Inhaltsverzeichnis

Institut für Informatik
Technische Universität München

Dynamische neuronale Netze und das fundamentale raumzeitliche Lernproblem

Jürgen Schmidhuber

Vollständiger Abdruck der von der Fakultät für Mathematik und Informatik der
Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. B. Radig

Prüfer der Dissertation:

1. Univ.-Prof. Dr. W. Brauer
2. Prof. K. Schulten, Ph. D.

Die Dissertation wurde am 7. November 1990 bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik und Informatik am 20. Dezember 1990 angenommen.

Zusammenfassung

Neuronale Netze, obwohl bisher fast ausschließlich in stationären Umgebungen eingesetzt, öffnen neue Perspektiven für die Lösung von Minskys *'basic credit-assignment problem'* in zeitlich variierenden reaktiven Umgebungen.

Im ersten Kapitel dieser Arbeit wird die allgemeine Problemstellung beschrieben. Die beiden folgenden Kapitel geben eine Einführung zu existierenden Lernalgorithmen für neuronale Netze mit nicht-stationären Eingaben.

Im originären Teil der Arbeit werden in drei aufeinanderfolgenden Kapiteln drei verschiedene neuartige Klassen von Echtzeit-Lernalgorithmen für neuronale Netze begründet und experimentell getestet. Vertreter aller drei Algorithmenklassen erlauben im Gegensatz zu praktisch allen anderen Verfahren *interne Rückkopplung*: Aufgrund zyklischer Verbindungen besteht das Potential für ein Gedächtnis in Form zirkulierender Aktivationen. Auch *externe Rückkopplung* ist im Prinzip möglich: Ausgabeaktionen eines Netzwerkes können Einfluß auf spätere Eingaben nehmen. Alle drei Ansätze können auf einen gut informierten Lehrer verzichten, der schon weiß, welche Aktion zu welchem Zeitpunkt hätte stattfinden sollen.

Die erste Methode zeigt erstmals auf konstruktive Weise: Zielgerichtetes Lernen mit 'versteckten Neuronen' ist kompatibel mit der vollständigen Lokalität des Lernalgorithmus.

Die zweite Methode zeigt, wie das fundamentale raumzeitliche Lernproblem mittels zweier interagierender rekurrenter Netzwerke durch Gradientenabstieg in einem adaptiven 'Umgebungsmodell' in Echtzeit angegangen werden kann.

Die dritte Methode zeigt erstmals, wie Suttons *Methoden der zeitlichen Differenzen* und adaptive Kritiker auf rekurrente Netzwerke angewendet werden können. Weiterhin werden unter Rückgriff auf die zweite Methode erstmals *multidimensionale* adaptive Kritiker beschrieben.

Das folgende Kapitel führt am Beispiel 'attentive vision' erstmals vor, daß dynamische selektive Aufmerksamkeit *gelernt* werden kann. Gleichzeitig illustriert diese Anwendung, wie man aus einem scheinbar *statischen* Problem unter Effizienzgewinn ein *dynamisches* machen kann.

Das abschließende Kapitel kritisiert alle vorangehenden, weil sie nicht das Problem des kompositionellen hierarchischen Lernens ansprechen. Die Kritik ist konstruktiv: Erstmalig wird ein *adaptiver neuronaler Subzielgenerator* entwickelt.

Kapitel 1

Einführung

1.1 Allgemeine Einführung

Man stelle sich einen autonomen Roboter vor, dessen Bewegungen durch ein adaptives Steuersystem kontrolliert werden. Durch seine Bewegungen ändert der Roboter den Zustand der Umgebung, wodurch sich im allgemeinen auch seine sensorische Wahrnehmung der Umgebung ändert. Im Badezimmer befindet sich eine Steckdose. Gelingt es dem Roboter, seinen Stecker in die Steckdose zu stecken, so wird seine Batterie aufgefrischt. In der Abstellkammer befindet sich ein Ölkännchen, mit dessen Hilfe der Roboter seine Gelenke schmieren kann. Der Roboter kann nun verschiedenartige *'unangenehme'* Erfahrungen machen, indem er zum Beispiel mit einer seiner Extremitäten zu heftig gegen ein Hindernis stößt. Weitere unangenehme Erfahrungen bestehen darin, daß die Batterieladung unter einen bestimmten Schwellwert sinkt, oder daß die Gelenke anfangen zu rosten. Im allgemeinen können beliebige zeitliche Verzögerungen zwischen bestimmten Aktionssequenzen und ihren späteren Konsequenzen auftreten. Die Aufgabe des Roboters ist es, unangenehmen Situationen aus dem Weg zu gehen.

Der Roboter ist autonom insofern, als kein intelligenter Lehrer vonnöten ist, um ihn mit irgendwelchen weiteren Zielen oder Subzielen auszustatten. Das Problem des Roboters besteht darin, herauszufinden und zu lernen, wie er sich verhalten soll, um unliebsame Erlebnisse zu vermeiden. Gerät der Roboter in unerwünschte Situationen, so wird er mit Minskys fundamentalem Lernproblem (*'basic credit-assignment problem'* [28]) konfrontiert: *Was* hätte er *wann* anders machen müssen, um Mißerfolge zu vermeiden? Da die Umgebung in der Regel nicht pausiert, um dem Roboter irgendwelche beliebig aufwendigen Adaptionsprozesse zu gestatten, verschärft sich sein

Lernproblem noch: Er muß es in *Echtzeit* lösen.

Der Roboter in dem oben angegebenen (zugegebenermaßen reichlich utopischen) Beispiel steht für einen beliebigen zu steuernden Prozeß, etwa für einen Bestückungsautomaten in einer Fabrik, für eine Maschinerie, die eine chemische Reaktion im Gleichgewicht halten soll, oder für einen Theorembeweiser. Genauso kann man ihn als ein Modell eines adaptiven biologischen Organismus ansehen. Wichtig ist in allen Fällen, daß in der Regel *kein Lehrer* existiert, der schon im voraus weiß, wie die jeweiligen in Frage kommenden Prozesse zu steuern sind. Das adaptive System ist gezwungen, selbst Techniken für die Lösung seiner Aufgaben zu entwickeln. Das dabei auftretende Problem der *'Adaption in Echtzeit'* ist damit von einer sehr allgemeinen Natur, in komplexen Umgebungen kann es sich in beliebig komplizierter Form stellen. Im folgenden wird es des öfteren auch als *die allgemeine Problemstellung* referenziert werden. Diese Arbeit versucht, der allgemeinen Problemstellung durch einige neuartige Echtzeit-Lernalgorithmen für dynamische neuronale Netze zu begegnen.

Was haben neuronale Netzwerke (NN) überhaupt mit dem fundamentalen Lernproblem zu tun? Gegenwärtig beschränken sich noch nahezu alle Arbeiten zum Thema 'adaptive neuronale Netzwerke' auf das Erlernen statischer Musterassoziation. Die dabei auftauchenden, teilweise noch ungelösten Probleme sind im Vergleich zu der allgemeinen Aufgabe unseres autonomen Roboters scheinbar geradezu trivial. Es fragt sich: Ist die Zeit überhaupt schon reif für die Untersuchung des generellen Falls? Gibt es nicht genügend offene Fragen schon für viel einfachere Probleme? (Ein paar Beispiele für gern gestellte Fragen: Wie generalisiert ein gegebenes statisches Netzwerk von einer Auswahl *vorgegebener* Trainingsbeispiele auf ungesehene Muster aus derselben Verteilung? Welche Aussagen kann man über die Konvergenz des statischen Netzwerkes X machen?)

Die Antwort lautet: Das sind oft teilweise *irrelevante* Fragen, zumindest tritt ihre Bedeutung in den Hintergrund, wenn man aufhört, die bisher im Rahmen der NN-Forschung kaum beachtete *zeitliche Dimension* weiterhin zu ignorieren. (So verliert zum Beispiel die oben erwähnte Frage zur Generalisierung ihren Sinn, wenn das Netzwerk einen dynamischen Einfluß darauf nehmen kann, *was* es lernt.)

Diese Arbeit konzentriert sich auf den *zeitlichen* Aspekt neuronalen Lernens, ohne dabei den strukturellen Aspekt zu vernachlässigen. Es wird gezeigt werden, daß man die Untersuchungen zur allgemeinen Problemstellung schon heute sinnvoll anwenden kann: Untersuchungen für den generellen Fall erlauben Anwendungen, die manche mit dem statischen Fall assoziierten Probleme erst gar nicht aufkommen lassen (ein Beispiel liefert Kapitel 7 im Kontext *'attentive vision'*). Es wird gezeigt werden, daß die explizite Beachtung der essentiellen Dimension *Zeit* manche Probleme vereinfachen

kann.

Wir werden in dieser Einführung nicht die vollständige in dieser Arbeit gebräuchliche Terminologie vorstellen. Vielmehr wird in jedem Kapitel die lokal benötigte Notation dort eingeführt werden, wo sie gebraucht wird. Einige wesentliche Begriffe und Aufgabenstellungen werden allerdings immer wieder auftauchen, sie seien deshalb im folgenden in einer einführenden gegliederten Übersicht zusammengestellt.

1.2 Aufgabenstellungen und Terminologie

1.2.1 Die Art der verwendeten neuronalen Netze

Alle in den folgenden Kapiteln untersuchten neuronalen Netze bestehen aus zu biologischen Neuronen korrespondierenden *Knoten* und und aus zu biologischen Nervenfasern korrespondierenden gerichteten *Kanten*. Zu jedem Knoten gehört zu jedem Zeitpunkt eine der Feuerrate eines Neurons entsprechende reelle *Aktivation*. Zu jeder Kante gehört ein zu einer Synapsenstärke korrespondierendes *Gewicht*. Eingabeknoten werden zu jedem Zeitpunkt durch sensorische Wahrnehmung von der Umgebung aktiviert. Die typische Aktion eines Nicht-Eingabeknotens k besteht zu einem gegebenen Zeitpunkt in der Aufsummierung der gewichteten Aktivationen derjenigen Knoten, von denen k Verbindungen erhält. Das Resultat wird einer monoton wachsenden beschränkten Aktivierungsfunktion zur Berechnung von k's eigener Aktivation übergeben. Die Aktivationen mancher der Nicht-Eingabeknoten werden als Steuersignale für Effektoren zur Manipulation externer Prozesse interpretiert. *Lernregeln* sollen die Gewichte mit der Zeit dergestalt verändern, daß sich ein vorgegebenes Performanzmaß für das vom Gesamtsystem gezeigte Verhalten verbessert.

Die grundlegenden Bausteine der im folgenden vorgestellten NN sind also recht konventionell. *Nicht* berücksichtigt werden zum Beispiel Konzepte wie 'Knoten höherer Ordnung' oder 'dynamische Gewichte'. Der Grund dafür liegt darin, daß gegenwärtig nicht klar ersichtlich ist, ob derartige Konstrukte entscheidende Vorteile mit sich bringen.

Interne Rückkopplung, algorithmische Dynamik

Ist es möglich, in einem gegebenen Netzwerk wenigstens einen Knoten k zu finden, von dem aus ein Kantenpfad auf k selbst führt, so spricht man von einem *zyklischen* Netzwerk oder einem Netzwerk mit zyklischen Verbindungen. Besitzt ein Netzwerk zyklische Verbindungen, so vermögen zu einem gegebenen Zeitpunkt stattfindende Eingabeereignisse einen Einfluß darauf auszuüben, wie *spätere* Eingaben verarbeitet werden. Man hat also

das Potential für ein *Kurzzeitgedächtnis in Form wandernder Aktivationen* und spricht von interner Rückkopplung. Im allgemeinen können beliebige zeitliche Verzögerungen zwischen irgendwelchen Eingaben und ihren späteren Konsequenzen auftreten. Will man einem Netzwerk einen Algorithmus beibringen, der über simple Eingabe/Ausgabe-Assoziation hinausgeht, so ist *interne Rückkopplung* eine Möglichkeit für die Implementierung *algorithmischer* Dynamik. *Algorithmische Dynamik* unterscheidet sich dabei von *trivialer Dynamik* im wesentlichen dadurch, daß sie die *Geschichte* der Evolution der Zustände einer 'wahrnehmbaren' Umgebung zur Entscheidungsfindung mit berücksichtiget (siehe auch den untenstehenden Abschnitt über Markov-Prozesse). Bei *trivialer Dynamik* ist der wahrnehmbare Umgebungszustand zu einem gegebenen Zeitschritt nur vom wahrnehmbaren Zustand während des vorhergehenden Zeitschrittes abhängig.

Zwar sind neben der internen Rückkopplung auch noch andere Möglichkeiten zur Implementierung algorithmischer Dynamik denkbar, z.B. die von v. d. Malsburg vorgeschlagenen sich schnell ändernden Gewichte [70]. Da man aber neue Konzepte nicht einführen sollte, solange keine offensichtliche Notwendigkeit dazu besteht, beschränkt sich die vorliegende Arbeit auf das durch interne Rückkopplung eröffnete Potential für nicht-trivale Dynamik.

Externe Rückkopplung

Ein neuronales Netz, welches Eingaben von einer nicht-stationären Umgebung erhält, kann in der Lage sein, die Umgebung durch als Steuersignale für einen Roboter interpretierte Ausgaben zu manipulieren. Die auf diese Art mitverursachten neuen Zustände der Umgebung führen im allgemeinen zu neuen Eingaben. Diesen Effekt nennen wir *externe Rückkopplung*. In der Regel können beliebige zeitliche Verzögerungen zwischen irgendwelchen Ausgabeaktionen und ihren späteren Konsequenzen auftreten.

Neuronale Netze und das fundamentale Lernproblem

Gerne hätte man, daß ein Netzwerk lernt, irgendwelche extern gestellten Aufgaben unter Inanspruchnahme möglichst spärlicher Lehrinformation zu lösen. Wenn die Performanz ungenügend ist, drängt sich dem Netzwerk ganz automatisch Minskys fundamentales Lernproblem [28] auf: *Welche* Komponenten des Netzes trugen zu *welchen* Zeitpunkten *wie* zu unerwünschten Zuständen bei? Wie sollten die *kritischen* Komponenten ihr Verhalten ändern?

An dieser Fragestellung ist schon ersichtlich, daß das fundamentale Lernproblem eine *zeitliche* und eine *strukturelle* Komponente besitzt. Nahezu alle Arbeiten zu adaptiven neuronalen Netzen beschäftigten sich bisher

ausschließlich mit der strukturellen Komponente. Das mag teilweise daran liegen, daß man im Zuge der Betonung des massiv parallelen Aspektes neuronaler Netze in ein dem bisherigen sequentiellen Paradigma entgegengesetztes Extremum verfällt. Ein weiterer Grund mag darin liegen, daß das Gebiet bis vor kurzem von Physikern dominiert war, die ihre etablierten mathematischen Werkzeuge zur Behandlung von statischer Equilibriumsdynamik ohne größere Schwierigkeiten auf entsprechende neuronale Netze anwenden konnten. Was immer der Grund für die Vernachlässigung des zeitlich-sequentiellen Aspektes der neuronalen Informationsverabeitung war, gerechtfertigt war diese Vernachlässigung nicht: Selbst scheinbar statische Probleme (z.B. Bilderkennung - ganz zu schweigen von den inhärent dynamischen Problemen) erscheinen unter expliziter Beachtung der zeitlichen Dimension in einem ganz neuen Licht, oft gibt es Anlaß zu bedeutenden Effizienzgewinnen. (Ein Beispiel dafür liefert auch diese Arbeit, und zwar im Kapitel zur selektiven dynamischen Aufmerksamkeit.)

Ein Beispiel für interne und externe Rückkopplung ist das folgende: Ein Roboter versucht, einen Stab zu balancieren. Um seiner Aufgabe gerecht zu werden, muß der Roboter nicht nur die gegenwärtige Position des Stabes, sondern auch seine Geschwindigkeit feststellen. Um die Geschwindigkeit zu ermitteln, muß er (falls ihm kein Lehrer Hilfestellung gibt) im allgemeinen vergangene Positionen des Stabes mit berücksichtigen. Dazu muß er sich vergangene Ereignisse merken können. Eine wichtige Frage lautet: Wie kann er lernen, sich *die für seine Aufgabe relevanten* vergangenen Ereignisse zu merken und die irrelevanten zu ignorieren?

Statische und dynamische Lernalgorithmen und Lernaufgaben

Die Möglichkeit der internen Rückkopplung allein reicht natürlich nicht aus, um aus einem Netzwerk vernünftige algorithmische Dynamik herauszuholen. Man braucht auch einen Lernalgorithmus, der die Gewichte so adjustiert, daß *'die richtigen'* Ereignisse im Kurzzeitgedächtnis gespeichert werden. Es gibt zwar eine ganze Reihe von Lernalgorithmen für Netzwerke mit interner Rückkopplung. *Fast alle* dieser Algorithmen funktionieren jedoch nur dann, wenn das Netzwerk bei *stationären* Eingaben stets in einen Gleichgewichtszustand gerät. Fast alle dieser Algorithmen sind damit *statische* Algorithmen, trotz des Vorhandenseins interner Rückkopplung.

Es kommt also darauf an, *was* ein NN mit seiner internen Rückkopplung anfängt (vorausgesetzt, es hat überhaupt eine). Erst in jüngster Zeit beginnt sich die Aufmerksamkeit auf wirklich *dynamische* Lernalgorithmen zu richten.

Dynamische Algorithmen zeichnen sich dadurch aus, daß sie für Netzwerke tauglich sind, deren Ein- und Ausgaben sich kontinuierlich ändern.

Es ist das Ein-/Ausgabeverhalten eines Netzes und die Natur des Lernproblems, die uns einen Algorithmus *dynamisch* oder *statisch* nennen läßt. Es ist nicht die eventuell vorhandene interne Netzwerkdynamik (erzeugt durch etwaige interne Rückkopplung).

Markov-Prozesse

Schränkt man sich auf geeignete Umgebungen ein, so vermindert sich die Bürde des fundamentalen Lernproblems unter Umständen ganz erheblich. Die wenigen Ansätze zum Lernen in reaktiver Umgebung beschränkten sich bisher auf *Markov*-artige Umgebungen.

Ein sogenannter diskreter n-stufiger Prozeß ist ein Prozeß, bei dem der Zustand des Prozesses durch n aufeinanderfolgende 'Entscheidungen' (Aktionen) geändert wird. Ein diskreter n-stufiger Prozeß hat die *Markov-Eigenschaft*, wenn nach k Entscheidungen der Effekt der ausstehenden $n-k$ Entscheidungen auf eine den 'Nutzen' von Entscheidungsfolgen bestimmende Kostenfunktion nur noch von dem Zustand des Systems nach der k-ten Entscheidung und den folgenden Entscheidungen abhängt.

Viele deterministische Prozesse sind Markov-Prozesse. Beispiele sind nahezu alle Brettspiele: Es kommt nicht darauf an, wie man zu einem Spielzustand gekommen ist. Zu jedem Zeitpunkt ist alle Information, die man zum Weiterspielen braucht, in dem gegenwärtigen Zustand enthalten. Auch das oben erwähnte Balancierproblem kann zu einem *Markov-Problem* vereinfacht werden, wenn der Roboter zu jedem Zeitpunkt die zeitlichen Ableitungen der Stabposition als zusätzliche Eingabe bekommt.

Viele Prozesse sind jedoch *keine* Markov-Prozesse. Insbesondere für biologische Systeme typische Handlungsweisen hängen oft nicht nur vom gegenwärtig wahrnehmbaren Zustand der Umgebung ab, sondern auch von vergangenen Perzeptionen. Wir unterscheiden daher im folgenden zwischen *Markov-Umgebungen* und *Nicht-Markov-Umgebungen* und beziehen uns dabei auf die Natur der Schnittstelle zwischen Umgebung und Lernsystem: In einer Markov-Umgebung reicht stets die letzte Eingabe zur Voraussage der nächsten Eingabe aus. In Nicht-Markov-Umgebungen sind unter Umständen beliebig weit zurückliegende vergangene Ereignisse mit zu berücksichtigen.

1.2.2 Zielgerichtetes Lernen

Zielgerichtetes Lernen wird zweckmäßigerweise unterteilt in *überwachtes* Lernen und Lernen, welches auf sogenannten *Reinforcement*signalen beruht. Unglücklicherweise gibt es keine befriedigende Übersetzung des Begriffes

'Reinforcement'. Im folgenden werden wir gegebenenfalls vom *'R-Lernen'* sprechen.

Überwachtes Lernen

Ein Lernproblem ist ein *überwachtes* Lernproblem, wenn zu gewissen Zeitpunkten einer Aktivationsausbreitungsphase eines Netzwerkes von einem externen Lehrer definierte *gewünschte* Ausgaben existieren, und das Netzwerk *niemals* selbst Ausgabeaktivationen *entdecken* muß. (Der Lehrer kann dabei unter Umständen durch die Umgebung selbst oder aber sogar durch ein anderes Netzwerk gegeben sein.) Für den überwachten Lerner stellt die externe Rückkopplung kein kritisches Problem dar. Es gibt *keine* durch die externe Rückkopplung verursachten unerwünschten *Eingaben.*

Beispielsweise kann man mit Hilfe eines überwachten Algorithmus einem unseren Roboter steuernden Netzwerk beibringen, seinen Stab zu balancieren. Voraussetzung ist allerdings, daß ein gut informierter externer Lehrer angibt, welche Kraft zu welchem Zeitpunkt auf den Stab ausgeübt werden soll, um die Balance zu halten. Die Aufgabe wird dadurch recht einfach und uninteressant, denn irgend jemand *muß ja vor dem Training schon wissen, wie das Problem zu lösen ist!*

R-Lernen

Ein Lernproblem ist ein Reinforcement-Lernproblem (ab jetzt auch oft 'R-Lernproblem' genannt), wenn ein evaluativer Prozeß zu isolierten Zeitpunkten der Aktivationsausbreitungsphase eines Netzwerkes lediglich feststellt, ob das System sich in einem wünschenswerten Zustand befindet oder nicht. *Keine* Information über Strategien zum Erreichen wünschenswerter Zustände wird bereitgestellt. Während der Trainingsphase wird vom Netzwerk erwartet, daß es *selbst* Ausgabeaktionen entdeckt, die letztlich zu wünschenswerten Zuständen führen. Vom Standpunkt des R-Lerners aus ist die Natur der externen Rückkopplung in hohem Maße relevant für das Erreichen seiner Ziele. *Offensichtlich ist R-Lernen viel schwieriger als überwachtes Lernen.*

R-Lernen und *adaptive Regelung* haben viel gemeinsam: Bei der adaptiven Regelung ist in der Regel einige Information über gewünschte Umgebungszustände vorhanden. Gerade wie beim R-Lernen ist aber *nicht* von vornherein bekannt, welche Ausgabesequenzen zur Erzielung gewünschter Umgebungszustände geeignet sind.

Von den drei Arten konnektionistischen Lernens (überwachtes Lernen, R-Lernen, unüberwachtes Lernen) ist R-Lernen diejenige, die am meisten mit dem aus der Biologie bekannten Lernverhalten zu tun hat: Simple

Evaluierungsfunktionen generieren abhängig von den Aktionen des Lernsystems Eingaben wie Schmerz, Lust etc... *Trotz ihrer Einfachheit 'wollen' simple Bewertungsfunktionen im allgemeinen komplizierte Aktionssequenzen erzwingen.*

Als Beispiel verwenden wir wieder unser Balancierproblem. Ein guter R-Algorithmus sollte imstande sein, dieses Problem (unter bestimmten jetzt nicht so wichtigen Zusatzbedingungen) auch dann schon zu lösen, wenn die einzige von der Umgebung zur Verfügung gestellte Lehrinformation in dem beim Umfallen des Stabes geäußerten Hinweis besteht: 'Das war schlecht'. Die schwierige Aufgabe des R-Algorithmus ergibt sich aus der zeitlichen Komponente des Lernproblems: Welche der vergangenen Aktionen *war* denn verantwortlich für den Mißerfolg?

1.2.3 Unüberwachtes Lernen

Der Begriff 'unüberwachte Lernalgorithmen' wird häufig nicht präzisiert. Normalerweise soll durch das Wort 'unüberwacht' jedoch zum Ausdruck gebracht werden, daß es im Gegensatz zum zielgerichteten Lernen keine instruktive oder evaluative sich auf etwaige 'Erfolge' oder 'Mißerfolge' beziehende Rückmeldung für das lernende System gibt. Das Konzept des '(Miß-)Erfolgs' existiert gar nicht. Es gibt also auch keine offensichtliche Zielgerichtetheit. Unüberwachtes Lernen muß dennoch nicht gänzlich ziellos sein (was hätte es denn dann auch für einen Sinn?): Es kann zum Auffinden gewisser *'Regelmäßigkeiten'* in den Eingaben dienen. (Dies wiederum mag es einem R-Lernalgorithmus erleichtern, Ziele zu verfolgen.)

Der Begriff *Regelmäßigkeit* ist mit gewissen Schwierigkeiten belastet. Was ist eine Regelmäßigkeit? Bei näherer Betrachtung dieser Frage wird ein Problem offenbar, das mit den unterschiedlichen Auffassungen des Begriffs 'Information' selbst zusammenhängt. Unterschiedliche Definitionen von Information (syntaktische Information oder Entropie, algorithmische Information, pragmatische Information) ziehen unterschiedliches Verständnis des Begriffes 'Regelmäßigkeit' nach sich. Eine nähere Beleuchtung dieser Problematik würde allerdings den Rahmen dieser Arbeit sprengen.

Weiterhin ist festzustellen, daß zu vielen möglichen Gewichtsmodifikationsregeln eine (intuitiv häufig unsinnige) Zielfunktion gefunden werden kann, welche durch die jeweilige Regel minimiert wird. Dadurch werden diese Regeln im nachhinein zu 'Lernregeln' (in Bezug auf die jeweilige Zielfunktion), und der Unterschied zwischen zielgerichtetem Lernen und unüberwachtem Lernen verschwimmt.

Spricht man im Rahmen von NN von Regelmäßigkeiten, so meint man meist Regelmäßigkeiten statistischer Natur, und faßt unter dem Begriff 'unüberwachte Lernalgorithmen' die sogenannten *'Clustering'*-Algorithmen

zusammen: 'Ähnliche' Muster sollen zum Beispiel in 'ähnliche' Kategorien klassifiziert werden, wobei das Maß für Ähnlichkeit meist recht einfach *und nicht selbst adaptiv* ist (z.B. euklidische Distanz o.ä.).

In den ersten 7 Kapiteln dieser Arbeit werden wir ohne unüberwacht lernende Komponenten auskommen, im wesentlichen deswegen, weil die von dieser Arbeit angesprochenen Probleme isoliert betrachtet werden sollen. Erst im Ausblick des letzten Kapitels werden Kombinationen aus zielgerichteten und unüberwachten dynamischen Lernsystemen vorgeschlagen werden.

1.2.4 Lokalität in Zeit und Raum

Wichtige Forderungen an Lernalgorithmen (und zugleich Schwerpunkte der Arbeit) sind die folgenden:

1. *'On-line'-Verwendbarkeit.* Weder in der Natur noch bei anspruchsvollen technischen Problemen macht die Umgebung plötzlich Pausen, um irgendwelchen lernenden Systemen Luft für aufwendige Lernprozesse zu schaffen. In der Regel fährt die Umgebung fort sich zu ändern, während das System lernt.

2. *Parallelisierbarkeit.* Aus Effizienzgründen wünscht man sich Lernalgorithmen, deren Ausführung auf viele Prozessoren verteilbar ist. Ideale Voraussetzung für die Parallelisierung von Algorithmen bietet die

3. *Lokalität in der Zeit und möglichst im Raum.* Am liebsten hätte man einen Lernalgorithmus, bei dem alle Knoten und Verbindungen zu jedem Zeitpunkt im wesentlichen dieselben Operationen ausführen. Dabei sollten sie sich jeweils nur um die engste räumliche und zeitliche Nachbarschaft kümmern, nicht jedoch um die globale Netztopologie, die zeitliche Struktur der Eingaben, etc... Ein lokaler Algorithmus ist aus heutiger Sicht auch *biologisch plausibler* als ein nicht lokaler: Bisher fand man bei neurophysiologischen Untersuchungen keine Anhaltspunkte für *nicht* lokale Berechnungen. Das Konzept der raumzeitlichen Lokalität ist wichtig und wird im folgenden detaillierter ausgeführt.

Schwache zeitliche Lokalität

Ein Lernalgorithmus für dynamische neuronale Netze heiße *zeitlich schwach lokal*, wenn beim *on-line*-Lernen für *gegebene* Netzwerkgröße (gemessen durch die Anzahl aller Verbindungen) und *gegebene* Netzwerktopologie der Spitzenberechnungsaufwand für jeden Zeitschritt gleich $O(1)$ ist, und zwar für zeitlich *beliebig* lange Eingabesequenzen.

Schwache räumliche Lokalität

Ein Lernalgorithmus für dynamische neuronale Netze heiße *räumlich schwach lokal*, wenn beim *on-line*-Lernen für *beschränkt* lange Eingabesequenzen und für *beliebige* Netzwerkgrößen (gemessen durch die Anzahl aller Verbindungen) und *beliebige* Netzwerktopologien der Spitzenberechnungsaufwand pro Verbindung und Zeitschritt gleich $O(1)$ ist.

Schwache Lokalität

Ein Lernalgorithmus für dynamische neuronale Netze heiße *schwach lokal*, wenn beim *on-line*-Lernen für *beliebig* lange Eingabesequenzen und für *beliebige* Netzwerkgrößen und *beliebige* Netzwerktopologien der Spitzenberechnungsaufwand pro Verbindung und Zeitschritt gleich $O(1)$ ist.

Starke Lokalität

Ein Lernalgorithmus für dynamische neuronale Netze heiße *stark lokal*, wenn

1. er schwach lokal ist,

2. jeder Knoten und jede Verbindung zu jedem Zeitpunkt dieselbe Operation ausführen, es keine zeitliche Trennung zwischen Gewichtsänderungsphase und Aktivationsausbreitung gibt, und keine von einem externen Lehrer definierte Trainigsintervallgrenzen für den Lernvorgang notwendig sind.

Ein stark lokaler Algorithmus kann auf beliebig strukturierten Netzen funktionieren. Jeder Knoten darf zu einem gegebenen Zeitpunkt *ausschließlich* auf Information von Knoten angewiesen sein, mit denen er verbunden ist. Jede Verbindung darf zu jedem Zeitpunkt ausschließlich auf Information von denjenigen Knoten angewiesen sein, die sie verbindet, sowie eventuell auf Information von einem externen Lehrer.

Adaption unter den Bedingungen der starken Lokalität entspricht weitgehend der *Selbstorganisation* von massiv parallelen, aus vielen lokal kommunizierenden Einzelteilen bestehenden Systemen.

1.3 Gliedernde Übersicht der Arbeit

Zwar konzentrieren sich nahezu alle bisherigen Arbeiten zu adaptiven NN auf überwachte Lernalgorithmen für statische Umgebung. NN bieten jedoch Möglichkeiten, auch das fundamentale raumzeitliche Lernproblem anzugreifen. In dieser Arbeit wird die bisher meist vernachlässigte zeitliche Komponente des allgemeinen Lernproblems betont.

Die allgemeine Problematik wird bei allen folgenden Kapiteln stets dieselbe bleiben. Die vorgestellten Algorithmen werden sich auf verschiedene Aspekte des allgemeinen Problems beziehen. Mit der Vorstellung jeder Algorithmenklasse ist eine experimentelle Demonstration ihrer Fähigkeiten (oder Grenzen) verbunden.

Zwar wird das fundamentale raumzeitliche Lernproblem am Ende nicht vollständig und in jeder Hinsicht befriedigend gelöst sein. Einige substantielle Beiträge dieser Arbeit bieten jedoch ermutigende Perspektiven für weiterführende Arbeiten. Die Dissertation ist wie folgt gegliedert:

Im 2. Kapitel wird Vorwissen zum überwachten Lernen zur Verfügung gestellt: Was ist bereits möglich? Der Schwerpunkt liegt dabei natürlich auf den wenigen existierenden Lernmethoden für algorithmische Dynamik. Es wird eingegangen auf den Gradientenabstieg in beliebigen azyklischen Netzwerken, in zyklischen Equilibriumsnetzen mit stationären Ein- und Ausgaben, und schließlich in dynamischen Netzen. Weiterhin werden die Methoden der zeitlichen Differenzen vorgestellt, welche als eine Generalisierung des Gradientenabstiegs angesehen werden können. Alle diese Ansätze sind in sinnvoller Weise in R-lernende System einbettbar, woraus dieses Kapitel auch seine Existenzberechtigung zieht.

Im 3. Kapitel wird Vorwissen zum R-Lernen zur Verfügung gestellt: Hierbei wird unterschieden zwischen auf *Modellbildung* basierenden R-Algorithmen (diese konstruieren adaptive Modelle bestimmter Aspekte der Umgebung) und solche ohne Umgebungsmodelle (Generate-and-Test-Verfahren). Die Generate-and-Test-Verfahren umfassen relevante nicht-neuronale Ansätze (Genetische Algorithmen, Dynamische Programmierung, etc.) sowie 'pures' neuronales R-Lernen in nicht-reaktiver Umgebung. Verschieden schwierige Unterfälle werden beleuchtet: Netzwerke ohne Rückkopplung und solche mit interner Rückkopplung. Die Motivation für den ersten eigenständigen Beitrag (die neuronale Eimerkette) wird gegeben.

Bei den modellbildenden Algorithmen werden zwei grundverschiedene Arten der Modellbildung getrennt: Einerseits kann man Modelle für Erwartungen an den jeweils nächsten Zeitschritt konstruieren, andererseits auch Modelle für kumulativ meßbare *zeitlich gedehnte* Ereignisse. In sehr unterschiedlicher Weise sind diese beiden Modellarten für die zielgerichtete Anpassung eines zentralen 'Steuernetzwerkes' ausnützbar.

In das zweite und das dritte Kapitel eingearbeitet finden sich Hinweise auf bisher ungelöste Probleme: *Bisher gab es keine stark lokalen Algorithmen, und vor allem gab es keine R-Algorithmen für interne und externe Rückkopplung.*

Im 4. Kapitel wird eine Klasse stark lokaler Lernalgorithmen beschrieben. Im 5. und 6. Kapitel werden nacheinander unterschiedliche Klassen von Lernalgorithmen für R-lernende Systeme mit interner und externer

Rückkopplung beschrieben.

Das erste Schema zeigt auf konstruktive Weise: Zielgerichtetheit und *völlig lokales* Lernen mit 'versteckten Knoten' sind kompatibel.

Das zweite Schema zeigt, wie man das fundamentale Lernproblem mittels *Systemidentifikation* und Gradientenabstieg in zwei vollständig rekurrenten interagierenden Netzen angreifen kann.

Das dritte Schema zeigt: 'Methoden der zeitlichen Differenzen' lassen sich auch auf Netze mit interner Rückkopplung anwenden. Unter Einbezug des zweiten Algorithmus lassen sich ferner in vorteilhafter Weise *mehrdimensionale* adaptive Kritiker konstruieren.

Die Anwendbarkeit der verschiedenen Methoden wird durch die Beschreibung mehrerer Experimente gezeigt. Dazu gehören u.a. klassische Experimente der nicht-linearen Art. Mit dem zweiten Verfahren wird erstmals ein Experiment zum Reinforcement-Lernen in Nicht-Markov-Umgebungen durchgeführt. Die Anwendung eines Verfahrens aus der dritten Algorithmenklasse zeigt anhand eines schwierigen Balancierproblems, daß das Konzept der mehrdimensionalen adaptiven Kritiker zu beträchtlichen Effizienzgewinnen (im Vergleich mit konkurrierenden Ansätzen) führen kann.

Im 7. Kapitel wird am Beispiel *'attentive vision'* gezeigt, daß das unüberwachte *Erlernen* selektiver *räumlicher* Aufmerksamkeit möglich ist.

Weitere Motivation ist bei diesem Beitrag, die kaum erfolgreichen und ineffizienten rein statischen Ansätze zur visuellen Mustererkennung durch einen effizienten, mehr sequentiellen Ansatz zu ersetzen, um damit die Vorteile der Beachtung der zeitlichen Dimension auch für scheinbar statische Probleme zu illustrieren. Dieser Ansatz ist inspiriert durch die Beobachtung, daß biologische Systeme den Mustererkennungsprozeß auf sequentielle Augenbewegungen abstützen. Ein aus zwei interagierenden Netzwerken bestehendes System soll lernen, sequentielle Fokustrajektorien zu erzeugen, so daß die finale Position eines durch 'Augenmuskulatur' bewegten Fokus einem zu findenden Objekt in einer visuellen Szene entspricht. Die einzige Zielinformation besteht aus einer zu dem zu findenden Objekt korrespondierenden gewünschten finalen Eingabe. Trotz der Komplexität des zugehörigen 'temporal credit-assignment problem' wird gezeigt, daß es möglich ist, korrekte Sequenzen von Fokusbewegungen unter Einschluß von Translationen und Rotationen lernen zu lassen.

Im Rahmen der Untersuchungen zur selektiven Aufmerksamkeit finden sich weiterhin einige Betrachtungen zum Thema Neugier und Langeweile. Die Motivation ist das für lernende Systeme oft notwendige Wechselspiel zwischen Exploration und Zielgerichtetheit. Die Wichtigkeit dieses Wechselspiels wird betont, und es wird ausgeführt, wie sich modellbildenden Algorithmen in natürlicher und sinnvoller Weise 'Neugierverhalten' bzw. 'Langeweileverhalten' einbauen läßt.

Das 8. Kapitel kritisiert alle vorangegangenen Kapitel. Trotz ihrer Allgemeinheit und ihrer experimentell unter Beweis gestellten Fähigkeit zur adaptiven Performanzverbesserung sind die dort vorgestellten Algorithmen nämlich in verschiedener Hinsicht immer noch unbefriedigend. Keiner dieser Algorithmen (und auch kein Algorithmus irgendeines anderen Autors) ermöglicht *selektive raumzeitliche Aufmerksamkeit* und kompositionelles Lernen.

Daher wird mit dem ersten *adaptiven neuronalen Subzielgenerator* auf konstruktive Weise gezeigt, daß das *Erlernen* des hierarchischen Aufstellens von Subzielen und das *Erlernen* von selektiver *zeitlicher* Aufmerksamkeit und von *'Teile und herrsche'* Strategien möglich ist. Ein Experiment zeigt, wie Subzielgenerierung *gelernt* werden kann.

Damit ergibt sich zum ersten Mal eine Möglichkeit, einen sogenannten *'higher-level-process'* (in diesem Fall zeitüberbrückende Planung) adaptiv zu machen. Der abschließende Ausblick weist auf perspektivenreiche Möglichkeiten für *introspektive* neuronale Lernalgorithmen hin. Einige Grundzüge für neuronales *Meta*-Lernen werden skizziert, im Rahmen der Dissertation allerdings nicht mehr implementiert.

Im Appendix schließlich finden sich verschiedene mathematische Details, die für die wesentlichen Aussagen dieser Arbeit nicht von zentraler Bedeutung sind.

1.4 Danksagung

Folgende Personen trugen zum Gelingen dieser Arbeit bei, dafür sei ihnen Dank ausgedrückt. Mein Doktorvater Wilfried Brauer und mein Betreuer Christian Freksa ließen mir viel Freiraum für das Verfolgen eigener Ideen. Bernd Schürmann unterzog die Arbeit einer kritischen Prüfung und wies mich auf manche Verbesserungsmöglichkeit hin. Auch Andy Barto, Thomas Laußermair, Richard Sutton, Gerhard Weiss, und Ron Williams (in alphabetischer Reihenfolge) trugen durch Kommentare zu Vorveröffentlichungen oder durch im Zusammenhang mit dieser Arbeit stehende Diskussionen zu ihrer letztendlichen Fassung bei. Auf das Betreiben von Werner Remmele und Wolfram Büttner gewährte mir die SIEMENS AG neben einem Stipendium gelegentlich auch finanzielle Unterstützung für Konferenzreisen.

Dank schulde ich vor allem meinem erfolgreichen Studententeam: Joseph Hochreiter, Rudolf Huber und Klaus Bergner führten einen Teil der Experimente durch und erwiesen sich als zuverlässige Programmierer und Tester. Besonderer Dank gebührt auch meiner Mutter, die sich freiwillig der Mühe des Korrekturlesens unterzog.

Ohne den Zugang zu Rechnern der TU-München (einer *Symbolics 3640*,

sowie mehreren *SUN SPARC stations*), insbesondere ohne den Zugang zur elektronischen Post wäre es mir nicht möglich gewesen, rechtzeitig die neuesten Informationen über relevante Entwicklungen zu beschaffen. Der vorliegende Text sowie die zugehörige Literatursammlung wurden (ebenfalls an der TUM) mit Hilfe des Textverarbeitungsprogramms LATEX formatiert.

Kapitel 2

Grundlagen: Überwachtes Lernen

2.1 Überblick

Das zentrale Thema dieser Dissertation sind Lernalgorithmen für die allgemeine Situation, in der kein intelligenter Lehrer zur Verfügung steht. Wozu dann ein Kapitel zum *überwachten* Lernen?

