetzte DCS mit Einer-Nachbarschaft, welche als einziges Modell adaptive, optimal topologieerhaltende Karten nutzt. Je nach Nachbarschaftsfunktion $Nh^{(i)}(.)$ variiert die DCS zwischen LUT und RBF-Netz, wobei die Glattheit der Evaluierungsfunktion für gegebene rezeptive Felder mit steigendem i zunimmt.

[13]Allerdings werden im Fall normalisierter Eingabevektoren euklidscher Abstand und Skalarprodukt und damit MLP- mit RBF-Netzen identisch.

[14]Zu beachten ist hier Fußnote 10 in Abschnitt 2.3.2

Kapitel 4

Überwachtes und Unüberwachtes Lernen - Ausgewählte Experimentelle Resultate

Nach der Einführung der DCS in Kapitel 3 wird in diesem Kapitel ihre Anwendung auf ausgewählte Probleme des überwachten und unüberwachten Lernens geschildert. Zuerst erfolgt eine Demonstration selbstorganisierenden Lernens von optimal topologieerhaltenden Merkmalskarten am Beispiel des 2-Spiralen-Problems. Daran anschließend wird anhand einer Reihe von Klassifikationsbenchmarks gezeigt, daß ein DCS-Netz auf überwachten Lernproblemen den Vergleich mit konkurrierenden KNN-Modellen nicht scheuen muß. Der Einsatz der DCS zur Klassifikation von Schlüsselpositionen in Gesichtsbildern stellt eine weitere interessante Anwendung dar, in der ein DCS-Netz als Verifikationsstufe in einem hybriden Klassifikationssystem zum Einsatz kommt. Die Kombination der modellbasierten Klassifikationsstufe und der DCS-gestützten Verifikationsstufe erfolgt nach dem Bayes Theorem. Das Kapitel schließt mit einem Experiment zum überwachten Lernen einer stabilen Regelungsstrategie für das Stabbalance-Problem ohne "Einfrieren" von Lernparametern.

Dieses Kapitel bietet eine Zusammenfassung der in [30, 29, 3, 56, 57] präsentierten Anwendungen und Ergebnisse zum überwachten und unüberwachten Lernen mit DCS.

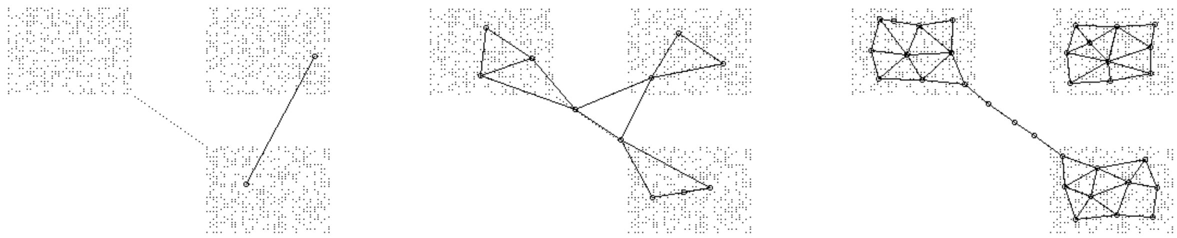
4.1 Selbstorganisierendes Lernen von OTPMs

In diesem Abschnitt wird anhand zweier 2-dimensionaler Datensätze illustriert, wie sich mittels eines einfachen unüberwachten Online-Lernverfahrens Zentren und laterale Verbindungen eines wachsenden DCS-Netzes so organisieren, daß die Zentren gleichförmig über die Eingabemannigfaltigkeit verteilt werden und gleichzeitig die Verbindungsstruktur gegen die OTPM der Zentren bezüglich der Trainingsmenge konvergiert.

4.1.1 Lernverfahren zum unüberwachten Online-Lernen

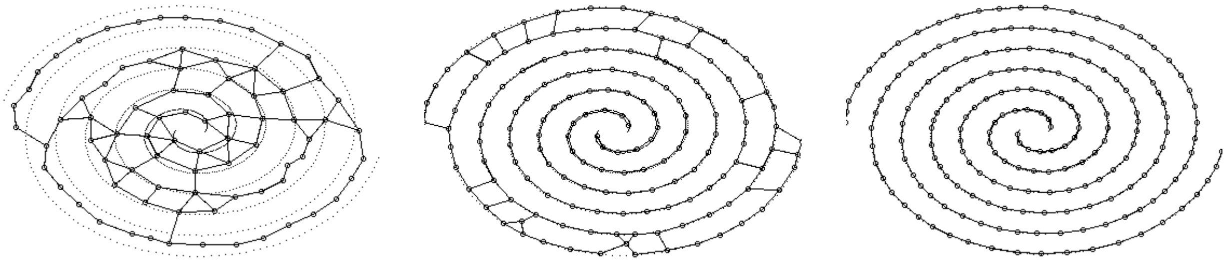
Das DCS-Netz wird mit zwei verbundenen Knoten initialisiert, deren Zentren zwei zufällig ausgewählten Trainingseingaben entsprechen. Online-Adaption der Zentren erfolgt nach der Fritzke-Lernregel (3.29), Online-Adaption der Verbindungsstruktur mittels der einfachen Hebb-Regel mit Abklingkonstante (3.40). Eine Adaption der Modellordnung (Wachstum des Netzes) erfolgt jeweils nach $\lambda_{ins} = |T|$ Schritten (Seite 3.4.6), wobei neue Knoten nach der Interpolationsmethode (3.51) initialisiert werden und der lokale Fehler über Online-Fehlerakkumulation (3.46) berechnet wird. Als Fehlersignal zur Adaption der Fehlervariablen dient der Vektorquantisierungsfehler $err^{VQ} = (x - c_{bmu_x})^2$. Der Abbruch des Lernverfahrens erfolgt, wenn das Netz eine vorgegebene Größe erreicht hat. Die Lernraten werden nicht eingefroren sondern bleiben über die gesamte Trainingsdauer konstant.

4.1.2 Experimentelle Resultate



a) b) c)

Abbildung 4.1: Selbstorganisierendes Lernen von OTPMs auf T_Q



a) b) c)

Abbildung 4.2: Selbstorganisierendes Lernen von OTPMs auf T_S

Als Trainingsdatensätze dienen zum einen eine Menge T_Q von 2000 zufälligen Abtastpunkten einer Eingabemannigfaltigkeit aus 3 Quadraten und einer Linie im $\mathbb{R}^2$, wobei letztere zwei der drei Quadrate verbindet, zum anderen eine Menge T_S von 962 Abtastpunkten einer Eingabemannigfaltigkeit aus zwei verschachtelten aber getrennten Spiralarmen. Abbildung

4.1 zeigt den Trainingsverlauf für T_Q, Abbildung 4.2 den Verlauf für T_S[1]. Deutlich zu sehen ist, wie die Zentren durch die über den Vektorquantisierungsfehler getriebene Einfügestrategie und die Fritzke-Lernregel eine den Quantisierungsfehler minimierende Verteilung annehmen (so fällt $Err^{VQ}(T_Q)$ von 100% bei 3 Knoten auf 3% bei 33 Knoten). Die Verteilung der Zentren folgt dabei der Eingabedichte der Trainingsmengen (die Zentrendichte nimmt vom Zentrum zur Peripherie der Spiralen monoton ab und folgt damit der ebenfalls abnehmenden Dichte der Abtastpunkte). Die Verbindungsstruktur der Knoten gleicht nach jedem Durchlauf durch die Trainingsmenge der $OTPM$ der Zentrenmenge bezüglich der jeweiligen Trainingsmenge. Letztere spiegelt die intrinsische topologische Struktur der Daten wieder: Im Falle von T_Q ergeben sich dort 2-dimensionale Verbindungsstrukturen, wo die 2-dimensionalen Quadrate abgetastet wurden, und eine 1-dimensionale Verbindungsstruktur, wo die 1-dimensionale Linie abgetastet wurde. Im Falle der Spiralen werden die zwei 1-dimensionalen Spiralarme durch zwei 1-dimensionale Verbindungsstrukturen separiert. Mit zunehmender Zentrendichte werden die optimal topologieerhaltenden Karten zu perfekt topologieerhaltenden Karten.

4.2 Die CMU-Klassifikations-Benchmarks

Anhand dreier Klassifikationsprobleme, die aus der Benchmarksammlung der Carnegie Mellon Universität stammen [35], soll jetzt die Eignung der DCS zum überwachten Lernen demonstriert werden.

4.2.1 Lernverfahren zum überwachten Online-Lernen

Das eingesetzte überwachte Online-Lernverfahren ähnelt dem unüberwachten Lernverfahren aus Abschnitt 4.1.1. Von diesem unterscheidet es sich nur dadurch, daß

- zusätzlich eine Ausgabeberechnung nach der Evaluierungsfunktion f^{rbf} (3.1) mit $\mathcal{A}_P(x) = Nh^{(1)}(bmu_x)$ (Einer-Nachbarschaft) stattfindet,
- die Ausgebevektoren nach der LMS-Regel (3.15) adaptiert werden und
- zur Online-Adaption der lokalen Fehlervariablen nicht err^{VQ}, sondern der quadratische Ausgabefehler $err^2 = (t - y(x))^2$ herangezogen wird.

Auch hier werden die Lernraten nicht eingefroren sondern bleiben über die gesamte Trainingsdauer konstant. Durch Minimierung der quadratischen Fehlerfunktion Err^2 bei Verwendung einer 1-aus-m Codierung der Klassen wird durch dieses Lernverfahren versucht, die a posteriori Klassenwahrscheinlichkeiten zu lernen (siehe Seite 14).

4.2.2 Experimentelle Resultate

Als stochastisches Approximationsverfahren ist das Lernverfahren abhängig von der Reihenfolge der Beispielpräsentationen. Aus diesem Grund wurde zur Ermittlung der Klas-

[1] Die Parameter für das unüberwachte Training auf T_Q und T_S finden sich in Tabelle 8.1.

sifikationsleistung das Anlernen des Klassifikators jeweils 20 mal mit unterschiedlichen zufälligen Präsentationsreihenfolgen wiederholt. Protokolliert wurde der minimale Fehler mit der zugehörigen Anzahl von Knoten (e_{min} und N_{min}) sowie der mittlere Fehler und die mittlere Knotenanzahl (e_{avg} und N_{avg}) zusammen mit ihren Standardabweichungen (σ_e und σ_N).

Das überwachte 2-Spiralen-Problem

Die Trainingsmenge für die überwachte Version des 2-Spiralen-Problems besteht aus 194 Ein-/Ausgabepaaren (x, t) mit $x \in \mathbb{R}^2$ und binärem Ausgabelabel $t \in \{0, 1\}$. Letzterer gibt an, welchem der offensichtlich nicht linear separablen Spiralarme die Eingabe x zugeordnet ist. Bei diesem Benchmark ist die Zeit zu protokollieren, die das Lernverfahren braucht, um eine 100%ige Klassifikation der Trainingsmenge zu erzielen [35]. Eine korrekte Klassifikation ist damit trivial über ein Tabel-Lookup-Verfahren zu erreichen, für ein MLP hingegen ist die Approximation der Entscheidungsgrenzen ein schwieriges Lernproblem. Das hier eingesetzte Lernverfahren[2] verwendet zwar lokal rezeptive Felder, doch werden die Zentren durch Verwendung der Interpolationsmethode nicht wie beim Tabel-Lookup unmittelbar auf die Eingabedaten gesetzt (keine Einfügung von Prototypen).

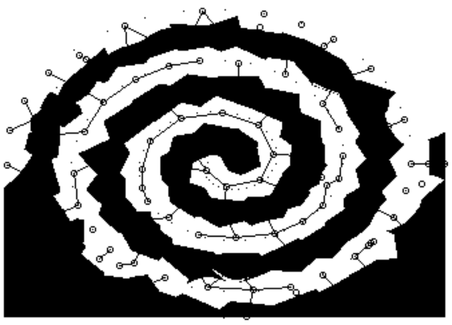


Abbildung 4.3: Überwachtes Lernen am Beispiel des 2-Spiralen-Benchmarks.

Abbildung 4.3 zeigt die Entscheidungsregionen nach 135 Epochen überwachten Trainings. Eingaben in schwarz gefärbten Gebieten werden dem ersten, Eingaben in weiß gefärbten Gebieten dem zweiten Spiralarm zugeordnet. Zusätzlich ist die Lage der Zentren und die Verbindungsstruktur erkennbar, soweit sie in den weiß gefärbten Bereich fällt. Der Klassifikationsfehler auf der Trainingsmenge beträgt 0%.

Trainingsmenge						**Testmenge**		
e_{min}	N_{min}	e_{avg}	N_{avg}	σ_e	σ_N	e_{min}	e_{avg}	σ_e
0%	135	0.36%	163	0.43%	14	0.7%	1.5%	0.7%

Tabelle 4.1: Klassifikationsergebnisse auf dem 2-Spiralen-Benchmark

[2]Die Parameter für das überwachte 2-Spiralen-Problem finden sich in Tabelle 8.2.

Tabelle 4.1 zeigt die Statistik der Trainingsverläufe für einen optimierten Satz von Lernparametern. Dabei fällt der Klassifikationsfehler auf der Trainingsmenge in 10 von 20 Läufen auf null, bevor eine Maximalzahl von 194 Knoten im Netz erreicht wird. Unter den restlichen Trainingsläufen werden maximal drei Trainingseingaben falsch klassifiziert. Die unterschiedlichen Klassifikationsergebnisse demonstrieren die Abhängigkeit von der Präsentationsreihenfolge. Ebenfalls in Tabelle 4.1 aufgeführt sind die Fehlerraten auf einer Testmenge von 962 Testdaten.

Ein Vergleich von Abbildung 4.3 mit 4.2 zeigt, daß beim überwachten Lernen der zwei Spiralen die Zentrendichte nicht mehr der Dichte der Trainingseingaben folgt. Durch die Orientierung am Klassifikationsfehler wird sie vielmehr in der Peripherie am größten, dort wo die Klassifikation aufgrund der geringeren Beispieldichte am schwierigsten ist. Diese geringe Dichte der Eingabedaten trägt auch die Verantwortung dafür, daß sich in der Peripherie im Gegensatz zu Abbildung 4.2 keine perfekt topologieerhaltende Karte ausbilden kann.

Tabelle 4.2 faßt die (minimale) Anzahl der benötigten Epochen zum Erlernen der korrekten Klassifikation der Trainingsdaten für DCS im Vergleich zu anderen bekannten KNN-Modellen zusammen.

Modell	# Epochen	Quelle
Backpropagation	20000	[74]
Cross Enropy BP	10000	[35]
Cascade-Correlation	1700	[35]
GCS	180	[37]
DCS	135	[29]

Tabelle 4.2: Modellvergleich für 2-Spiralen-Benchmark

Sprecherunabhängige Erkennung von Vokalen

Der Benchmark zur sprecherunabhängigen Erkennung von Vokalen umfaßt eine Trainingsmenge von 582 Beispielen und eine Testmenge von 462 Beispielen. Die Eingaben sind 10-dimensional, die Ausgaben 11-dimensional (1-aus-11 Kodierung der Klassenzugehörigkeit)[3]. Der Abbruch des Lernverfahrens erfolgt durch Kreuzvalidierung mit der Testmenge.

Testmenge						**Trainingsmenge**		
e_{min}	N_{min}	e_{avg}	N_{avg}	σ_e	σ_N	e_{min}	e_{avg}	σ_e
35%	108	40%	97	2%	32	0.5%	7%	5%

Tabelle 4.3: Klassifikationsergebnisse auf dem Vokalerkennungs-Benchmark

Für einen ausgewählten Satz von Lernparametern zeigt Tabelle 4.3 die Statistik über 20

[3] Die Details der Generierung des 10-dimensionalen Eingabe-Merkmalsvektors finden sich in [112].

Trainingsläufe[4]. Als bestes Klassifikationsergebnis wurde dabei ein Fehler von 35% mit 108 Knoten erzielt. Tabelle 4.4 faßt die von konkurrierenden Modellen erzielten Klassifikationsergebnisse zusammen. Der erste Teil dieser Tabelle findet sich in [112], wo Klassifikationsfehler nach ca. 3000 Trainingsepochen protokolliert wurden[5], der zweite Teil in [37], wo Spitzenklassifikationsergebnisse für 3- und 5-dimensionale GCS angegeben werden. Die Tabellen zeigen, daß das DCS-basierte Lernverfahren hier besser als die Vergleichsverfahren abschneidet und qualitativ ähnliche Resultate wie GCS erzielt (Spitzenklassifikationsleistung über 60%). Dabei bauen die DCS-Netze zusätzlich eine topologieerhaltende Karte der Eingabemannigfaltigkeit auf, während diese in [37] für die GCS-Netze(durch Ausprobieren) vorgegeben wurde.

Modell	# verdeckter Neuronen	% korrekt
Einschicht Perceptron		33
Mehrschicht Perceptron	88	51
Modifiziertes Kanerva Modell	528	50
RBF-Netz	528	53
Square-Node Netzwerk	88	55
Nearest Neighbor		56
3-dim GCS	158, 165, 154	61, 62, 67
5-dim GCS	135, 196	66, 66
DCS	108	65

Tabelle 4.4: Modellvergleich für Vokalerkennungs-Benchmark

Ultraschall-basierte Diskriminierung von Minen und Felsen

Der Benchmark zur Ultraschall-basierten Diskriminierung von Minen und Felsen basiert auf einem von Gorman und Sejnowski [42] benutzten Datensatz zur Diskriminierung von Ultraschallreflektionen von Metallzylindern und rauhen zylindrischen Felsen. Trainings- und Testdatensatz umfassen jeweils 104 Beispiele, wobei die Eingabevektoren 60-dimensional ($x \in [0,1]^{60}$) sind.

Testmenge						**Trainingsmenge**		
e_{min}	N_{min}	e_{avg}	N_{avg}	σ_e	σ_N	e_{min}	e_{avg}	σ_e
5%	88	8%	85	2%	12	0%	2%	3%

Tabelle 4.5: Klassifikationsergebnisse zu Benchmark “Ultraschall-basierte Diskriminierung von Minen und Felsen”

Gorman und Sejnowski erzielten mit einem Backprop-MLP Netz mit 12 verdeckten Knoten eine Spitzenklassifikationsleistung von 90.4% (Testmenge) und 87% für einen Nearest-Neighbor-Klassifikator. Das DCS-basierte Lernverfahren[6] hingegen erreicht eine Spitzen-

[4] Die Parameter zur sprecherunabhängigen Erkennung von Vokalen finden sich in Tabelle 8.3.
[5] Robinson berichtet von einer Spitzenklassifikationsleistung von ca. 54% für die meisten Modelle.
[6] Die Parameter zur Diskriminierung von Minen und Felsen finden sich in Tabelle 8.4.

klassifikationsleistung von 95% auf der Testmenge nach nur 88 Epochen Training. Tabelle 4.5 zeigt die Statistik über 20 Trainingsläufe.

Anmerkungen

Die Tests der DCS auf den CMU-Benchmarks [29] dienen dem direkten Vergleich mit Fritzkes GCS, und hier insbesondere der Klärung der Frage, ob das zusätzliche Erlernen der topologischen Struktur der Eingabedaten einen qualitativen Unterschied im Vergleich zur Vorauswahl einer möglichst optimalen Verbindungsstruktur bei GCS bewirkt. Dies ist nach den aufgeführten Ergebnissen zu verneinen. Zum Zwecke der Vergleichbarkeit unterscheidet sich das verwendete DCS-basierte Lernverfahren daher von dem in [37] publizierten Lernverfahren im wesentlichen nur durch den Austausch der vorgegebenen Verbindungsstrukturen aus k-dimensionalen Hypertetraedern mit selbstorganisierenden OTPMs. Kritisch ist zu diesem von Fritzke vorgeschlagenen Lernverfahren anzumerken, daß zur Adaption der Zentren eine unüberwachte, nur der Eingabedichte folgende Lernregel eingesetzt wird, die mit der überwachten Optimierung der Klassifikationsleistung in keinem direkten Zusammenhang steht. Eine fehlerabhängige Verteilung erfolgt allein durch den fehlergesteuerten Einfügeprozeß. Der Einsatz überwachter Zentrenadaptionsregeln wie der Gradienten-basierten (3.34) oder fehlermodulierten Lernregel (3.35) wäre hier von Vorteil gewesen.

Zu den CMU-Benchmarks selbst bleibt anzumerken, daß diese keine getrennten Validierungs- und Testmengen aufweisen. So wird die Testmenge sowohl zur Kreuzvalidierung (und damit zur Bestimmung der Modellordnung) als auch zum Test der Klassifikationsleistung genutzt. Dies ist zwar eine in der Praxis aufgrund geringer Datenmengen durchaus übliche Vorgehensweise, korrekt aber wäre Kreuzvalidierung und Test auf jeweils getrennten Validierungs- und Testmengen (siehe auch [100]).

4.3 Verifikation von Schlüsselpositionen in Gesichtsbildern

In diesem Abschnitt wird die Anwendung eines DCS-Netzes in einem hybriden Klassifikationsverfahren beschrieben, in dem charakteristische Punkte in Gesichtsbildern zuerst in einer modellbasierten Stufe detektiert und dann in einer zweiten, DCS-basierten Stufe verifiziert werden. Diese Anwendung entstand in Zusammenarbeit mit der GSF-MedIS, wobei der Autor dieser Arbeit sowohl die Grundlagen des verwendeten DCS-Lernverfahrens als auch die Kombination der Detektions- und Verifikationsstufe im Rahmen der Bayes-Theorie beisteuerte.

Im Rahmen der Klassifikationsaufgabe werden zunächst Gesichtsausschnitte durch eine attentive Regionensuche extrahiert [55] und anschließend modellgestützt nach markanten Punkten untersucht. Die Detektion der charakteristischen Punkte (Schlüsselpositionen), beispielsweise der Augenecken, basiert dabei auf einer sequentiellen Auswertung der Kanten- und Linieninformation. Der Kantenverfolgung liegt dabei ein Modell der untersuchten Bildregion zugrunde. An die Detektion schließt sich die Verifikation der gefundenen

Punkte an. Dazu wird aus der lokalen Bildinformation ein Merkmalsvektor generiert und durch ein DCS-Netzwerk klassifiziert. Ordnet das DCS-Netzwerk den Merkmalsvektor des gefundenen Bildpunktes der Klasse des gesuchten charakteristischen Punktes zu, so gilt der Punkt als verifiziert.

Die Grundlage der Merkmalsextraktion für die Detektion als auch für die Verifikation bildet ein steuerbares Filterschema, welches Kanten- und Linienfilter einer hohen Orientierungsselektivität in beliebigen Skalen und Orientierungen auf das Bild anwendet. Dazu wird nur ein kleiner Satz sogenannter Basisfunktionen mit dem Bild gefaltet, von dem alle benötigten Merkmale während der Suche der Punkte bedarfsgesteuert abgeleitet werden.

Das steuerbare Filterschema, welches die Grundlage sowohl für die Detektion als auch für die anschließende Verifikation der markanten Bildpunkte darstellt, findet sich in [57] und basiert auf der Arbeit von Michaelis [84]. Ebenfalls in [57] sind die Detektion der markanten Punkte durch die modellgesteuerte Kantenverfolgung am Beispiel der Augenregion und die Details des verwendeten DCS-Lernverfahrens dargestellt, wobei letzteres sich an dem in Abschnitt 4.2.1 vorgestellten überwachten Lernverfahren orientiert. Im folgenden wird der Schwerpunkt auf das Training des zur Verifikation genutzten DCS-Netzwerkes und die Verknüpfung mit der Detektionsstufe über den Satz von Bayes gelegt.

4.3.1 Die Trainingsmenge

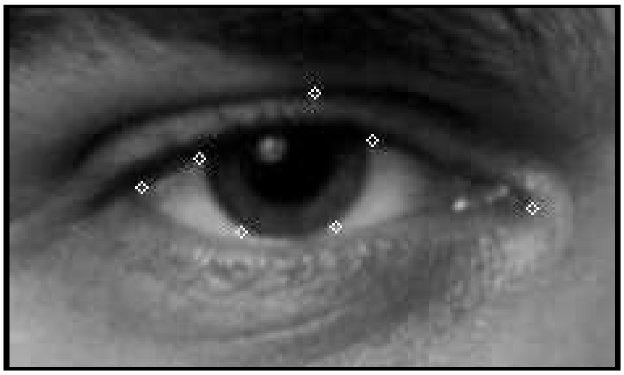


Abbildung 4.4: Gesichtsausschnitt eines linken Auges mit den charakteristischen Bildpunkten "innerer und äußerer Augenwinkel", "obere und untere Schnittpunkte des Irisrandes mit den Lidern" sowie ein Punkt auf der oberen Lidfalte.

Es werden 9 Klassen charakteristischer Bildpunkte unterschieden (Abbildung 4.4): Innerer und äußerer Augenwinkel (jeweils für das rechte und linke Auge getrennt), die 4 Schnittpunkte des Irisrandes mit den Lidern und Punkte auf der Oberlidfalte[7]. Die meisten Pixel in einer Augenregion sind damit keine charakteristischen Punkte. Um zu verhindern, daß das DCS-Netzwerk Berechnungsknoten zur Repräsentation dieser Punkte verschwendet,

[7] Aufgrund der fehlenden Lokalisierbarkeit wird die Klasse der Punkte auf der Oberlidfalte von der weiteren Betrachtung ausgeschlossen.

wird die Restklasse nicht explizit sondern implizit durch einen klassenspezifischen Schwellwert definiert: Liegt die Aktivierung der bmu_x für eine Bildposition bei der Klassifikation unterhalb des Schwellwertes, wird sie der Restklasse zugeordnet. Für das Training des DCS-Netzwerkes werden nur Merkmalsvektoren von vorselektierten charakteristischen Bildpunkten verwendet, vgl. [57].

Als 70-dimensionaler Merkmalsvektor für das Training des DCS-Netzwerkes und auch für die Klassifikation eines unbekannten Bildpunktes dienen 70 Projektionskoeffizienten einer 27×27 Pixel Bildregion auf die Basisfunktionen des steuerbaren Filterschemas [57]. In diesen Projektionskoeffizienten ist die wesentliche Kanten- und Linieninformation der lokalen Bildstruktur enthalten. Da die Koeffizienten bereits zur Detektion der Punkte berechnet wurden, entsteht kein zusätzlicher Berechnungsaufwand. Die Verwendung der Projektionskoeffizienten anstelle der direkten Bildinformation (originale Grauwertverteilung in der 27×27 Umgebung) ermöglicht dabei eine erhebliche Dimensionsreduzierung des Merkmalraumes, ohne auf die wesentliche Information der lokalen Bildstruktur verzichten zu müssen.

668 Merkmalvektoren wurden aus insgesamt 110 Augenregionen von rechten und linken Augen abgeleitet, wobei 87 Augenregionen für die Trainingsmenge und 23 für die Testmenge herangezogen wurden. Neben den exakten Bildpositionen der markanten Bildpunkte als Aufpunkte wurden zusätzlich noch Punkte in der direkten Nachbarschaft (3 x 3) der markanten Bildpunkte zufällig ausgewählt und zur Generierung von Merkmalvektoren verwendet, wodurch die Generalisierungsfähigkeit des Netzwerkes gesteigert wird.

Netzwerkgröße						
Anzahl Knoten	23	32	59	87	115	127
Trainingsmenge						
Fehler (%)	9.7	8.2	5.5	3.4	1.5	0
Testmenge						
Fehler (%)	8.5	7.8	7.0	6.3	5.6	4.9

Tabelle 4.6: Klassifikationsfehler in Abhängigkeit von der Größe des DCS-Netzwerkes.

Der Klassifikationsfehler des Netzwerkes (gemittelt über alle Klassen) bezüglich der Trainingsmenge und der Testmenge in Abhängigkeit von der Anzahl der während des Trainings eingefügten Berechnungsknoten ist in Tabelle 4.6 dargestellt. Das Ende des Wachstumsprozesses wird durch Kreuzvalidierung bei 127 Knoten und einem Klassifikationsfehler von 4,9% (Testmenge) erreicht.

4.3.2 Der Bayessche Verifikationsansatz

Um die in dieser Anwendung besonders wichtige falsch-positive Fehlklassifikationswahrscheinlichkeit der sequentiellen Suche zu reduzieren, wird der oben beschriebene DCS-basierte Klassifikator zur Verifikation der Bildposition, an der die sequentielle Suche abbricht, genutzt.

Die theoretische Basis der Kombination aus DCS-basiertem Klassifikator und modellgestützter Detektionsstufe liefert der Satz von Bayes. Modellbasierte Detektion und neuronale Verifikation eines markanten Punktes einer Klasse C_i werden als unabhängige Zufallsexperimente betrachtet, wobei die Klasse des Punktes im Falle der sequentiellen Suche der Zufallsvariablen X und im Falle der Verifikation der Zufallsvariablen Y zugewiesen wird. Die Wahrscheinlichkeit, daß die Bildposition, an der die sequentielle Suche terminiert, der gesuchten Klasse C_i angehört, ist die a priori Wahrscheinlichkeit $P(X = C_i)$. Die a priori Fehlerwahrscheinlichkeit, daß das sequentielle Suchschema einen Punkt detektiert, der nicht zur Klasse C_i gehört, ist $P(X \neq C_i)$. Wenn Y die Klasse bezeichnet, die der betrachteten Bildposition durch die Verifikation zugeordnet wird, dann ist $P(Y = C_i|X \neq C_i)$ die bedingte Wahrscheinlichkeit für eine falsch-positive Klassifikation und $P(Y \neq C_i|X = C_i)$ die bedingte Wahrscheinlichkeit für eine falsch-negative Klassifikation. Nach dem Satz von Bayes läßt sich nun die a posteriori falsch-positive Fehlerwahrscheinlichkeit, daß eine detektierte Bildposition trotz Verifikation nicht der gesuchten Klasse angehört, zu

$$P(X \neq C_i|Y = C_i) = \frac{P(Y = C_i|X \neq C_i)\ P(X \neq C_i)}{P(Y = C_i)} \tag{4.1}$$

$$= \frac{P(Y = C_i|X \neq C_i)\ P(X \neq C_i)}{P(Y = C_i|X \neq C_i)\ P(X \neq C_i) + P(Y = C_i|X = C_i)\ P(X = C_i)}$$

berechnen. Offensichtlich ist eine Verifikation nur dann sinnvoll, wenn die a-posteriori-Fehlerwahrscheinlichkeit $P(X \neq C_i|Y = C_i)$ kleiner als die a-priori-Fehlerwahrscheinlichkeit $P(X \neq C_i)$ der Detektionsstufe ist.

char. Bildpos.	$\hat{P}(X \neq C_i)$	$\hat{P}(Y = C_i\|X \neq C_i)$	$\hat{P}(Y \neq C_i\|X = C_i)$	$\hat{P}(X \neq C_i\|Y = C_i)$
i. Augenecke	14.4	6.7	2.2	1.1
a. Augenecke	8.3	11.1	1.0	1.0
o. Schnittp. Iris/Lid	3.2	14.3	0.9	0.5
u. Schnittp. Iris/Lid	4.1	0.0	0.9	0.0
gesamt	6.5	8.3	1.2	0.2

Tabelle 4.7: Empirische Fehlerwahrscheinlichkeiten in %

Tabelle 4.7 zeigt, daß die falsch-positive Fehlklassifikationsrate durch die zusätzliche Anwendung der Verifikationsstufe tatsächlich erheblich verkleinert werden kann. Sie sinkt von insgesamt 6.5% auf 0.2 %.

4.4 Online-Stabbalancieren

Während in den vorangegangenen Beispielen fest vorgegebene Trainingsmengen zum überwachten Lernen von Klassifikatoren verwendet wurden, wird in diesem Abschnitt eine Regelungsfunktion anhand von online generierten Trainingsbeispielen gelernt, die aus der Beobachtung eines dynamischen Systems und eines existierenden Reglers für dieses System gewonnen werden [3].

Als dynamisches System dient der von Ritter [109] zum Training einer erweiterten Kohonenkarte verwendete vereinfachte Stabbalancierer. Dieser besteht aus einem Stab, der an seinem Fußpunkt auf einem in eine Richtung frei beweglichen Wagen befestigt ist. Durch Anlegen einer Kraft f am Fußpunkt des Stabes soll erreicht werden, daß der Stab aufrecht balanciert wird (siehe Abbildung 6.5 auf Seite 118). Als Eingabegrößen für den Regler werden in dieser vereinfachten Version nur der Stabwinkel ϕ und die Winkelgeschwindigkeit $\dot{\phi}$ genutzt. Der Stab hat die Länge 1 m und die Masse 1 kg, die als punktförmig an der Spitze des Stabes angenommen wird. Der Wagen, auf dem der Stab befestigt ist, besitzt die Masse $m_{cart} = 1.0\ kg$. Als Gravitationskonstante wird $g = 1\ \frac{\mathrm{m}}{\mathrm{s}^2}$ verwendet. Unter diesen Annahmen vereinfachen sich die Gleichungen (6.31) und (6.32), so daß sich die Dynamik des Systems durch eine einzige Differentialgleichung

$$(m_{cart} + \sin^2\phi)\ddot{\phi} + \frac{1}{2}\dot{\phi}^2 \sin(2\phi) - (m_{cart} + 1)\sin\phi = -f\cos\phi \tag{4.2}$$

beschreiben läßt. Die Simulation der Dynamik des Systems erfolgt mittels Eulerintegration und einer Zeitschrittweite von $\Delta t = 0.3s$.

Als Sollausgabe zum Training des DCS-Netzwerkes dient die Funktion $f^L(\phi, \dot{\phi}) = 5\sin\phi + \dot{\phi}$. In der Simulation wird der Stab zufällig mit einem Startwinkel $-\frac{\pi}{4} \leq \phi \leq \frac{\pi}{4}$ aus der Ruhelage losgelassen. Die Kraft, die auf das Pendel angewendet wird, ergibt sich aus

$$f = \beta(t_k)y^{DCS}(\phi, \dot{\phi}) + (1 - \beta(t_k))f^L(\phi, \dot{\phi}), \tag{4.3}$$

wobei $\beta : [0, T] \rightarrow [0, 1]$, $\beta(t) = \frac{t_k}{T}$ eine monoton steigende Funktion ist. Mit zunehmender Zeit t_k tritt so an die Stelle der vorgegebenen Sollkraft die durch das Netzwerk berechnete Kraft. Nach einem Fehlversuch ($|\phi| > \frac{\pi}{4}$) oder nach $k_{try} = 100$ erfolgreichen Schritten wird die Simulation angehalten und der Stabbalancierer neu gestartet.

Das Training des DCS-Netzwerkes basiert auf dem überwachten Online-Lernverfahren[8] aus Abschnitt 4.2.1, wobei hier jedoch nicht die einfache Fritzke-, sondern die fehlermodulierte Lernregel (3.35) zur Adaption der Zentren eingesetzt wird, und zur Adaption der Fehlervariablen die gleitende Mittelwertbildung nach (3.47). Hierdurch wird eine Abhängigkeit der Zentrenadaption von der Wahrscheinlichkeitsdichte der Eingaben vermieden, vgl. Seite 51. Das Einfügen neuer Knoten und damit das Wachstum des Netzes erfolgt nach einer gemischten fehler- und distanzgetriebenen Strategie, bis eine maximale Zahl von $N_{max} = 70$ Knoten im DCS erreicht ist. Durch die distanzabhängigen Einfügungen erfolgt im Anfangsstadium des Trainings eine grobe Approximation der Regelfunktion, die im weiteren Verlauf durch fehlergetriebene Einfügungen sukzessive verfeinert wird. Letztere erfolgen nach jeweils $\lambda_{ins} = 70$ Schritten (vgl. Seite 58), die Initialisierung neuer Knoten erfolgt nach der Interpolationsmethode (3.51). Die Adaption der Topologie schließlich erfolgt nach der unsymmetrischen Hebb-Regel (3.41).

Nach 3000 Lernschritten ist das Netzwerk mit 70 Knoten in der Lage, den Stab zu balancieren. Die Verteilung der Knoten im Eingaberaum bzw. Ein- und Ausgaberaum ist in Abbildung 4.5, a) und b), dargestellt, die erlernte Regelungsfunktion in Abbildung 4.5 c) und das Regelungsverhalten in Abbildung 4.5 d). Im Vergleich zu dem analogen Experiment in

[8] Die Parameter zum Online Stabbalancieren finden sich in Tabelle 8.5.

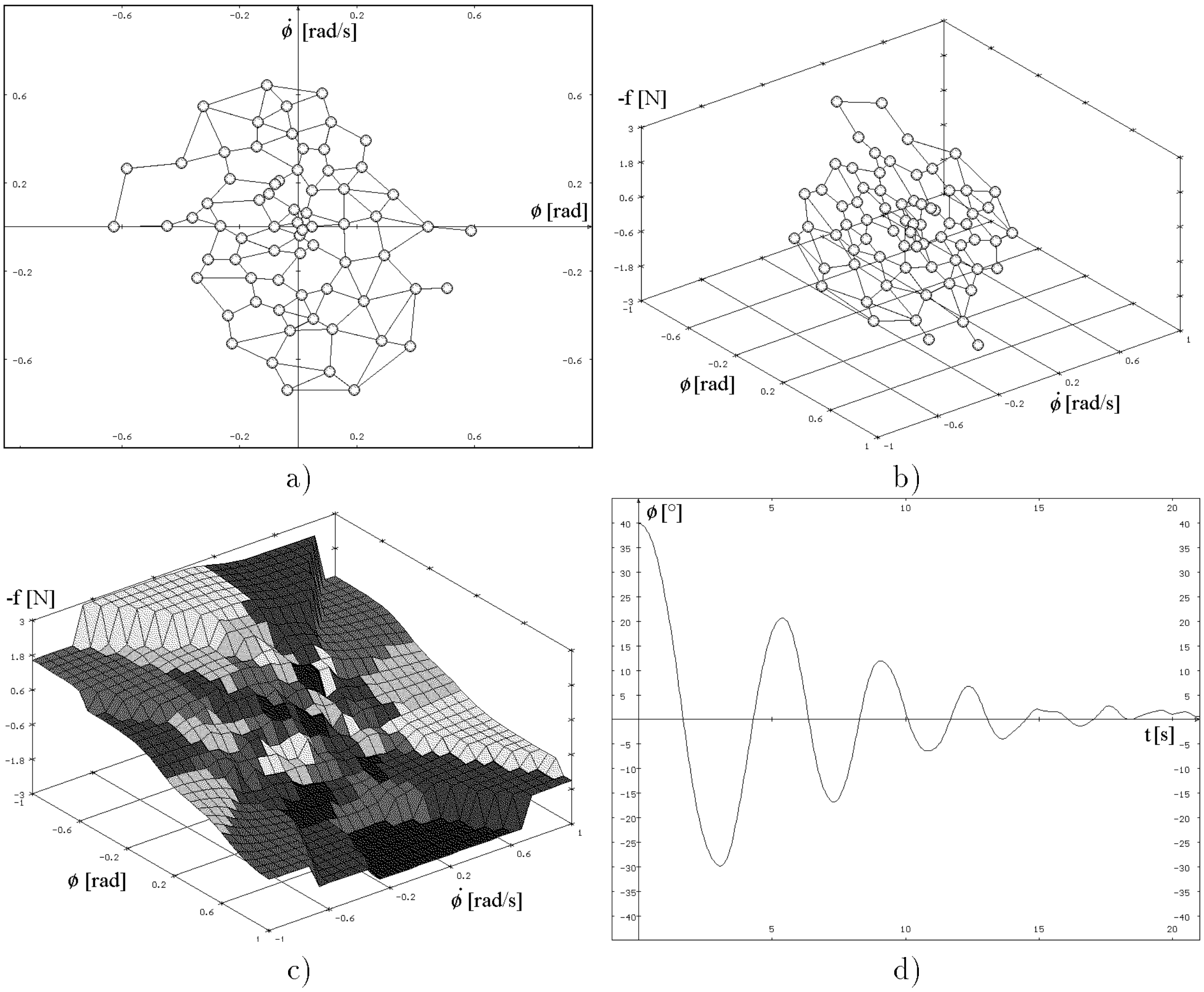


Abbildung 4.5: In a) und b) ist die Verteilung der Neuronen im Eingaberaum bzw. Ein- und Ausgaberaum abgebildet; c) zeigt die approximierte Regelungsfunktion, die die negative Kraft zur Kontrolle des Stabes angibt. Die verschiedenen Grautöne kennzeichnen unterschiedliche Voronoizellen einzelner Neuronen. In d) ist schließlich eine Schwingung $\phi(t)$ des Pendels während der Balance durch den trainierten Regler dargestellt.

[109] zum Training der erweiterten Kohonenkarte ist bemerkenswert, daß das Lernverfahren ohne “Einfrieren” von Lernparametern auskommt - das Stabilität/Plastizität-Dilemma wird über die fehlermodulierte Lernregel gelöst. Hingegen ist das Einfrieren der Lernkonstanten bei der Kohonenregel unerläßlich, da sich sonst alle Knoten um den Ursprung des $(\phi, \dot{\phi})$ Eingaberaumes konzentrieren würden, wo mit Abstand die höchste Eingabedichte herrscht. Auch kommt das DCS-Netz mit nur 70 Knoten im Vergleich zu den 625 Knoten der erweiterten Kohonenkarte in [109] aus.

Kapitel 5

Fehlerrückkopplungs-Lernen mit DCS

In diesem Kapitel steht das Training der DCS mittels Fehlerrückkopplungs-Lernen im Mittelpunkt, das hier anhand zweier Anwendungen aus der Robotik näher betrachtet wird. Beim Fehlerrückkopplungs-Lernen erhält der Lerner ein Fehlersignal, das proportional zum tatsächlichen Fehler ist. Dieses Signal hat damit eine höhere Qualität als das bloße Bewertungssignal des Reinforcement-Lernens (Kapitel 6), denn es gibt die Richtung an, in die die Ausgabe zu adaptieren ist. Es enthält aber weniger Information als das Fehlersignal beim überwachten Lernen, welches direkt die gewünschte Ausgabe darstellt. Nimmt man die Qualität des Fehlersignals als Maßstab, so ist das Fehlerrückkopplungs-Lernen zwischen Reinforcement-Lernen und überwachtem Lernen einzuordnen.

Fehlerrückkopplungs-Lernen ist ein reines Online-Lernverfahren mit typischem Anwendungspotential in der Robotik, speziell hinsichtlich des Erlernens der inversen Kinematik bzw. Dynamik. Das Erlernen der inversen Kinematik ist auch Gegenstand der beiden hier vorgestellten Applikationen, der kalibrierungsfreien adaptiven Sakkadensteuerung eines Stereo-Kamerakopfes und der kalibrierungsfreien adaptiven Hand-Auge-Koordination eines Roboterarmes und Stereokamerasystems. Während im Fall der Sakkaden-Steuerung ein einfacher Proportionalregler zur Generierung des Fehlersignals ausreicht, muß im zweiten Fall innerhalb eines hierarchischen Regelkreises der Regler selbst adaptiert werden. Es gibt starke Hinweise, daß Fehlerrückkopplungs-Lernen auch in biologischen Systemen eine entscheidende Rolle spielt [68, 59].