Der Zweck dieses Kapitels ist ein zweifacher. Zum einen werden wir später sehen, daß man überwachte Techniken in sehr nützlicher Weise auch bei generellen Lernsituationen mit einbinden kann. Zwar kann man überwachte Netze per definitionem nicht direkt auf Probleme ansetzen, bei denen zwischen Ausgabeaktionen und erwünschten Zuständen eine Lücke (geschaffen durch unbekannte Eigenschaften einer externen Umgebung) klafft. Allerdings kann man überwachte Netze verwenden, um bestimmte Aspekte dieser Lücke zu modellieren. Dabei kann die Umgebung selbst eine lehrende Funktion übernehmen. Modelle von Umgebungseigenschaften können ihrerseits eine entscheidende Rolle als Hilfsmodule für ein allgemeineres System spielen, in das sie eingebettet sind. Drei der in dieser Arbeit vorgestellten Algorithmen sind in verschiedener Weise auf 'selbstüberwachte' Hilfsmodule angewiesen. Man kann sie nicht verstehen, wenn man nicht vorher die Konzepte verstanden hat, die in diesem Kapitel vorgestellt werden.

Zum anderen soll aufgezeigt werden, welche Schwierigkeiten mit den existierenden überwachten Techniken verbunden sind. Im Rahmen der Vorstellung dieser Algorithmen wird insbesondere auf ihre relativen Stärken und Schwächen eingegangen. Schwerpunkt der Kritik ist die nicht vorhandene Lokalität in Zeit und Raum. Die Kritik liefert zusätzliche Motivation für

weiterführende Kapitel.

Das Kapitel ist wie folgt gegliedert: In knapper Form wird zunächst auf die bekannten Algorithmen für azyklische Netze eingegangen. Dies dient im wesentlichen der Vorbereitung auf den zyklischen Fall. (Zusätzliche Motivation für die Beschreibung azyklischer Netze erwächst aus der Tatsache, daß in späteren Kapiteln auch statische Netzwerke sinnvoll als Module in größere dynamische Systeme eingebettet werden. Auch zeigt sich schon am azyklischen Fall ein für bisherige Verfahren typischer Nachteil, nämlich die nicht vorhandene Lokalität in Zeit und Raum.)

Im Anschluß werden Algorithmen für zyklische Netze beschrieben. Die Menge der zyklischen Netze zerfällt in zwei Untersorten: Equilibriumsnetze und andere. Erstere werden in ebenfalls knapper Form exemplarisch betrachtet, letztere machen den Hauptteil dieses Kapitels aus. Die allgemeinsten der Algorithmen für dynamische zyklische Netze sind (wenigstens prinzipiell) geeignet, beliebige Zeitverzögerungen zwischen beliebigen Ereignisfolgen und späteren Konsequenzen zu entdecken.

Schließlich wird im Rahmen des überwachten Lernens auf Methoden der zeitlichen Differenzen eingegangen. Die in dem entsprechenden Abschnitt beschriebenen Konzepte sind insofern allgemeiner Natur, als sie nicht an bestimmte Netztopologien gebunden sind. Sie sind ebenfalls relevant für spätere Kapitel zum Reinforcement-Lernen.

2.2 Statische Netzwerke

Wir rufen uns in Erinnerung, daß ein Netzwerk statisch genannt wird, wenn seine Ein- und Ausgaben nicht zeitvariant sein dürfen. Dies schließt nicht etwa aus, daß vielleicht interne Rückkopplung für interne Dynamik sorgt, wie sie zum Beispiel in Equilibriumsnetzen von Bedeutung ist.

2.2.1 Statische Netze ohne Rückkopplung

Zunächst definieren wir, was wir bei einem gegebenen Netz unter einer *Lage* verstehen.

Die erste Lage ist die Menge aller Eingabeknoten. Die n-te Lage ist die Menge aller Knoten, zu denen ausgehend von der ersten Lage mindestens ein Kantenpfad der Länge $n-1$ führt, aber kein Kantenpfad der Länge $m \geq n$.

Der Sinn dieser Definition ist, auch direkte Verbindungen ('Abkürzungen') zwischen nicht aufeinanderfolgenden Lagen in angemessener Weise zu berücksichtigen.

Die Lage mit der höchsten Nummer wird meist als Ausgabelage benutzt. Eine Aktivationsausbreitungsphase verläuft wie folgt (alle zum Algorithmus gehörigen Aussagen werden in Schrägdruck wiedergegeben):

Ein Eingabemuster aus dem R^m wird an der aus m Knoten bestehenden Eingabelage angelegt, die Aktivation der Eingabeknoten ist gleich ihrer Eingabe. In sukzessiver Manier berechnet jeder Knoten k einer Lage n die gewichtete Summe der Aktivationen von Knoten aus Lagen mit Nummern kleiner als n (sofern er von diesen Verbindungen erhält), und übergibt das Resultat einer monoton wachsenden differenzierbaren Aktivierungsfunktion, welche zur Berechnung der Knotenausgabe von k dient.

Offensichtlich dürfen nicht alle Knoten parallel behandelt werden, sondern nur die jeweils zu einer Lage gehörigen.

Nach Abschluß der Ausbreitungsphase wird in der Regel eine Diskrepanz zwischen dem *tatsächlichen* Ausgabevektor x in der Ausgabelage und dem durch einen wohlinformierten Lehrer definierten *gewünschten* Ausgabevektor d offenbar. Das Problem besteht nun darin, eine Fehlerfunktion zu minimieren, die diese Diskrepanz zum Ausdruck bringt. Typischerweise verwendet man als Fehlerfunktion $\|d-x\|^2$ und berechnet ihren Gradienten bezüglich der Gewichte. Eine Lernregel kommt zur Anwendung und wirkt sich auf die Gewichte aus:

$$\triangle w = -\eta \left(\frac{\partial \|d-x\|^2}{\partial w} \right)^T .$$

Dabei ist w der Gewichtsvektor des Netzes, $\triangle w$ ist die von der Lernregel induzierte Gewichtsänderung, und η ist eine positive *Lernrate*.

Bei obiger Formel haben wir uns die Indizes erspart, die für die Unterscheidung verschiedener Musterpaare notwendig wären. Meist soll jedoch ein und dasselbe Netz trainiert werden, viele verschiedene Musterassoziationen abzuspeichern. Hat man also ein Ensemble von zu erlernenden Ein-/Ausgabeassoziationen, so wird als Gesamtfehlerfunktion die Summe der zu den jeweiligen Musterpaaren gehörigen Fehlerfunktionen verwendet. Da der Gradient der Summe gleich der Summe der Gradienten ist, ändert sich für ein Musterpaar im wesentlichen nichts an der im folgenden beschriebenen Fortführung des Lernalgorithmus, welche sich an die oben beschriebene Ausbreitungsphase anschließt:

Die Richtungen aller Verbindungen im Netz werden umgekehrt. Die Fehlerdifferenzen werden an der Ausgabelage angelegt und jeweils multipliziert mit der Ableitung der Aktivierungsfunktion an der Stelle, die durch die Aktivationsausbreitungsphase bestimmt wurde. Die Fehlersignale der Ausgabeknoten sind damit definiert. In sukzessiv absteigender Manier berechnet jeder Knoten k einer Lage n die gewichtete Summe der Fehlersignale von

Knoten aus Lagen mit Nummern größer als n. Um sein eigenes Fehlersignal zu erhalten, multipliziert k das Resultat mit der Ableitung der Aktivierungsfunktion an der Stelle, die durch die Aktivationsausbreitungsphase bestimmt wurde.

Offensichtlich muß sich also jeder Knoten aus der ersten Phase noch seine Aktivation gemerkt haben. Wiederum dürfen nicht alle Knoten parallel behandelt werden, sondern nur die jeweils zu einer Lage gehörigen. Man beachte die fast vollständige Symmetrie zwischen Vorwärts- und Rückwärtsausbreitung.

Schließlich ändert sich jedes Gewicht proportional zur Aktivation seines Quellknotens und zum Fehler seines Zielknotens.

Das Resultat ist im wesentlichen ein Gradientenabstieg im Gewichtsraum, wie er durch obige Gleichung gefordert wird. (Mathematisch gerechtfertigt wären Gewichtsänderungen eigentlich erst am Ende der Gradientenberechnung für *alle* einzulernenden Musterassoziationen, nicht schon nach jeder Musterpräsentation. In der Praxis nimmt man jedoch oft mit Erfolg an, daß die Lernrate klein genug ist, um Instabilitäten zu vermeiden.)

Spezialfälle dieses Algorithmus (für 2-lagige Netzwerke) wurden bereits in den 60er Jahren gefunden. Werbos [72] beschrieb als erster den allgemeineren Fall, welcher 'versteckte Knoten' *(hidden units)* erlaubt. Der Algorithmus mußte jedoch dreimal wiederentdeckt werden [35][25][44], bevor er einer breiteren Öffentlichkeit bekannt wurde. Heute verwendet die überwiegende Mehrzahl der Anwender und auch der Forscher Varianten dieses Verfahrens, obwohl es in mancher Hinsicht äußerst unbefriedigend ist. So ist es zum Beispiel in offensichtlicher Weise nicht im engeren Sinne lokal in Zeit und Raum. Eine globale Instanz muß darüber wachen, ob die Ausbreitungsphase abzuschließen ist und die Rückphase anzufangen hat, wann welcher Knoten welche Berechnungen auszuführen hat, etc.etc. (Kapitel 4 dieser Dissertation beschreibt den ersten wirklich lokalen Lernalgorithmus für neuronale Netzwerke mit 'versteckten Knoten').

Mittlerweile wurde einige Arbeit investiert, um das Gradientenabstiegsverfahren mit Hilfe von Verfahren der zweiten Ordnung zu beschleunigen ([36] und [9] sind nur zwei von vielen Referenzen zu diesem Thema). Statt in aufwendiger Weise zweite Ableitungen der Fehlerfunktion zu berechnen oder zu approximieren, kann man auch versuchen, durch große Sprünge im Gewichtsraum die *Nullstellen* der Fehlerfunktion zu finden [49]. Die entsprechende Methode wird im Anhang kurz umrissen.

Wie alle anderen werden auch wir das oben beschriebene Vefahren als 'Back-Propagation' (BP) bezeichnen.

2.2.2 Statische Netzwerke mit interner Rückkopplung

Wie schon verschiedentlich bemerkt, darf ein statisches Netzwerk durchaus interne Rückkopplung erlauben. Voraussetzung ist allerdings, daß die interne Netzwerkdynamik schließlich zu einem stationären Equilibriumszustand führt. Verschiedene Lernalgorithmen für überwachte Equilibriumsnetzwerke wurden vorgeschlagen. Nach unserer Definition von Überwachtheit gehören dazu Hintons und Sejnowskis Boltzmann-Maschine [14] ebenso wie die vor allem bei Physikern beliebten Hopfield-Netzwerke [17] und manche ihrer asymmetrischen Erweiterungen (siehe z.B. [65]). Hopfield-Netzwerke illustrieren, wie man die Hebbsche Regel [12] in überwachter Manier einsetzen kann.

Stellvertretend für alle Equilibriumsnetze betrachten wir in diesem Abschnitt lediglich eine relativ junge Entwicklung: den Gradientenabstiegsalgorithmus für zyklische statische Netzwerke, wie er von Almeida [1] und davon unabhängig von Pineda [38] vorgeschlagen wurde. Auch Rohwer und Forrest haben im selben Jahr ein verwandtes Verfahren beschrieben [43].

Der Algorithmus minimiert für ein einzuspeicherndes Musterpaar (i, d) dieselbe Fehlerfunktion wie konventionelles Back-Propagation: $\|d - x\|^2$, wobei x tatsächlicher Ausgabevektor und d gewünschter Ausgabevektor ist.

Das Netz kann vollständig rückgekoppelt sein. Natürlich gibt es in voll vernetzten Systemen keine Netzlagen im Sinne obiger Definition mehr: Zu jedem Knoten führen Kantenpfade beliebiger Länge. Dennoch läuft der Algorithmus weitgehend genauso ab wie derjenige für azyklische Netzwerke. Die Aktivationsausbreitungsphase verläuft in mehreren von einem globalen Taktgeber gesteuerten Schritten wie folgt:

Ein Eingabemuster wird an bestimmten Netzknoten angelegt, die Aktivation der Eingabeknoten ist gleich ihrer externen Eingabe und ändert sich von nun an nicht mehr. Jeder Knoten k, der kein Eingabeknoten ist, berechnet synchron mit allen anderen die gewichtete Summe der Aktivationen aller Knoten, von denen eine Verbindung auf k führt. Das Resultat wird der Aktivierungsfunktion übergeben, welche zur Berechnung von k's neuer Aktivation dient. Dieses Vorgehen wird iteriert, bis sich bei aufeinanderfolgenden Iterationen keine Knotenaktivation mehr signifikant ändert.

Anschließend findet die Fehlerpropagierungsphase statt:

Die Richtungen aller Verbindungen im Netz werden umgekehrt. Die Fehlerdifferenzen der Ausgabeknoten werden multipliziert mit der Ableitung der Aktivierungsfunktion an der Stelle, die durch die Aktivationsausbreitungsphase bestimmt wurde. Das Fehlersignal der Ausgabeknoten ist damit definiert. Jeder Knoten k, der kein Eingabeknoten ist, berechnet nun die gewichtete Summe der Fehlersignale von allen Knoten, von denen eine Ver-

bindung auf k führt. Um sein eigenes Fehlersignal zu erhalten, multipliziert k das Resultat mit der Ableitung der Aktivierungsfunktion an der Stelle, die durch die Aktivationsausbreitungsphase bestimmt wurde. Dieses Vorgehen wird iteriert, bis sich bei aufeinanderfolgenden Iterationen keine Fehlersignale mehr signifikant ändern.

Schließlich ändert sich jedes Gewicht proportional zur Aktivation seines Quellknotens und zum Fehlersignal seines Zielknotens.

Führt die Aktivationsausbreitungsphase in ein Equilibrium, so auch die Fehlerpropagierungsphase. Dies ist eine Konsequenz der Tatsache, daß die Gewichtsmatrix für die Rückwärtsphase der Transponierten der Gewichtsmatrix für die Vorwärtsphase entspricht. Da die Stabilität eines Systems nur von den Eigenwerten seiner Gewichtsmatrix abhängt und die Eigenwerte einer Matrix gleich den Eigenwerten der Transponierten sind, ergibt sich obige Aussage [1]. Man beachte erneut die fast vollständige Symmetrie zwischen Vorwärts- und Rückwärtsausbreitung.

2.2.3 Dynamik von der trivialen Art

In den letzten beiden Abschnitten wurden Algorithmen zur statischen Musterassoziation beschrieben. Mit solchen Algorithmen läßt sich eine triviale Art von dynamischem Verhalten erlernen: Man trainiert ein Netzwerk mittels einer geordneten Sequenz von Musterpaaren $(i_t, d_t), t \in \{1, \ldots, n\}$, bei der die Ausgabe d_t gleich der Eingabe i_{t+1} ist, für $t \in \{1, \ldots, n-1\}$. Im Arbeitsmodus kopiert man zwischen aufeinanderfolgenden Zeitschritten die Ausgaben eines solchermaßen trainierten Netzwerkes zurück auf seine Eingabeknoten. Selbstverständlich lassen sich auf diese Weise niemals über mehr als einen Zeitschritt reichenden temporalen Abhängigkeiten erreichen: Die Ausgabe zur Zeit $t+1$ ist durch die Ausgabe zur Zeit t vollständig determiniert.

Keinen qualitativen Fortschritt erzielt man, wenn man 'Zeitfenster' in der Eingabe zuläßt. (Eine ganze Reihe von Ansätzen zur temporalen Mustererkennung beruht auf dem Zeitfensterprinzip (manchmal läuft es auch unter einem anderen Namen wie z.B. *'distribution of delays'* [13]).) Zeitfenster erfordern, die Menge der Eingabeknoten aufzublähen und Eingaben aus den jeweils letzten k Zeitschritten für die Generierung von Ausgaben zu verwenden. Solch ein Vorgehen kann sich als nützlich erweisen, wenn bekannt ist, daß man wirklich nie mehr als k vergangene Zeitschritte berücksichtigen muß, um korrekte Ausgaben zu erzielen. Selbst in diesem Fall wird die Methode jedoch in der Regel ineffizient sein. Für den allgemeinen Fall bietet sie keine Perspektive.

2.3 Interne Rückkopplung: Dynamische Netzwerke

Im folgenden beschreiben wir Algorithmen für den allgemeinsten Fall überwachten Lernens. Ein- und Ausgaben brauchen nicht mehr stationär zu sein, Eingaben zu bestimmten Zeitpunkten sollen unter Umständen Ausgaben zu beliebigen späteren Zeitpunkten beeinflussen können. Dazu braucht man interne Rückkopplung. Erst seit kurzem kennt man funktionstüchtige Lernalgorithmen für das auftretende raumzeitliche *'credit-assignment problem'*.

Die Aufgabe für einen solchen Lernalgorithmus stellt sich im allgemeinen wie folgt: Gegeben eine Sequenz von Eingabevektoren i_t, eine Sequenz von gewünschten Ausgabevektoren d_t, und eine Fehlerfunktion, typischerweise $\sum_t \|d_t - x_t\|^2$. Minimiere die Fehlerfunktion!

(Dabei ist x_t der aktuelle Ausgabevektor zur Zeit t, und t durchläuft alle Zeitschritte eines Trainingsintervalls.)

Man beachte, daß die Aufgabenstellung allgemein genug ist, um zeitliche Abhängigkeiten beliebiger Natur zuzulassen. Dies bedeutet einen qualitativen Sprung von der *'Dynamik der trivialen Art'* hin zur *'algorithmischen Dynamik'*. Wer obige Fehlerfunktion minimieren kann, kann nicht nur sequentielle Assoziationen, sondern im Prinzip auch *Algorithmen im weitesten Sinn* lernen lassen.

Alle bekannten Ansätze zielen nun darauf ab, den Gradienten der Fehlerfunktion bezüglich der Gewichtsmatrix zu berechnen oder zu approximieren:

$$\triangle w^T = -\eta \frac{\partial \sum_t \|d_t - x_t\|^2}{\partial w}.$$

Dabei ist w der Gewichtsvektor des Netzes, $\triangle w$ ist die von der Lernregel induzierte Gewichtsänderung, und η ist eine positive Lernrate.

Wer den Gradienten obiger Fehlerfunktion bestimmen kann, kann auch den Gradienten einer Summe derartiger Fehlerfunktionen bestimmen (der Gradient der Summe ist die Summe der Gradienten). Damit können mehrere Sequenzen oder Algorithmen in überlagerter Form in das Netzwerk eingespeichert werden.

2.3.1 Brauchbare Ad-Hoc Lösungen für dynamische Netze

Ein im eingeschränkten Sinne funktionstüchtiger Ansatz für das Lernen von Sequenzen wurde von Jordan beschrieben [20]. Bei Jordans Methode

wird nicht wirklich die oben angegebene Fehlerfunktion minimiert. Vielmehr wird ein konventionelles azyklisches Back-Propagation-Netz mit einigen Zusatzeigenschaften versehen, die es ihm erlauben, bestimmte einfachere Arten 'algorithmischer' zeitlicher Abhängigkeiten zu erlernen.

Die Eingabelage eines solchen Netzes ist zweigeteilt. Es existiert eine Menge N von 'normalen' Eingabeknoten, und eine Menge Z von 'Zustandsknoten'. $\mid Z \mid = \mid O \mid$, wobei O die Menge der Ausgabeknoten ist.

Zu Beginn werden die Knotenaktivationen von Z mit 0 initialisiert. Zu jedem Zeitpunkt t wird N mit einer externen Eingabe versehen, worauf in konventioneller Weise eine Aktivationsausbreitungsphase durch die 'versteckten Knoten' bis hin zu den Ausgabeknoten stattfindet. Ausgehend von den an der Ausgabelage O auftretenden Fehlern schließt sich in konventioneller Weise eine Fehlerausbreitungsphase zurück zur Eingabelage an. Zum Abschluß des Zeitschrittes ändern sich die Gewichte nach den Regeln des statischen Gradientenabstiegs.

Die Aktivationen für Z zum Zeitpunkt $t+1$ berechnen sich nun wie folgt: Jeder Knoten $o \in O$ beeinflußt mit seiner Aktivation o_t zum Zeitpunkt t genau einen Knoten $z \in Z$, so daß dessen Aktivation z_{t+1} zum nächsten Zeitschritt gegeben ist durch

$$z_{t+1} = f(\gamma z_t + o_t).$$

Hierbei ist f die sigmoide differenzierbare Aktivierungsfunktion, und $0 \leq \gamma \leq 1$ eine Abschwächungsrate. Der Effekt der Prozedur ist, daß Z Spuren vergangener Aktivationen der Ausgabeeinheiten in sich trägt. Durch den Beitrag des exponentiellen Schwunds von Aktivationen der Knoten aus Z kann der Lernprozeß zumindest im Prinzip beliebig weit zurückliegende Ereignisse mit berücksichtigen.

Elman [8] beschrieb eine Modifikation des obigen Verfahrens. In seiner Version ist es nicht die Ausgabelage, sodern eine Lage mit 'versteckten Knoten', die in analoger Weise zur Beeinflussung von Z dient. Dadurch verschwindet die Abhängigkeit von den Ausgabeknoten (deren Aktivationen ja auch den externen Wünschen gehorchen sollen), man gewinnt etwas an Allgemeinheit.

Einige interessante Experimente wurden mit den in diesem Abschnitt beschriebenen (und verwandten) Methoden durchgeführt, darunter erfolgreiche Experimente zur Sequenzerkennung und zur Sequenzgenerierung (e.g. [8]).

Jordans und Elmans Algorithmen haben den Vorteil, daß sie zumindest eingeschränkt lokal in Raum und Zeit sind. Sie genügen jedoch nicht der starken Definition von Lokalität. Und sie sind weniger generell als die im nächsten Kapitel beschriebenen Algorithmen.

2.3.2 Generelle Lösungen für dynamische Netze

Entfaltung eines zyklischen Netzes zu einem azyklischen Netz

Es gibt eine Erweiterung des Algorithmus für azyklische Netze auf den zyklischen Fall. Das Verfahren beruht auf einem Prinzip, das den etwas unglücklichen Namen *'unfolding in time'* trägt ('unfolding in space' wäre vernünftiger gewesen, wie man im folgenden sehen wird). Das Prinzip geht zurück auf Minsky und Papert [29], die darauf hinwiesen, daß die Aktivationsausbreitungsphase eines zyklischen dynamischen Netzes durch die Aktivationsausbreitungsphase eines wesentlich größeren azyklischen statischen Netzes beschrieben werden kann.

Für jeden Zeitschritt der Aktivationsausbreitungsphase, die ein Netzwerk mit rekurrenten Verbindungen durchläuft, legt man eine neue Kopie aller Knoten samt ihrer gegenwärtigen Aktivationen an. Die Topologie des zu konstruierenden 'virtuellen' azyklischen Netzes ergibt sich sofort dadurch, daß jede Kante des ursprünglichen Netzes transformiert wird in eine 'virtuelle' Kante, die in dem azyklischen Netz die Kopien der entsprechenden Knoten zu aufeinanderfolgenden Zeitschritten verbindet. Mit dem azyklischen Netz kann nun Back-Propagation betrieben werden wie gehabt. Da jeder Originalkante in der Regel eine Reihe von virtuellen Kanten (zwischen zu verschiedenen Zeitpunkten gehörigen Knotenkopien) entspricht, muß jede Gewichtsänderung einer Kante im zyklischen Netz gleich der Summe der Gewichtsänderungen der entsprechenden virtuellen Kanten im azyklischen Netz sein, um einen Gradientenabstieg im Gewichtsraum zu gewährleisten.

Es ist dabei natürlich nicht nötig, auch die Verbindungen und ihre Gewichte mehrfach zu kopieren, da es sich ja für jeden Zeitschritt um dieselben Verbindungen handelt. Zweckmäßigerweise führt man für jeden Knoten des zyklischen Netzes einen Stapel ein, auf den bei sukzessiven Zeitschritten der Aktivationsausbreitungsphase die jeweiligen Aktivationen gestapelt werden. Bei der anschließenden Fehlerpropagierungsphase werden die Aktivationen nach dem *'last-in-first-out'*-Prinzip wieder vom Stapel geholt, um die neuen Fehlersignale zu berechnen (siehe den Abschnitt über azyklische Netze).

Egal, wie man den Algorithmus implementiert, in jedem Fall braucht man Speicherplatz unbekannter Größe für den Fall, daß die Zeitdauern der Trainingsintervalle nicht alle bekannt sind. Bei vollständig zyklischen Netzen beträgt die Speicherkomplexität für einen Durchgang $O(sn+n^2)$, wobei s die Anzahl der Zeitschritte des Durchgangs und n die Anzahl der Knoten des Netzes ist. Die Zeitkomplexität beträgt $O(sn^2)$.

Beim 'On-line'-Lernen beträgt der Spitzenberechnungsaufwand pro Verbindung und Zeitschritt $O(s)$. Das Verfahren ist also weit davon entfernt,

lokal in der Zeit (im eingeschränkten Sinne) zu sein. Das liegt natürlich daran, daß im jeweils letzten Zeitschritt eines Trainingsintervalls der gesamte Fehlerpropagierungs- und Gewichtsänderungsprozeß untergebracht werden muß.

Auch ist die Methode darauf angewiesen, daß der externe Lehrer über Trainingsintervallsgrenzen Bescheid weiß.

All diese unangenehmen Begleiterscheinungen des Algorithmus lassen ihn biologisch unplausibel erscheinen. Falls jedoch die erwähnten Einschränkungen bei einer technischen Anwendung keine Rolle spielen, kann sich die Methode als brauchbarer erweisen als das im nächsten Abschnitt beschriebene Verfahren, welches dieselbe Fehlerfunktion minimiert.

Ein Verfahren, das lokal in der Zeit ist

Robinson und Fallside wiesen als erste darauf hin, daß die Fehlerpropagierungsphase nicht unbedingt notwendig ist [40](siehe auch [39]). Es ist möglich, schon zur Laufzeit der Aktivierungsausbreitungsphase Information aufzusammeln, die bei der Beobachtung von späteren Fehlern sofort zur Berechnung eines Fehlergradienten herangezogen werden kann. Vorteile des Algorithmus sind, daß er eingeschränkt lokal in der Zeit ist, und daß er bei fixer Netzgröße unabhängig von der Länge zu erlernender Sequenzen mit einer fixen Menge an Speicherplatz auskommt. Ein Nachteil ist, daß er nicht einmal eingeschränkt lokal im Raum ist. Der Speicherbedarf wächst schlimmstenfalls mit der dritten Potenz der Anzahl der Knoten.

Weil seine 'Lokalität in der Zeit' das Verfahren für unsere Zwecke (siehe Kapitel 6) jedoch sehr interessant macht, und weil die Methode noch nicht gut bekannt ist, werden wir in diesem Abschnitt genauer auf sie eingehen und auch ihre Herleitung aufschreiben. Dabei halten wir uns nicht an Robinsons und Fallsides Originalarbeit, sondern lehnen uns an Williams und Zipsers Beschreibung an [80]. Verwandte Methoden finden sich in [37], [10] und [42].

Die Menge der Eingabeknoten des Netzes sei I. Die Menge der Knoten des Netzes, die keine Eingabeknoten sind, sei U. $y_k(t)$ ist die Aktivation des Knotens k aus $U \cup I$ zur Zeit t, $d_k(t)$ ist die gewünschte Aktivation des Knotens k zur Zeit t, falls diese von einem Lehrer definiert worden ist, w_{ij} ist das Gewicht der gerichteten Verbindung von Knoten j zum Knoten i, falls diese Verbindung existiert, f_k ist die sigmoide differenzierbare monoton wachsende Aktivierungsfunktion des Knotens k, $net_k(t) = \sum_l w_{kl} y_l(t)$, $y_k(t+1) = f_k(net_k(t))$, und t rangiert im folgenden über alle Zeitschritte eines Trainingsintervalls.

$J(t) = \frac{1}{2} \sum_{d_k \ existiert} (d_k(t) - y_k(t))^2$ ist der zu einem Zeitschritt t auftretende Fehler, und $E = \sum_t J(t)$ ist die zu minimierende Fehlerfunktion.

Der Effekt der Lernregel soll sich wie folgt ausdrücken lassen:

$$\triangle w = -\alpha \frac{\partial E}{\partial w}.$$

Dabei ist w der Gewichtsvektor des Netzes, α eine positive Lernrate, und $\triangle w$ die von der Lernregel induzierte Gewichtsänderung.

Da der Gradient der Summe aller $J(t)$ gleich der Summe der entsprechenden Gradienten ist, genügt es, für jedes Gewicht w_{ij} den Wert

$$\triangle w_{ij}(t) = -\alpha \frac{\partial J(t)}{\partial w_{ij}} = -\alpha \sum_{k \in U} (d_k(t) - y_k(t)) \frac{\partial y_k(t)}{\partial w_{ij}}$$

zu berechnen. $\sum_t \triangle w_{ij}(t)$ liefert dann die Gesamtgewichtsänderung für w_{ij} am Ende des Trainingsintervalls.

Nun ist für alle Zeitschritte t außer dem letzten

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f'_k(net_k(t))(\sum_{l \in U} w_{kl} \frac{\partial y_l(t)}{\partial w_{ij}} + \delta_{ik} y_j(t)).$$

(δ_{ik} bezeichnet hier das Kroneckersche Delta und ist 1 für $i = k$ und 0 sonst.)

Für den ersten Zeitschritt ist die entsprechende Ableitung 0. Also kann man mit 0 initialisierte Variablen $p^k_{ij_{new}}$ und $p^k_{ij_{old}}$ zur inkrementellen Berechnung der $\frac{\partial y_k(t)}{\partial w_{ij}}$ einführen. Zu jedem Zeitpunkt t werden diese Variablen gemäß

$$p^k_{ij_{new}} \leftarrow f'_k(net_k(t))(\sum_{l \in U} w_{kl} p^k_{ij_{old}} + \delta_{ik} y_j(t))$$

aktualisiert und anschließend, wie oben ausgeführt, zur Berechnung von $\triangle w_{ij}(t)$ herangezogen.

Das im letzten Abschnitt erwähnte Problem mit den vordefinierten Trainingsintervallsgrenzen kann man jetzt umgehen: Statt die Gewichte erst am Ende eines Intervalls zu ändern (wie es mathematisch eigentlich korrekt wäre), ändert man sie zu jedem Zeitschritt. Die Lernrate muß dabei klein genug gewählt werden, um Instabilitäten zu vermeiden. Der Effekt ist, daß sich alle Trainingsintervalle 'überlappen'; die Intervallgrenzen verschwimmen und müssen nicht extern definiert werden.

2.4 Methoden der zeitlichen Differenzen

Dieser Abschnitt behandelt einen für uns sehr wichtigen Spezialfall überwachten Lernens. Es geht dabei darum, Voraussagen über Aspekte des

zukünftigen Verhaltens eines sich zeitlich ändernden Systems zu treffen. Während reine Gradientenabstiegsverfahren Gewichtsänderungen anhand von Unterschieden zwischen vorausgesagten und tatsächlichen Zuständen durchführen, verwenden Methoden der zeitlichen Differenzen (kurz 'TD-Methoden', 'TD' steht für *'Temporal Differences'*) Unterschiede zwischen *aufeinanderfolgenden Vorhersagen*. Erstaunlicherweise kommen TD-Methoden dort, wo sie anwendbar sind, nicht nur mit weniger Spitzenberechnungsaufwand als konventionelle Techniken aus, sondern produzieren unter bestimmten Voraussetzungen auch genauere Vorhersagen. *Wo* sind sie anwendbar? Zum Beispiel dort, wo es darum geht, den Endzustand eines dynamischen Prozesses vorherzusagen.

Für einige der in späteren Kapiteln vorgestellten Problembereiche bieten sich Methoden der zeitlichen Differenzen an, um bestimmte Eigenschaften einer dynamischen Umgebung zu modellieren. Das durch TD-Methoden gewonnene Modell dieser Eigenschaften kann für die adaptive Konstruktion zielgerichteter Programme ausgenützt werden. Dieser Abschnitt liefert die nötigen Grundlagen.

Es liege eine Sequenz von Beobachtungen $x_t, t \in \{1 \ldots n\}$ eines dynamischen Systems zu aufeinanderfolgenden Zeitpunkten t vor. Das Problem besteht darin, zu jedem Zeitpunkt $t \leq n$ eine Vorhersage P_t über den finalen Zustand $x_n = P_n$ zu treffen. Purer Gradientenabstieg würde bedeuten:

$$\triangle w^T = -\alpha \sum_{t=1}^{n-1} (P_n - P_t) \frac{\partial P_t}{\partial w}.$$

Dabei ist w wieder der Gewichtsvektor des Netzes, α eine positive Lernrate, und $\triangle w$ die von der Lernregel induzierte Gewichtsänderung.

TD(λ)-Methoden werden im Gegensatz dazu nun wie folgt definiert:

$$\triangle w^T = -\alpha \sum_{t=1}^{n-1} (P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \frac{\partial P_k}{\partial w}.$$

Hierbei ist $0 \leq \lambda \leq 1$ ein 'Schwundfaktor', welcher bestimmt, wie stark vergangene *Voraussageänderungen* in die Gewichtsänderung mit einfließen.

Bei den TD(λ)-Methoden handelt es sich um eine *Generalisierung des Gradientenabstiegs*. Für $\lambda = 1$ ergibt sich mit etwas Hin- und Herschaufeln von Indizes nämlich exakt der pure Gradientenabstieg.

Für $\lambda \neq 1$ allerdings bekommt man etwas, was kein Gradientenabstieg ist. Ist das, was man da erhält, nun gut, oder ist es schlecht? Es läßt sich zeigen, daß TD(λ) mit $\lambda \leq 1$ zumindest für spezielle Arten von dynamischen Systemen bessere Voraussagen ermöglicht als der pure Gradientenabstieg TD(1): Unter der Voraussetzung, daß das vorherzusagende System

sich als ein absorbierender Markov-Prozeß beschreiben läßt, konnte Sutton demonstrieren, daß *lineare* TD(0) zu den optimalen Voraussagen (im Sinne der 'maximum likelihood estimates') konvergiert [67]. (Wiederholung: Ein Markov-Prozeß ist ein Prozeß, bei dem die Weiterevolution von einem gegebenen Zustand zur Zeit t nicht von früheren Zuständen abhängt. Ein absorbierender Markov-Prozeß ist einer, bei dem die Wahrscheinlichkeit der Terminierung 1 ist.)

Ein weiterer wichtiger Vorteil der TD(λ)-Methoden ist ihre Zugänglichkeit für inkrementelle Gewichtsänderungen und dem damit verbundenen niedrigen Spitzenberechnungsaufwand. Am deutlichsten wird das für den Fall, der uns im weiteren am meisten interessieren soll, nämlich TD(0). Man führt für jeden Zeitpunkt $t < n$ einen Wert $\triangle w_t$ ein, welcher sich wie folgt berechnet:

$$\triangle w_t^T = -\alpha(P_{t+1} - P_t)\frac{\partial P_t}{\partial w}.$$

$\triangle w = \sum_t \triangle w_t$ liefert die Gesamtgewichtsänderung, welche stattfindet, nachdem alle Beobachtungen gemacht wurden. Schon während das dynamische System beobachtet wird, kann man in akkumulativen Variablen die Summe der bisher berechneten Gewichtsänderungen für jedes Gewicht inkrementell auf den neuesten Stand bringen. Daher ist der Spitzenberechnungsaufwand pro Zeitschritt und Gewicht $O(1)$.

Will man ein Verfahren, das komplett 'on-line' arbeitet, so kann man jede Gewichtsänderung in dem Moment durchführen, in dem sie berechnet wird. Für den TD(0)-Fall bedeutet das:

$$\triangle w_t^T = -\alpha(P_{x_{t+1},w_t} - P_{x_t,w_t})\frac{\partial P_{x_t}}{\partial w}.$$

Dabei ist P_{x_t,w_t} die entsprechende Voraussage basierend auf x_t und w_t.

Auch für Voraussagen *kumulativer Art* sind TD-Methoden geeignet. Später wird es zum Beispiel darauf ankommen, zu jedem Zeitpunkt t eine Summe noch ausstehender Reinforcementsignale R_k zu allen späteren Zeitpunkten k vorherzusagen. Nach der Lernphase soll für alle t gelten:

$$P_t = \sum_{k=0}^{m} \gamma^k R_{t+k+1}.$$

Hierbei kann m für angemessenes γ unendlich sein, denn $0 < \gamma \leq 1$ ist wieder eine 'Discountrate', welche bestimmt, wie stark *Erwartungen* über verschieden weit in der Zukunft liegende Ereignisse die Gewichtsänderung mit beeinflussen sollen. Da nach obiger Gleichung

$$P_t = R_{t+1} + \gamma P_{t+1},$$

gelten soll, muß $\triangle w_t$ im Fall TD(0) nun wie folgt berechnet werden:

$$\triangle w_t^T = -\alpha(R_{t+1} + \gamma P_{t+1} - P_t)\frac{\partial P_t}{\partial w}.$$

Für den 'on-line' Fall muß man Voraussagen wiederum mit Indizes versehen, die Gewichtsvektoren zu verschiedenen Zeitpunkten anzeigen.

Man sollte nicht vergessen, daß die vorausgesetzte Markov-Eigenschaft in der Praxis oft nicht gegeben ist. Für den Fall, daß die Umgebung (oder zumindest die für ein lernendes System *wahrnehmbare* Umgebung) nicht vom Markov-Typ ist, kann man allerdings TD-Methoden und generelle Gradientenabstiegsverfahren für rekurrente Netze in sinnvoller Weise kombinieren, wie sich in Kapitel 6 zeigen wird.

Kapitel 3

Grundlagen: R-Lernen und adaptive Steuerung

'Reinforcement'-Lernen (R-Lernen) und adaptive Steuerung haben etwas gemeinsam: Verglichen mit überwachtem Lernen sind sie 'auf dieselbe Art' wesentlich schwieriger. Weder beim R-Lernen noch bei der adaptiven Regelung physikalischer Prozesse steht nämlich im allgemeinen Fall ein wohlinformierter Lehrer zur Verfügung, der dem lernenden System mitteilt, *was* es *wann* zu tun hat.

Man mag einwenden, daß z.B. bei überwachten Gradientenabstiegsmechanismen für 'versteckte Knoten' ebenfalls viele interne Vorgänge vom Lehrer undefiniert bleiben (z.B. welcher versteckte Knoten wann wie aktiviert sein soll). Der fundamentale Unterschied zwischen R-Lernen und überwachtem Lernen ist jedoch: Beim R-Lernen erstreckt sich das, was unbekannt ist, auch auf die Aktivationen der *Ausgabeknoten* zu den verschiedenen Zeitpunkten.

Solange die Aktivationen der Ausgabeknoten bekannt sind, kann es, wie im letzten Kapitel ausgeführt, wenigstens möglich sein, einen Gradienten für Gewichte versteckter Verbindungen anzugeben. Diese Möglichkeit ergibt sich daraus, daß die für eine bestimmte zu lernende Aufgabe erfoderliche interne Rückkopplung, wenn auch nicht vollständig, so doch von ihrer prinzipiellen Natur her bekannt ist: So ist bei Back-Propagation die prinzipielle Natur der internen Rückkopplung durch differenzierbare bekannte Funktionen von gewichteten Aktivationssummen gegeben. Die Einschränkung aller möglichen Arten von Rückkopplung auf eine durch die möglichen Gewichtskombinationen *parametrisierte Familie* von Rückkopplungen erlaubt es, bekannte differenzierbare Funktionen von Netzausgaben (z.B. Fehler-

funktionen) ihrerseits bezüglich der Gewichte zu differenzieren. Das ist das ganze Geheimnis überwachten Lernens.

Beim R-Lernen ist das Problem, daß die Funktion, welche Ausgaben auf Erfolge oder Mißerfolge abbildet, auch von ihrer prinzipiellen Natur her im allgemeinen nicht bekannt ist. Ein *evaluativer* Kritiker (anstelle eines *instruktiven* Lehrers) bewertet mittels einer Evaluierungsfunktion von Zeit zu Zeit die Effekte, die durch in der Regel zeitlich variierende Ausgaben eines lernenden Systems produziert werden.

Diese Bewertung durch die externe Evaluierungsfunktion ist oft sehr uninformativ: Beim R-Lernen beschränkt sie sich gerne auf binäre Entscheidungen wie 'ja, das war gut' und 'nein, das war schlecht'. Bildlich gesprochen übersetzt sie das Verhalten eines lernenden 'künstlichen Organismus' in unerwünschte Schmerzsignale oder erwünschte Lustsignale. Die Implementierung einer komplexen Evaluierungsfunktion kann für den 'Netzwerkarchitekten' (Eckmiller hat diesen Ausdruck geprägt) trivial sein, sofern er sich schon auf eine komplexe Umgebung abstützen kann und diese nicht selbst zu programmieren braucht (zukünftige Anwendungen neuronaler Netze zielen natürlich alle auf die 'reale' Umgebung).

Bei der *adaptiven Regelung* ist oft mehr Information über gewünschte Effekte vorhanden als beim R-Lernen, wie z.B. 'der Endpunkt des Roboterarms soll sich nach Abschluß seiner Bewegung bei den Koordinaten (3.55; 2.711; 0.22) befinden'. Sowohl beim R-Lernen als auch bei der adaptiven Steuerung ist aber zunächst unklar, welche Ausgaben zu welchem Zeitpunkt dazu beitragen, gewünschte Zustände zu erreichen. Die Evaluierungsfunktion, welche die Ausgaben bewertet, ist in der Regel unbekannt.

Wenn die Evaluierungsfunktion unbekannt ist, muß sie erforscht werden. Dieses Kapitel ist folgendermaßen gegliedert: Zunächst werden einige Methoden für R-Lernen beschrieben, die unter dem Stichwort *'generate-and-test'* zusammenfaßbar sind. Gelernt wird dabei im wesentlichen durch mehr oder weniger raffiniertes Probieren und Wahrscheinlichermachen bzw. Unwahrscheinlichermachen von Aktionssequenzen, die zu Erfolgen bzw. Mißerfolgen führten. In diesem Kontext werden neben entsprechenden Algorithmen für neuronale Netze auch nicht-neuronale Ansätze beschrieben, sofern sie potentiell relevant für neuronale Algorithmen sind: Wir gehen kurz ein auf genetische Algorithmen, auf *'dynamische Programmierung'* (einem etablierten Ansatz aus der mathematischen Steuerungstheorie, der für das Verständnis der später zu beschreibenden 'heuristischen dynamischen Programmierung' von Bedeutung ist), und insbesondere auf Hollands *'Bucket Brigade'* Prinzip für regelbasierte Systeme, welches eine Inspirationsquelle für den ersten eigenständigen Beitrag dieser Arbeit darstellt.

Anschließend beschreiben wir *'modellbildende'* Algorithmen (siehe auch [59]). Die Menge der modellbildenden Algorithmen zerfällt in mindestens

zwei Klassen.

Zum einen gibt es Algorithmen, die intelligenten Gebrauch von einem adaptiven 'kumulativen' Modell einer relevanten Größe (wie etwa dem gesamten zu allen späteren Zeitpunkten zu erwartendem Reinforcement) machen. Zu diesen Ansätzen der 'adaptiven Kritiker' gehören Reinforcementvergleichsalgorithmen basierend auf TD-Methoden, und ein Ansatz zur dynamischen heuristischen Programmierung. Einer der Beiträge dieser Dissertation erweitert diese Art von Algorithmen später auf einen allgemeinen Fall und illustriert ihre Anwendbarkeit durch Experimente.

Zum anderen gibt es Algorithmen, die versuchen, mit Hilfe eines adaptiven Modells Aspekte der Umwelt zu *simulieren*, um das Modell für zielgerichtete Lernprozesse auszunützen. Das ist der *Systemidentifikationsansatz*. Ein weiterer Beitrag dieser Dissertation erweitert auch diese Art von Algorithmen auf den allgemeinen Fall.

Um die modellbildenden Algorithmen zu beschreiben, werden wir uns häufig auf das Kapitel zum überwachten Lernen beziehen.

3.1 'Generate-and-Test'-Verfahren

3.1.1 Potentiell relevante nicht-neuronale Methoden

In diesem Abschnitt werden verschiedene Ansätze nicht-neuronaler Natur und ihre potentielle Anwendung auf neuronale Netze vorgestellt.

Erschöpfende Suche

Ist es nicht möglich oder ist man nicht gewillt, Performanzgradienten für ein lernendes System zu bestimmen, bietet einem die erschöpfende Suche den ersten Ausweg an. Kann ein lernendes System nur endlich viele interne Zustände annehmen, und sind die Prozesse, die es auszuführen lernen soll, von beschränkter zeitlicher Dauer (dies, um Schwierigkeiten mit dem Halteproblem zu vermeiden), so findet erschöpfende Suche stets 'beste' Lösungen.

Man mag argumentieren, daß erschöpfende Suche im Gewichtsraum eines neuronalen Netzes schon prinzipiell nicht möglich ist, da Gewichte durch reelle Zahlen dargestellt werden und somit ein Potential für überabzählbar viele potentielle Zustände vorhanden ist. Dem ist entgegenzuhalten, daß für alle gebräuchlichen Netzwerktypen und Aktivierungsfunktionen gewisse sich auf endlich viele Stützpunkte abstützende Varianten der erschöpfenden Suche zumindest im Prinzip erfolgversprechend wären. Dies liegt daran, daß 'fast immer' sehr nahe beieinanderliegende Gewichtsvektoren praktisch identisches Ausgabeverhalten zeitigen. (Man beachte, daß dem Problem *chaotischer* Dynamik durch die Einschränkung von Zeitdauern aus dem

Weg gegangen wurde.) Dennoch sind auch diese Varianten erschöpfender Suche für größere Netzwerke zu zeitaufwendig: Der Suchaufwand wächst exponentiell mit der Anzahl der Gewichte.

Dynamische Programmierung

Für gewisse 'mehrstufige Entscheidungsprozesse' (was das genau ist, wird unten beschrieben) gibt es etablierte Techniken, die in vieler Hinsicht wesentlich günstiger sind als erschöpfende Suche.

Ein 'diskreter deterministischer mehrstufiger Entscheidungsprozeß' (wir werden uns hier auf diese Sorte beschränken, obwohl die Verallgemeinerung keine Schwierigkeiten bereitet) ist ein Prozeß, bei dem endlich oder höchstens abzählbar viele Transformationen (auch Entscheidungen genannt) den Prozeßverlauf bestimmen. Der Prozeß startet im Zustand p_1, für $t > 1$ wird der Zustand p_t durch eine deterministische Entscheidung q_{t-1} (aus einer gegebenen Menge möglicher Entscheidungen) und p_{t-1} bestimmt:

$$p_t = T(p_{t-1}, q_{t-1}).$$

Zu maximieren sei für einen n-stufigen Prozeß eine Kostenfunktion $U(q_1, q_2, \ldots, q_n)$.

Unter der Voraussetzung, daß der Prozeß die Markov-Eigenschaft besitzt (letztere wird unten nochmals in Erinnerung gerufen), braucht man nicht alle möglichen Kombinationen von Aktionen durchzuprobieren, sondern man kann sich auf relativ wenige Kombinationen beschränken.

Ein n-stufiger Prozeß hat die Markov-Eigenschaft, wenn nach k Entscheidungen der Effekt der ausstehenden $n - k$ Entscheidungen auf die Kostenfunktion U nur noch von dem Zustand des Systems nach der k-ten Entscheidung und den folgenden Entscheidungen abhängt.

Hat U z.B. die Form

$$U = u(p_1, q_1) + u(p_2, q_2) + \ldots + u(p_n, q_n),$$

so ist die Markov-Eigenschaft gewährleistet.

Viele deterministische Prozesse sind Markov-Prozesse. Ein Beispiel sind nahezu alle Brettspiele: Es kommt nicht darauf an, wie man zu einem Spielzustand gekommen ist. Alle Information, die man zum Weiterspielen braucht, ist in dem gegenwärtigen Zustand enthalten.

Viele Prozesse sind jedoch *keine* Markov-Prozesse. Insbesondere für biologische Systeme typische Handlungsweisen hängen oft nicht nur vom gegenwärtig wahrnehmbaren Zustand der Umgebung ab, sondern auch von vergangenen Perzeptionen.

Für Markov-Prozesse spielt nun Bellmans *Optimalitätsprinzip* eine entscheidende Rolle:

Eine optimale Folge von sukzessiven Entscheidungen (eine optimale Strategie*) hat die Eigenschaft, daß unabhängig vom Initialzustand des Prozesses und der ersten Entscheidung die noch ausstehenden Entscheidungen eine optimale Strategie bezüglich des Zustands darstellen, der aus der ersten Entscheidung resultiert.*

Das Optimalitätsprinzip führt für Markov-Prozesse zum Ansatz der *dynamischen Programmierung* [6]. Rekursiv definiert man sich eine Menge von Funktionen wie folgt:

$$f_1(p_1) = Max_{q_1} u(p_1, q_1)$$

und

$$f_t(p_1) = Max_{q_1}(u(p_1, q_1) + f_{t-1}(T(p_1, q_1)))$$

Durch das Optimalitätsprinzip ist der Beitrag der letzten $n-1$ Schritte gleich $f_{n-1}(T(p_1, q_1))$.

Die wesentliche Auswirkung der dynamischen Programmierung ist: Die Auswahloperation eines Punktes im pn-dimensionalen Raum wird reduziert auf n Auswahloperationen im p-dimensionalen Raum (hierbei ist p die Zahl der Dimensionen, die zur Beschreibung der Prozeßzustände notwendig sind, im allgemeinen kann man sich also, wenn auch nicht notwendigerweise auf einen 1-dimensionalen, so doch zumindest auf einen *nieder*dimensionalen Raum beschränken). Damit erspart man sich bei der Suche oft sehr viele von vornherein auszuschließende Sackgassen.

In nichtstationären Umgebungen kann die Kunst bei der dynamischen Programmierung darin gesehen werden, eine Funktion J anzugeben, deren Minimierung zum Zeitpunkt t schon bedeutet, diejenige Aktion für die Transformation von R_t nach R_{t+1} zu wählen, die auch für die Minimierung von U notwendig ist.

Ein Problem bei der dynamischen Programmierung ist, daß trotz der gewaltigen Einsparungen im Vergleich zur erschöpfenden Suche die Anzahl der Berechnungen immer noch exponentiell mit der Anzahl der Komponenten der Prozeßzustandsbeschreibungen steigt.

Genetische Algorithmen

Eine sehr unterschiedliche Art gesteuerter Suche erfreut sich in den letzten Jahren wachsender Beliebtheit. Genetische Algorithmen durchforsten den Suchraum auf eine Weise, die von der biologischen Evolution inspiriert ist.

In der Hollandschen Version [16] repräsentiert eine Menge (genannt der *'Pool'*) von Bitsequenzen fester Länge n sogenannte *Genotypen*. Jeder Genotyp entspricht einem durch ihn definierten Phänotyp. Jeder Phänotyp wird an einer zu lösenden Aufgabe getestet, und sein Genotyp erhält abhängig

von der Güte der Performanz eine reelle Zahl, die sogenannte *Fitness*, zugesprochen.

Je höher die *Fitness* eines Genotyps, desto höher ist die Wahrscheinlichkeit, daß er nun am 'Fortpflanzungsprozeß' teilnehmen darf. Dieser läuft wie folgt ab: Zwei Genotypen (erfolgreiche werden wie gesagt bevorzugt) tauschen 'genetisches Material' in Form von Bitsubsequenzen aus. Dabei werden einfach eine Anzahl aufeinanderfolgender Bits in der einen Sequenz ersetzt durch die entsprechenden Bits der anderen. Gelegentlich kommt auch eine 'Mutation' vor, wobei zufällig ausgewählte Bits durch ihr Komplement ersetzt werden. Mit den Mutationen wird ein exploratives Element in den Suchprozeß eingeführt.

Die auf diese Weise neu konstruierten Genotypen werden auf dem Umweg über ihre zugehörigen Phänotypen getestet. Solche mit hoher *Fitness* verdrängen nun aus dem *Pool* solche mit niedriger *Fitness*. Verschiedenste Verdrängungsstrategien können dabei zur Anwendung kommen.

Das Verfahren wird iteriert, bis ein Genotyp entstanden ist, der genügend hohe *Fitness* aufweist, oder bis ein sonstiges Abbruchkriterium erfüllt ist.

Auf neuronale Netze mit binären Gewichten läßt sich das Verfahren ohne große Umschweife anwenden. Auch falls jedes Gewicht nur endlich viele verschiedene potentielle Werte annehmen kann, findet man schnell anwendbare Varianten genetischer Algorithmen. Ein entsprechendes Verfahren wird im folgenden als *neurogenetischer* Algorithmus bezeichnet.

Trotz der Allgemeinheit neurogenetischer Algorithmen (sie sind ja nicht abhängig von Netztopologie oder Gradienteninformation) hat man bisher merkwürdigerweise nur azyklische Netze damit trainiert, und das lediglich anhand von im Prinzip einfachen Problemen, die der allgemeinen Natur des genetischen Algorithmus nicht gerecht werden. Das liegt vor allem daran, daß sich viele Forscher über die fundamentalen Unterschiede der verschiedenen Arten des Lernens noch nicht im klaren sind, und somit auch die damit verbundenen Schwierigkeiten nicht zu würdigen wissen. Es ist bestimmt unklug, mit einem genetischen Algorithmus in direkte Konkurrenz zu den viel 'informierteren' Gradientenabstiegsverfahren treten zu wollen (z.B. bei Klassifikationsaufgaben) [30]. Geschickter wäre es, sich mit neurogenetischen Algorithmen an Problemen zu versuchen, bei denen ein Verfahren wie Back-Propagation von vornherein keine Chance hat. Das sind zum Beispiel Probleme, bei denen nur uninformierte Reinforcementsignale zur Auswertung zur Verfügung stehen. Gegenwärtig wird in der Forschungsgruppe TIGKI der TUM im Rahmen eines Fortgeschrittenenpraktikums die Anwendbarkeit neurogenetischer Algorithmen auf vollständig rekurrente Netzwerke für die adaptive zeitabhängige Steuerung einfacher physikalischer Prozesse untersucht.

Etwas allgemeine Kritik neurogenetischer Algorithmen basierend auf

dem Hollandschen Prinzip kann man allerdings jetzt schon formulieren:

Zum einen sprechen die experimentellen Befunde dafür, daß genetische Algorithmen nicht gut funktionieren, wenn sehr viele Parameter zu adjustieren sind. Die Konsequenz für die oben beschriebenen neurogenetischen Algorithmen wäre, daß sie nur für sehr kleine Netze brauchbar wären. Ein Weg, die Anzahl der Parameter zu reduzieren, ist, von einem gegebenen neuronalen Algorithmus auszugehen und nur noch einige wenige Netzeigenschaften wie z.B. die Netzwerktopologie der Auswahl durch einen genetischen Algorithmus zu überlassen. Damit schränkt man sich natürlich auf solche Aufgaben ein, die der vorgegebene neuronale Algorithmus im Prinzip zu lösen imstande ist.

Zum anderen sind neurogenetische Algorithmen nicht einmal eingeschränkt lokal in Raum und Zeit. Jeder Genotyp entspricht einem vollständigen Netzwerk. Steigt die Netzwerkgröße, braucht man auch mehr Genotypen, um den Gesamtraum aller Gewichtskombinationen hinreichend abdecken zu können.

Schließlich (und mit dem letzten Punkt zusammenhängend) betonen genetische Algorithmen den Wettbewerb und vernachlässigen die Kooperation. Zwar kooperieren Bitsequenzen während der Lernphase durch gegenseitigen Austausch von Information. Das *Ziel* jedoch ist das *universelle Genie*, nämlich diejenige *einzelne* Sequenz, die der Lösung der Aufgabe entspricht. Zumindest die Hollandschen genetischen Algorithmen zielen in keiner Weise auf irgendeine Form der Arbeitsteilung *nach* der Lernphase.

Es wird schon einen Grund für die biologischen genetischen Algorithmen gegeben haben, irgendwann einmal die neuronalen Lernalgorithmen zu erfinden ...

Die Eimerkette für regelbasierte Systeme

Konzeptionelle Vorteile gegenüber den genetischen Algorithmen bietet Hollands *bucket brigade* Prinzip für regelbasierte Systeme. Es geht hierbei darum, viele einfache regelähnliche Entitäten (sogenannte *Klassifikatoren*) in einer Weise zur Kooperation zu veranlassen, die zum erfolgreichen Verhalten des Gesamtsystems führt.

Botschaften in Form von Bitsequenzen der Länge n können sowohl von der Umgebung eines lernenden Systems als auch von zu dem System gehörigen Klassifikatoren auf der globalen *Botschaftentafel* plaziert werden. Ein Klassifikator selbst ist eine zweigeteilte Zeichensequenz bestehend aus einer 'Bedingungssequenz' und einer 'Aktionssequenz'. Beide Teile sind Sequenzen aus $\{0, 1, \_\}^n$, wobei die Semantik des '_' im Falle des Auftretens in einer Bedingungssequenz besagt: 'Ignoriere mich!'.

Eine positive reelle Variable ist mit jedem Klassifikator assoziiert und zeigt seine gegenwärtige *'Stärke'* an.

Im Laufe eines Zyklus (ein Zyklus entspricht einem Zeitschritt) werden alle Botschaften auf der Botschaftentafel mit den Bedingungssequenzen aller Klassifikatoren verglichen. Jeder Klassifikator, dessen Bedingungsteil mit wenigstens einer Botschaft übereinstimmt, berechnet seinen *'Wetteinsatz'*, indem er seine *Spezifizität* (die Anzahl der Bits in seinem Bedingungsteil, die keine '_'s sind) mit dem Produkt seiner Stärke und einer kleinen Konstanten multipliziert.

Die höchstbietenden Klassifikatoren dürfen für den nächsten Zyklus nun ihren *Aktionsteil* auf die Botschaftentafel schreiben. (Tritt dabei das '_' in einer Aktionssequenz auf, so bewirkt es eine Durchreicheoperation: Das entsprechende Bit der 'triggernden' Botschaft wird übernommen.)

Für das Schreibeprivileg müssen die Gewinner allerdings mit ihrem *Wetteinsatz* bezahlen, welcher unter denjenigen Klassifikatoren verteilt wird, die während des letzten Zyklus überhaupt erst die Vorraussetzungen für die Schreiboperationen der Gewinner schufen. Jeder in einer bestimmten Situation 'aktive' Klassifikator wird also schwächer werden, wenn er seine Verluste im nächsten Zyklus nicht wieder ausgleichen oder gar mehr als wettmachen kann. So wie bei einer Eimerkette zum Feuerlöschen viele Eimer gleichzeitig von einer Station zur nächsten weitertransportiert werden, so werden bei Hollands Algorithmus *Klassifikatorenstärken* 'durchgereicht'. Daher der Name *'Eimerkette'*.

Bestimmte Botschaften auf der Botschaftentafel werden von der Peripherie als Anweisung interpretiert, eine bestimmte Operation in der Umgebung auszuführen. Manche dieser Aktionen mögen von einem externen Kritiker als brauchbar angesehen werden. Der Kritiker verteilt in solchen Fällen eine Belohnung (Auszahlung) an die gegenwärtig aktiven Klassifikatoren, deren Stärke dadurch wächst.

Die zentrale Idee des ganzen Verfahrens ist: 'Versteckte' Klassifikatoren, die zwar im Augenblick der externen Auszahlung nicht aktiv sind, die jedoch eine wichtige Rolle spielten, um überhaupt erst einmal die Vorbedingungen für die direkt belohnten Klassifikatoren zu schaffen, partizipieren am Lernprozeß durch ihre Einbindung in Eimerketten. Der Erfolg eines zu einem gegebenen Zeitpunkt aktiven Klassifikators hängt damit rekursiv von den Erfolgen seiner Nachfolger ab. Der externe Kritiker beendet die Rekursion. (Als ein zusätzliches Mittel zur Verbesserung der Performanz führt Holland genetische Algorithmen für die Konstruktion neuer Klassifikatoren ein.)

Die Kommunikation über die zentrale Botschaftentafel sowie der globale Wettbewerb aller Klassifikatoren machen das Verfahren weder lokal im Raum noch in der Zeit. Dennoch: Einer der Beiträge dieser Dissertation ist durch das Eimerkettenprinzip inspiriert und wendet es auf neuronale Netze

an. Die Motivation dabei ist, *den ersten wirklich lokalen* Lernalgorithmus für neuronale Netze zu schaffen.

3.1.2 Neuronale Ansätze

Die im letzten Abschnitt beschriebenen Konzepte verfügen zwar über ein Potential zur Anwendung auf neuronale Netze, sind jedoch allgemein genug, um auf neuronale Netze verzichten zu können. Der gegenwärtige Abschnitt behandelt Verfahren, deren *Voraussetzung* ein netzähnliches Gebilde ist.

Keine Rückkopplung

Stochastische Lernautomaten sind abstrakte Maschinen, die ausgehend von einer Wahrscheinlichkeitsverteilung möglicher Aktionen Ausgaben produzieren. *Keine* Eingaben kommen von der Umgebung. Abhängig von der Bewertung eines evaluativen Kritikers werden bestimmte Aktionen jeweils wahrscheinlicher oder unwahrscheinlicher gemacht. Für diesen Zweck eignen sich viele einfache Lernalgorithmen, unter anderem die vielgestaltigen *Belohnungs-/Bestrafungsalgorithmen* (*reward-penalty-algorithms*). Wird eine spezifische Aktion ausgewählt und als erfolgreich bewertet, so soll ihre Auswahlwahrscheinlichkeit auf Kosten der Auswahlwahrscheinlichkeiten anderer Aktionen erhöht werden. Führt sie jedoch zum Mißerfolg, so soll ihre Auswahlwahrscheinlichkeit in Zukunft niedriger sein. Es gibt umfassende mathematische Literatur zu solchen stochastischen Lernautomaten (siehe [33]).

Assoziative Stochastische Lernautomaten (der Begriff stammt von Williams [78]) sind für uns wesentlich interessanter, da sie zumindest schon einmal Reaktionen auf Eingaben von der Umgebung erlauben. Von den assoziativen stochastischen Lernautomaten interessiert uns zunächst die *quasilineare stochastische Einheit*. Die Wahrscheinlichkeit, daß solch eine Einheit (im folgenden auch 'Knoten' genannt) mit der Nummer i die Ausgabe y_i produziert, hängt von einer Wahrscheinlichkeitsverteilung ab, deren Dichtefunktion nur einen Parameter $p_i = f_i(\sum_j w_{ij} x_j^i)$ besitzt. Dabei ist w_{ij} die j-te Komponente des Gewichtsvektors w_i des Knotens i, x_j^i die j-te Komponente seines Eingabevektors x^i, und f_i eine mit i assoziierte sigmoide Funktion. Als Spezialfall erwähnen wir den *quasilinearen Bernoulliknoten* i: Für ihn ist $y_i \in \{0,1\}$, mit $P(y_i = 0 \mid w_i, x^i) = 1 - p_i$ und $P(y_i = 1 \mid w_i, x^i) = p_i$.

Barto und Anandan [3] nennen Lernaufgaben, bei denen es dem lernenden System möglich ist, Aktionen mit kontextueller Zusatzinformation (gewonnen aus Eingaben von der Umgebung) zu assoziieren, *assoziative Reinforcement Lernaufgaben* (*associative reinforcement learning tasks*). Barto

und Anandan entwarfen den *Assoziativen Belohnungs-/Bestrafungsalgorithmus* (*associative reward-penalty algorithm*) für den Fall binären Reinforcements $r \in \{0, 1\}$:

$$\triangle w_{ij} = \alpha(y_i - p_i)x_j^i \ \ falls \ r = 1, \ \ und \ \triangle w_{ij} = \alpha\lambda(1 - y_i - p_i)x_j^i \ \ falls \ r = 0.$$

Dabei ist $\triangle w_{ij}$ wie immer die Gewichtsänderung von w_{ij}, α eine Lernrate, und $0 \leq \lambda \leq 1$. Barto und Jordan haben auch eine Verallgemeinerung für kontinuierliches Reinforcement vorgeschlagen [4].

Williams setzte mehrere quasilineare Knoten zu einem azyklischen Netzwerk zusammen. Eine Aktivationsausbreitungsphase in solch einem Netz läuft analog zu der Aktivationsausbreitungsphase in einem Back-Propagation Netz ab. Der wesentliche Unterschied besteht in der stochastischen Natur der Aktivierungsfunktionen.

Das Reinforcementsignal r ist wieder ein - diesmal allen Knoten zugänglicher - skalarer Wert. Vorausgesetzt wird nun, daß für alle $\xi \in Y_i$ (der Menge aller möglichen Ausgaben von i)

$$\frac{\partial P\{y_i = \xi \mid w_i, x^i\}}{\partial w_{ij}}$$

existiert, und daß die Gewichte der Änderungsregel

$$\triangle w_{ij} = \alpha_{ij}(r - b_{ij})e_{ij}$$

gehorchen. Dabei ist α_{ij} eine positive Lernrate, b_{ij} ein *Offset*, welches für gegebenes w und x^i unabhängig von y_i ist, und e_{ij} die *charakteristische Eligibilität*

$$e_{ij} := \frac{\partial ln \ P\{y_i = \xi \mid w_i, x^i\}}{\partial w_{ij}}.$$

Unter diesen Voraussetzungen bewies Williams eine interessante sich auf die Performanzverbesserung solcher Netzwerke beziehende Aussage [78]: Das innere Produkt

$$\frac{\partial E\{r \mid w\}}{\partial w} E\{\triangle w \mid w\}$$

ist positiv, solange der zweite Faktor nicht Null ist.

Was bedeutet dieses Resultat? Es zeigt, daß man für eine große Klasse von Lernalgorithmen *im Mittel erwarten kann, daß sich die Gewichte in Richtung des Gradienten des Erwartungswerts des Reinforcements ändern.*

Das ist ein sehr allgemeines und sehr wünschenswertes Ergebnis. Allerdings sind die von Williams so getauften 'REINFORCE'-Algorithmen (*'REward Increment = Nonnegative Factor * Offset Reinforcement * Characteristic Eligibility'*) nur dann zweckmäßig, wenn die Umgebung außer Reinforcementsignalen keine Zusatzinformation über wünschenswerte Ausgaben bereitstellt. Zwar sind REINFORCE-Algorithmen bei weitem allgemeiner als etwa Back-Propagation. Wo jedoch beide Paradigmen anwendbar sind, zieht man überwachte Gradientenabstiegsverfahren wegen ihrer erfahrungsgemäß weit schnelleren Konvergenz vor.

Die Lernregel für die REINFORCE-Algorithmen erlaubt eingeschränkte Lokalität in Zeit und Raum: Kein Analogon zur Fehlerpropagierungsphase beim überwachten Lernen ist notwendig. Jeder Knoten erhält nach der Aktivationsausbreitung dasselbe Reinforcementsignal. Allerdings müssen auch bei REINFORCE-Algorithmen Aktivationsausbreitung und Gewichtsänderung durch eine globale Instanz zeitlich getrennt werden.

Interne Rückkopplung

Durch Minskys und Paperts in Kapitel 2 beschriebenes Prinzip des *'unfolding in time'* lassen sich REINFORCE-Algorithmen auf zyklische Netzwerke ohne externe Rückkopplung erweitern. Reinforcement $r(t)$ kann dabei zu verschiedenen Zeitpunkten eines Trainingsintervalls vergeben werden. Die Lernregel für den erweiterten Fall definiert in leichter Abwandlung des azyklischen Falles für jedes Gewicht w_{ij} eine Gewichtsänderung $\triangle w_{ij}$:

$$\triangle w_{ij} = \alpha_{ij}(\sum_t r(t) - b_{ij}) \sum_t e_{ij}(t)$$

mit

$$e_{ij}(t) := \frac{\partial ln\ P\{y_i(t) = \xi \mid w_i, x^i(t-1)\}}{\partial w_{ij}},$$

wobei $y_i(t)$ die Aktivation und $x^i(t)$ den Eingabevektor von i abhängig vom Zeitpunkt t darstellt, und t über alle Zeitpunkte außer dem ersten des Trainingsintervalls rangiert.

Das zentrale Theorem besagt nun, daß das innere Produkt

$$\frac{\partial E\{\sum_t r(t) \mid w\}}{\partial w} E\{\triangle w \mid w\}$$

positiv ist, solange der zweite Faktor nicht Null ist.

Damit haben wir den bisher allgemeinsten Lernalgorithmus für neuronale Netze kennengelernt. Er ist (wenigstens im Prinzip) tauglich für

Reinforcement-Lernen mit interner Rückkopplung. Auch diese *erweiterten* REINFORCE-Algorithmen können *schwach* lokal in Raum und Zeit implementiert werden. Die *starke* Lokalität ist jedoch nicht gegeben, da wieder eine externe Instanz über die Trennung zwischen Aktivationsausbreitung und Gewichtsänderung wachen muß.

Wir betonen hier noch einmal, daß Williams' Resultate *nicht* den allgemeineren Fall der *externen* Rückkopplung mit einbeziehen.

3.2 Modellbildende Verfahren

Die im letzten Abschnitt beschriebenen Verfahren verfolgen mehr oder weniger raffinierte Strategien, um Aktionen solange auszuprobieren, bis sich der Erfolg einstellt.

Wenn man in einer dynamischen Umgebung bestimmte Ziele erreichen will, so lohnt es sich oft, nicht nur Probierverfahren anzuwenden, sondern sich auch ein *Modell* der Umwelt oder bestimmter Aspekte der Umwelt zu konstruieren. *Versteht man bis zu einem gewissen Grade, wie sich die Umwelt verhält, so kann man im allgemeinen auf einem viel direkteren Wege seine Ziele verfolgen als ohne ein derartiges Verständnis.*

Es gibt einige erst vor kurzem in näheren Augenschein genommene Ansätze, adaptive 'neuronale Umgebungsmodelle' für zielgerichtetes Lernen einzusetzen. Die beiden bedeutsamsten Ansätze sind die Systemidentifikationsansätze und die adaptiven Kritiker. Auf beide gehen wir im folgenden ein. Beide sind relevant für Kapitel 5 bzw. 6 dieser Arbeit.

3.2.1 Der Systemidentifikationsansatz

Der Systemidentifikationsansatz erfordert, ein sogenanntes *Modellnetzwerk* daraufhin zu trainieren, bestimmte Eigenschaften der externen Umwelt vollständig zu simulieren. Nach Abschluß der Trainingsphase des Modellnetzwerkes (oder auch schon vorher) *identifiziert* man das Modellnetzwerk mit der Umgebung und verwendet es, um Gradienteninformation für das eigentliche Hauptnetzwerk (im folgenden auch das *Steuernetzwerk* genannt) zu berechnen. Wie wir es schon aus früheren Abschnitten gewohnt sind, werden wir auch im folgenden wieder zwischen verschieden komplexen Unterfällen des Systemidentifikationsprinzips unterscheiden.

Keine Rückkopplung

1. Als einer der ersten hat Munro das Systemidentifikationsprinzip im Kontext Reinforcement-lernender azyklischer Netze beschrieben [31]. Munro

verwendet ein BP-Netzwerk C mit Gewichtsvektor W_C , um für gegebene Eingabemuster Ausgaben zu produzieren. Im Kontrast zum normalen BP sind die gewünschten Ausgaben jedoch nicht bekannt, die einzige Information über die Güte der Ausgabe wird (wie beim Reinforcement-Lernen eben üblich) durch einen evaluativen Kritiker gegeben, der nur einen skalaren Wert $r \in [0 \ldots 1]$ zur Verfügung stellt. Das Ziel des Hauptnetzwerkes ist, für jedes Eingabemuster dasjenige Ausgabemuster zu produzieren, welches r maximiert. Um nun die unbekannte 'Lücke' zwischen Netzwerkausgaben und korrespondierender externer Bewertung zu schließen, trainiert Munro ein zweites Netzwerk M (eben das Modellnetzwerk) mit Gewichtsvektor W_M darauf, für gegebene Ein- und Ausgaben von C das entsprechende Reinforcement vorherzusagen. Die Dimension von M's Eingabevektor ist also stets gleich der Summe der Dimensionen der Ein- und Ausgabevektoren von C. M's Ausgabelage ist 1-dimensional.

Das Modellnetzwerk wird in einer ersten Phase anhand zufällig ausgewählter Kombinationen von Ein- und Ausgaben des Hauptnetzwerkes und der von dem externen Kritiker gelieferten zugehörigen Reinforcementsignale trainiert. Das BP-Verfahren dient dabei zur Berechnung von

$$\frac{\partial(r_p - x_p)}{\partial W_M}$$

für alle Eingabekombinationen p mit zugehörigen Bewertungen r_p und Modellausgaben x_p.

Sagt das Modellnetzwerk stets gut genug voraus, so werden seine Gewichte *eingefroren* und ändern sich von nun an nicht mehr. Nun findet der zweite Teil der Lernphase statt: Aus dem Modellnetzwerk und dem (noch untrainierten) Hauptnetzwerk wird (im wesentlichen durch Konkatenation) *ein* größeres Netzwerk G bestehend aus den beiden Teilen C und M konstruiert: M erhält seine Eingaben nun direkt von C's Ein- und Ausgabelagen.

Eingabemuster k werden nun bei G's Eingabelage (welche identisch ist mit C's Eingabelage) angelegt und produzieren G's skalare Ausgabe x_k. Der gewünschte Wert für x_k ist immer 1. Die entsprechenden Fehlersignale werden nun durch G nach hinten propagiert, sie durchlaufen dabei zuerst M und dann C. Anschließend ändern sich *ausschließlich* C's Gewichte. Das BP-Verfahren dient dabei also im wesentlichen zur Berechnung von

$$\frac{\partial(r_k - x_k)}{\partial W_C}$$

unter der Bedingung W_M. So wird das Modellnetzwerk seiner Bezeichnung gerecht.

Nur externe Rückkopplung

Widrow, der sich vor allem mit adaptiver Regelung beschäftigt, hat einen ganz analogen Ansatz vorgeschlagen. Ein nicht sehr wesentlicher Unterschied zu Munros Verfahren besteht darin, daß die zu modellierende Umwelteigenschaft mehrdimensional sein kann. Typische Beispiele für mehrdimensionale Umwelteigenschaften sind beispielsweise durch Zustandsvariablen eines physikalischen Prozesses gegeben.

Ein wesentlicher Unterschied zwischen Widrows Arbeiten und dem Ansatz von Munro besteht in der Möglichkeit der externen (nicht aber der internen) Rückkopplung [34]. Der Grundansatz ist der folgende: Ein azyklisches Modellnetzwerk M wird mit Hilfe von zufällig ausgewählten Trainingsbeispielen und BP darauf trainiert, aus Ein- und Ausgaben eines den gesamten Umgebungszustand wahrnehmenden Steuernetzwerkes C Voraussagen über C's Eingaben zum nächsten Zeitpunkt zu machen. Damit modelliert M also abhängig vom Umgebungskontext die Effekte möglicher Aktionen von C. Das funktioniert natürlich nur dann richtig, wenn der wahrnehmbare Zustand der Umgebung zum Zeitpunkt $t+1$ nicht von Zeitpunkten $s < t$ abhängt, wenn also die sichtbare Umgebung der Markov-Eigenschaft genügt.

Nach dieser ersten Lernphase werden M's Gewichte eingefroren wie gehabt, und die Lernphase für das ebenfalls azyklische Steuernetzwerk C beginnt. Zwar wird C nie in der Lage sein, anders als in statischer Weise auf bestimmte Eingaben zu reagieren. Damit ist die von C implementierbare Dynamik von der trivialen Art. In Markov-Umgebungen kann die Kombination von C und M jedoch zur Lösung ziemlich schwieriger zeitlicher *'credit-assignment'*-Probleme eingesetzt werden. Mit Hilfe des *'unfolding in time'*-Prinzips kann das rekurrente Netzwerk G, das durch Identifizierung von M's Ausgabelage mit C's Eingabelage und der Identifizierung von C's Ausgabelage mit M's Eingabelage entsteht, dazu veranlaßt werden, zielgerichtete Trajektorien im Ausgaberaum von C zu erzeugen. Damit entfällt der Lehrer, der zu verschiedenen Zeitpunkten anzeigt, *welche* Ausgabeaktion die externe Umgebung manipulieren soll. Die zu lernenden Trajektorien brauchen *nicht* vordefiniert zu sein, lediglich gewünschte Zielzustände z.B. am Ende einer Trajektorie müssen bekannt sein.

Zusätzliche Beschränkungen für Ausgabeknoten

Jordan hat das Systemidentifikationsprinzip ebenfalls im Kontext von Steuerungsaufgaben verwendet [21]. In Jordans Anwendungen gibt ein Lehrer für jeden Zeitpunkt einen zu erreichenden Zustand der Umgebung in *'Zielknoten'* vor (z.B. die gewünschten Positionen eines Roboterarms). Damit

entfällt zumindest einmal die zeitliche Komponente des *'credit-assignment'* Problems.

Ein in einer separaten Phase trainiertes Modellnetzwerk sagt wieder aus den Ausgaben des Steuernetzwerkes Effekte auf die Zielknoten voraus. Das Modellnetzwerk hilft dem Steuernetzwerk, die richtigen Ausgaben zu generieren (wie gehabt). Jordan betont nun die Möglichkeit der Einführung zusätzlicher Einschränkungen für die Ausgabeknoten des Steuerungsnetzes. Ein Roboterarm, der zu vier aufeinanderfolgenden Zeitpunkten vier verschiedene Punkte im zweidimensionalen Raum erreichen soll, verfügt dank seiner Bewegungsmöglichkeiten über mehr Freiheitsgrade, als zur Lösung der Aufgabe notwendig wären. Wie kann man die zusätzlichen Freiheitsgrade sinnvoll ausnützen? Man konstruiert z.B. eine abgewandelte Fehlerfunktion, die zusätzliche Terme beinhaltet, welche die 'Glattheit' der Bewegung messen. Das geht ganz einfach: Man addiert zur ursprünglichen Fehlerfunktion Differenzen sukzessiver Ausgaben des Steuernetzwerkes. Die Minimierung der neuen Fehlerfunktion erzwingt 'glatte' Trajektorien, nämlich solche, die im Rahmen der vorgegebenen Aufgabe den Gesamtbewegungsaufwand so weit wie möglich reduzieren.

Um die Entscheidung des Steuernetzes zur Zeit t von den Ausgaben zur Zeit $t-1$ abhängig zu machen, benutzt Jordan wie bei seinem schon im Kapitel zum überwachten Lernen beschriebenen Verfahren eine zweigeteilte Eingabelage. Neben den normalen Eingabeknoten gibt es noch die *'Zustandsknoten'*, in die die Aktivationen der Ausgabeknoten des Steuernetzes vom jeweils letzten Zeitschritt kopiert werden. Der resultierende Effekt gleicht dem, den man mit entsprechenden rekurrenten starren Verbindungen mit jeweiligem Gewicht 1 erreichen würde. Auch von den 'Zielknoten' führen Verbindungen zurück auf die Zustandsknoten.

Eingeschränkte externe und interne Rückkopplung.

Robinson und Fallside haben Munros Ansatz zum R-Lernen wesentlich erweitert [41][39]. Sowohl das Modellnetzwerk M als auch das Steuernetzwerk C können im Prinzip voll rückgekoppelt sein. Das Modellnetzwerk sagt zu einem gegebenen Zeitpunkt t in seiner eindimensionalen Ausgabe das Reinforcement zum jeweils nächsten Zeitpunkt $t+1$ voraus. Durch M's Rekurrenz können die Voraussagen auf allen vergangenen Ein- und Ausgaben des ebenfalls rekurrenten Steuernetzwerkes basieren.

Robinson wendet sich als erster dem Problem des parallelen Lernens von C und M zu. Um aus *zwei* Netzwerken *eines* zu machen, addiert er M's und C's Fehlerterme. Dabei nimmt er in Kauf, daß die Fehlersignale für *beide* Netzwerke weniger Information tragen: Fehlersignale für das Steuernetz werden mit solchen für das Modellnetz gemixt, und umgekehrt.

In den Experimenten seiner Ph.D. Arbeit [39] wendet Robinson sein Verfahren mit mäßigem Erfolg auf ein einfaches Brettspiel (*'Tic-Tac-Toe'*) an. Dabei benutzt er das *'unfolding in time'*-Prinzip, welches bekanntermaßen zu einem Algorithmus führt, der nicht lokal in der Zeit ist.

Man beachte, daß für komplexe (nicht Markov-mäßige) Umgebungen auch das Robinsonsche Modellnetzwerk nicht hinreichend ist. In Kapitel 5 werden wir die Arbeiten von Munro, Widrow, Jordan, Robinson und Fallside auf den allgemeinen Fall erweitern. Das resultierende Verfahren wird auf Umgebungen anwendbar sein, die nicht vom Markov-Typ sind, es wird lokal in der Zeit sein, es wird ausschließlich auf reinforcementartige Lerninformation (gewonnen aus einer beliebigen Zahl von 'Schmerzzellen' und 'Lustzellen') angewiesen sein, und es wird ein Modell *aller* durch die externe Dynamik verursachten Eingaben konstruieren, um damit vollständiges *'credit-assignment'* in die Vergangenheit zu gewährleisten.

3.2.2 Adaptive Kritiker

Der Systemidentifikationsansatz versucht zunächst nur, ausgehend von einem gegebenen Zeitpunkt, Eigenschaften der Umgebung zum nächsten Zeitpunkt durch 'Simulation' vorherzusagen. Im Gegensatz dazu versuchen *adaptive Kritiker* nicht, die Umwelt zu simulieren. Vielmehr werden sie mit geschickt ausgedachten Verfahren darauf trainiert, aus einer unzulänglichen *externen* Bewertungsfunktion (welche sich oft nur durch zu seltenen Zeiten spärlich verteiltes Reinforcement ausdrückt) eine informiertere *interne* Bewertungsfunktion zu konstruieren. Es ist die *interne* Bewertungsfunktion, die zu Gewichtsänderungen in einem Steuernetzwerk führt.

Wir werden adaptive Kritiker aus der Perspektive zweier sehr verwandter Standpunkte begutachten: Da ist zum einen der Standpunkt der Reinforcementvergleichsalgorithmen basierend auf den Methoden der zeitlichen Differenzen, und zum anderen der Standpunkt der dynamischen heuristischen Programmierung.

Es sei angemerkt, daß adaptive Kritiker bisher (abgesehen vom Kapitel 6 dieser Arbeit) nur für den Fall der externen Rückkopplung studiert wurden.