Wachsende Dynamische Zellstrukturen lassen sich besonders vorteilhaft mit Fehlerrückkopplungs-Lernen kombinieren. Am Beispiel der lernenden Sakkaden-Steuerung wird gezeigt, wie das Fehlersignal direkt für ein fehlergesteuertes Wachstum der Netze, das bei Erreichen einer geforderten Präzison abbricht, genutzt werden kann. Die resultierenden Netze sind dann verhältnismäßig klein und lassen eine schnelle Ausgabeberechnung zu, die in unseren Anwendungen Echtzeitanforderungen genügt. Gerade in physikalischen Systemen liegen die Zustandsgrößen aufgrund von Zwangsbedingungen oft in Untermannigfaltigkeiten, die von weit kleinerer intrinsischer Dimensionalität als der Eingaberaum selbst sind. Die Hand-Auge-Koordination (Abschnitt 5.3) bietet hier ein besonders anschauliches Beispiel. Dynamische Zellstrukturen sind durch die Ausbildung von OTPMs in der Lage, sich diesen Untermannigfaltigkeiten optimal anzupassen.

Teile dieses Kapitels wurden in [25, 27, 26] (Sakkadensteuerung) sowie [20] (Hand-Auge-Koordination) publiziert. In [26] findet sich auch eine kurze Diskussion der biologischen Plausibilität der in Abschnitt 5.2 vorgestellten DCS-basierten Sakkadensteuerung.

5.1 Fehlerrückkopplungs-Lernen

Fehlerrückkopplungs-Lernen (Feedback-Error Learning) ist ein von Kawato et. al. [68], [59] in Anlehnung an neurobiologische Erkenntnisse eingeführtes Lernschema zur inversen Modellierung einer Regelstrecke durch ein KNN mittels eines dem Netz parallel geschalteten zweiten Reglers. Das auf der Basis der rückgekoppelten Regeldifferenz zwischen Soll- und Istwert, $e_y = y^{soll} - y^{ist}$, durch den Regler erzeugte Stellsignal u_{fb} dient als direktes Fehlermaß für die Adaption des KNN. Der zweite Regler ist üblicherweise ein konventioneller PID Regler, der nur grob justiert sein muß. Das Diagramm eines adaptiven Regelkreises nach dem Fehlerrückkopplungsprinzip ist in Abbildung 5.1 gezeigt. Die Lernregel für die Parameter w des KNN lautet

$$\Delta w = \alpha \, u_{fb} \, \frac{\partial u_{NN}}{\partial w}. \tag{5.1}$$

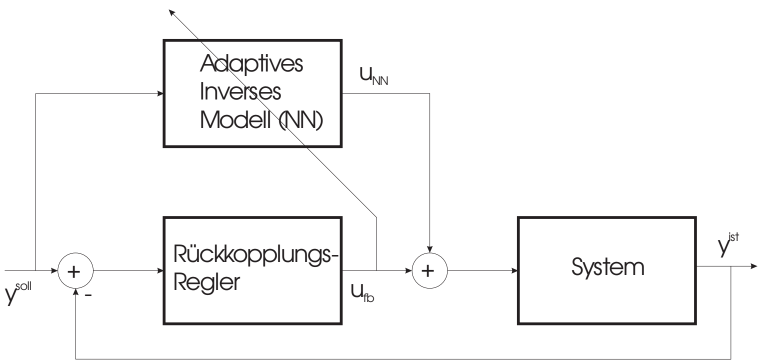


Abbildung 5.1: Diagramm eines adaptiven Regelkreises nach dem Fehlerrückkopplungsprinzip.

Neben starken Hinweisen, daß Fehlerrückkopplungs-Lernen von Primaten zum Erlernen der Oculomotorik und anderen motorischen Aufgaben genutzt wird [32, 67], konnten Kawato et. al. das Potential dieser Lernregel erfolgreich an mehreren Anwendungen aus der Robotik, insbesondere der Bahnreglung industrieller Roboterarme [59, 65] demonstrieren. Vorteilhafte Eigenschaften des Fehlerrückkopplungs-Lernens sind seine *Zielgerichtetheit* (*goal directedness*) und seine *Einfachheit*. *Zielgerichtetheit* bedeutet in diesem Zusammenhang, daß das Lernverfahren direkt die Regeldifferenz minimiert und im Gegensatz z.B. zur *direkten inversen Modellierung* [63] nicht an dem Problem krankt, daß das globale Optimum bzgl. der eingesetzten Fehlerfunktion nicht notwendigerweise mit einem gültigen inversen Modell korrespondiert. Hat das Netz das inverse Modell erlernt, so wird die Regeldifferenz und damit das Fehlersignal zur Adaption null. *Einfachheit* bedeutet insbesondere den Verzicht auf ein zweites KNN in der Regelschleife, wie z.B. bei der ebenfalls zielgerichteten *indirekten adaptiven Regelung* [63]. Allerdings wird zum Erlernen komplexer motorischer Aufgaben ein einfacher PID-Regler nicht mehr ausreichen, und die Schwierigkeit verlagert sich in den Entwurf eines adäquaten Rückkopplungsreglers. Am Beispiel der Hand-Auge-Koordination wird in Abschnitt 5.3 demonstriert, wie der Rückkopplungsregler selbst adaptiv gestaltet und gelernt werden kann.

Während Konvergenzbeweise für das Fehlerrückkopplungs-Lernen sich i. allg. (allgemeine Strecke, allgemeiner Regler) schwierig gestalten (siehe hierzu [66]), soll im folgenden in Anlehnung an [96] gezeigt werden, daß Fehlerrückkopplungs-Lernen für einen P-Regler mit Verstärkungsmatrix K_p ($u_{fb} = K_p\, e_y$) in Zusammenhang mit einer Strecke der Form $y^{ist} = S_u\, u$ mit Übertragungsmatrix S_u und Stellsignal u einen Gradientenabstieg auf der Fehlerfunktion

$$E_{K_p} = \frac{1}{2}(K_p e_y)^2 \tag{5.2}$$

darstellt. Nach Abbildung 5.1 gilt

$$y^{ist} = S_u[K_p(y^{soll} - y^{ist}) + u_{NN}] \tag{5.3}$$

womit sich die Ableitung des Istwertes nach den Parametern des Netzwerkes zu

$$\frac{\partial y^{ist}}{\partial w} = S_u[-K_p\frac{\partial y^{ist}}{\partial w} + \frac{\partial u_{NN}}{\partial w}] = \frac{S_u\frac{\partial u_{NN}}{\partial w}}{1 + S_u K_p} \tag{5.4}$$

berechnet. Der Gradient der Kostenfunktion E_{K_p} nach den Parametern des Netzwerkes ist

$$\frac{\partial E_{K_p}}{\partial w} = \frac{\partial E_{K_p}}{\partial y^{ist}}\frac{\partial y^{ist}}{\partial w}\ . \tag{5.5}$$

Gilt nun $K_p \cong S_u^{-1}$, daß also die Verstärkungsmatrix der inversen Übertragungsmatrix entspricht, so erhält man mit (5.4)

$$\frac{\partial E_{K_p}}{\partial w} = -K_p\frac{e_y}{2}\frac{\partial u_{NN}}{\partial w} = u_{fb}\frac{\partial u_{NN}}{\partial w}\ . \tag{5.6}$$

Man sieht an (5.1) und (5.6), daß das Fehlerrückkopplungssignal u_{fb} dieselbe Rolle für den Gradientenabstieg auf E_{K_p} spielt wie der Ausgabefehler eines Netzwerkes beim überwachten Training durch Gradientenabstieg. Beide Lernverfahren, überwachtes Training und Fehlerrückkopplungs-Lernen durch Gradientenabstieg, haben damit dieselbe Form und sind durch einfaches "Umschalten" der Fehlersignale leicht kombinierbar.

Natürlich wird die Verstärkungsmatrix nur selten exakt der inversen Übertragungsmatrix entsprechen. Wichtig für eine Minimierung von E_{K_p} ist aber nur die Forderung

$$\langle\frac{\partial E_{K_p}}{\partial w}, \frac{\partial u_{NN}}{\partial w}\rangle > 0\ ,$$

die auch von grob angepaßten Reglern erfüllt werden kann.

In den Anwendungen zur Sakkaden-Steuerung und Hand-Auge-Koordination kommt ein leicht modifiziertes Fehlerrückkopplungs-Lernen zur Anwendung. Hier wird das Stellsignal des P-Reglers nicht additiv überlagert, sondern es erfolgt in einem ersten Schritt eine Steuerung des Systems allein durch das Stellsignal des Netzes u_{NN} und in den darauffolgenden Schritten eine Regelung (Feinkorrektur) allein durch das Stellsignal u_{fb} des P-Reglers, das wie im normalen Fehlerrückkopplungs-Lernen gemäß (5.1) zur Adaption des Netzes genutzt wird. Auch diese Adaption kann als Gradientenabstieg auf E_{K_p} gedeutet werden:

Die Ausgabe des Systems aufgrund der Steuerung durch das Netz (unter denselben Voraussetzungen wie oben) ergibt sich zu:

$$y^{ist} = S_u u_{NN}. \tag{5.7}$$

Damit folgt für die Ableitung des Istwertes nach den Parametern des Netzwerkes

$$\frac{\partial y^{ist}}{\partial w} = S_u \frac{\partial u_{NN}}{\partial w}. \tag{5.8}$$

Gilt nun wiederum $K_p = S_u^{-1}$, so erhält man mit (5.8) die schon bekannte Gleichung

$$\frac{\partial E_{K_p}}{\partial w} = -K_p \frac{e_y}{2} \frac{\partial u_{NN}}{\partial w} \ . \tag{5.9}$$

"Normales" und modifiziertes Fehlerrückkopplungs-Lernen minimieren damit dieselbe Fehlerfunktion und benutzen die gleiche Adaptionsregel.

5.2 Kalibrierungsfreie Sakkaden-Steuerung

Sakkaden sind schnelle, ruckartige Änderungen der Blickrichtung, die dafür sorgen, daß die Fovea auf verschiedene Orte im Blickfeld gerichtet wird. Während normalerweise Sakkaden nur die Bewegung der Augen bezeichnen, wird im folgenden der Begriff *Sakkade* für die kombinierte Bewegung der Augen und des Kopfes benutzt. Da bei der Sakkadensteuerung die Betonung auf der Schnelligkeit liegt, scheidet eine reine Rückkopplungsregelung sowohl beim Menschen als auch bei künstlichen Sehsystemen wegen der mit Rückkopplungsschleifen verbundenen längeren Verzögerungszeiten und Einschwingeffekte aus. So werden beim Menschen Sakkadengeschwindigkeiten von bis zu $1000°/sec$ gemessen, während die Verzögerung visuellen Feedbacks ca. 200 ms beträgt. Dies ist für eine Regelung über Feedback zu schnell [31] und scheint ohne Lösung der inversen Kinematik nicht erklärbar [82].

Mayhew et al. stellten in [82, 32] die Hypothese auf, daß die Sacccadensteuerung bei Primaten durch Fehlerrückkopplungs-Lernen erlernt wird, d.h. ein Fehlersignal proportional zur Abweichung des Bildpunktes auf der Retina vom Mittelpunkt der Fovea zum Training des biologischen Sehsystems genutzt wird. Diese Idee auf künstliche Sehsysteme übertragend, studierten Mayhew et al. [82] systematisch die Eignung verschiedener konventioneller KNN-Architekturen zur Sakkaden-Steuerung eines simulierten Kamerakopfes.

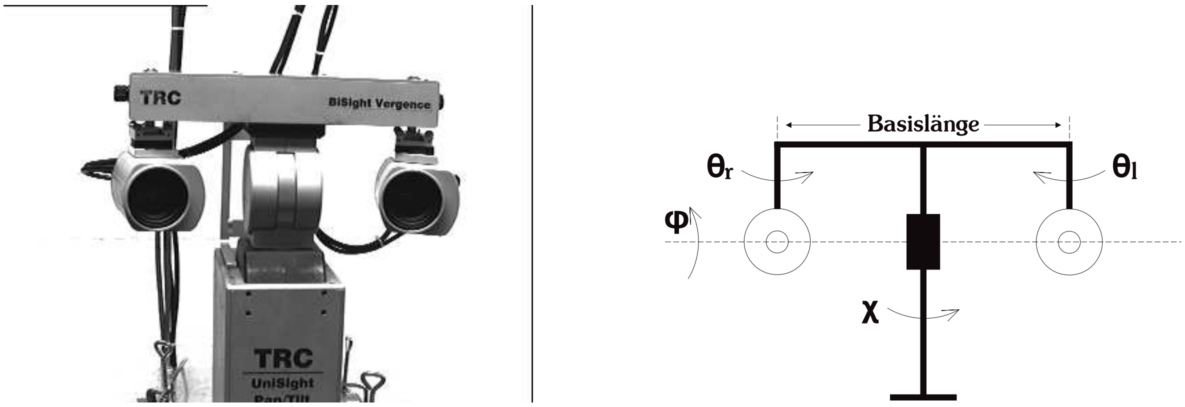
Durch diese Vorarbeiten inspiriert, wird im folgenden ein ähnlicher Ansatz zur adaptiven Sakkaden-Steuerung des (realen) TRC[1] Bisight Kamerasystems vorgestellt, allerdings auf Basis eines DCS-Netzes [26, 107]. Unter Verwendung eines lokal linearen Approximationssschemas weist dieser Ansatz eine Reihe von vorteilhaften Eigenschaften auf: Erstens ergeben sich durch fehlergesteuertes Wachstum des Netzes bis zu einer vorgegebenen Sakkadengenauigkeit an diese angepaßte relativ kleine Netze, die für eine Sakkaden-Steuerung auch in Videoechtzeit geeignet sind. Zweitens stellt sich heraus, daß die zur Fixierung notwendigen Drehungen um die Schwenk-, Neigungs- und Vergenzachsen (vgl. Abbildung 5.2) nur für kleine initiale Winkel und ein eingeschränktes Blickfeld (Gültigkeit der linearen Approximation kleiner Winkel) in guter Näherung in linearem Zusammenhang mit den retinalen Koordinaten des Zielpunktes stehen. Mit zunehmendem initialen Neigungswinkel liegen die optischen Achsen der Kameras und die Schwenkachse nicht mehr rechtwinklig zueinander, so daß der funktionale Zusammenhang zunehmend nichtlinear wird. Um dennoch einen gleichförmigen Approximationsfehler zu erzielen, muß ein lokal lineares Approximationsschema eine höhere Stützstellendichte in den Regionen stärkerer Nichtlinearität aufweisen. Im Gegensatz zu den allein von der Wahrscheinlichkeitsdichte der Eingaben abhängigen erweiterten Kohonenkarten wird demonstriert, daß DCS-Netze zur Generierung dieser problemangepaßten Stützstellenverteilung in der Lage sind. Als letzten Bonus besitzt die DCS die Eigenschaft, sich der Topologie der Eingabemannigfaltigkeit durch Ausbildung von OTPMs anzupassen.

[1] Transition Research Company, Danburry, Connecticut USA

5.2.1 Der TRC-Bisight Stereo-Kamerakopf

In diesem Abschnitt wird das binokulare Kamerasystem TRC-Bisight vorgestellt und durch ein geometrisches Modell beschrieben, welches die direkte Berechnung der inversen Kinematik erlaubt. Aus der Analyse dieses Modells lassen sich nützliche Rückschlüsse auf die zu realisierende modell- und kalibrierungsfreie adaptive Steuerung ziehen. Neben dem binokularen wird auch der einfacher zu handhabende monokulare Fall erörtert.

Das Kamerasystem (siehe Abbildung 5.2) verfügt über vier mechanische Freiheitsgrade, beschrieben durch den *Schwenkwinkel* χ um den Hals des Systems, den *Neigungswinkel* ϕ und die *Vergenzwinkel* [2] θ_l und θ_r. Die maximalen Stellgrößen betragen für den Schwenkwinkel $\pm 160°$, für den Neigungswinkel $\pm 90°$ und für die Vergenzwinkel $\pm 45°$. Auf der Neigungsachse sind die beiden CCD-Kameras montiert. Die Stereobasis wird mit b notiert und beträgt 25cm. Jede Kamera verfügt über drei optische Freiheitsgrade, den Zoom (11.5 - 69 mm), den Fokus (50cm - ∞) und die Blende. Die Genauigkeit der Winkelansteuerung liegt bei $\pm 0.0225°$ für den Schwenk- und den Neigungswinkel und bei $\pm 0.006°$ für die Vergenzwinkel.



(a) (b)

Abbildung 5.2: (a) Der TRC-Bisight Stereokamerakopf, (b) Diagramm des TRC-Bisights: Die Drehung um die Schwenkachse wird durch χ, um die Neigungsachse durch Φ und um die Vergenzachsen durch θ_l und θ_r beschrieben.

Geometrie des Kamerakopfes

Im Falle des TRC-Bisights wird die Abbildung eines Punktes $p_w = (x_w, y_w, z_w, 1)$ in homogenen Weltkoordinaten auf einen Punkt $p_c = (x_c, y_c, z_c, 1)$ in Kamerakoordinaten durch

$$p_{c_{l/r}} = T^{-1}_{\theta_{l/r}} T^{-1}_{\phi} T^{-1}_{\chi} p_w \tag{5.10}$$

[2] Die Terminologie weicht hier von der Standardterminologie ab, in der die Größe $180° - \theta_l - \theta_r$ als Vergenzwinkel bezeichnet wird.

mit den Transformationsmatrizen

$$T_\chi = \begin{pmatrix} \cos\chi & 0 & -\sin\chi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\chi & 0 & \cos\chi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; T_\phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \; T_{\theta_l} = \begin{pmatrix} \cos\theta_l & 0 & \sin\theta_l & -b/2 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_l & 0 & \cos\theta_l & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

beschrieben (T_{θ_r} entspricht T_{θ_l} bis auf die Vorzeichen der Translation und Rotation). Der Ursprung des Weltkoordinatensystems liegt dabei im Schnittpunkt von Schwenk- und Neigungsachse, der Ursprung der Kamerakoordinatensysteme im Schnittpunkt von Neigungs- und Vergenzachse.
Die Abbildung eines Punktes $p_c = (x_c, y_c, z_c, 1)$ in Kamerakoordinaten auf einen Bildpunkt $p_i = (x_i, y_i, 1)$ in Bildkoordinaten wird beschrieben durch

$$p_i = P_c \left(\frac{p_c}{z_c}\right) \quad \text{mit} \quad P_c = \begin{pmatrix} \alpha_{xc} & 0 & x_{0c} & 0 \\ 0 & \alpha_{yc} & y_{0c} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{5.11}$$

Die Parameter α_{xc}, α_{yc}, x_{0c} und y_{0c} stellen die *intrinsischen Kameraparameter* dar und müssen bei Verwendung einer modellbasierten Steuerung sorgfältig kalibriert werden.

Inverse Kinematik

Mit Kenntnis der Geometrie kann die inverse Kinematik für den monokularen und binokularen Fall direkt gelöst werden. Dabei müssen allerdings Randbedingungen angegeben werden, um eine eindeutige Lösung zu erhalten. Diese sind im vorliegenden Fall:

- Starre Schwenkachse (monokulare Fixation)
- Symmetrische Vergenz, d.h. $\theta_l = \theta_r = \theta_{l,r}$ (binokulare Fixation)

Die Lösung der Gleichungen (5.10) - (5.11) nach den zur Fixierung eines Weltpunktes nötigen Winkeln ω^* , gegeben dessen Bildkoordinaten und die Winkelgrößen, ergibt die inverse Kinematik[3]:

$$\omega^* = (\phi^*, \theta^*) = f_{direkt}^{monokular}(x, y, \theta, \phi) \quad \text{(monokularer Fall)} \tag{5.12}$$
$$\omega^* = (\chi^*, \phi^*, \theta_l^*) = f_{direkt}^{binokular}(x_l, y_l, x_r, y_r, \theta_l, \phi, \chi) \quad \text{(binokularer Fall)} \tag{5.13}$$

Dabei stellt es sich als vorteilhaft heraus, nicht direkt die neuen Winkel sondern deren Änderungen $\Delta\omega$ zu berechnen, da sich die Änderungen im monokularen Fall als unabhängig vom Neigungswinkel und im binokularen Fall als unabhängig vom Schwenkwinkel erweisen [107]. Man erhält

$$\Delta\omega = (\Delta\phi, \Delta\theta) = f_{delta}^{monokular}(x, y, \theta) \quad \text{(monokularer Fall)} \tag{5.14}$$
$$\Delta\omega = (\Delta\chi, \Delta\phi, \Delta\theta_{l,r}) = f_{delta}^{binokular}(x_l, y_l, x_r, y_r, \theta_{l,r}, \phi) \quad \text{(binokularer Fall)} \tag{5.15}$$

[3]Die im Kontext dieser Arbeit nicht weiter interessanten Formeln für die Gleichungen (5.12) bis (5.15) finden sich in [107].

Die Modelle der inversen Kinematik (5.14)-(5.15) können nun direkt zur modellbasierten Sakkadensteuerung genutzt werden, wobei man sich jedoch die wesentlichen Annahmen zur Gültigkeit der Modelle vergegenwärtigen sollte:

1. Neigungs- und Vergenzachse schneiden sich im optischen Zentrum,
2. die CCD-Chips der Kameras sind optimal justiert,
3. die Basislänge ist exakt bekannt und
4. die intrinsischen Parameter sind durch Kalibrierung bestimmt.

Die erste Annahme ist abhängig von der Qualität der Herstellung. Die zweite Annahme ist oft nur näherungsweise gültig, da die meisten CCD-Chips eine leichte Rotation aufweisen. Zwar lassen sich die Modelle erweitern, um eine Rotation der Chips und sogar eine Verschiebung des optischen Zentrums zu berücksichtigen, doch dies würde nicht nur die Modelle sondern auch die Kalibrierungsprozeduren wesentlich komplizierter gestaltet. Schließlich erfordert jede Veränderung der Parameter eine komplette Neukalibrierung.

Aus diesen Gründen kann selbst im Fall eines brauchbaren Modells eine adaptive, kalibrierungsfreie Steuerung von Vorteil sein. Dennoch ist die Analyse der inversen Kinematik nicht umsonst, gibt sie doch wertvolle Hinweise für die Entwicklung einer problemangepaßten adaptiven Steuerung: Im *monokularen* Fall kann, wie schon erwähnt, der Neigungswinkel als Eingabe entfallen. Des weiteren weist das inverse Modell eine starke lineare Komponente für kleine Änderungen $\Delta\phi$ und $\Delta\theta$ bei kleinen Startwinkeln ϕ und θ auf. Abbildung 5.3 zeigt, daß für kleine initiale Vergenzwinkel θ die Abbildung von (θ, x, y) nach $(\Delta\phi, \Delta\theta)$ nahezu linear ist (oben links), während sie für größere initiale Vergenzwinkel zunehmend nichtlinear wird. Im *binokularen* Fall erweist sich $\Delta\omega$ als unabhängig von χ, dem initialen Schwenkwinkel, und die Nichtlinearität nimmt mit zunehmenden initialen Neige- und Vergenzwinkeln zu.

Diese Eigenschaften lassen sich bei der Entwicklung der DCS-basierten Sakkadensteuerung nutzen, indem eine lokal lineare Approximation der optimalen Steuerungsfunktion gelernt wird und irrelevante Eingabewinkel von vornherein eliminiert werden. Um eine gleichmäßig gute Approximation über dem gesamten trainierten Eingaberaum zu gewährleisten, werden Lernregeln eingesetzt, die in den Bereichen starker Nichtlinearität (größere Initialwinkel) eine höhere Neuronendichte aufbauen.

5.2.2 DCS zur Adaptiven Sakkadensteuerung

Das Training der DCS erfolgt in zwei Phasen, wobei die Zellstruktur zur Beschleunigung des Lernvorganges in der vorgeschalteten *Simulationsphase* mittels eines groben Kameramodells vortrainiert wird. In der anschließenden *Arbeitsphase* bleibt die Steuerung weiter adaptiv. Nach Vorstellung dieses Zwei-Phasen-Konzeptes werden in diesem Abschnitt die Ausgabeberechnung des DCS-Netzwerkes und seine Adaption über das Fehlerrückkopplungsprinzip beschrieben.

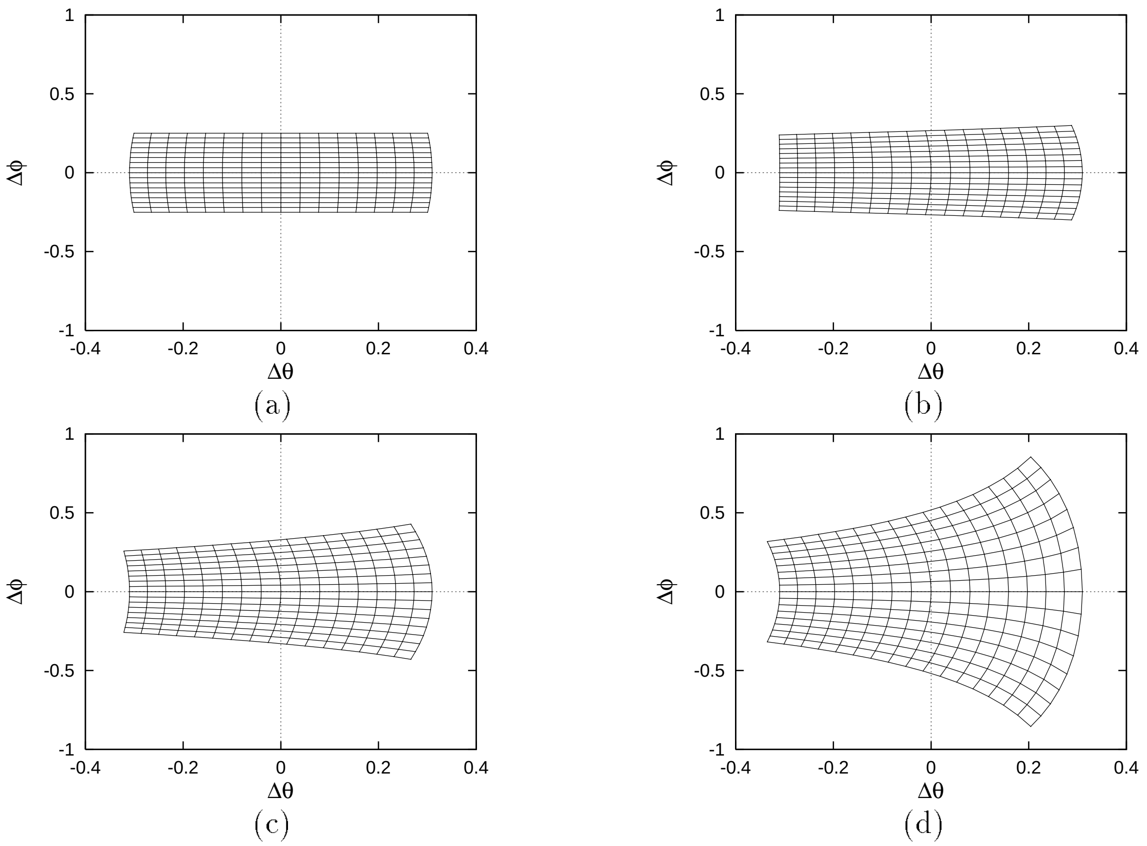


Abbildung 5.3: Die Abbildung des Bildkoordinatengitters $(x_i,\, y_i) \in [0, 512]^2$ auf die Stellwinkel $(\Delta\phi,\, \Delta\theta)$ in Abhängigkeit vom initialen Vergenzwinkel θ: $(x_i,\, y_i)|_{\phi,\theta} \to (\Delta\phi,\, \Delta\theta)$ zeigt die wachsende Nichtlinearität dieser Abbildung mit zunehmendem initialen Vergenzwinkel. (a) $\theta = 0°$, (b) $\theta = 20°$, (c) $\theta = 40°$, (d) $\theta = 60°$.

Simulationsphase In der Simulationsphase werden die RBF-Knoten über den Eingaberaum verteilt, eine OTPM aufgebaut und eine Initialisierung der Ausgabe (hier Matrizen) vorgenommen. So wird neben der erheblichen Beschleunigung des Trainings[4] erreicht, daß zu Beginn der Arbeitsphase bereits eine grob angepaßte Steuerung vorliegt. Die Stimuli in der Arbeitsphase werden zufällig generiert, wobei die Vergenz $\theta_{l,r}$ und der Neigungswinkel ϕ auf den Arbeitsbereich und die retinalen Koordinaten x_l, y_l und x_r auf das Gesichtsfeld der Kamera beschränkt bleiben. Zur Berechnung der retinalen Koordinaten nach der Fixation wird das geometrische Modell des TRC-Bisights (Gleichungen 5.10-5.11) mit grob geschätzten Kameraparametern benutzt. Das Training erfolgt wie in der darauffolgenden Arbeitsphase durch Fehlerrückkopplungs-Lernen (*kein* überwachtes Lernen). Man beachte, daß insbesondere kein Modell der inversen Kinematik benötigt wird und die Simulationsphase prinzipiell entfallen kann.

[4] Die gesamte Simulationsphase mit ca. 20000 Stimuli benötigt weniger als 5 Minuten auf einer Sparc4 Workstation. Dies entspricht 5.5 Stunden Training mit dem realen System (mit einer Sekunde für jede Armbewegung des Roboters zur Generierung des neuen Stimulus).

Arbeitsphase In der Arbeitsphase werden die Ausgabematrizen weiter durch Fehlerrückkopplungs-Lernen adaptiert, das Wachstum des Netzes und die Adaption der OTPM aber wird zugunsten gesteigerter Stabilität gestoppt.

Zielpunkte werden in der Arbeitsphase durch den Stäubli Roboterarm generiert, der mit einer am Manipulator befestigten Lichtquelle zufällige Punkte in seinem Arbeitsbereich anfährt. Mit einem Datacube Bildverarbeitungssystem werden die retinalen Koordinaten des Stimulus bezüglich der beiden Kameras in Videoechtzeit berechnet. Diese und die Stellwinkel dienen dann zur Berechnung der zur Sakkade führenden Stellwinkeldifferenz gemäß (5.16) und (5.18). Nach erfolgter Sakkade wird die Ausgabe des P-Reglers zur Adaption der Ausgabematrizen nach (5.24) und für eine Korrektursakkade genutzt.

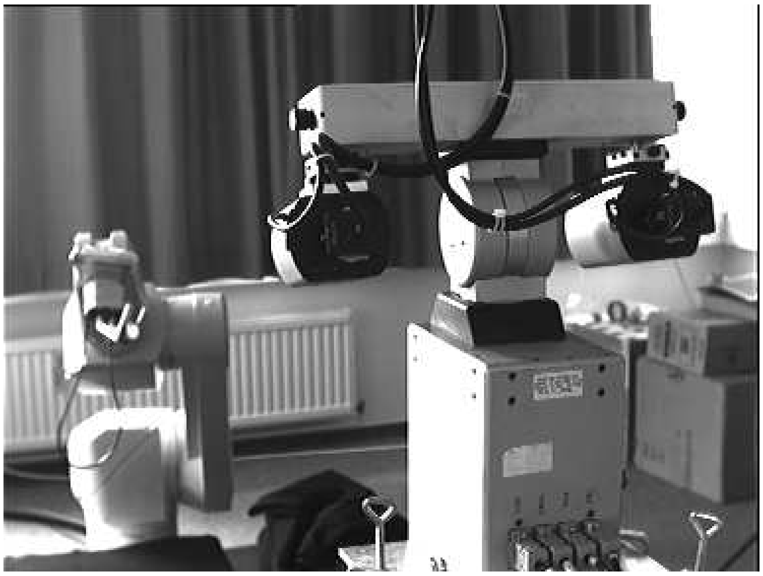


Abbildung 5.4: Versuchsaufbau zum Training in der Arbeitsphase: Stäubli R90 Roboterarm mit Lichtquelle im Hintergrund und TRC-Bisight im Vordergrund.

Ausgabeberechnung

Die stark linearen funktionalen Zusammenhänge des Sakkadensteuerungsproblems ausnutzend, wird der Eingabevektor retinaler Koordinaten und initialer Winkel $u = (p, \omega)$ nicht direkt mit den erforderlichen Stellwerten ω assoziiert, sondern mit einer Jacobi-Matrix A. Diese Matrix wird dann mit dem Vektor der retinalen Koordinaten p multipliziert und ergibt die gewünschte Winkeländerung $\Delta\omega$ (siehe Abbildung 5.5). Somit erfolgt eine lokal lineare Interpolation der inversen Kinematik auf Grundlage der den einzelnen Berechnungsknoten des DCS-Netzes zugeordneten Ausgabematrizen A_i.

Im *binokularen* Fall wird ein 5 dimensionaler Eingabevektor $u = (x_l, y_l, x_r, \theta_l, \phi)$ mit einer 3×3 Jacobi-Matrix assoziiert. Der Eingabevektor der initialen Winkel $(\theta_l, \phi) = \omega$ kommt aufgrund symmetrischer Vergenz $(\theta_l = \theta_r)$ mit nur einem Vergenzwert aus, und auch auf die Eingabe des Schwenkwinkels χ wird aufgrund der Unabhängigkeit von $\Delta\omega$

von χ verzichtet. Da die vier retinalen Koordinaten x_l, y_l, x_r, y_r nicht unabhängig sind[5], enthält der retinale Koordinatenvektor $p = (x_l, y_l, x_r)$ desweiteren nicht y_r. Als Ausgabe der Sakkadensteuerung ergibt sich

$$\Delta\omega = (\Delta\chi, \Delta\theta_{l,r}, \Delta\phi) = A^{bino}p \quad \text{(binokularer Fall).} \tag{5.16}$$

Im *monokularen* Fall wird ein 3 dimensionaler Eingabevektor $u = (x, y, \theta)$ mit einer 2×2 Jacobi-Matrix assoziiert. Mit dem retinalen Koordinatenvektor $p = (x, y)$ berechnet sich die Ausgabe zu

$$\Delta\omega = (\Delta\theta, \Delta\phi) = A^{mono}p \quad \text{(monokularer Fall).} \tag{5.17}$$

Die Matrix A wird als normalisierte gewichtete Summe der individuellen Ausgabematrizen A_i durch das DCS-Netzwerk berechnet:

$$A = \sum_{i \in Nh^{(1)}(bmu_u)} A_i a_i \quad \text{mit} \quad a_i = \frac{h(\|u - c_i\|/\sigma)}{\sum_{j \in Nh^{(1)}(bmu_u)} h(\|u - c_j\|/\sigma)} \,. \tag{5.18}$$

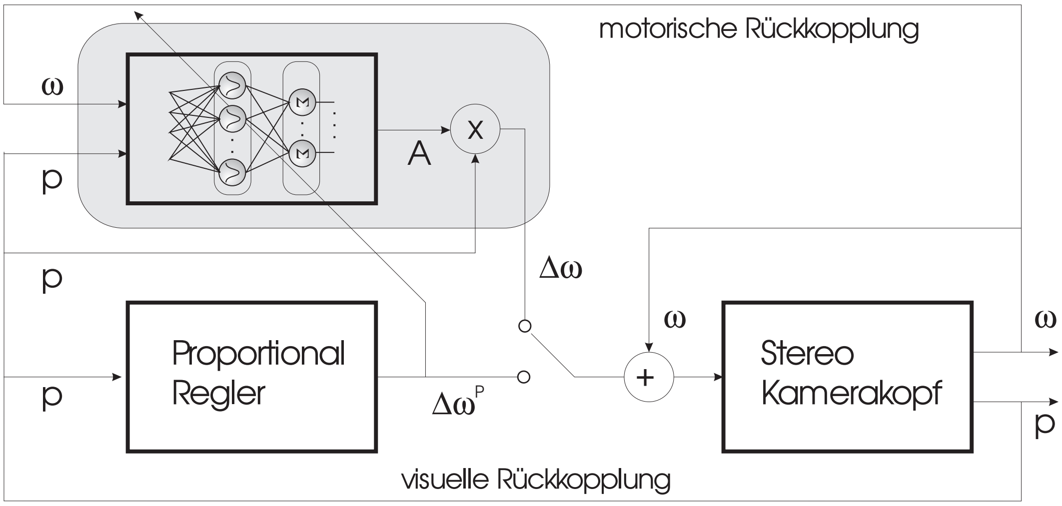


Abbildung 5.5: DCS basiertes Fehlerrückkopplungs-Lernen zur Sakkadensteuerung.

Fehlerrückkopplung zur Adaption

Nach erfolgter Sakkade genügt im Falle der binokularen Sakkadensteuerung der Stellwert eines einfachen Proportionalreglers, um das Fehlerrückkopplungssignal zu liefern. Seine Ausgabe $\Delta\omega^p = (\Delta\chi^p, \Delta\phi^p, \Delta\theta^p_{l,r})$ ist proportional zur mittleren x-Koordinate des Bildpunktes, der linken y-Koordinate und der Differenz der x-Koordinaten:

$$\Delta\omega^p = (\Delta\chi^p, \Delta\phi^p, \Delta\theta^p_{l,r}) = (k_1(x_l + x_r), k_2 y_l, k_3(x_l - x_r)) \,. \tag{5.19}$$

[5] Der Zielpunkt ist eindeutig als Schnittpunkt der beiden durch (x_l, y_l) und (x_r, y_r) festgelegten Sehstrahlen bestimmt. Genauso eindeutig ist er aber auch als Schnittpunkt des durch (x_l, y_l) festgelegten Sehstrahls mit der durch x_r definierten "Sehebene" bestimmt.

Dabei sind $k_1, \ldots, k_3$ die Verstärkungsfaktoren des P-Reglers. Für den Fall perfekter Fixierung ergibt sich damit $\Delta\omega^p = 0$ – eine notwendige Bedingung für die Verwendung als Fehlerrückkopplungssignal.

Analog läßt sich im monokularen Fall ein P-Regler benutzen, dessen Stellwerte proportional zur y- und x-Koordinate nach erfolgter Sakkade sind:

$$\Delta\omega^p = (\Delta\phi^p, \Delta\theta_l^p) = (k_1 y, k_2 x) \,. \tag{5.20}$$

Es sei an dieser Stelle darauf hingewiesen, daß sowohl für den monokularen als auch für den binokularen Fall die einfachen P-Regler genügen, um den Fixationsfehler auf null zu regeln (perfekt zu fixieren). Der einzige Grund, zusätzlich die inverse Kinematik zu lernen, ist die schnelle Fixation in einem Schritt unter Umgehung der Regelungsschleife.

Adaption der Ausgabeschicht Um die Ausgabeschicht des DCS-Netzes, d.h. die Ausgabematrizen A_i, zu adaptieren, muß das Fehlerrückkopplungssignal $\Delta\omega^p$ in eine Matrixdifferenz umgerechnet werden. Das Rückkopplungssignal $\Delta\omega^p$ zeigt an, daß

$$\Delta\omega^* = \Delta\omega + \Delta\omega^p = Ap + \Delta\omega^p \tag{5.21}$$

eine verbesserte Ausgabe der Steuerung ist. Diese Ausgabe kann mit

$$A^* = A + \Delta\omega^p \frac{p^T}{\|p\|^2} \tag{5.22}$$

erzielt werden, wie leicht durch Multiplikation von (5.22) mit p überprüft werden kann. Mit A^* als verbessertem Schätzer für die gewünschte Ausgabe der Steuerung erhält man als Differenzmatrix

$$\Delta A = A^* - A = \Delta\omega^p \frac{p^T}{\|p\|^2} \,, \tag{5.23}$$

welche in Form einer α-LMS-Regel zur Online-Adaption der Ausgabematrizen durch stochastischen Gradientenabstieg auf $(\Delta A)^2$ genutzt wird:

$$\Delta A_i = \alpha(\Delta\omega^p \frac{p^T}{\|p\|^2}) a_i \,, \; i \in Nh^{(1)}(bmu_u) \,. \tag{5.24}$$

Adaption der Neuronen-Zentren Zur Adaption der Zentrumsvektoren der Neuronen werden im Rahmen der Sakkadensteuerung drei Lernregeln experimentell untersucht, deren theoretische Eigenschaften in Abschnitt 3.4.3 diskutiert wurden:

- die Fritzke-Lernregel (3.4.3),
- die fehlermodulierte Lernregel (3.35) und
- die restringierte Gradientenabstiegsregel (3.34), die nur Bewegungen der Zentren in Richtung des negativen Gradienten des Fehlersignals erlaubt und mit $h_i = h(\|u - c_i\|/\sigma)$ hier folgende Gestalt

$$\Delta c_i = \begin{cases} \epsilon_i(u - c_i) : & \frac{\partial(\Delta A)^2}{\partial h_i} > 0 \\ 0 \quad : & \text{sonst} \end{cases} , i \in Nh^{(1)}(bmu_u) \tag{5.25}$$

mit
$$\frac{\partial(\Delta A)^2}{\partial h_i} = \nabla_A(\Delta A)^2 \frac{\partial A}{\partial h_i} = \sum_{kl} \left[\Delta\omega^p \frac{p^T}{\|p\|^2}\right]_{kl} \left[\frac{A_i - A}{\sum_{j\in Nh^{(1)}(bmu_u)} h_j}\right]_{kl}$$
annimmt, wobei $\epsilon_i = \epsilon_{bmu}$ für die bmu_u und $\epsilon_i = \epsilon_{Nh}$ für die direkten Nachbarn der bmu_u.

Daß diese Regel tatsächlich einen Gradientenabstieg auf $(\Delta A)^2$ darstellt, folgt aus der Tatsache, daß der Gradient einer radialen Basisfunktion bezüglich ihres Zentrumsvektors stets in Richtung der Differenz von Stimulus und Zentrumsvektor weist, d.h. $\nabla_c h(\|u-c\|^2) \sim -(u-c)$.

Adaption der Modellordnung Die Adaption der Modellordnung erfolgt über einen durch den Fixationsfehler gesteuerten Wachstumsprozeß, und zwar derart, daß ein neuer RBF-Knoten immer dann nach der präzisionsabhängigen Online-Wachstumsstrategie (3.48) eingefügt wird, wenn die Fehlervariable r_{bmu_u} der bmu_u und der momentane Fixationsfehler e_{fix} die geforderte Präzision δ_{err} überschreiten:

$$r_{bmu_u} > \delta_{err} \quad \text{und} \quad e_{fix} > \delta_{err}.$$

Die Adaption der Fehlervariablen r_i erfolgt nach der gleitenden Mittelwertberechnung (3.47) auf Basis des Fixationsfehlers e_{fix}. Die Wachstumssteuerung ist damit rein lokal und rein fehlergetrieben. Die Ausgabematrix A_{new} eines neu eingefügten Knotens wird mit A_{bmu_u} und der Zentrumsvektor mit c_{bmu_u} initialisiert.

Zwischen aufeinanderfolgenden Insertionen werden mindestens $n\log(n)$ Trainingsschritte gefordert (n ist die Anzahl der Knoten), damit sich möglichst komplette OTPMs ausbilden können. Diese Heuristik ergibt sich aus den Überlegungen zum Aufbau von OTPMs in Abschnitt 3.1. Sie nimmt zugunsten möglichst gut ausgebildeter OTPMs bewußt eine hohe Anzahl von Trainingsschritten in Kauf. Die Zeit für den Aufbau der OTPMs fällt nicht ins Gewicht, da der gesamte Aufbau in der Simulationsphase erfolgt. Die darauf folgende Arbeitsphase kann dann auf kleine Netze mit ausgeprägten OTPMs aufsetzen.

Die Wachstumsphase bricht ab, wenn der durchschnittliche Fixationsfehler unter die vorgegebene Präzision δ_{err} fällt.

Adaption der Topologie Als Online-Lernregel zur Adaption der Topologie wird die modifizierte Hebb-Regel (Seite 55) genutzt.