Reinforcementvergleichsalgorithmen

Ohne es so zu nennen, wendete Samuel schon 1959 als erster ein *Reinforcementvergleichsverfahren* an [47]. Samuel schrieb ein Programm, welches lernte, 'Dame' zu spielen. Dabei verwendete er eine Vielzahl von Techniken, was es zunächst nicht leicht macht, das Wesentliche an seinem Lernverfahren zu sehen. (Unter anderem war Samuel der erste, der das heute weithin gebräuchliche *'alpha-beta-pruning'* zur planenden Vorausschau benutzte.)

Das Wesentliche an Samuels Lernverfahren läßt sich durch folgendes Prinzip beschreiben:

Findet ein lernendes System eine Transition von einem von ihm selbst als schlecht eingeschätzten Zustand in einen von ihm selbst als gut eingeschätzten Zustand, so sollte diese Transition in ähnlicher Situation künftig ermutigt werden. Außerdem sollte der früher als schlecht eingeschätzte Zustand von nun an eher als gut beurteilt werden.

Ganz analog gilt:

Erfährt ein lernendes System, daß eine bestimmte Aktion in einem von ihm selbst als gut eingeschätzten Zustand stets einen als schlecht eingeschätzten Zustand nach sich zieht, so sollte diese Aktion in ähnlicher Situation künftig vermieden werden. Außerdem sollte der früher als gut eingeschätzte Zustand von nun an eher als schlecht beurteilt werden.

Die durch diese Prinzipien implizierte Rekursion wird durch die Umgebung beendet: Manchmal gibt es eben Situationen, bei denen man objektiv *weiß*, ob sie 'schlecht' oder 'gut' sind.

Bei Spielen wie 'Dame' gibt es nur eine am Schluß des Spiels vergebene binäre Bewertung: 'Gut' ist der Sieg, 'schlecht' ist die Niederlage. Dem oben implizierten 'Beurteiler' oder *Kritiker* obliegt es, aus dieser uninformierten Bewertung eine *informiertere* zu machen.

Samuels Kritiker war adaptiv: Er bestand aus einem linearen Polynom, dessen Variablen nach jedem Zug durch bestimmte vorprogrammierte Spielzustandsanalysen gesetzt wurden, und dessen Koeffizienten im wesentlichen nach den obigen einfachen Prinzipien adjustiert wurden. Mit der Zeit lernte das System durch das 'Nachhintenschieben von Erwartungen über den Ausgang eines Spiels', schon frühzeitig (lange vor Ende des Spiels) 'in die Zukunft zu blicken' und die Prognosen für sofortiges Handeln in Form von Wahrscheinlichermachen oder Unwahrscheinlichermachen bestimmter Züge auszunützen.

Heute würde man sagen, daß Samuels Kritiker die Form eines Perzeptrons hatte. Es ist das Verdienst von Barto, Sutton, und Anderson, Samuels Prinzip in isolierter Form studiert zu haben [5]. Sie kombinierten zwei neuronenähnliche Elemente namens *'associative search element'* (ASE) und *'adaptive critic element'* (ACE) auf folgende Weise:

Von der Umgebung kommt zu jedem Zeitpunkt t ein Eingabevektor x_t sowohl für ASE als auch für ACE. x_t hat die Eigenschaft, daß alle seiner Komponenten gleich 0 sind bis auf eine, weche den Wert 1 hat und den sichtbaren Zustand der externen Umgebung repräsentiert. Die Anzahl der möglichen Zustände der Umgebung sollte die Dimensionalität des Eingabevektors also nicht übersteigen.

Die Ausgabe von ASE ist definiert als

$$y_t = f(w_t^T x_t) + \ noise_t,$$

wobei w_t ASE's Gewichtsvektor, f eine sigmoide Funktion oder eine Schwellenfunktion und $noise_t$ zufälliges Rauschen bedeutet. Die Gewichtsänderung wird definiert durch

$$\triangle w_t = \alpha \hat{r}_t x_t.$$

Dabei ist α eine positive Lernrate, und $\hat{r}_t$ das *interne Reinforcement* zur Zeit t. (In einer Erweiterung dieser Regel taucht auch noch ein *Eligibilitätsvektor* e_t auf, dieser erschwert allerdings nur den Blick auf das Wesentliche des Verfahrens.)

Der Trick ist nun die Berechnung des internen Reinforcements $\hat{r}_t$. Dafür ist ACE zuständig. ACE macht eine Voraussage P_t über das in Zukunft zu erwartende Reinforcement. Seine Ausgabe ist gegeben durch

$$P_t = v_t^T x_t,$$

wobei v_t AHC's Gewichtsvektor ist. Die Gewichtsänderung wird definiert durch

$$\triangle v_t = \beta(r_t + \gamma P_t - P_{t-1})x_t.$$

Dabei ist β eine positive Lernrate, r_t das *externe Reinforcement* zur Zeit t und $0 < \gamma \leq 1$ ein Schwundfaktor, von dem wir vorläufig annehmen, daß er gleich 1 ist. (In einer Erweiterung dieser Regel taucht auch noch eine exponentiell schwindende *Spur* vergangener Eingabevektoren auf, um die wir uns jetzt nicht kümmern wollen, da sie wiederum nur den Blick auf das Wesentliche des Lernverfahrens verstellt.)

Was AHC tut, ist eigentlich nichts anderes, als die TD(0)-Methode für den Fall kumulativen Reinforcements zu implementieren (siehe Kapitel 2, Abschnitt 4). Betrachtet man die Berechnung von

$$\hat{r}_t = r_t + \gamma P_t - P_{t-1},$$

so stellt man für $\gamma = 1$ fest, daß Samuels oben angegebenes Prinzip zur Anwendung kommt: Der Vergleich sukzessiver Einschätzungen des zu *erwartenden* Reinforcements führt zur Gewichtsänderung für das Steuerelement ASE. Eine Verallgemeinerung dieses Prinzips für unbegrenzt lange Zeitdauern wird durch $\gamma < 1$ ermöglicht (γ verhindert potentiell unendliche Summen).

Es muß erwähnt werden, daß es bisher keine allgemeine Theorie für das gleichzeitige Lernen von interagierenden AHC- und ASE-Elementen gibt. (Die existierende Theorie der TD-Methoden sagt nämlich zunächst nichts über Konvergenzeigenschaften von Reinforcementvergleichsalgorithmen aus.) So ist nicht auszuschließen, daß das oben beschriebene *on-line*-Verfahren für Instabilitäten anfällig ist. In jüngster Zeit zeigten allerdings Untersuchungen zu extrem simplifizierten Versionen des Ansatzes, daß zumindest unter bestimmten einschränkenden Bedingungen die Konvergenz beider parallel lernender Systeme garantiert werden kann [79].

Suttons Ph.-D.-Arbeit besteht im wesentlichen in der Isolierung des Prinzips der Reinforcementvergleichsverfahren und seiner Illustrierung durch ein Balancierproblem [66]. Seine weiterführenden Arbeiten über TD-Methoden kamen erst später und waren durch die Ergebnisse mit den Reinforcementvergleichsalgorithmen motiviert [67].

Anderson führte Suttons Arbeit über Reinforcementsvergleichsalgorithmen fort. Der Beitrag seiner Ph.-D.-Arbeit besteht im wesentlichen darin, die Perzeptron-ähnlichen Einheiten durch jeweils ein azyklisches Netzwerk zu ersetzen [2]. Andersons Kritiker ist ein BP-Netzwerk, welches in TD(0)-Manier auf 'selbstüberwachte' Weise die entsprechenden Gewichtsänderungen berechnet. Damit ist man nicht mehr durch den Zwang zur linearen Separabilität eingeschränkt. (Auch Watkins' Algorithmen sind mit Reinforcementsvergleichsalgorithmen eng verwandt [71].)

In Kapitel 6 werden wir Reinforcementvergleichsalgorithmen auf den allgemeinen Fall rekurrenter Netze erweitern. Auch wird eine Möglichkeit zur Implementierung *mehrdimensionaler* adaptiver Kritiker beschrieben werden. (Alle bisherigen adaptiven Kritiker machen ja nur eine eindimensionale Voraussage über eine *skalare* Größe. Damit verzichten sie auf Information, die man aus verschiedenen Arten von Reinforcement (oder 'Schmerz') ziehen könnte.)

Heuristische dynamische Programmierung

Werbos' *heuristische dynamische Programmierung* (HDP) [74] [73] [75] begreift sich als eine Erweiterung der adaptiven Kritiker.

Es wird wieder angenommen, daß der komplette Zustand R_t der Umgebung eines lernenden Systems zur Zeit t als Eingabevektor vorliegt. Gegeben sei weiterhin eine Nutzenfunktion U, welche Aussagen über die Güte von Aktionssequenzen (Sequenzen von Transformationen, die den Umweltzustand verändern) liefert.

Die Kunst bei der dynamischen Programmierung kann in nichtstationären Umgebungen (wie bereits früher ausgeführt) darin gesehen werden, eine Funktion J anzugeben, deren Minimierung zum Zeitpunkt t schon

bedeutet, diejenige Aktion für die Transformation von R_t nach R_{t+1} zu wählen, die auch für die Minimierung von U notwendig ist. Ein Problem bei der dynamischen Programmierung ist, daß trotz der gewaltigen Einsparungen im Vergleich zur erschöpfenden Suche die Anzahl der Berechnungen in der Regel immer noch exponentiell mit der Anzahl der Komponenten von R steigt.

Die heuristische dynamische Programmierung versucht nun, J nicht zu berechnen, sondern nur zu *schätzen*. Der Schätzer ist ein überwacht lernendes neuronales Netzwerk, typischerweise ein BP-Netz, und wird im folgenden in Analogie zu den Reinforcementvergleichsalgorithmen wieder der *Kritiker* genannt.

Die Nutzenfunktion U habe die Form

$$U = u(R_1) + u(R_2) + \ldots + u(R_m),$$

wobei m unendlich sein kann.

Die Überwachung ist nicht durch einen externen Lehrer gegeben, sondern durch sukzessive Ausgaben des Kritikers selbst: Die gewünschte Ausgabe zum Zeitpunkt t eines Trainingsintervalls ist gegeben durch

$$J(R_{t+1}) + u(R_t).$$

Dabei hängt J außer von der Umgebung nur von dem gegenwärtigen Gewichtsvektor w ab, der für die Dauer eines Trainingsintervalls allerdings konstant bleibt. J soll zu einem gegebenen Zeitpunkt ausdrücken, was zu erwarten ist, wenn man mit dem gegenwärtigen Steuernetzwerk weitermacht. Konventionelles Back-Propagation dient zur Berechnung der Gewichtsinkremente für alle Zeitpunkte. Erst am Ende des Trainingsintervalls wird jedoch die Gewichtsänderung wirklich ausgeführt.

Eigentlich gibt es bei der dynamischen heuristischen Programmierung keinen sehr wesentlichen Unterschied zu dem im letzten Abschnitt beschriebenen Ansatz. Ein kleinerer Unterschied zu Barto, Sutton, und Andersons Verfahren besteht in der Tatsache, daß die Gewichte nicht *on-line* geändert werden. Damit begibt man sich auf mathematisch sichereres Pflaster. Werbos konnte unter den gegebenen Bedingungen nachweisen, daß für ein einfaches Lernproblem (bei dem die 'idealen' Gewichte auch analytisch berechenbar sind), HDP zu den 'idealen' Gewichten hinkonvergiert [75].

Was macht man mit den Voraussagen, die J liefert? Sie können z.B. analog zu Suttons Arbeiten direkt zur Berechnung des *internen Reinforcements* herangezogen werden. Oder man könnte versuchen, ein drittes azyklisches 'Modellnetzwerk' die Umwelt simulieren zu lassen und Fehler vom Kritiker durch das Modellnetz in das Steuernetz zu propagieren. Damit würde man

als zusätzliche Komponente den Systemidentifikationsansatz ins Spiel bringen [73] [74][19]. Das Modellnetzwerk könnte (wie früher ausgeführt) dazu beitragen, aus immer noch relativ 'uninformierten' internen Reinforcementsignalen 'informiertere' zu machen. Vor allem für große Steuernetzwerke ist diese Alternative bedenkenswert.

Kapitel 4

Die neuronale Eimerkette

4.1 Einführung

Bislang haben wir gesehen, daß die meisten Lernalgorithmen für neuronale Netze in nichtstationären Umgebungen nicht einmal *eingeschränkt* lokal in Zeit und Raum sind. Weiterhin ist *keine* der bisher bekannten Prozeduren für zielgerichtetes Lernen mit 'versteckten Knoten' *vollständig* lokal. Am deutlichsten wird die Nichtlokalität bei überwachten Algorithmen für rekurrente Netzwerke. Aber auch schon die normale Back-Propagation-Prozedur für statische azyklische Netzwerke erfordert eine globale Instanz, um die aufeinanderfolgenden Berechnungen von Knotenaktivationen sukzessiver Netzlagen und die sich abwechselnden Aktivationsausbreitungs-, Fehlerpropagierungs- und Gewichtsänderungsphasen zu kontrollieren.

Soweit wir das heute beurteilen können, sind die Lernregeln in biologischen Systemen mit vielen 'versteckten Knoten' jedoch vollständig lokal in Raum und Zeit. Trotzdem (oder gerade deshalb?) sind biologische Systeme imstande, komplexe Lernprobleme raumzeitlicher Art zu lösen.

In diesem Kapitel wird im ersten wirklich originären Beitrag dieser Arbeit gezeigt, daß stark lokales zielgerichtetes Lernen mit 'versteckten Knoten' kein Widerspruch in sich ist: Wir treten nämlich einen konstruktiven Gegenbeweis an.

Wir konstruieren eine Klasse von Lernalgorithmen, die folgende Eigenschaften besitzen: *Alle* Knoten und *alle* Verbindungen führen zu *jedem* Zeitpunkt im wesentlichen dieselbe Operation aus. *Kein* Knoten und *keine* Verbindung kümmert sich zu *irgendeinem* Zeitpunkt darum, ob er/sie Teil eines zyklischen oder eines azyklischen Netzwerkes ist, wie etwa die globale Netzlagenstruktur aussieht, wie lange Sequenzen von Netzeingaben dauern, etc...

Zum Lernen ist lediglich reinforcement-artige Information notwendig. Aber auch Information überwachender Natur kann ohne zusätzlichen Aufwand in sinnvoller Weise mitverwendet werden. Gewichtsänderungen finden *ständig* statt und sind von der Aktivationsausbreitung nicht zeitlich getrennt.

Das den Algorithmen zugrundeliegende Schema ist inspiriert durch Hollands *'bucket brigade algorithm'* für regelbasierte Systeme, wie er im letzten Kapitel beschrieben wurde. Allerdings gibt es schon vom Grundkonzept her eine Reihe signifikanter Unterschiede zu Hollands Ansatz. Das Schema macht aus dem Netzwerk ein *dissipatives System*, durch welches ständig eine von einem externen Kritiker im Erfolgsfall spendierte *'Gewichtssubstanz'* fließt. Es wird permanent versucht, diesen Gewichtsstrom zu erhalten oder zu verstärken.

Es wird nicht behauptet, daß biologische Neuronen *tatsächlich* nach dem vorgestellten Schema funktionieren. Es wird auch nicht behauptet, daß Vertreter der vorgestellten Algorithmenklasse bestimmte Probleme etwa schneller als die konkurrenzfähigsten nicht-lokalen Algorithmen zu lösen imstande sind (in der Tat wurden einige Experimente durchgeführt, die eher auf das Gegenteil hinweisen und deutlich die Grenzen des Vefahrens aufzeigen). Es wird aber gezeigt: Die Grundidee des 'bucket brigade algorithmus' ist in sinnvoller Weise auf neuronale Netze transformierbar, und *räumlich und zeitlich lokales Lernen in Netzwerken mit versteckten Knoten ist möglich.*

Das Kapitel ist wie folgt gegliedert: Nach einer intuitiven Erklärung des Verfahrens werden die exakten Gleichungen für eine Version mit diskreter Zeit angegeben. Im Anschluß wird eine mögliche Erweiterung für kontinuierliche Zeit besprochen. Daraufhin beschreiben wir eine Reihe erfolgreicher Anwendungen auf nicht-lineare Probleme, sowie zwei Probleme, bei denen das Verfahren scheitert. Letztere motivieren die beiden nachfolgenden Kapitel. Schließlich werden einige Brücken zu weitläufig verwandten Konzepten aus dem Bereich massiv paralleler Lernsysteme geschlagen.

4.2 Der grundlegende Algorithmus

4.2.1 Zusammenfassung

Verbindungen zu Netzknoten, die im Moment des erfolgreichen Abschlusses einer extern gestellten Aufgabe aktiv sind, werden etwas gestärkt, sie erhalten "Gewichtssubstanz" von der Umgebung. Wie nehmen nun diejenigen Gewichte, die eventuell zu früheren Zeitpunkten in maßgeblicher Weise zum späteren Erfolg beitrugen, am Lernprozeß teil? Jede Verbindung, die irgendwann Aktivationsinformation von einem Netzknoten zum

nächsten leiten darf, muß dafür "zahlen" in Form von etwas Gewichtssubstanz, welche an diejenigen Verbindungen verteilt wird, die den Quellknoten der zahlenden Verbindung überhaupt erst aktiviert haben. Dieser durch lokalen Wettbewerb benachbarter Netzknoten unterstützte "Eimerketten"-Mechanismus etabliert rekursive Abhängigkeiten zwischen sukzessiv leitenden Verbindungen. Nur solche Eimerketten, die zum erfolgreichen Abschluß einer Aktionssequenz beitragen, können sich auf Dauer durchsetzen [50].

Das Schema wird als 'neuronale Eimerkette' und gelegentlich auch als 'A1' bezeichnet werden. Weiter unten werden wir sehen, daß es verschiedene Möglichkeiten gibt, Lehrinformation für die neuronale Eimerkette zur Verfügung zu stellen: $A1$ ist der Vater einer Klasse von Lernalgorithmen, die sich durch den Grad ihrer Überwachtheit unterscheiden.

4.2.2 Intuitive Erklärung

Wir wollen das Eimerkettenprinzip für regelbasierte Systeme (siehe letztes Kapitel) transformieren und so auf neuronale Netzwerke anwenden, daß der resultierende Lernalgorithmus lokal in Raum und Zeit ist.

Wir gehen aus von einem willkürlich gewählten (möglicherweise zyklischen) gerichteten Graphen, der uns die Netztopologie definiert. Jeder Knoten entspricht einer neuronenähnlichen konventionellen Prozessoreinheit. Zu jeder Kante gehört ein mit einem positiven Wert initialisiertes reelles Gewicht. Manche der Knoten dienen als Eingabeknoten, andere als Ausgabeknoten. Aktivierungen von Ausgabeknoten werden interpretiert als Steuersignale für Effektoren, die den Zustand der Umgebung beeinflussen können. Damit mag sich auch die Aktivierung der Eingabeknoten zum nächsten Zeitpunkt ändern. Also sind sowohl interne als auch externe Rückkopplung prinzipiell erlaubt (Abb. 4.1).

Zunächst brauchen wir ein Analogon zu dem Prozeß des 'Gewinnens' bei Klassifikatorensystemen.

Das Element des wechselseitigen Wettbewerbs kann bei neuronalen Netzen in traditioneller Art durch laterale Inhibition der Netzknoten eingeführt werden. Um statt *globalen* Wettbewerbs einen *lokalen* zu bekommen, partitionieren wir die Menge der Nicht-Eingabeknoten des Netzwerkes in disjunkte Untermengen, von nun an als WTA-Einheiten ('WTA' für *'winner-take-all'*) bezeichnet. Zu einem gegebenen Zeitpunkt hemmen sich alle Knoten einer WTA-Einheit gegenseitig proportional zur Stärke ihrer gegenwärtigen Eingabe von anderen Knoten. Zunächst nehmen wir an, daß nur der 'Sieger' selbst aktiv wird, und die 'Verlierer schweigen'. Eine WTA-Einheit mit n Knoten kann also nur n verschiedene Zustände haben. Jede WTA-Einheit muß demnach mindestens zwei Knoten enthalten, um überhaupt Sinn zu machen.

Abbildung 4.1: Von der Umgebung im Erfolgsfall vergebene Gewichtssubstanz fließt durch einen lernenden autonomen Agenten, der in einer veränderlichen manipulierbaren Umgebung lebt. Die Richtung des von den Perzeptoren ausgehenden Aktivationsstromes ist der Richtung des den Ausgabeeinheiten entspringenden Gewichtsstromes entgegengesetzt. Ausschließlich lokale Berechnungen dienen der Anpassung des Agenten. (Nähere Erläuterungen im Text.)

Zunächst nehmen wir an, daß alle Knoten dem 'sofortigen Aktivationsschwund' unterliegen: Einem gegenwärtig aktiven Knoten gelingt es noch, seine Aktivationsbeiträge für Aktivationen von anderen Knoten zum nächsten Zeitpunkt zu verschicken. Danach wird er sofort 'ausgeschaltet'.

Der Lernvorgang für *Ausgabeknoten* ist nun trivial: Ein externer Kritiker beurteilt die Effekte der Aktionen des Gesamtsystems. Ist der Kritiker zufrieden, so stärkt er die Gewichte aller Verbindungen auf die gegenwärtig aktiven Knoten proportional zu dem jeweiligen *'Beitrag'*, den diese Verbindung zuletzt geliefert hat. Der Beitrag ist einfach das Produkt des entsprechenden Gewichtes und der entsprechenden Aktivation.

Um auch diejenigen Gewichte am Lernprozeß zu beteiligen, die zu früheren Zeitpunkten überhaupt erst die Voraussetzungen für den späteren Erfolg schufen, brauchen wir jetzt noch ein Analogon zu dem Prozeß des 'Bietens' und des 'Verteilens von Stärke' bei Klassifikatorensystemen.

Dazu gehen wir wie folgt vor: *Alle von einem zur Zeit t aktiven Netzknoten k ausgehenden Verbindungen (falls vorhanden), die zu einem zur Zeit $t+1$ aktiven Netzknoten führen, opfern einen Teil ihrer Gewichtssubstanz. Diese Gewichtssubstanz wird unter denjenigen Verbindungen (falls vorhanden) proportional zu ihren jeweiligen Beiträgen verteilt, die durch die Weiterleitung von Aktivationen des Zeitpunkts $t-1$ die Aktivation von k auslösten.*

Da die Gewichte die kontextabhängige Aktivation von Knoten bestimmen, werden 'Gewinner' somit dafür 'bezahlt', ihren Nachfolgern das Privileg der Aktivierung zu gestatten. Eingabeknoten haben keine eingehenden Verbindungen, die sie stärken könnten, sie werden nur durch die Umgebung aktiviert. Daher repräsentieren die Eingabeknoten *Löcher*, durch welche Gewichtssubstanz auf Nimmerwiedersehen verschwindet.

Abgesehen von der Gewichtssubstanz, die das Netzwerk durch die Eingabeknoten verläßt, und von der, die durch die Ausgabeknoten in das Netzwerk hineinfließt, bleibt die Gesamtmenge aller sich im System befindlichen Gewichtssubstanz offensichtlich konstant.

Damit bekommen wir ein *dissipatives System*, welches von der Umgebung zur Verfügung gestellte Gewichtssubstanz konsumiert. Die einzige Möglichkeit für das System, sein Konsumverhalten aufrecht zu erhalten oder gar den Durchsatz zu erhöhen, besteht in der Ausführung von erfolgreichen Aktionssequenzen.

4.2.3 Gleichungen für diskrete Zeit

Zur Zeit t bezeichnen wir die Aktivation des j-ten Knotens mit $x_j(t)$, das Gewicht der gerichteten Verbindung von i nach j mit $w_{ij}(t)$ und ihren Beitrag mit $c_{ij}(t) = x_i(t-1)w_{ij}(t-1)$.

Die Aktivationsausbreitungsregel lautet:

Knoten j wird zur Zeit t aktiviert, falls er als Eingabeknoten eine sensorische Perzeption macht, oder falls er als Nicht-Eingabeknoten den lokalen Wettbewerb seiner WTA-Einheit gewinnt, weil ihm unter allen Mitbewerbern die größte positive Netzeingabe $net_j(t) = \sum_i c_{ij}(t)$ *zu eigen ist. Wir beschränken uns hier auf den einfachsten Fall: $x_j(t)$ ist gleich* 1*, falls j aktiv ist, und* 0 *sonst.*

Die Gewichtsänderungsregel findet simultan mit der Aktivationsausbreitungsregel ihre Anwendung (wir wollen ja Lokalität nicht nur im Raum, sondern auch in der Zeit):

Ist der Nicht-Eingabeknoten j aktiv, ändern sich die Gewichte gemäß

$$\Delta w_{ij}(t) = -\alpha c_{ij}(t) + \frac{c_{ij}(t-1)}{\sum_i c_{ij}(t-1)} \sum_{k\ wins} \alpha c_{jk}(t) + Ext_{ij}(t)$$

Dabei ist $0 < \alpha < 1$ eine positive Lernrate. $Ext_{ij}(t)$ ist die zum Zeitpunkt t vom externen Kritiker an w_{ij} spendierte Gewichtssubstanz, welche z.B wie folgt berechnet werden kann: Falls der Kritiker (noch) nicht weiß, ob das Systemverhalten belohnenswert war, ist $Ext_{ij}(t) = 0$. Ist der Kritiker aber der Ansicht, daß Belohnung vergeben werden sollte, und war der Knoten j zur Zeit t aktiv, so ist $Ext_{ij}(t) = \eta c_{ij}(t)$. Dabei ist η ein positiver Proportionalitätsfaktor.

Im Abschnitt über die Experimente mit der *neuronalen Eimerkette* werden wir sehen, daß es viel Raum für mehr oder weniger überwachte Strategien gibt, das System durch die Bestimmung von $Ext_{ij}(t)$ mit Lehrinformation zu versorgen: Es ist ohne weiteres möglich, *jeden* Knoten zu *jedem* Zeitpunkt oder aber auch nur *einige wenige* Knoten zu *isolierten* Zeitpunkten von externer Seite her zu instruieren.

Dank der zeitlich veränderlichen Umgebung sind es im allgemeinen nicht die *Aktivationen* (wie bei Hopfield-Netzen oder bei Equilibriums-BP), sondern höchstens die *Gewichte*, die einen Zustand des dynamischen Gleichgewichts erreichen können. Ein stabiler Zustand stellt sich ein, wenn jede Verbindung zu jedem Zeitpunkt gerade soviel Gewichtssubstanz verliert, wie sie im nächsten Zeitschritt wieder gewinnt. Das bedeutet, daß (parallel laufende) bereits etablierte 'Eimerketten' sequentielle Kooperation und Arbeitsteilung hervorrufen.

Der *lokale* Charakter aller für die neuronale Eimerkette notwendigen Berechnungen sei hier noch einmal betont. Es gibt keinen Bedarf nach extern definierten Trainingsintervallgrenzen. Es ist nicht notwendig, über weit vergangene Aktivationen Buch zu führen. Es wird nicht einmal eine kumulative Berechnung (etwa von exponentiell gewichteten Summen vergangener Aktivationen) gefordert. Jeder Knoten und jede Verbindung führen zu je-

dem Zeitpunkt im wesentlichen dieselben Berechnungen aus. Für beliebige Netzwerkstrukturen ist damit der Spitzenberechnungsaufwand pro Verbindung und Zeitschritt $O(1)$. Auf einer sequentiellen v. Neumann Maschine kann das Verfahren mit einem Spitzenberechnungsaufwand pro Zeitschritt von $O(dim(w))$ implementiert werden, wobei $dim(w)$ die Dimension des Gesamtgewichtsvektors w ist.

4.2.4 Mögliche Erweiterung für kontinuierliche Zeit

Ein letzter Rest globalen Charakters ist bei allen diskreten dynamischen Systemen noch durch die Notwendigkeit eines *Taktgebers* vorhanden. Das Eimerkettenprinzip ist potentiell jedoch auch für auf kontinuierlicher Zeit basierende Modelle neuronaler Informationsverarbeitung von Interesse. Um den asynchronen biologischen Vorbildern noch näher zu kommen, geben wir nun die Voraussetzung der vordefinierten WTA-Einheiten und des 'sofortigen Aktivitätsschwunds' auf. Um das Konzept der 'gewinnenden Knoten' zu retten, führen wir eine Nachbarschaftsordnung auf der Menge der Knoten sowie *starre* lokal inhibitorische Verbindungen zwischen benachbarten Knoten ein. Damit ist die Netztopologie durch eine (in ähnlicher Weise vielerorts im Gehirn vorgefundene) *'On-center-off-surround'*-Struktur eingeschränkt. Sowohl Kohonen als auch Grossberg verwenden (allerdings in ganz unterschiedlichem Kontext des unüberwachten Lernens) *'On-center-off-surround'*-Strukturen, um einen Wettbewerb zwischen Knoten zu implementieren [23] [11].

Wir nehmen an, daß sowohl die nun nicht mehr binären, sondern reellwertigen Ausgaben $x_j \in [0,1]$ des Knotens j als auch die Eigenschaften der variablen exzitatorischen Verbindungen durch Differentialgleichungen bestimmt sind, welche aussagen, daß x_j während der Zeit, die notwendig ist, um Aktivationsinformation von einem Knoten zu einem Nachfolger zu leiten, sich nicht signifikant ändert. Dann können wir eine Version für den Gewichtsänderungsalgorithmus angeben, die auf kontinuierlicher Zeit beruht ($net_j > 0$):

$$\frac{\partial w_{ij}}{\partial t} = -\alpha x_i w_{ij} x_j + \frac{x_i w_{ij}}{\sum_i x_i w_{ij}} \sum_k \alpha x_j w_{jk} x_k + Ext_{ij}$$

Ausschließlich positive Gewichte treten in dieser Formel auf, die inhibitorischen Verbindungen müssen ja starr bleiben. Bezeichnet man in suggestiver Weise $\sum_k w_{jk} x_k$ mit $back_j$, so beobachtet man (durch Gleichsetzung von $\frac{\partial w_{ij}}{\partial t}$ mit 0), daß der Gewichtsfluß durch ein positives Gewicht w_{ij}, welches keine externe Gewichtssubstanz abbekommt, genau dann ein dynamisches Gleichgewicht erreicht hat, wenn stets $net_j = back_j$ gilt.

Es sollte angemerkt werden, daß es einen wichtigen Unterschied zwischen der kontinuierlichen Version basierend auf *'On-center-off-surround'*-Verdrahtung und der weiter oben behandelten diskreten Version gibt. Während die diskrete Version für den Fall des Ausschaltens einer Eingabe zu einer WTA-Einheit den 'sofortigen Aktivitätsschwund' voraussetzt, wird unter der gleichen Voraussetzung bei *'On-center-off-surround'*-Verdrahtung kein *sofortiges* Abfallen der Aktivation zu messen sein. Es ist gegenwärtig unklar, ob die mögliche Erweiterung der neuronalen Eimerkette durch solche *Hystereseeffekte* in ihrer Funktionstüchtigkeit beeinträchtigt wird. Die im folgenden beschriebenen Experimente beziehen sich alle auf die diskrete Version.

4.3 Die Experimente

Für alle im folgenden beschriebenen Experimente wurden dieselben Initialisierungsbedingungen und dieselben Lernraten verwendet. Zu Beginn einer Trainingsphase wurden alle modifizierbaren Gewichte mit zufällig aus dem Intervall $[0.999, 1.001]$ gewählten Werten vorbesetzt. Sowohl η als auch α waren stets gleich 0.005. Es wurden keine systematischen Versuche unternommen, für ein gegebenes Problem durch geschickte Parameterwahl eine schnellere Konvergenz zu erreichen.

Für alle Experimente wurde die diskrete Version der neuronalen Eimerkette verwendet (siehe auch [50] und [53]). Die Rechengenauigkeit war durch 32-Bit-Gleitkommaarithmetik gegeben.

4.3.1 XOR-Varianten

Statische Probleme können als Spezialfall der sequentiellen Probleme aufgefaßt werden. Um die prinzipiellen Fähigkeiten des Eimerkettenalgorithmus zu illustrieren, wurde er zunächst anhand eines Problems vom *nicht linear separablen* Typ getestet. Das klassische Beispiel für solch ein Problem ist das XOR-Problem.

Minsky und Papert zeigten 1969, daß dieses Problem mit den damaligen Lernverfahren für neuronale Netze nicht lösbar war. Neuronale Netze erlebten ihre große Renaissance erst Mitte der 80er Jahre, als Methoden bekannt wurden, mit deren Hilfe gerade das XOR-Problem (und andere Probleme verwandten Typs) lösbar wurden.

Die verschiedenen bei den in diesem Unterabschnitt beschriebenen Experimenten verwendeten Netzwerke waren von der azyklischen Art. Im ersten Experiment wurde eine Lage mit drei Eingabeknoten vollständig mit einer drei 'versteckte' Knoten umfassenden 'versteckten' WTA-Einheit und

einer zwei Ausgabeknoten umfassenden WTA-Einheit 'vorwärtsvernetzt'. Auch die 'versteckte' WTA-Einheit hatte vorwärtsgerichtete Verbindungen zu den Ausgabeknoten.

Zu Beginn eines *'Zyklus'* wurden alle Knotenaktivationen mit 0 initialisiert, und eines der vier binären Eingabemuster wurde zufällig ausgewählt. Ein Zyklus dauerte sechs Zeitschritte, während derer das Muster den beiden ersten Eingabeknoten präsentiert wurde. Die Aktivation des dritten Eingabeknotens (des sogenannten *wahren* Knotens) war immer gleich 1, um eine modifizierbare 'Schwelle' für jeden Nicht-Eingabeknoten zu bieten. ('Wahre' Knoten sind für viele konnektionistische Ansätze gang und gäbe. Sowohl BP-Netze als auch die Boltzmann-Maschine werden häufig mit 'wahren' Eingabeknoten ausgestattet.)

Die Aufgabe für das Netzwerk bestand darin, den ersten bzw. den zweiten Ausgabeknoten anzuschalten, je nachdem, ob die Anwendung der XOR-Funktion auf das Eingabemuster 1 oder 0 ergab. Das Problem war dabei als Reinforcement-Lernaufgabe definiert: Zu jedem Zeitpunkt t vergab ein externer Kritiker genau dann Gewichtssubstanz $Ext_{ij}(t) = \eta c_{ij}(t)$ an ein Gewicht w_{ij}, wenn j ein Ausgabeknoten und noch dazu korrekt angeschaltet war. In allen anderen Fällen galt $Ext_{ij}(t) = 0$.

Ein Eingabemuster galt als vom System korrekt klassifiziert, wenn bei *ausgeschaltetem* Gewichtsänderungsmechanismus während der letzten drei Zeitschritte eines Zyklus der richtige Ausgabeknoten aktiviert war. Um dieses Kriterium zu testen, wurde der Änderungsalgorithmus während sogenannter *Prüfzyklen* abgeschaltet. Das diente zur Verhinderung der Möglichkeit, daß das Netzwerk *während* eines Prüfzyklus noch aus einer falschen Klassifikation eine richtige machen konnte.

Die Aufgabe galt als vom System gelöst, wenn es alle vier Eingabemuster korrekt klassifizierte. Um herauszufinden, ob das Netzwerk schon eine Lösung gefunden hatte, wurde der Eimerkettenmechanismus nach jedem Trainingszyklus suspendiert und die Klassifikationsfähigkeit des Netzes anhand der vier Eingabemuster getestet.

Bei 20 verschiedenen Testläufen mit unterschiedlichen Anfangsbedingungen waren jeweils durchschnittlich 619 Musterpräsentationen nötig, um eine Lösung zu finden. Das bedeutet, daß jedes der vier Eingabemuster ungefähr 155 mal präsentiert werden mußte. Dies entspricht dem in der Literatur häufig verwendeten Begriff von 155 'Epochen'.

Die meisten der 20 Lösungen waren unstabil insofern, als weiteres Training sie nicht notwendigerweise fixierte. So konnten zum Beispiel nach dem Finden einer Lösung 5 weitere Trainingszyklen dazu führen, daß die Performanz sich wieder verschlechterte. Daher wurden in weiteren Experimenten auch die Zyklenzahlen gemessen, die für eine stabile Lösung notwendig waren.

Eine Lösung wurde als 'stabil' angesehen, wenn 100 zusätzliche Musterpräsentationen die Klassifikationsperformanz nicht störten. Die exakte Testprozedur lief wie folgt ab: Nach jedem Trainingszyklus wurde der Gewichtsänderungsmechanismus abgeschaltet. Daraufhin wurde geprüft, ob das Netz ein neues zufällig ausgewähltes Muster richtig klassifizierte. Schließlich wurde der Eimerkettenalgorithmus wieder angeworfen. Dieses Verfahren wurde iteriert, bis das Netzwerk bei 100 aufeinanderfolgenden Prüfzyklen 100 korrekte Klassifikationen produzierte. Bei 10 verschiedenen Testläufen stellte sich heraus, daß jedes Muster durchschnittlich 674 mal angeboten werden mußte, um dem obigen Kriterium Genüge zu tun.

Verwendete man zwei anstelle von drei 'versteckten' Knoten, so waren durchschnittlich 160 Präsentationen pro Muster zum Auffinden einer Lösung notwendig. Es schien jedoch nicht möglich, unter diesen Bedingungen stabile Lösungen (im Sinne des obigen Kriteriums) zu finden.

Das XOR-Problem wurde auch mit einer anderen Netzwerkarchitektur getestet. Statt direkte vorwärtsgerichtete Verbindungen von den Eingabeknoten zu den Ausgabeknoten zuzulassen, wurden die Eingabeknoten lediglich mit zwei WTA-Einheiten verbunden, von denen jede zwei 'versteckte' Knoten enthielt. Die versteckten' WTA-Einheiten besaßen ihrerseits Verbindungen zu den Ausgabeknoten.

Bei zehn Testläufen verfehlte es der Algorithmus unter diesen Umständen zweimal, in weniger als 4000 Trainingszyklen eine Lösung zu finden. In den verbliebenen 8 Fällen waren durchschnittlich 263 Musterpräsentationen zum Auffinden einer Lösung erforderlich. 911 Präsentationen waren für die Ausprägung einer stabilen Lösung (siehe obiges Kriterium) notwendig.

Verglichen mit der Boltzmann-Maschine [14] schneidet das Verfahren vom Gesamtrechenaufwand her gut ab, verglichen mit BP-Netzen erweist es sich als langsam. Wie schon in der Einleitung erwähnt, geht es uns jedoch nicht um die Lerngeschwindigkeit, sondern um den experimentellen Nachweis, daß vollständig lokales Lernen auch in Netzen mit versteckten Knoten möglich ist. Sowohl die Boltzmann Maschine als auch BP sind *nicht* stark lokal.

Woher können Instabilitäten kommen?

Was ist die Quelle möglicher Instabilitäten? Wie zu erwarten, ergaben experimentelle Beobachtungen, daß miteinander im Wettbewerb stehende Knoten einer WTA-Einheit zu einem gegebenen Zeitpunkt oft sehr ähnliche Netzeingaben hatten. Das wiederum kann man auf eine Eigenschaft der neuronalen Eimerkette zurückführen: Das Gewicht einer Verbindung, die zu einem den Wettbewerb verlierenden Knoten führt, ändert sich nicht. Man betrachte einen Knoten j, welcher *nicht* an einer die korrekte Klassi-

fikation eines Musters A (mit)verursachenden Eimerkette teilnimmt. Auf j führende Verbindungen ändern sich nicht. Werden sie jedoch während einer in ähnlichem Kontext stattfindenden korrekten Klassifikation eines Musters B leicht erhöht, so mag das auch bei der nächsten Präsentierung von A zu einer Gewinnersituation für j führen. Daraus wiederum mag eine inkorrekte Klassifikation von A resultieren. Das Zusammenspiel solcher Effekte stellt eine potentielle Quelle von Instabilitäten dar.

Es wurden auch Experimente für den Fall durchgeführt, daß die Eingabemuster für aufeinanderfolgende Trainingszyklen nicht aufgrund zufälliger Auswahl, sondern in periodischer sequentieller Ordnung angeboten werden. Dabei stieg die durchschnittliche Zeitdauer bis zum Auffinden sowohl unstabiler als auch stabiler Lösungen an. Dies suggeriert einen stabilisierenden Effekt des Zufallselements bei der Musterauswahl.

Es gibt mindestens zwei Möglichkeiten, anfänglichen Instabilitäten während der Lernphase entgegenzutreten. Von der wohl einfachsten Möglichkeit wurde bereits bei den obigen Experimenten mit Erfolg Gebrauch gemacht: Man wartet einfach, bis die Fluktuationen zurückgehen, und sich das System stabilisiert. Die zweite Möglichkeit ist: Man wartet bei einer gegebenen Aufgabe auf das erste Auftreten einer Lösung, und fixiert dann alle Gewichte.

4.3.2 Varianten der Dekodierprobleme

Ein weiterer Aufgabentyp, an dem Lernverfahren für versteckte Knoten (BP, Boltzmann Maschine) häufig getestet werden, ist durch die *Dekodierprobleme* gegeben. In unserem Fall bestand das Problem darin, die acht achtdimensionalen binären Muster mit Einheitslänge jeweils mit sich selbst zu assoziieren. Ein 'Informationsflaschenhals' (*bottleneck*) sorgte dafür, daß das Problem nicht trivial wurde: Acht Eingabeknoten waren mit drei jeweils zwei Knoten umfassenden WTA-Einheiten 'vorwärtsverbunden'. Von den drei versteckten WTA-Einheiten gingen Verbindungen zu einer acht Ausgabeknoten umfassenden WTA-Einheit aus. Zusätzlich war ein 'wahrer' Knoten mit allen Nicht-Eingabeknoten verbunden. Da jede versteckte WTA-Einheit nur zwei verschiedene Aktivationszustände annehmen konnte, war der Lernalgorithmus gezwungen, eine *extreme* Lösung zu finden: Die gesamte Repräsentationskapazität ($2^3 = 8$ mögliche Zustände) des 'Flaschenhalses' mußte ausgeschöpft werden. Man beachte, daß bei BP-Netzen mit analoger Struktur (mit 3 versteckten Knoten) dank der reellwertigen Aktivierungsfunktionen wesentlich mehr Repräsentationskapazität im Flaschenhals steckt.