5.2.3 Experimentelle Resultate

In der Simulationsphase werden zunächst die drei in Abschnitt 5.2.2 vorgestellten Lernregeln am Beispiel der (einfacheren) monokularen Fixation bewertet, wobei sich die restringierte Gradientenabstiegsregel als überlegen erweisen wird. Diese Lernregel wird dann zum Erlernen der binokularen Sakkadensteuerung in der Arbeitsphase eingesetzt. Schließlich werden die Auswirkungen von Parameteränderungen untersucht.

Für alle folgenden Experimente kommt derselbe Satz an Lernparametern zum Einsatz (siehe Tabelle 8.6). Eingaben wurden auf das Intervall $[-1, 1]$ normalisiert. Der Fixationsfehler wird als mittlere Abweichung der retinalen Koordinaten vom Zentrum gemessen,

$$e_{fix} = \frac{|x| + |y|}{2} \text{ (monokular)} \quad \text{und} \quad e_{fix} = \frac{|x_l| + |x_r| + |y_l|}{3} \text{ (binokular).} \tag{5.26}$$

Dabei geht im binokularen Fall nur eine y-Koordinate in den Fixationsfehler ein, denn aufgrund des Aufbaus des TRC-Bisights mit nur einer gemeinsamen Neigungsachse können die y-Koordinaten nicht unabhängig geregelt werden.

Monokulare Adaptive Sakkadensteuerung

Es erfolgt nun der Vergleich der drei Lernregeln zur Zentrenadaption anhand der adaptiven monokularen Sakkadensteuerung. Der Arbeitsbereich für den Vergenzwinkel θ umfaßt dabei $[-40°, 40°]$.

	Lernregel		
δ_{err}	Fritzke-Regel	fehlermoduliert	gradientenmoduliert
3.5 Pixel	n: 10542 ± 460 N: 65 ± 1.3	n: 10284 ± 1008 N: 64 ± 2.7	n: 8184 ± 1017 N: 57 ± 3.7
2.9 Pixel	n: 30879 ± 1855 N: 105 ± 2.9	n: 25225 ± 2796 N: 95 ± 4.8	n: 17988 ± 2890 N: 82 ± 5.9
2.5 Pixel	n: 65788 ± 6252 N: 147 ± 6.0	n: 53156 ± 4617 N: 134 ± 4.7	n: 39216 ± 2827 N: 117 ± 4.0

Tabelle 5.1: Gemittelte Anzahl an Trainingsschritten n und eingefügten RBF-Knoten N mit Standardabweichung für die drei Lernregeln bei unterschiedlicher Präzision δ_{err}.

Tabelle 5.1 zeigt die gemittelte Anzahl an Trainingsschritten und die gemittelte Anzahl eingefügter RBF-Knoten für alle drei Regeln als Funktion der geforderten Präzision. Die Mittelung erfolgte jeweils über 10 Simulationen. Es ist zu beobachten, daß die gradientenmodulierte Lernregel sowohl hinsichtlich der Anzahl benötigter Trainingsschritte als auch bezüglich der Anzahl benötigter Knoten die beiden anderen Regeln unterbietet. Die fehlermodulierte Lernregel nimmt eine Mittelstellung ein. Die Tabelle zeigt des weiteren, daß die Anzahl der RBF-Knoten superlinear mit der Präzision wächst.

Die Verteilung der Zentrumsvektoren als Funktion des initialen Vergenzwinkels zeigt Abbildung 5.6. Die obere Zeile zeigt die Dichte der Einfügungsorte neuer RBF-Knoten in Abhängigkeit des Vergenzwinkels, die untere die Knotendichte nach Abschluß der Trainingsphase. Die Histogramme wurden ebenfalls über 10 Simulationen gemittelt.

Alle drei Lernregeln zeigen dasselbe qualitative Verhalten: Trotz Gleichverteilung der Stimuli bezüglich der initialen Vergenzwinkel werden weit mehr RBF-Knoten bei großen Vergenzwinkeln eingefügt als bei kleinen (obere Zeile). Dies ist in Übereinstimmung mit der inversen Kinematik, vgl. die Diskussion in Abschnitt 5.2.1: Die Nichtlinearität der Steuerung wächst mit steigenden initialen Vergenzwinkeln und erschwert so die Interpolation mit einem lokal linearen Interpolationsschema. Folglich steigt auch der Fixationsfehler,

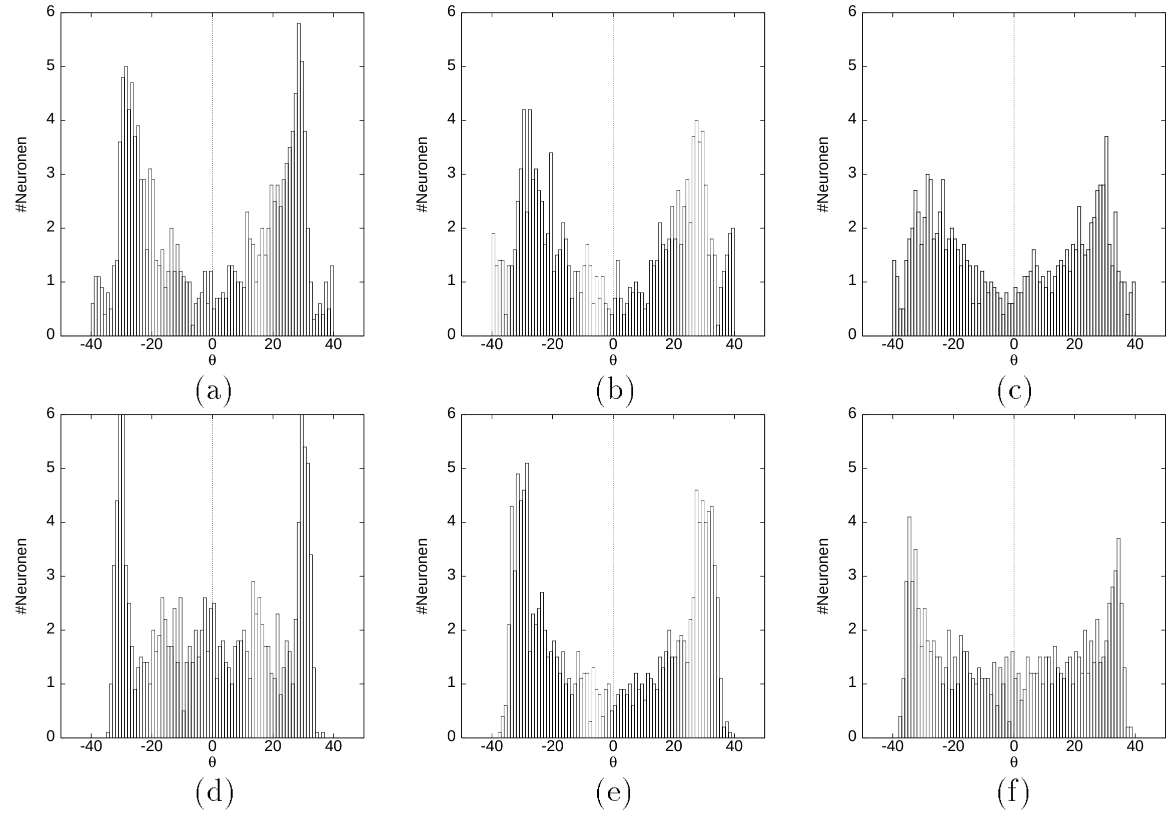


Abbildung 5.6: Verteilung der Einfügeorte neuer Neuronen in Abhängigkeit vom Vergenzwinkel (oben) und Neuronenverteilung nach Beendigung der Vortrainingsphase (unten) unter Verwendung der Fritzke-Regel (a und d), der fehlermodulierten (b und e) und der gradientenmodulierten Regel (c und f), gemittelt über 10 Versuche für monokulare Fixation.

was die fehlergetriebene Einfügestrategie durch vermehrte Insertion bei wachsenden Vergenzwinkeln auszugleichen sucht.

Die durch die fehlergetriebenen Insertionen vorgegebene Verteilung der Zentren wird im weiteren Verlauf des Trainings von den Zentrenadaptionsregeln in unterschiedlicher Weise modifiziert (Abbildung 5.6, unten). Die Fritzke-Regel, die auf eine allein von der (uniformen) Eingabedichte abhängige Zentrenverteilung hinausläuft, zeigt einen deutlichen "Verflachungseffekt" für kleine und mittlere Winkel. Die beiden Peaks sind dabei Relikte des fehlergetriebenen Wachstumsprozesses, verstärkt durch den sogenannten Randeffekt, der Zentren am Rand der Eingabemannigfaltigkeit in Richtung des Schwerpunktes verschiebt. Die gradientenbasierte und die fehlermodulierte Lernregel hingegen erzeugen mit dem Vergenzwinkel (und damit der Nichtlinearität) steigende Zentrendichten, die für das im Vergleich zur Fritzke-Regel bessere Steuerungsverhalten (Tabelle 5.1) verantwortlich sind.

Binokulare Adaptive Sakkadensteuerung

Der Arbeitsbereich in der Simulationsphase der binokularen Sakkadensteuerung umfaßt $[-30°, 30°]$ für den Neigungswinkel ϕ und $[0°, 10°]$ für den Vergenzwinkel $\theta_{l,r}$. Er umfaßt den Arbeitsbereich der Arbeitsphase ($\phi \in [-14°, 2°]$ und $\theta \in [2°, 4°]$).

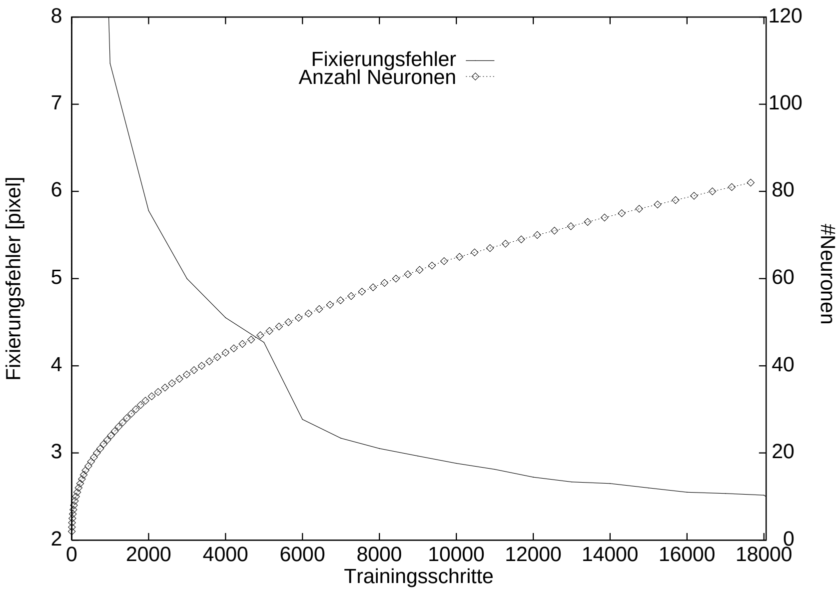
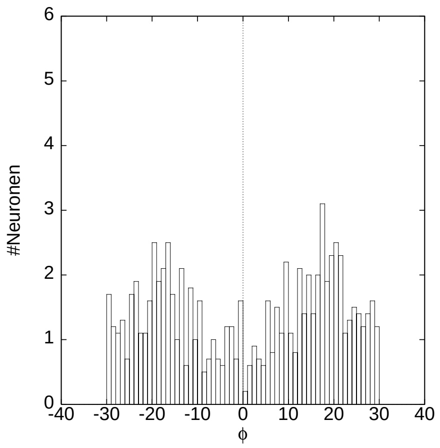


Abbildung 5.7: Entwicklung des binokularen Fixierungsfehlers in Pixel und der Anzahl der Neuronen im DCS-Netz in der Simulationsphase.

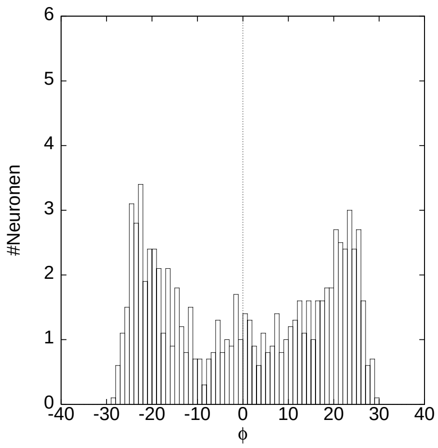
Simulationsphase Abbildung 5.7 zeigt den Verlauf des Fixationsfehlers und die Anzahl an RBF-Knoten in Abhängigkeit von der Zahl der Trainingsschritte. In diesem Fall wird die geforderte Präzision von 0.5% nach 18000 simulierten Sakkaden mit 82 Knoten errreicht. Im Durchschnitt (Mittelung über 10 Simulationen) wird diese Präzision nach 19155 ± 1340 Trainingssakkaden mit nur 84 ± 2.6 RBF-Knoten erreicht. Wiederum hätten weniger Trainingsschritte ausgereicht, allerdings auf Kosten einer größeren Neuronenzahl und einer weniger gut ausgeprägten lateralen Verbindungsstruktur. Die Trainingszeit von weniger als 5 Minuten in der Simulationsphase für 20000 Sakkaden auf einer Sparc4 Workstation ist praktisch vernachlässigbar. Man beachte, daß bereits nach 3500 Schritten mit weniger als 40 Neuronen die beachtliche Präzision von 1.0% erreicht ist (Abbildung 5.7).

Abbildung 5.8 zeigt das gemittelte Histogramm der Verteilung der Zentrumsvektoren in Abhängigkeit vom Neigungswinkel zum Zeitpunkt des Einfügens (links) und nach Beendigung der Simulationsphase (rechts) für die binokulare Sakkadensteuerung unter Benutzung der gradientenbasierten Lernregel. Wiederum reflektiert die mit dem Neigungswinkel wachsende Einfügehäufigkeit und Zentrendichte die zunehmende Nichtlinearität.

Arbeitsphase Die vortrainierte Sakkadensteuerung kommt nun auf dem realen TRC Kamerakopf zum Einsatz. Abbildung 5.9 zeigt, daß der Fehler aufgrund von Abweichungen des (groben) Simulationsmodells von der Realität zu Beginn der Arbeitsphase ca. 7 Pixel (1.5%) beträgt. Durch Online-Adaption nach dem Fehlerrückkopplungsprinzip nimmt dieser Fehler schnell ab und fällt bereits nach 1000 Sakkaden unter 2.0 Pixel.



(a)



(b)

Abbildung 5.8: Binokulare Sakkadensteuerung: (a) Verteilung der Einfügeorte neuer Neuronen in Abhängigkeit vom Neigungswinkel und (b) Neuronenverteilung nach Beendigung der Vortrainingsphase unter Verwendung der gradientenmodulierten Regel, gemittelt über 10 Versuche.

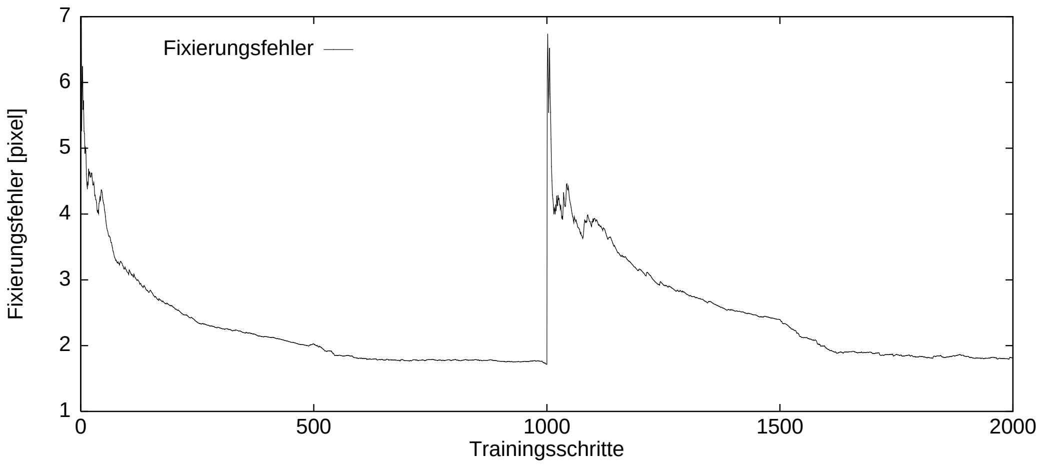


Abbildung 5.9: Verlauf des Fixierungsfehlers in der Arbeitsphase. Nach 1000 Trainingsschritten erfolg eine Änderung des Zooms.

Den Abschluß der Untersuchungen zur Sakkadensteuerung bildet ein an [54] angelehntes Experiment, in dem demonstriert wurde, daß das oculomotorische System des Menschen Änderungen des Zusammenhangs zwischen visuellem Input und notwendiger Sakkade zur Fixierung adaptiv folgen kann. In [54] wurde einer Versuchsperson eine "Kontaktlinsen-Brillen-Kombination" aufgesetzt, bei der sich die Sehschärfenkorrekturen von Brille und Kontaktlinse gegenseitig aufheben. Aufgrund des Restabstandes zwischen Brille und Kontaktlinse wirkt eine solche Kombination wie ein schwaches Galileisches Fernrohr, und vorher korrekte Sakkaden verfehlen nun ihr Ziel um einige Grad. Es wurde beobachtet, daß das Auge anfänglich mit zusätzlichen Korrekturbewegungen nach jeder Sakkade den künstlich erzeugten Fehler ausgleicht. Bereits nach 9-14 Minuten aber hatte das oculomotorische System die Sakkaden wieder soweit angeglichen, daß sich die Augenbewegungen nicht mehr von denen ohne Kontaktlinsen-Brillen-Kombination unterschieden und keine größeren Korrekturbewegungen mehr erforderlich waren.

Einen ähnlichen Effekt ergibt die Vergrößerung des Zooms der Kameras[6]. Abbildung 5.9 zeigt, wie sich nach einer Vergrößerung des Zooms (nach 1000 Sakkaden in der Arbeitsphase) der Fixationsfehler vergrößert, die adaptive Sakkadensteuerung sich jedoch ähnlich dem oculomotorischen System der Testpersonen den veränderten Parametern in nur weiteren 1000 Sakkaden anpassen kann. Durch die Vergrößerung werden nun kleinere Winkeländerungen zur Kompensation derselben retinalen Abweichungen benötigt.

5.2.4 Alternative Ansätze zur adaptiven Steuerung von Kameraköpfen mit KNN

Der dem hier geschilderten Verfahren zur adaptiven Sakkadensteuerung mit KNN ähnlichste Ansatz wurde von Mayhew et al. [82, 83] beschrieben. In [82] wurden mehrere KNN-Typen auf ihre Tauglichkeit zur Sakkadensteuerung eines simulierten Kamerakopfes mit vier Freiheitsgraden untersucht. Die getesteten Netztypen umfaßten einen linearen Assoziator, ein MLP, einen *Cerebellar Model Arithmetic Computer* (CMAC) und zweistufige hierarchische Netzwerke. Letztere bestanden aus einem linearen Assoziator plus einem CMAC oder MLP zur Kompensation des Restfehlers des linearen Assoziators. Es wurden sowohl überwachtes Lernen als auch Fehlerrückkopplungs-Lernen eingesetzt und ein 7-dimensionaler Eingabevektor (einschließlich Schwenkwinkel) mit den zur Fixierung nötigen Änderungen des Schwenk-, Neige- und Vergenzwinkels assoziiert. Dabei wurde eine starke lineare Komponente beobachtet, die den Einsatz der zweistufigen Netze motivierte[7]. Mayhew et al. schlußfolgerten insbesondere, daß es für weiterführende Arbeiten nützlich sei, KNN zu entwickeln und einzusetzen, die ihre Granularität anpassen und inkrementell trainiert werden können. Die in diesem Kapitel beschriebene DCS-basierte adaptive Sakkadensteuerung greift diesen Gedanken auf.

Eine von Mayhew et al. durchgeführte Gewichtsanalyse ihrer MLP-Netze ergab relativ kleine Gewichte für Neuronen, die den Schwenkwinkel als Eingabe erhielten, und relativ große Gewichte für Neuronen, die den Neigungswinkel des Kamerakopfes als Eingabe erhielten. Im Licht der inversen Kinematik läßt sich diese Beobachtung leicht erklären, denn die zur Fixierung nötige Winkeldifferenz ist unabhängig vom Schwenkwinkel.

In [83] verwenden Mayhew et al. ihre *Parametrized Interpolating Look-Up Table* (PILUT) zur adaptiven Sakkadensteuerung. Es werden keine quantitativen Resultate angegeben, doch hat die PILUT-Architektur gewisse Ähnlichkeiten zu dem hier geschilderten Ansatz: Eine erste Schicht fungiert als eine grobcodierte Indizierungsschicht (vergleichbar der Schicht aus RBF-Knoten), mit deren Hilfe in der zweiten Schicht eine stückweise lineare Approximation erzeugt wird. Es wird jedoch weder von Ansätzen berichtet, die Indizierungs-

[6] Dies entspricht einer Vergrößerung der Brennweite und damit der intrinsischen Parameter α_{xc} und α_{yc} sowie x_{0c} und y_{0c}, falls letztere von null verschieden sind.

[7] Mit ihren besten Netzwerken, den zweistufigen, erzielten Mayhew et al. einen Fixationsfehler von 5 Pixel mit simulierten Kameras von 256^2 Pixel pro Bild. Im Vergleich hierzu erreicht die DCS-basierte Steuerung in einem ähnlichen Arbeitsbereich und nach vergleichbarer Trainingszeit eine 2.5 Pixel Genauigkeit auf 512^2 Pixel Bildern.

schicht wachsen zu lassen, noch von Ansätzen, die PILUT topologische Zusammenhänge lernen zu lassen.

5.3 Kalibrierungsfreie Hand-Auge-Koordination

Dieser Abschnitt orientiert sich an den fast klassisch zu nennenden Versuchen von H. Ritter et al. [109] zum Erlernen der Hand-Auge-Koordination eines simulierten Roboterarms und eines den Arbeitsbereich beobachtenden simulierten Stereokamerasystems. Das System lernt allein aus der Beobachtung des Robotermanipulators durch das Kamerasystem, einen 4-dimensionalen Vektor, der die Projektion eines Punktes im 3D-Arbeitsbereich auf die Bildebenen der beiden Kameras repräsentiert, mit den zum Anfahren dieses Punktes nötigen Gelenkstellwinkeln zu assoziieren. Das Lernverfahren beruht auf dem Prinzip der Fehlerrückkopplung, da das System den beim Anfahren zufälliger Zielpunkte wahrgenommenen Fehler in Kamera-Bildkoordinaten benutzt, um sich zu verbessern. Zur Fehlerminimierung wird eine LMS-Methode eingesetzt, gleichzeitig organisieren sich die verwendeten neuronalen Einheiten auf einem 3D-Gitter nach der selbstorganisierenden Lernregel von T. Kohonen. Trotz fehlenden Vorwissens und ungünstiger Randbedingungen (zufällige Initialisierung des Netzes und zufälliges Abtasten des Arbeitsraumes) lernt das System in deutlich weniger als 10000 Bewegungen Punkte im Arbeitsraum anzufahren. Ritter et al. zeigten des weiteren, daß das System innerhalb seines Arbeitsbereichs in der Lage ist, sich schnell an Veränderungen der Kinematik des Roboterarmes, z.B. eine Verkürzung eines Armsegmentes, anzupassen. Im Vergleich hierzu bildet ein klassischer Ansatz zuerst die Bildkoordinaten des Punktes im Arbeitsraum auf seine 3D-Koordinaten ab, wozu Kenntnisse über die genau Position und Orientierung der Kameras sowie deren Abbildungseigenschaften (Modell der Kamera) erforderlich sind. Die 3D-Koordinaten schließlich werden über ein Modell der inversen Kinematik des Roboterarmes in Gelenkwinkelstellungen umgerechnet. Sämtliche Fehler im Kamera- und Robotermodell gehen in die Ansteuerung des Punktes ein, und unter veränderten Betriebsbedingungen ist eine Neubestimmung der Modellparameter (Rekalibrierung) unumgänglich.

Der Versuchsaufbau wurde von zahlreichen Forschern auf dem Gebiet visuomotorischer Koordination mittels KNN aufgegriffen, so auch von Martinetz [80] und Walter et al. [131]. Beide Autoren demonstrierten erfolgreich die Anwendbarkeit der Erweiterten Kohonenkarte (EKK) und des Neuronengas (ENG) auf die Steuerung eines realen Industrieroboters. In diesem Abschnitt soll nun als weiterer Test der DCS sowie zur Demonstration der unterschiedlichen Eigenschaften von EKK, ENG und DCS auch letztere im Versuchsaufbau nach Ritter et al. zum Einsatz kommen [20, 1].

5.3.1 Der Positioniervorgang

Abbildung 5.10 zeigt den Versuchsaufbau. Ein (starres) Stereokamerasystem erfaßt einen vorgegebenen Zielpunkt und liefert dessen retinale Koordinaten $p = (x_l, y_l, x_r, y_r)$. Der vierdimensionale Vektor p kodiert eindeutig die Position des Zielpunktes im dreidimensionalen Arbeitsbereich und wird genutzt, um aus einer Menge von Neuronen dasjenige zu ermitteln, dessen Zentrumsvektor c_i den geringsten euklidischen Abstand zu p hat (Ermittlung der bmu_p). Jedem Neuron ist eine Jacobi-Matrix A_i und ein Gelenkwinkelvektor θ_i zugeordnet. Der zum Anfahren des Zielpunktes mit dem Roboterarm erforderliche Ge-

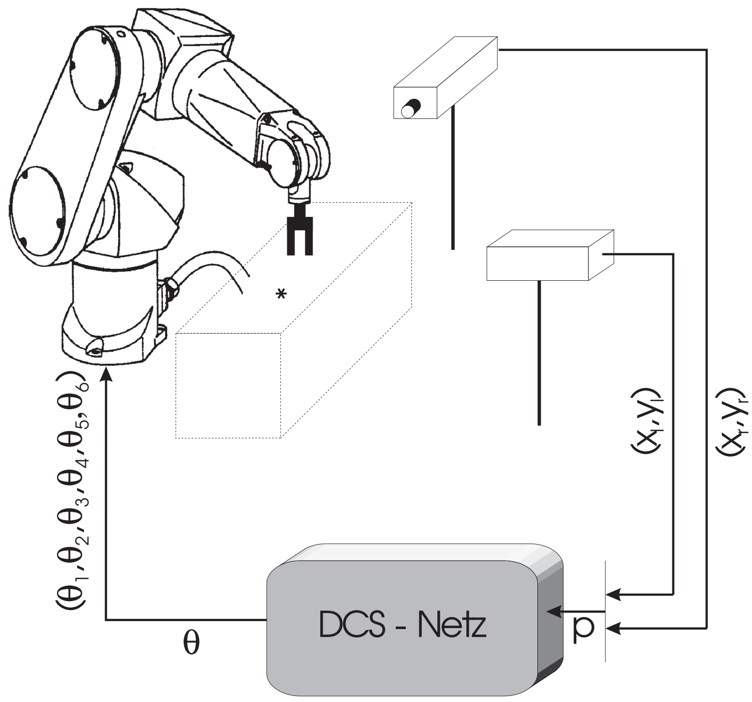


Abbildung 5.10: Versuchsaufbau zum Erlernen der Hand-Auge-Koordination. Abweichend vom Text bezeichnet in dieser Abbildung θ_i die i.te Komponente des 6-dimensionalen Gelenkwinkelvektors, und nicht den Ausgabevektor des i.ten Knotens. Die sechs Freiheitsgrade (6 DOF) des Stäubli RX90 Roboterarm sind: Drehung um die vertikale Achse, Bewegung des unteren und oberen Armsegmentes in der vertikalen Ebene, Drehung des oberen Armsegmentes sowie Kippen und Drehen des Endeffektors.

lenkwinkelvektor θ^0 wird damit zu

$$\theta^0 = \begin{cases} \theta_{bmu_p} + A_{bmu_p}(p - c_{bmu_p}) & : \text{ EKK und ENG} \\ \sum_{i \in Nh^{(1)}(bmu_p)} [\theta_i + A_i(p - c_i)] a_i : & \text{ DCS} \end{cases} . \tag{5.27}$$

berechnet, wobei die Aktivierungen a_i für das normalisierte DCS-Netz durch

$$a_i = \frac{h(\| p - c_i \| / \sigma)}{\sum_{j \in Nh^{(1)}(bmu_p)} h(\| p - c_j \| / \sigma)} . \tag{5.28}$$

gegeben sind.

In allen drei Fällen findet eine lokal lineare Approximation statt. Der einzige Unterschied besteht darin, daß im ersten Fall der Term nullter Ordnung (die Gelenkwinkel θ_{bmu_p}) und der Term erster Ordnung (die Jacobi-Matrix A_{bmu_p}) in der gesamten Voronoizelle der bmu_p konstant angenommen werden, während dies im Falle des DCS-Netzes durch RBF-Interpolation zwischen der bmu_p und ihren Nachbarn nicht der Fall ist. Damit sollte - und dies bestätigt sich im Experiment - bei gleicher Neuronenzahl mit dem DCS-Netz eine höhere Genauigkeit erzielbar sein.

5.3.2 Fehlerrückkopplung zur Adaption

Nach Anfahren der Gelenkwinkel θ^0 nimmt das Kamerasystem den Endeffektor an den retinalen Koordinaten p^0 wahr. Die Differenz zwischen Sollwert p und Istwert p^0 in Bildkoordinaten wird nun nach dem Fehlerrückkopplungsprinzip in eine Differenz $\Delta\theta$ für verbesserte Ausgabegelenkwinkel umgerechnet. Ein einfacher P-Regler wie im Falle der Sakkadensteuerung reicht hier nicht aus. Ist aber die Jacobi-Matrix A,

$$A = \begin{cases} A_{bmu_p} & : \text{ EKK und ENG} \\ \sum_{i \in Nh^{(1)}(bmu_p)} A_i a_i & : \text{ DCS} \end{cases}, \tag{5.29}$$

lokal gültig, so kann diese genutzt werden, und man erhält als Fehlerrückkopplungssignal

$$\Delta\theta = A(p - p^0). \tag{5.30}$$

Die Jacobi-Matrix A wird jedoch zu Beginn entsprechend der Initialisierung der A_i einen eher zufälligen Wert annehmen. Deshalb gilt es, zuerst die Ausgabematrizen A_i zu adaptieren, um dann das Fehlerrückkopplungssignal nach (5.30) zur Adaption der Ausgabegelenkwinkel θ_i nutzen zu können. Hierzu werden eine zweite Gelenkwinkelstellung θ^1 und ihre zugehörigen Bildkoordinaten p^1 benötigt. Liegt p^1 dicht bei p^0, so kann der quadratische Fehler

$$E(A) = \frac{1}{2}[(\theta^1 - \theta^0) - A(p^1 - p^0)]^2 \tag{5.31}$$

genutzt werden, um mittels stochastischen Gradientenabstieges eine Verbesserung von A zu erzielen [109]:

$$\Delta A = \nabla_A E(A) = [(\theta^1 - \theta^0) - A(p^1 - p^0)](p^1 - p^0)^T. \tag{5.32}$$

Die zweite Gelenkwinkelstellung wird unter Ausnutzung des Fehlerrückkopplungssignals $\Delta\theta$ angefahren,

$$\theta^1 = \theta^0 + A(p - p^0), \tag{5.33}$$

und stellt bei fortgeschrittenem Training einen Korrekturschritt in Richtung des Zielpunktes dar. Mit (5.33) lassen sich die Gelenkwinkel θ^1 und θ^0 aus (5.32) eliminieren und man erhält

$$\Delta A = [A(p - p^1)](p^1 - p^0)^T. \tag{5.34}$$

Der Korrekturschritt ist beliebig oft wiederholbar. Man beachte, daß die Gelenkwinkel θ^1 auch dann zur Adaption von A mittels (5.34) genutzt werden können, wenn sie (am Anfang des Trainings) keine Verbesserung bezüglich der Annäherung des Endeffektors an das Zielobjekt darstellen.

Das eben vorgestellte Lernverfahren besteht somit in der Adaption eines hierarchischen Regelkreises: Der erste Regler, der Rückkopplungsregler, wird von der gewichteten Summe der Ausgabematrizen A_i realisiert und unter Ausnutzung von (5.34) adaptiert. Er liefert dann das Fehlerrückkopplungssignal $\Delta\theta$, das zur Adaption der eigentlichen Steuerung, gebildet durch die gewichtete Summe der θ_i, und für eventuelle Korrekturbewegungen genutzt werden kann. Dies ist in Abbildung 5.11 verdeutlicht.

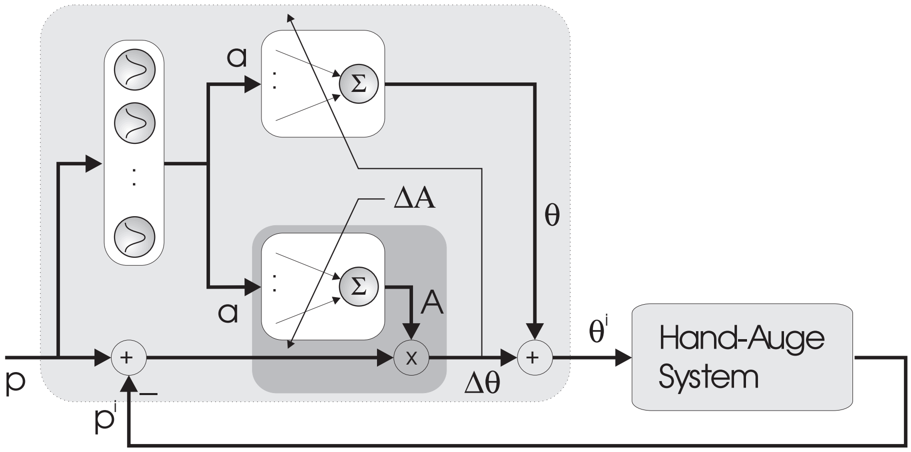


Abbildung 5.11: DCS basiertes Fehlerrückkopplungs-Lernen zur Hand-Auge-Koordination

Das komplette Lernverfahren zur Adaption der drei KNN ist in Abbildung 5.12 zusammengefaßt. Es sollen jetzt die noch offenen Schritte erläutert werden. Diese sind: Initialisierung, Adaption der Zentrumsvektoren c_i, Adaption der Ausgabematrizen A_i und der Ausgabegelenkwinkel θ_i sowie die Adaption der Topologie im Falle des DCS-Netzwerkes.

```
 Initialisierung
do {
    Gib zufälligen Zielpunkt innerhalb des Arbeitsbereiches an;
    Registriere seine retinalen Koordinaten p;
    Bestimme bmu_p und smu_p;
    Adaptiere die Zentrumsvektoren c_i;
    Adaptiere die Topologie (Adjazenzmatrix C); % nur DCS
    Fahre den Zielpunkt an, d.h. stelle Gelenkwinkel θ_i nach (5.27) ein;
    Registriere die retinalen Manipulatorkoordinaten p^i;
    do {
        Führe Korrekturschritt aus, i.e stelle neue Gelenkwinkel θ^{i+1} nach (5.33) ein;
        Registriere die neuen retinalen Koordinaten des Manipulators p^{i+1};
        Adaptiere die Ausgabematrizen A_i und die Ausgabegelenkwinkel θ_i;
    } while (noch nicht genügend Korrekturschritte)
} loop
```

Abbildung 5.12: Lernalgorithmus zur Hand-Auge-Koordination

Initialisierung

Um einen Vergleich der KNN-Modelle zu ermöglichen, ist die Initialisierung für alle drei Netzwerke identisch: Es werden N Knoten vorgegeben, deren Zentren c_i zufällig im retinalen

Eingaberaum initialisiert und deren Ausgabematrizen A_i mit kleinen zufälligen Anfangswerten belegt werden. Die Ausgabegelenkwinkel θ_i werden alle mit denselben Winkelstellungen initialisiert, die eine Manipulatorposition im Arbeitsbereich beschreiben. Im Falle des DCS-Netzwerkes wird zudem beim Einfügen ein neuer Knoten mit seinem im retinalen Eingaberaum nächsten Nachbarn verbunden und so die Adjazenzmatrix initialisiert.

Es bleibt anzumerken, daß mit dieser Initialisierung bewußt ein wesentlicher Vorteil der DCS-Netze, nämlich ihr fehlergesteuertes Wachstum, nicht ausgenutzt wird. Zudem muß das DCS-Netzwerk erst lernen, daß die intrinsische Dimensionalität des Eingaberaumes dreidimensional ist, während dieses Vorwissen in der Form der Auswahl eines dreidimensionalen Gitters bei den Kohonenkarten von vornherein mit einfließt.

Adaption der Zentren

Die Adaption der Zentren erfolgt nach den selbstorganisierenden Lernregeln von Kohonen für die Kohonenkarte (2.12), von Martinetz für das Neuronengas (2.19) und der Fritzke-Lernregel für das DCS-Netzwerk (3.4.3):

$$\Delta c_i = \begin{cases} \epsilon(t)(p - c_i) g^{Kohonen}_{bmu_p,i}(t) & : \text{ EKK} \\ \epsilon(t)(p - c_i) g^{NG}_{ord(i)}(t) & : \text{ ENG} \\ \epsilon_i(t)(p - c_i)\ ,\ i \in Nh^{(1)}(bmu_p) & : \text{ DCS} \end{cases} . \tag{5.35}$$

Wie üblich ist der Lernparameter ϵ und die Stärke der Nachbarschaftskooperation sowohl bei der Lernregel für die Kohonenkarte als auch bei der Lernregel für das Neuronengas zeitabhängig und nehmen mit der Zeit auf null ab. Eine Besonderheit ist, daß hier zum Zwecke der Vergleichbarkeit auch für das DCS-Netz die Lernparameter eingefroren werden.

Adaption der Topologie

Die Topologie des DCS-Netzwerkes wird nach der modifizierten Hebb-Regel (Seite 55) adaptiert. Das Kohonennetzwerk hat eine starre dreidimensionale Topologie. Beim Neuronengas erfolgt eine Neuberechnung der Ordnungsrelation in jedem Schritt.

Adaption der Ausgabematrizen und -gelenkwinkel

Zur Adaption der Ausgabegelenkwinkel θ_i wird in allen drei Fällen das Fehlerrückkopplungssignal $\Delta\theta$ nach (5.30) verwendet. Die Adaption der Ausgabematrizen erfolgt aufgrund (5.34). Mit diesen Größen lassen sich verbesserte Schätzer für θ und A angeben:

$$\theta^* = \theta^0 + \Delta = \theta^0 + A(p - p^0) \quad \text{und} \tag{5.36}$$

$$A^* = A + \frac{\Delta A}{\| p^1 - p^0 \|^2} = A + \frac{[A(p - p^1)](p^1 - p^0)^T}{\| p^1 - p^0 \|^2}. \tag{5.37}$$

Mit den Lernregeln für die erweiterte Kohonenkarte (2.16), das erweiterte Neuronengas

(2.23) und die LMS-Regel (3.15) für die DCS ergibt sich

$$\Delta A_i = \begin{cases} \alpha(t)(A^* - A_i)g^{EKK}_{bmu_p,i}(t) & : \text{EKK} \\ \alpha(t)(A^* - A_i)g^{ENG}_{ord(i)}(t) & : \text{ENG} \\ \alpha(t)(A^* - A)a_i \;,\; i \in Nh^{(1)}(bmu_p) & : \text{DCS} \end{cases} \quad (5.38)$$

$$\Delta \theta_i = \begin{cases} \alpha(t)(\theta^* - \theta_i)g^{EKK}_{bmu_p,i}(t) & : \text{EKK} \\ \alpha(t)(\theta^* - \theta_i)g^{ENG}_{ord(i)}(t) & : \text{ENG} \\ \alpha(t)(\theta^* - \theta^0)a_i \;,\; i \in Nh^{(1)}(bmu_p) & : \text{DCS} \end{cases} . \quad (5.39)$$

Die Lernregeln zeigen, daß in den ersten beiden Modellen ein wirklicher Gradientenabstieg nur bezüglich der bmu_p stattfindet, während im DCS alle an der Ausgabeberechnung beteiligten Knoten nach den Gradienten adaptiert werden. Die Adaption der topologischen Nachbarn in der Kohonenkarte und im Neuronengas basiert allein auf Nachbarschaftskooperation. Dies führt zwar in dieser Anwendung zu qualitativ ähnlichen Ergebnissen, es läßt sich aber keine zu minimierende Fehlerfunktion angeben.

5.3.3 Experimentelle Resultate

Im folgenden werden die experimentellen Ergebnisse zur Hand-Auge-Koordination sowohl auf Basis einer Simulation (mit dem Simderella-Simulator [127]) als auch einer Implementierung auf dem realen Stäubli RX90 Industrieroboter in Zusammenspiel mit einem Stereokamerasystem beschrieben.

Simulation

Abbildung 5.13 zeigt die geglätteten Verläufe der Positionierungsfehler für die drei Netzwerkmodelle, wobei ein $7 \times 7 \times 7$ Kohonengitter sowie ein Neuronengas und ein DCS-Netz mit jeweils 300 Neuronen verwendet wurden[8]. Links ist der Positionierungsfehler über der Anzahl der Trainingsschritte aufgetragen, rechts über der verbrauchten CPU-Zeit zur Ausgabeberechnung und Adaption (wobei die Lernkurve des DCS-Netzwerkes 15000 Schritte umfaßt, die der Kohnenkarte und des Neuronengases jeweils nur 4000). Tabelle 5.2 zeigt den Positionierungsfehler nach 4000 und nach 15000 Schritten Training sowie die relative Zeit, die ein Trainingsschritt durchschnittlich benötigt. Der simulierte Arbeitsbereich für diese Experimente umfaßte $30 \times 30 \times 30\,cm^3$, der Fehler mißt die Abweichung zwischen Zielpunkt und Manipulatorkoordinaten im Arbeitsbereich in cm. Zu sehen ist, daß das Neuronengas mit Abstand die wenigsten *Trainingsschritte*, das DCS hingegen die meisten benötigt. Zudem fällt der Positionierungsfehler des Neuronengases und der Kohonenkarte nach 4000 Zeitschritten geringer aus als der des DCS-Netzwerkes. Wird jedoch über 10000 Schritte trainiert, so ist zu bemerken, daß das DCS-Netz einen leicht geringeren Endpositionierungsfehler erreicht. Betrachtet man hingegen den Verlauf des Positionierungsfehlers über der *Trainingszeit*, wobei hier die CPU-Zeit gemessen wurde, die jedes Modell zur Berechnung der Anfahrpunkte und der Adaption benötigt, so schneidet das DCS-Netz am besten ab, das Neuronengas am schlechtesten. Die CPU-Zeit wurde dabei

[8]Die Parameter für die drei verwendeten Netze finden sich in den Tabellen 8.7, 8.8 und 8.9.