Das Vorgehen in diesem Fall illustriert, wie der Eimerkettenmechanismus in *überwachter* Manier angewendet werden kann. Im Gegensatz zu dem bei den XOR-Experimenten verwendeten Verfahren wurden nämlich

auch Verbindungen zu denjenigen Ausgabeknoten gestärkt, die in Reaktion auf ein angebotenes Eingabemuster aktiv *hätten sein sollen*, obwohl sie vielleicht irrtümlicherweise ausgeschaltet blieben. Abgesehen von dieser Änderung wurde die oben beschriebene Lern- und Prüfprozedur vollständig übernommen.

Bei zehn Testläufen erwiesen sich durchschnittlich 1364 Musterpräsentationen pro Muster als notwendig, um eine Lösung zu finden. Es schien jedoch nicht möglich, die Lösungen für drei versteckte WTA-Einheiten vollständig zu stabilisieren. Erweiterte man den Flaschenhals auf vier versteckte WTA-Einheiten, so waren 2460 Präsentationen pro Muster notwendig, um dem Stabilitätskriterium Genüge zu tun.

Mit dem analogen 2^4-4-2^4-Dekodierproblem schienen die Grenzen des Lernverfahrens erreicht: Hier wurden bei verschiedenen Testläufen keine vollständigen Lösungen erzielt. Erneut sei darauf hingewiesen, daß nicht behauptet werden soll, daß die neuronale Eimerkette perfekt sei. Es geht vielmehr darum, nachzuweisen, daß zumindest in einigen Fällen vollständig lokales Lernen und nicht-lineare Lernprobleme kompatibel sind.

4.3.3 Sequenzgenerierung

Die hier beschriebene Aufgabe erfordert das oszillierende Verhalten gewisser Ausgabeknoten als Antwort auf eine stationäre Eingabe.

Zwei Eingabeknoten wurden mit einer aus drei Ausgabeknoten bestehenden WTA-Einheit 'vorwärtsverbunden'. Die Ausgabeknoten waren untereinander bidirektional vollständig vernetzt. Die Aufgabe war es, den ersten und den zweiten Ausgabeknoten in alternierender Weise an- bzw. auszuschalten, solange der erste Eingabeknoten aktiv war. Der zweite Eingabeknoten diente der Wahrnehmung eines Stopsignals: Seine Aktivation mußte mit einer stationären Ausgabe des dritten Ausgabeknotens beantwortet werden.

Die Lehrmethode für dieses Problem demonstriert, wie ohne nennenswerten Aufwand ein *'Lehrerzwang' (teacher forcing,* ein im Zusammenhang mit einem ganz anderen überwachten Lernalgorithmus von Williams und Zipser geprägter Begriff [80]) eingeführt weden kann: Statt die *tatsächlichen* Aktivationen der Ausgabeknoten zur Zeit t zur Berechnung der Ausgaben zur Zeit $t+1$ heranzuziehen, verwendet man einfach die *gewünschten* Aktivationen zur Zeit t.

Während das rekurrente Netzwerk ohne Vorgabe irgendwelcher Trainingsintervallgrenzen 'vor sich hintickte', wurden die binären Eingabeaktivationen zu jedem Zeitpunkt zufällig gewählt. Zu einem gegebenen Zeitpunkt war genau ein Eingabeknoten aktiv. Dabei war die Wahrscheinlichkeit der Aktivierung des ersten Eingabeknotens gleich 75 Prozent, die Wahr-

scheinlichkeit der Aktivierung des zweiten Eingabeknotens betrug also 25 Prozent. Gewichtssubstanz wurde immer dann vergeben, wenn die 'richtige' Ausgabeeinheit angeschaltet war. Innerhalb von meist weniger als 30 Zeitschritten fand das Netzwerk eine zufriedenstellende stabile Lösung seiner Aufgabe. Gerade wie bei Williams' und Zipsers Experimenten mit einem ganz ähnlichen Problem (und einem *nicht* lokalen Algorithmus) konnte die Aufgabe *ohne* Lehrerzwang nicht immer gelernt werden.

4.3.4 Sequenzerkennung

Eine wichtige Aufgabe im Kontext nichtstationärer Umgebungen ist es, zwischen verschiedenen Arten von Bewegung unterscheiden zu lernen. Ein einfaches Experiment mit zeitlich veränderlichen Wahrnehmungen wurde durchgeführt. Eine eindimensionale 'Retina', bestehend aus 5 normalen Eingabeknoten plus einem 'wahren' Knoten, wurde mit einer zwei Ausgabeknoten umfassenden WTA-Einheit 'vorwärtsvernetzt'. Die Ausgabeknoten waren ihrerseits untereinander vollständig bidirektional vernetzt. Die Aufgabe bestand darin, zumindest während der letzten drei Zeitschritte einer fünf Zeitschritte dauernden Eingabesequenzpräsentation den ersten Ausgabeknoten dann anzuschalten, wenn ein 'Illuminationspunkt' die Retina von 'rechts' nach 'links' durchquerte, bzw. den zweiten Ausgabeknoten anzuschalten, wenn der Illuminationspunkt von 'links' nach 'rechts' wanderte.

Während eines Trainingszyklus' wurde eine der zufällig gewählten Eingabesequenzen dem Netzwerk zweimal hintereinander angeboten. Gewichtssubstanz wurde analog zu den stationären XOR-Experimenten vergeben (es handelte sich also um eine Reinforcement-Lernaufgabe). In einem von 10 Testläufen schaffte es das System nicht, innerhalb von 3000 Präsentationen eine stabile Lösung zu finden (das Kriterium für 'stabil' entsprach dem für stationäre Experimente). In den verbleibenden 9 Testläufen waren durchschnittlich 223 Trainingszyklen pro Sequenz zur Auffindung einer stabilen Lösung erforderlich.

4.3.5 Grenzen der neuronalen Eimerkette

Weitere Versuche wurden ausgeführt, um die Grenzen des Schemas zu erkunden. Im folgenden beschreiben wir zwei Probleme, die vom Eimerkettenalgorithmus *nicht* befriedigend gelöst wurden (ein Balancierproblem, und ein Flip-Flop-Problem). Bei beiden Problemen sind unbekannte Verzögerungsdauern zwischen relevanten Ereignissen zu berüchsichtigen. Aus dem Scheitern der Eimerkette bei diesen Aufgaben erwächst Motivation für die beiden nachfolgenden Kapitel, in welchen mächtigere Algorithmen vorgestellt werden. Diese verfügen allerdings nicht mehr über die starke Lokalität

der neuronalen Eimerkette.

Ein Balancierproblem

Bei dem Balancierproblem ging es darum, zu jedem Zeitschritt auf einen sich auf einer eindimensionalen Spur befindlichen Wagen eine Kraft auszuüben, und zwar dergestalt, daß ein auf dem Wagen montierter und entlang der Bewegungsrichtung des Wagens frei schwingender Stab nicht zu weit ausschlug. Auch durfte der Wagen nicht an die Begrenzung der Spur stoßen. Es wurde nur 'verzögerte' Reinforcement-Information zur Verfügung gestellt. Wir übernahmen das Problem, das Anderson in Erweiterung einer von Barto, Sutton und Anderson gestellten Aufgabe [5] formuliert hat [2]. Nur wenn das System in einen Fehlerzustand eintrat (d.h. wenn der Wagen an den Rand stieß oder der Stab umfiel), wurde *keine* Gewichtssubstanz vergeben.

Die im Anhang beschriebenen Differentialgleichungen wurden zur Modellierung des Wagen/Stab-Systems herangezogen. Die Rechnersimulation beruhte auf der Eulerschen Methode, zwei aufeinanderfolgende Zeitschritte lagen 0.02 Sekunden auseinander. Die vier Netzeingaben, die dem Netzwerk zu einem gegebenen Zeitpunkt die Position des Wagens, den Winkel des Stabes mit der Vertikalen sowie die zeitlichen Ableitungen dieser beiden Größen zugänglich machten, sind ebenfalls im Anhang definiert. Zu Beginn eines Versuchs wurden die vier physikalischen Zustandsvariablen jeweils mit 0 initialisiert.

Die Aktivationen der Elemente einer aus zwei vollständig miteinander vernetzten Ausgabeknoten bestehenden WTA-Einheit wurden zu jedem Zeitschritt wie folgt interpretiert: War der erste Knoten aktiv, so wurde parallel zur Spur eine Kraft von $10N$ auf den Schwerpunkt des Wagens ausgeübt. War der zweite Knoten aktiv, so betrug die entsprechende Kraft $-10N$. Wendete man zu jedem Zeitpunkt eine entweder $10N$ oder $-10N$ betragende *zufällig* gewählte Kraft an, so lag die durchschnittliche Dauer bis zum Eintritt in einen Fehlerzustand bei ca. 20 Zeitschritten.

Zu fast allen Zeitpunkten wurde Gewichtssubstanz an den jeweils gerade aktiven Ausgabeknoten vergeben. Die einzige Ausnahme waren Zeitpunkte, in denen das System in einen Fehlerzustand geriet: In solchen Fällen wurde keine Gewichtssubstanz spendiert.

Die Aufgabe erwies sich als für die neuronale Eimerkette als zu schwierig. Die längste Balancierdauer, die das System unter verschiedensten Bedingungen (Anzahl der versteckten Knoten, Netztopologie) erreichte, betrug 31 Zeitschritte. Bei näherer Betrachtung der Lösung stellte sich folgendes heraus: Die Ausgaben des Netzwerkes waren im wesentlichen *unabhängig* von den Eingaben. Der simple vom Netz implementierte Algorithmus bestand

einfach darin, abwechselnd den ersten und den zweiten Ausgabeknoten zu aktivieren. Das reichte zwar aus, um besser als das Zufallsverfahren abzuschneiden. Als eine befriedigende Lösung kann das aber kaum akzeptiert werden.

Ein Flip-Flop Problem

Bei diesem Problem hätte das Eimerkettennetzwerk lernen sollen, sich analog zu einem Flip-Flop zu verhalten. Die Aufgabe bestand dabei darin, auf einen kontinuierlichen Strom von Eingabeereignissen (es gab 3 mögliche lokal repräsentierte Eingabeereignisse 'A', 'B' und 'C') in folgender Weise zu reagieren: Wann immer ein Ereignis 'B' zum ersten Mal nach einem Ereignis 'A' stattfand, sollte der erste Ausgabeknoten einer 2 Knoten enthaltenden WTA-Einheit aktiviert werden, in allen anderen Fällen der zweite Knoten.

Eine Schwierigkeit bestand darin, daß die Ereignisse in der Umgebung auf zufällige Weise generiert wurden und *beliebig* lange Verzögerungen zwischen relevanten Ereignissen auftreten konnten. C mußte also *lernen*, sich das Eintreten des Ereignis 'A' über einen im Prinzip beliebig langen Zeitraum zu merken, nachfolgende unter Umständen irrelevante Ereignisse zu ignorieren und beim Eintreten von 'B' in korrekter Weise zu reagieren sowie die 'Speicherung' von 'A' rückgängig zu machen.

Die Hauptschwierigkeit bestand in der Abwesenheit eines informativen Lehrers. Gewichtssubstanz wurde wie beim XOR-Experiment dann vergeben, wenn das Netz den richtigen Knoten zum richtigen Zeitpunkt aktivierte.

Unter verschiedensten Bedingungen (Anzahl der versteckten Knoten, Netztopologie) war es dem Eimerkettenalgorithmus nicht möglich, die Aufgabe zu lösen.

Die Tatsache, daß die neuronale Eimerkette bei langen Verzögerungen zwischen Aktionen und ihren späteren Konsequenzen sich nicht als das geeignete Verfahren erweist, legt es nahe, für solche Fälle nach tragfähigeren Alternativen für Netzwerke mit interner und externer Rückkopplung zu suchen. Die Kapitel 5 und 6 widmen sich diesem Anliegen, dort wird allerdings die starke Lokalität aufgeben.

4.4 Vergleich mit anderen Ansätzen

Zusammenfassend beschreiben wir in den folgenden Unterabschnitten nochmals wesentliche Gemeinsamkeiten mit und Unterschiede zu konzeptuell verwandten Methoden.

4.4.1 Bezug zu Hollands 'bucket brigade' für regelbasierte Systeme

Die Gemeinsamkeiten zu Hollands Ansatz sind offensichtlich, schließlich bot Holland die Inspiration für die vorgestellte Methode. Wir konzentrieren uns hier lediglich auf die wesentlichen Unterschiede.

Ein wichtiger Unterschied ist natürlich die nicht-lokale Natur mancher von Hollands System auszuführenden Berechnungen. Dazu gehört der globale Wettbewerb aller Klassifikatoren genauso wie die globale Kommunikation über den durch die Botschaftentafel gegebenen 'Flaschenhals'. Daraus erwachsen im Gegensatz zur neuronalen Eimerkette große Schwierigkeiten für eine wirklich parallele Implementierung.

Ein weiterer Unterschied zwischen der neuronalen Eimerkette und Hollands Ansatz besteht darin, daß es keinen zur Kreierung neuer Klassifikatoren analogen Prozeß gibt: Die Anzahl der Verbindungen im Netzwerk bleibt konstant. Dieser Unterschied wird durch die Modifizierbarkeit der Verbindungen gerechtfertigt. Gewichte dürfen geändert werden, die *Spezifizität* und sonstige der Informationsübermittlung dienende Eigenschaften von Klassifikatoren jedoch nicht. (Compiani, Montanari, Serra und Valastro machen einige weitere Aussagen über Beziehungen zwischen Klassifikatorsystemen und neuronalen Netzen im allgemeinen [7]).

4.4.2 Bezug zu 'Competitive Learning'

Es gibt eine Analogie zwischen der zielgerichteten neuronalen Eimerkette und dem statischen 'Wettbewerbslernen' (*competitive learning*) der *unüberwachten* Art. Statisches Wettbewerbslernen [23] [11] [46] kann ebenfalls als eine lokale Umverteilung von Gewichtssubstanz interpretiert werden. Beim Wettbewerbslernen sind es jedoch schwache Beiträge leistende *eingehende* Verbindungen eines 'gewinnenden' Knotens, die etwas von ihrem Gewicht an starke Beiträge leistende eingehende Verbindungen abtreten müssen. Im Gegensatz dazu verursacht die neuronale Eimerkette einen Gewichtssubstanztransport von den *ausgehenden* Verbindungen zu den eingehenden. Dies ist die Schlüsseleigenschaft des Algorithmus, die den Erfolg vergangener Aktionen zum Erfolg zukünftiger Aktionen in Beziehung setzt.

4.4.3 Bezug zu TD-Methoden

Es ist unwahrscheinlich, daß die neuronale Eimerkette einen *Gradientenabstieg* in einem vernünftigen globalen Performanzmaß durchführt. Möglicherweise bieten jedoch Suttons in Kapitel 2 beschriebene TD-Methoden [67] einen Rahmen für die Analyse der Konvergenzeigenschaften des Verfahrens

[61] (TD-Methoden sind ja als eine Verallgemeinerung des Gradientenabstiegs anzusehen):

Sutton skizzierte einige Beziehungen zwischen TD-Methoden und Hollands *'bucket brigade'* für regelbasierte Systeme [67]. Betrachtet man Suttons Skizze im Lichte der Konzepte dieses Kapitels, so bietet sich folgende Sichtweise an: Zu einem gegebenen Zeitpunkt kann man das Gewicht w_{ij} einer zu einem aktiven Knoten j führenden Verbindung (oder ihren Wetteinsatz αw_{ij}) als eine *Voraussage* der Gewichtssubstanz interpretieren, die diese Verbindung erhalten wird. Diese Vorraussage hängt rekursiv von den Voraussagen derjenigen Gewichte ab, die zu späteren Zeitpunkten Aktivierungsbeiträge leisten werden. Daher macht w_{ij} auch eine Voraussage über die vom externen Kritiker zu vergebende Gewichtssubstanz, welche die Rekursion beendet. Ein dynamisches Equilibrium des Gewichtsflusses bedeutet, daß Voraussagen stets in Erfüllung gehen.

Unglücklicherweise gelang es aufgrund des durch die WTA-Einheiten eingeführten kompetitiven Elements nicht, eine Konvergenzanalyse der neuronalen Eimerkette durchzuführen. Ähnliches gilt allerdings für die etablierten Klassifikatorsysteme: Bis jetzt hat noch niemand ein Theorem bewiesen, welches aussagt, daß der *'bucket brigade algorithm'* für regelbasierte Systeme *immer* wie gewünscht funktionieren *muß*.

4.4.4 Kritik und Ausblick

Die Motivation dieses Kapitels war es, die Kompatibilität vollständiger Lokalität mit zielgerichtetem Lernen unter Einschluß versteckter Knoten zu demonstrieren. Dieses Ziel wurde erreicht insofern, als zumindest bei einigen Experimenten nicht-linearer Natur gezeigt werden konnte, daß das vollständig lokale Eimerkettenprinzip zu brauchbaren Lösungen führen kann.

Andererseits wurden auch die Schranken des Algorithmus offenbar: Bei vergleichsweise großen Netzen sowie bei Aufgaben mit vergleichsweise langen zeitlichen Verzögerungen zwischen relevanten Ereignissen fand A1 keine befriedigenden Lösungen. Geht es um großmaßstäbliche Anwendungen, so ist vom Gebrauch der neuronalen Eimerkette abzuraten. (Das gleiche gilt allerdings auch für manches *nicht-lokale* Lernverfahren, wie z.B. die Boltzmann-Maschine.)

Obwohl (oder gerade weil?) die neuronale Eimerkette auf generelle Lernsituationen abzielt, muß sie sich den Vorwurf gefallen lassen, zu den 'schwachen Methoden' *('weak methods')* zu gehören. Die neuronale Eimerkette ist unter die *generate-and-test*-Verfahren einzureihen. Sie bemüht sich nicht darum, ein Modell externer Zusammenhänge zu konstruieren und dieses Modell zielgerichtet auszunutzen.

Die Tatsache, daß wenigstens für relativ kleine Probleme ein vollständig

lokaler neuronaler Lernalgorithmus existiert, mag als Grundlage für weiterführende Arbeiten dienen. Es wäre zum Beispiel denkbar, daß man mehrere 'Eimerkettenmodule' für verschieden hohe Ebenen der Problemlösung einsetzen kann. Die Motivation einer solchen Unternehmung bestünde gerade für lange Zeitdauern umfassende Aktionssequenzen darin, die Länge von Eimerketten kurz zu halten, um sie unanfälliger gegen Störungen verschiedenster Art zu machen.

Statt diese Möglichkeit in näheren Augenschein zu nehmen, werden wir uns in den nächsten Kapiteln jedoch zunächst mit modellbildenden Verfahren auseinandersetzen. Die Idee, in hierarchischer Weise größere Programme durch Komposition kleinerer Unterprogramme zu erlernen, wird aber im letzten Kapitel dieser Arbeit eine zentrale Rolle spielen.

Kapitel 5

'Rekurrente' Umgebungsmodelle für R-Lernen

Die in diesem Kapitel vorgestellte Klasse von Algorithmen für rekurrente Steuernetzwerke basiert auf *Systemidentifikation*. Der allen Verfahren zugrundeliegende Algorithmus wird im folgenden A2 genannt. A2 zielt explizit auf die allgemeinste Sorte externer Dynamik: Er beinhaltet eine Komponente, die auch für *nicht* Markov-artige externe Dynamik einen allgemeinen Performanzgradienten approximiert und diese Approximation für zielgerichtetes Verhalten ausnutzt. Allerdings wird diese Fähigkeit des Algorithmus erkauft durch einen Verlust der räumlichen Lokalität.

A2 stellt eine Erweiterung der im 3. Kapitel beschriebenen Arbeiten von Munro, Jordan, Werbos, Widrow und Robinson dar. Dieses Kapitel ist wie folgt gegliedert: Nach einer Zusammenfassung und einer intuitiven Erklärung des Prinzips wird der Algorithmus mathematisch begründet und anschließend in implementierfähiger Form beschrieben. Zwei Versionen von A2 werden betrachtet: Bei der *sequentiellen* Version wird zunächst eine Systemidentifikationskomponente daraufhin trainiert, die Umgebung zu simulieren. Danach schließt sich die Lernphase einer Steuerkomponente an. Bei der *parallelen* Version lernen beide Komponenten gleichzeitig. A2's potentielle Mächtigkeit wird illustriert durch das (nach bestem Wissen des Autors) erste Experiment zum R-Lernen in einer Nicht-Markov-Umgebung.

Danach wird ausgeführt, wie A2 unter Verlust an Effizienz für vorausschauendes adaptives Planen ausgenützt werden kann. Wesentliche Unterschiede zu den adaptiven Kritikern werden dabei besprochen. Gewisse Vor-

teile des Konzepts der adaptiven Kritiker motivieren das nachfolgende Kapitel.

Die Schlußkritik bezieht sich vor allem auf die nicht vorhandene Kompositionalität des Algorithmus und stellt zusätzliche Motivation für das abschließende 8. Kapitel dar.

5.1 Zusammenfassung

Der im folgenden vorgestellte Algorithmus A2 wurde für R-Lernsituationen mit interner und externer Rückkopplung entwickelt. Das Hauptziel des Netzwerkes ist, so viel zeitlich kumulative 'Lust' oder so wenig zeitlich kumulativen 'Schmerz' wie möglich zu erleben.

A2 basiert auf zwei gekoppelten Netzwerken. Beide Netzwerke sind vollständig zyklisch. Das "Modellnetzwerk" dient dazu, die externe Umgebung mittels Voraussagen über die zukünftigen Eingaben des "Steuernetzwerkes" zu modellieren. Zu diesen Eingaben gehören u.a. auch spezielle "Schmerzerlebnisse" oder "Lusterlebnisse", welche durch "unerwünschte" bzw. "erwünschte" Aktivationen bestimmter Netzknoten realisiert werden. Die Voraussage des Modellnetzwerkes basiert auf vergangenen Eingaben und Ausgaben des Steuernetzwerkes. Die einzige Aufgabe des adaptiven Modellnetzwerkes ist es, die externe Dynamik in einer Form zu repräsentieren, die es erlaubt, "Schmerz- und Lustgradienten" für das ebenfalls adaptive Steuernetzwerk zu berechnen. Zumindest im Prinzip sind beliebige Verzögerungen zwischen Aktionen und späteren Konsequenzen erlaubt.

5.2 Intuitive Erklärung des Algorithmus A2

Wie schon im einführenden Kapitel verdeutlicht wurde, ist für *'on-line'* Lernen die Lokalität in der Zeit wichtiger als die Lokalität im Raum.

Im Gegensatz zu den bei allen anderen Algorithmen für R-Lernen verfolgten Ansätzen betrachten wir Reinforcement im folgenden *als eine weitere Eingabe* für ein neuronales Steuernetzwerk C. Die Menge der Eingabeknoten von C wird einfach unterteilt in zwei Untermengen: Die Menge I der 'normalen' perzeptiven Eingabeknoten, und die Menge P der sogenannten 'Schmerzknoten' und 'Lustknoten' (auch 'Reinforcement-Knoten' oder 'R-Knoten' genannt).

Damit führen wir ein im allgemeinen *multidimensionales* R-Signal ein und unterscheiden uns damit von anderen Ansätzen, bei denen das Reinforcement stets eine skalare Größe ist. Multidimensionales (oder auch *vektorwertiges*) Reinforcement entspricht den vielfältigen Schmerz- oder Lust-

wahrnehmungsmöglichkeiten biolgischer Systeme.

Alle Eingabeknoten sind mit allen Nicht-Eingabeknoten von C vorwärtsverbunden, die Nicht-Eingabeknoten sind ihrerseits vollständig bidirektional untereinander vernetzt.

Die Schmerz- und Lustknoten zeichnen sich gegenüber den anderen Eingabeknoten lediglich dadurch aus, daß für sie zu gegebenen Zeitpunkten eine *erwünschte* Aktivation definiert sein kann. Für unsere Zwecke wird die erwünschte Aktivation eines Schmerzknotens im folgenden zu jedem Zeitpunkt stets gleich *Null* sein. Die erwünschte Aktivation eines Lustknotens wird zu jedem Zeitpunkt gleich einem vordefinierten positiven Skalar sein. In der Implementation sind 'Schmerz' und 'Lust' durch entsprechende Skalierung ineinander überführbar.

Für den Fall, daß extern definierte Trainingsintervallgrenzen vorgegeben sind (später werden wir solche Episodengrenzen vermeiden), ist die zu minimierende Größe durch

$$\sum_{t,i}(y_i(t) - c_i)^2$$

gegeben. $y_i(t)$ ist dabei die tatsächliche und c_i die gewünschte Aktivation des $i$ten Schmerz- oder Lustknotens zur Zeit t, t rangiert über alle Zeitschritte eines Trainingsintervalls.

Falls verschiedene Arten von R-Signalen in einer asymmetrischen Weise gewichtet werden sollen, mag die zu minimierende Größe auch eine nichttriviale Funktion der Aktivationen verschiedener R-Knoten sein. Ist diese Funktion eine Linearkombination, so kann sie sofort durch einen linearen Knoten implementiert werden, dessen Eingabe zu einem gegebenen Zeitpunkt aus dem gegenwärtigen Aktivationsvektor der R-Knoten besteht, und dessen fixe Synapsen bestimmte Schmerz- oder Lustarten stärker gewichten als andere.

Als ein (utopisches) Beispiel betrachte man wieder einen autonomen Agenten, dessen Bewegungen durch C's Ausgaben gesteuert werden. In der physikalischen Umgebung des Agenten befindet sich eine Steckdose. Gelingt es dem Agenten, seinen Stecker in die Steckdose zu stecken, so wird seine Batterie aufgefrischt. Der Agent kann nun verschiedenartige unliebsame Erfahrungen machen, indem er zum Beispiel mit einer seiner Extremitäten zu heftig gegen ein Hindernis stößt. Dies zieht nämlich die Aktivation bestimmter Schmerzknoten nach sich. Andere Schmerzknoten werden aktiv, wann immer die Batterieladung unter einen bestimmten Schwellwert sinkt. Der Agent ist autonom insofern, als kein intelligenter Lehrer vonnöten ist, um ihn mit irgendwelchen weiteren Zielen oder Subzielen auszustatten.

Wie schon in den einführenden Kapiteln gesehen, hilft ein purer überwachter Lernalgorithmus dem Agenten nicht weiter, sein Ziel zu erreichen

(nämlich zu existieren, ohne unerwünschte Eingaben zu bekommen). Mit dem Systemidentifikationsansatz kann jedoch ein vollständig rekurrentes Modellnetzwerk M trainiert werden, die Beziehungen zwischen Eingaben von der Umgebung, Ausgaben von C und dem negativen Reinforcement (den unerwünschten Eingaben) zu modellieren. M hilft schließlich dem Gesamtsystem, 'Schmerz- oder Lustgradienten' für C zu berechnen, um das kumulative Reinforcement zu minimieren.

Da die Gradienten für Schmerz und Lust von 'rückwärts in die Zeit gerichteten durch die Umgebung führenden Lernpfaden' abhängen, sollte M nicht nur die Aktivationen der R-Knoten, sondern auch die der normalen Eingabeknoten vorhersagen. Gute Voraussagen über zukünftiges Reinforcement hängen im allgemeinen von einer guten Kenntnis der gesamten externen Dynamik ab. Hier erweitert der Algorithmus A2 den Ansatz von Robinson [41][39]. Der Zweck des adaptiven Modellnetzwerkes ist es, *die gesamte sichtbare Dynamik der Umgebung differenzierbar zu machen* [54] [56] [58].

Man betrachte Abbildung 6.1. Zu sehen ist ein Steuernetz mit interner Rückkopplung, welches durch seine Ausgaben einen Roboter steuern soll. Von dem ebenfalls abgebildeten rekurrenten Modellnetzwerk wird angenommen, daß es ein Modell der externen Dynamik repräsentiert und C's neue Eingaben (unter Einschluß von R-Signalen) aus C's vergangenen Ein- und Ausgaben vorhersagt.

Unter Verwendung von Jordans Terminologie [21](siehe auch das Einführungskapitel zum R-Lernen) könnte man sagen, daß der Zweck von M's 'Zielknoten' darin besteht, die Aktivationen sowohl der normalen als auch der R-Knoten vorherzusagen. Nur *einige wenige* der Zielknoten, die zu den R-Knoten korrespondierenden nämlich, 'wollen' stets einen Wert von Null voraussagen. Aber *alle* Zielknoten tragen zum Lernprozeß bei, wie man im folgenden sehen wird.

M überbrückt die 'Lücke' zwischen den Ausgaben und den späteren Eingaben von C. Da M voll rekurrent ist, ist das durch M repräsentierte Umgebungsmodell potentiell so vollständig, wie es nur sein kann. Im Gegensatz zu Robinson und Fallsides Ansatz werden Lernpfade zur Verfügung gestellt, die von den R-Knoten zurück zu den Ausgabeknoten und noch weiter zurück zu allen Eingabeknoten usw. führen. Dies ermöglicht *'credit-assignment'* für Ausgabeknoten, die spätere Eingaben hervorriefen, welche ihrerseits neue Ausgaben zur Folge hatten, die schließlich zu Schmerz oder Lust führten ...

Man beachte, daß M auch die 'Lücken' schließt, die zwischen zeitlich varianten Aktivationen der Eingabeknoten selbst klaffen. So gibt es zum Beispiel 'rückwärts durch die Zeit reichende Lernpfade', die von der Aktivation eines Eingabeknoten zu einem gegebenen Zeitpunkt zurück zu früheren

Abbildung 5.1: Abgebildet ist ein rekurrentes Steuernetzwerk mit externer Rückkopplung (durch die 'WELT'). Der Übersichtlichkeit halber sind nur ein Eingabeknoten (EIN), ein Ausgabeknoten (AUS), ein Schmerz- oder Lustknoten (R) und ein versteckter Knoten zu sehen. Von dem ebenfalls vollständig rekurrenten Modellnetzwerk sind nur ein versteckter Knoten sowie ein Knoten für die Reinforcementvoraussage ($PRED_R$) und ein Knoten für die Voraussage der 'normalen' Eingabe ($PRED_{EIN}$) abgebildet. Das Modellnetzwerk dient zur Berechnung von Schmerz- bzw. Lustgradienten für das Steuernetzwerk. (Siehe Text für nähere Erläuterungen.)

Aktivationen anderer Eingabeknoten und von dort weiter zurück zu Ausgabeknoten führen. Solche Pfade braucht man, wenn die Umgebung sich verändern kann, ohne daß die letzten Ausgaben von C dafür verantwortlich waren.

Im Gegensatz zu den implementierten Ansätzen für adaptive Kritiker [5] [2] [51] [27] sowie zu den implementierten Systemidentifikationsansätzen [34] [41] sind wir nicht auf Markov-artige Umgebungen beschränkt. Präziser ausgedrückt heißt das, daß das Modellnetzwerk potentiell fähig ist, die Umgebungsdynamik sogar dann hinreichend zu repräsentieren, wenn zukünftige Eingaben nicht immer vollständig durch C's gegenwärtige Ein- und Ausgaben determiniert sind, sondern unter Berücksichtigung vergangener Ein- und Ausgaben abgeleitet werden müssen. Dadurch kann das Steuernetzwerk auch im generellen Fall etwas Vernünftiges lernen.

5.3 Begründung des Verfahrens

5.3.1 Mathematische Begründung

Wir nehmen zunächst an, daß erst M und dann C trainiert wird, und daß alle Gewichte sich nicht *während*, sondern erst *nach* einer Trainingsepisode ändern. Dann bekommen wir zwar kein *'on-line'* Verfahren, dafür aber eine mathematische Rechtfertigung der Methode. Bei der Implementierung des praktischen Algorithmus werden wir dann aus pragmatischen Gründen in verschiedener Hinsicht wieder vom off-line Ansatz abweichen und dieses Vorgehen nachträglich durch Experimente rechtfertigen.

Für M's Lernphase wird in diesem Abschnitt angenommen, daß die Ausgaben von C zufällig gewählt werden. In M's Ausgabeknoten werden zu jedem Zeitpunkt durch konventionelle Aktivationsausbreitung (in M) die Voraussagen für C's neue Eingaben (hervorgerufen durch die externe Rückkopplung oder durch von den Effektoren unabhängige Umgebungsänderungen) berechnet. Wir betrachten den Fall vorgegebener Trainingsintervallgrenzen. M's zu minimierende Fehlerfunktion ist

$$E_M = \sum_t \sum_{k \in I \cup P} (x_{k_t} - y_{k_t})^2,$$

wobei y_{k_t} die Aktivation des $k$ten vorhersagenden Knotens von M zur Zeit t ist, x_{k_t} die Aktivation des entsprechenden vorhergesagten Knotens von C ist und t über alle Zeitschritte eines Trainingsintervalls rangiert. Jedes Gewicht w_{ij} von M ändert sich dabei proportional zu

$$\frac{\partial E_M}{\partial w_{ij}}.$$

Ist M's Lernphase abgeschlossen, so werden seine Gewichte eingefroren. M und C werden zu einem einzigen großen Netzwerk konkateniert, wobei M's Ausgabeknoten mit C's Eingabeknoten identifiziert werden. Jetzt beginnt C's Lernphase. C's zu minimierende Fehlerfunktion E_C ist

$$E_C = \sum_{t,i} (c_i - y_i(t))^2.$$

Dabei ist $y_i(t)$ die Aktivation des $i$ten R-Knotens zur Zeit t, und c_i seine unveränderliche gewünschte Aktivation. Für die mathematische Begründung wird angenommen, daß E_C nur noch von W_C und W_M abhängt, nicht mehr jedoch von der Umgebung. (Das ist der Systemidentifikationsansatz in seiner allgemeinsten Form.)

Um Gradientenabstieg zu bekommen, muß ein Gewicht w_{ij} von C sich proportional zu

$$\frac{\partial E_C}{\partial w_{ij}}$$

ändern, und zwar *unter der konstanten Nebenbedingung* W_M. Abgesehen von einigen sehr plausiblen, langfristig geradezu notwendigen, im folgenden ausgeführten Abweichungen implementiert A2 unter Verwendung des IID-Algorithmus gerade den Gradientenabstieg in E_C.

5.3.2 Gründe für paralleles Lernen von Modell- und Steuernetz

Das oben ausgeführte Schema setzt voraus, daß M schon gelernt hat, ein perfekter Voraussager zu sein. Bevor C's Training beginnen kann, ist also eine explorative Lernphase für M erforderlich. Die explorative Suche, die für R-Lernsysteme normalerweise durch probabilistische Aktivationsregeln eingeführt wird (siehe Kapitel 3), steckt also in der möglichst erschöpfenden ersten Lernphase. Theoretisch müßten alle möglichen Beziehungen zwischen Ein- und Ausgaben des Kontrollnetzes und zukünftigen Eingaben exploriert werden. Das ist natürlich nicht machbar, denn im allgemeinen ist die Menge dieser potentiellen Beziehungen unendlich groß. Ein alternativer Ansatz würde darin bestehen, dem Modellnetzwerk sorgfältig ausgewählte Beispiele *typischer* Ereignissequenzen zu präsentieren, in der Hoffnung, daß M korrekt auf unbekannte Ereignissequenzen verallgemeinert. Dazu muß jedoch der externe Lehrer viel klüger sein, als wir ihn voraussetzen wollen.

Für realistische Anwendungen großen Maßstabs ist paralleles Training von M und C wünschenswert und wohl geradezu unvermeidlich: C's Gewichtsänderungen sollten bereits in Gang gesetzt werden, wenn die externe Dynamik noch *nicht* vollständig durch M's Gewichte repräsentiert ist.

M sollte sich auf diejenigen Aspekte der Umgebungsdynamik konzentrieren, die für das Erreichen von C's Zielen relevant sein könnten. Gerade so wie Kohonens selbstorganisierende Karten [23] automatisch mehr Speicherplatz für die detaillierte interne Repräsentation *häufiger* Eingaben zur Verfügung stellen, gerade so sollte M seine Speicherkapazität vorzugsweise in den Dienst der detaillierten Repräsentation derjenigen Aspekte der externen Umgebung stellen, die vermutlich relevant für das Hauptziel des Systems sind (nämlich möglichst 'lustvoll' zu existieren, ohne 'Schmerzen' zu erleiden).

Neben solchen Effizienzgründen gibt es aber auch noch weitere wichtige Gründe, parallele 'on-line' -Lernprozeduren zu studieren. Man betrachte das Problem der *Evolution von Sprache* im Fall zweier *kommunizierfähiger* Agenten, wobei jeder Agent ein Modell der Bedeutung der Ausgaben des anderen hat (später werden wir ein sich auf diese Situation beziehendes Experiment kurz beschreiben). Soll sich die Kommunizierfähigkeit der Agenten tatsächlich durch Erfahrung verbessern, so heißt das, daß sich die Ausgaben der Agenten sowie ihre Bedeutungen dynamisch verändern müssen. Dies wiederum erfordert, daß sich die jeweiligen Modelle des Gegenübers ändern *müssen*.

Robinson und Fallsides Ansatz zum parallelen Lernen wurde bereits im Kapitel 3 erwähnt und kritisiert (es sollte noch angemerkt werden, daß der Algorithmus, den sie für ihre Experimente benutzten, nicht lokal in der Zeit war). Im Kontext von Algorithmen für Markov-Umgebungen stellt auch Jordan fest, daß ein Modellnetzwerk nicht 'perfekt' sein muß, um Performanzverbesserung für ein Steuernetzwerk zu unterstützen [21].

Ist C's Fehler nicht durch den Unterschied zwischen C's gewünschter Eingabe (z. B. null Schmerz) und M's Ausgabe gegeben, sondern durch den Unterschied zwischen C's gewünschter Eingabe und C's tatsächlicher Eingabe, dann sind die Minima dieses Fehlers immer noch Fixpunkte des Gewichtsänderungsalgorithmus, solange M bereits ein lokales Minimum seines Prediktionsfehlers erreicht hat. Die Nullstellen von C's Fehler sind sogar dann schon Fixpunkte, wenn M sich noch nicht in einem lokalen Minimum gefangen hat.

Die Minima von C's Fehler lassen sich finden, *wenn die inneren Produkte der approximierten Gradienten für C's Gewichte und der exakten (mit einem hypothetischen perfekten Modellnetzwerk zu berechnenden) Gradienten dazu tendieren, positiv zu sein.*

5.3.3 A2's Abweichen vom reinen Gradientenabstieg

Natürlich muß man bei der Entscheidung zwischen einer *'on-line'* und einer off-line Prozedur einen *'trade-off'* berücksichtigen: Man bezahlt für

die gewonnene Effizienz mit dem 'Grad der mathematischen Exaktheit' des Verfahrens. Um eine *'on-line'* Lernprozedur zu erhalten, werden wir für den unten beschriebenen Algorithmus A2 in mancher Beziehung vom 'wahren' Gradientenabstieg abweichen.

1. Statt Gewichtsänderungsbeiträge zeitlich zu akkumulieren und erst nach den Aktivationsausbreitungsphasen eines Trainingsintervalls die Gewichte tatsächlich zu ändern, ändern wir die Gewichte *sofort* in jedem Zeitschritt. Dabei folgen wir Williams und Zipser [80] und nehmen an, daß die Lernraten klein genug sind, um Instabilitäten zu vermeiden. In den weiter unten beschriebenen Experimenten bestätigt sich die Zulässigkeit dieser Annahme. Sofortige Gewichtsänderungen ziehen als schöne Konsequenz nach sich, daß man nicht auf von einem externen Lehrer definierte Intervallgrenzen angewiesen ist.

2. Besonders zu Beginn einer On-Line Lernphase wird das Modellnetzwerk, welches ja zur Gradientenbestimmung für C's Gewichte dient, kein perfekter Voraussager sein. Was sind die Konsequenzen?

Man beachte, daß diejenigen Variablen von M, welche zur Speicherung von Gradienteninformation bezüglich E_M herangezogen werden müssen, *unabhängig* von den Variablen von C sind, welche Gradienteninformation bezüglich E_C speichern. Eine Situation, in welcher C 'Schmerz' erleidet, C's Gewichte jedoch nur durch ein ungenaues Modellnetzwerk 'gerechtfertigt' sind, ist nicht stabil, sofern nicht beide Netzwerke in lokalen Minima ihrer jeweiligen Fehlerfunktionen gefangen sind. *Unter der Voraussetzung*, daß M stets eine Nullstelle seiner Fehlerfunktion findet, können wir davon ausgehen, daß C nach einiger Zeit einen Gradientenabstieg gemäß einem *perfekten* Modell der sichtbaren Umgebung durchmacht. (Wie jedoch schon oben ausgeführt, kann sich C's Performanz auch mit einem nicht perfekten M schon verbessern.)

Eine theoretisch nicht geklärte Frage betrifft gerade diese Voraussetzung: Um ein gutes Umgebungsmodell zu bekommen, müssen hinreichend viele Trainingsbeispiele präsentiert werden. Eine mögliche Gefahr für ein parallel lernendes Gesamtsystem besteht darin, daß das Steuernetzwerk irgendwann in ein lokales Minimum relativ zu dem noch nicht perfekten Modellnetzwerk gerät. Dieses Minimum braucht gemessen an einem hypothetischen bereits perfekten Umgebungsmodell gar keines zu sein! Dennoch mag es dazu führen, daß das Steuernetzwerk in gegebenem Kontext stets dieselben Aktionen ausgibt, so daß das Modellnetzwerk keine Chance besitzt, etwas über die *Konsequenzen alternativer Aktionen* in Erfahrung zu bringen. Dies kann zur Folge haben, daß die Steuerer/Modell-Kombination in einen Zustand gerät, aus dem es kein Entkommen mehr gibt. Die sequentielle Version des unten beschriebenen A2 stellt einen sicheren Weg zur Umgehung zumindest dieses Problems dar, dafür sieht sie sich jedoch

mit den in 5.3.2 erwähnten Problemen konfrontiert.

Um dem Problem der gerade beschriebenen möglichen *'deadlocks'* zu begegnen, werden wir weiter unten probabilistische Ausgabeknoten für C sowie ein dazugehöriges modifiziertes Lernschema einführen.

3. Zusammenfassend kann man folgendes feststellen: Solange M in der parallelen Version von A2 bei den von C ausgesuchten 'Unterdomänen' keine akkuraten Voraussagen zu leisten imstande ist, führt C einen Gradientenabstieg in einer sich ändernden Funktion durch. Damit kann keine Garantie für sofortige Konvergenz des Verfahrens gegeben werden. Da jedoch M das Innere von C als eine 'Black Box' ansieht, macht es Sinn, darauf zu hoffen, daß M konvergiert (solange sich die Umgebung nicht chaotisch verhält). Eine für den allgemeinen Fall unbeantwortbare Frage lautet natürlich: Wie sieht es aus mit lokalen Minima für M? Eine weitere Frage lautet: Besteht ein Potential für Instabilitäten, wenn M schon gelernte Information über bestimmte Situationsfolgen 'vergißt', weil C's Aktivitäten in eine neue 'Unterdomäne' führen und langsam Gewichte überschrieben werden, die für die Modellierung der alten 'Unterdomäne' verantwortlich waren?

Dynamische Instabilitätsprobleme dieser Art scheinen mathematisch kaum angreifbar zu sein, da sie in hohem Maße domänenabhängig sind. Im experimentellen Abschnitt werden wir die Performanz der sequentiellen Version von A2 mit der Performanz der parallelen Version im Rahmen eines Nicht-Markov-Experiments vergleichen.

5.4 Der Algorithmus

In diesem Abschnitt wird die parallele Version des Algorithmus A2 (siehe auch [56] und [58] sowie die verbesserte Version in [52]) detailliert aufgeschrieben. (Die sequentielle Version ergibt sich sofort aus der parallelen Version durch geringfügige Modifikationen.)

A2 unterscheidet sich von jedem der Algorithmen anderer Autoren in wenigstens einigen der folgenden Punkte: A2 verfügt über ein Potential zum *'on-line'*-Lernen, die parallele Version ist lokal in der Zeit, A2 erlaubt sowohl interne als auch externe Rückkopplung mit (wenigstens im Prinzip) beliebigen zeitlichen Verzögerungen, er kümmert sich nicht um Trainingsintervallgrenzen, er braucht lediglich Reinforcement-artige Information zum Lernen, er erlaubt verschiedene Arten von Reinforcement oder 'Schmerz' bzw. 'Lust', und er kann ein Modell *aller* Eingaben aus der wahrnehmbaren Umgebung konstruieren, um möglichst vollständige rückwärts in die Zeit gerichtete 'Lernpfade' zur Verfügung zu stellen.

Mit anderen Worten, A2 ist tatsächlich ein sehr allgemeiner Algorith-

mus. *A2 zielt darauf ab, das fundamentale Lernproblem in der* 'on-line' *Lernsituation anzugreifen, soweit dies durch das Konzept des 'raumzeitlichen Gradientenabstiegs' überhaupt möglich ist.*

Die *on-line*-Version von Robinson und Fallsides *'Infinite Input Duration'* (IID-) Lernalgorithmus [40](siehe auch Kapitel 2) wird verwendet, um sowohl C als auch M zu trainieren. (Zur Erinnerung: Der IID-Algorithmus wurde zum ersten Mal durch Williams und Zipser getestet [80]). Wir konzentrieren uns erneut auf die Version für diskrete Zeit. Allerdings sollte es keine größeren Schwierigkeiten bereiten, A2's Prinzip mit Hilfe der Arbeiten von Pearlmutter [37] und Gherrity [10] auf kontinuierliche Zeit zu übertragen.

A2's Hauptschleife wird durch ein geeignetes domänenabhängiges Abbruchkriterium terminiert. Der Abbruch kann beispielsweise durch das Feststellen hinreichender Performanz ausgelöst werden.

Im 1. Schritt der Hauptschleife berechnet A2 die Aktionen der Effektoren des Systems. Dies geschieht einfach durch Ablesen der auf konventionelle Weise berechneten Aktivationen von C's Ausgabeknoten und ihrer Interpretation als Steuersignale für die Motorik. Für alle neuen Aktivationen werden die entsprechenden Ableitungen bezüglich aller Steuernetzgewichte auf den neuesten Stand gebracht.

Im 2. Schritt werden die im 1. Schritt berechneten Aktionen ausgeführt, und ihre Effekte (oder auch die Effekte früherer Aktionen) werden sichtbar.

Im 3. Schritt werden in M's Ausgabeknoten die Voraussagen für C's neue Eingaben (hervorgerufen durch die externe Rückkopplung oder durch von den Effektoren unabhängige Umgebungsänderungen) berechnet, wobei M C's neue Eingaben nicht kennt. Wiederum wird relevante Gradienteninformation berechnet.

Im 4. Schritt werden M's Gewichte sofort so verändert, daß in ähnlichem raumzeitlichen Kontext in Zukunft bessere Voraussagen zu erwarten sind. Für den Fall der Existenz von Trainingsintervallgrenzen wäre die von M zu minimierende Fehlerfunktion

$$E_M = \sum_t \sum_{k \in I \cup P} (x_{k_t} - y_{k_t})^2,$$

wobei y_{k_t} die Aktivation des $k$ten vorhersagenden Knotens von M zur Zeit t ist, x_{k_t} die Aktivation der entsprechenden vorhergesagten Knotens von C ist, und t über alle Zeitschritte eines Trainingsintervalls rangierte. Jedes Gewicht w_{ij} von M ändert sich dabei proportional zu

$$\frac{\partial E_M}{\partial w_{ij}}.$$

Schließlich werden C's Gewichte so geändert, daß die kumulativen Differenzen zwischen gewünschten und tatsächlichen Aktivationen der Schmerz- und Lustknoten minimiert werden unter der Annahme, daß M bereits ein guter Prophet ist. Jedes Gewicht w_{ij} von C ändert sich proportional zu

$$\frac{\partial \sum_{t,i}(c_i - y_i(t))^2}{\partial w_{ij}}$$

unter der Annahme, daß die durch W_M definierte Dynamik als Ersatz für die Umgebungsdynamik verwendet werden kann. (Das ist der Systemidentifikationsansatz.) Dabei ist $y_i(t)$ die tatsächliche und c_i die gewünschte Aktivation des $i$ten R-Knotens zur Zeit t. Da es keine Trainingsintervallgrenzen mehr gibt (alle Episoden gehen ineinander über [80]), rangiert t nun über alle vergangenen Zeitschritte. Im 4. Schritt wird auch sogenannter 'Lehrerzwang' *('teacher forcing')* [80] auf M ausgeübt. Dies bedeutet, daß der Zustand von M's Ausgabeknoten durch den neuen Zustand von C's Eingabeknoten ersetzt wird. Um mathematisch korrekt zu bleiben, müssen anschließend alle Variablen, die für M's Ausgabeknoten Gradienteninformation abspeichern, gleich Null gesetzt werden.

Der Ansatz des Lehrerzwangs ist ein natürlicher. Schließlich hängt der Fortgang von C's Aktivationsausbreitung von den tatsächlichen neuen Eingaben ab und *nicht* von M's Voraussagen. M's Dynamik wird also der 'realen' Umgebung angepaßt. Im Kontext des überwachten Lernens haben Williams und Zipser ein Experiment beschrieben, bei dem der 'Lehrerzwang' geradezu notwendig war, um befriedigende Performanz zu erzielen.

C's Fehler wird nicht durch die vorhergesagten Aktivationen von C's Eingabeknoten, sondern durch die tatsächlichen Aktivationen bestimmt. Siehe dazu auch obigen Abschnitt über paralleles Lernen sowie einige nachfolgende Kommentare zur *'on-line'* Natur von A2.

A2 bietet ein Gerüst für ein sogar noch etwas allgemeineres Schema. Er basiert auf der logistischen Funktion $f(x) = \frac{1}{1+e^{-x}}$ als Aktivierungsfunktion für alle Nicht-Eingabeknoten. Die allgemeinere Form erlaubt verschiedene Aktivierungsfunktionen für jeden Knoten, verschiedene Fehlerfunktionen für sowohl C als auch M, sowie probabilistische Ausgabeknoten für C. Zu den probabilistischen Ausgabeknoten werden gleich im Anschluß an die Beschreibung von A2 Details geliefert werden. (Weiterhin werden einige zusätzliche Kommentare folgen.) Für den Leser mag es hilfreich sein, mit dem in Kapitel 2 beschriebenen Verfahren für zeitlich lokales überwachtes Lernen zu vergleichen.
Notation :

C ist die Menge aller Nicht-Eingabeknoten des Steuernetzwerkes,

A ist die Menge seiner Ausgabeknoten,
I ist die Menge seiner 'normalen' Eingabeknoten,
P ist die Menge seiner Lust- und Schmerzknoten,
M ist die Menge aller Knoten des Modellnetzwerkes,
O ist die Menge seiner Ausgabeknoten,
$O_P \subset O$ *ist die Menge aller Knoten, die Schmerz oder Lust voraussagen,*
W_M *ist die Menge der Variablen für die Gewichte von* M,
W_C *ist die Menge der Variablen für die Gewichte von* C,
$y_{k_{new}}$ *ist die Variable für die auf den neuesten Stand gebrachte Aktivation des* k*ten Knotens von* $M \cup C \cup I \cup P$,
$y_{k_{old}}$ *ist die Variable für den letzten Wert von* $y_{k_{new}}$,
w_{ij} *ist die Variable für das Gewicht der gerichteten Verbindung vom Knoten* j *zum Knoten* i,
$p^k_{ij_{new}}$ *ist die Variable für den gegenwärtigen angenäherten Wert von* $\frac{\partial y_{k_{new}}}{\partial w_{ij}}$,
$p^k_{ij_{old}}$ *ist die Variable für den letzten Wert von* $p^k_{ij_{new}}$,
falls $k \in P$, *dann ist* c_k k*'s unveränderliche gewünschte Aktivation,*
falls $k \in I \cup P$, *dann ist kpred der Knoten aus* O, *welcher* k*'s Aktivationen voraussagt,*
α_C *ist die Lernrate des Steuernetzwerkes,*
α_M *ist die Lernrate des Modellnetzwerkes.*

$| I \cup P | = | O |$,
$| O_P | = | P |$.

Von jedem Knoten aus $I \cup P \cup A$ *führt eine Vorwärtsverbindung zu jedem Knoten aus* $M \cup C$,
jeder Knoten aus M *ist bidirektional mit jedem anderen Knoten aus* M *vernezt,*
jeder Knoten aus C *ist bidirektional mit jedem anderen Knoten aus* C *vernezt,*
jede Verbindung, die auf einen Knoten aus M *weist, gehört zu* W_M,
jede Verbindung, die auf einen Knoten aus C *weist, gehört zu* W_C,
jedes Gewicht $w_{ij} \in W_M$ *benötigt* p^k_{ij}*-Variablen für alle* $k \in M$,
jedes Gewicht $w_{ij} \in W_C$ *benötigt* p^k_{ij}*-Variablen für alle* $k \in M \cup C \cup I \cup P$.

Die parallele Version des Algorithmus sieht wie folgt aus:

INITIALISIERUNG:
Für alle $w_{ij} \in W_M \cup W_C$*:*
$w_{ij} \leftarrow$ *Zufallsgenerator,*
für alle möglichen k*:* $p^k_{ij_{old}} \leftarrow 0, p^k_{ij_{new}} \leftarrow 0$.
Für alle $k \in M \cup C : y_{k_{old}} \leftarrow 0, y_{k_{new}} \leftarrow 0.$
Für alle $k \in I \cup P$ *:*
Bestimme $y_{k_{old}}$ *durch Wahrnehmung aus der Umgebung,* $y_{k_{new}} \leftarrow 0.$

WIEDERHOLE, BIS ABBRUCHKRITERIUM ERFÜLLT:
1. Für alle $i \in C : y_{i_{new}} \leftarrow \frac{1}{1+e^{-\sum_j w_{ij} y_{j_{old}}}}.$
Für alle $w_{ij} \in W_C, k \in C$*:*
$p^k_{ij_{new}} \leftarrow y_{k_{new}}(1 - y_{k_{new}})(\sum_l w_{kl} p^l_{ij_{old}} + \delta_{ik} y_{j_{old}}).$
Für alle $k \in C$*:*
$y_{k_{old}} \leftarrow y_{k_{new}}$,
für alle $w_{ij} \in W_C$ *:* $p^k_{ij_{old}} \leftarrow p^k_{ij_{new}}$.
2. Führe alle auf Aktivationen der Knoten aus A *basierenden motorischen Aktionen aus.*
Für alle $i \in I \cup P$*:*
Bestimme $y_{i_{new}}$ *durch Wahrnehmung aus der Umgebung.*
3. Für alle $i \in M : y_{i_{new}} \leftarrow \frac{1}{1+e^{-\sum_j w_{ij} y_{j_{old}}}}.$
Für alle $w_{ij} \in W_M \cup W_C, k \in M$*:*
$p^k_{ij_{new}} \leftarrow y_{k_{new}}(1 - y_{k_{new}})(\sum_l w_{kl} p^l_{ij_{old}} + \delta_{ik} y_{j_{old}}).$
Für alle $k \in M$*:*
$y_{k_{old}} \leftarrow y_{k_{new}}$,
für alle $w_{ij} \in W_C \cup W_M$ *:* $p^k_{ij_{old}} \leftarrow p^k_{ij_{new}}$.
4. Für alle $w_{ij} \in W_M$*:*
$w_{ij} \leftarrow w_{ij} + \alpha_M \sum_{k \in I \cup P} (y_{k_{new}} - y_{kpred_{old}}) p^{kpred}_{ij_{old}}.$
Für alle $w_{ij} \in W_C$*:*
$w_{ij} \leftarrow w_{ij} + \alpha_C \sum_{k \in P} (c_k - y_{k_{new}}) p^{kpred}_{ij_{old}}.$
Für alle $k \in I \cup P$*:*
$y_{k_{old}} \leftarrow y_{k_{new}}, y_{kpred_{old}} \leftarrow y_{k_{new}}$,
für alle $w_{ij} \in W_M : p^{kpred}_{ij_{old}} \leftarrow 0,$
für alle $w_{ij} \in W_C : p^k_{ij_{old}} \leftarrow p^{kpred}_{ij_{old}}$.

5.4.1 Einführung probabilistischer Ausgabeknoten

Solange das Modell noch ungenau ist, fungiert C teilweise wie ein Zufallsgenerator, der auf uninformierte Weise Situationen verursacht, die es dem Modell ermöglichen, neue Daten über typische Ereignisabläufe in der Umgebung zu sammeln.

So wie A2 oben beschrieben wurde, stehen dem Steuernetz keine von der Umgebung unabhängigen explorativen Fähigkeiten zur Verfügung. Man könnte sagen, die Zufälligkeit wird aus der Umgebung *importiert*.

Um C's explorative Fähigkeit von der Umgebung zu emanzipieren, kann man ihm probabilistische Ausgabeknoten verpassen. Williams hat das Konzept des 'Back-Propagation durch Zufallsgeneratoren' eingeführt [77]. Er beschrieb Knoten mit einer durch eine kontinuierliche differenzierbare Wahrscheinlichkeitsverteilung gegebenen Aktivierungsfunktion, deren Mittelwert und *Varianz* durch einen BP-Prozeß adjustiert werden können. Mit solch einem Konzept können unabhängige explorative Möglichkeiten mit A2 konsistent gemacht werden. Dies erfordert im Kontext des IID-Algorithmus allerdings folgende Modifikationen:

Ein probabilistischer Ausgabeknoten k besteht aus einem konventionellen Knoten $k\mu$, der als Mittelwertgenerator fungiert, sowie einem weiteren konventionellen Knoten $k\sigma$, welcher als Varianzgenerator fungiert. Zu einem gegebenen Zeitpunkt wird $y_{k_{new}}$ berechnet durch

$$y_{k_{new}} = y_{k\mu_{new}} + z y_{k\sigma_{new}},$$

wobei z beispielsweise eine normalverteilte Zufallsvariable darstellt. Um den gewünschten Gradientenabstieg zu ermöglichen, müssen die korrespondierenden $p^k_{ij_{new}}$ nach folgender Regel berechnet werden:

$$p^k_{ij_{new}} \leftarrow p^{k\mu}_{ij_{new}} + \frac{y_{k_{new}} - y_{k\mu_{new}}}{y_{k\sigma_{new}}} p^{k\sigma}_{ij_{new}}.$$

5.4.2 Kommentare zum Algorithmus.

1. A2 ist lokal in der Zeit, aber nicht im Raum. A2's Berechnungsaufwand pro Zeitschritt ist durch

$$O(\mid W_M + W_C \mid\mid M \mid\mid M \cup I \cup P \cup A \mid + \mid W_C \mid\mid C \mid\mid I \cup P \cup C \mid)$$

gegeben.

2. Man beachte, daß die akkumulative Berechnung der $\frac{\partial y_{k_{new}}}{\partial w_{ij}}$-Variablen für C's Gewichte keine Information über entsprechende Variablen von M benötigt. Allerdings wird Wissen über M's Knoten*aktivationen* benötigt.

3. Für C wird natürlicherweise *kein* Lehrerzwang verwendet. Lehrerzwang würde nämlich bedeuten, selbst dann mit Nullaktivation für die R-Knoten fortzufahren, wenn es unerwünschte Aktivationen gab. Die Idee ist hier, daß Lust und Schmerz für den Agenten durchaus informativ sein kann. Lust und Schmerz sollten expliziten Einfluß auf zukünftige Aktionen haben können.

4. A2 in der oben beschriebenen Form geht davon aus, daß sich die Umgebung von einem Zeitschritt zum nächsten stets auf eine Weise ändert, die durch linear separable Abbildungen der internen Repräsentation vergangener Zustände auf die nächsten Eingaben beschreibbar ist. Gehorcht die Umgebung jedoch einem 'höheren Grad von Nichtlinearität', muß der Algorithmus so modifiziert werden, daß die Netzwerke mit einer höheren Frequenz 'ticken' als die Umgebung. Dies kann auf triviale Weise durch jeweils mehrere Iterationen (pro Zeitschritt) der Schritte 1 und 3 erreicht werden. Theoretisch genügt es in jedem Fall, wenn pro 'Umgebungszeitschritt' 4 'Netzwerkzeitschritte' ablaufen. Das liegt daran, daß 4-Lagen-Operationen im Prinzip ausreichen, um jede gewünschte nicht-lineare Abbildung mit beliebiger Genauigkeit zu approximieren [24].

5.5 Experimente zum R-Lernen in Nicht-Markov-Umgebungen

Im Gegensatz zu allen anderen bisher implementierten modell-bildenden R-Lernalgorithmen besitzt A2 ein theoretisches Potential für das Lernen in Nicht-Markov-Umgebungen. Läßt sich dieses Potential in der Praxis verwirklichen, insbesondere wenn das mathematisch exakte Verfahren zugunsten der *on-line*-Methode aufgeweicht wird?

5.5.1 Evolution eines Flip-Flops durch R-Lernen

Dieser Abschnitt zeigt anhand des ersten mir bekannten Experiments zum R-Lernen in einer Nicht-Markov-Umgebung (einem 'Flip-Flop-Experiment'), daß die Antwort auf die einführende Frage 'ja' lautet [**?**]. Programmierung und Tests wurden von Josef Hochreiter im Rahmen eines Fortgeschrittenenpraktikums durchgeführt [15].

Das Steuernetzwerk C sollte lernen, sich analog zu einem Flip-Flop zu verhalten. C's Aufgabe bestand dabei darin, auf einem kontinuierlichen Strom von Eingabeereignissen in folgender Weise zu reagieren: Wann immer das Ereignis 'B' zum ersten Mal nach dem Ereignis 'A' stattfand, sollte die Aktion 'Eins' ausgegeben werden, in allen anderen Fällen die Aktion 'Null'. Ein ähnliches Experiment wurde von Williams und Zipser im

Kontext des *überwachten* Lernens zur Demostration der Fähigkeiten des IID-Algorithmus verwendet [80].

Eine Schwierigkeit bestand darin, daß die Ereignisse in der Umgebung auf zufällige Weise generiert wurden und *beliebig* lange Verzögerungen zwischen relevanten Ereignissen auftreten konnten. (Da haben wir gerade die Nicht-Markov-Eigenschaft.) C mußte also *lernen*, sich das Eintreten des Ereignis 'A' über einen im Prinzip beliebig langen Zeitraum zu merken, nachfolgende unter Umständen irrelevante Ereignisse zu ignorieren und beim Eintreten von 'B' in korrekter Weise zu reagieren sowie die 'Speicherung' von 'A' rückgängig zu machen.

Eine weitere Schwierigkeit für das lernende System bestand darin, daß es keine Trainingsintervallsgrenzen gab. Auf diese Weise konnten durch frühere Ereignisse verursachte Aktivationsspuren einen störenden Einfluß auf Aktivationszustände während späterer 'Versuche' nehmen.

Die Hauptschwierigkeit bestand natürlich (wie stets beim R-Lernen, hier liegt der wesentliche Unterschied zu Williams und Zipsers Experiment) in der Abwesenheit eines informativen Lehrers. C wurde seine gewünschte Ausgabe niemals von externer Seite mitgeteilt, vielmehr mußte C in einer Vielfalt raum-zeitlicher Kontexte alternative Aktionen ausprobieren und sich ein dynamisches Modell der entsprechenden unter Umständen 'schmerzhaften' Konsequenzen verschaffen. Die einzige Zielinformation für C bestand zu einem gegebenen Zeitpunkt nämlich in der Aktivation eines Schmerz*eingabe*knotens proportional zum Unterschied zwischen C's jeweils letzter (reellwertigen) Aktion aus dem Intervall $[0 \ldots 1]$ und der von der Flip-Flop-Umgebung erwünschten Aktion. (Der Proportionalitätsfaktor betrug 0.5.)

In Abbildung 5.2 ist die Topologie der interagierenden rekurrenten Netze für diese Aufgabe dargestellt. C besaß 5 Eingabeknoten, darunter 3 'normale' Eingabeknoten für 3 mögliche lokal repräsentierte Ereignisse 'A', 'B', 'D'. Das Eintreten eines dieser Ereignisse wurde dadurch repräsentiert, daß der entsprchende Knoten mit einem Wert von 1.0 aktiviert wurde und die anderen Eingabeknoten ausgeschaltet wurden. Weiter gab es einen stets mit dem Wert 1.0 aktivierten 'wahren' Eingabeknoten (für modifizierbare 'Schwellwerte') sowie einen Schmerzknoten. C's Ausgabeknoten war probabilistisch und bestand seinerseits aus 3 Subknoten, 2 davon zur Mittelwert- und Varianzgenerierung. Der Beitrag des Varianzknotens bestand in seiner gegenwärtigen Aktivation multipliziert mit

$$ln(\frac{1 - rnd}{rnd})$$

(hier kommt die Umkehrfunktion der logistischen Funktion ins Spiel), wobei rnd eine gleichverteilte Zufallsvariable mit Werten zwischen 0.2 und 0.8

war. Die Aktivation des Ausgabeknotens ergab sich schließlich zu 0, falls die Summe s des Beitrags des Varianzknotens und der Aktivation des Mittelwertknotens kleiner als 0 war, sie ergab sich zu 1, falls $s > 1$ galt und zu s sonst. Das Modellnetzwerk M besaß 3 versteckte Knoten sowie einen Ausgabeknoten zur Vorhersage der Aktivationen des Schmerzknotens. (Wegen der zufälligen Natur der anderen Eingabeknoten von C wurden diese nicht von M vorhergesagt, um Rechenzeit zu sparen.)

Es wurden Testläufe sowohl für die parallele Version als auch für die sequentielle Version von A2 veranstaltet. In beiden Fällen führte C stets *eine* Neuberechnung seiner Aktivationen pro Umgebungszeitschritt aus. M, dessen Aufgabe in gewissem Sinne komplizierter war als C's Aufgabe (es mußte ja auch die Konsequenzen von C's Fehlern modellieren), wurden *zwei* Iterationen pro Umgebungszeitschritt gestattet (Schritt 3 des Algorithmus).

Bei der sequentiellen Version wurde zunächst M für 150.000 Zeitschritte mit Hilfe von zufällig gewählten gleichverteilten Steuernetzwerkausgaben trainiert. Nach dem anschließenden Einfrieren von M's Gewichten wurden weitere 50.000 Zeitschritte auf das Training von C verwendet. Sowohl α_M als auch α_C betrugen 1.0. *In 10 Versuchen fand der Lernalgorithmus stets eine brauchbare rekurrente Gewichtskonfiguration für das Steuernetzwerk.* (Um zu entscheiden, ob eine Gewichtskonfiguration 'brauchbar' war oder nicht, wurde C's Gewichtsmatrix nach der Lernphase 'manuell' inspiziert und ihre Arbeitsweise nachvollzogen.)

Wie zu erwarten war, benötigte A2 wesentlich mehr Zeit für die Lösung des Flip-Flop-Problems als der *überwachte* IID-algorithmus: Gibt man nämlich den Ausgabeknoten die Lösung vor, so ist die entsprechende überwachte Lernaufgabe innerhalb weniger 1000 Zeitschritte zu lösen [80]. Der Grund dafür liegt natürlich darin, daß der gesuchte Steuergradient beim R-Lernen nicht *vorgegeben* ist, sondern vom Lernsystem erst 'entdeckt' werden muß.

Auch mit der parallelen Version konnte das Problem gelöst werden. Es erwiesen sich allerdings längere Lernzeiten als erforderlich. Mit $\alpha_M = 1.0$ und $\alpha_C = 0.2$ führten von 30 Testläufen nur 20 innerhalb von 1.000.000 Zeitschritten zum Erfolg.

Die Experimente zeigen, daß zumindest bei relativ kleinen Problemen die sequentielle Version gegenüber der parallelen Version vorteilhaft abschneiden kann. Andererseits liefern sie erstmalig den Nachweis, daß durch die Einführung der probabilistischen Ausgabeknoten sowie durch die entsprechende Modifikation des Lernverfahrens tatsächlich parallel gelernt werden kann, was keineswegs von vornherein klar war. Auch im Kapitel 7 wird die Einführung probabilistischer Ausgabeknoten das parallele Lernen eines Steuer- und eines Modellnetzes ermöglichen. Allerdings ist auch dort die Anwendungsdomäne nicht komplex genug, um die hypothetisch zu erwartenden Effizienzvorteile parallelen Lernens experimentell zu verfizieren.

Abbildung 5.2: Topologie der Steuernetz/Modellnetz-Kombination für das Flip-Flop Experiment. Der Balken unten stellt die Umgebung dar. Die gepunktete Linie umfaßt die zum probabilistischen Ausgabeknoten gehörigen Subknoten für die Mittelwert- und Varianzgenerierung. Siehe Text für nähere Erläuterungen.)

Effizienzgewinn sollte man ja vor allem dann erwarten, wenn es im Rahmen einer gestellten Aufgabe auch viel Irrelevantes zu lernen gibt.

Um sich mehr Klarheit über Vor- und Nachteile beider Versionen zu verschaffen[1], wären Experimente mit komplizierteren und vielschichtigeren Problemen sowie den dazugehörigen größeren Netzen notwendig. Allerdings stießen wir bereits mit dem obigen relativ einfachen Experiment an die Grenzen des auf unserer Maschine noch erträglichen Berechnungsaufwands. Die sequentielle Simulation spielte sich auf einer SUN SPARC Station ab, ein einziges Flip-Flop-Experiment verbrauchte dabei ca. einen Tag Rechenzeit. Mit einer entsprechenden voll parallelen Simulation (ein Prozessor für jede p_{ij}^k-Variable) ließe sich der Zeitaufwand allerdings um mehrere Größenordnungen drücken, und zwar sogar um eine *mit der Größe der Netzwerke wachsende* Zahl von Größenordnungen.

Modifizierte man A2 so, daß C's Fehler nicht durch den Unterschied zwischen C's gewünschter Eingabe (null Schmerz) und C's tatsächlicher Eingabe gegeben war, sondern durch den Unterschied zwischen C's gewünschter Eingabe und M's entsprechender Vorhersage, so war *keine* signifikante Performanzverbesserung möglich. Das lag daran, daß sich M niemals zu einem wirklich perfekten Propheten entwickelte. Oft sagte M nämlich ungerechtfertigterweise zwar schwache, aber doch langfristig schädliche Konsequenzen zur Folge habende Schmerzaktivationen voraus. Es ist wichtig für A2, sich an den *realen* (statt an den vorhergesagten) Fehler zu halten.

5.5.2 Nicht-Markov-mäßiges Balancieren mit 'perfektem Modell'

Statt ein Modellnetzwerk daraufhin zu trainieren, die Umgebung zu simulieren, kann man in gewissen Fällen ein 'perfektes Modell' durch eine mathematisch geeignete Darstellung der Umgebungsdynamik gewinnen. Das spart die Rechenzeit für das Lernen des Modells. Ein derartiger Fall kann z.B. eintreten, wenn die Umgebung in Form von Differentialgleichungen beschreibbar ist. Im Kontext des obigen Algorithmus A2 bedeutet dies lediglich, statt für die Knoten des nicht mehr notwendigen Modellnetzes nun für jede relevante Zustandsgröße $\eta(t)$ des zu modellierenden Systems und für jedes Gewicht w_{ij} von C eine $p_{ij_{new}}^{\eta}$-Variable für die Akkumulation der angenäherten Werte von $\frac{\partial \eta(t)}{\partial w_{ij}}$ mitzuführen. Diese Variablen müssen durch Differentation der die Umgebung beschreibenden Gleichungen gesetzt werden. Die über die Zeit hinweg kumulativen Einflüsse von C's Gewichten auf die Zustandsvariablen müssen analog zu den kumulativen Beiträgen für M's Knoten (beim Algorithmus A2) in Gewichtsänderungen umgesetzt werden.

[1] Zumindest ist die parallele Version ästhethisch reizvoller als die sequentielle.

Im Rahmen seines Fortgeschrittenenpraktikums an der TUM implementierte Josef Hochreiter auch diese Modifikation für den speziellen Fall eines nicht-Markov-mäßigen Balancierproblems [15].

Dabei ging es für C darum, zu jedem Zeitschritt auf einen sich auf einer eindimensionalen Spur befindlichen Wagen eine Kraft auszuüben, und zwar dergestalt, daß ein auf dem Wagen montierter und entlang der Bewegungsrichtung des Wagens frei schwingender Stab nicht zu weit ausschlug. Auch durfte der Wagen nicht an die Begrenzung der Spur stoßen.

Um die Schwierigkeiten bei unserer Balancieraufgabe richtig würdigen zu können, wird im folgenden noch einmal verdeutlicht, wie man lernenden Netzwerken in der Vergangenheit für ähnliche Aufgaben Hilfestellung gegeben hat, und welche Art von Lehrinformation bei unserem Experiment *nicht* zur Verfügung gestellt wurde.

Im einfachsten Fall gibt ein Lehrer für jeden Zeitpunkt und für jeden Zustand des zu kontrollierenden Systems die Aktivationen derjenigen Ausgabeknoten vor, die zur Steuerung der Kraftausübung verwendet werden. Dazu reichen simple Musterassoziationsalgorithmen.

In einem schwierigeren Fall gibt ein Lehrer für jeden Zeitpunkt nur Eigenschaften des gewünschten Zustandes des Wagen/Stab-Systems an (z.B. wird von Widrow für jeden Zeitpunkt ein gewünschter Winkel von 0 Grad angegeben). Der vollständige gegenwärtige Umgebungszustand ist dem System über seine Eingabeknoten zugänglich. Mit purem überwachten Lernen ist diese Aufgabe nicht mehr zu lösen, zumindest hat man aber kein *zeitliches* Lernproblem.

Daher wurde für das in diesem Abschnitt beschriebene Experiment wieder eine *nicht* Markov-mäßige Umgebung gewählt: *Zu keinem Zeitpunkt wurde Information über die zeitlichen Ableitungen der wesentlichen Zustandsvariablen z und θ zur Verfügung gestellt.*

C bestand aus 8 Nicht-Eingabeknoten und 3 Eingabeknoten. Die Eingabeknoten wurden als 'normale' Eingabeknoten für die beiden sichtbaren, skalierten, asymmetrischen, im Anhang definierten Zustandsvariablen $\bar{z}, \bar{\theta}$ sowie für einen stets aktiven 'wahren' Knoten verwendet. Einer der Nicht-Eingabeknoten (genannt Knoten o) diente als Ausgabeknoten. Im 2. Schritt von A2 wurde stets eine parallel zur Spur gerichtete Kraft von $(50 y_{0_{new}} - 25)$ Newton auf den Schwerpunkt des Wagens ausgeübt. Der zu minimierende 'Schmerz' betrug

$$\frac{1}{2}((\frac{\theta}{0.21})^2 + (\frac{z}{2.4})^2).$$

Da $+/-0.21$ die maximale Auslenkung des Stabes und $+/-2.4$ die maximale Abweichung des Wagens von der Mitte der Spur war, war der Gesamtschmerz also maximal 1.

C führte eine Aktivationsneuberechnung pro Umgebungszeitschitt aus. M wurde wie oben beschrieben durch ein 'perfektes Modell' ersetzt. Da es im Gegensatz zum Flip-Flop-Experiment nicht notwendig war, *beliebig* weit in die Vergangenheit sehen zu können, wurden alle p_{ij}^k-Variablen des Systems bei jedem 8. Zeitschritt auf Null gesetzt. (Dies entspricht einer maximalen Rückschau von 8 Zeitschritten.) Zu Beginn wurden alle Gewichte zufällig zwischen -0.1 und 0.1 initialisiert. Im Schritt 2 des modifizierten A2 wurden die im Anhang beschriebenen Differentialgleichungen zur Modellierung des Wagen/Stab-Systems herangezogen. Die Rechnersimulation beruhte auf der Eulerschen Methode, zwei aufeinanderfolgende Zeitschritte waren durch 0.02s getrennt, zur exakteren Simulation wurde die Euler-Methode allerdings mit einer Frequenz von 100 Hertz angewandt. Für den allerersten Zeitschritt sowie für jeden neuen Versuch wurde gemäß folgender Prozedur ein neuer Initialzustand für das physikalische System generiert: Die Wagenposition wurde zufällig anhand einer Gleichverteilung aller möglichen Positionen bestimmt. Der Winkel des Stabs mit der Vertikalen wurde zufällig anhand einer Gleichverteilung aller Werte aus dem Intervall $[-0.1 \ldots + 0.1]$ bestimmt. Die Ableitungen der beiden sichtbaren Zustandsvariablen wurden jeweils mit Null vorbesetzt.

Zu Beginn jedes Testlaufs (nach der Gewichtsinitialisierung) wurden zu Vergleichszwecken 100 Versuche ohne Lernen durchgeführt. Daraufhin wurden 1000 Lernversuche mit $\alpha_C = 1.0$ gestartet. Nach jedem Lernversuch wurde C's Gewichtsmatrix zu Testzwecken eingefroren und in 100 Versuchen die durchschnittliche Balancierzeit ermittelt. (Die zu Beginn jedes Versuchs eingeführte Zufälligkeit hatte manchmal einen Beinahe-Fehlerzustand zur Folge, welcher es unmöglich machte, einen langen Lauf zu erzielen.) Die dem höchsten Durchschnitt entsprechende Gewichtsmatrix wurde erneut anhand von 100 Versuchen (mit ausgeschaltetem Lernmechanismus) getestet. Es stellte sich heraus, daß eine deutliche Performanzverbesserung erzielt werden konnte. So wurden bei 17 von 20 Testläufen innerhalb von wenigen 100 Trainingsepisoden Läufe von mehr als 1000 Zeitschritten erreicht. Die durchschnittliche Balancierzeit der besten Gewichtsmatrix eines Laufs stieg des öfteren von ca. 23 Zeitschritten auf mehr als das Dreißigfache. Begann man einen Testversuch mit 'freundlichen' Anfangsbedingungen, so wurden Spitzenbalancierdauern von 3000 und teilweise sogar wesentlich mehr Zeitschritten erreicht [54].

5.5.3 Vorschlag für ein Experiment zur Evolution von Sprache

In naher Zukunft soll folgendes Experiment in Angriff genommen werden: Zwei kommunizierende Agenten, beide ausgestattet mit A2, sind nicht nur

in der Lage, ihre physikalische Umwelt zu manipulieren, sondern können auch 'akustische' Ausgaben produzieren und 'akustische' Eingaben verarbeiten. Jeder Agent besitzt ein adaptives Modell der 'Bedeutung' der Ausgaben des anderen. 'Bedeutung' ist dabei nur implizit durch einen evaluativen Kritiker gegeben, der die Effekte der Aktionen der beiden Agenten beurteilt. Spezielle Aufgaben sollen den Agenten gestellt werden, und zwar derart, daß eine Aufgabe nicht von einem einzelnen Agenten alleine gelöst werden kann. Dadurch ergibt sich eine Notwendigkeit für Kooperation: Die Agenten sollen gezwungen sein, zu lernen, in sequentieller Manier zu kommunizieren.

5.6 Abschließende Bemerkungen

5.6.1 Umgebungsmodelle zum Planen von Handlungssequenzen

'Planen' und 'planende Vorausschau' galt bis vor kurzem als etwas, das im Rahmen des maschinellen Lernens höchstens 'symbolorientierten' AI-Programmen vorbehalten war. In diesem Abschnitt werden wir jedoch kurz ausführen, wie mindestens *ein* 'konnektionistischer' Lernalgorithmus (nämlich A2) zum Planen von Handlungssequenzen verwendet werden kann. (Ein ganz unterschiedlicher, aber weniger allgemeiner Ansatz zur planenden Vorausschau findet sich in [68]).

Obwohl A2 normalerweise zum Zeitschritt t nur eine Voraussage für den Zeitschritt $t+1$ leistet, ist die Kombination aus C und M auch für beliebig weit in die Zukunft reichende Vorausschau nützlich. Die Zukunft kann durch den leicht modifizierten A2 'mental' simuliert werden. Man braucht nur C's Eingabeknoten von der Umgebung abzukoppeln und die Umwelteingaben durch die jeweiligen Voraussagen des die Umwelt simulierenden Modellnetzwerkes zu ersetzen. Läßt eine mentale Simulation Schmerz erwarten, *so kann A2 Gradientenabstieg im simulierten Schmerz durchführen*, ohne daß der Agent tatsächlich Schmerz erleidet. Es kann eine sofortige Entscheidung über die Änderung zukünftigen Verhaltens getroffen werden.

Man sieht, daß planende Vorausschau und rückwirkendes Lernen sehr viel miteinander gemeinsam haben. Die Ähnlichkeit zwischen diesen Prozessen ist so groß, daß sie im wesentlichen als verschiedene Aspekte ein und desselben Prozesses angesehen werden dürfen.

Beim *'on-line'* Lernen besteht das Problem mit der planenden Vorausschau darin, daß eine große Spitzenberechnungskapazität zur Extraktion von Information über zukünftige Ereignisse vonnöten ist. Nehmen wir zum Beispiel an, daß der Agent zu bestimmten Zeitpunkten 10 Zeitschrit-

te in die Zukunft blickt, so konsumiert er $10 * m$ soviel Berechnungszeit, als wenn er auf mentale Simulation verzichten würde. (m ist hierbei die Anzahl sukzessiver Simulationswiederholungen, die für die Konvergenz der Gradientenabstiegsprozedur notwendig sind.)

Die Methode der adaptiven Kritiker (siehe das nachfolgende Kapitel) ist da in einem gewissen Sinne sympathischer: Adaptive Kritiker repräsentieren kein perfektes Modell aller möglichen Umgebungsereignisse, sondern lediglich ein Modell gewisser *relevanter* Aspekte von Umgebungsereignissen. Während aufeinanderfolgender Trainingszyklen versuchen sie, Erwartungen über unter Umständen weit in der Zukunft liegende Ereignisse 'zurück in die Vergangenheit zu schieben'.