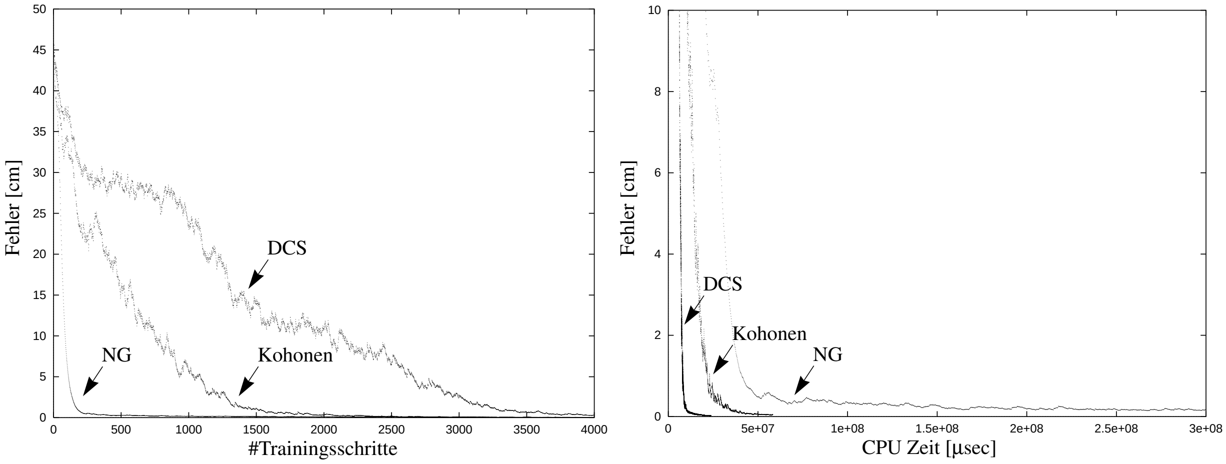


Abbildung 5.13: Positionierungsfehler im Arbeitsbereich für Kohonengitter, Neuronengas und DCS mit $7 \times 7 \times 7$ bzw. 300 Neuronen. Links: über der Anzahl der Trainingsschritte. Rechts: über der zur Ausgabeberechnung und Adaption benötigten CPU-Zeit.

Netztyp	Pos.fehler [cm] (4000 Schritte)	Pos.fehler [cm] (15000 Schritte)	rel. CPU-Zeit/ Trainingsschritt
Kohonenkarte	0.04	0.04	7
Neuronengas	0.03	0.03	80
DCS-Netz	0.3	0.01	3

Tabelle 5.2: Präzision und relative CPU-Zeit pro Trainingsschritt für Kohonenkarte, Neuronengas und DCS-Netz mit $7 \times 7 \times 7$ bzw. 300 Neuronen

auf einer Sparc10 Workstation gemessen. Für keines der Modelle erfolgte eine spezielle Optimierung. Zur Sortierung nach dem euklidischen Abstand im Neuronengas wurde eine Standardimplementierung des Quicksortverfahrens genutzt.

Das Experiment verdeutlicht die Unterschiede in der Lernphase zwischen den drei Netztypen: Im Neuronengas sind die Neuronen nach der zufälligen Initialisierung im Arbeitsbereich schon nahezu optimal verteilt, und die topologische Beziehung zwischen den Neuronen muß nicht erst gelernt werden, sondern wird durch Sortierung in jedem Schritt neu berechnet. Damit ist die nötige Zentrenadaption minimal und es wird sofort mit dem Erlernen korrekter Ausgabematrizen und -gelenkwinkel begonnen. Sortierung aber benötigt auf einem seriellen Rechner $O(n \log n)$ Zeit, weshalb die Berechnungszeit pro Trainingsschritt ebenso steigt und bei 300 Neuronen deutlich höher als bei den beiden anderen Modellen ist[9]. Die Kohonenkarte kommt ebenfalls ohne Topologielernen aus und ist in diesem Fall optimal an die dreidimensionale Geometrie des Arbeitsraumes angepaßt. Allerdings sind zu Beginn die Neuronen zufällig im Arbeitsraum verteilt, so daß sich das Netz erst entfalten muß.

Im DCS-Netz hingegen muß erst die dreidimensionale Topologie des Arbeitsraumes er-

[9]Beim Neuronengas läßt sich jedoch noch eine Beschleunigung erzielen, wenn mit fortschreitender Trainingsdauer nur eine feste Zahl von $k < n$ euklidisch nächsten Neuronen berücksichtigt wird.

faßt und seine laterale Verbindungsstruktur aufgebaut werden, bevor es zu einer korrekten Adaption der Ausgabematrizen und -gelenkwinkel kommen kann. Im Experiment waren es am Ende des Trainings ca. 1800 laterale Verbindungen, die aufgrund des zufälligen Abtastens des Arbeitsraumes erlernt wurden. Nach dieser "Investitionsphase" hat das DCS-Netz allerdings die topologische Information, die sich das Neuronengas in jedem Schritt neu errechnen muß, in seiner lateralen Verbindungsstruktur gespeichert. Die Ermittlung der bmu_p und ihrer direkten topologischen Nachbarn kann nun in der Zeit $O(n)$ erfolgen. Es ist auch nur diese Teilmenge von Neuronen, die in einem Trainingsschritt aktiviert und adaptiert werden muß. Zugleich nimmt das DCS-Netz als einziges eine Interpolation zwischen den Ausgaben benachbarter Neuronen vor und ist dadurch in der Lage, bei gleicher Neuronenanzahl eine höhere Approximationsgüte als die Kohonenkarte oder das Neuronengas zu erzielen.

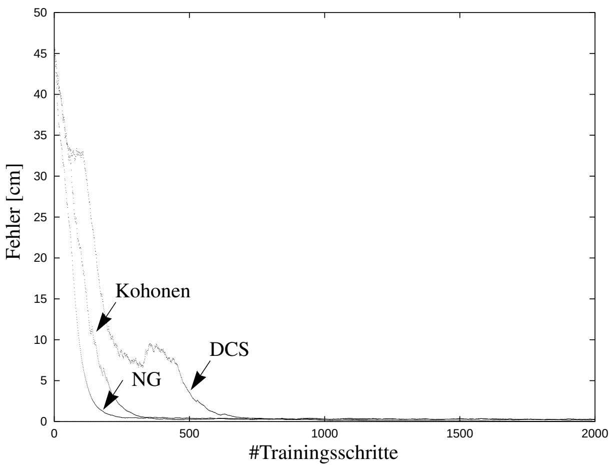


Abbildung 5.14: Positionierungsfehler im Arbeitsbereich für Kohonengitter, Neuronengas und DCS mit $3 \times 3 \times 3$ bzw. 30 Neuronen.

Einen ähnlichen Lernverlauf bezüglich der Anzahl der Trainingsschritte zeigt Abbildung 5.14, in der auf demselben Arbeitsbereich wesentlich kleinere Netze trainiert wurden (DCS und Neuronengas jeweils 30 Neuronen, $3 \times 3 \times 3$ Kohonenkarte). Zwar ist das Neuronengas immer noch das schnellste Netz, doch sind die Unterschiede in der Anzahl der benötigten Trainingsschritte weniger gravierend. Dies liegt daran, daß der Entfaltungsprozeß der Kohonenkarte kürzer ist und das DCS-Netz weniger laterale Verbindungen aufbauen muß. Interessant ist der Peak in der DCS-Lernkurve: Hier ist durch Ab- und Neuaufbau der lateralen Verbindungsstruktur die Zunahme des Positionierungsfehlers größer als seine durch fortdauernde Adaption der Ausgabeschicht bewirkte Abnahme.

Reales Experiment

Zielstellung des realen Experimentes ist es, die Übertragbarkeit der Simulationsresultate insbesondere des DCS-Netzes auf reale Einsatzbedingungen zu testen. Als Hardware dient der Stäubli RX90 Roboterarm mit einem am Manipulator befestigten Lämpchen, zwei Sony CCD Kameras mit einer Auflösung von je 768×576 Pixeln (durch Unterabtastung auf 384×288 Pixel reduziert) und eine Sparc10 Workstation der Firma SUN inklusive zweier Framegrabber. Der Arbeitsbereich des Roboters umfaßt einen Bereich von $20 \times 40 \times 20\,cm^3$, die Kameras werden mit einem Abstand von je $1.5\,m$ zum Arbeitsbereich und einer Basislänge von $1\,m$ aufgestellt.

In einer Initialisierungsphase werden vom Roboter zufällig soviele Positionen im Arbeitsbereich angefahren, wie das Netz Neuronen hat. Die zugehörigen retinalen Koordinaten p_i und Gelenkwinkel θ_i dienen zur Initialisierung der zugehörigen Zentren respektive Ausgabegelenkwinkel (Einfügen von Prototypen). Die Matrizen A_i werden mit kleinen, zufälligen Einträgen belegt. Diese Initialisierung verhindert, daß der Roboterarm in der Anfangsphase des Trainings erratische Stellwinkel übermittelt bekommt, die entweder außerhalb des Arbeitsbereiches liegen, nicht anfahrbar sind oder gar zu Zerstörungen führen.

Zur Generierung der Zielpunkte in der Trainingsphase wird der Roboterarm selbst verwendet. Unter Ausnutzung der bekannten inversen Kinematik wird jeweils ein Punkt im Arbeitsbereich zufällig ausgesucht und angefahren. Nun wird auf “Neuronale Steuerung” umgeschaltet, die retinalen Koordinaten werden registriert und dem Netz zur Generierung der Stellwinkel θ^0 eingespeist. Nach Anfahren von θ^0 wird wie in der Simulation eine Korrekturbewegung zur Position θ^1 durchgeführt, die dem Training der Jacobimatrizen dient.

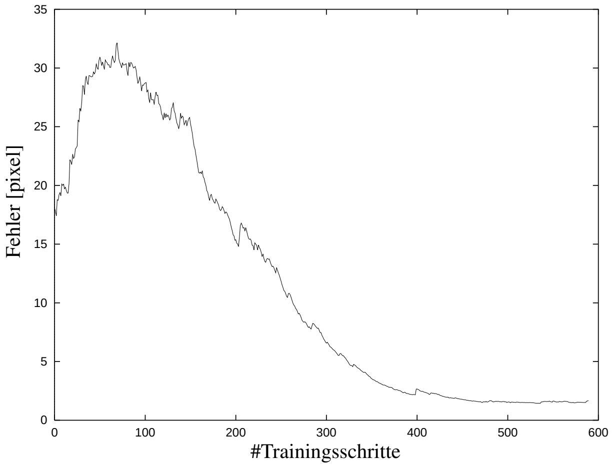


Abbildung 5.15: Positionierungsfehler in Pixel eines DCS-Netzwerkes mit 30 Neuronen in realer Arbeitsumgebung

Abbildung 5.15 zeigt den Positionierungsfehlerverlauf in Pixel für ein DCS-Netz mit 30 RBF-Knoten[10]. Nach ca. 500 Trainingsschritten erreicht das Netz einen minimalen Fehler von 1.4 Pixel. Dieser durch die Auflösung der Kameras und die Detektionsgenauigkeit des Lämpchens bedingte Restfehler kann auch von größeren Netzen nicht unterschritten werden. Erweiterte Kohonenkarten und Neuronengas verhalten sich qualitativ wie in der Simulation. Mit einem $3 \times 3 \times 3$ Gitter und einem 30-Neuronen-Gas wurde derselbe Restfehler in 400 bzw. 300 Schritten erreicht.

Ein interessanter Aspekt des Experiments mit dem realen Roboterarm wie auch der Simulation ist, daß sämtliche 6 Freiheitsgrade des Stäubli RX90 genutzt werden. Die Jacobi-Matrizen A_i werden damit 6×4 dimensional, und die Aufgabe, einen Punkt im Arbeits-

[10] Die Parameter für dieses Experiment finden sich in Tabelle 8.10.

bereich anzufahren, wird durch die redundanten Freiheitsgrade des Roboterarms "schlecht gestellt". In ihren Simulationen zur visuomotorischen Koordination eines Roboterarms konnten Ritter et al. zeigen, daß aufgrund der Nachbarschaftskooperation im erweiterten Modell von Kohonen benachbarte Gitterknoten "gezwungen" werden, ihre Ausgabegrößen (Jacobi-Matrizen und Gelenkwinkel) hin zu ähnlichen Werten zu adaptieren, und so die Ausgabegrößen nur wenig variieren. Das Resultat ist eine "glatte" Bewegung des Roboterarmes, wenn dieser im Arbeitsbereich eine Reihe von benachbarten Punkten anfahren soll. Die Wirkung der Nachbarschaftskooperation kann damit als eine Regularisierung des Koordinationsproblems gedeutet werden. Im Experiment zeigt sich, daß derselbe Effekt auch für das trainierte DCS-Netz zu beobachten ist: Beim Abfahren einer Linie im Arbeitsbereich (definiert durch eine Folge benachbarter Zielpunkte) ergibt sich ebenfalls eine glatte Armbewegung. Dies weist darauf hin, daß die Einer-Nachbarschaft der Lernregel nach Fritzke für den Nachbarschaftskooperationseffekt bereits ausreichend ist und auch die reine Gradientenregel zur Adaption der Ausgabegrößen (5.38) dem nicht entgegenwirkt.

5.3.4 Anmerkungen

Zum Abschluß der Betrachtungen und Experimente zur kalibrierungsfreien Hand-Auge-Koordination seien an dieser Stelle noch ein paar Anmerkungen zum Vergleich der drei Netztypen und dem Ritterschen Ansatz im allgemeinen angebracht.

Wie schon mehrfach betont, sollen diese Versuche weniger die generelle Überlegenheit des einen über das andere KNN-Modell als vielmehr ihre unterschiedliche Arbeitsweise demonstrieren. Hierzu wurden die Experimente unter vereinheitlichten Rahmenbedingungen durchgeführt, die bewußt die Stärken gerade des DCS-Netzes nicht nutzten. Auf fehlergesteuertes Wachstum oder Vorinitialisierung der lateralen Verbindungsstruktur des DCS-Netzes wurde verzichtet. Da für die Approximation in den ausgewählten Arbeitsbereichen eine gleichförmige Verteilung der Zentrumsvektoren die besten Ergebnisse zu liefern scheint, sind die selbstorganisierenden Lernregeln zur Generierung einer dichteabhängigen Zentrenverteilung auch die geeignetsten. Mit einer uniformen Eingabedichte erzeugen sie gerade eine gleichförmige Zentrenverteilung. Eine gleichförmige ist jedoch im allgemeinen nicht die beste Zentrenverteilung, sie sollte vielmehr am lokalen Approximationsfehler orientiert sein (Gleichverteilung bezüglich des lokalen Approximationsfehlers).

Abschließend sei darauf hingewiesen werden, daß die Ritterschen Versuche nicht nur die Vorteile von KNN in der Robotersteuerung demonstrieren, sondern auch deren größte Beschränkung: Alle Lernverfahren setzen voraus, daß die zu lernenden Funktionen stetig und möglichst glatt sind. Dies ist innerhalb der sorgsam ausgewählten Arbeitsbereiche auch der Fall, doch größere Arbeitsbereiche werden immer Singularitäten umfassen, an denen kleine Änderungen der Manipulatorposition sehr große Änderungen der Gelenkwinkel erfordern (z.B. ein Kippen des "Ellbogen" um 180°). An diesen Stellen ist die Steuerungsfunktion unstetig, die Jacobi-Matrizen sind schlecht konditioniert und die eingesetzten Netze versagen zwangsläufig. Eine "neuronale" Lösung dieses Problems steht aus.

Kapitel 6

Reinforcement-Lernen mit DCS

Gegenstand dieses Kapitels ist das Reinforcement-Lernen und der Einsatz von DCS-Netzen im Zusammenhang mit diesem Lernparadigma. Zunächst erfolgt eine allgemeine Einführung (Abschnitt 6.1) in das für die Robotik grundlegende Problem des Reinforcement-Lernens, in dem der Agent nur ein Bewertungssignal seiner Aktion(en) bekommt, nicht aber wie im Falle des überwachten oder des Fehlerrückkopplungs-Lernens einen direkten Hinweis, wie er besser agieren könnte. Anschließend werden in Abschnitt 6.2 und 6.3 die beiden Hauptvarianten dieses Problems vorgestellt: Reinforcement-Lernen in Markovschen Umgebungen als das diskrete Markovsche Entscheidungsproblem, falls Zustands- und Aktionsraum abzählbar sind, und Reellwertiges Reinforcement-Lernen, falls Zustands- und Aktionsraum reellwertig sind. Für beide Varianten werden DCS basierte Lernarchitekturen vorgestellt, deren besondere Vorteile aber auch Begrenzungen diskutiert und anhand zweier konkreter Anwendungen demonstriert werden. Besondere Merkmale der vorgestellten Architekturen sind

- die Verwendung von Mehrschrittprädiktoren nach dem in Abschnitt 6.2.3 vorzustellenden TD(λ)-Verfahren zur Beschleunigung des Lernens,
- ihre Fähigkeit, mittels der Reinforcement-Signale nicht nur vorhandenes Wissen zu adaptieren sondern auch neues Wissen zu generieren,
- ihre Kombinationsfähigkeit mit überwachtem Lernen oder Fehlerrückkopplungs-Lernen,
- ihr potentieller Beitrag zur Vermeidung des "Curse of Dimensionality" bei höherdimensionalen Zustandsräumen und
- ihr Verzicht auf explizite Weltmodellierung.

Die Anwendungen sind im Rahmen des diskreten Reinforcement-Lernens das Benchmarkproblem "Stabbalancieren" [89] und im Rahmen des reellwertigen Reinforcement-Lernens das Erlernen eines hindernisvermeidenden Navigationsverhaltens für einen autonomen mobilen Roboter.

Als "Nebenprodukt" wird des weiteren eine neue Variante des Q-Lernens vorgestellt, das $Q^\lambda(m)$-Lernen, das eine Synthese zwischen Suttons TD-Lernen [9] und Watkins' Q-Lernen [133] darstellt.

Teile dieses Kapitels wurden in [23] (modifiziertes Q-Lernen zum Stabbalancieren) sowie [22, 24] (reellwertiges Reinforcement-Lernen zur Hindernisvermeidung) publiziert.

6.1 Reinforcement-Lernen

Ähnlich dem überwachten oder dem Fehlerrückkopplungs-Lernen erhält der Lerner beim Reinforcement-Lernen ein Rückkopplungssignal. Dieses ist jedoch im Gegensatz zu den erst genannten Verfahren nicht *instruktiv*, also mit einem Hinweis zur Verbesserung verbunden, sondern rein *evaluativ* (bewertend) [97]. In der Hierarchie der Lernverfahren ist Reinforcement-Lernen damit qualitativ zwischen den unüberwachten Lernverfahren (keine Rückkopplung) und dem Fehlerrückkopplungs-Lernen einzuordnen. Es bietet einen allgemeineren Rahmen als die instruktiven Lernverfahren und besitzt ein weites Anwendungsfeld insbesondere im Kontext autonomer lernender Agenten, die kompetentes Verhalten autonom (ohne Lehrer) erlernen oder adaptieren müssen. Die Tatsache, daß Reinforcement-Lernen bei biologischen Systemen eine herausragende Rolle spielt, macht dieses Lernparadigma zusätzlich interessant. So wird Reinforcement-Lernen auch von der experimentellen Psychologie zur Modellierung des Verhaltenserwerbes bei Tieren und Menschen genutzt.

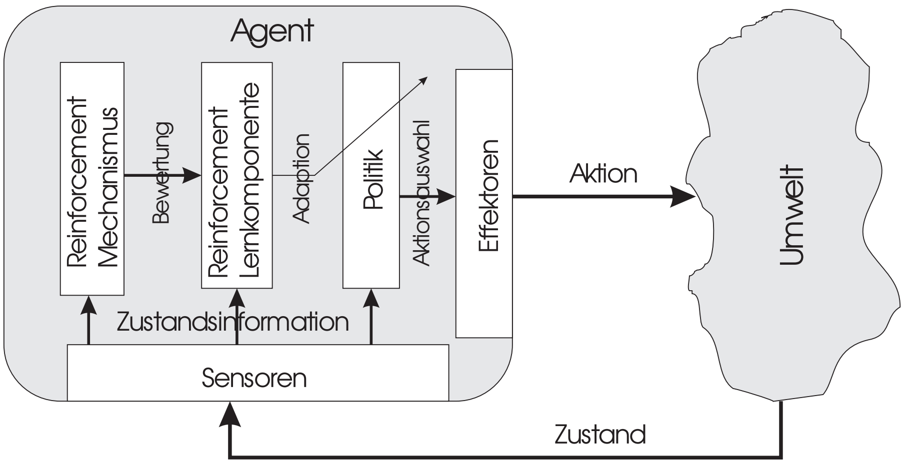


Abbildung 6.1: Szenario des Reinforcement-Lernens

Das Paradigma des Reinforcement-Lernens im Kontext situierter Agenten veranschaulicht Abbildung 6.1. Der Agent besitzt *Sensoren*, durch die er den Zustand s_t der Umgebung wahrnehmen kann, *Effektoren*, durch die er die Umgebung verändern kann, eine *Politik* (engl. control policy) π, die sensorische Situationen auf Aktionen abbildet, einen *Reinforcement-Mechanismus*, der nach jeder Aktion eine Bewertung der Aktion vornimmt (das Reinforcement- Signal r_t generiert), und eine *Reinforcement-Lernkomponente*, die auf der Basis des Reinforcement-Signals die Politik adaptiert. Das Ziel ist, eine Politik zu lernen, die eine bestimmte *Ertragsfunktion* maximiert. Die im Rahmen von Reinforcement-Lernen am häufigsten anzutreffende Ertragsfunktion ist das diskontierte kumulative Reinforcement (engl. discounted cumulative reinforcement)

$$V^\pi(s) = E^\pi[\sum_{t=0}^{\infty} \gamma^t r(\pi(s_t)|s_t)|s_0 = s], \tag{6.1}$$

von dem auch im folgenden ausgegangen wird. Hierbei bezeichnet $r(\pi(s_t)|s_t)$ das Reinforcement-Signal, das der Agent bei Anwendung der Aktion $\pi(s_t)$ im Zustand s_t erhält, und E^π den Erwartungswert unter der Politik π. Der Faktor $0 \leq \gamma < 1$ sorgt bei beschränktem Reinforcement-Signal für eine Beschränkung der Summe. Für $\gamma = 0$ spricht man vom Ein-Schritt Reinforcement-Lernen. Alternative Ertragsfunktionen sind denkbar und gegebenenfalls vorteilhaft, siehe hierzu [52].

Im Gegensatz zum überwachten Lernen bedarf das Reinforcement-Lernen keines Lehrers (Instruktors), sondern erfordert lediglich die Aufstellung des Reinforcement-Mechanismus (der Aktionsbewertung). Die Aufstellung des Reinforcement-Mechanismus zum Erlernen komplexerer Verhaltensweisen stellt allerdings ein nicht zu unterschätzendes Problem dar, für das bislang keine systematischen Zugänge existieren.

Erkundung/Nutzung-Dilemma

Da der Reinforcement-Lerner keine Instruktionen erhält, wie er sein Verhalten zu verbessern hat, ist er auf die Erkundung seiner Umwelt und das Ausprobieren verschiedener Aktionen im selben Zustand angewiesen. Er wird dann seine Politik in Richtung der Aktionen adaptieren, die einen Zuwachs der Ertragsfunktion bewirken. Gerade in stochastischen Umgebungen ist es theoretisch nötig, jede Aktion sogar unendlich oft auszuprobieren. Auf der anderen Seite kann es sein, daß der Agent bereits eine akzeptable Politik gefunden hat und weiteres Experimentieren mehr schadet als nützt. Dieses grundlegende Dilemma wird als *Erkundung/Nutzung-Dilemma* (engl. exploration/ exploitation dilemma) bezeichnet. Es wird typischerweise dadurch gelöst, daß man die Neigung zur Erkundung mit der Zeit abnehmen läßt. Dadurch allerdings verliert das System mit der Zeit jegliche Adaptivität, das eng mit dem Erkundung/Nutzung-Dilemma verbundene Stabilität/Plastizität-Dilemma wird auf Kosten der Plastizität gelöst.

Das Problem der zeitlichen Zuordnung von verzögerten Bewertungen

Oft zeigt eine Aktion erst später Auswirkungen, und der Agent erhält erst nach einer ganzen Reihe von Aktionen ein Bewertungssignal (ein Schachspieler möglicherweise erst am Ende der Partie). Hier entsteht das *Problem der zeitlichen Zuordnung verzögerter Bewertungen* (engl. temporal credit assignment problem). Die Ertragsfunktion nach (6.1) ist ein Ansatz zur Lösung dieses Problems, der jedoch für jede Anwendung neu auf seine Eignung überprüft werden muß.

Das Problem der strukturellen Zuordnung von Bewertungen

Ist das Problem der zeitlichen Zuordnung gelöst, bleibt weiter das *Problem der strukturellen Zuordnung* (engl. structural credit assignment problem) zu lösen. Dieses Problem beschreibt, wie die Parameter (Gewichte) als Reaktion auf das Bewertungssignal zu ändern sind, und besteht für alle Lernparadigmen. Für überwachtes Lernen wird es im Zusammenhang mit KNN üblicherweise durch Backpropagation [114] gelöst.

Verdeckte Zustände

Ein weiteres schwer zu lösendes Problem bereitet die Existenz verdeckter Zustände (engl. hidden states). Es tritt immer dann auf, wenn die Sensorik des Roboters nicht ausreicht, um den Zustand der Welt vollständig zu erfassen. Im Rahmen von Reinforcement-Lernen mit KNN bieten hier rekurrente Netze einen Ansatz (siehe [76]). In den folgenden Anwendungen wird stets voller Zugriff auf den Zustand vorausgesetzt.

Direktes versus indirektes Reinforcement-Lernen

Grundsätzlich gibt es zwei Ansätze zum Erlernen optimaler Politiken. Beim *indirekten* oder *modellbasierten* Ansatz wird zuerst (oder parallel) ein Modell der Wechselwirkung des Agents mit der Umwelt erlernt und dieses dann zur Adaption der Politik genutzt. Suttons Dyna-Architektur [121] ist ein Beispiel für einen indirekten Ansatz. Der *direkte* Ansatz zeichnet sich durch Verzicht auf explizite Weltmodellierung aus, die Adaption der Politik erfolgt allein aufgrund der wahrgenommenen Bewertungssignale. Ein typischer Vertreter ist das im folgenden noch ausführlich zu diskutierende Q-Lernen von Watkins [133]. Der indirekte Ansatz nimmt einen Umweg über ein eventuell schwer erlernbares Weltmodell in Kauf, erlaubt aber auf dessen Basis eine schnelle, nur durch die verfügbare Rechenzeit begrenzte Adaption durch Simulation. Die Lerngeschwindigkeit beim direkten Ansatz hingegen ist von der tatsächlichen Zeitskala abhängig, da mit jedem Perzeptions-Aktionszyklus jeweils nur ein Adaptionsschritt erfolgt. Es bleibt anzumerken, daß der indirekte Ansatz keines der oben aufgelisteten Probleme vermeidet. Insbesondere löst er nicht das Erkundung/Nutzung-Dilemma.

Diese Arbeit orientiert sich am Paradigma der verhaltensbasierten Robotik, das einen weitestgehenden Verzicht auf Weltmodellierung fordert und “die Welt als ihr eigenes Modell” nutzt ([18], [118]). Es werden daher ausschließlich die direkten Ansätze diskutiert.

Skalierungsverhalten

Reinforcement-Lernverfahren, die wie das in Abschnitt 6.2.2 vorzustellende Q-Lernen auf einer tabularen Funktionsrepräsentation beruhen, skalieren exponentiell mit der Dimension des Zustandsraumes. Da das Erlernen der zu einem Zustand gehörigen Aktion durch Ausprobieren und nicht durch Instruktion erfolgt, wirkt sich der Fluch der Dimensionalität besonders stark aus. Zur Lösung des Skalierungsproblems gibt es im wesentlichen zwei Ansätze. Zum einen ist dies die geschickt Partitionierung des Zustandsraumes, zum anderen die Verwendung von nicht-tabularen Funktionsapproximatoren wie KNN.

Hinsichtlich der geschickten Partitionierung bietet die hierarchische, Verhaltens-orientierte Partitionierung des Zustandsraumes nach dem "Teile und Herrsche"-Prinzip einen natürlichen Lösungsweg. Hierbei wird auf der untersten Stufe eine Basismenge primitiver Verhalten (jeweils charakterisiert durch niedrigdimensionale Zustandsräume und direkten Durchgriff auf die Aktoren) gelernt. Auf höheren Stufen werden dann komplexere Verhalten als Koordination primitiverer Verhalten gelernt, wobei ein sensorischer Eindruck nicht direkt mit einer (motorischen) Aktion sonder der Auslösung eines untergeordneten Verhaltens assoziiert wird. Ein instruktives Beispiel für die Verhaltens-orientierte hierarchische Partitionierung im Zusammenhang mit Reinforcement-Lernen liefert [33].

Als direkte Konsequenz aus dem Bias-Variance-Dilemma (Abschnitt 2.1.2) macht Landelius [73] darauf aufmerksam, daß auch bei der Verwendung von nicht-tabularen Funktionsapproximatoren das mit der Begrenztheit der Trainingsmengen verbundene Problem der Varianz nur durch starke Vorannahmen und Einschränkungen (Einführung eines Bias) gelöst werden kann. Als minimalen und mit der vorliegenden Arbeit konsistenten Bias schlägt Landelius erstens die Stetigkeitsannahme und zweitens die Verwendung lokaler Modelle (speziell RBF-Netze) vor. In diesem Sinne stellt natürlich auch die oben angesprochene Verhaltens-orientierte Partitionierung nicht anderes als die Einführung eines weiteren Bias dar.

6.2 Reinforcement-Lernen als diskretes Markovsches Entscheidungsproblem

In diesem Abschnitt wird das Reinforcement-Lernen als zeit-diskretes Markovsches Entscheidungsproblem behandelt. Dies ist immer dann möglich, wenn Zustands- und Aktionsraum sowie die Zeitskala diskret sind und die Umwelt als Markov-Prozeß modellierbar ist. In Anlehnung an [10] wird dargestellt, daß in diesem Rahmen das Problem durch Dynamische Programmierung gelöst werden kann und sich Watkins' Q-Lernen [133] als stochastische Approximation der Dynamischen Programmierung ergibt. Im Anschluß daran wird Suttons AHC(λ)-Lernen zur Lösung desselben Problems vorgestellt. Es ist zwar weniger kompakt und seine Konvergenz gegen eine optimale Lösung läßt sich im Gegensatz zum Q-Lernen im allgemeinen nicht beweisen, praktisch konvergiert es jedoch aufgrund der Verwendung von TD(λ)-Prädiktoren oft schneller als Q-Lernen. Als ein erster Kombinationsversuch von Q-Lernen und TD(λ)-Lernen wird anschließend MQ-Lernen vorgestellt, das wie Q-Lernen auf einer tabularen Funktionsrepräsentation beruht. Mit dem QDCS-Lernen wird letztere durch eine DCS-basierte Repräsentation ersetzt. Während diese beiden Lernverfahren hinsichtlich der Lösung des Stabbalanceproblems optimiert wurden, wird schließlich ein drittes Lernverfahren zur Lösung des generellen Markovschen Entscheidungsproblems vorgestellt, das $Q^{\lambda}(m)$-Lernen. Den Exkurs ins diskrete Reinforcement-Lernen beendet eine Übersicht über alternative Ansätze zur Kombination von Q-, TD(λ)-Lernen und KNN.

Definition 6.1 (zeit-diskreter Markovscher Entscheidungsprozeß) *Ein* zeit-diskreter Markovscher Entscheidungsprozeß *ist ein 5-tupel* $(\Delta t, S, U, P, r)$*, wobei*

- Δt *die Diskretisierung der Zeitachse angibt,*
- $S = \{1, \ldots, n\}$ *eine endliche Menge von Zuständen ist,*
- $U = \{1, \ldots, m\}$ *eine endliche Menge von Aktionen bezeichnet,*
- $P : S \times S \times U \rightarrow \mathbb{R}$ *eine probabilistische Zustandsübergangsfunktion ist, mit* $P_{ij}(u)$ *der Übergangswahrscheinlichkeit von Zustand* $i \in S$ *nach Zustand* $j \in S$ *nach* Δt *bei Ausführung der Aktion* $u \in U$ *mit* $\sum_j P_{ij}(u) = 1 \; \forall i \in S$*, und*
- $r : S \times U \rightarrow \mathbb{R}$ *eine Bewertungssfunktion ist, die den Wert der (unverzögerten) Bewertung* $r(i, u)$ *mit Erwartungswert* $R(i, u)$ *bei Ausführung von Aktion* $u \in U$ *in Zustand* $i \in S$ *angibt.*

Definition 6.2 (stationäre Politik) *Eine* stationäre Politik *ist eine Abbildung* $\pi : S \rightarrow U$*, die jedem Zustand* $i \in S$ *eine Aktion* $u \in U$ *zuordnet.*

Definition 6.3 (zeit-diskretes Markovsches Entscheidungsproblem) *Gegeben einen zeit-diskreten Markovschen Entscheidungsprozeß, so besteht das* zeit-diskrete Markovsche Entscheidungsproblem *(MDP) darin, eine stationäre Politik* π *zu finden, welche die Ertragsfunktion*

$$V^{\pi}(s) = E^{\pi}[\sum_{t=0}^{\infty} \gamma^t r(\pi(s_t)|s_t)|s_0 = s] \tag{6.2}$$

für alle Zustände $s \in S$ *maximiert* [1]*. Eine solche Politik wird im folgenden als* optimale Politik *bezeichnet, die zugehörige Ertragsfunktion als* Optimalwertfunktion .

6.2.1 Dynamische Programmierung

Der Schlüssel zur Lösung des MDPs durch Dynamische Programmierung liegt in der Tatsache, daß die Optimalwertfunktion V^{opt} dem *Bellmanschen Optimalitätskriterium* genügt [113],

$$V^{opt}(s) = \max_{u \in U}\{R(u|s) + \gamma \sum_{j \in S} P_{sj}(u) V^{opt}(j)\} \quad \forall s \in S. \tag{6.3}$$

Dies ist gleichbedeutend damit, daß eine optimale Politik *gierig* bezüglich ihrer Optimalwertfunktion ist. In jedem Zustand ist diejenige Aktion zu wählen, die den Erwartungswert der zukünftigen Bewertung maximiert. Zur Bezeichnung des Erwartungswertes der zukünftigen Bewertung unter einer bestimmten Politik in Abhängigkeit vom Zustand und der ausgeführten Aktion führte Watkins in [133] die Q-Notation ein:

$$Q^{\pi}(s, u) := R(u|s) + \gamma \sum_{j \in S} P_{sj}(u) V^{\pi}(j) \quad s \in S, u \in U. \tag{6.4}$$

[1] Die Maximierung einer Ertragsfunktion ist offensichtlich äquivalent mit der Minimierung einer Kostenfunktion, falls nur r_t durch $-r_t$ ersetzt wird.

Mit der Q-Notation läßt sich das Bellmansche Optimalitätskriterium für MDP als

$$V^{opt}(s) = \max_{u \in U}\{Q^{opt}(s,u)\} \quad \forall s \in S. \tag{6.5}$$

schreiben. Die Gültigkeit des Bellmanschen Optimalitätskriteriums impliziert, daß Dynamische Programmierung zur sukzessiven Approximation der Optimalwertfunktion (und damit der implizit gegebenen optimalen Politik) eingesetzt werden kann [113]. Die rekursive Berechnungsvorschrift lautet

$$V_n(s) = \max_{u \in U}\{R(u|s) + \gamma \sum_{j \in S} P_{sj}(u) V_{n-1}\} = \max_{u \in U}\{Q_{n-1}(s,u)\}, \tag{6.6}$$

wobei die Startwerte $V_0(s)$ beliebige (beschränkte) Werte annehmen dürfen. Je nach Art der Zustandsmenge, für die im Schritt $n-1 \rightarrow n$ die Werte aktualisiert werden, unterscheidet man im wesentlichen drei Varianten der Dynamischen Programmierung [10]:

- *Synchrone Werte-Iteration* (engl. synchronous value iteration), falls die Menge alle Zustänge umfaßt,
- *Gauß-Seidel DP*, falls die Menge jeweils nur den ($n+1$ modulo $|S|$).ten Zustand enthält,
- *Asynchrone DP*, falls die Menge jeweils eine Teilmenge von S enthält, die gewissen Randbedingungen genügen muß.

In allen drei Fällen konvergiert (6.6) gegen die Optimalwertfunktion für $n \rightarrow \infty$, [10, 14]. Zu einem bestimmten Zeitpunkt wird es jedoch im allgemeinen nicht möglich sein, die Entfernung vom Optimum abzuschätzen. Insbesondere implizieren die V_n-Werte keine Politik, doch wird üblicherweise gierig bezüglich der $Q_n(s,u)$-Werte gehandelt werden. Wird die Regelung auf diese Weise ausgeführt, d.h. parallel zur Lösung des MDPs durch DP nach dem jeweiligen Wissensstand gehandelt, so spricht man von *Dynamischer Programmierung in Echtzeit* (engl. real-time dynamic programming). Die Dynamische Programmierung läuft im Hintergrund, während der Agent in Echtzeit handelt.

Die bislang vorgestellten DP-Methoden gingen von vollständiger und genauer Kenntnis des zugrundeliegenden Markovschen Entscheidungsproblems aus, insbesondere der Kenntnis der Übergangswahrscheinlichkeiten $P_{ij}(u)$ und der Erwartungswerte der Bewertungssignale $R(u)$. Sind diese nicht bekannt, so können *adaptive* DP-Verfahren aus dem Bereich der stochastischen Approximation eingesetzt werden. Hier werden zwei Klassen von Verfahren unterschieden [10], die indirekten und direkten Methoden (siehe auch Abschnitt 6.1). Bei den *indirekten* Verfahren werden die Übergangswahrscheinlichkeiten und mittleren Bewertungssignale geschätzt, um dann auf die oben angeführten DP Methoden auf Grundlage der aktuellen Schätzer zurückgreifen zu können. Dies führt zur *adaptiven Dynamischen Programmierung in Echtzeit* [10]. Gullapalli und Barto beweisen in [45] folgenden wichtigen Satz zur Konvergenz von indirekter adaptiver Dynamischer Programmierung:

Satz 6.1 (Gullapalli 93) *Für jeden Markovschen Entscheidungsprozeß konvergiert ein beliebiger indirekter adaptiver asynchroner DP-Algorithmus (für den der nicht-adaptive Algorithmus konvergiert) gegen die optimale Kostenfunktion mit Wahrscheinlichkeit 1, falls*

1. *die Bedingungen für die Konvergenz des nicht-adaptiven DP-Algorithmus erfüllt sind,*

2. *im Grenzübergang jede Aktion unendlich oft ausgeführt und jeder Zustand unendlich oft besucht wird und*

3. *die Schätzer der Übergangswahrscheinlichkeiten und der erwarteten Bewertungssignale begrenzt bleiben und für $n \to \infty$ gegen ihre wahren Werte mit Wahrscheinlichkeit 1 konvergieren.*

Im Gegensatz zu den indirekten Verfahren werden bei den *direkten* adaptiven Verfahren ohne Umweg über Schätzung der Übergangswahrscheinlichkeiten und Bewertungssignale Schätzer für die Q^{opt}-Werte entwickelt. Der prominenteste Vertreter dieser Gattung ist das Q-Lernen.

6.2.2 Q-Lernen

Q-Lernen [133] ist ein direktes adaptives Verfahren zur Lösung des zeit-diskreten Markovschen Entscheidungsproblems in Echtzeit. Es basiert allein auf Schätzung der Q^{opt}-Werte und ist damit modellfrei. Der Agent führt im Zustand $s_t \in S$ eine Aktion $u_t \in U$ aus, die ein Bewertungssignal $r(u_t|s_t)$ bewirkt und den Agenten in den Zustand s_{t+1} bringt. Die Q-Schätzer (Schätzer der Q^{opt}-Werte) werden nach der Lerngleichung

$$\begin{aligned} Q_{t+1}(s_t, u_t) &= Q_t(s_t, u_t) + \alpha_t(s_t, u_t)(r(u_t|s_t) + \gamma V_t(s_{t+1}) - Q(s_t, u_t)) && (6.7) \\ &= Q_t(s_t, u_t) + \alpha_t(s_t, u_t)(r(u_t|s_t) + \gamma \max_{u \in U} Q_t(s_{t+1}, u) - Q(s_t, u_t)) && (6.8) \end{aligned}$$

berechnet. Der Q-Lernalgorithmus ist in Abbildung 6.2 zusammengefaßt.

Zufällige Initialisierung der $Q(s, u)$-Schätzer $\forall s \in S, u \in U$;
$t = 0$; $s_0 =$ Startzustand;
do {
  Wähle u_t nach Explorationsstrategie;
  Führe Aktion u_t aus; registriere r_t und s_{t+1};
  $Q(s_t, u_t) := Q(s_t, u_t) + \alpha_t(r_t + \gamma \max_{u \in U} Q(s_{t+1}, u) - Q(s_t, u_t))$;
  $t := t + 1$;
} **loop**;

Abbildung 6.2: Q-Lernen

Unter der Vorraussetzung, daß alle Q-Schätzer unendlich oft aktualisiert werden (also jeder Zustand unendlich oft erreicht und in ihm jede Aktion unendlich oft durchgeführt wird), konnte Watkins in [134] folgenden Satz zur Konvergenz des Q-Lernens beweisen:

Satz 6.2 (Watkins) *Für beschränkte Bewertungssignale* $|r_t| \leq C$ *und Lernraten* $0 \leq \alpha_t < 1$ *mit*

$$\sum_t \alpha_t(s,u) = \infty,\ \sum_t \alpha_t^2(s,u) < \infty\ \forall s \in S, u \in U \tag{6.9}$$

konvergiert $Q_t(s,u) \rightarrow Q^{opt}(s,u)$ *mit* $t \rightarrow \infty$ $\forall s \in S, u \in U$ *mit Wahrscheinlichkeit 1.*

Watkins' ursprünglicher Beweis ist recht langwierig. Jaakola et al. [62] gelang es, Q-Lernen als stochastischen Prozeß zu beschreiben und die Konvergenz elegant im Rahmen der stochastischen Approximationstheorie nachzuweisen. Sie bewiesen hierzu folgenden wichtigen Satz, der die von Robbins und Monro begründete stochastische Approximationstheorie auf die Maximumsnorm ausdehnt:

Satz 6.3 (Jaakola et al. 94) *Ein iterativer Zufallsprozeß der Form*

$$\Delta_{n+1}(x) = (1 - \alpha_n(x))\Delta_n(x) + \beta_n(x)F_n(x)$$

konvergiert gegen null mit Wahrscheinlichkeit 1 falls:

1. *der Zustandsraum endlich ist,*
2. $\sum_n \alpha_n(x) = \infty$, $\sum_n \alpha_n^2(x) \leq \infty$, $\sum_n \beta_n(x) = \infty$, $\sum_n \beta_n^2(x) \leq \infty$ *und* $E\{\beta_n(x)|P_n\} \leq E\{\alpha_n(x)|P_n\}$,
3. $\| E\{F_n(x)|P_n\} \|_W \leq \gamma \| \Delta_n \|_W$ *mit* $\gamma \in (0,1)$ *und*
4. $Var\{F_n(x)|P_n\} \leq C(1 + \| \Delta_n \|_W)^2$, *mit* C *konstant.*

Dabei steht $P_n = \{\Delta_n, \Delta_{n-1}, \ldots, F_{n-1}, \ldots, \alpha_{n-1}, \ldots, \beta_{n-1}, \ldots\}$ *für die Geschichte zum Zeitpunkt* n. $F_n(x)$, $\alpha_n(x)$ *und* $\beta_n(x)$ *dürfen von der Geschichte abhängen, solange die obigen Bedingungen erfüllt bleiben. Die Schreibweise* $\| \,.\, \|_W$ *bezeichnet eine beliebige gewichtete Maximumsnorm.*

In der Anwendung dieses Satzes stellt der Prozeß Δ_n die Differenz zwischen dem stochastischen Prozeß, dessen Konvergenz zu beweisen ist, und seinem Optimalwert dar. Für Q-Lernen ist die Existenz des Optimalwertes durch die Ergebnisse der Dynamischen Programmierung gesichert. Mit Gleichsetzung von

$$\begin{aligned} \Delta_t(s,u) &:= Q_t(s,u) - Q^{opt}(s,u) \\ F_t(s,u) &:= r(u|s) + \gamma V_t(s_{t+1}) - Q^{opt}(s,u) \end{aligned}$$

muß jetzt nur noch die Kontraktionseigenschaft von $F_t(s,u)$ gezeigt werden. Diese folgt aus (siehe [62])

$$\max_u \| E\{F_t(s,u)\} \| = \gamma \max_u \| \sum_j P_{sj}(u)[V_t(j) - V^{opt}(j)] \| \tag{6.10}$$

$$\leq \gamma \max_u \sum_j P_{sj}(u) \max_v \| Q_t(j,v) - Q^{opt}(j,v) \| \tag{6.11}$$

$$\leq \gamma \max_j \max_v \| Q_t(j,v) - Q^{opt}(j,v) \| . \tag{6.12}$$

Die gleichfalls geforderte Beschränktheit der Varianz von $F_t(s, u)$ folgt aus der geforderten Beschränktheit der Varianz von $r(u|s)$.

Es seien an dieser Stelle die zum Vergleich mit Suttons AHC-Lernen wesentlichen Eigenschaften des Q-Lernens zusammengefaßt: Q-Lernen ist ein modellfreies, direktes adaptives Verfahren zur Lösung des zeit-diskreten MDP. Es kann als stochastisches Approximationsverfahren aufgefaßt werden, dessen Konvergenz gegen die Optimalwertfunktion garantiert ist. Die Konvergenzeigenschaft ist insbesondere unabhängig von der gewählten Explorationsstrategie (*Explorationsunabhängigkeit*, engl. exploration-insensitive, [97]), solange nur die Voraussetzungen von Satz 6.2 eingehalten werden. Es gibt keinen Unterschied zwischen dem Erlernen der Optimalwertfunktion und dem Erlernen der optimalen Politik. Neben diesen positiven Eigenschaften besitzt Q-Lernen jedoch den großen Nachteil der inhärenten Sequentialität. In jedem Schritt kann aufgrund des Maximumoperators jeweils nur ein Q-Schätzer aktualisiert werden, und das Bewertungssignal kann im allgemeinen nicht zur Aktualisierung von Q-Schätzern anderer (vergangener) Zustand/Aktion-Paare genutzt werden. Diesen Nachteil besitzt das im folgenden vorzustellende AHC(λ)-Lernen nicht.

6.2.3 AHC-Lernen

Das von Barto und Sutton eingeführte AHC-Lernen [9] steht für "Lernen mit einem Adaptiven Heuristischen Kritikelement" (engl. adaptive heuristic critique) und wurde wie das Q-Lernen zur modellfreien Lösung des zeitdiskreten MDP entwickelt. Im Unterschied zum Q-Lernen besitzt es neben einer Lernkomponente für die Bewertung von Zustand/Aktion-Paaren (der Politik-Lernkomponente) eine zweite Komponente zur Schätzung der Optimalwertfunktion (die Prädiktionskomponente). AHC-Lernen ist in Abbildung 6.3 veranschaulicht. Die Adaption der in Tabellenformat gespeicherten Schätzer für die Optimal-

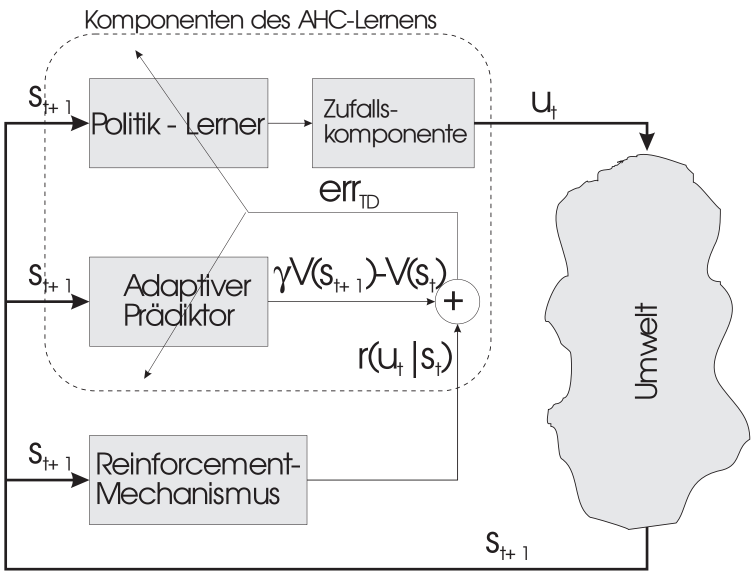


Abbildung 6.3: Prinzip des AHC-Lernens

wertfunktion $V(s)$ erfolgt nach der Lernregel

$$\begin{aligned} V_{t+1}(s_t) &= V_t(s_t) + \alpha_t \left(r(u_t|s_t) + \gamma V(s_{t+1}) - V(s_t)\right) && (6.13) \\ &= V_t + \alpha_t \, err_{TD}(s_t), && (6.14) \end{aligned}$$

wobei $V(s_t)$ die Schätzung der Optimalwertfunktion für den letzten Zustand des Agents bezeichnet und $V(s_{t+1})$ die Schätzung für den Zustand, den der Agent nach Ausführung der Aktion u_t im Zustand s_t erreicht. Der *TD-Fehler* (von engl. temporal difference error, TD-error)

$$err_{TD}(s_t) = r(u_t|s_t) + \gamma V_t(s_{t+1}) - V_t(s_t) \qquad (6.15)$$

gibt dabei den Unterschied zwischen dem alten Schätzwert $V_t(s_t)$ und dem TD-Schätzer $r(u_t|s_t) + \gamma V_t(s_{t+1})$ aufgrund des Bewertungssignals r und dem Schätzwert $V(s_{t+1})$ für den neuen Zustand an.

Der TD-Fehler wird auch zur Adaption der gleichfalls in Tabellenformat gespeicherten Bewertungsfunktion der Zustand/Aktion-Paare, der Politik-Lernkomponente, genutzt. Die Lernregel lautet hier

$$\Delta q_{t+1}(s_t, u_t) = q_t(s_t, u_t) + \beta_t \, err_{TD}(s_t). \qquad (6.16)$$

Wie im Falle des Q-Lernens wird der Agent in zunehmendem Maße die Aktionen bevorzugen, deren $q(s,u)$-Werte einen höheren Nutzen signalisieren, jedoch stets weiter durch Auswahl zufälliger Aktionen den Effekt jeder Aktion in jedem Zustand erkunden müssen.

Zufällige Initialisierung der $V(s)$ und $q(s,u)$ Schätzer $\forall s \in S, u \in U$;
$t = 0$; $s_0 =$ Startzustand;
do {
  Wähle u_t nach Explorationsstrategie;
  Führe Aktion u_t aus; registriere r_t und s_{t+1};
  $err_{TD} = r(u_t|s_t) + \gamma V(s_{t+1}) - V(s_t)$;
  $V(s_t) := V(s_t) + \alpha_t \, err_{TD}$;
  $q(s_t, u_t) := q(s_t, u_t) + \beta_t \, err_{TD}$;
  $t := t + 1$;
} **loop**;

Abbildung 6.4: AHC-Lernen: Algorithmus

Die Einführung getrennter Lernkomponenten für die Optimalwertfunktion und die Politik hat die schwerwiegende Konsequenz, daß es im allgemeinen nicht mehr möglich ist, die Konvergenz der $V(s)$-Werte gegen $V^{opt}(s)$ bzw. der $q(s,u)$-Werte gegen $Q^{opt}(s,u)$ zu zeigen. Ob AHC-Lernen gegen eine Lösung des MDP konvergiert oder nicht, hängt von der Explorationsstrategie des Agenten ab (engl. experimentation sensitiveness), über die aber a priori keine Aussage getroffen werden kann. Diese beiden Nachteile, die fehlende Konvergenzeigenschaft und die Abhängigkeit von der Explorationsstrategie, haben zu einer weitgehenden Verdrängung von AHC-Lernen durch Q-Lernen geführt. Im Gegensatz zum Q-Lernen bietet AHC-Lernen in Kombination mit Suttons TD(λ)-Lernen allerdings die Möglichkeit, in einem Zeitschritt nicht nur die V- und q-Werte des letzten Zustandes, sondern aller bisher besuchten Zustände und Zustand/Aktion-Paare auf sehr effiziente Weise zu adaptieren. Diese Kombination, obwohl ebenfalls nicht beweisbar konvergent, zeigt in der Praxis häufig ein schnelleres und besseres Konvergenzverhalten als das Q-Lernen. Hierauf wird im folgenden näher eingegangen.

TD(λ)-Lernen

TD(λ)-Lernen wurde von R. Sutton in seinem klassischen Aufsatz [122] zur Lösung von Mehrschrittprädiktionsproblemen eingeführt. Im Gegensatz zu Einschritt-Prädiktionsproblemen, bei denen die Bekanntgabe des korrekten Wertes nach jedem Zeitschritt erfolgt, wird bei der Mehrschrittprädiktion das endgültige Ergebnis (der korrekte Wert) z erst nach mehreren Zeitschritten bekannt. Der Lerner besitzt eine differenzierbare Prädiktionsfunktion $P(s_t, w)$, die abhängig vom momentanen Zustand s_t und dem zu adaptierenden Parametervektor w zu jedem Zeitpunkt eine Prädiktion $P_t = P(s_t, w)$ des endgültigen Ergebnisses liefern soll.
Eine Adaption von w ist durch Gradientenabstieg auf der Summe der quadratischen Prädiktionsfehler durch

$$w_{n+1} = w_n + \alpha \frac{\partial}{\partial w}[\frac{1}{2}\sum_{t=1}^{T}(z - P_t)^2] = w_n + \alpha \sum_{t=1}^{T}(z - P_t)\nabla_w P_t \tag{6.17}$$

möglich. Nachteil dieses Verfahrens ist, daß sämtliche Berechnungen erst bei Bekanntwerden des Endergebnisses ausgeführt werden können. So erscheint die Speicherung aller Prädiktionswerte und aller Zustände über die Zeit notwendig. Sutton erkannte, daß sich durch einfaches Umschreiben des Prädiktionsfehlers als eine Summe von Prädiktionsdifferenzen,

$$z - P_t = \sum_{k=t}^{T}(P_{k+1} - P_k) \quad \text{mit} \quad P_{T+1} := z, \tag{6.18}$$

Gleichung (6.17) umschreiben läßt als

$$\begin{aligned} w_{n+1} = w_n + \alpha \sum_{t=1}^{T}(z - P_t)\nabla_w P_t &= w_n + \alpha \sum_{t=1}^{T}\sum_{k=1}^{T}(P_{k+1} - P_k)\nabla_w P_t \\ &= w_n + \alpha \sum_{k=1}^{T}\sum_{t=1} k(P_{k+1} - P_k)\nabla_w P_t \\ &= w_n + \sum_{t=1}^{T}\alpha(P_{t+1} - P_t)\sum_{k=1}^{t}\nabla_w P_k. \end{aligned}$$

Dies heißt aber, daß die Lerngleichung

$$w_{n+1} = w_n + \sum_{t=1}^{T}\Delta w_t \quad \text{mit} \quad \Delta w_t = \alpha(P_{t+1} - P_t)\sum_{k=1}^{t}\nabla_w P_k \tag{6.19}$$

zum selben Ergebnis führt wie Gleichung (6.17) - mit dem großen Vorteil, daß nur noch zwei Vektoren zu speichern sind (die Summe der Parameteränderungen Δw_t und die Summe der Gradienten $\nabla_w P_k$), und der Berechnungsaufwand sich verteilt. Die Lerngleichung (6.19) erklärt auch den Namen des Lernverfahrens: Temporal Difference Learning (zeitliches Differenzenlernen) arbeitet mit den Prädiktionsdifferenzen $P_{t+1} - P_t$ anstelle der tatsächlichen Fehler $z - P_t$.

Die TD(λ)-Familie von Lernprozeduren ergibt sich aus (6.19) durch Einführung eines Faktors $0 \leq \lambda \leq 1$, der den Einfluß des momentanen Prädiktionsfehlers auf die Prädiktionen der Vergangenheit exponentiell abschwächt und sich im Falle von $\lambda = 1$ zu (6.19) reduziert

$$w_{n+1} = w_n + \sum_{t=1}^{T} \Delta w_t \quad \text{mit} \quad \Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k. \tag{6.20}$$

Im Rahmen der Lösung des MDP-Problems wird TD(λ)-Lernen interessant, weil es zur Schätzung des diskontierten kumulativen Reinforcements $V^\pi(s)$ nach (6.1) bei konstanter Politik π eingesetzt werden kann. Mit den Prädiktionen $P_t = V(s_t)$ und $P_{t+1} = r(u_t|s_t) + \gamma V(s_{t+1})$ ergibt sich der aus (6.15) bekannte Prädiktionsfehler

$$err_{TD}(s_t) = P_{t+1} - P_t = r(u_t|s_t) + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{6.21}$$

und man erhält

$$\Delta w_t = \alpha \, err_{TD}(s_t) \sum_{k=1}^{t} (\lambda\gamma)^{t-k} \nabla_w V_k. \tag{6.22}$$

Im Falle der im folgenden zu betrachtenden tabularen Repräsentation der V-Funktion wird (6.22) zu

$$\Delta V_t(s) = \alpha \, err_{TD}(s_t) \sum_{k=1}^{t} (\lambda\gamma)^{t-k} \chi_s(k), \tag{6.23}$$

wobei $\chi_s(k)$ die Indikatorfunktion für Zustand s ist. Der Beitrag des Prädiktionsfehlers $err_{TD}(s_t)$ zur Schätzung des kumulativen Reinforcements bezüglich Zustand s_{t-m} ist nach (6.23)

$$err_{TD}^m(s_{t-m}) = (\lambda\gamma)^m err_{TD}(s_t) = (\lambda\gamma)^m (r(u_t|s_t) + \gamma V(s_{t+1}) - V(s_t)),$$

woraus ersichtlich wird, daß beim TD(λ)-Lernen für $\lambda > 0$ der momentane Prädiktionsfehler einen Beitrag nicht nur zur Schätzung der V-Funktion in Zustand s_t sondern in allen besuchten Zustände liefert. Im Gegensatz zu TD(0)- oder Q-Lernen wirkt sich der momentane Prädiktionsfehler unmittelbar auf alle vorhergehenden Zustände aus und nicht erst bei wiederholtem Durchlauf der Zustandskette, was für die erhöhte Konvergenzgeschwindigkeit der TD($\lambda > 0$)-Verfahren verantwortlich ist. Nach m Schritten ergibt sich durch (6.23) als *m-Schritt TD(λ)-Schätzer*

$$\hat{V}_m^\lambda(s_t) = \sum_{k=0}^{m-1} (\gamma\lambda)^k err_{TD}(s_{t+k}) + V_t(s_t) \tag{6.24}$$

$$= r_t + \gamma V_t(s_{t+1}) + \sum_{k=1}^{m-1} (\gamma\lambda)^k [r_{t+k} + \gamma V_t(s_{t+k+1} - V_t(s_{t+k})]. \tag{6.25}$$

Dieser ist, wie bereits von Watkins [133] nachgewiesen, ein im Sinne des Erwartungswertes besserer Schätzer als der beim TD(0)- oder Q-Lernen verwendete 1-Schritt TD(0)-Schätzer

$$\hat{V}_m^0(s_t) = r_t + \gamma V_t(s_{t+1}). \tag{6.26}$$

Dabei spielt λ die Rolle eines Parameters, der die Balance zwischen Bias und Varianz der Schätzung regelt [133]. Ein Erhöhung von λ erhöht die Qualität des Schätzers im Sinne des Erwartungswertes, vergrößert aber gleichzeitig dessen Varianz.

Es werden nun zwei Versionen des TD(λ)-Lernens unterschieden, die *Online*- und die *Batch*-Version. Beim Online-TD(λ)-Lernen wird der sich zum Zeitpunkt t ergebende Beitrag zur Änderung der Schätzer direkt für einen Update-Schritt benutzt, und man erhält

$$\begin{aligned} V_{t+1}(s) &= V_t(s) + \Delta V_t(s) && (6.27)\\ &= V_t(s) + \alpha_t(s)[r_{s_t} + \gamma V_t(s_{t+1}) - V_t(s_t)] \sum_{k=1}^{t} (\gamma\lambda)^{t-k} \chi_s(k). && (6.28)\end{aligned}$$

In der Batch-Version werden die Änderungen $\Delta V_t(s)$ erst über m Schritte akkumuliert (wobei im Falle absorbierender Markov-Ketten m im allgemeinen der Länge der Kette entspricht). Man erhält in Analogie zu (6.20)

$$\begin{aligned} V_{n+1}(s) &= V_n(s) + \alpha_n(s) \sum_{t=1}^{m} \Delta V_t(s) && (6.29)\\ &= V_n(s) + \alpha_n(s) \sum_{t=1}^{m} [r_{s_t} + \gamma V_n(s_{t+1}) - V_n(s_t)] \sum_{k=1}^{t} (\gamma\lambda)^{t-k} \chi_s(k). && (6.30)\end{aligned}$$

Die von Sutton teils nur vermuteten Konvergenzeigenschaften von $TD(\lambda)$-Lernen wurden erst von Jaakkola et al. in vollem Umfang nachgewiesen. Sie sind in folgendem Satz zusammengefaßt [62].

Satz 6.4 (Jaakkola et al. 94) *Für eine beliebige endliche Markov-Kette und eine beliebige Verteilung der Startzustände sowie eine beliebige Verteilung der Kosten mit beschränkter Varianz konvergieren unter der Voraussetzung $\gamma, \lambda \in [0,1]$ mit $\gamma\lambda < 1$ die folgenden zwei $TD(\lambda)$-Algorithmen gegen die optimalen Vorhersagen:*

1. $V_{n+1}(i) = V_n(i) + \alpha_n(i) \sum_{t=1}^{m} [r_{i_t} + \gamma V_n(i_{i_{t+1}}) - V_n(i_t)] \sum_{k=1}^{t} (\gamma\lambda)^{t-k} \chi_i(k)$ *gleichmäßig über i mit Wahrscheinlichkeit 1, falls* $\sum_n \alpha_n(i) = \infty$ *und* $\sum_n \alpha_n^2(i) \leq \infty$.
2. $V_{t+1}(i) = V_t(i) + \alpha_t(i)[r_{i_t} + \gamma V_t(i_{i_{t+1}}) - V_t(i_t)] \sum_{k=1}^{t} (\gamma\lambda)^{t-k} \chi_i(k)$ *gleichmäßig über i mit Wahrscheinlichkeit 1, falls* $\sum_t \alpha_t(i) = \infty$ *und* $\sum_t \alpha_t^2(i) \leq \infty$.

Dabei steht der erste Fall für die Batch-Version, der zweite Fall für die Online-Version.

Der Beweis erfolgt wie im Falle des Q-Lernens durch Anwendung von Satz (6.3), wobei zuerst Konvergenz für die Batch-Version bewiesen wird und dann gezeigt wird, daß der Effekt der Online-Adaption im Grenzfalle $t \to \infty$ verschwindet, d.h. Batch- und Online-Version asymptotisch identisch werden [62].

AHC-Lernen mit TD(λ)-Prädiktoren

AHC-Lernen mit TD(λ)-Prädiktoren unterscheidet sich vom AHC-Lernen dadurch, daß anstelle der TD(0)-Prädiktoren TD(λ)-Prädiktoren mit $0 \leq \lambda \leq 1$ verwendet werden. Dies bietet den Vorteil, daß durch das TD(λ)-Lernen in einem Zeitschritt nicht nur die V- und q-Schätzer des letzten Zustandes, sondern aller bisher besuchten Zustände und Zustand/Aktion-Paare auf effiziente Weise adaptiert werden und die momentane Information zeitlich zurück zu allen Zuständen propagiert wird. Für die V- und q-Schätzer sind dabei im allgemeinen zwei getrennte Aktivitätsspuren zu verwalten.

Der wesentliche Unterschied zwischen AHC(λ)-Lernen und TD(λ)-Lernen besteht darin, daß die Konvergenzaussagen für TD(λ)-Lernen nur für eine konstante Politik π gelten, während sich diese beim AHC(λ)-Lernen ständig ändert (Exploration). Damit lassen sich die Konvergenzaussagen für TD(λ)- nicht auf das AHC(λ)-Lernen übertragen, die Konvergenz von AHC(λ)-Lernen hängt vielmehr von der Explorationsstrategie ab.

6.2.4 Modifiziertes Q-Lernen zum Stabbalancieren

Aus den Bemerkungen zum Q- und zum AHC(λ)-Lernen folgt, daß diese beiden Lernverfahren zur Lösung des MDP weitgehend komplementäre Eigenschaften aufweisen, die in Tabelle 6.1 zusammengefaßt sind. Diese Beobachtung gibt Anlaß, über eine Kombination

Q-Lernen	AHC(λ)-Lernen
(+) kompakt (eine Funktion)	(-) nicht kompakt (zwei Funktionen)
(+) garantierte Konvergenz	(-) Konvergenz nicht garantiert
(+) experimentationsunabhängig	(-) experimentationsabhängig
(-) inhärent sequentiell, langsam	(+) inhärent parallel, schnell

Tabelle 6.1: Die komplementären Eigenschaften von Q- und AHC(λ)-Lernen

von Q- und AHC-Lernen nachzudenken, welche die Vorteile beider Verfahren vereint. Bei dem in diesem Abschnitt vorzustellenden *Modifizierten Q-Lernen* (MQ-Lernen) handelt es sich um eine heuristische Kombination, die die Kompaktheit von Q-Lernen mit der inhärenten Schnelligkeit und Parallelität von AHC(λ)-Lernen kombiniert. Es wurde gezielt zur Lösung von Problemen ähnlich dem Stabbalancieren (siehe Abschnitt 6.2.4) entwickelt und zeigt dort auch sehr zufriedenstellende Resultate, weist jedoch wie das AHC(λ)-Lernen keine allgemeinen Konvergenzeigenschaften auf [23]. Die grundlegenden Ideen lassen sich aber zu einer konvergenten Synthese aus AHC- und Q-Lernen nutzen (siehe Abschnitt 6.2.6).

Das Stabbalance-Problem

Beim Stabbalance-Problem (siehe Abbildung 6.5) gilt es, einen Kraft-Regler zur Beschleunigung eines Wagens zu finden, so daß weder der auf diesen Wagen montierte Stab einen bestimmten Winkel θ relativ zur Vertikalen überschreitet, noch der Wagen ein bestimmtes x-Intervall verläßt. Von diesem Problem existieren verschiedene Varianten, deren wesentlicher Unterschied darin liegt, ob die Kraft beliebige reelle oder aber nur bestimmte diskrete Werte annehmen darf. Im Rahmen des Lernens aus verzögertem Reinforcement hat sich das u.a. in [89] beschriebene Szenario als Quasi-Benchmark etabliert. Hierbei erhält der Regler als Eingangsgrößen die momentane x-Koordinate x und die Geschwindigkeit $\dot{x}$ sowie den Winkel θ und die Winkelgeschwindigkeit $\dot{\theta}$. Mit diesen Eingangsgrößen muß er eine Kraft assoziieren, die nur die beiden diskreten Werte +10N oder -10N annehmen darf (Bang-Bang-Regelung). Die Dynamik des Wagens (vgl. [9]) ist durch

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta[\frac{-F - ml\dot{\theta}^2 \sin\theta + \mu_C \mathrm{sgn}(\dot{x})}{m_C + m}] - \frac{\mu_P \dot{\theta}}{ml}}{l[\frac{4}{3} - \frac{m \cos^2\theta}{m_C + m}]} \tag{6.31}$$

$$\ddot{x} = \frac{F + ml[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta] - \mu_C \mathrm{sgn}(\dot{x})}{m_C + m} \tag{6.32}$$

beschrieben und wird durch Eulerintegration mit einer Zeitdiskretisierung von $\Delta t = 20\ ms$ gelöst. Dabei ist $g = 9.8\ ms^{-2}$ die Erdbeschleunigung, $m_C = 1.0\ kg$ die Masse des Wagens, m die Masse des Stabes, $l = 0.5\ m$ die halbe Stablänge, $\mu_c = 0.0005$ der Reibungskoeffizient des Wagens auf seiner Unterlage, $\mu_p = 0.000002$ der Reibungskoeffizient des Stabes auf dem Wagen und $F = \pm 10\ N$ die jeweilige Kraft, die auf den Wagen wirkt.

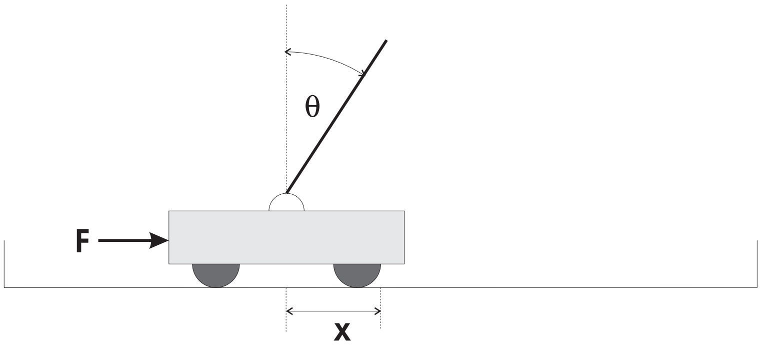


Abbildung 6.5: Versuchsaufbau Stabbalance

Zusätzlich ist eine bestimmte Quantisierung des beobachtbaren Zustandsraumes gefordert, und zwar in 3 Intervalle bzgl. der Position, 6 bzgl. des Winkels, 3 bzgl. der Geschwindigkeit und 3 bzgl. der Winkelgeschwindigkeit:

$$\begin{aligned} x &: \pm 0.8, \pm 2.4\, m, \\ \theta &: 0, \pm 1, \pm 6, \pm 12°, \\ \dot{x} &: \pm 0.5, \pm\infty\, ms^{-1}, \\ \dot{\theta} &: \pm 50, \pm\infty\, ms^{-1}. \end{aligned}$$

Dies ergibt $3 \times 3 \times 6 \times 3 = 162$ disjunkte Quader im Zustandsraum, also 162 für den Regler unterscheidbare Zustände.

Der Wagen startet stets im Zustand ($x = 0$, $\dot{x} = 0$, $\theta = 0$ und $\dot{\theta} = 0$). Wenn der Wagen das x-Intervall $[-2m, \ldots, 2m]$ oder der Stab den Winkelbereich $[-12°, \ldots, 12°]$ verläßt, ist der Versuch durch einen Fehler beendet. In diesem Fall erhält der zu testende Lernalgorithmus ein Reinforcement-Signal von $r_{final} = -1$, auf dessen Grundlage er den Regler adaptieren muß. Danach wird das System in den Startzustand zurückversetzt und ein neuer Versuch beginnt. Über die (erfolgreiche) Dauer des Versuches ist das Reinforcement-Signal $r_t = 0$.

Im Rahmen dieser Arbeit wird das Experiment dadurch erschwert, daß zusätzlich die Eingabegrößen mit einem 5%igen gleichförmigen Rauschen gestört werden. Hierdurch kann die Rauschempfindlichkeit der Lernalgorithmen getestet und über die ansonsten deterministisch arbeitenden Verfahren können statistische Aussagen getroffen werden.

Ähnlich wie in [9] dauert eine Versuchsreihe maximal 500000 Zeitschritte. Protokolliert wird dabei jeweils die Anzahl der erfolgreichen Ballanceschritte über der Versuchsnummer. Wird innerhalb einer Versuchsreihe ein Versuch abgebrochen, weil die Zeit abgelaufen ist, so wird allen folgenden (nicht tatsächlich stattfindenden) Versuchen dieser Versuchsreihe die Anzahl der Ballanceschritte des letzten Versuches zugeschrieben. Die in den beiden folgenden Abschnitten gemittelten Werte zum Q-Lernen entstanden durch Mittelung über jeweils 20 Versuchsreihen. Dabei zählt ein Versuch mit *mehr* als 10000 Balanceschritten mit *nur* 10000 Schritten, um eine positive Verzerrung des Ergebnisses zu vermeiden.

Das Stabbalance-Experiment testet die Fähigkeit des Lernalgorithmus, aus sehr spärlich (nur beim Umfallen) anfallenden und unter Umständen sehr lang (oft über 1000 Schritte) verzögerten Reinforcementsignalen zu lernen. Man beachte, daß das Stabbalance-Problem im Grunde kein MDP darstellt: Die Übergangswahrscheinlichkeiten erscheinen nur deshalb als zufällig, weil aufgrund der Zustandsdiskretisierung unterschiedliche Zustände innerhalb einer Gitterzelle als gleich behandelt werden. Tatsächlich arbeitet das System deterministisch (siehe Systemdynamik) und die Übergangswahrscheinlichkeiten hängen von der Geschichte ab (Verletzung der Markoveigenschaft). Des weiteren ist die Anzahl der verfügbaren Aktionen minimal (2, Bang-Bang-Regelung), und die Regelbarkeit des Systems hängt in kritischer Weise von der Quantisierung des Zustandsraumes ab. So führen schon kleine Änderungen der Quantisierung dazu, daß das System nicht mehr regelbar ist, d.h. keine stationäre Politik existiert, die den Stab beliebig lange vor dem Umfallen bewahrt. Schließlich ist die Annahme eines Reset-Operators, der das System jederzeit in den Startzustand zurückversetzen kann, unbefriedigend. Alle diese Eigenschaften stellen in Retrospektive die Geeignetheit des Stabbalancierens als Benchmark in Frage.

MQ-Lernen

Das vom Autor in [23] vorgestellte modifizierte Q-Lernen beruht auf einer Reihe von heuristischen Modifikationen des ursprünglichen Q-Lernens und wurde gezielt zur Lösung des Stabbalance-Problems entwickelt.

Die erste Modifikation des Q-Lernens betrifft das Stabilität/Plastizität-Dilemma. Im Gegensatz zu dem geforderten "Einfrieren" der Lernrate wird die Priorität von der Stabilität auf die Plastizität verlagert und eine *konstante* Lernrate für alle folgenden Stabbalance-Experimente genutzt. Diese Modifikation findet sich u.a. auch in [6, 76, 33].

Während das Erkundung/Nutzung-Dilemma für gewöhnlich durch eine Boltzmannverteilung gelöst wird, deren Temperaturparameter ständig verkleinert wird, zeigt sich im Stabbalance-Experiment, daß eine *optimistische Initialisierung* in Zusammenhang mit einer *gierigen Politik* (engl. greedy policy) in dieser Anwendung bessere Ergebnisse erzielt. Dabei bedeutet eine optimistische Initialisierung, daß die initialen Q-Werte allesamt größer als die optimalen Q-Werte sind, d.h.

$$Q_0(s,u) \geq Q^{opt}(s,u) \quad \forall s \in S, u \in U. \tag{6.33}$$

Die gierige Politik besagt, daß in jedem Zustand stets die Aktion mit dem größten Q-Schätzer gewählt wird. Mit dieser zweiten Modifikation wird MQ-Lernen deterministisch und weist Ähnlichkeiten zum Q-Lernansatz von Heger [53] auf. Im Rahmen des Stabbalanceexperiments[2] wird auf diese Art in 60% der Fälle ein erfolgreiches Ballanceverhalten erlernt (TD(0)-Kurve in Abbildung 6.7).

Erst durch zwei weitere heuristische Modifikationen des ursprünglichen Q-Lernens wird eine nahezu 100%ige Balancewahrscheinlichkeit mit gleichzeitig hoher Konvergenzgeschwindigkeit erlernt. Beide stellen eine Anleihe an das TD-Lernen dar und schlagen damit die angekündigte Brücke zwischen Q- und AHC(λ)-Lernen. Die erste beruht auf der Beobachtung, daß zu häufiger Wechsel der Politik sich negativ auf das Lernverhalten auswirkt. Daher wird die Politik im folgenden stets über längere Zeitperioden konstant gehalten, Fehler werden akkumuliert und wie bei der batch-Version vom TD-Lernen werden erst nach diesen Zeitperioden die Q-Schätzer aktualisiert. Die Motivation hinter diesem *Prinzip der stückweise konstanten Politik* ist einerseits, daß TD-Lernen nur für eine konstante Politik konvergent ist, andererseits, daß die akkumulierten Änderungen im Erwartungswert bessere Schätzer als die Einschritt-Schätzer darstellen [62]. Tatsächlich findet sich dieses Prinzip der stückweise konstanten Politik in vielen erfolgreichen Anwendungen im Zusammenhang mit verzögertem Reinforcement-Lernen wieder, oft nur "zwischen den Zeilen". Beispiele sind [9, 86, 72] im Zusammenhang mit Stabbalancieren und [88] zur Roboternavigation. Die konkrete Implementierung dieses Prinzips erfolgt derart, daß Prädiktionsdifferenzen stets solange akkumuliert werden, bis der Stab umfällt. Erst dann erfolgt der Update der Q-Schätzer und damit implizit die Änderung der Politik. Die Lerngleichung ist

$$Q_{n+1}(s,u) = Q_n(s,u) + \frac{\alpha}{T}\sum_{t=1}^{T} \Delta Q_t^0(s,u) \tag{6.34}$$

mit T der Anzahl der Schritte bis zum Umfallen des Stabes und

$$\Delta Q_t^0(s,u) = [r_t + \gamma \max_{u \in U} Q(s_{t+1},u) - Q(s_t,u_t)]\chi_{(s,u)}(s_k,u_k). \tag{6.35}$$

Aus der Tatsache, daß eine Änderung der Politik erst mit einem fehlgeschlagenen Balanciervers uch stattfindet, folgt, daß der Lerner eine erfolgreiche Politik nie ändern wird. Diese Eigenschaft fehlt vielen Lernschemata zum Stabbalancieren. Sie können üblicherweise nicht garantieren, daß eine erfolgreiche Balancierperiode auch reproduzierbar ist. Die effektive Lernrate in (6.34) wird mit zunehmendem Erfolg kleiner (wachsendes T im Nenner) und nicht automatisch mit der Zeit eingefroren.

[2]Die Parameter zu diesem Experiment finden sich in Tabelle 8.11.

Gleichungen (6.34) und (6.35) implementieren auf den Zustand/Aktion-Paaren ein Batch-TD(0)-Lernverfahren. Die zweite wichtige Anlehnung an das TD-Lernen erfolgt durch Erweiterung auf Batch-TD(1)-Lernen. Die wesentliche Idee hierbei ist, den momentanen Prädiktionsfehler nicht nur wie in (6.35) zum Update des letzten Zustand/Aktion-Paares beitragen zu lassen, sondern wie beim TD(λ)-Lernen zum Update aller besuchter Zustand/Aktion-Paare zu nutzen, um so den Lernprozeß zu beschleunigen. Hierzu wird Gleichung (6.35) durch

$$\Delta Q_t^1(s,u) = (r_t + \gamma \max_{u \in U} Q(s_{t+1},u) - Q(s_t,u_t)) \sum_{k=1}^{t} \gamma^{t-k} \chi_{(s,u)}(s_k,u_k) \tag{6.36}$$

ersetzt und man erhält als Update beim Umfallen des Stabes

$$Q_{n+1}(s,u) = Q_n(s,u) + \frac{\alpha}{T} \sum_{t=1}^{T} \Delta Q_t^1(s,u). \tag{6.37}$$

Die Berechnung der $\Delta Q_t^1(s,u)$ erfolgt rekursiv durch

$$\Delta Q_t^1(s,u) = (r_t + \gamma \max_{u \in U} Q(s_{t+1},u) - Q(s_t,u_t))\, a_t(s,u) \tag{6.38}$$

mit

$$a_t(s,u) = \gamma a_t(s,u) + \chi_{(s,u)}(s_t,u_t) \tag{6.39}$$

und $a_t(s,u) = 0 \quad \forall (s,u)$ bei einem Reset (Neustart des Versuches). Die Folge der $a_t(s,u)$ wird als *Aktivitätsspur* (engl. activity trace, eligibility trace) bezeichnet. Der modifizierte Q-Lernalgorithmus ist in Abbildung 6.6 zusammengefaßt.

$Q(s,u) = 0 \quad \forall s \in S, u \in U$;
do {
  $\Delta Q(s,u) = 0$ und $a(s,u) = 0 \quad \forall s \in S, u \in U$;
  s_t = Startzustand; $T = 0$;
  repeat {
    $u_t = \arg\max_u Q(s_t,u)$;
    Führe Aktion u_t aus; registriere r_t und s_{t+1};
    $a(s,u) := \gamma a(s,u) + \chi_{(s,u)}(s_t,u_t) \quad \forall s \in S, u \in U$;
    $\Delta Q(s,u) := \Delta Q(s,u) + (r_t + \gamma \max_{u\in U} Q(s_{t+1},u) - Q(s_t,u_t))\, a_t(s,u) \quad \forall s \in S, u \in U$;
    $T := T + 1$;
  } **until** (Fehler) ;
  $Q(s,u) += \frac{\alpha}{T} \Delta Q(s,u)$;
} **loop**;

Abbildung 6.6: Modifiziertes Q-Lernen

Modifiziertes Q-Lernen erweist sich im Stabbalance-Test[3] als sehr zuverlässig und schnell, wie die TD(1)-Kurve in Abbildung 6.7 zeigt. Nahezu alle Balancierversuche sind in der

[3] Die Parameter für diesen Test finden sich in Tabelle 8.12.

vorgegebenen Zeit erfolgreich. Auch hinsichtlich der Schnelligkeit, in der der Lerner eine Lösung findet, kann sich modifizierte Q-Lernen mit seinem klassischen Vorbild, dem AHC-basierten Stabbalancierer von Barto et al. [9], messen. Mit nur einer Q-Funktion, einer Aktivitätsspur und einer zweizeiligen Lerngleichung ist es jedoch wesentlich kompakter und transparenter als sein AHC-basierter Vorgänger.

Der Vergleich mit modifiziertem Q-Lernen ohne TD(1)-Lernen (siehe Abbildung 6.7) unterstreicht die Wichtigkeit der Aktivitätsspuren. Sie erlauben es dem Lerner, sehr viel schneller Informationen über den Weg im Zustandsraum zurück zu propagieren und bei derselben Anzahl von Schritten weitreichendere Vorhersagen zu treffen als beim Q-Lernen. Dieses Ergebnis ist in Einklang mit ähnlichen Beobachtungen in [97, 75, 72, 115, 43].

Obwohl sich für das modifizierte Q-Lernen keine allgemeinen Konvergenzaussagen treffen lassen, bleibt anzumerken, daß das Verfahren "abschnittsweise" konvergieren würde, falls die Lernkonstante wie beim TD(λ)-Lernen gefordert eingefroren wird: Über die Perioden konstanter Politik erfolgt ein reines TD(1)-Lernen, das nach Satz 6.4 gegen den Wert des kumulativen diskontierten Reinforcements für diese Politik konvergiert.

Experimentelle Resultate

Abbildungen 6.7 und 6.8 zeigen die experimentellen Resultate[4], die mit modifiziertem Q-Lernen beim Stabbalancieren erzielt wurden. In Abbildung 6.7 sind die über 20 Versuchsreihen gemittelten Ergebnisse für MQ-Lernen in Kombination mit und ohne TD(1)-Lernen (d.h. TD(0)-Lernen) dargestellt. Man beachte, daß durch das "Abschneiden" der erfolgreichen Balancierschritte bei 10000 (vgl. Abschnitt 6.2.4) die Kurven Aufschluß über den Prozentsatz gelungener bzw. gescheiterter Versuche geben. So ist MQ-Lernen ohne TD(1)-Aufsatz in 60% der Versuchsreihen erfolgreich, während die Kombination mit TD(1)-Lernen in nur zwei von 20 Versuchsreihen fehlschlägt. Ein genauere Untersuchung der fehlgeschlagenen Versuchsreihen ergab, daß auch in diesen Fällen eine Balancierpolitik erlernt worden wäre, allerdings erst nach mehr als 500000 Schritten. Zum Vergleich mit einer ähnlichen Grafik für Bartos AHC(λ)-Klassiker in [9] zeigt Abbildung 6.8 den Verlauf von 10 einzelnen Versuchsreihen mit der TD(1)-Modifikation.

6.2.5 Modifiziertes Q-Lernen mit DCS

Q-Lernen mit tabularer Funktionsrepräsentation leidet unter dem "Curse of Dimensionality", d.h. daß der Speicherplatz zur Repräsentation der Q-Funktion exponentiell mit der Dimensionalität des Zustandsraumes ansteigt, und stellt damit eine untere Schranke für die Zeitkomplexität des Verfahrens dar. Dies ist eine erste Motivation zum Einsatz von generalisierenden KNN im Rahmen von Q-Lernen. Die Voraussetzung einer a priori Quantisierung des Zustandsraumes zur tabularen Funktionsrepräsentation ist die zweite Motivation, denn oft liegt gerade im Auffinden einer geeigneten Quantisierung das wesentliche Problem. Es wird im Zusammenhang mit Reinforcement-Lernen auch als Problem der Adaptiven Perzeptualisierung (engl. adaptive perceptualization) [136] bezeichnet.

[4] Die Parameter finden sich in den Tabellen 8.11 und 8.12.

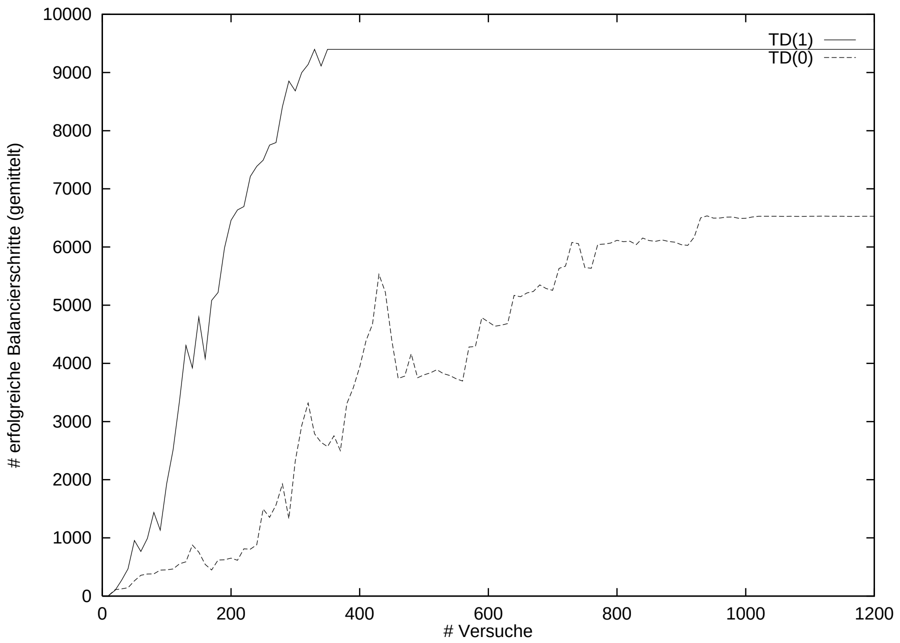


Abbildung 6.7: Stabbalance mit MQ-Lernen (gemittelt)

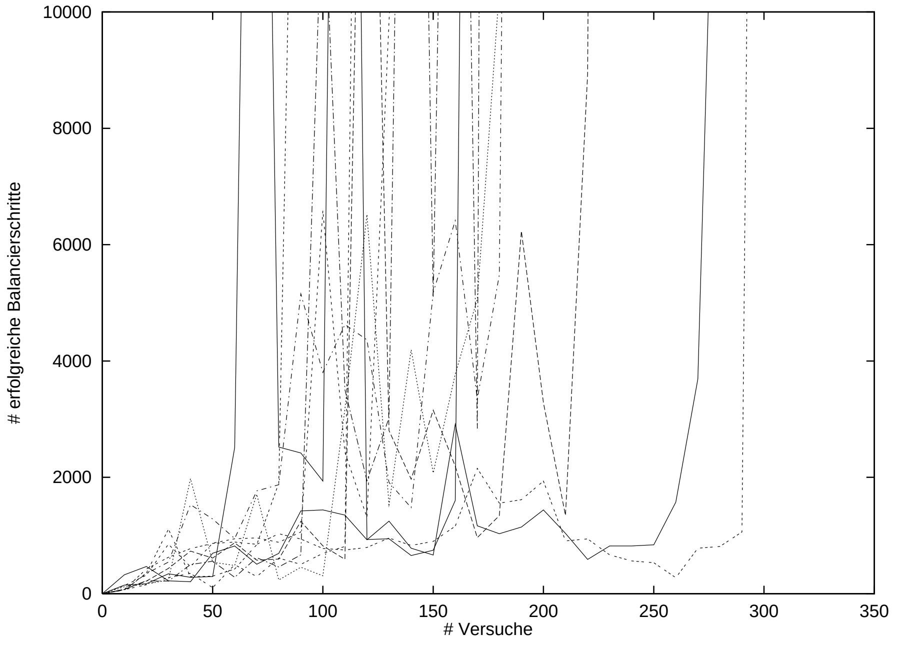


Abbildung 6.8: Stabbalance mit MQ-Lernen: 10 einzelne Versuchsreihen mit TD(1) Modifikation

Der Einsatz von RBF-Netzen ist hier besonders naheliegend, ergibt sich doch die tabulare Funktionsrepräsentation gerade als Grenzfall unendlich kleiner rezeptiver Felder. Im Rahmen des Stabbalancierens mit Q-Lernen wurden sie erstmals von Anderson [6] eingesetzt.