Ein Problem der adaptiven Kritiker ist natürlich, daß sie in der Regel viele aufeinanderfolgende Trainingszyklen brauchen, bis Erwartungen über die Qualität zukünftiger Ereignisse dort sitzen, wo sie sitzen sollen. Außerdem müssen die relevanten Ereignisse vom Programmierer vordefiniert werden. (Im vorangegangen Kapitel bestand der modellierte Aspekt der Umgebung stets im kumulativen über die Zeit hinweg zu erwartenden Reinforcement. Sutton und Pinette [69] haben in verwandter Weise modelliert, wie oft ein Markov-Prozeß in jeden seiner möglichen Zustände eintreten wird.) Das nächste Kapitel versucht unter anderem, Vorteile der adaptiven Kritiker mit Vorteilen des Systemidentifikationansatzes zu verschmelzen.

Im allgemeinen Fall erhebt sich das Problem, *welche* zukünftigen Ereignisse fürs Planen relevant sind und welche nicht. *Damit sind wir auch bei konnektionistischen Algorithmen beim altbekannten* frame-problem *aus der konventionellen AI.*

5.6.2 Sichtweise: Ziele nach Programmen differenzieren

Im letzten Abschnitt wurde gesehen, daß man mit konnektionistischen Algorithmen bereits in Bereiche vorstoßen kann, die bis vor kurzem noch konventioneller AI vorbehalten schienen. Hier stellen wir nun einige Betrachtungen zu den Unterschieden zwischen konventionellen AI-Methoden und den oben vorgestellten Methoden an.

Man betrachte ein Netzwerk mit vorgegebener Topologie und vorgegebenen Aktivierungsfunktionen als einen Rechner. Sein *Programm* ist die Gewichtsmatrix. Eine der interessantesten Eigenheiten vieler neuronaler Netze besteht darin, daß die Netzausgaben nach der Gewichtsmatrix differenzierbar sind. Damit sind also *Programmausgaben nach Programmen differenzierbar.* Ein einfacher *Programmgenerator*, nämlich die Gradientenabstiegsprozedur, erlaubt die Generierung zunehmend erfolgreicherer Programme, falls die gewünschten Netzausgaben bekannt sind.

In für R-Lernen typischen Situationen ist die Umgebung *nicht a priori* in differenzierbarer Form repräsentiert. A2 zieht seine Existenzberechtigung aus seiner Fähigkeit, *die Umgebung durch ein differenzierbares Umgebungsmodell zu ersetzen.* Die Gewichtsmatrix des Umgebungsmodells dient ihrerseits als Programm, dessen Eingaben die Ein- und Ausgaben des zu verbessernden 'Hauptprogramms' (der Steuermatrix) sind. Mit Hilfe des Umgebungsmodells (und der Kettenregel) werden sogar gewisse Programm*eingaben* nach dem Steuerprogramm differenzierbar. Das differenzierbare Umgebungsmodell erlaubt also dem Programmgenerator eine informierte Suche nach zunehmend besseren Programmen. (Im 8. Kapitel gehen wir sogar noch einen Schritt weiter, dort machen wir Programmeingaben differenzierbar in Bezug auf *Programmnamen.*)

Der Grad der Informiertheit dieser Suche ist der gewichtigste Unterschied zwischen A2's Arbeitsweise und den Arbeitsweisen anderer Algorithmen für R-Lernen. A2 braucht zwar mehr Spitzenberechnungsaufwand als zum Beispiel die völlig lokale neuronale Eimerkette. Dank seiner informierteren Suche nach adäquaten Programmen kann von A2 jedoch erwartet werden, daß er weniger *Gesamt*berechnungsaufwand für viele nicht-triviale Probleme benötigt. Z.B. war das von A2 gelöste Flip-Flop-Problem von der neuronalen Eimerkette überhaupt nicht lösbar.

5.6.3 Kritik und Ausblick

Obwohl A2 einen allgemeinen Performanzgradienten approximiert und zielgerichtet ausnützt (und mehr kann, als ich mir zu Beginn meiner Arbeit hätte träumen lassen), gibt es immer noch Grund zur Unzufriedenheit. A2 ist nicht lokal im Raum: Mit steigender Netzwerkgröße wächst der Berechnungsaufwand pro Zeitschritt viel schneller als die Anzahl der Verbindungen. Weiterhin ist A2 nicht kompositionell und nicht selektiv in seiner Modellbildung. Die nächsten Kapitel liefern daher weitere Beiträge zur Lösung dieser Probleme.

Kapitel 6

Mehrdimensionale adaptive Kritiker & zyklische Netze

6.1 Einführung

Aufbauend auf den bisher dargestellten Grundlagen beschreiben wir in diesem Kapitel eine weitere Klasse von Lernverfahren für Netzwerke mit interner und externer Rückkopplung. Im Gegensatz zu den bisher beschriebenen Methoden bedienen sich die im folgenden vorgestellten Ansätze eines *adaptiven Kritikers*, um die zeitliche Evolution von Zuständen dynamischer Netzwerke zu beobachten und zu bewerten.

Die Algorithmen können als Reinforcementvergleichsalgorithmen (siehe Kapitel 3) für i. a. zyklische Netze angesehen werden. Dieses Kapitel stellt eine Erweiterung der beiden Ph.-D.-Arbeiten von Sutton und Anderson dar [66][2]. Zwei Arten von Erweiterungen werden angesprochen: Erstens werden unsere Kritiker im allgemeinen *multidimensional* sein (im Gegensatz zu den bisher verwendeten *skalaren* Kritikern), und zweitens werden Strategien zur adaptiven Kritik *rekurrenter* Netze eingeführt werden.

Wichtige Motivation ist wieder die wünschenswerte Lokalität in Zeit und Raum. Zwar ist das generellste der vorgestellten Verfahren nicht räumlich schwach lokal; alle Verfahren sind jedoch zeitlich schwach lokal. Dort, wo wir uns in besonderem Maße von Andersons Arbeit abheben wollen, ist räumliche und zeitliche Lokalität gegeben.

Der wesentliche Beitrag dieses Kapitels besteht in der Demonstrati-

on der Anwendbarkeit von TD-Varianten auf rekurrente Netze sowie der Demonstration von Lerngeschwindigkeitsvorteilen mehrdimensionaler Kritiker. Alle vorgestellten Algorithmen basieren im wesentlichen auf zwei interagierenden Netzwerken. Ein statischer oder dynamischer adaptiver Kritiker beobachtet die temporale Evolution eines u.U. vollständig rückgekoppelten "Steuernetzwerkes", welches durch seine Ausgaben eine Umgebung manipulieren kann. Der Kritiker lernt, zu jedem Zeitpunkt Voraussagen über den Endeffekt der vom Steuernetzwerk ausgeführten Prozedur zu machen. Unterschiede sukzessiver Voraussagen dienen einerseits dazu, den Kritiker zu verbessern, und andererseits dazu, korrespondierende Transitionen im Steuernetzwerk wahrscheinlicher oder unwahrscheinlicher zu machen, je nachdem, ob sich die Erwartung bezüglich eines gewünschten Endeffekts verbessert oder verschlechtert hat.

Das Kapitel ist wie folgt gegliedert: Nach einer intuitiven Erklärung des allen Verfahren gemeinsamen Prinzips wird zunächst ein auf diesem Prinzip beruhender Algorithmus namens A3 beschrieben. Experimentell wird gezeigt, wie ein *linearer* Kritiker zur Lösung einer *nicht-linearen* Aufgabe beitragen kann.

Anschließend werden Varianten des Algorithmus beschrieben. Unter anderem wird gezeigt, wie ein drittes adaptives Netzwerk sinnvoll den Systemidentifikationsansatz mit ins Spiel bringen kann. Eine simplifizierte Version dieser Erweiterung, bei der ein Systemidentifikationsnetz in sinnvoller Weise mit dem adaptiven Kritiker verschmolzen wird, erlaubt im Gegensatz zu bisherigen Methoden die natürliche Einführung *vektorwertiger* Kritiker. Experimentell wird anhand einer schwierigen Balancieraufgabe gezeigt, daß mehrdimensionale Kritiker höhere Lerngeschwindigkeiten erlauben als Ansätze mit skalaren Kritikern.

Zum Abschluß wird eine Erweiterung auf den Fall eines rekurrenten Kritikers angegeben. Letztere ist zwar nicht mehr lokal im Raum, dafür aber für allgemeine Umgebungsdynamik geeignet, auch wenn letztere nicht vom Markov-Typ ist.

6.2 Intuitive Erklärung des Grundprinzips

In Kapitel 3, Abschnitt 2.2 wurden die Grundlagen der adaptiven Kritiker und der Reinforcementvergleichsverfahren beschrieben. In diesem Kapitel wenden wir das Reinforcementvergleichsverfahren auf zyklische Netze an.

Wir betrachten ein vollständig rekurrentes Steuernetzwerk C, dessen Knoten ihre Aktivationen mittels stochastischer Aktivierungsfunktionen bestimmen. C bekommt Eingaben von der Umgebung und produziert u.U. umgebungsverändernde Ausgaben. Somit kann neben der *internen* in der

Regel auch *externe* Rückkopplung existieren. Zu jedem Zeitpunkt ändert sich C's Zustand im allgemeinen also abhängig sowohl von seinem alten Zustand als auch abhängig vom Umgebungszustand. Zu jedem Zeitpunkt erhält ferner der Kritiker den vollständigen Aktivierungszustand aller Knoten des Steuernetzwerkes als Eingabe. Weiterhin macht er zu jedem Zeitpunkt eine Voraussage über das noch ausstehende in Zukunft zu erwartende externe Reinforcement des sich im Programmablauf befindlichen Steuernetzwerkes.

Vergibt der externe Kritiker zu einem bestimmten Zeitpunkt Reinforcement, so werden die Gewichte des Kritikers in trivialer Weise abhängig von C's letztem Zustand dergestalt geändert, daß in ähnlicher Situation das nächste Mal eine bessere Voraussage zu erwarten ist (Terminierung von Samuels Prinzip, siehe Kapitel 3). Vergibt der externe Kritiker zu einem bestimmten Zeitpunkt *kein* Reinforcement, so dient (wieder abhängig von C's Zustand) die *neue* Voraussage des Kritikers zur Verbesserung seiner *früheren* Voraussage durch entsprechende Adjustierung seiner Gewichte (Nichtterminierungsfall von Samuels Prinzip). Unterschiede aufeinanderfolgender Voraussagen dienen gleichzeitig zur Änderung der Wahrscheinlichkeiten bestimmter Transitionen im rekurrenten Netzwerk C.

Man könnte sagen, daß Erwartungen über den Ausgang von Programmabläufen während aufeinanderfolgender Lernzyklen *'zurück in die Vergangenheit transportiert werden'*. Voraussagen werden mit Systemzuständen assoziiert, die unter Umständen zeitlich sehr stark getrennt von denjenigen Zuständen sind, auf die sich die Voraussage bezieht. Der Kritiker lernt mit der Zeit, weit in die Zukunft zu blicken, obwohl er zu *jedem* Zeitpunkt ausschließlich *zeitlich lokale* Operationen durchführt. (Natürlich ergibt die Methode nur dann Sinn, wenn die Umgebung sich regelmäßig verhält, wenn sie 'sich wiederholt'.)

6.3 Der lokale Algorithmus A3

Verwendet man z.B. einen *linearen* Kritiker, so bekommt man ein Verfahren, das lokal in Zeit und Raum ist. Interessanterweise stellt die Verwendung eines linearen Kritikers kein Hindernis für die Lösung von Aufgaben des *nicht linear separablen* Typs dar, wie man später sehen wird (siehe auch [51], [26]). Der resultierende Algorithmus A3 ist viel einfacher als Andersons.

6.3.1 Detaillierte Beschreibung von A3

Der Aktivationsvektor eines vollständig bidirektional vernetzten Steuernetzwerkes C zum Zeitpunkt t wird mit $x(t)$ bezeichnet, sein Gewichts-

vektor mit $w(t)$. Die Aktivation des i-ten Knotens von C heißt $x_i(t)$, und das Gewicht der gerichteten Verbindung von j nach i heißt $w_{ij}(t)$. C besteht aus linearen Eingabeknoten und stochastischen Nicht-Eingabeknoten. Der adaptive Kritiker A besteht aus *einem* Knoten mit *linearer* Aktivierungsfunktion und einem zugehörigem Gewichtsvektor $v(t)$. Für alle t gilt

$$dim(v(t)) = dim(x(t)).$$

Zunächst betrachten wir den Fall, daß die Trainingsphase in *Trainingsepisoden* untergliederbar ist. Eine Episode beginnt mit der Initialisierung von C's Aktivationsvektor und endet mit dem Bekanntwerden des externen Reinforcements R. Mit α und η werden positive Lernraten bezeichnet.

Initialisiere alle Gewichte mit zufällig gewählten reellen Werten.

Für alle Episoden:

Initialisiere die Aktivationen der Eingabeknoten im ersten Zeitschritt durch sensorische Wahrnehmung. Initialisiere die Aktivationen der anderen Knoten mit 0.

Für alle folgenden Zeitschritte t bis zum abschließenden Bekanntwerden von R:

1. Berechne A's Voraussage von R durch $r = x^T(t-1)v(t)$.

2. Bringe die Aktivitäten des rekurrenten Netzes C auf den neuesten Stand: Für alle Knoten i berechne

$$net_i(t) = \sum_j w_{ij}(t-1)x_i(t-1).$$

Die logistische Funktion $l(net_i(t)) = \frac{1}{1+e^{-net_i(t)}}$ liefert die Wahrscheinlichkeit dafür, daß $x_i(t)$ den Wert 1 *bzw.* 0 *annimmt. Jeder Knoten i merkt sich für später seine letzte Aktivation $x_i(t-1)$.*

3.A. Falls t der letzte Zeitschritt der Episode ist: Setze $r' = R$.

3.B. Andernfalls berechne A's neue Voraussage $r' = x^T(t)v(t)$.

In jedem Fall ist der Fehler des Kritikers gleich $r' - r$. Gradientenabstieg bezüglich der letzten Eingabe $x(t-1)$ (mit Lernrate η) ergibt A's neuen Gewichtsvektor $v(t+1)$.

4. Berechne alle Gewichtsinkremente

$$\Delta w_{ij}(t) = \alpha(r' - r)x_i(t-1)x_j(t)$$

und führe die Gewichtsänderungen durch: $w_{ij}(t) = w_{ij}(t-1) + \Delta w_{ij}(t)$.

Die Differenz sukzessiver Voraussagen des Kritikers bestimmt also die Lernrate einer Hebb-ähnlichen Lernregel für C (Schritt 4), (siehe auch [57] [55] [61]). Wollte man in Begriffen für kontinuierliche Zeit sprechen, könnte man sagen: *Die temporale Ableitung der Erwartung zukünftigen Reinforcements ist gleich dem effektiven Reinforcement.*

Man vergegenwärtige sich die extreme Einfachheit des Algorithmus. Jeder Knoten hat zu jedem Zeitpunkt dieselbe simple Folge von Berechnungen auszuführen. Der Spitzenberechnungsaufwand pro Zeitschritt und Verbindung ist $O(1)$. Für eine sequentielle Simulation auf einer von Neumann-Maschine ist der Spitzenberechnungsaufwand pro Zeitschritt $O(dim(w) + dim(v))$.

Um beliebig lange Zeiträume in den Griff zu bekommen, sollte man beim 3. Schritt von A3 eine kleine Modifikation anbringen:

3.B. Andernfalls berechne A's neue Voraussage $r' = \gamma x^T(t)v(t)$.

Dabei ist $0 < \gamma < 1$ wieder ein Abschwächungsfaktor, welcher in *naher* Zukunft erwartetes Reinforcement stärker gewichtet als in *ferner* Zukunft erwartetes Reinforcement. γ dient im wesentlichen der Vermeidung der Möglichkeit unendlicher Summen bei Voraussagen über kumulatives Reinforcement (siehe auch das Kapitel zum überwachten Lernen, Abschnitt 'TD-Methoden').

Schließlich sollte noch darauf hingewiesen werden, daß die in Schritt 4 angegebene Lernregel für C nur den *allereinfachsten* Repräsentanten einer ganzen Klasse anwendbarer einfacher Reinforcement-Lernregeln darstellt. Um z.B. *unwahrscheinliche* Transitionen von einem Netzwerkzustand zum nächsten stärker zu berücksichtigen als *wahrscheinliche*, braucht man die Lernregel nur leicht zu modifizieren:

$$\Delta w_{ij}(t) = \alpha(r' - r)x_i(t-1)(x_j(t) - P(x_j = 1 \mid x(t-1), w(t-1)).$$

Es wäre jedoch ebenso möglich, etwa Barto und Anandans Assoziative Bestrafungs- und Belohnungsregel [3] zu benutzen (siehe auch das Kapitel über R-Lernen, Abschnitt 'Neuronale Ansätze').

6.3.2 Ein Experiment mit 'verzögertem XOR'

In diesem Abschnitt wird experimentell demonstriert, daß der aus Lokalitäts- und Simplizitätsgründen eingeführte *lineare* Kritiker nicht notwendigerweise ein Hindernis für die Lösung von Problemen des *nicht* linear separablen Typs darstellt. Dabei werden auch die Unterschiede zu Andersons viel komplizierterem System aufgezeigt.

Beim 'verzögerten XOR' mit statischen Eingaben' geht es darum, ein rekurrentes Netzwerk zur Lösung eines XOR-Problems mit verzögerter Bewertung der Netzausgaben zu veranlassen. Es existiert keine externe Rückkopplung.

Für die Dauer eines Trainingsintervalls wurden zwei der drei Eingabeknoten des vollständig bidirektional vernetzten Hauptnetzwerkes C mit zufällig gewählten binären Werten besetzt. Der dritte Eingabeknoten war ein 'wahrer' Knoten, seine Aktivation war stets gleich 1, um allen Nicht-Eingabeknoten einen modifizierbaren Schwellwert zur Verfügung zu stellen. C verfügte über h versteckte Knoten.

C's Aufgabe bestand darin, eine vordefinierte Anzahl k von Zeitschritten lang zu laufen, und im letzten Zeitschritt das XOR der Aktivationen des ersten und des zweiten Eingabeknoten in einem einzelnen Ausgabeknoten sichtbar zu machen.

Für verschiedene Anzahlen versteckter Knoten sowie für verschiedene Verzögerungszeitdauern fand der lokale Algorithmus Lösungen für seine Aufgabe. So waren zum Beispiel bei einer Gewichtsinitialisierung zwischen -0.1 und 0.1 und unter den Voraussetzungen $h = 3$, $k = 3$ sowie $\eta = \alpha = 0.2$ in 7 von 10 durchgeführten Testläufen durchschnittlich 2000 Präsentationen pro Muster notwendig, um mehr als 97 Prozent korrekter Klassifikationen zu erzielen. Bei 4 Läufen war die Rate korrekter Klassifikationen größer als 99 Prozent.

Daß die Rate nicht immer 100 Prozent betrug, lag natürlich an der stochastischen Natur der Aktivierungsfunktionen von C's Knoten. Es war jedoch möglich, perfekte Klassifikation aller Muster zu erzwingen: Man mußte lediglich nach der Lernphase aus dem nicht-deterministischen Netzwerk ein deterministisches machen. Dies erforderte nichts weiter, als daß jeder Knoten von C stets die wahrscheinlichste Aktivation (abhängig von seinen Gewichten und seiner gegenwärtigen Eingabe) wählte. Schon relativ wenige Trainingszyklen reichten aus, um eine deterministische Lösung des Problems zu finden.

Wie ist es möglich, daß ein linearer Kritiker ausreicht, obwohl doch die Aufgabe vom *nicht* linear separablen Typ ist? Dieses kontraintuitive Faktum läßt sich dadurch erklären, daß die Aufgabe des Kritikers in der Regel einfacher ist als die Aufgabe des rekurrenten nicht-linearen Hauptnetzwerkes C. C muß in seinen rekurrenten Verbindungen das Programm für die XOR-Abbildung implementieren, während der Kritiker lediglich die Abbildung von Netzwerkzuständen auf zukünftiges Reinforcement zu implementieren braucht. Im allgemeinen (insbesondere zu Beginn der Lernphase) kann die Abbildung von Netzwerkzuständen auf zukünftiges zu erwartendes Reinforcement selbst eine linear nicht separable Funktion darstellen. Hat jedoch C schon gelernt, auf eine Untermenge der möglichen Eingaben

mit der korrekten verzögerten Antwort zu reagieren, so vereinfacht sich die Aufgabe des Kritikers. In der Tat wird seine Aufgabe nach dem Auffinden einer perfekten Lösung trivial: Dann muß er nämlich nur noch für alle Netzwerkzustände den immer gleichbleibenden Wert des finalen Reinforcements vorhersagen, welcher 1 ist.

Eine derartige abschließende triviale Abbildung kann durch eine schwergewichtige Verbindung vom 'wahren' Knoten auf den Kritiker implementiert werden. Tatsächlich war diese Art von Verbindung in manchen Fällen genau das, was nach dem Training beobachtet werden konnte.

Die wesentlichsten Unterschiede zu Andersons System [2] seien hier noch einmal aufgelistet. Im Gegensatz zu dem in Andersons Arbeit beschriebenen Algorithmus wurde kein BP-Netzwerk, sondern ein einzelner *linearer* Knoten als Kritiker verwendet. Weiterhin wurde kein statisches azyklisches mehrlagriges Netzwerk für die Berechnung von Ausgabeaktionen verwendet, sondern ein dynamisches vollständig zyklisches Netzwerk. Auch wurden versteckte Knoten *nicht* wie bei Andersons Methode mit *anderen* Lernregeln behandelt als die Ausgabeknoten.

Auch das Konzept der diskreten Zeitschritte war ein anderes: Während Andersons Zeitschritte zwei mehrlagige Aktivationsausbreitungs- und Fehlerrückpropagierungsphasen umfaßten, führt das oben beschriebene Verfahren in jedem Zeitschritt lediglich *Ein-Lagen*-Operationen aus. (Dabei sehen wir den *'update'*-Schritt des rekurrenten Netzes als eine Ein-Lagen-Operation an.) Man beachte, daß ein *'update'*-Schritt des Steuernetzes auch eine Änderung der Eingabeaktivationen aufgrund der externen Rückkopplung bedeutet. Dies hat eine Verzögerung von mindestens *zwei* Zeitschritten zwischen nicht-linear zu transformierenden Eingaben und entsprechenden Ausgabeaktionen zur Folge. Eine *neue* Eingabe kann unter Umständen bereits anliegen, bevor die Antwort auf die *alte* Eingabe berechnet ist.

Es muß darauf hingewiesen werden, daß ein linearer Kritiker nicht in allen Fällen ausreichend sein *muß*. Hieraus erwächst Motivation für die folgenden Abschnitte.

6.3.3 Kompliziertere statische Kritiker, kompliziertere R-Lernregeln

Statt des linearen Kritikers kann man ein BP-Netzwerk oder eine Boltzmann-Maschine verwenden. In der Regel zahlt man dabei mit dem 'Grad' der Lokalität des Verfahrens. Statt der einfachen Hebb-ähnlichen Lernregel kann man raffiniertere statische R-Lernregeln verwenden [3]. Im nächsten Abschnitt überspringen wir allerdings diese Möglichkeiten und führen ein Schema für die Berechnung wesentlich informierterer Gewichtsänderungen ein, als mit einer statistischen Lernregel möglich wären.

6.4 Multidimensionale adaptive Kritiker

Durch den adaptiven Kritiker verbessert sich die Aussagekraft des effektiven Reinforcementsignals im Vergleich zu Systemen ohne interne Bewertungsfunktionsgeneratoren ganz gewaltig. Allerdings sind A3's Kritiker sowie die von allen anderen Autoren verwendeten Kritiker *eindimensional*. Ihre Voraussage bezieht sich stets auf einen *skalaren* Wert, wie z.B. das kumulative zu erwartende Reinforcement. In globaler Weise wird ein und dasselbe interne R-Signal zur recht pauschalen Änderung aller Gewichte im Steuernetz herangezogen. Das R-Signal wirkt also zu einem gegebenen Zeitpunkt immer noch unspezifisch auf *alle* Netzwerkverbindungen ein. Es gibt keine 'individuell maßgeschneiderten Reinforcementsignale' [77]. Kein Unterschied wird z.B. zwischen verschiedenen Arten von Reinforcement getroffen. Dies scheint im Kontrast zur Funktionsweise biologischer Organismen zu stehen: Letzere verfügen i.a. über verschiedenartigste Schmerz- bzw. Lustsensoren und scheinen diese Reinforcementvielfalt nicht in einen einzigen skalaren Wert zu kollabieren.

Was sind die zu erwartenden Vorteile multidimensionaler R-Voraussagen? Intuitiv würde man erhoffen: Ein *verfeinertes* Modell der zu erwartenden Vor- und Nachteile bestimmter Handlungsweisen sollte auch eine *informiertere* Modifikation erfolgloser Handlungen ermöglichen. Weiß man, wie stark *welcher* Ausgabeknoten zu *welchen* Komponenten des internen Reinforcementvektors beitrug, so kann man *maßgeschneiderte* Gewichtsänderungen für das Steuernetz generieren. Weiterhin kann es sein, daß eine Abbildung von Eingaben auf vektorwertiges Reinforcement leichter zu erlernen ist als die entsprechende Abbildung auf skalares Reinforcement.

Ein verfeinertes Modell bekommt man allerdings nicht einfach durch Erweiterung der Ausgabedimensionalität des Kritikers zur Vorhersage verschiedener Arten von Reinforcement. Zusätzlicher Aufwand muß für die sinnvolle Umsetzung der internen Voraussagen in Steuernetzänderungen getrieben werden. Im folgenden wird ein 3-Netzwerk-Schema zur Berechnung informierter Gewichtsänderungen aus einem u.U. vektorwertigen internen R-Signal vorgestellt.

6.4.1 *Drei* interagierende Netzwerke

Wie kann man aus einem sowieso schon recht informierten internen vektorwertigen R-Signal des Kritikers ein noch informierteres machen?

Eine Antwort liefert das vorangehende 5. Kapitel: Wir verwenden ein drittes Netzwerk M, um *Differenzen aufeinanderfolgender Vorhersagen des Kritikers* (identisch mit den jeweiligen internen Reinforcementvektoren) in Abhängigkeit von aufeinanderfolgenden Netzwerkzuständen von C zu mo-

dellieren. Um den Systemidentifikationsansatz vernünftig ins Spiel zu brigen, muß aber der Alles-oder-Nichts-Charakter von C's stochastischer Aktivierungsfunktionen verschwinden (wir brauchen eine kontinuierliche differenzierbare Wahrscheinlichkeitsverteilung für die Ausgaben von C). Zu diesem Zweck definieren wir für jeden Knoten i eine neue Aktivierungsfunktion:

$$x_i(t) = f(\sum_j w_{ij}(t-1)x_i(t-1) + noise(t)).$$

Gehorcht $noise(t)$ z.B. einer Gauß-Verteilung (oder einer anderen differenzierbaren Verteilung), so kann man Williams' Konzept der 'Fehlerpropagierung durch Zufallsgeneratoren' anwenden [77]. Dem Systemidentifikationsansatz folgend propagiert man zu jedem Zeitpunkt die Differenz zwischen tatsächlichem und gewünschtem *internen* Reinforcementvektor durch C's Ausgabeknoten zurück 'in die Vergangenheit'. Zu diesem Zweck muß C sich gerade so viele vergangene Aktivationen pro Knoten merken, wie die Fehlerrückpropagierungsphase Schritte umfaßt (typischerweise also nur eine Aktivation pro Knoten.) Nur C's Gewichte ändern sich während der Fehlerpropagierung, M's Gewichte bleiben fix.

6.4.2 Verschmelzen der drei Netze in zwei

Um das Gesamtsystem zu vereinfachen, verschmelzen wir nun das Modellnetzwerk mit dem adaptiven Kritiker und nennen das Resultat einen Modellkritiker. Statt Differenzen aufeinanderfolgender Kritikervorhersagen zu modellieren, gibt der Modellkritiker in Abhängigkeit von der gegenwärtigen Ein- und Ausgabe des Steuernetzes eine Voraussage über die Summe aller in Zukunft zu erwartenden Schmerz- bzw. Lustvektoren aus. Sein Fehler wird durch die TD-Methode berechnet. *Die Gewichtsänderungen des Steuernetzes ergeben sich durch Gradientenabstieg in der Steuernetzwerk/Modellkritiker-Kombination, wobei die Gewichte des Modellkritikers kurz eingefroren werden (siehe Kapitel 2 und 5).* Das System lernt *on-line* und ist schwach lokal in Zeit und Raum.

Bemerkenswert ist bei diesem Ansatz, daß er im Gegensatz zu Suttons Methode, welche nur Zustände kritisiert, *Paare von Zuständen und Aktionen* bewertet. Damit besteht eine Ähnlichkeit zu den eindimensionalen Kritikern von Watkins [71] und Jordan und Jacobs [22]. Die Hauptschleife des Algorithmus sieht zum Zeitpunkt t wie folgt aus:

1. *Führe die zu C's letztem Ausgabevektor $a(t-1)$ korrespondierende*

Aktion in der Umgebung aus und gewinne den neuen Eingabevektor $x(t)$ aus den geänderten Zustandsvariablen.

2. Berechne die neue Steueraktion $a(t) = f(x(t))$, wobei f durch die gegenwärtige Gewichtsmatrix des BP-Steuernetzes C gegeben ist.

3. Berechne den Ausgabevektor $r(t) = g(x(t), a(t))$ des Modellkritikers MAC, wobei g durch MAC's gegenwärtige Gewichtsmatrix gegeben ist.

4. Verwende das Systemidentifikationsprinzip (Kapitel 3), um den Unterschied zwischen $r(t)$ und dem gewünschten internen R-Vektor rückwärts durch MAC und durch C's Ausgabeknoten zu propagieren. Ändere nur C's Gewichtsmatrix.

5. Verwende $x(t-1)$, $a(t-1)$, den 'Discountfaktor' $0 < \gamma < 1$ und die kumulative TD-Methode (Kapitel 2), um MAC's Gewichtsmatrix zu ändern.

Im folgenden Beitrag wird ein vierdimensionaler Kritiker auf eine schwierige Balancieraufgabe angewendet.

6.4.3 Ein schwieriges Balancierexperiment

Wiederholende Vorbemerkung. Sind die gewünschten Ausgaben dem Lehrer schon bekannt, so macht es natürlich mehr Sinn, zur Beschleunigung der Konvergenz einen überwachten gradientenbasierten Lernalgorithmus zu verwenden. Das gilt ganz allgemein: Mit Reinforcement-Algorithmen sollte man nicht in Konkurrenz zu überwachten Algorithmen treten wollen, so lange beide Lernparadigmen anwendbar sind. Vielmehr sollte man sich solche Probleme aussuchen, bei denen überwachte Algorithmen per definitionem scheitern müssen. Genau das werden wir im folgenden tun.

Wir versuchen uns an dem von Anderson [2] angegebenen Balancierproblem. Das aus Stab und Wagen bestehende physikalische System wurde durch die im Anhang definierten Differentialgleichungen modelliert. Die Aufgabe bestand wieder darin, den Stab solange wie möglich zu balancieren, ohne daß der Wagen an den Begrenzungen der Spur anstieß.

Das einzige Lehrsignal für das Steuernetzwerk C bestand in 'Schmerzsignalen' für den Zeitpunkt t, in dem der Wagen an einer der Spurbegrenzungen anstieß oder der Stab umkippte. Letztere Ereignisse bedeuteten gleichzeitig das Ende der entsprechenden Episode. Dank der Abwesenheit weiterer Lehrinformation stellte sich dem System trotz der Markov-Umgebung also ein schwieriges raum-zeitliches Lernproblem.

Es muß gleich betont werden, daß die Aufgabe (im Gefolge Andersons) viel schwieriger gestellt war als die ähnliche Aufgabe, die in Bar-

to, Sutton und Andersons vielzitiertem Artikel von 1983 beschrieben ist [5]. Barto, Sutton und Anderson verwendeten einen vorverdrahteten Dekoder, der in sorgfältig ausgeklügelter Weise Umgebungszustände in 160-dimensionale Einheitsvektoren als Eingabevektoren für das *ASE* Element übersetzte (siehe Kapitel 3). Statt dessen verwendete Anderson *reellwertige* Zustandsvariablen des Wagen/Stab-Systems als Eingaben für das Steuernetz. Damit mußte das Steuernetz selbst nicht-triviale Repräsentationen für die Zustände der externen Umgebung entwickeln.

Es sei darauf hingewiesen, daß der direkte Gebrauch der physikalischen Zustandsvariablen die Aufgabe vereinfacht. Anderson hat den Grund in der Symmetrie optimaler Strategien für positive und negative Werte der Zustandsvariablen identifiziert: Eine erfolgreiche Balancierstrategie für Fälle, in denen alle Zustandsvariablen im positiven Bereich liegen, bedeutet automatisch auch schon eine erfolgreiche Balancierstrategie für betragsmäßig gleiche negative Werte (und umgekehrt). Daher wurden in den Experimenten sowohl die asymmetrisch skalierten als auch die unskalierten Werte verwendet.

C bestand aus 4 Eingabeknoten für die im Anhang definierten sichtbaren Zustandsvariablen $z, \dot{z}, \theta, \dot{\theta}$ (bzw. die ebenfalls im Anhang definierten asymmetrisch skalierten Zustandsvariablen $\bar{z}, \bar{\dot{z}}, \bar{\theta}, \bar{\dot{\theta}}$), einem konstant aktivierten Eingabeknoten der Stärke 1 (der 'wahren' Eingabe), 5 versteckten Knoten mit logistischer Aktivierungsfunktion $f(x) = \frac{1}{1+e^{-x}}$, und einem linearen Ausgabeknoten. C's Ausgabeknoten war probabilistisch und bestand seinerseits aus 3 Subknoten, 2 davon zur Mittelwert- und Varianzgenerierung. Der Beitrag des Varianzknotens zur Aktivation des Ausgabeknotens bestand in seiner gegenwärtigen Aktivation multipliziert mit $ln(\frac{1-rnd}{rnd})$, wobei rnd eine gleichverteilte Zufallsvariable mit Werten zwischen $\frac{1}{2^{16}}$ und 1 war. Die einzigen Verbindungen von den Eingabeknoten zu den Ausgabeknoten führten über die versteckten Knoten. Es gab keine interne Rückkopplung.

Der (ebenfalls azyklische) Modellkritiker bestand aus 5 Eingabeknoten (vier für die Zustandsvariablen, einen für die Steuerausgabe), 5 logistischen versteckten Knoten sowie 4 linearen Ausgabeknoten zur Vorhersage vierer verschiedener möglicher Arten von 'Schmerz', je eine für die vier möglichen Fehlersituationen 'Wagen bumst gegen rechten Rand', 'Wagen bumst gegen linken Rand', 'Stab fällt nach links um', und 'Stab fällt nach rechts um'. Im Fehlerfall betrug der 'Schmerzbeitrag' für die entsprechende Voraussage 1.0. Aus Skalierungsgründen gab es eine fixe Verbindung mit Gewicht 0.1 zwischen dem Ausgabeknoten des Steuernetzes und der entsprechenden Eingabe des Modellkritikers. Diese wurde natürlich in jeder Phase der Fehlerpropagierung (vom Modellkritiker in das Steuernetz) berücksichtigt. Alle linearen Knoten im System hatten die Identitätsabbildung als Akti-

vierungsfunktion.

Zu Beginn einer Episode wurden die Zustandsvariablen $z, \dot{z}, \theta, \dot{\theta}$ zufällig innerhalb ihres Wertebereichs initialisiert. Zu einem gegebenen Zeitpunkt wurde die Aktivation des Ausgabeknotens als auf den Wagen auszuübende Kraft (gemessen in Newton) interpretiert. Im nächsten Zeitschritt änderte sich C's Eingabevektor gemäß einer Simulation des physikalischen Systems mittels Eulers Methode. Die zeitliche Distanz zwischen zwei Zeitschritten betrug dabei 0.02 s.

Es galt $\alpha = 0.2, \eta = 100, \gamma = 0.95$. Alle Gewichte wurden zufällig zwischen -0.05 und 0.05 initialisiert. Es wurden 5 Testläufe durchgeführt. Dabei wurde jeweils die Anzahl der Episoden gezählt, die bis zum Erreichen der ersten Episode mit mehr als 10 Minuten Balancierzeit notwendig waren (jeder Versuch wurde nach 30.000 Zeitschritten abgebrochen). (Bei zufälliger Auswahl der Ausgabeaktionen betrug die durchschnittliche Zeitdauer bis zum Eintritt in einen Fehlerzustand ca. 25 Zeitschritte (= 0.5 Sekunden).)

Die fünf Resultate waren: 713, 486, 536, 614, 513.

In allen durchgeführten Experimenten fand das Netzwerk also in weniger als 800 Versuchen eine befriedigende Lösung [?]. *Damit liegt im Vergleich zu Andersons eindimensionalen Kritiker [2] ein Geschwindigkeitsvorteil im Bereich einer Größenordnung vor.* Es war experimentell *nicht* möglich, mit einem *eindimensionalen* Kritiker vergleichbare Ergebnisse zu erzielen.

Bei unskalierten *Zustandsvariablen sank die Anzahl der für das Erreichen des Abbruchkriteriums notwendigen Lernversuche etwa auf ein Drittel.*

Die entsprechenden fünf Resultate waren: 174, 180, 144, 119, 155.

Es steht zu erwarten, daß sich das Konzept der mehrdimensionalen Kritiker gerade bei komplexen Umgebungen als überlegen erweist. Ein Beispiel könnte die Trajektoriengenerierung für Industrieroboter liefern. Sowohl die Ausgabemotorik als auch die 'Schmerzsensorik' (z. B. für das unerwünschte Anstoßen an Hindernissen) ist dabei i.a. so vielfältig, daß ein starker Bedarf nach 'maßgeschneiderten' Reinforcementsignalen besteht.

6.5 Einführung eines rekurrenten Kritikers

Um die Vorhersagen des adaptiven Kritikers *direkt von der Geschichte* der zeitlichen Evolution eines Steuernetzes C abhängig zu machen, bedarf es eines 'selbstüberwachten' Lernalgorithmus' für zyklische Netze. Zwar ist C aufgrund seiner eigenen zyklischen Verbindungen im Prinzip in der Lage, Information über vergangene Zustände zu repräsentieren. Erst durch einen kontinuierlich laufenden rekurrenten Kritiker ist jedoch gewährleistet, daß auch in Nicht-Markov-Umgebungen der Kritiker zutreffende Bewertungen

von Systemzuständen (und vergangenen Sequenzen von Systemzuständen) zu lernen imstande ist.

Wie immer zielen wir auf *'On-line'*-Lernen. Daher muß der Lernalgorithmus für den Kritiker lokal in der Zeit sein. Wir bedienen uns Robinson und Fallsides Verfahren [40], welches erstmals von Williams und Zipser implementiert wurde [80]. (Es ist in Kapitel 2 beschrieben. Sein Nachteil ist die nicht vorhandene Lokalität im Raum.) Das zentrale Grundprinzip ist wieder das schon früher beim *lokalen Algorithmus* A3 verwendete. Wir geben gleich den Algorithmus für die Maximierung *kumulativen* Reinforcements an.

Die Notation für die Aktivationen und Gewichte des Steuernetzwerkes C wird unverändert von der Beschreibung des lokalen Algorithmus übernommen. Über den Kritiker A wird lediglich ausgesagt, daß sein Eingabevektor die Dimensionalität von C's Aktivationsvektor besitzt. A's Eingabeknoten sind jeweils mit allen Nicht-Eingabeknoten von A 'vorwärtsverbunden'. Letztere sind ihrerseits vollständig bidirektional miteinander vernetzt. Einer der Nicht-Eingabeknoten von A wird der *Ausgabeknoten* genannt. Seine Aktivation wird zu einem gegebenen Zeitpunkt als eine Vorhersage des noch ausstehenden kumulativen Reinforcements interpretiert.

Initialisiere alle Gewichte von A und C mit zufällig gewählten reellen Werten.

Initialisiere die Aktivationen der Eingabeknoten von C durch sensorische Wahrnehmung. Initialisiere die Aktivationen von C's Nicht-Eingabeknoten mit 0.

Initialisiere die Aktivationen der Eingabeknoten von A mit C's Aktivationsvektor. Berechne A's erste Voraussage $P(1)$ des in Zukunft noch zu erwartenden kumulativen Reinforcements durch Ausführung eines Update-Schrittes für A (siehe Kapitel 2) und anschließendes Ablesen der Aktivation von A's Ausgabeknoten. Setze $r = P(1)$.

Für alle Zeitschritte t:

1. Bringe die Aktivitäten des rekurrenten Netzes C auf den neuesten Stand: Für alle Knoten i berechne zu diesem Zweck

$$net_i(t) = \sum_j w_{ij}(t-1)x_i(t-1).$$

Die logistische Funktion $l(net_i(t)) = \frac{1}{1+e^{-net_i(t)}}$ liefert die Wahrscheinlichkeit dafür, daß $x_i(t)$ den Wert 1 *bzw.* 0 *annimmt.*

2. Bringe die Aktivationen des rekurrenten Kritikers A auf den neuesten Stand: Führe dazu einen Update-Schritt für A (siehe Kapitel 2) aus. Gewin-

ne A's Voraussage $P(t)$ des in Zukunft noch zu erwartenden kumulativen Reinforcements durch Ablesen der Aktivation von A's Ausgabeknoten.