Als Weiterentwicklung der RBF-Netze sollen hier DCS-Netze mit Einer-Nachbarschaft zum Q-Lernen eingesetzt werden. Dabei ähnelt der Versuchsaufbau dem Andersons, indem RBF-Knoten mit konstanten Radien in Zusammenhang mit stochastischem Gradientenabstieg zur Adaption der Ausgabeschicht und der Zentren verwendet werden. Zusätzlich aber wird das Prinzip der stückweise konstanten Politik sowie Aktivitätsspuren genutzt. Wie im Falle des modifizierten Q-Lernens wird eine konstante Lernrate und optimistische Initialisierung eingesetzt sowie stets der gierigen Politik gefolgt. Damit ist der DCS-basierte Q-Lernalgorithmus identisch mit dem modifizierten Q-Lernen (vgl. Abbildung 6.6), mit Ausnahme der DCS-basierten Funktionsrepräsentation inklusive adaptiver Perzeptualisierung (Zentrenadaption und Einfügen neuer RBF-Knoten). Der Q-Schätzer für Zustand s und Aktion u berechnet sich unter Verwendung normalisierter DCS-Netzwerke zu

$$Q^{DCS}(s,u) = \sum_{i \in Nh^{(1)}(bmu_s)} q(i,u)\, a_i \quad \text{mit} \quad a_i = \frac{h(\|s-c_i\|/\sigma)}{\sum_{j \in Nh^{(1)}(bmu_s)} h(\|s-c_j\|/\sigma)} \,. \tag{6.40}$$

Er ist damit die mit den Aktivitäten a_i gewichtete Summe der Q-Schätzer $q(i,u)$, den Ausgabegewichten der "Zustandsneuronen".

Adaption des QDCS-Netzwerkes

Im folgenden sollen nun die Lernregeln zur Parameter- und Modelladaption des DCS-basierten Q-Lernens angegeben werden.

Adaption der Ausgabeschicht In Analogie zum modifizierten Q-Lernen (6.36 - 6.39) wird

$$\Delta q_t(i,u) = (r_t + \gamma \max_{u \in U} Q^{DCS}(s_{t+1},u) - Q^{DCS}(s_t,u_t)) \sum_{k=1}^{t} \gamma^{t-k} \nabla_{q(i,u)} Q^{DCS}(s_k,u_k) \tag{6.41}$$

gesetzt, die Gewichte $q(i,u)$ der Ausgangsschicht beim Umfallen des Stabes werden nach

$$q_{n+1}(i,u) = q_n(i,u) + \frac{\alpha}{T} \sum_{t=1}^{T} \Delta q_t(i,u). \tag{6.42}$$

adaptiert. Die Berechnung der $\Delta q_t(i,u)$ erfolgt dabei wiederum rekursiv durch

$$\Delta q_t(i,u) = (r_t + \gamma \max_{u \in U} Q^{DCS}(s_{t+1},u) - Q^{DCS}(s_t,u_t))\, e_t(i,u) \tag{6.43}$$

mit den Aktivitätsspuren

$$e_{t+1}(i,u) = \gamma\, e_t(i,u) + \chi_u(u_t)\, a_i(s_t) \tag{6.44}$$

und $e_t(s,u) = 0 \;\forall s \in S, u \in U$ bei einem Reset.
Die Adaption der Ausgabegewichte erfolgt damit analog zur Adaption der Tabelleneinträge beim modifizierten Q-Lernen durch TD(1)-Lernen.

Adaption der Zentren Die Adaption der Zentren erfolgt in Anlehnung an das Andersonsche Experiment [6] durch Gradientenabstieg auf dem TD(0)-Prädiktionsfehler. Da gerade das Stabbalancieren sehr sensibel auf Änderungen der Quantisierung reagiert, muß durch relativ kleine Lernraten ϵ eine langsame und vorsichtige Adaption der Zentren erzwungen werden. Eine Beschleunigung des Lernvorganges durch TD(λ)-Lernen mit $\lambda \neq 0$ würde hier keinen Sinn ergeben. Des weiteren erspart TD(0)-Lernen das Verwalten von Aktivitätsspuren für die Komponenten der Zentrumsvektoren.
Damit ergibt sich

$$\Delta c_i^t = (r_t + \gamma \max_{u \in U} Q^{DCS}(s_{t+1}, u) - Q^{DCS}(s_t, u_t)) \nabla_{c_i} Q^{DCS}(s_t, u_t) \tag{6.45}$$

und zur Adaption der Zentren beim Umfallen des Stabes:

$$c_i^{n+1} = c_i^n + \frac{\epsilon}{T} \sum_{t=1}^{T} \Delta c_i(t). \tag{6.46}$$

Der Einsatz alternativer Lernregeln zur Adaption der Zentren ist denkbar (siehe Abschnitt 3.4.3), wichtig aber ist der Bezug zur Optimierung des Ausgabefehlers (des Prädiktionsfehlers). So scheidet beispielswiese die Frizke-Lernregel (3.29) ohne Einfrieren der Lernparameter aus, denn angenommen, das System hätte eine stabile Trajektorie im Zustandsraum erreicht, dann würde diese allein von der Eingabedichte abhängige Regel unweigerlich zum Kollabieren des Netzwerkes auf der Trajektorie führen.

Wachstum des Netzes Neben der sehr vorsichtigen Adaption der Zentren durch Gradientenabstieg ist das Einfügen neuer RBF-Knoten hauptverantwortlich für die adaptive Perzeptualisierung. Die hier verfolgte Wachsstumsstrategie ist eine einfache distanzgetriebene Strategie (siehe Abschnitt 3.4.6), d.h. ein neuer RBF-Knoten wird immer dann eingefügt, wenn der Abstand der bmu_s zum Stimulus einen bestimmten Wert δ_{dist} überschreitet. Sein Zentrumsvektor wird auf den Wert des Stimulus gesetzt, sein Ausgabevektor $q(i_{new}, .)$ mit dem der bmu_s initialisiert. Das Wachstum des Netzes wird beim Erreichen einer Maximalzahl N_{max} von Knoten beendet, von wo ab eine Änderung der Perzeptualisierung allein durch Zentrenadaption erfolgt.

Adaption der Topologie Die Adaption der Topologie erfolgt mittels der einfachen Hebb-Regel mit Abklingkonstante (3.40).

Experimentelle Resultate

Abbildungen 6.9 und 6.10 zeigen die experimentellen Resultate, die mit QDCS-Lernen erzielt wurden[5]. Dabei wurde wie im Falle des tabularen modifizierten Q-Lernens das Lernverfahren mit und ohne Aktivitätsspuren (TD(1)- und TD(0)-Lernen) zur Adaption der Ausgabeschicht getestet. Die Lernkurven in Abbildung 6.9 wurden durch Mittelung über 20 Versuchsreihen gewonnen. Die maximale Zahl von eingefügten Neuronen beträgt $N_{max} = 25$.

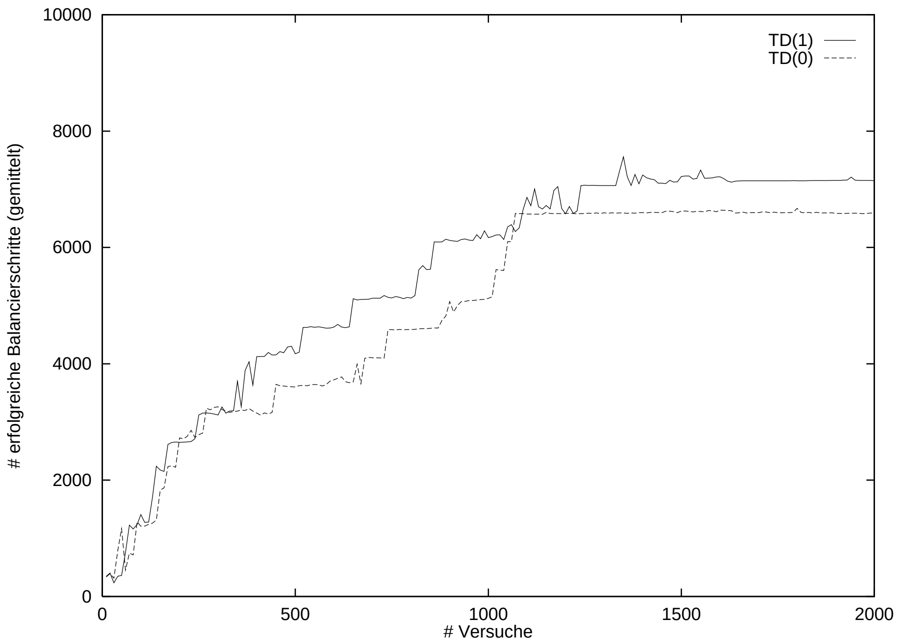


Abbildung 6.9: Stabbalance mit QDCS-Lernen (gemittelt)

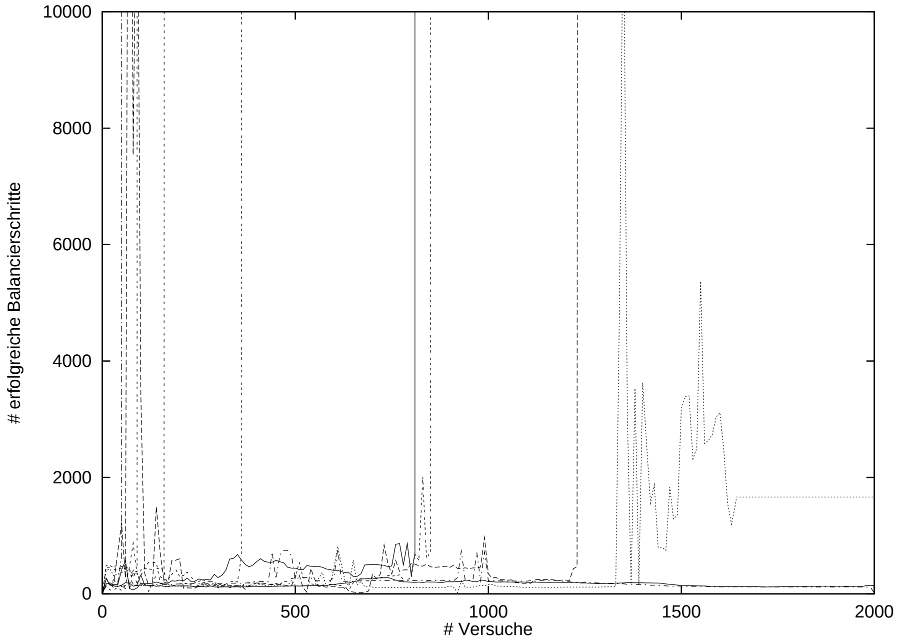


Abbildung 6.10: Stabbalance mit QDCS-Lernen: 10 einzelne Versuchsreihen mit der TD(1)-Modifikation

Der Vergleich der beiden Lernkurven mit und ohne Aktivitätsspur zeigt, daß deren Einfluß weit weniger ausgeprägt ist als beim tabularen MQ-Lernen. Die Erklärung hierfür ist, daß beim QDCS-Lernen durch die überlappenden rezeptiven Felder der RBF-Knoten nicht nur die Schätzer *zeitlich* sondern auch *räumlich* benachbarter Zustände auf Grundlage des

[5] Die Parameter zum QDCS-Lernen finden sich in Tabelle 8.13.

aktuellen Prädiktionsfehlers adaptiert werden (Gleichungen 6.40 und 6.41). Da aufgrund der Stetigkeit der Systemtrajektorie im Zustandsraum zeitlich benachbarte Zustände auch räumlich benachbart sind, erfolgt so auch ohne Aktivitätsspur eine Adaption von zeitlich aufeinanderfolgenden Zuständen. Durch die exponentielle Abnahme kann die zeitliche Wechselwirkung schon nach wenigen Schritten von der räumlichen Wechselwirkung (Nachbarschaftskooperation im DCS-Netz) dominiert werden. Diesem Effekt ließe sich nur durch eine Verringerung des Überlapps entgegenwirken. Damit aber würde das DCS-Netz an Generalisierungsfähigkeit verlieren und damit seines wesentlichen Vorteils gegenüber der tabularen Funktionsrepräsentation beraubt werden.

Des weiteren fällt im Vergleich zum MQ-Lernen auf, daß QDCS-Lernen mit und ohne Aktivitätsspur nicht die Erfolgswahrscheinlichkeit und Schnelligkeit des tabularen modifizierten Q-Lerners mit Aktivitätsspur erreicht. Berücksichtigt man jedoch die Tatsache, daß der QDCS-Lerner die zusätzliche Aufgabe der adaptiven Perzeptualisierung zu leisten hat und, wie bereits angesprochen, die Regelbarkeit des Stabbalancierers in kritischer Weise von der Quantisierung des Zustandsraumes abhängt, so ist es eher erstaunlich, daß überhaupt eine Balancewahrscheinlichkeit von 70% erreicht wird.

Anderson [6] berichtet von einer nahezu 100%igen Erfolgsquote[6] seines RBF-basierten Q-Lerners mit 30 Neuronen. Diese läßt sich dadurch erklären, daß Anderson keine Zentrenadaption sondern ein "Restarting" zur adaptiven Perzeptualisierung benutzt. Erweist sich ein RBF-Knoten an seiner momentanen Position im Zustandsraum als uneffektiv, so wird er entfernt und an einer neuen Stelle wieder eingefügt (restarted). Dieser Prozeß, der nicht gestoppt wird, tastet den Zustandsraum solange nach regelbaren Quantisierungen ab, bis er eine findet. Dabei bietet das Restarting eine Möglichkeit, lokalen Minima zu entgehen. Anderson berichtet, daß sein RBF-Q-Lerner im Mittel 3300 Versuche bis zur Balance benötigt. Im Gegensatz hierzu benötigen die erfolgreichen Versuchsreihen mit QDCS sämtlich weniger als 1500 Versuche.

Zusätzlich zu den hier geschilderten Versuchen wurden Versuchsreihen mit unterschiedlichen Netzgrößen bis zu $N_{max} = 75$ und variierendem Überlapp der rezeptiven Felder durchgeführt. Diese Versuche ließen jedoch keine systematischen Abhängigkeiten bzgl. des Überlapps und der Knotenzahl erkennen. Auch das läßt sich durch die destruktiv wirkende Abhängigkeit der Regelbarkeit von der Perzeptualisierung erklären. Lediglich für weniger als 25 Knoten war ein rapides Abnehmen der Balancierwahrscheinlichkeit zu erkennen.

Zusammenfassend läßt sich festhalten, daß QDCS-Lernen in der Lage ist, mit nur 25 Knoten, deren Position vom Lernverfahren zu ermitteln ist, eine Balancestrategie zu finden - im Gegensatz zum MQ-Lernen, das von 162 vorgegebenen ausgeht. Allerdings erweist sich QDCS-Lernen als weniger zuverlässig. Auch wurde bei den QDCS-Versuchen keine signifikante Verbesserung durch den Einsatz von Aktivitätsspuren festgestellt. Dies läßt sich dadurch erklären, daß hier das für tabulare Funktionsrepräsentationen entwickelte TD(λ)-Lernen mit den Interpolationseigenschaften von RBF-Netzwerken interferiert. Schließlich erscheint das Stabbalance-Problem mit Bang-Bang-Kraftregelung als Testfeld zur adaptiven Perzeptualisierung eher ungeeignet, denn eine Vielzahl unterschiedlicher Neuro-Fuzzy-Regler mit reellwertiger Kraftregelung zum Stabbalancieren deutet darauf hin, daß sich

[6] Diese Resultate konnten in einer Reimplementierung nicht reproduziert werden.

das Perzeptualisierungsproblem mit einer zunehmenden Anzahl von Aktionen entschärft. Dies ist auch der Grund, warum in Abschnitt 6.3.4 nicht Q-Lernen sondern reellwertiges Reinforcement-Lernen zum Erlernen einer Hindernisvermeidungsstrategie zum Einsatz kommt.

6.2.6 $Q^\lambda(m)$-Lernen

Bevor mit einem Rückblick auf Arbeiten anderer Autoren das Thema Q-Lernen abgeschlossen wird, soll an dieser Stelle noch einmal das MQ-Lernen aufgegriffen werden. Es verbindet die Kompaktheit des Q-Lernens mit dem den Lernvorgang wesentlich beschleunigenden TD(1)-Lernen, ist aber zur Lösung des allgemeinen MDPs aufgrund fehlender Konvergenzeigenschaften nicht geeignet. Durch eine kleine Änderung ergibt sich allerdings ein Lernalgorithmus, der Q- und $TD(\lambda)$-Lernen in einfacher Weise verbindet und, so die wohlbegründete Vermutung, auch konvergiert. Hierzu wird MQ-Lernen so abgeändert, daß in größeren Zeitintervallen (alle m Schritte) eine Aktion wie beim Q-Lernen beliebig (zufällig) ausgewählt wird, dann aber über das nächste Zeitintervall gierig bezüglich der momentanen Q-Werte gehandelt und danach Batch-TD(λ)-Lernen zum Aktualisieren der Q-Schätzer durchgeführt wird. Außerdem muß die Lernrate mit der Zeit gegen null gehen, um die Chance eines Konvergenzbeweises gegen die Optimalwertfunktion mit Wahrscheinlichkeit eins zu wahren. Abbildung 6.11 faßt diesen Algorithmus, der im folgenden $Q^\lambda(m)$-Lernen genannt wird, zusammen.

Zufällige Initialisierung der $Q(s,u)$-Schätzer $\forall s \in S, u \in U$;
s_t = Startzustand; $n = 0$;
do {
    $\Delta Q(s,u) = 0$ und $a(s,u) = 0 \quad \forall s \in S, u \in U$;
    for $i = 1$ **to** m **step** 1 {
        if $(i == 1)$ **then**
            Wähle u_t nach Explorationsstrategie;
        else
            $u_t = \arg\max_u Q(s_t, u)$;
        Führe Aktion u_t aus; registriere r_t und s_{t+1};
        $a(s,u) = \gamma\lambda a(s,u) + \chi_{(s,u)}(s_t, u_t) \quad \forall s \in S, u \in U$;
        $\Delta Q(s,u) := \Delta Q(s,u) + (r_t + \gamma \max_{u \in U} Q(s_{t+1}, u) - Q(s_t, u_t))\, a_t(s,u) \quad \forall s \in S, u \in U$;
        $n := n + 1$;
    }
    $Q(s,u) := Q(s,u) + \alpha_n \Delta Q(s,u)$;
} **loop**;

Abbildung 6.11: $Q^\lambda(m)$-Lernen

Vermutung 6.1 *Unter der Voraussetzung, daß alle Q-Schätzer unendlich oft aktualisiert werden (also jeder Zustand unendlich oft erreicht und in ihm jede Aktion unendlich oft* *ausgeführt wird)*[7]*, und für beschränkte Bewertungssignale* $|r_t| \leq C$ *und Lernraten* $0 \leq \alpha_n < 1$ *mit*

$$\sum_n \alpha_n(s,u) = \infty, \ \sum_n \alpha_n^2(s,u) < \infty \ \forall s \in S, u \in U \tag{6.47}$$

konvergiert $Q_n(s,u) \rightarrow Q^{opt}(s,u)$ *mit* $n \rightarrow \infty$ $\forall s \in S, u \in U$ *mit Wahrscheinlichkeit 1.*

Für diese Vermutung sprechen folgende Beobachtungen:

1. Für $m = 1$ geht $Q^\lambda(m)$-Lernen in das Q-Lernen über, für das Konvergenz mit Wahrscheinlichkeit 1 bewiesen ist.

2. Für $m > 1$ verhält sich $Q^\lambda(m)$-Lernen wie das Q-Lernen, mit der Abweichung, daß statt der 1-Schritt-TD(0)-Schätzer eine gewichtete Summe von bis zu m-Schritt-TD(λ)-Schätzern nach dem batch-TD(λ)-Verfahren zur Adaption verwendet wird.

3. Bei einer konstanten Politik π würden dabei aufgrund der Konvergenzeigenschaften von batch-TD(λ)-Lernen die Q(s,u)-Schätzer mit Wahrscheinlichkeit 1 gegen $Q^\pi(s,u)$ konvergieren. Dabei sind die m-Schritt-TD(λ)-Schätzer sogar strikt bessere Schätzer (i.S.d. Erwartungswertes) als die beim Q-Lernen verwendeten 1-Schritt-TD(λ)-Schätzer (siehe Abschnitt 6.2.3).

4. Auch bei einer alle m Schritte wechselnden Politik spricht alles dafür, daß die m-Schritt-TD(λ)-Schätzer mit Q-Lernen konsistente Schätzer sind: Für $m = 1$ entspricht der m-Schritt-Schätzer gerade dem Prädiktionsfehler für das Q-Lernen. Die Schätzer für $m > 2$ aber sind in dem Sinne konsistent, daß stets der gierigen Politik gefolgt wird und daher dieselbe Folge von Prädiktionsfehlern auch von Q-Lernen erzeugt werden kann.

Gerade die zuletzt genannte Eigenschaft der Konsistenz der verwendeten TD(λ)-Schätzer unterscheidet $Q^\lambda(m)$-Lernen von den meisten vergleichbaren Ansätzen (siehe Abschnitt 6.2.7). Die Schwierigkeit für einen Beweis der Konvergenzvermutung liegt darin, daß für einen Beweis analog zur Konvergenz des Q-Lernens in Abschnitt 6.2.2 der Erwartungswert-Operator aus Satz 6.3, dritte Bedingung, nun über eine Kette von Nachfolgezuständen operieren würde. Dies wird deutlich, wenn der potentielle Δ-Prozeß für das $Q^\lambda(m)$-Lernen zur Anwendung von Satz 6.3 formuliert wird. Mit den m-Schritt-TD(λ)-Schätzern des $Q^\lambda(m)$-Lernens ergibt sich

$$\begin{aligned}\Delta_n(s,u) &:= Q_n(s,u) - Q^{opt}(s,u)\\ F_n(s,u) &:= \hat{V}_m^\lambda(s) - Q^{opt}(s,u).\end{aligned}$$

Für den Erwartungswert von F_t erhält man

$$E[F_n(s,u)] = E[r_s + \gamma V_t(s_{+1}) - Q^{opt}(s,u)] + \sum_{k=1}^{m-1} (\gamma\lambda)^k E[(r_{s_{+k}} + \gamma V_t(s_{+k+1}) - V_t(s_{+k}))]. \tag{6.48}$$

[7] Sollte diese Voraussetzung im Widerspruch zur Forderung, die Politik über je m Schritte konstant zu halten, stehen, so genügt es beim $Q^\lambda(m)$-Lernen, die Politik “im Mittel” m Schritte konstant zu halten.

Der Beweis zur Konvergenz des Q-Lernens (Gleichungen (6.10) - (6.12)) ließe sich nun direkt übertragen, wenn der zweite Term in (6.48) wegfallen würde. Dies ist jedoch leider nicht der Fall, auch wenn die Konvergenz des TD(λ)-Lernens nahe legt, daß er beliebig klein wird.

Lernverfahren:	Q	AHC(λ)	MQ	$Q^\lambda(m)$
Kompaktheit	+	-	+	+
Konvergenz	+	-	-	(+)
Experimentationsunabhängigkeit	+	-	+	(+)
Sequentiell/ Parallel	s	p	p	p

Tabelle 6.2: Zusammenfassung: Q-, AHC(λ)-, MQ- und $Q^\lambda(m)$-Lernen.

Tabelle 6.2 faßt die bislang diskutierten direkten Methoden zur Lösung des MDPs noch einmal zusammen. Die Angaben für das $Q^\lambda(m)$-Lernen erscheinen dabei in Klammern, um die Notwendigkeit weiterer theoretischer und experimenteller Untersuchungen anzudeuten.

6.2.7 Alternative Ansätze

Die naheliegende Idee einer Kombination von Q-Lernen mit $TD(\lambda)$-Lernen fand in einer ganzen Reihe von Forschungsgruppen Anklang. Hier soll in Anlehnung an [115] ein kurzer Überblick über vergleichbare Arbeiten gegeben werden.

Schon Watkins [133] schlug vor, TD-Lernen mit Q-Lernen zu kombinieren, um das Training zu beschleunigen. Bereits er erkannte das Problem, daß die vom TD(λ)-Lernen gelieferten TD(λ)-Schätzer inkonsistent mit Q-Lernen sind, wenn bezüglich der momentanen Q-Schätzer nicht gierig gehandelt wird (also exploriert wird). Er schlug daher vor, in der an das TD(λ)-Lernen angelehnten Formel

$$\Delta Q_t(s,u) = \alpha[r_t + \gamma \max_{u \in U} Q_t(s_{t+1}, u)) - Q_t(s_t, u_t)] \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \chi_{(s,u)}(s_k, u_k) \tag{6.49}$$

den Gewichtungsfaktor λ stets dann auf null zu setzen, wenn eine nicht-gierige Aktion gewählt wird.

Diese Heuristik wurde von Lin [75], in seiner Kombination von Q-, TD(λ)-Lernen und MLP-Netzen angewendet, allerdings in Verbindung mit seinem *Experience-Replay Algorithmus*. Dieser basiert auf der Idee, in einem Kurzzeitgedächtnis jeweils die Kette der m letzten Zustände, Aktionen und empfangenen Bewertungssignale zu speichern. Diese Kette wird dann im Hintergrund rückwärts durchlaufen, wobei Adaptionen nach

$$\Delta w_t = \alpha(r_t + \gamma[(1-\lambda) \max_{u \in U} Q_t^{NN}(s_{t+1}, u)) + \lambda \hat{V}_m^\lambda(s_{t+1})] - Q_t^{NN}(s_t, u_t)) \nabla_w Q_t^{NN}(s_t, u_t) \tag{6.50}$$

stattfinden, wobei $\hat{V}_m^\lambda(s_{t+1})$ der m-Schritt-$TD(\lambda)$-Prädiktor ist (und m jeweils die Länge der Kette von Zustand s_{t+1} aus gesehen angibt). Lin berichtet von einer Beschleunigung der

Konvergenz, doch ist die Notwendigkeit des Zwischenspeicherns der Zustände, Aktionen und Bewertungssignale ein offensichtlicher Nachteil gegenüber den auf Aktivitätsspuren basierenden Verfahren. Das gleiche Verfahren benutzt auch Thrun, z.B. [123], jedoch in Kombination mit seinem *Explanation Based Neural Network Learning* (EBNN) , bei dem nicht nur der Funktionswert der zu approximierenden Funktion (hier der Q-Funktion) sondern auch deren erste Ableitung bekannt sein muß. Die Berechnung der Ableitung über eine Zustandskette kann dabei über die Kettenregel erfolgen, erfordert aber das zusätzliche Erlernen eines differenzierbaren Weltmodells (siehe [123]).

Ein weiterer Ansatz zur Kombination von Q- und TD-Lernen (ohne KNN) ist das $Q(\lambda)$-Lernen von Peng [97]. Das Problem inkonsistenter $TD(\lambda)$-Schätzer wird hier dadurch abzuschwächen versucht, daß zur Adaption der Q-Schätzer zeitlich weiter zurückliegender Zustand/Aktion-Paare ein TD(λ)-Schätzer über aufeinanderfolgenden gierigen Prädiktionen benutzt wird, unabhängig davon, ob tatsächlich der gierigen Politik gefolgt oder exploriert wird. Die Lernregel lautet

$$\begin{aligned}\Delta Q_t(s,u) &= \alpha\left([r_t + \gamma \max_{u\in U} Q_t(s_{t+1},u)) - Q_t(s_t,u_t)]\chi_{(s,u)}(s_t,u_t)\right.\\ &\quad \left. +[r_t + \gamma \max_{u\in U} Q_t(s_{t+1},u)) - \max_{u\in U} Q_t(s_t,u)]\sum_{k=0}^{t}(\gamma\lambda)^{t-k}\chi_{(s,u)}(s_k,u_k)\right).\end{aligned}$$

Neben der Tatsache, daß hier in jedem Schritt zwei Prädiktionsfehler zu berechnen sind, ist vor allem anzumerken, daß die von Peng vorgeschlagene Heuristik das Problem inkonsistenter $TD(\lambda)$-Schätzer nicht wirklich löst.

Auch Rummery [115], schlägt eine heuristische Verbindung von Q-, TD(λ)-Lernen und KNN vor, die er *Modified Connectionist Q-Learning* (MCQ-Lernen) nennt. Er argumentiert, daß das von Watkins vorgeschlagene Nullsetzen von λ bei nicht-gierigen Aktionen gerade in der Anfangsphase des Lernens dem Lernerfolg eher abträglich sei. Weiterhin führt er an, daß der Maximumsoperator entfallen kann, da bei entsprechender Explorationsstrategie in zunehmendem Maße sowieso nur noch gierige Aktionen ausgeführt würden. Seine Lernregel lautet

$$\Delta w_t = \alpha([r_t + \gamma Q_t^{NN}(s_{t+1},u_{t+1})) - Q_t^{NN}(s_t,u_t)]\sum_{k=0}^{t}(\gamma\lambda)^{t-k}\nabla_w Q_k^{NN}. \qquad (6.51)$$

Offensichtlich sind die hier auftretenden $TD(\lambda)$-Schätzer weiterhin inkonsistent mit dem Q-Lernen.

Herrmann und Der [58] benutzten das Neuronengas zur adaptiven Perzeptualisierung für einen Q-Lerner, der ebenfalls am Beispiel des Stabbalancierens getestet wurde. Es erfolgte keine Kombination mit TD(λ)-Lernen. Ihre Perzeptualisierungsstrategie besteht darin, den Wechselwirkungsparameter λ des Neuronengases (siehe 2.3) so zu modulieren, daß Knoten, deren gierige Aktion oft wechselt (die "unsicheren" Knoten), andere Knoten stärker anziehen als solche, deren gierige Aktion nicht mehr wechselt (die "sicheren" Knoten). Dadurch soll die Auflösung gezielt in unsicheren Regionen erhöht werden - eine Strategie, die stark an die fehlermodulierte Zentrenadaption (siehe 3.4.3) erinnert. Herrmann und Der benutzen dabei eine feste Zahl von 162 Knoten. Ihre Ergebnisse sind insofern mit den

QDCS-Lernresultaten vergleichbar, als daß eine 80%ige Balancierwahrscheinlichkeit bei maximal 300000 Zeitschritten nicht überschritten wird (ein um 10% schlechteres Ergebnis erreicht QDCS mit nur 25 Knoten). Gegenüber dem normalen Q-Lernen (ohne TD-Lernen) mit fester Zustandsquantisierung wird eine Erhöhung der Lerngeschwindigkeit festgestellt. Im Gegensatz zum QDCS-Lernen wird nicht interpoliert sondern stets nur der nächste Nachbar (die bmu_s) zur Ausgabeberechnung und Adaption herangezogen.

Abschließend sei noch auf eine Arbeit von Thrun [125] hingewiesen, die ein generelles Problem beim Einsatz von Funktionsapproximatoren zum Erlernen der Q-Funktion über Q-Lernen aufzeigt. Dieses ist das Problem einer systematischen Überschätzung von Q-Werten, verursacht durch den Generalisierungsfehler der verwendeten Funktionsapproximatoren in Zusammenhang mit dem Maximums-Operator der Q-Lernregel (6.7). Thrun zeigte theoretisch und praktisch, daß dieser Effekt das Erlernen einer optimalen Politik unter bestimmten Umständen unmöglich machen kann und im allgemeinen die Q-Schätzer systematisch verzerrt. Seine Ergebnisse wurden allerdings nur im Rahmen des 1-Schritt-Q-Lernens hergeleitet. Im Falle von m-Schritt-TD(λ)-Prädiktoren wie beim QDCS-Lernen tritt dieser Effekt nicht oder nur in sehr abgeschwächter Form auf, da sich nach der Differenzengleichung (6.18) Prädiktionsfehler nicht akkumulieren. Dies ist ein weiteres Argument zur Kombination von Q- mit TD(λ)-Lernen.

6.3 Reellwertiges Reinforcement-Lernen

Nachdem im letzten Abschnitt Reinforcement-Lernen in Markovschen Umgebungen mit diskreten Zustands- und Aktionsräumen behandelt wurde, soll in diesem Abschnitt Reinforcement-Lernen in reellwertigen Zustands- und Aktionsräumen betrachtet werden. Hierbei wird anstelle der Markoveigenschaft die Forderung nach stationären Zufallsprozessen treten. Während auf den ersten Blick die reellwertige Lernaufgabe schwieriger erscheint als die diskrete - schließlich stehen überabzählbar viele Aktionen für jeden der überabzählbar vielen Zustände zur Auswahl -, zeigt das Beispiel des Stabbalancierens (siehe 6.2.4), daß dies ein Trugschluß ist. Dort entsteht das eigentliche Problem erst dadurch, daß der Zustandsraum durch eine grob quantisierte "Brille" betrachtet wird (und so der tatsächliche Zustand des Systems nur grob bekannt ist) und nur zwei Aktionen als Stellgröße zur Verfügung stehen. Geringe Änderungen der Quantisierung können bewirken, daß das System nicht mehr regelbar ist. Die Unstetigkeit der Optimalwertfunktion und der Politik erschweren den Einsatz von KNN, die zur Approximation stetiger Funktionen entwickelt wurden. Hingegen ermöglichen reellwertige Zustands- und Aktionsräume die Definition stetiger und differenzierbarer Ertragsfunktionen und Politiken, die für den Einsatz von KNN in Kombination mit gradientenbasierten Lernverfahren wesentlich geeigneter sind. Die Familie der von Williams [137] eingeführten REINFORCE-Lerner, die im folgenden vorgestellt wird, beruht auf diesem Prinzip. Auf Grundlage des REINFORCE-Lernens wird dann eine DCS-basierte Reinforcement-Lernarchitektur entwickelt, die schließlich zur adaptiven Hindernisvermeidung eines simulierten TRC Labmates eingesetzt wird. Eine Abgrenzung zu ähnlichen Lernarchitekturen, auch aus dem Bereich der adaptiven Fuzzy-Regler, bildet den Abschluß dieses Kapitels.

6.3.1 Die Familie der REINFORCE-Lerner

Im Gegensatz zum Reinforcement-Lernen mit zeitlich verzögerten Bewertungssignalen gehen die von Williams [137] eingeführten REINFORCE-Lerner von einem unmittelbaren, d.h. unverzögerten Bewertungssignal aus. Das Explorationsverhalten der REINFORCE-Lerner basiert auf Zufälligkeit in der Aktionsgenerierung des Lerners, es erfolgt keine systematischen Suche im Zustandsraum. REINFORCE-Lernen wurde speziell für KNN entwickelt und basiert auf stochastischem Gradientenabstieg. Die Gradienten werden jedoch nicht explizit berechnet, und es wird auch keine Information gespeichert, die ihre direkte Berechnung erlaubt. REINFORCE-Lerner arbeiten daher nicht modellbasiert.

Beim REINFORCE-Lernen generiert eine Schicht spezieller stochastischer Ausgabeneuronen einen Ausgabevektor y (Stellgrößen), wobei die i-te Komponente des Vektors nach der Dichtefunktion $p_i(y_i|\theta_i)$ des i-ten Ausgabeneurons verteilt ist. Die Parameter θ_i der Dichtefunktion werden von einem KNN als Funktion der Eingabe x berechnet, so daß die Dichtefunktion allein von den Parametern W des KNN und der Eingabe x abhängt. Es gilt daher $p_i(y_i|\theta_i) = p_i(y_i|W, x)$.

Nach Anlegen der Stellgrößen registriert der Lerner das unmittelbare Bewertungssignal r. Ziel des REINFORCE-Lernens ist die Optimierung des Erwartungswertes $E[r|W]$ nach den Parametern des KNNs. Dabei ist der Erwartungswert nur definiert, wenn sowohl das

Bewertungssignal als auch die Eingaben in das Netzwerk stationären Prozessen gehorchen. Letzteres wird im folgenden stets vorausgesetzt.

Definition 6.4 (REINFORCE-Algorithmen, Williams 92) *Sei $\mathcal{A}$ ein Lernalgorithmus, der die Parameter W eines KNN nach*

$$\Delta w_{ij} = \alpha_{ij}(r - b_{ij})e_{ij}$$

adaptiert, wobei α_{ij} eine Lernrate, b_{ij} einen Basiswert und $e_{ij} = \frac{\partial \ln p_i}{\partial w_{ij}}$ den Durchgriff *(engl. Eligibilität) auf Parameter w_{ij} angibt ($w_{ij} \in W$ bezeichnet dabei einen Parameter, der die Dichtefunktion p_i beeinflußt). Sei ferner der Basiswert b_{ij} unabhängig von der Ausgabe y_i bei gegebenem W und x, und sei die Lernrate α_{ij} nicht negativ und höchstens von w_{ij}, den Parametern, die die Dichtefunktion p_i bestimmen, abhängig. Dann ist $\mathcal{A}$ ein REINFORCE-Algorithmus, wobei REINFORCE für "*REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility*" steht.*

Für diese Klasse von Lernalgorithmen konnte Williams [137] folgenden wichtigen Satz zeigen:

Satz 6.5 (Williams 92) *Für jeden REINFORCE-Algorithmus ist das innere Produkt von $E[\Delta W|W]$ und $\nabla_W E[r|W]$ nicht negativ. Ist $\alpha_{ij} > 0$ für alle i, j, dann ist dieses innere Produkt null gdw $\nabla_W E[r|W] = 0$. Ist α_{ij} unabhängig von i, j, d.h. $\alpha_{ij} = \alpha$, dann ist $E[\Delta W|W] = \nabla_W E[r|W]$.*

Dieser Satz besagt, daß für jeden REINFORCE-Algorithmus die mittlere Änderung der Parameter in Richtung steigender Bewertung erfolgt und somit das Lernverfahren zu einem lokalen Optimum konvergiert. Für den Fall konstanter Lernraten ist $\alpha_{ij}(r - b_{ij})e_{ij}$ ein unverzerrter Schätzer für $\frac{\partial E[r|W]}{\partial w_{ij}}$.
Williams konnte zeigen, daß eine ganze Reihe bekannter Reinforcement-Lernverfahren in die Klasse der REINFORCE-Algorithmen fallen, so z.B. Narendra's L_{R-I} lernende Automaten [93], Barto's A_{R-P} Algorithmen [8] und Sutton's *Reinforcement-Comparison* Verfahren mit adaptivem Basiswert [120].

Gaußsche Ausgabeneuronen Als stochastische Ausgabeneuronen sind insbesondere solche mit einer Gaußschen Dichteverteilung interessant, im folgenden Gaußneuronen genannt, denn die Standardabweichung d steuert hier direkt das Explorationsverhalten um die mittlere Stellgröße μ. Gaußneuronen kommen auch in unserem DCS-basierten Reinforcement-Lerner zum Einsatz. Mit der Gaußverteilung

$$p(y|\mu, d) = \frac{1}{\sqrt{2\pi}d} e^{-\frac{(y-\mu)^2}{2d^2}} \tag{6.52}$$

ergibt sich der Durchgriff auf den Parameter μ zu

$$e_\mu = \frac{\partial \ln p}{\partial \mu} = \frac{(y - \mu)}{d^2} \tag{6.53}$$

und der Durchgriff auf den Parameter d zu

$$e_d = \frac{\partial \ln p}{\partial d} = \frac{(y-\mu)^2 - d^2}{d^3}. \tag{6.54}$$

Damit haben REINFORCE-Algorithmen für Gaußneuronen die Form

$$\Delta\mu = \alpha_\mu (r - b_\mu) \frac{(y-\mu)}{d^2} \tag{6.55}$$

und

$$\Delta d = \alpha_d (r - b_d) \frac{(y-\mu)^2 - d^2}{d^3}. \tag{6.56}$$

Das Explorationsverhalten von Gaußneuronen (vgl. [137]) läßt sich nun aufgrund (6.55) und (6.56) wie folgt erklären: Wird ein Bewertungssignal empfangen, das über dem Basiswert liegt, so wird μ in Richtung des gemäß (6.52) zufällig generierten Stellwerts y adaptiert. Andernfalls wird μ in entgegengesetzter Richtung verändert[8]. Auch der Parameter d wird von REINFORCE-Algorithmen in ähnlich "vernünftiger" Weise adaptiert. Übertrifft das Bewertungssignal den Basiswert, dann wird d verkleinert, falls $|y - \mu| < d$, und vergrößert, falls $|y-\mu| > d$. Damit wird die Standardabweichung so verändert, daß die positiv bewertete Aktion y wahrscheinlicher wird. Umgekehrt bewirkt ein unterdurchschnittliches[9] Bewertungssignal eine Adaption von d derart, daß y weniger wahrscheinlicher wird. Damit wird der Suchbereich um μ verkleinert, falls ein besserer Stellwert in unmittelbarer Nähe von μ gefunden wird, oder ein schlechterer weit entfernt von μ. Der Suchbereich wird vergrößert, falls ein schlechterer Stellwert in unmittelbarer Nähe von μ gefunden wird, oder ein besserer weit entfernt von μ. Wenn daher μ einmal einen Wert in der Nähe eines im Verhältnis zu d genügend breiten Maximums angenommen hat, verkleinert sich d automatisch, schränkt so die Exploration immer mehr ein und erlaubt die Konvergenz des Lernverfahrens. Auf diese Funktion der Standardabweichung als gleichzeitiger Explorations-Steuerparameter und Konvergenzindikator wird bei der Konstruktion des DCS-basierten Reinforcement-Lerners noch zurückgekommen.

Kombination mit AHC(λ)-Lernen Nach Satz 6.5 muß der Basiswert in REINFORCE-Algorithmen vom Stellwert y und damit vom Bewertungssignal r unabhängig sein. Dies ist konsistent mit dem *Reinforcement-Comparison* Ansatz von Sutton [121], bei dem der Basiswert als gleitendes Mittel vergangener Bewertungssignale berechnet wird, beispielsweise nach

$$b(t) = \bar{r}(t) = \gamma\, r(t-1) + (1-\gamma)\bar{r}(t-1). \tag{6.57}$$

Der Basiswert darf auch von x abhängen und damit beispielsweise als das mittlere zu erwartende Bewertungssignal im Zustand x durch ein weiteres KNN berechnet werden,

$$b(x) = \bar{r}(x) = y_{NN}(x). \tag{6.58}$$

[8]Eine ähnliche Vorgehensweise wird von einer Vielzahl von Reinforcement-Lernverfahren genutzt, meist allerdings ohne theoretische Begründung.

[9]Der Basiswert kann als gleitender Mittelwert vergangener Bewertungssignale berechnet werden (siehe 6.3.1).

Die naheliegende Kombination mit AHC-Lernen nach

$$\Delta w_{ij} = \alpha_{ij}\, err_{TD}(x_t)\, e_{ij} \tag{6.59}$$

fällt jedoch nicht in die Klasse der REINFORCE-Lernverfahren, denn mit $err_{TD}(x) = \gamma V(x_{t+1}) + r_t - V(x_t)$ nach (6.15) ergibt sich als adaptiver Basiswert

$$b(x) = V(x_t) - \gamma V(x_{t+1}), \tag{6.60}$$

wobei $V(x_{t+1})$ und damit der Basiswert unmittelbar vom Stellwert y_t abhängen. Wäre $b(x)$ nach obiger Gleichung ein REINFORCE-Algorithmus, so hätte sich hier eine Möglichkeit eröffnet, einen garantiert konvergenten Algorithmus auch für verzögertes Reinforcement-Lernen abzuleiten. In Ermangelung alternativer Lernverfahren zum Lernen aus verzögertem Reinforcement und aufgrund positiver experimenteller Ergebnisse wird dennoch häufig AHC(λ)- mit REINFORCE-Lernen verknüpft (siehe z.B. [102]). Auch der DCS-basierte Reinforcement-Lerner wird diese Kombination nutzen. Die Lerngleichungen für diese im folgenden als *AHCR*-Verfahren bezeichneten Algorithmen lauten:

$$\Delta w_{ij}^{Regler} = \alpha_{ij}^{Regler}\, err_{TD}(x_t)\, e_{ij} \tag{6.61}$$

zur Adaption der Parameter des Reglers (der Politik-Lernkomponente) und

$$\Delta w_{ij}^{Prädiktor} = \alpha_{ij}^{Prädiktor} err_{TD}(x_t) \sum_{k=1}^{t} (\lambda\gamma)^{t-k} \nabla_{w_{ij}^{Prädiktor}} V(x_k) \tag{6.62}$$

zur Adaption der Parameter des Prädiktionsnetzwerkes nach der TD(λ)-Regel, wobei die Summe in (6.62) wiederum effizient über Aktivitätsspuren berechnet werden kann.

6.3.2 REINFORCE-Lernen mit DCS

Abbildung 6.12 zeigt die DCS-basierte Lernarchitektur zum Lernen aus reellwertigem, verzögertem Reinforcement in reellwertigen Zustands- und Aktionsräumen. Sie besteht aus einem DCS-Netzwerk, das Politik- und Prädiktionskomponente integriert, und einer Schicht stochastischer Gaußneuronen, die hier der von Sutton eingeführten Terminologie folgend als *Adaptive Search Element* (ASE) bezeichnet wird. Die Adaption der Netzwerkparameter (Ausgabeschicht und Zentren) erfolgt nach einem AHCR-Verfahren, das AHC- und REINFORCE-Lernen kombiniert (siehe Abschnitt 6.3.1). Die Generierung einer Stellgröße u, gegeben den momentanen Zustand s, geschieht folgendermaßen: Zuerst werden nach dem normalisierten DCS-Schema die Aktivitäten der zustandskodierenden RBF-Knoten berechnet[10]:

$$a_i = \frac{h(\|s - c^i\|/\sigma)}{\sum_{j \in Nh^{(1)}(bmu_s)} h(\|s - c^j\|/\sigma)}\,. \tag{6.63}$$

[10] Um Verwirrung zwischen Vektoren und Skalaren zu vermeiden, wird für den Rest dieses Kapitels vereinbart, daß für eine Aufzählung von Vektoren Superskripte (x^i) und für den Zugriff auf ihre Komponenten Subskripte (x_j) verwendet werden.

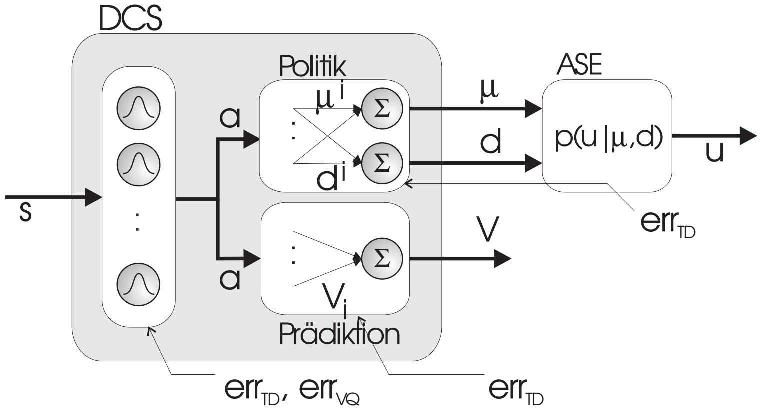


Abbildung 6.12: DCS-basierte Reinforcement-Lernarchitektur

Diese werden dann dazu genutzt, sowohl *prototypische Stellgrößen* μ und *Suchbereiche* d als auch eine Prädiktion des zu erwartenden diskontierten kumulativen Reinforcements V zu berechnen:

$$\mu = \sum_{i \in Nh^{(1)}(bmu_s)} a_i \, \mu^i \tag{6.64}$$

$$d = \sum_{i \in Nh^{(1)}(bmu_s)} a_i \, d^i \tag{6.65}$$

$$V = \sum_{i \in Nh^{(1)}(bmu_s)} a_i \, V_i. \tag{6.66}$$

Die prototypischen Stellgrößen und Suchbereiche dienen nun als Parameter (Mittelwerte und Standardabweichungen) der stochastischen Gaußneuronen der ASE, die schließlich die Stellgrößen u gemäß ihrer parametrisierten Gaußschen Dichtefunktion erzeugen:

$$p(u_j|s) = p_j(u_j|\mu_j(s), d_j(s)) = \frac{1}{\sqrt{(2\pi)}d_j} e^{-\frac{(u_j-\mu_j)^2}{2d_j^2}}. \tag{6.67}$$

Die DCS-basierte Reinforcement-Lernarchitektur nach Abbildung 6.12 bietet eine Reihe von vorteilhaften Eigenschaften. Hier ist zuerst die *Integration von Politik- und Prädiktionskomponente* in ein einziges Netzwerk zu nennen, die das in manchen konkurrierenden Ansätzen (z.B. [13]) anzutreffende Problem der Verwaltung und des koordinierten Trainings zweier getrennter Netze umgeht. Diese Integration wird dadurch erzielt, daß dieselbe Zustandsrepräsentation (Kodierung durch die RBF-Knoten) zur Aktionsberechnung und Zustandsbewertung genutzt wird. Es ist auch wenig sinnvoll, getrennte Kodierungen zu nutzen, denn in einem bezüglich der Ertragsfunktion nur grob aufgelösten Teil des Zustandsraumes kann keine differenzierte Politik ausgebildet werden, und umgekehrt kann sich bei einer grobkörnigen Politik keine differenzierte Ertragsfunktion ergeben.

Eine zweite bemerkenswerte Eigenschaft ist die *Verwendung individueller Suchbereichsvektoren* d^i, die mit jedem RBF-Knoten neben ihrem prototypischen Stellvektor μ^i assoziiert werden. In Zusammenhang mit den Gaußneuronen der ASE erlauben sie einerseits, den Suchbereich für die Modifikation der prototypischen Aktionen vorzugeben, andererseits läßt ihre Entwicklung bei Adaption im Rahmen eines REINFORCE-Verfahrens Rückschlüsse auf die Konvergenz des Lernvorganges zu (vgl. 6.3.1).

Drittens erlaubt die vorgestellte Architektur, nicht nur vorhandenes Wissen in Form von Zentren der RBF-Knoten und assoziierten prototypischen Stellgrößen und Suchbereichen zu *adaptieren*, sondern auch neues Wissen in Form von neuen RBF-Knoten und assoziierten Ausgaben zu *generieren*.

Schließlich erlaubt die Nähe der RBF- und DCS-Netze zur Fuzzy-Logic die konsistente Interpretation als Reinforcement-lernenden Fuzzy-Regler, worauf in Abschnitt 6.3.3 noch näher eingegangen wird.

Durch Angabe der Lernregeln für das DCS-Netz soll nun die Adaption des Reinforcement-Lerners, wie sie zum Erlernen einer Hindernisvermeidungsstrategie in Abschnitt 6.3.4 erfolgt, beschrieben werden.

Adaption der Ausgabeschicht

Die Parameter der Ausgabeschicht, die prototypischen Stellgrößen μ^i, Suchbereiche d^i und Ertragsschätzer V_i werden online nach dem in Abschnitt 6.3.1 beschriebenen AHCR-Verfahren adaptiert. Dabei erfolgt die Adaption der prototypischen Stellgrößen und Suchbereiche nach der REINFORCE-Lernregel (6.61), die der Ertragsschätzer nach der TD(λ)-Regel (6.62). Für die Adaption der Ertragsschätzer des DCS-Netzes ergibt sich damit:

$$\begin{aligned}
\Delta V_i &= \alpha_V \, err_{TD}(s_t) \sum_{k=1}^{t} (\lambda\gamma)^{t-k} \nabla_{V_i} V(s_k) && (6.68) \\
&= \alpha_V \left(\gamma V(s_{t+1}) + r_t - V(s_t)\right) \sum_{k=1}^{t} (\lambda\gamma)^{t-k} a_i(s_k). && (6.69)
\end{aligned}$$

Natürlich wird dabei die Summe in (6.68) wieder rekursiv über Aktivitätsspuren berechnet. Für die Adaption der prototypischen Stellgrößen und Suchbereiche folgt

$$\begin{aligned}
\Delta \mu_j^i &= \alpha_\mu \, err_{TD}(s_t) \frac{\partial \ln p_j(u_j|\mu_j, d_j)}{\partial \mu_j} \frac{\partial \mu_j}{\partial \mu_j^i} && (6.70) \\
&= \alpha_\mu \left(\gamma V(s_{t+1}) + r_t - V(s_t)\right) \frac{(u_j - \mu_j)}{d_j^2} a_i && (6.71)
\end{aligned}$$

und

$$\begin{aligned}
\Delta d_j^i &= \alpha_d \, err_{TD}(s_t) \frac{\partial \ln p_j(u_j|\mu_j, d_j)}{\partial d_j} \frac{\partial d_j}{\partial d_j^i} && (6.72) \\
&= \alpha_d \left(\gamma V(s_{t+1}) + r_t - V(s_t)\right) \frac{(u_j - \mu_j)^2 - d_j^2}{d_j^3} a_i. && (6.73)
\end{aligned}$$

Adaption der Zentren

Die Adaption der Zentren erfolgt nach der fehlermodulierten Lernregel (3.35):

$$\Delta c^i = \epsilon_i \, g(\frac{r_i}{r_{bmu_s}}) \, (s_t - c^i) \; , \; i \in Nh^{(1)}(bmu_s) \tag{6.74}$$

mit $\epsilon_i = \epsilon_{bmu}$ für die bmu_s und $\epsilon_i = \epsilon_{Nh}$ für die direkten Nachbarn der bmu_s und Fehlermodulationsfunktion $g(\frac{r_i}{r_{bmu_s}})$. Als Fehlerquelle für die Fehlervariablen r_i dient aufgrund mangelnder Kenntnis der optimalen Stellgrößen der quadrierte TD-Fehler, $err^2_{TD}(s_t)$. Man beachte, daß diese Verwendung des TD-Fehlers erst durch die Kopplung von Politik- und Prädiktionskomponente über die gemeinsame zustandskodierende RBF-Schicht ermöglicht wird: Ein hoher (lokaler) TD-Fehler entsteht dann, wenn der Zustandsraum (lokal) zu grob aufgelöst ist, so daß gleiche Aktionen zu sehr unterschiedlichen Bewertungen führen. So führt eine Adaption zur Minimierung des TD-Fehlers zu einer geeigneten Repräsentation auch der Politik.

Wachstum des Netzes

Das Wachstum des Netzes erfolgt nach einer gemischten fehler- und distanzgetriebenen Strategie (Seiten 57 und 59), wobei die distanzgetriebene Strategie zur schnellen Überdeckung des Zustandsraumes genutzt wird. Fehlergetriebene Einfügungen finden nach jeweils λ_{ins} Schritten statt (Seite 58), die maximale Anzahl von Knoten wird auf N_{max} beschränkt. Dabei erfolgt die Schätzung des lokalen Fehlers und damit die Adaption der Fehlervariablen durch gleitende Mittelwertbildung nach (3.47) auf Grundlage des quadratischen TD-Fehlers $err^2_{TD}(s_t)$. Die Initialisierung neuer Knoten erfolgt nach der Interpolationsmethode (3.51). Mit der gleichen Argumentation wie bei der Zentrenadaption wird diese Wachstumsstrategie nur durch die Kopplung von Politik- und Prädiktionskomponente über die gemeinsame RBF-Schicht möglich.

Adaption der Topologie

Die Adaption der Topologie erfolgt nach der fürs Online-Lernen entwickelten unsymmetrischen Hebb-Regel (3.41).

6.3.3 Interpretation als Lernender Fuzzy-Regler

Es sei an dieser Stelle noch einmal explizit auf die Beziehung des DCS-basierten reellwertigen Reinforcement-Lernens zu adaptiven Fuzzy-Reglern hingewiesen, da gerade auf diesem Gebiet eine ganze Reihe von Reinforcement-Lernverfahren vorgeschlagen wurden (siehe Abschnitt 6.3.5). In Abschnitt 3.3.5 wurde dargelegt, daß DCS-Netzwerke als Fuzzy-Regler vom Sugeno-Typ interpretiert werden können, bei denen nur die euklidisch nächsten Regeln zum Inferenzprozeß beitragen. Unter diesem Blickwinkel stellt der präsentierte DCS-basierte Lerner einen Fuzzy-Regler dar, der mit a priori Wissen in Form von Fuzzy-Regeln initialisert, dieses Wissen mittels Lernen aus verzögertem Reinforcement verfeinert und zusätzlich neues Wissen (neue Regeln) inkrementell generiert [22].

6.3.4 Lernende Hindernisvermeidung mit einem TRC Labmate

Als Anwendung des DCS-basierten Reinforcement-Lernverfahrens dient das Erlernen eines reaktiven Hindernisvermeidungsverhaltens mit der mobilen TRC Labmate Plattform [24, 2]. Die Aufgabe besteht darin, eine reellwertige sensomotorische Abbildung zu lernen, die unvorverarbeitete Daten von acht Ultraschallsensoren mit einer zur Hindernisvermeidung geeigneten Vorwärts- und Drehgeschwindigkeit assoziiert. Da das Lernverhalten unabhängig von Vorwissen und anderen Lernmechanismen getestet werden soll, startet der Lerner ohne Vorwissen und lernt alleine auf Basis von (verzögerten) Reinforcement-Signalen. Wie sich herausstellt, ist diese Art des Lernens jedoch sehr langsam und führt in der Anfangsphase zu häufigen Kollisionen. Deshalb wurden die Experimente in Simulationen durchgeführt. Deren Ziel ist es, nachzuweisen, daß a) der Lerner tatsächlich sein Verhalten durch Adaption und Regelgenerierung verbessert und b) er keine plötzlichen Kompetenzeinbrüche (plötzliches Verlernen) zeigt.

Simulationen eines physikalischen Systems sind unter der Bedingung sinnvoll, daß das simulierte Lernproblem dieselbe Charakteristik und dieselbe Komplexität wie das Realwelt-Problem aufweist. Diese Kriterien wurden durch gewissenhafte Modellierung sowohl der Dynamik des Labmates als auch der Ultraschallsensoren zu erfüllen versucht. So wurde der Simulator in [2] auf der Basis von handkodierten Regeln mit dem Verhalten des realen Roboters verglichen und zeigte eine gute Übereinstimmung.

Die Simulation

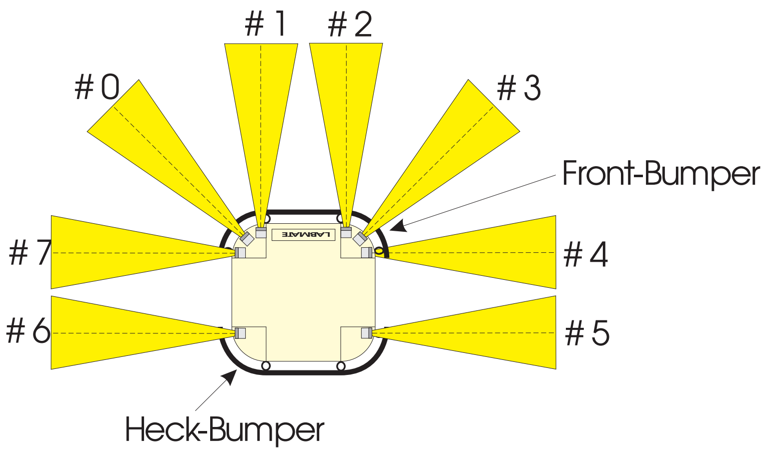


Abbildung 6.13: Skizze des simulierten TRC Labmates

Abbildung 6.13 zeigt den simulierten TRC Labmate mit acht Ultraschallsensoren und zwei taktilen Sensoren (Bumper). Der Roboter hat einen differentiellen Antrieb mit zwei aktiven Rädern auf einer zentralen Achse, der ihm erlaubt, auf dem Punkt zu drehen. Er wird lediglich durch Vorgabe der Vorwärts- und Drehgeschwindigkeit durch den Reinforcement-Lerner gesteuert. Nur im Fall drohender (sehr kleine Sensorwerte) oder erfolgter Kollision wird der Roboter durch ein Notfallverhalten gestoppt, setzt gegebenenfalls zurück und rotiert dann solange auf der Stelle, bis er einen Freiraum findet. Die Polaroid Ultraschallsensoren wurden durch 20° breite Kegel modelliert, die Dynamik aufgrund der Handbücher [126] simuliert. Weitere Details zur Simulation finden sich in [2].

Der simulierte Labmate wird in einer Trainingsumgebung plaziert, in der er jeweils solange frei (d.h. nur unter Einfluß der adaptiven Steuerung) umherfahren darf, bis er einem Hindernis gefährlich nahe gekommen ist ($\leq$ 20cm) oder 200 Zeitschritte vergangen sind. Im ersten Fall wird er aufgrund seines Notfallverhaltens neu orientiert, im zweiten Fall erfolgt ein Stop und eine zufällige Neuorientiertung des Roboters. Diese erzwungenen Stops und die zufälligen Neuorientierungen halten den Roboter davon ab, ständig auf derselben einmal gefundenen kollisionsfreien Bahn durch die Testumgebung zu fahren und so täuschend gute Lernergebnisse ohne Generalisierungsfähigkeit zu erzielen. Ein Nachteil dieser Trainingsmethode ist, daß der Roboter durch die zufällige Drehung häufig in Situationen kommt, die die adaptive Steuerung vermieden hätte und in denen dann eventuell nur noch ein Notstop eine Kollision verhindert. Dies führt zu einer negativen Verzerrung des Lernverhaltens und stellt eine zusätzliche Schwierigkeit für den Lernalgorithmus dar.

A Priori Wissen Da das Lernverhalten der adaptiven Steuerung unabhängig von a priori Wissen zu testen ist, startet die Steuerung mit einer einzigen Fuzzy-Regel als Vorwissen:

$$\begin{aligned}&\textbf{if } (s_0 \approx 5000) \wedge \cdots \wedge (s_7 \approx 5000) \textbf{ then}\\ &\qquad (\mu_v(s) \approx 400) \wedge (d_v(s) \approx d_v^0) \wedge (\mu_\omega(s) \approx 0) \wedge (d_\omega(s) \approx d_\omega^0) \wedge (V(s) \approx 0).\end{aligned}$$

Diese besagt, daß bei einem relativ großen Abstand nach allen acht Seiten (alle Ultraschallsensoren zeigen Entfernungswerte von ca. 5m) geradeaus (Rotationsgeschwindigkeit $\omega \approx 0\,^\circ\, s^{-1}$) mit einer Vorwärtsgeschwindigkeit von $v \approx 0.4\, m\, s^{-1}$ gefahren wird. Als initiale Suchbereiche zur Adaption der Rotations- und Vorwärtsgeschwindigkeit werden relativ kleine Standardabweichungen von $d_v^0 = 1$ und $d_\omega^0 = 1$ vorgegeben.

Wahl der Reinforcement-Funktion Als Reinforcement-Funktion wird eine Kombination aus unverzögertem und verzögertem Reinforcement gewählt,

$$r_t = r_t^{unverz} + r_t^{verz}.$$

Das unverzögerte Reinforcementsignal wird dabei als Differenz der Zustandsbewertungen zweier aufeinanderfolgender Zustände berechnet,

$$r_t^{unverz} = \psi(s_{t+1}) - \psi(s_t).$$

Dabei ist $\psi : s \in \mathbb{R}^8 \rightarrow \psi(s) \in \mathbb{R}$ die Zustandsbewertungsfunktion. Diese soll einen Zuwachs an Abstand zu frontalen und halb seitlichen Hindernissen belohnen und berechnet sich als Konvexkombination nichtlinearer Transformationen $\Phi(s_i)$ der acht Ultraschall-Entfernungsmessungen s_i mit Gewichtungsfaktoren $c_i = 1$ für $i = 0, \ldots, 3$ und $c_i = 0$ für $i = 4, \ldots, 7$:

$$\psi(s) = \frac{\sum_{i=1}^{8} c_i \Phi(s_i)}{\sum_{i=1}^{8} c_i}.$$

Als nichtlineare Transformation $\Phi : x \in \mathbb{R} \to \Phi(x) \in \mathbb{R}$ wird die Quadratwurzel des normierten Abstandwertes

$$\Phi(x) = \sqrt{\frac{x - x_{min}}{x_{max} - x_{min}}}$$

verwendet, wobei $x_{min} = 20\ cm$ den minimal meßbaren Abstand eines Ultraschallsensors angibt und $x_{max} = 2\ m$ die (über den "time out" Wert einstellbare) maximal meßbare Distanz. Die nichtlineare Transformation wird nötig, um den Roboter zum Fahren mit maximalen Seitenabständen zu motivieren: Bei einer einfachen Addition der Sensorwerte würde beispielsweise beim Fahren in einem Korridor der Zuwachs der Bewertung durch Abdriften von der einen Wand des Korridors von der Abnahme der Bewertung durch Annäherung an die andere Wand gerade kompensiert. Durch die nichtlineare Kombination hingegen ergibt sich ein Maximum der Zustandsbewertung gerade in der Mitte des Korridors, wie mittels eines Lagrangschen Multiplikators gezeigt werden kann.

Das verzögerte Reinforcementsignal schließlich ist null, solange der Roboter nicht sein Notfallverhalten zur Vermeidung einer unmittelbar drohenden Kollision aktivieren muß. In diesem Fall nimmt es den im Vergleich zum unverzögerten Reinforcement recht hohen Wert von $r^{verz} = -100$ zur Bestrafung der zur Kollision führenden Aktionenfolge an.

Experimentelle Resultate

Die experimentellen Resultate (vgl. [2]) sind in den Abbildungen 6.14 bis 6.16 zusammengefaßt. Abbildung 6.14 zeigt das Verhalten des Roboters zu Beginn der Lernphase. Der Roboter fährt mit nahezu konstanter Geschwindigkeit von Kollision zu Kollision. Nach ca. 10000 Versuchen (ein Versuch endet mit einer Kollision oder der zufälligen Neuorientierung nach 200 Zeiteinheiten) hat die adaptive Steuerung jedoch eine Hindernisvermeidungsstrategie erlernt, die den Roboter kollisionsfrei durch die Trainingsumgebung fahren läßt, siehe Abbildung 6.15. Daß diese Hindernisvermeidungsstrategie tatsächlich umgebungsunabhängig ist, das Netz also nicht die Trainingsumgebung "auswendig" gelernt hat, zeigt Abbildung 6.16: Der Roboter ist zur Hindernisvermeidung auch in unbekannten Testumgebungen fähig.

Abbildungen 6.17 bis 6.19 verdeutlichen den zeitlichen Verlauf des Lernvorganges[11]. Dabei zeigt Abbildung 6.17 die gemittelte Anzahl kollisionsfreier Schritte, Abbildung 6.18 den gemittelten Verlauf des TD-Fehlers und Abbildung 6.19 den Verlauf des gemittelten unverzögerten Reinforcements. Gemittelt wird dabei jeweils über 100 Versuche. Die Abbildungen zeigen den zeitlichen Verlauf dieser Größen sowohl für realistisch simulierte Ultraschallsensoren mit einem Öffnungswinkel der Ultraschallkeulen von 20° und gaußverteiltem Rauschen mit einer Standardabweichung von 5 cm auf den Meßwerten als auch für idealisierte Sensoren (Öffnungswinkel 0° und kein Rauschen).

Zu erkennen ist, daß sich die Anzahl der kollisionsfreien Schritte und das gemittelte unverzögerte Bewertungssignal assymptotisch einem Maximalwert und der TD-Fehler einem Minimalwert nähern, um den sie dann schwanken. Daß die Kurven nicht ihre theoretischen Maximal- bzw. Minimalwerte erreichen, liegt an den zufälligen Neuorientierungen, die, wie

[11] Die Parameter finden sich in Tabelle 8.14.

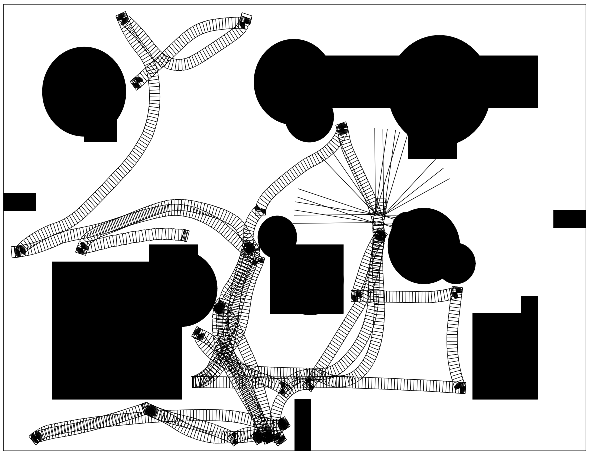


Abbildung 6.14: Beginn der Lernphase

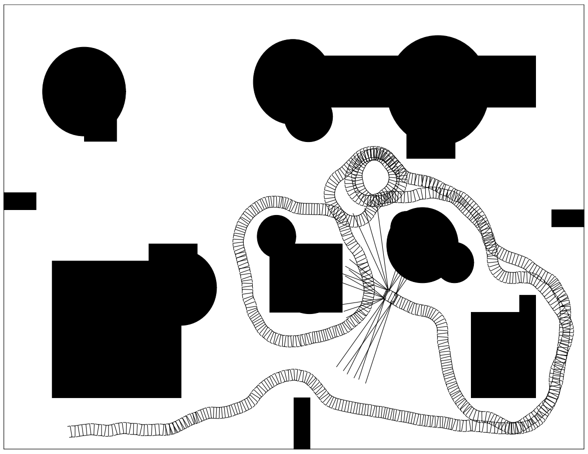


Abbildung 6.15: Erlerntes Verhalten

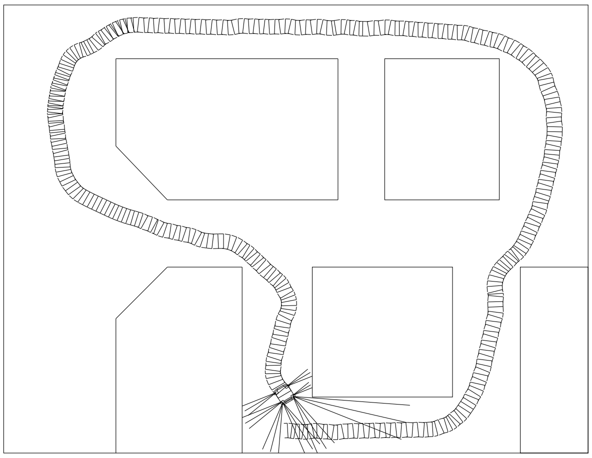


Abbildung 6.16: Erlerntes Verhalten in einer neuen Umgebung

oben diskutiert, zu Kollisionen führen können. Der Peak in den Kurven des TD-Fehlers und der Bewertung weisen auf ein Festfahren des Roboters in einer Ecke der Trainingsumgebung hin, aus der er sich dann aber wieder befreien konnte. Des weiteren verdeutlichen die Kurven, daß es zwischen den idealisierten und "realistisch simulierten" Ultraschallsensoren kaum einen qualitativen Unterschied hinsichtlich des Lernverhaltens gibt. Wichtig ist auch, daß die Kurven keine plötzlichen Kompetenzeinbrüche zeigen: Trotz Schwankungen um die asymptotischen Werte verlernt der Roboter sein Hindernisvermeidungsverhalten nicht mehr. Dabei ist noch einmal darauf hinzuweisen, daß der Lerner seine Plastizität über die gesamte "Lebensdauer" beibehält, also insbesondere keine Lernkonstanten eingefroren werden.

Betrachtet man die Zeit, die der simulierte Labmate zum Erlernen einer Hindernisvermeidungsstrategie benötigt, so ergäbe sich bei ca. 10000 Versuchen über durchschnittlich 100 Zeiteinheiten á 0.5 *sec* fast eine Woche ununterbrochenen Trainings für das reale System. Während es schwerfällt, diese Zeit im Rahmen des reellwertigen, verzögerten

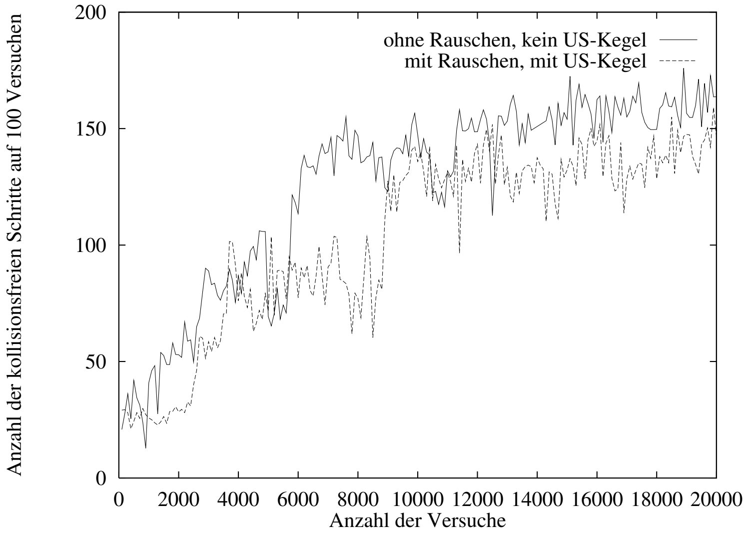


Abbildung 6.17: Schrittzahl

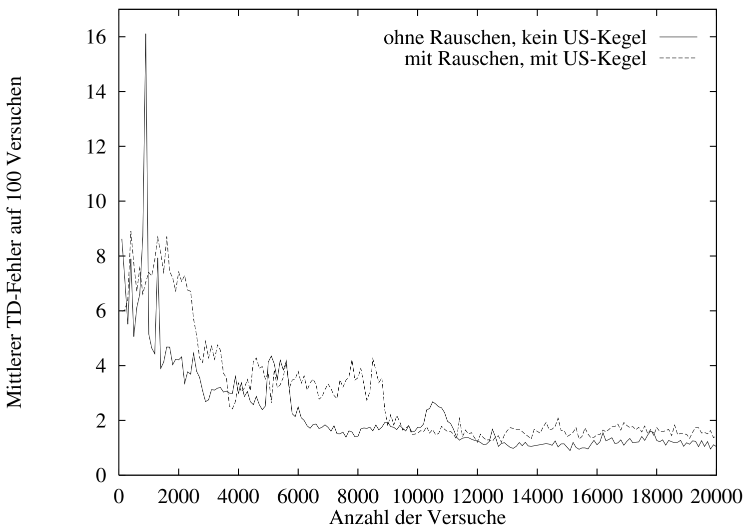


Abbildung 6.18: TD-Fehler

Reinforcement-Lernens zu beurteilen, ist sie doch offensichtlich zu lang, um einem realen AMR auf diese Weise ein Hindernisvermeidungsverhalten beizubringen. Für eine Anwendung unter Realweltbedingungen ist daher das Einbringen von Vorwissen und/oder die Kombination mit anderen Lernparadigmen aus Geschwindigkeits- und Sicherheitsgründen unerläßlich. Dieser Einsicht folgt auch Millan [87], der versucht, durch eine Kombination mit selbstüberwachtem Lernen das langsame Reinforcement-Lernen zu beschleunigen.

Das Erlernen einer Hindernisvermeidungsstrategie gelang in ca. 70% der Experimente. In den fehlgeschlagenen Experimenten gelang es dem Roboter nicht, innerhalb von 20000 Versuchen eine Hindernisvermeidungsstrategie zu erlernen. Der Roboter hatte sich in der Trainingsumgebung jeweils so festgefahren, daß er durch eine ganze Serie vieler kurz aufeinanderfolgender Kollisionen mit ständigem stark negativem Reinforcement in übertragenem

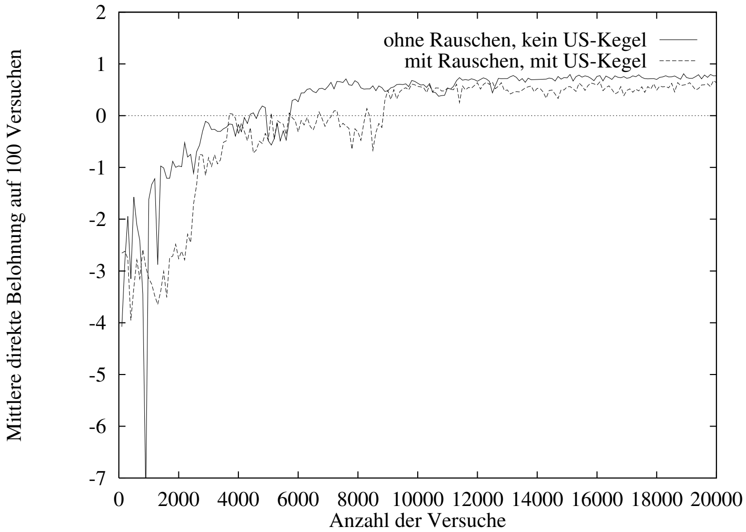


Abbildung 6.19: Bewertungen

aber treffendem Sinne "in den Wahnsinn" getrieben wurde.

Aufgrund des chaotischen[12] Verlaufs der Lernphase wurde im Rahmen der Experimente zur Hindernisvermeidung davon abgesehen, die Einflüsse der Lernparameter näher zu untersuchen. Kleine Änderungen der Lernparameter und (durch zufällige Exploration der Gaußneuronen) leicht unterschiedliche Aktionen können zu völlig verschiedenen Pfaden und damit sehr unterschiedlichen Ergebnissen führen, die einen Vergleich der zugehörigen TD-Fehler und Reinforcement-Verläufe als wenig sinnvoll erscheinen lassen. Die chaotische Abhängigkeit von der Exploration macht es auch schwierig, mittlere Performance-Kurven für einen Parametersatz aufzunehmen. Es ergäben sich sehr große Varianzen, und die Aussagekraft derart gemittelter Kurven wäre zu bezweifeln.

6.3.5 Alternative Ansätze

Das hier vorgestellte DCS-basierte Reinforcement-Lernen weist Ähnlichkeiten zu den Reinforcement-Lernverfahren von Berns [61] und Millan [87] auf:

Berns [61] setzt eine RBF-basierte Lernarchitektur zur adaptiven Steuerung einer Gehmaschine ein. Er benutzt ein dem in dieser Arbeit vorgestellten Ansatz ähnliches Reinforcement-Lernverfahren zur Adaption der prototypischen Aktionen und zeigt, wie RBF-basierte Reinforcement-Lerner zu hierarchischen Netzwerken zusammengefaßt werden können. Letztere Idee ist auf die DCS-basierten Reinforcement-Lerner übertragbar. Berns versucht jedoch weder, die Zentren der RBF-Knoten zu adaptieren, noch den TD-Fehler zur adaptiven Vektorquantisierung zu nutzen. Auch verzichtet er auf die Generierung und Nutzung topologieerhaltender Karten. Im Gegensatz zu Berns wird des weiteren auf die konzeptionelle Nähe der vorgestellten Architektur zur Fuzzy-Regelung hingewiesen, die die für das langsame Reinforcement-Lernen sehr hilfreiche Einbindung von a priori Wissen

[12]Chaotisch in dem Sinne, daß kleine Änderungen einer Größe im Verlauf des Trainings zu komplett anderen Lernverläufen führen können.

ermöglicht.

Millans Lernarchitektur [87] zum Reinforcement-Lernen einer zielgerichteten Hindernisvermeidungsstrategie für einen simulierten mobilen Nomad 200 Roboter basiert auf Alpaydins GAL-Architektur [5]. GAL beruht ebenfalls auf Berechnungsknoten mit lokalisierten rezeptiven Feldern, weicht aber durch Einführung sogenannter *Kategorie-Knoten* (engl. category units) von RBF-Netzwerken und Fuzzy-Reglern ab. Wie RBF-Netzwerke verzichtet GAL auf Generierung und Nutzung topologieerhaltender Karten. Millans Reinforcement-Lernverfahren verwendet TD(0)-Prädiktoren und orientiert sich an einem von Gullapalli [44] eingeführten Ansatz zum Reinforcement-Lernen, der von den hier erläuterten REINFORCE-Verfahren in der Adaption der Standardabweichungen (Suchbereiche) der Gaußneuronen differiert. Die Verwendung individueller Suchbereichsvektoren für jeden Knoten (jede Fuzzy-Regel) ist in diesem Ansatz nicht möglich. Millans Hauptbeitrag ist die Verwendung einer zusätzlichen Planungskomponente für selbstüberwachtes Lernen, die zur Beschleunigung des Reinforcement-Lernens genutzt wird. Seine Arbeit unterstreicht die Wichtigkeit der Integration unterschiedlicher Lernparadigmen und damit die Bedeutung einer Lernarchitektur, die diese Integration leisten kann.

Berenjis GARIC-Architektur [13] stellt das wohl bekannteste Beispiel eines durch Reinforcement lernenden hybriden Neuro-Fuzzy-Reglers (engl. hybrid neuro fuzzy controller, HNFC) dar. In GARIC werden die Parameter eines Netzes zur Aktionsauswahl (engl. Action Selection Network, ASN), welches den Fuzzy-Regler darstellt, mittels eines Aktionsevaluierungsnetzwerkes (engl. Action Evaluation Network, AEN) und einer stochastisch arbeitenden Komponente (engl. Stochastic Action Modifier, SAM) adaptiert. Während GARIC damit konzeptuell dem hier vorgestellten Ansatz ähnelt, bietet letzterer jedoch den Vorteil, ASN und AEN in einem DCS-Netzwerk zu integrieren, welches gleichermaßen der adaptiven Perzeptualisierung, dem Erlernen einer geeigneten Politik und der Generierung und Adaption der Prädiktionen dient. Im Gegensatz zu GARIC ist es mit dem hier vorgestellten Ansatz über die Adaption der Parameter hinaus möglich, komplett neue Regeln (Knoten) zu generieren. Des weiteren erscheint der hier gewählte AHCR-Ansatz durch seine Anlehnung an die REINFORCE-Algorithmen eleganter und weniger heuristisch als der in GARIC gewählte Zugang zum Reinforcement-Lernen.

Hindernisvermeidung mittels Ultraschallsensoren und (nicht adaptiven) Fuzzy-Logic-Reglern wurde unter anderen auch von Reignier [106] untersucht. Derselbe Autor berichtet in einer Anschlußarbeit [105] über die Generierung neuer Fuzzy-Regeln durch überwachtes Training, wobei sich viele seiner Aussagen direkt auf die Generierung neuer RBF-Knoten in DCS-Netzen übertragen lassen.

Schließlich weist unser Ansatz zur Hindernisvermeidung auch einen Bezug zu Potentialfeldansätzen [69] auf. Der Hauptunterschied liegt darin, daß die Potentialfelder, aus denen ein Vektorfeld zur hindernisvermeidenden Navigation abgeleitet wird, a priori festzulegen sind, während in dieser Arbeit versucht wird, die entsprechende sensomototrische Abbildung zu erlernen. Die von einem KNN mit universeller Approximationseigenschaft repräsentierten Vektorfelder unterliegen auch nicht denselben Beschränkungen wie den aus Potentialfeldern abgeleiteten, sie müssen beispielsweise nicht rotationsfrei sein.

Kapitel 7

Zusammenfassung und Ausblick

Ausgehend von der Forderung des verhaltensbasierten Entwurfsprinzips nach einer *integrierten Lernarchitektur*, d.h. einer Lernarchitektur, die ein Höchstmaß an Flexibilität und die Fähigkeit, die verschiedensten Lernparadigmen zu nutzen, vereint, wurde in dieser Arbeit die Dynamische Zellstruktur (DCS) entwickelt. Ihre Kennzeichen sind die Kombination radialer Basisfunktionen mit optimal topologieerhaltenden Karten sowie die fehlergesteuerte Adaption nicht nur der Modellparameter sondern auch der Modellordnung und der topologieerhaltenden Karten.

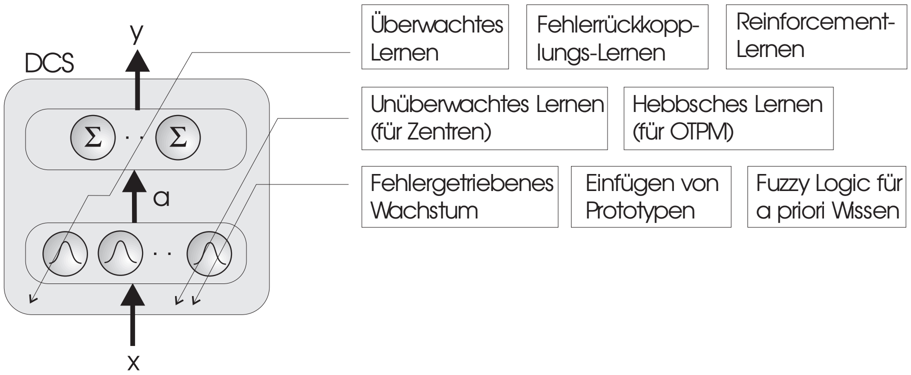


Abbildung 7.1: DCS-basierte integrierte Lernarchitektur zur Kombination verschiedener Lernparadigmen

Wie zusammenfassend in Abbildung 7.1 dargestellt, kann die lineare Ausgabeschicht eines DCS-Netzwerkes mit jeder Lernregel trainiert werden, die die Ableitung der Netzausgabe nach den mit den RBF-Knoten assoziierten Ausgabevektoren nutzt. Hierzu gehören überwachtes Lernen (siehe Kapitel 4), Fehlerrückkopplungs-Lernen (siehe Kapitel 5) und Reinforcement-Lernen (als Markovsches Entscheidungsproblem und in seiner reellwertigen Variante, siehe Kapitel 6). Da die Netzausgabe differenzierbar nach den Zentrumsvektoren der RBF-Knoten ist, können die aufgeführten Lernprinzipien durch Backpropagation auch zur Adaption der Zentren genutzt werden. Zusätzlich lassen sich die Zentren durch selbstorganisierende Lernregeln, wie z.B. die Fritzke-Regel, adaptieren, falls eine Minimierung des durch die Zentren induzierten Quantisierungsfehlers erwünscht ist. Im Unterschied zu RBF-Netzen kann hierbei auf die laterale Verbindungsstruktur zurückgegriffen werden, die ihrerseits durch Hebbsches Wettbewerbslernen zu einer optimal topologieerhaltenden Karte des Eingaberaumes konvergiert. Die ebenfalls mit den RBF-Knoten assoziierten Fehlervariablen lassen sich, wie vielfach in dieser Arbeit demonstriert, sowohl zu inkrementellem, fehlergetriebenem Wachstum als auch zur Fehlermodulation selbstorganisierender Lernregeln nutzen. Dabei muß die Fehlerfunktion im Gegensatz zu gradientenbasierten Verfahren nicht differenzierbar sein. Schließlich kann a priori Wissen in Form von Fuzzy-Logic-Regeln im Zuge einer Vorstrukturierung in das Netzwerk eingebracht werden, und das Einfügen prototypischer Situation/Ausgabe-Paare als neue RBF-Knoten erlaubt sogenanntes "one shot learning". Die DCS bietet damit ungleich mehr Kombinationsmöglichkeiten verschiedener Lernverfahren als MLP-Netze.

Die praktische Einsatzfähigkeit der DCS wurde in verschiedenen Anwendungen nachgewiesen. Diese reichen von der topologieerhaltenden Repräsentation von Eingabemannigfaltigkeiten über die überwachte Konstruktion diverser Klassifikatoren und das überwachte Lernen der Regelungsfunktion eines invertierten Pendels (Kapitel 4) bis zum Erlernen der Sakkadensteuerung eines Stereokamerakopfes und der visuomotorischen Koordination eines Roboterarmes durch Fehlerrückkopplungs-Lernen (Kapitel 5) sowie dem Erlernen einer Hindernisvermeidungsstrategie für einen autonomen mobilen Roboter durch reellwertiges, verzögertes Reinforcement (Kapitel 6). Obwohl auf Spezialhardware zur Simulation der DCS durchgängig verzichtet wurde, genügten Ausgabeberechnungs- und Adaptionsgeschwindigkeit der DCS den Echtzeitanforderungen aller aufgeführten Anwendungen.

Die Anwendungen zeigen aber auch, daß es ein universell einsetzbares Lernparadigma nicht gibt, und unterstreichen damit die Forderung nach der integrierten Lernarchitektur. So kommt z.B. reines Reinforcement-Lernen mit sehr spärlicher, evaluativer Information aus, ist aber für Realweltprobleme wie Szenarien autonomer Agenten zu langsam und, wie im Falle der Q-Lernversuche, auch zu unsicher. Der Schlüssel liegt in der Kombination der verschiedenen Lernparadigmen - ein Beispiel ist hier [87] mit der Verknüpfung von selbstüberwachtem Lernen und Reinforcement-Lernen - und der massiven Einbringung und Ausnutzung von Vorwissen und Randbedingungen. In dieser Arbeit wurde gezeigt, daß die DCS hierfür einen geeigneten Rahmen bietet, dessen Ausnutzung noch viel Gestaltungsraum für Anschlußarbeiten läßt.

Die Akzeptanz von DCS und KNN im allgemeinen als adaptive Module in verhaltensbasierten autonomen Systemen wird dabei ganz wesentlich von Antworten zur Lösung des Stabilität/Plastizität-Dilemmas im Zusammenhang mit der sogenannten "Lifelong Learning" [124] Problematik abhängen. Der in dieser Arbeit eingeschlagene Weg, die Adaptivität des Systems an dessen Performanz zu koppeln (Beispiele sind hier die fehlermodulierte Lernregel und die fehlergetriebene Modelladaption), erscheint vielversprechend, bedarf aber einer weiteren theoretischen und experimentellen Untermauerung.

Schließlich erscheint auch das Potential der optimal topologieerhaltenden Karten noch nicht erschöpft. Werden diese von der DCS im wesentlichen zu einer beschleunigten Ausgabeberechnung und Adaption genutzt, wobei sie durch Nachbarschaftskooperation auch zu einer Regularisierung des Lernproblems und einer effektiven Projektion auf die Eingabemannigfaltigkeit beitragen, so basiert die DCS doch weiterhin auf Interpolation durch radiale Basisfunktionen. Wie erste Resultate laufender Untersuchungen zeigen, lassen sich optimal topologieerhaltende Karten auch zur Schätzung der intrinsischen Dimensionalität der Eingabemannigfaltigkeit einsetzen. Dies ermöglicht die lokal lineare Approximation der Eingabemannigfaltigkeit und die Konstruktion an die intrinsische Dimensionalität angepaßter Basisfunktionen, was eine weitere Verbesserung der Approximations- und Generalisierungseigenschaften erwarten läßt. Ein solches, nur von der intrinsischen Dimensionalität abhängiges Approximationsschema ist insbesondere für das Erlernen visueller Aufgaben interessant. Am Beispiel des Ansichten-basierten Lernens [92] wird deutlich, daß die Eingaben (Bilder) trotz eines sehr hochdimensionalen Eingaberaumes (dem Bildraum) typischerweise auf niedrigdimensionalen Eingabemannigfaltigkeiten liegen. Erste positive Ergebnisse stützen die Erwartungen auch in diese Richtung.

Kapitel 8

Anhang

Parameter

Dieser Anhang faßt die in den Kapiteln 4 bis 6 zur Adaption der KNN genutzten Lernparameter in tabellarischer Form zusammen. Der Bestimmung der Lernparamater für die DCS-Netze liegt dabei keine "Magie" zugrunde, die Parameter unterliegen im Gegenteil einer Vielzahl von Randbedingungen, die den Suchraum erheblich einschränken. Die Erfahrung zeigt, daß sich unter Berücksichtigung dieser Randbedingungen meist auf Anhieb Parameterkombinationen für ein fehlerminimierendes Online-Training finden lassen. Kleinere Variationen der Parameter führen dabei erfahrungsgemäß nur zu geringfügigen Änderungen des Lernverhaltens. Für optimale Lernergebnisse freilich ist das Tuning der Parameter unerläßlich. Dieses Tuning erfolgte im Rahmen dieser Arbeit durch systematisches Ausprobieren verschiedener Parameterkombinationen. Genetische Algorithmen für eine automatische Parameterselektion bieten eine Alternative, deren Nutzen aber aufgrund der a priori möglichen starken Einschränkungen des Suchraums fraglich erscheint[1]. Randbedingungen an die Wahl der Parameter sind:

- Parameter für stochastische Gradientenabstiegsregeln, die aber nicht der Bedingung (2.9) gehorchen, sondern zugunsten der Plastizität konstant gehalten werden, sollten klein gewählt werden (i.allg. $<< 1$) um Instabilitäten zu vermeiden (Parameter α, ϵ_i, β).

- Die Adaption der Zentren, von denen die Netzausgabe nichtlinear abhängt, muß erfahrungsgemäß vorsichtiger als die der (linearen) Ausgabeschicht erfolgen (i.allg. $\alpha > \epsilon_i$).

- Die Zentrenadaptionsparameter ϵ_{bmu} und ϵ_{Nh} sind durch $\epsilon_{bmu} > \epsilon_{Nh}$ verknüpft (Daumenregel: $\epsilon_{Nh} \approx 0.1\epsilon_{bmu}$).

- Die "Vergessens"-Parameter der modifizierten Hebb-Regeln zum Aufbau der optimal topologieerhaltenden Karten, die Abklingkonstante κ und der Schwellwert ξ zum Löschen von Kanten, sind funktional abhängig (ein freier Parameter).

[1]Experimente zur Parameterauswahl der Q-Lerner über genetische Algorithmen beispielsweise brachten eine im Verhältnis zum Zeitaufwand dieser Verfahren nur marginale Verbesserung der Lernresultate.

- Der Parameter zur distanzabhängigen Einfügung neuer Knoten sollte abhängig vom vorgegebenen Radius σ der Basisfunktionen gewählt werden (ein freier Parameter).
- Kann der mittlere Abstand zweier Zentren geschätzt werden, so sollte der Radius σ nach der Othonormalitäts- und Interpolationsbedingung (Seite 3.5) ermittelt werden.

Auf ein "Einfrieren" von Lernraten wurde generell zugunsten permanenter Plastizität verzichtet. Lediglich die Hand-Auge-Koordination (Tabellen 8.7,8.8, 8.9 u. 8.10) bildet zum Zwecke der Vergleichbarkeit mit den in [109, 131] präsentierten Resultaten eine Ausnahme. Parameter $p(t)$ weisen hier die Zeitabhängigkeit $p(t) = p_s(p_e/p_s)^{t/t_{max}}$ auf und werden durch Angabe von p_s und p_e sowie t_{max} spezifiziert. Hierbei ist t die Anzahl der erfolgten Lernschritte.

Adaption der Zentren nach Fritzke-Lernregel	
(3.29)	$\epsilon_{bmu} = 0.1$, $\epsilon_{Nh} = 0.006$
Adaption der Topologie nach einfacher Hebb-Regel mit Abklingkonstante	
(3.40)	$\kappa = 0.9$, $\xi = \sqrt[\lvert T\rvert]{\kappa}$
Adaption der Fehlervariablen mit Online-Fehlerakkumulation	
(3.46)	$\lambda = \lvert T\rvert$

Tabelle 8.1: Parameter: Unüberwachtes Training auf T_Q und T_S

Adaption der Ausgaben nach LMS-Regel	
(3.15)	$\alpha = 0.3$
Adaption der Zentren nach Fritzke-Lernregel	
(3.29)	$\epsilon_{bmu} = 0.2$, $\epsilon_{Nh} = 0.012$
Adaption der Topologie nach einfacher Hebb-Regel mit Abklingkonstante	
(3.40)	$\kappa = 0.9$, $\xi = \sqrt[\lvert T\rvert]{\kappa}$
Adaption der Fehlervariablen mit Online-Fehlerakkumulation	
(3.46)	$\lambda = \lvert T\rvert$
Basisfunktion: $h(z) = 1/(\frac{z^2}{\sigma^2} + 1)$ mit $\sigma = 0.7$	

Tabelle 8.2: Parameter: Überwachtes 2-Spiralen-Problem

Adaption der Ausgaben nach LMS-Regel	
(3.15)	$\alpha = 0.075$
Adaption der Zentren nach Fritzke-Lernregel	
(3.29)	$\epsilon_{bmu} = 0.05$, $\epsilon_{Nh} = 0.006$
Adaption der Topologie nach einfacher Hebb-Regel mit Abklingkonstante	
(3.40)	$\kappa = 0.9$, $\xi = \sqrt[\lvert T \rvert]{\kappa}$
Adaption der Fehlervariablen mit Online-Fehlerakkumulation	
(3.46)	$\lambda = \lvert T \rvert$
Basisfunktion: $h(z) = 1/(\frac{z^2}{\sigma^2} + 1)$ mit $\sigma = 0.7$	

Tabelle 8.3: Parameter: Sprecherunabhängige Erkennung von Vokalen

Adaption der Ausgaben nach LMS-Regel	
(3.15)	$\alpha = 0.3$
Adaption der Zentren nach Fritzke-Lernregel	
(3.29)	$\epsilon_{bmu} = 0.2$, $\epsilon_{Nh} = 0.006$
Adaption der Topologie nach einfacher Hebb-Regel mit Abklingkonstante	
(3.40)	$\kappa = 0.9$, $\xi = \sqrt[\lvert T \rvert]{\kappa}$
Adaption der Fehlervariablen mit Online-Fehlerakkumulation	
(3.46)	$\lambda = \lvert T \rvert$
Basisfunktion: $h(z) = 1/(\frac{z^2}{\sigma^2} + 1)$ mit $\sigma = 1.41$	

Tabelle 8.4: Parameter: Diskriminierung von Minen und Felsen

Adaption der Ausgaben nach LMS-Regel	
(3.15)	$\alpha = 0.001$
Adaption der Zentren nach Fehlermodulierter Lernregel	
(3.35)	$\epsilon_{bmu} = 0.1$, $\epsilon_{Nh} = 0.001$
Adaption der Topologie nach unsymmetrischer Hebb-Regel	
(3.41)	$\kappa = 0.88$, $\xi = 0.01$
Adaption der Fehlervariablen nach gleitender Mittelwertbildung	
(3.47)	$\beta = 0.2$
Adaption der Modellordnung: distanz- und fehlerabhängig	
(3.52)	$\delta_{dist} = 0.08$
(Seite 57)	$\lambda_{ins} = 70$
Basisfunktion: $h(z) = \exp(-\frac{z^2}{\sigma^2})$ mit $\sigma = 0.2$	

Tabelle 8.5: Parameter: Online-Stabbalancieren

Verstärkungsfaktoren (monokular u. binokular)	
(5.19)	$k_1 = 0.001$, $k_2 = 0.0012$
(5.20)	$k_1 = 0.0007$, $k_2 = 0.001$, $k_3 = 0.0006$
Adaption der Ausgaben nach LMS-Regel	
(5.24)	$\alpha = 0.7$
Adaption der Zentren nach drei alternativen Lernregeln (Fritzke-Regel, fehlermodulierte Regel o. restringierter Gradientenabstieg)	
(3.29), (3.35) o. (5.25)	$\epsilon_{bmu} = 0.05$, $\epsilon_{Nh} = 0.006$
Adaption der Topologie nach modifizierter Hebb-Regel	
(Seite 55)	$\kappa = 0.1$, $\xi = 0.33$
Adaption der Fehlervariablen nach gleitender Mittelwertbildung	
(3.47)	$\beta = 0.03$
Adaption der Modellordnung: präzisionsgesteuert	
(3.48)	$\delta_{err} \in \{3.5, 2.9, 2.5\}$
Basisfunktion: $h(z) = 1/(\frac{z^2}{\sigma^2} + 1)$ mit $\sigma = 1.0$	

Tabelle 8.6: Parameter: Kalibrierungsfreie Sakkadensteuerung

Adaption der Ausgaben	
(2.16)	$\alpha(t) = \alpha = 0.9$, $\sigma'_s = 2.5$, $\sigma'_e = 0.5$
Adaption der Zentren	
(2.12)	$\epsilon_s = 1.0$, $\epsilon_e = 0.01$, $\sigma_s = 2.5$, $\sigma_e = 0.01$

Tabelle 8.7: Parameter: Erweiterte Kohonenkarte mit $7 \times 7 \times 7$ Knoten, $t_{max} = 4000$

Adaption der Ausgaben	
(2.23)	$\alpha(t) = \alpha = 0.9$, $\sigma'_s = 50$, $\sigma'_e = 1$
Adaption der Zentren	
(2.19)	$\epsilon_s = 0.3$, $\epsilon_e = 0.05$, $\sigma_s = 150$, $\sigma_e = 1$

Tabelle 8.8: Parameter: Neuronengas mit 300 Knoten, $t_{max} = 4000$

Adaption der Ausgaben nach LMS-Regel	
(3.15)	$\alpha = 0.9$
Adaption der Zentren nach Fritzke-Lernregel	
(3.29)	$\epsilon^s_{bmu} = 1$, $\epsilon^e_{bmu} = 0.05$, $\epsilon^s_{Nh} = 1$, $\epsilon^e_{Nh} = 0.01$
Adaption der Topologie nach modifizierter Hebb-Regel	
(Seite 55)	$\kappa = 0.1$, $\xi = 0.33$
Basisfunktion: $h(z) = 1/(\frac{z^2}{\sigma^2} + 1)$ mit $\sigma = 0.06$	

Tabelle 8.9: Parameter: DCS mit 300 Knoten, $t_{max} = 4000$

Adaption der Ausgaben nach LMS-Regel	
(3.15)	$\alpha = 0.9$
Adaption der Zentren nach Fritzke-Lernregel	
(3.29)	$\epsilon^s_{bmu} = 1$, $\epsilon^e_{bmu} = 0.05$, $\epsilon^s_{Nh} = 1$, $\epsilon^e_{Nh} = 0.01$
Adaption der Topologie nach modifizierter Hebb-Regel	
(Seite 55)	$\kappa = 0.1$, $\xi = 0.33$
Basisfunktion: $h(z) = 1/(\frac{z^2}{\sigma^2} + 1)$ mit $\sigma = 0.31$	

Tabelle 8.10: Parameter: DCS mit 30 Knoten (reales Experiment), $t_{max} = 1500$

Q-Lernregel	
(6.7)	$\gamma = 0.9$, $\alpha = 0.075$

Tabelle 8.11: Parameter: Q-Lernen mit optimistischer Initialisierung und gieriger Politik

MQ-Lernregel	
(6.34) u. (6.35)	$\gamma = 0.9$, $\alpha = 0.5$

Tabelle 8.12: Parameter: MQ-Lernen

Adaption der Ausgaben	
(6.41) und (6.42)	$\gamma = 0.9$, $\alpha = 0.5$
Adaption der Zentren nach Gradientenregel	
(6.46)	$\epsilon = 1.0$
Adaption der Topologie nach einfacher Hebb-Regel mit Abklingkonstante	
(3.40)	$\kappa = 0.95$, $\xi = 0.05$
Adaption der Modellordnung: distanzabhängig	
(3.52)	$\delta_{dist} = 0.2$
Basisfunktion: $h(z) = 1/(\frac{z^2}{\sigma^2} + 1)$ mit $\sigma = 0.07$	

Tabelle 8.13: Parameter: QDCS-Netz mit Aktivitätsspur

Adaption der Ausgabeschicht gem. AHCR	
(6.68),(6.70),(6.72)	$\gamma = 0.99$ $\alpha_V = 0.005$, $\alpha_\mu = 0.31$, $\alpha_\sigma = 0.061$
Adaption der Zentren nach Fehlermodulierter Lernregel	
(3.35)	$\epsilon_{bmu} = 0.0001$, $\epsilon_{Nh} = 0.00001$
Adaption der Topologie nach unsymmetrischer Hebb-Regel	
(3.41)	$\kappa = 0.99$, $\xi = 0.3$
Adaption der Fehlervariablen nach gleitender Mittelwertbildung	
(3.47)	$\beta = 0.1$
Adaption der Modellordnung: distanz- u. fehlerabhängig, $N_{max} = 100$	
(3.52)	$\delta_{dist} = 0.4$
(Seite 57)	$\lambda_{ins} = 5000$
Basisfunktion: $h(z) = \exp(-\frac{z^2}{\sigma^2})$ mit $\sigma = 0.2$	

Tabelle 8.14: Parameter: Lernende Hindernisvermeidung

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

I	Eingaberaum	13
A	Ausgaberaum	13
Err^2	Quadratische Fehlerfunktion	14
Err^{VQ}	Vektorquantisierungsfehler	15
$<>, E$	Erwartungswert	16
ζ	Auflösungsfaktor	16

DCS

o_i	Ausgabevektor des i.ten Knotens	33
σ_i	Radius der i.ten Basisfunktion	33
c_i	Zentrumsvektor des i.ten Knotens	33
r_i	Fehlervariable des i.ten Knotens	47
$h(.)$	Aktivierungsfunktion	34
$\mathcal{A}_P(.)$	Aktive Knotenmenge	34
$Nh^{(i)}(.)$	Nachbarschaftsfunktion (bis Tiefe i)	35
α	Lernparameter für Ausgabeschicht	46
β	Adaptionsrate der Fehlervariablen	58
ϵ	Lernparameter für Zentren	49
ϵ_{bmu}	Lernparameter für Zentrum der bmu	49
ϵ_{Nh}	Lernparameter für Zentren der Nachbarn der bmu	49
κ	Abklingkonstante für Kanten der OTPM	54
χ	Schwellwert für Kanten der OTPM	54
λ	# Lernschritte bei Online-Fehlerakkumulation	57
λ_{ins}	# Lernschritte zwischen Neueinfügungen	58
δ_{err}	Präzision (präzisionsgesteuerte Online-Wachstumsstrategie)	58
δ_{dist}	Schwellwert für distanzgetriebenes Wachstum	59

Sakkadensteuerung

χ	Schwenkwinkel	79
ϕ	Neigungswinkel	79
$\theta_{l/r}$	Linker bzw. rechter Vergenzwinkel	79

ω	Ausgabe (Stellwinkel) der Sakkadensteuerung	80
b	Stereobasis	79
p_w	Punkt im Weltkoordinatensystem	79
p_c	Punkt im Kamerakoordinatensystem	79
p_i	Punkt im Bildkoordinatensystem	80
$T_\chi, T_\phi, T_{\theta_l}, T_{\theta_r}$	Rotationsmatrizen	80
$\alpha_{xc}, \alpha_{yc}, x_{0c}, y_{0c}$	Intrinsische Kameraparameter	80
P_c	Matrix der intrinsischen Kameraparameter	80
A, A_i	Jacobi-Ausgabematrix des Netzes / des i.ten Knotens	83
A^*	Schätzer für A	85
ω	Gelenkwinkel	83
$\Delta\omega$	Ausgabe der Sakkadensteuerung (Gelenkwinkeldifferenz)	80
$\Delta\omega^p$	Ausgabe des Proportionalreglers (Gelenkwinkeldifferenz)	84
$\Delta\omega^*$	Schätzer für $\Delta\omega$	85
u	Eingabevektor in DCS-Netz	83

Hand-Auge-Koordination

θ, θ_i	Ausgabegelenkwinkel des Netzes / des i.ten Knotens	94
A, A_i	Ausgabematrix des Netzes / des i.ten Knotens	94
θ^i	Gelenkwinkel des Roboterarms nach i.tem Korrekturschritt	94
p	Bildkoordinaten des Zielpunktes	93
p^i	Bildkoordinaten des Endeffektors in Stellung θ^i	95
$\theta^\star, A^\star$	Verbesserte Schätzer für θ, $A^\star$	97

Reinforcement-Lernen

$V^\pi(.)$	Ertragsfunktion unter Politik π	105
$V^{opt}(.)$	Optimalwertfunktion	108
$Q^\pi(., a)$	Ertragsfunktion unter Politik π bei Ausführung von Aktion a	108
$Q^{opt}(., a)$	Optimalwertfunktion bei Ausführung von Aktion a	109
$Q_t(., .)$	Q-Schätzer zur Zeit t	110
$V_t(.)$	V-Schätzer zur Zeit t	110
$q_t(., .)$	Q-Schätzer zur Zeit t beim AHC-Lernen	113
	(auch Ausgabegewichte beim Q-DCS Netz)	124
$err_{TD}(.)$	TD-Fehler	113
P_t	Prädiktion im Zustand s_t	114
λ	Abschwächungsfaktor des TD(λ)-Lernens	115
γ	Diskontierungsrate	105
μ, μ^i	prototypische Stellgrößen des Netzes / des i.ten Knotens	137
d, d^i	Suchbereichsvektor des Netzes / des i.ten Knotens	137
V, V^i	Prädiktion des Netzes / des i.ten Knotens	137

Literaturverzeichnis

[1] E. Abraham. Neuronale Netze zur Visuomotorischen Regelung. Studienarbeit, Inst. f. Inf. u. Prakt. Math., CAU zu Kiel, 1996.

[2] I. Ahrns. Ultraschallbasierte Navigation und adaptive Hindernisvermeidung eines autonomen mobilen Roboters. Diplomarbeit, Inst. f. Inf. u. Prakt. Math., CAU zu Kiel, 1996.

[3] I. Ahrns, J. Bruske und G. Sommer. On-line learning with dynamic cell stuctures. In *ICANN'95*, Seiten 141–146, 1995.

[4] H. Akaike. Statistical predictor identification. *Annals of the Institute of Statistical Mathematics*, 22:203, 1970.

[5] Ethem Alpaydin. GAL: Networks that grow when they learn and shrink when they forget. Technical report, FTP: 128.32.201.55 — pub/techreports — tr-91-032.ps. Z (binary mode) Technical Report 91-032, 1991.

[6] C.W. Anderson. Q-learning with hidden units restarting. In *Advances in Neural Information Processing Systems*, Seiten 81–87, 1992.

[7] A. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inf. Theory*, 39(3):930–945, 1993.

[8] A. Barto und P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Trans. on Systems, Man and Cybernetics*, 15:360–374, 1985.

[9] A. Barto, R. Sutton und C. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man and Cybernetics*, 13:834–846, 1983.

[10] Andrew G. Barto und Steven J. Bradtke. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[11] H. Bauer, R. Der und M. Herrmann. Controlling the magnification factor of self-organizing feature maps. *Neural Computation*, 8:757–771, 1996.

[12] M. Benaim. On functional approximation with normalized gaussian units. *Neural Computation*, 6:319–333, 1994.

[13] H. R. Berenji und Pratap Khedar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks*, 3(5):724–740, 1992.

[14] D. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice Hall, 1987.

[15] C. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, 1995.

[16] C. M. Bishop. Regularization and complexity control in feed-forward networks. In *ICANN 95*, volume 1, Seiten 141–148, 1995.

[17] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation*, 2(1):14–23, 1986.

[18] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[19] J. Bruske. Robotersteuerung mit RBF Netzen. In B. Krieg-Brueckner und Ch. Herwig, editor, *Tagungsband des Workshops "Kognitive Robotik".* Zentrum fuer Kognitionswissenschaften, Universitaet Bremen, 1995.

[20] J. Bruske, E. Abraham-Mumm und G. Sommer. Visuomotorische Koordination eines Roboterarmes mit Kohonen-Karten, Neuronalem Gas und Dynamischen Zellstrukturen. In *Workshop SOAVE'97, Ilmenau*, VDI Fortschrittberichte, Reihe 8, Nr. 663, Seiten 203–211, 1997.

[21] J. Bruske, I. Ahrns und G. Sommer. An integrated architecture for learning of reactive behaviors based on dynamic cell structures. Technical Report 9604, Inst. f. Inf. u. Prakt. Math. Christian-Albrechts-Universitaet zu Kiel, 1996.

[22] J. Bruske, I. Ahrns und G. Sommer. Neural fuzzy control based on reinforcement learning and dynamic cell structures. In *Proc. of EUFIT'96*, Seiten 710–714, 1996.

[23] J. Bruske, I. Ahrns und G. Sommer. Practicing q-learning. In *Proceedings of ESANN'96, Brugges*, Seiten 25–30, 1996.

[24] J. Bruske, I. Ahrns und G. Sommer. An integrated architecture for learning of reactive behaviors based on dynamic cell structures. *Robotics and Autonomous Systems*, 22(2):87–101, 1998.

[25] J. Bruske, M. Hansen, L. Riehn und G. Sommer. Adaptive saccade control of a binocular head with dynamic cell structures. In *Proc. of the International Conference on Artificial Neural Networks*, Seiten 215–220, 1996.

[26] J. Bruske, M. Hansen, L. Riehn und G. Sommer. Biologically inspired calibration-free adaptive saccade control of a binocular camera-head. *Biological Cybernetics*, 77(6):433–446, 1997.

[27] J. Bruske, L. Riehn, M. Hansen und G. Sommer. Dynamic cell structures for calibration-free adaptive saccade control of a four-degrees-of-freedom binocular head. Technical Report 9608, Inst. f. Inf. u. Prakt. Math. Christian-Albrechts-Universitaet zu Kiel, 1996.

[28] J. Bruske und G. Sommer. Dynamic cell structures: Radial basis function networks with perfect topology preservation. Technical Report 9403, Inst. f. Inf. u. Prakt. Math CAU, 1994.

[29] J. Bruske und G. Sommer. Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7(4):845–865, 1995.

[30] J. Bruske und G. Sommer. Dynamic cell structures. In *Advances in Neural Information Processing Systems, NIPS 94*, Seiten 497–504, 1995.

[31] R. Carpenter. *Movement of the eyes.* Pion London, 1988.

[32] P. Dean, J. Mayhew und P.M. Langdon. Learning and maintaining saccadic accuracy: A model of brainstem-cerebellar interactions. *Journal of Cognitive Neuroscience*, 6(2):117–138, 1994.

[33] Bruce L. Digney. *Emergent Intelligence in a Distributed Adaptive Control System.* PhD thesis, Intelligent Systems Research Laboratory, University of Saskatchewan, 1994.

[34] G. Dorffner. Kuenstliche Neuronale Netze: Bestandsaufnahme, K. Moeller und G. Pass (Hrsg.). *KI*, (4):37–61, 1994.

[35] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical report, Carnegie Mellon University, 1988.

[36] I. Farkas und L. Chudy. Application of a growing self-organizing map to thinning of binary characters with noise. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6*, Seiten 215–219. Helsinki University of Technology, Neural Networks Research Centre, 1997.

[37] B. Fritzke. Growing cell structures - a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1995.

[38] Bernd Fritzke. A growing neural gas network learns topologies. In *NIPS 7*, Seiten 625–632, 1995.

[39] S. Geman, E. Bienenstock und R. Doursat. Neural networks and the bias/variance dilema. *Neural Computation*, 4(1):1–58, 1992.

[40] F. Girosi und G. Anzellotti. Convergence rates of approximation by translates. Technical Report A.I. Memo No. 1288, MIT AI lab, 1995.

[41] F. Girosi und T. Poggio. Networks and the best approximation property. Technical Report A.I. Memo No. 1164, MIT AI lab, 1989.

[42] R.P.. Gorman und T.J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[43] D. Graham. A fast reinforcement learning paradigm with application to cmac control systems. In *Proc. of the International Conference on Artificial Neural Networks*, Seiten 671–674, 1994.

[44] V. Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3:671–692, 1990.

[45] V. Gullapalli und A.G. Barto. Convergence of indirect adaptive asynchronous value iteration algorithms. In *Advances in Neural Information Processing Systems*, Seiten 695–703, 1993.

[46] J. Hakala. *Bedarfsangepasste neuronale Netze mit topographischer Kodierung zur Bestimmung der inversen Kinematik von Gelenkarmrobotern.* PhD thesis, Mat. Nat. Fak. der Friedrich-Wilhelms-Univ. zu Bonn, 1994.

[47] J. Hakala und R. Eckmiller. Partition of unity rbf networks are universal function approximators. In *Proc. of the International Conference on Artificial Neural Networks*, Seiten 459–462, 1994.

[48] J. Hakala und H.W. Werntege. Node allocation and topographical encoding (nate) for inverse kinematics of a redundant robot arm - learning without a teacher. In *Proc. of the International Conference on Artificial Neural Networks*, Seiten 615–618, 1992.

[49] F. Hamker und H.-M.Gross. Task-based representation in lifelong learning incremental neural networks. In *Workshop SOAVE'97 - Selbstorganisation von adaptivem Verhalten, Ilmenau*, VDI Fortschrittberichte, Reihe 8, Seiten 99–108, 1997.

[50] E. Hartman, J. Keeler und J. Kowalski. Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, 2(2):210–215, 1990.

[51] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, 1990.

[52] Matthias Heger. Risk and reinforcement learning: Concepts and dynamic programming. Technical Report ZKW Bericht Nr. 8/94, Uni Bremen, 1992.

[53] Matthias Heger und Karsten Berns. Risikoloses Reinforcement-Lernen. *KI*, (4):26–32, 1992.

[54] D. Henson. Corrective saccades: effects of altering visual feedback. *Vision Research*, Seiten 63–67, 1978.

[55] R. Herpers, H. Kattner und G. Sommer. GAZE: Eine attentive Verarbeitungsstrategie zum Erkennen und Lernen der visuell auffälligen Gesichtsregionen . In *Proc. Mustererkennung (DAGM)*, Seiten 341–349. Springer Verlag, 1994.

[56] R. Herpers, L. Witta, M.Michaelis, J. Bruske und G. Sommer. Detektion und Verifikation von charakteristischen Bildpunkten in Gesichtsbildern. In *Proc. of DAGM'96*, Seiten 435–446. Springer Verlag, 1996.

[57] R. Herpers, L. Witta, M.Michaelis, J. Bruske und G. Sommer. Dynamic cell structures for the evaluation of keypoints in facial images. *International Journal of Neural Systems*, 8(1):27–39, 1997.

[58] M. Herrmann und R. Der. Efficient q-learning by division of labor. In *ICANN 95*, volume 2, Seiten 129–134, 1995.

[59] Tohru Setoyama Hiroyuki Miyamoto, Mitsuo Kawato. Feedback-error-learning neural network for trajectory control of robotic manipulator. *Neural Networks*, Seiten 251–256, 1988.

[60] Juergen Hollatz. *Integration von regelbasiertem Wissen in neuronale Netze.* PhD thesis, Inst. f. Inf. TU Muenchen, 1993.

[61] W. Ilg und K. Berns. A learning architecture based on reinforcement learning for adaptive control of the walking machine lauron. *Robotics and Autonomous Systems*, 15:321–324, 1995.

[62] T. Jaakkola, M.I.Jordan und S.P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6), 1994.

[63] M.I. Jordan und D.E. Rumelhart. Forward models: Supervised learning with a distal teacher. MIT Center for Cognitive Science, Occasional Paper No. 40, 1992.

[64] R. Kashyap, C. Blaydon und K. Fu. *Adaptive, Learning, and Pattern Recognition Systems: Theory and Application*, chapter Stochastic Approximation, Seiten 329–356. Academic Press, 1970.

[65] M. Kawato. Computational schemes and neural network models for formation and control of multijoint arm trajectory. In *Neural Networks for Control*, Seiten 197–228. MIT Press, 1990.

[66] M. Kawato. Feedback-error-learning neural network for supervised motor learning. In R. Eckmiller, editor, *Advanced Neural Computers*, Seiten 365–372. Elsevier Science Publishers, 1990.

[67] M. Kawato. *The Handbook of Brain Theory and Neural Networks*, chapter Cerebellum and Motor Control, Seiten 172–178. MIT Press, 1995.

[68] M. Kawato, K. Furukawa und R. Suzuki. A hierachical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185, 1987.

[69] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[70] T. Kohonen. *Self-Organizing Maps.* Springer, 1995.

[71] Teuvo Kohonen. The self organizing map. *Proc. of the IEEE*, 78(9), September 1990.

[72] D. Kontoravdis. Efficient reinforcement learning strategies for the pole balancing problem. In *Proc. of the International Conference on Artificial Neural Networks*, Seiten 659–662, 1994.

[73] T. Landelius. *Reinforcement Learning and Distributed Local Model Synthesis.* PhD thesis, Department of Electrical Engineering, Univ. of Linkoeping, 1997.

[74] K. Lang und M. Witbrock. Learning to tell two spirals apart. In *Proc. of the 1988 Connectionist Models Summer School*, Seiten 52–59. Morgan Kaufmann, 1988.

[75] Long-Ji Lin. Hierarchical learning of robot skills by reinforcement. In *IEEE International Conference on Neural Networks 1993*, 1993.

[76] Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks.* PhD thesis, Carnegie Mellon University, 1993.

[77] Y. Linde, A. Buzo und R. Gray. An algorithm for vector quantizer design. *IEEE Transaction on Communications*, 28(1):84–95, 1980.

[78] D. Lowe. *Handbook of Brain Theory and Neural Networks*, chapter Radial Basis Function Networks, Seiten 779–782. MIT Press, 1995.

[79] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, Seiten 281–297. Berkeley: University of California Press, 1967.

[80] T. Martinetz. *Selbstorganisierende Neuronale Netzwerkmodelle zur Bewegungssteuerung.* PhD thesis, Physik-Department TU Muenchen, 1992.

[81] T. Martinetz und K. Schulten. Topology representing networks. In *Neural Networks*, volume 7, Seiten 505–522, 1994.

[82] J. Mayhew und P. Dean. Saccade control in a simulated robot camera-head system: neural net architectures for efficient learning of inverse kinematics. *Biol. Cybern.*, 66:27–36, 1991.

[83] J. Mayhew, Y. Zheng und S. Cornell. The adaptive control of a four-degrees-of-freedom stereo camera head. *Phil. Trans. R. Soc. Lond.*, 337:315–326, 1992.

[84] M. Michaelis. *Low level image processing using steerable filters.* PhD thesis, Christian-Albrechts-Universität, Kiel, Germany, 1995.

[85] C. Michelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximations*, 2:11–22, 1986.

[86] D. Michie und R.A. Chambers. Boxes: An experiment in adaptive control. In E. Dale und D. Michie, editors, *Machine Intelligence 2*, Seiten 137–152, 1968.

[87] J. R. Millan. Reinforcement learning of goal-directed obstacle avoiding reaction strategies in an autonomous mobile robot. *Robotics and Autonomous Systems*, 5:275–299, 1995.

[88] J. R. Millan und C. Torras. A reinforcement connectionist approach to robot pathe finding in non-maze-like environments. *Machine Learning*, 8:363–395, 1992.

[89] W.T. Miller, R.S. Sutton und P.J. Werbos, editors. *Neural Networks for Control*. MIT Press, 1990.

[90] J. Moody. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1988.

[91] J. Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In *Advances in Neural Information Processing Systems*, volume 4, Seiten 847–854, 1992.

[92] H. Murase und S. Nayar. Visual learning and recognition of 3-d objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.

[93] K. Narendra und M. Thathatchar. *Learning Automata: An introduction*. Prentice Hall, 1989.

[94] D. Nauck, F. Klawonn und R. Kruse. *Neuronale Netze und Fuzzy-Systeme*. Vieweg, 1994.

[95] J. Park und I. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3:246–257, 1991.

[96] F. Peinemann. Neuronale identifikation und regelung dynamischer systeme. Technical report, Zentrum fuer Kognitionswissenschaften Uni Bremen, 1994.

[97] Jing Peng. *Efficient Dynamic Programming-based Learning for Control*. PhD thesis, College of Computer Science of Northeastern University, 1993.

[98] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[99] T. Poggio und F. Girosi. Networks for approximation and learning. *Proc. of the IEEE*, 78(9):1481–1497, 1990.

[100] L. Prechelt. Proben1 - a set of neural network benchmark problems and benchmarking rules. Technical report, Fakultaet fuer Informatik, Uni Karlsruhe, 1994.

[101] Franco P. Preparata und Michael Ian Shamos. *Computational Geometry - An Introduction*. Springer-Verlag, 1985.

[102] T. Prescott und J. Mayhew. Obstacle avoidance through reinforcement learning. In *Advances in Neural Information Processing Systems*, Seiten 523–530, 1992.

[103] W.H. Press, S.A. Teukolsky, W.T. Vetterling und B.P. Flannery. *Numerical Recipes in C - The Art of Scientific Computing.* Cambridge University Press, 1988.

[104] C. Rao und S. Mitra. *Generalized Inverse of Matrices and Its Applications.* John Wiley, 1971.

[105] P. Reignier. Supervised incremental learning of fuzzy rules. *Robotics and Autonomous Systems*, 16:57–71, 1995.

[106] Patrick Reignier. Fuzzy logic techniques for mobile robot obstacle avoidance. *Robotics and Autonomous Systems*, 12:143–153, 1994.

[107] L. Riehn. Vergleich neuronaler und konventioneller Sakkadensteuerungen unter Realweltbedingungen. Diplomarbeit, Inst. f. Inf. u. Prakt. Math., CAU zu Kiel, 1996.

[108] H. Ritter. Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements. In R. Eckmiller und C.v.d. Malsburg, editors, *Neural Computers*, Seiten 395–406. Springer Verlag, 1987.

[109] H. Ritter, T. Martinetz und K. Schulten. *Neuronale Netze.* Addison-Wesley, 1991.

[110] H. Ritter und K. Schulten. On the stationary state of Kohonen's selforganizing sensory mapping. *Biological Cybernetics*, 54:99–106, 1986.

[111] H. Robbins und S. Monro. A stochastic approximation model. *Ann. Math. Stat.*, 22:400–407, 1951.

[112] A.J. Robinson. *Dynamic Error Propagation Networks.* PhD thesis, Cambridge Univ. Engeneering Department, 1989.

[113] S. Ross. *Introduction to Stochastic Dynamic Programming.* Academic Press, NY, 1983.

[114] D. Rumelhart, D. Hinton und R. Williams. Learning internal representations by error propagation. In D. Rumelhart und J. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT press, 1986.

[115] G. Rummery und M. Niranjan. On-line q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engeneering Dept., 94.

[116] C. Scheier. Incremental category learning in a real world artifact using growing dynamic cell structures. In *Proc. of ESANN'96*, Seiten 117–122, 1996.

[117] D. Scott. *Multivariate Density Estimation: Theory, Practice, And Visualization.* John Wiley, 1992.

[118] G. Sommer. Verhaltensbasierter Entwurf technischer visueller Systeme. *KI*, (3):42–45, 1995.

[119] G. Sommer. Algebraic aspects of designing behavior based systems. In G. Sommer und J. Koenderink, editors, *Algebraic Frames for the Perception-Action Cycle*, Lecture Notes in Computer Science 1315, Seiten 1–28. Springer Verlag, 1997.

[120] R. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, Dept. of Computer and Information Science, Univ. of Massachusetts, Amherst, MA, 1984.

[121] R. S. Sutton, editor. *First International Conference on Simulation of Adaptive Behavior*, chapter Reinforcement Learning Architectures for Animats, Seiten 288–296. The MIT Press, 1990.

[122] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[123] S. Thrun. An approach to learning mobile robot navigation. *Robotics and Autonomous Systems*, special issue on Robot Learning, 1995.

[124] S. Thrun. Lifelong robot learning. *Robotics and Autonomous Systems*, 15:25–46, 1995.

[125] S.B. Thrun und A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proc. of the Fourth Connectionist Models Summer School*, 1993.

[126] Transitions Research Corporation (TRC), Danburry, Connecticut. *LABMATE User Manual*, Release 5.21L-f edition, 1991.

[127] P. van der Smagt. Simderella: A robot simulator for neuro-controller design. *Neurocomputing*, 6:281–285, 1994.

[128] T. Villmann, R. Der, M. Herrmann und T. Martinetz. Topology preservation in self-organizing feature maps: Exact definition and measurement. *IEEE Transactions on Neural Networks*, 8(2):256–265, 1997.

[129] T. Villmann, R. Der und T. Martinetz. A novel approach to measure the topology preservation of feature maps. *ICANN*, Seiten 289–301, 1994.

[130] J. Walter. *Rapid Learning in Robotics*. PhD thesis, Technische Fakultät, Universität Bielefeld, 1996.

[131] J. Walter und K. Schulten. Implementation of self-organizing neural networks for visuo-motor control of an industrial robot. *IEEE Transactions on Neural Networks*, 4(1):86–95, January 1993.

[132] L. Wang und J. Mendel. Fuzzy basis functions, universal approximation, orthogonal least-squares learning. *IEEE Transactions on Neural Networks*, Seiten 807–814, 1992.

[133] C. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, UK, 1989.

[134] C. Watkins. Q-learning. *Machine Learning*, 8:279–292, 1992.

[135] A. Webb. Functional approximation by feed-forward networks: a least-squares approach to generalization. *IEEE Transactions on Neural Networks*, 5(3):363–371, 1994.

[136] S. Whitehead und D. Ballard. Active perception and reinforcement learning. *Neural Computation*, 2:409–419, 1990.

[137] R.J. Williams. Simple statisticle gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[138] P. L. Zador. Asymptotic quantization error of continuous signals and the quantization dimension. *IEEE Transactions on Information Theory*, 28(2):139–149, 1982.

[139] A. Zell. *Simulation Neuronaler Netze.* Addison-Wesley, 1994.

Index