3. Berechne $r' = \gamma P(t) + R(t)$. ($R(t)$ ist das externe Reinforcement zur Zeit t.)

Der 'selbstgenerierte' Fehler des Kritikers ist gleich $E = r' - r$. Gemäß dem in Kapitel 2 vorgestellten Lernalgorithmus ändern sich A's Gewichte sofort proportional zu E's Gradienten bezüglich A's Gewichtsvektor.

4. Der Fehler des Kritikers ist gleichzeitig das Reinforcement für C's Reinforcement-Lernalgorithmus.

5. Setze r gleich r'.

Die Berechnung der Fehlersignale ist wieder inspiriert durch Suttons TD-Methoden. TD-Methoden erfordern jedoch zwei sukzessive Vorhersagen während desselben Zeitschrittes, um Abhängigkeiten von Gewichtsänderungen auszuschalten. Da die Ausgabe des rekurrenten Kritikers dank seiner zyklischen Verbindungen sowieso von vergangenen Zuständen des Steuernetzwerkes und auch von vergangenen Gewichten abhängt, begnügt sich die soeben beschriebene Methode mit nur *einer* Vorhersage pro Zeiteinheit.

C's in Schritt 4 referenzierter R-Lernalgorithmus kann z.B. durch die im Kontext des lokalen Algorithmus A3 beschriebene simple Lernregel gegeben sein, oder aber auch durch eines der im dritten Kapitel erwähnten komplizierteren Verfahren.

6.5.1 Rekurrente Kritiker und der Systemidentifikationsansatz

Auch das im letzten Unterabschnitt beschriebene Verfahren ist noch einer Erweiterung durch die im vorletzten Unterabschnitt angegebene Methode zugänglich. Man ändert die Aktivierungsfunktion von C's Knoten wieder so ab, daß die Aktivierungsverteilungen differenzierbar sind (z.B. mit Hilfe von Gauß-Verteilungen). Die vom rekurrenten Kritiker aufgrund von Unterschieden sukzessiver Voraussagen produzierten internen R-Signale muß man mittels eines dritten (im allgemeinen Fall ebenfalls rekurrenten) Netzwerkes M in Abhängigkeit von vergangenen Zuständen von C modellieren. Gradientenabstieg im Sinne des Systemidentifikationsansatzes führt dann unter Zuhilfenahme von M zu informierten R-Signalen für C [60]. Eine vereinfachte Version erhält man wiederum dadurch, daß man Kritiker und Modellnetzwerk analog zum letzten Abschnitt zu einem einzigen *rekurrenten Modellkritiker* kollabiert [54].

Im 5. Kapitel wurde bereits vorgeführt, wie zunächst nur *zwei* (statt

drei) interagierende rekurrente Netzwerke den Systemidentifikationsansatz für R-Lernen dienstbar machen können. Um aus diesem Ansatz einen rekurrenten Modellkritiker zu gewinnen, muß M's Fehlerfunktion in A2 in *einem* wesentlichen Punkt geändert werden: Der Fehler für die Vorhersagen der Reinforcementknoten muß gleich der Differenz zwischen *der Summe des nächsten externen Reinforcements und M's nächster gewichteter Vorhersage* und der gegenwärtigen Vorhersage sein.

6.5.2 Beschreibung einer verwandten Idee für *lokales* überwachtes Lernen

In [57] wurde unter anderem eine Idee zum *lokalen überwachten* Lernen beschrieben. Dabei bestand die Ausgabe des Kritikers in einem durch überwachtes Lernen gewonnenen Fehlervektor. Des jeweiligen Fehlervektors Dimensionalität war gleich der Anzahl der Nicht-Eingabeknoten eines dynamischen rekurrenten Netzes C und wurde mittels eines statischen überwachten Netzwerkes A mit den jeweiligen Zuständen von C assoziiert. Zu einem gegebenen Zeitpunkt t wurde A's auf C's gegenwärtigem Zustand beruhende Ausgabe als eine Schätzung desjenigen Fehlervektors interpretiert, der normalerweise durch einen konventionellen nicht-lokalen BP-Prozeß zustandegekommenen wäre. Der geschätzte Fehlervektor wurde durch C *einen* Schritt 'in die Vergangenheit' propagiert (das war eine lokale Operation), wobei die Summe des daraus resultierenden neuen Fehlervektors und des von einem Lehrer definierten *externen* Fehlers mittels A assoziiert wurde mit C's letztem Zustandsvektor. Die vom Lehrer gelieferten Fehlersignale für C's Ausgabeknoten beendeten die in obiger Ausführung implizite Rekursion.

6.6 Abschließende Bemerkungen

6.6.1 Konzeptuelle Gemeinsamkeiten mit der neuronalen Eimerbrigade

Zwischen der neuronalen Eimerkette und A3 bestehen zwar einige gravierende Unterschiede, aber auch mancherlei Gemeinsamkeiten [61]. Beide erlauben (wenigstens im Prinzip) beliebige Verzögerungen zwischen Aktionen und ihren späteren Konsequenzen. Dennoch betrachten beide während der Lernphase zu einem gegebenen Zeitpunkt t ausschließlich die engste zeitliche Umgebung von t. Bei beiden Verfahren lassen sich gewisse dynamisch veränderliche Größen als Voraussagen zukünftigen zu erwartenden Reinforcements interpretieren (bei der Eimerkette entspricht das Reinforcement

der Gewichtssubstanz).

6.6.2 Kritik und Ausblick

Keiner der vorgestellten Algorithmen (sowie keiner der Algorithmen eines anderen Autors) versucht die Vorteile wahrzunehmen, die man sich durch *kompositionelles Lernen* (Lernen durch Komposition von Unterprogrammen) verschaffen könnte. Es sei in diesem Zusammenhang hier nur auf das abschließende Kapitel dieser Arbeit verwiesen.

Kapitel 7

Dynamische adaptive selektive Aufmerksamkeit

In den bisherigen Kapiteln wurden einige allgemeine Ansätze zum Lernen mit interner und externer Rückkopplung vorgestellt. Obwohl nicht explizit erwähnt, bieten mindestens zwei dieser Ansätze ein Potential für adaptive selektive Aufmerksamkeit.

In diesem Kapitel wird erklärt, was unter selektiver *räumlicher* Aufmerksamkeit zu verstehen ist, und warum sie so wichtig ist. Am Beispiel *'attentive vision'* wird anschließend experimentell vorgeführt, daß das *Erlernen* selektiver *räumlicher* Aufmerksamkeit zumindest in gewissen Schranken möglich ist.

Weitere Motivation ist bei diesem Beitrag, die kaum erfolgreichen und ineffizienten rein statischen 'neuronalen' Ansätze zur visuellen Mustererkennung durch einen effizienteren und biologisch adäquateren Ansatz zu ersetzen, um damit die Vorteile der Beachtung der zeitlichen Dimension zu illustrieren. Als Seiteneffekt der selektiven Aufmerksamkeit ergibt sich nämlich ein Beweis dafür, daß man aus einer 'statischen' Aufgabe (Musterfindung) unter Effizienzgewinn eine dynamische machen kann.

Der sequentielle Ansatz ist inspiriert durch die Beobachtung, daß biologische Systeme den Mustererkennungsprozeß auf sequentielle Augenbewegungen abstützen. Ein aus zwei interagierenden Netzwerken bestehendes System lernt, sequentielle Fokustrajektorien zu erzeugen, so daß die finale Position eines durch 'Augenmuskulatur' bewegten Fokus einem zu findenden Objekt in einer visuellen Szene entspricht. Die einzige Zielinformation besteht in einer zu dem zu findenden Objekt korrespondierenden *gewünschten finalen Eingabe*. Trotz der Komplexität des zugehörigen *'tem-*

poral credit-assignment problem' wird gezeigt, daß es tatsächlich möglich ist, korrekte Sequenzen von Fokusbewegungen unter Einschluß von Translationen und Rotationen lernen zu lassen.

Im Rahmen der selektiven räumlichen Aufmerksamkeit finden sich auch einige Betrachtungen zum Thema Neugier und Langeweile. Die Motivation ist das für lernende Systeme oft notwendige Wechselspiel zwischen Exploration und Zielgerichtetheit. Die Wichtigkeit dieses Wechselspiels wird betont, und es wird ausgeführt, wie sich A2 in natürlicher und sinnvoller Weise um 'Neugierverhalten' erweitern läßt.

7.1 Wozu selektive räumliche Aufmerksamkeit?

Man vergegenwärtige sich nochmals die Arbeitsweise von A2: Zu jedem Zeitpunkt versucht A2, die Eingaben zum nächsten Zeitpunkt vollständig vorherzusagen. Der für A2 geforderte Gradientenabstieg profitiert zwar von dieser Arbeitsweise. Es scheint aber angebracht, das Konzept des allgemeinen raumzeitlichen Gradientenabstiegs selbst in Frage zu stellen: Für viele Zwecke genügt es oft, nur einen kleinen Teil des Zustands der Umgebung als *relevant* für die Erreichung des gegenwärtigen Ziels anzusehen. Die schwierige Frage lautet: Zu welchem Zeitpunkt sind welche Eingaben relevant und welche nicht?

7.2 Wie implementiert man adaptive räumliche Aufmerksamkeit?

Man gibt dem lernenden NN-System die Möglichkeit, durch adaptive Aktionen bestimmte Eingaben von der Umgebung 'auszublenden' und andere 'hervorzuheben'. Das ist im Prinzip auf vielfältige Weise machbar: Zum Beispiel könnte man sich vorstellen, daß für jeden Eingabeknoten e_k ein von einem zentralen adaptiven NN aktivierter Ausgabeknoten a_k existiert, dessen Aufgabe darin besteht, die gegenwärtig bei e_k anliegende Eingabe 'durchzulassen' oder nicht.

Mit Hilfe des Systemidentifikationsansatzes (oder anderer Ansätze für R-Lernen) können derartige Ein-/Ausblendeaktionen in sinnvoller Weise adaptiv gemacht werden. Was man für den Systemidentifikationsansatz braucht, ist ein adaptives Modell des *Nutzens* von bestimmten Aus- und Einblendeoperationen in bestimmten Kontexten. Wie im 6. Kapitel ausgeführt, kann solch ein Modell zur Berechnung von Fehlergradienten für

das die Ein- und Ausblendeaktionen triggernde Netzwerk verwendet werden.

Die einzige weitere Arbeit, die sich bisher mit adaptiver selektiver räumlicher Aufmerksamkeit beschäftigt hat, ist der vor kurzem erschienene technische Bericht von Whitehead und Ballard [76]. Whitehead verwendet sogenannte *'marker'*, die im Kontext einer Blockwelt an bestimmte Eigenschaften des Umgebungszustandes *gebunden* werden und diese hervorheben können. Das *Erlernen* einer gewissen Art von selektiver Aufmerksamkeit findet durch einen von TD-Methoden inspirierten Algorithmus statt.

7.3 Ein System für das Lernen von 'attentive vision'

Das im folgenden vorgestellte System besitzt (stark limitierte) Fähigkeiten, *sich dynamisch zu einem bestimmten Zeitpunkt auf bestimmte Details der Umgebung zu konzentrieren.* Es kann selbst Einfluß nehmen auf das, was es von der Umgebung wahrnimmt. Es kann in gewisser Weise *lernen*, sich mit der Zeit auf die *'richtigen'* Aspekte der Umgebung zu konzentrieren.

Dynamische Aufmerksamkeit wird in unserem Beispiel durch *externe Rückkopplung* implementiert: Ein Steuernetzwerk C steuert sequentielle Bewegungen eines 'Fokus' auf einer 'visuellen' Eingabeebene. Im Gegensatz zu dem im letzten Unterabschnitt beschriebenen allgemeinen Aufmerksamkeitslenker ist der Einfluß, den das System auf seine Eingaben nehmen kann, durch verschiedene Randbedingungen eingeschränkt: Zum Beispiel besteht der Fokus immer aus derselben Anzahl von Eingabeknoten und kann seine Topologie nicht ändern. Allerdings ist die Generalisierung auf den allgemeineren Fall trivial, zumindest was die Beschreibung anbetrifft.

Der Fokus liefert hohe Auflösung in seinem physikalischen Zentrum und niedrige Auflösung im Randbereich. Motorische Aktionen wie *'schiebe Fokus 15 Pixel nach rechts'* oder *'rotiere Fokus um 9 Grad'* werden durch lineare Transformationen der Aktivationen von C's Ausgabeknoten gesteuert. Die Aktivationen der Ausgabeknoten zu einem gegebenen Zeitschritt führen also in der Regel zu neuen Aktivationen für die Eingabeknoten zum nächsten Zeitschritt. So kommt die externe Rückkopplung ins Spiel.

Die *gewünschte finale Eingabe* am Ende einer von C zu generierenden *Fokustrajektorie* ist ein Aktivationsmuster, das gerade einem zu findenden Ziel in einer durch die Pixelebene gegebenen visuellen Szene entspricht. Die Aufgabe für C besteht also darin, in sequentieller Manier Fokusbewegungen zu erzeugen, und zwar so, daß unabhängig von der Startposition und der Ausgangslage des Fokus stets das Zielobjekt in der Szene gefun-

den wird. *Die Schwierigkeit besteht wieder einmal darin, daß kein Lehrer Aussagen über günstige Aktivationen der 'Augenmuskulatur' zu bestimmten Zeitpunkten zur Verfügung stellt.* C's einzige Fehlerinformation am Ende eines sequentiellen Erkennungsprozesses ist durch den Unterschied zwischen der *gewünschten* finalen Eingabe und der *tatsächlichen* finalen Eingabe gegeben. (Die Regelungstheorie nennt das ein 'terminales Steuerproblem'). Die Aufgabe schließt ein komplexes raumzeitliches Lernproblem und ein Aufmerksamkeitslenkungsproblem mit ein.

Zum Lernen verwenden wir eine im Vergleich zu A2 weniger allgemeine, dafür aber vom Berechnungsaufwand her günstigere Implementierung des Systemidentifikationsprinzips. Ein statisches Modellnetzwerk M lernt zunächst, ein Modell der sichtbaren externen Dynamik (erzeugt durch mögliche Fokusbewegungen) zu repräsentieren. Der 'unfolding-in-time' Algorithmus für Gradientenabstieg in dynamischen rekurrenten Netzen (hier war er trotz seiner nicht vorhandenen zeitlichen Lokalität anwendbar, siehe auch Kapitel 2) dient zur Berechnung von Gradienten für C's Ausgabeknoten (siehe Kapitel 3).

7.3.1 Detaillierte Beschreibung des Verfahrens

Jede Fokustrajektorie schließt k diskrete Zeitschritte 1 ... k ein. Zum Zeitpunkt t der Trajektorie p heißt C's Eingabevektor $x_p(t)$. $x_p(t)$ ergibt sich aus den sensorischen Eingaben des Fokus zur Zeit t. C's Ausgabevektor zur Zeit t heißt $c_p(t)$. $c_p(t)$ wird als Steuersignal für den Fokus interpretiert und verursacht eine Fokusbewegung und damit eine neue Eingabe $x_p(t+1)$. Die finale gewünschte Eingabe d_{pfin} der Trajektorie p ist ein extern vorgegebenes Aktivationsmuster und korrespondiert zu dem in der visuellen Szene zu findenden Objekt. Für alle t ist $dim(d_{pfin}) = dim(x_p(t))$. Zunächst nehmen wir an, daß d_{pfin} für alle Trajektorien p konstant ist (es soll also immer dasselbe Detail der Szene gefunden werden). C's Aufgabe besteht darin, startend von beliebig vorgegebenen Anfangspositionen Sequenzen von Fokusbewegungen zu produzieren, so daß für alle Trajektorien p gilt: $d_{pfin} = x_p(k)$. Der finale *Eingabefehler* e_{pfin} der zum Zeitschritt k unterbrochenen Trajektorie p ist

$$e_{pfin} = (d_{pfin} - x_p(k))^T (d_{pfin} - x_p(k)).$$

Die e_{pfin} ergeben sich also aus den Differenzen zwischen den gewünschten und den tatsächlichen finalen Eingaben.

Das Modellnetzwerk M sieht zu einem gegebenen Zeitpunkt t C's Ein- und Ausgabe und wird darauf trainiert, C's nächste Eingabe zu prophezeien. Die folgende Diskussion bezieht sich auf den Fall, daß M und C parallel

lernen. In einigen Experimenten werden wir separate Trainingsphasen für M und C verwenden, die Änderungen für diesen Fall sind aber trivial und hauptsächlich notationeller Art.

M's Eingabevektor zur Zeit t der Trajektorie p ist die Konkatenation von $c_p(t)$ und $x_p(t)$. M's Ausgabevektor zur Zeit t der Trajektorie p ist $m_p(t)$, wobei $\mid m_p(t) \mid = \mid x_p(t) \mid$. M's Fehler zur Zeit $0 < t < k$ der Trajektorie p ist

$$E_p(t) = (x_p(t+1) - m_p(t))^T (x_p(t+1) - m_p(t)).$$

M's Ziel ist die Minimierung von $\sum_{p,t} E_p(t)$, wozu BP verwendet wird:

$$\triangle W_M^T = -\alpha_M \frac{\partial \sum_{p,t} E_p(t)}{\partial W_M}.$$

Hierbei ist W_M M's Gewichtsvektor, $\triangle W_M$ dessen Inkrement, und α_M M's konstante Lernrate. (In den unten beschriebenen Experimenten weichen wir allerdings wieder vom reinen Gradientenabstieg ab und ändern M's Gewichte nach jedem Zeitschritt einer Trajektorie.)

Dem Systemidentifikationsansatz folgend nehmen wir nun an, daß $\sum_p e_{pfin}$ durch eine differenzierbare Funktion von C's Gewichtsvektor W_C angenähert werden kann. Um

$$\frac{\partial \sum_p e_{pfin}}{\partial W_C},$$

zu approximieren, wird vorausgesetzt, daß M *mit fixem* W_M *die Umgebung simulieren kann.* Die Approximation wird durch Anwendung des *unfolding in time*-Algorithmus (siehe Kapitel 2) auf das rekurrente Netzwerk berechnet, welches man dadurch erhält, daß man C's Eingabeknoten mit M's Ausgabeknoten und M's Eingabeknoten mit C's Ein- und Ausgabeknoten identifiziert:

$$\triangle W_C = -\alpha_C \sum_p \left(\frac{\partial m_p(k)}{\partial W_C} \right)^T (d_{pfin} - x_p(k)).$$

Dabei ist $\triangle W_C$ das Inkrement von W_C und α_C C's Lernrate. Man beachte, daß die Differenz zwischen der gewünschten finalen Eingabe und der tatsächlichen finalen Eingabe zur Berechnung eines Gradienten für C unter Zuhilfenahme von M herangezogen wird, *nicht* die Differenz zwischen der gewünschten finalen Eingabe und der von M vorhergesagten finalen Eingabe (siehe auch Kapitel 6). (In den unten beschriebenen Experimenten weichen wir vom reinen Gradientenabstieg ab und ändern C's Gewichte am Ende jeder Trajektorie.)

7.3.2 Das notwendigerweise nicht perfekte Modellnetzwerk

Man beachte, daß es unter der Voraussetzung zufällig auf der Pixelebene verteilter Objekte für M in der Regel unmöglich ist, *exakte* Voraussagen über zukünftige Fokuseingaben zu machen. Im Gegensatz zum bekannten 'truck backer upper'-Experiment, bei dem eine Handvoll Variabler genügt, um den kompletten Zustand der Umgebung hinreichend zu beschreiben [34], sehen im hier beschriebenen Experiment sowohl C als auch M *niemals* den gesamten Zustand der Umgebung, sondern immer nur ein paar lokale Details.

Genau dafür jedoch ist gelenkte Aufmerksamkeit gut: Aufmerksamkeit sollte auf diejenigen Teile der visuellen Szene gerichtet werden, welche detailliertere Information über die weitere Strategie für den Zielfindungsprozeß preisgeben können. M's Hauptaufgabe besteht darin, C in diejenigen Regionen der Pixelebene zu leiten, die eine Trajektorienfortführung mit infomierteren Bewegungen gestatten. (Beispiel: Man kann nicht genau vorhersagen, was man sehen wird, wenn man seine Augen auf die Zimmertür richtet. Man schafft allerdings die Voraussetzungen, um mit Hilfe weiterer Augenbewegungen das Gesicht der eintretenden Person zu erkennen.)

Würden die Dinge, auf die man seine Aufmerksamkeit richtet, niemals unerwartete Information liefern, so hätte das ganze Konzept der selektiven Aufmerksamkeit keinen Sinn. Man könnte das auch so formulieren: Wäre die Situation so, daß man M darauf trainieren könnte, stets *perfekte* Voraussagen zu machen, *würde M seine Existenzberechtigung verlieren.* Dann würde nämlich schon ein *einzelnes* Netzwerk ausreichen, die *gesamte* Information über die Umgebung zu speichern. Für alle interessanten Fälle *darf* das Modellnetzwerk gar nicht perfekt gemacht werden *können.*

Im Gegensatz zum 'truck backer upper' [34] ist es also nicht beabsichtigt, M zu einem perfekten Vorhersager zu machen, dessen Ausgaben die Eingaben von der Umgebung ersetzen könnten (in diesem Fall würde im Vergleich zum statischen Musterfindungsansatz nicht viel zu gewinnen sein). Vielmehr reicht es aus, wenn die inneren Produkte der auf einem ungenauen Modell beruhenden approximierten Gradienten für C und der auf einem hypothetischen exakten Modell beruhenden exakten Gradienten dazu tendieren, positiv zu sein.

Ein erklärtes Ziel dieses Beitrags ist, zu zeigen, daß ungenaue Modelle zu *perfekten* Lösungen beitragen können.

7.4 Experimente zur adaptiven räumlichen Aufmerksamkeit

Die im folgenden geschilderten Experimente wurden von Rudolf Huber im Rahmen seiner Diplomarbeit an der TUM durchgeführt [18].

7.4.1 Zielerkennung ohne Rotationen

Man betrachte Abbildung 7.1. Eine visuelle Szene ist durch ein schwarzes Objekt vor weißem Hintergrund oder ein weißes Objekt vor schwarzem Hintergrund gegeben, welches auf einem 512 x 512 Pixel umfassenden Pixelfeld plaziert ist. Sowohl während des Trainings als auch in der Testphase wurde die Position des Objektes für jeden neuen 'Versuch' zufällig gewählt. Statt (wie im einfachsten *statischen* Ansatz) Hunderttausende von Eingabeknoten zu verwenden, wurden nur 40 Eingabeknoten benötigt. Zum Ausgleich saßen diese dafür auf dem beweglichen Fokus, welcher im wesentlichen einer zweidimensionalen Retina nachgebildet war. Der Fokusdurchmesser war ungefähr gleich dem Objektdurchmesser. Die 'rezeptiven Felder' der 40 Eingabeknoten sind in der ersten Abbildung durch Kreise mit entsprechendem Radius symbolisiert. Zu einem gegebenen Zeitpunkt wurde die Aktivation eines Eingabeknotens durch die durchschnittlichen Werte (Schwarz = 1, weiß = 0) aller im zugehörigen rezeptiven Feld liegenden Pixel berechnet. Auch die Position des Fokus relativ zum Objekt wurde zu Beginn jeder Fokustrajektorie zufällig gewählt, allerdings so, daß eine teilweise Überlappung des Objektes durch die rezeptiven Felder des Fokus gegeben war. Auf den Abbildungen (z.B. Abbildung 7.2) wird die Position des Fokuszentrum für verschiedene Zeitpunkte jeweils durch einen Pfeil symbolisiert. Der Fokus wurde im Laufe eines Versuches durch die Aktivationen von 4 Ausgabeknoten des Steuernetzwerkes C bewegt. Diese waren für horizontale und vertikale Bewegungen zuständig: Für jede der Richtungen 'Nord', 'Süd', 'Ost' und 'West' gab es einen Ausgabeknoten. Zu jedem Zeitschritt wurde die Aktivation jedes Ausgabeknotens durch eine einfache Multiplikationsoperation in den Bereich zwischen 0 Pixeldurchmessern und 20 Pixeldurchmessern transformiert. Das Resultat wurde als die Länge eines Vektors in der dem jeweiligen Ausgabeknoten entsprechenden Richtung interpretiert. Schließlich wurde die Fokusbewegung durch Addition der vier entsprechenden Vektoren errechnet. C besaß 20 versteckte Knoten. M verfügte über 40+4 Eingabeknoten sowie 40 Ausgabeknoten und 40 versteckte Knoten. Alle Nicht-Eingabeknoten verwendeten die logistische Aktivierungsfunktion $f(x) = \frac{1}{1+e^{-x}}$. Sowohl M als auch C waren intern vollständig vorwärtsvernetzt.

Zu Beginn der Lernphasen wurden alle Gewichte zufällig mit Zahlen aus dem Intervall $[-0.1, 0.1]$ vorbesetzt. Bei der zuerst getesteten sequentiellen Version des Algorithmus wurde zunächst M anhand von 50.000 zufällig gewählten Situations/Aktions-Paaren trainiert. Daraufhin lernte C anhand von 10.000 Trainingstrajektorien. Während C's Lernphase galt $k = 5$. Dies entspricht $5 * (2 + 2) = 20$ Netzlagen in dem räumlich entfalteten dynamischen Netzwerk. Während der Arbeitsphase wurden 50 Zeitschritte pro Trajektorie gestattet. Der Wert 0.1 erwies sich als günstige Lernrate für sowohl C als auch M.

Die Experimente zeigten, daß das System in der Lage ist, *ohne Lehrer korrekte Sequenzen von Fokusbewegungen zu erlernen, obwohl das Modellnetzwerk häufig falsche Voraussagen liefert* [64]. Voraussetzung war dabei, daß das Objekt zu Beginn einer Trajektorie wenigstens teilweise durch den Einzugsbereich der Retina überlappt wurde. Am Ende einer Trajektorie pflegte der Fokus dergestalt auf dem Zieldetail des Objektes zu sitzen, daß die finale Eingabe der gewünschten entsprach. *Dabei war die Genauigkeit der Zielfindung nahezu optimal: Die finale Abweichung von der gewünschten Position betrug nie mehr als ein oder zwei Pixel.*

Jede der abgebildeten 50-schrittigen Trajektorien benötigte auf einer *SUN SPARC station* etwa eine Sekunde Echtzeit (samt Graphikausgabe). Hätte man einen voll parallelen statischen Ansatz für dasselbe Zielfindungsproblem verwendet (indem man *alle* Pixel auf einmal betrachtet hätte), so hätte der Effizienzverlust Größenordnungen betragen. (Es ist auch anzunehmen, daß viel mehr Trainingsbeispiele vonnöten gewesen wären, dies konnte allerdings aus Mangel an Rechenzeit nicht getestet werden.)

7.4.2 Ein Netz für mehrere Ziele

Indem man eine für die Dauer einer Trajektorie zeitinvariante zusätzliche statische Eingabe für C einführt, können für ein und dasselbe Netzwerk während aufeinanderfolgender Versuche *verschiedene* Zieldetails definiert werden. Zu diesem Zweck muß die Anzahl der Eingabeknoten von C verdoppelt werden. M ändert sich nicht, es sagt nach wie vor lediglich die zeitlich variierenden Eingaben von C voraus.

Solch ein erweitertes System ist fähig zu lernen, denjenigen Teil der Szene zu finden, der mit der zeitinvarianten Eingabe übereinstimmt (siehe Abbildung 7.3). Diese Möglichkeit ist bedeutsam für die später anzusprechenden Subzielgeneratoren.

7.4.3 Zielerkennung mit Rotationen

Es wurden zwei zusätzliche logistische Ausgabeknoten für C eingeführt. Sie waren für Fokusrotationen in der Bildebene (um das Fokuszentrum) zuständig. Dabei wurde die Aktivation des ersten zusätzlichen Ausgabeknotens zu jedem Zeitschritt durch eine einfache Multiplikationsoperation in den Bereich zwischen 0 und 50 Winkelgraden transformiert, die Aktivation des zweiten zusätzlichen Ausgabeknotens in den Bereich zwischen -50 und 0 Winkelgraden. Der Betrag der Rotation des Fokus um sein Zentrum ergab sich schließlich durch Addition der beiden Werte. Natürlich erhöhte sich auch die Zahl der Eingabeknoten von M um 2.

Das Erlernen korrekter Fokustrajektorien unter Einschluß von Rotationen erwies sich erwartungsgemäß als langwieriger als das Erlernen reiner Translationssequenzen. Bei den Experimenten mit der sequentiellen Version des Algorithmus erwiesen sich 100.000 Trainingsbeispiele für M und 20.000 Trainingstrajektorien für C als zweckmäßig. (Die sonstigen Parameter und Details wurden von den reinen Translationsexperimenten übernommen.)

In den Abbildungen 7.4 bis 7.7 wird die Rotation des Fokus' zu einem bestimmten Zeitschritt einer Trajektorie jeweils durch die Richtung des abgebildeten Pfeils symbolisiert. Nachdem M's Training abgeschlossen war, bestand die Aufgabe für C bei jedem Versuch darin, eine Trajektorie zu erzeugen, die das Zentrum des Fokus (dessen Position und Rotation zu Beginn einer Trajektorie zufällig gewählt wurden) in sequentieller Weise zu dem jeweils vorgegebenen Detail des Testobjektes zu führen hatte, wobei der Rotation des Objektes durch entsprechende Retinarotation Rechnung getragen werden mußte.

Die Experimente zeigten wiederum, daß das System in der Lage war, unter der Voraussetzung der teilweisen Überlappung des Objektes durch die rezeptiven Felder zu Beginn einer Trajektorie *ohne Lehrer korrekte Sequenzen von Fokusbewegungen zu erlernen, obwohl das Modellnetzwerk häufig falsche Voraussagen lieferte* [64].

7.4.4 Zielverfolgung

Es wurden weitere Experimente durchgeführt, welche zeigten, daß das System gut für das *Verfolgen* von sich bewegenden Objekten geeignet ist. So wurde z.B. das gewünschte Detail des wandernden Dreiecks (aus Abbildung 7.4) fokussiert und verfolgt, solange seine Geschwindigkeit nicht über der maximalen Fokusgeschwindigkeit lag.

Die potentiellen Anwendungen des Systems sind vielfältig. So ist beispielsweise beabsichtigt, die Anwendbarkeit des Systems auf folgende Aufgabe zu untersuchen: Werkstücke werden auf einem Fließband durch eine

Abbildung 7.1: Einer für die Experimente typischen visuellen Szene (schwarzes Objekt auf weißem Hintergrund) sind die rezeptiven Felder der sich auf der beweglichen 'Retina' befindlichen Eingabeknoten überlagert.

Abbildung 7.2: Translationen: Der Fokus findet nach der Trainingsphase seinen Weg von verschiedenen Teilen der Pixelebene zu seinem Ziel, dem Zentrum des Kreuzungspunktes in der Ziffer '4'. *Kein Lehrer sagte ihm, wie das zu machen sei!* Man beachte, daß der Fokus typischerweise *nicht* den kürzesten Weg nimmt, sondern eine Vorliebe für Kanten entwickelt.

Abbildung 7.3: *Ein* Netzwerk für mehrere Ziele: Durch eine zusätzliche stationäre Eingabe für das Steuernetzwerk können verschiedene Ziele in ein und derselben Szene definiert werden.

Abbildung 7.4: Ein Experiment mit Rotationen und Translationen: Nach dem Training findet der Fokus Wege von Startpunkten in der Umgebung des Objekts zu seinem Ziel.

Abbildung 7.5: Die Pixelebene ist durch Pseudo-Zufallsrauschen verunreinigt. Der Fokus fährt dennoch an das Ziel heran.

Abbildung 7.6: Auch bei diesem aus einem Ball und einem Kreuz komponierten Objekt entdeckte das System eine erfolgreiche Strategie zur Lösung seiner Aufgabe: Erst versuchte der Fokus, den Rand des Objektes zu finden und sich in eine Art Normalstellung zu begeben. Dann rutschte er solange am Rand des Balles entlang, bis er den Auswuchs des Kreuzes wahrnehmen konnte. Von diesem Punkt an lief er mehr oder weniger geradlinig zum Ziel, dem Zentrum des Kreuzes. *Man beachte erneut, daß kein überwachender Lehrer zu irgendeinem Zeitpunkt dem Fokus mitteilte, daß dies eine gute Strategie ist!*

Abbildung 7.7: Ein Werkstück aus der SIEMENS-Datenbank und zwei in der Umgebung des Werkstücks beginnende Fokustrajektorien. Das Ziel befindet sich im Inneren des Objektes.

Fabrik transportiert. Mit beweglichen Kameras ausgestattete Roboter stehen neben dem Fließband und müssen vorbeifahrende Objekte erkennen, visuell verfolgen, und für die weitere Verarbeitung vom Fließband nehmen.

7.4.5 Paralleles Lernen von C und M

Die oben beschriebenen Experimente gingen von separaten Trainingsphasen für C und M aus. Wie schon im 5. Kapitel ausgeführt, gibt es für den Fall *komplexer* Umgebungen gewichtige Gründe, C und M parallel lernen zu lassen.

Es wurden einige *'On-line'*-Experimente durchgeführt. Es stellte sich heraus, daß zwei interagierende konventionelle *deterministische* Netzwerke für die Aufgabe *nicht* geeignet waren. Ein deterministisches System fing sich bald in einem Zustand, in dem C den Fokus *niemals* in Regionen transportierte, die es dem Modellnetzwerk erlaubt hätten, neue relevante Daten über die externe Umgebung zu sammeln. (Das ist der schon im 5. Kapitel beschriebene *'deadlock'*.) Die aus der Umgebung *importierte* Zufälligkeit reichte *nicht* zur Lösung der Aufgabe aus.

Daher wurden einige Modifikationen für das Steuernetz eingeführt, um es mit *expliziter* Suchfähigkeit auszustatten: Jeder der 4 bzw. 6 Ausgabeknoten wurde durch ein aus zwei Knoten bestehendes kleines Netzwerk ersetzt. Einer dieser Knoten lieferte jeweils den *Mittelwert*, der andere die *Varianz* für einen Zufallsgenerator, welcher Zufallszahlen anhand einer stetigen differenzierbaren Wahrscheinlichkeitsverteilung produzierte. (Eine Gaussverteilung wurde dabei durch eine Bernoulliverteilung approximiert.) Gewichtsgradienten wurden mit Hilfe von Williams Konzept des 'BP durch Zufallszahlengeneratoren' berechnet [77].

Es stellte sich heraus, daß die parallele Version fähig war, geeignete Fokustrajektorien zu erlernen. Meist wurden dabei weniger als 100.000 Trainingstrajektorien benötigt. Wie erwartet, war das Modellnetzwerk nach dem Training nur in denjenigen Situationen ein relativ guter Prophet, die vom Steuernetzwerk im Verlauf typischer Trajektorien herbeigeführt wurden. Gegen Ende des Trainings ging die Varianz von C's Zufallsgeneratoren wie zu erwarten gegen Null.

Der Trainingsaufwand der parallelen Version war in etwa dem der sequentiellen Version vergleichbar, allerdings waren die Ergebnisse statistisch nicht signifikant genug, um eine eindeutige Aussage zuzulassen. Wie schon im Kapitel 5 ist die wesentliche Erkenntnis aus diesen Experimenten, daß der Ansatz nach Modifikation (Einführung probabilistischer Knoten für C) tatsächlich paralleles Lernen erlaubt, was keineswegs von vornherein klar war.

7.4.6 Sichtweise: Equilibria unter Einschluß der Umgebungsdynamik

Der oben beschriebene Fokusbeweger verfolgt das Ziel, auf einem bestimmten Teil der Pixelebene zur Ruhe zu kommen. Betrachtet man die Kombination aus Steuernetzwerk, gesteuertem Fokus und Pixelebene selbst als ein einziges Gesamtsystem, so ergibt sich eine interessante Perspektive: Man könnte in Analogie zu Equilibriumsnetzwerken (Hopfield-Netze, Boltzmann-Maschine) von einem dynamischen Equilibrium sprechen, *welches die Umgebungsdynamik mit einbezieht.* Teile der Pixelebene, die dem zu findenden Objekt ähneln, definieren Attraktoren oder lokale Minima in einer durch das Gesamtsystem definierten 'Energielandschaft'. Diese ist nicht mehr nur von der internen, *sondern auch von der externen Rückkopplung abhängig.* Ändert sich die Umgebung, so ändert sich auch das Energiegebirge. Eine schwierige, bislang unbeantwortete Frage erhebt sich: Für *welche* Art von Fokuskonstruktionen und für *welche* visuellen Szenen stellen *welche* zu entdeckenden Ziele tatsächlich Attraktoren und vielleicht gar *globale* Minima dar?

7.4.7 Zukünftige Untersuchungen

Das oben vorgestellte System löst gewiß nicht alle mit dem Thema *'attentive vision'* zusammenhängenden Probleme. Bisher wurden nur relativ einfache visuelle Szenen mit relativ einfachen geometrisch geformten einzelnen Objekten untersucht. Schon hier konnte Rudolf Huber gewisse im nächsten Unterabschnitt beschriebenen Probleme mit lokalen Minima feststellen [18]. Es ergeben sich jedoch interessante Perspektiven für zukünftige Arbeiten.

Szenen mit mehreren Objekten

Befinden sich mehrere (gar ähnlich aussehende) Objekte oder auch Objekte mit reicher interner Detailstruktur in der Szene, so benötigt man entweder rekurrente Verbindungen (à la A2) in M und C, oder aber irgendeinen anderen Mechanismus für die Flucht aus *lokalen Minima.* Lokale Minima ergeben sich zum Beispiel durch Teile der Pixelebene, die dem Zieldetail ähneln, während die nähere Umgebung dies nicht tut. In solchen Fällen wird die *relevante* externe Rückkopplung im allgemeinen *nicht-Markovsch.* Rudolf Huber hat für derartige Situationen noch einige Experimente mit multiplen Objekten und interagierenden rekurrenten Netzen durchgeführt [18]. Es stellte sich heraus, daß interne Rückkopplung in M und C in gewissen Fällen auch dann noch zum Erfolg führen kann, wenn externe Rück-

kopplung alleine nicht mehr ausreicht [18]. Einem sehr unterschiedlichen Ansatz werden allerdings langfristig mehr Chancen eingeräumt, nämlich dem *adaptiven Generieren von Subzielen.* Das nächste Kapitel zeigt, wie so etwas funktionieren kann.

Belohnung temporaler Invarianzen

Um die Fehlerfunktion des Systems zu *glätten* und unter Umständen die Konvergenz zu beschleunigen, kann man versuchen, den Eingabeknoten zusätzliche zeitliche Beschränkungen aufzuerlegen. Dies kann dadurch erreicht werden, daß man eine neue Fehlerfunktion konstruiert, indem man zur alten Fehlerfunktion Terme addiert, welche die Differenzen aufeinanderfolgender Fokuseingaben ausdrücken. Der Ansatz erinnert an Jordans Arbeit [21]. Man beachte jedoch, daß es bei Jordan die *Ausgabeknoten* sind, denen zeitliche Beschränkungen auferlegt werden.

Der zu erwartende Effekt besteht darin, daß das System eine Vorliebe für zeitliche Invarianzen im Eingaberaum entwickelt. Solche Invarianzen können zum Beispiel durch Fokusbewegungen entlang von Kanten verursacht werden. Damit fließt ein *unüberwachtes* Element (eine Suche nach 'Regelmäßigkeiten') in den Lernprozeß mit ein. Triviale zeitliche Invarianzen, die das System z.B. durch einen Fokusstop erzielen könnte, werden durch den *zielgerichteten* Teil der Fehlerfunktion ausgeschlossen.

7.5 Neugier und Langeweile

Viele biologische Lernsysteme, insbesondere die komplexeren, zeigen ein Wechselspiel zwischen zielgerichtetem und explorativem Lernen. Zusätzlich zu gewissen permanenten Zielen (wie zum Beispiel die Vermeidung von Schmerz) werden auch weitere Ziele generiert, deren direkter Nutzen lediglich darin besteht, das Wissen über die externe Welt zu erhöhen. Bisher wurde dieses Wechselspiel in der konnektionistischen Literatur noch überhaupt nicht beachtet.

Die explorative Seite des Lernens steht mit etwas in Beziehung, das normalerweise *'Neugier'* genannt wird. Neugier ist nicht völlig ziellos, wie manchmal argumentiert wird. Neugier hilft zu verstehen, wie die Welt funktioniert, was wiederum dem Erreichen bestimmter Ziele dient. Neugier ist eine weitere Form der *selektiven Aufmerksamkeit* auf bestimmte Aspekte der Umgebung. Die Zielgerichtetheit von Neugier ist jedoch weniger offensichtlich als beispielsweise die Zielgerichtetheit von A2 oder von weniger allgemeinen Algorithmen anderer Autoren.

Neugier hat mit dem zu tun, *was man bereits über die Umgebung weiß.*

Man wird neugierig, *wenn man glaubt*, daß es etwas gibt, *was man nicht weiß*. Das Ziel, zu verstehen, wie die Welt funktioniert, wird allerdings von anderen Zielen dominiert: Man weiß nicht *genau*, wie es sich anfühlt, wenn man die eigene Hand durch den Fleischwolf dreht. Man *möchte* es aber auch gar nicht wissen.

Neugier macht für ein lernendes System nur dann einen Sinn, wenn es auf das, *was* es lernt, *dynamischen Einfluß* nehmen kann. Weiterhin zielt Neugier auf die Minimierung einer dynamisch veränderlichen Größe, nämlich des 'Unwissenheitsgrads' über irgend etwas. Daher ist Neugier nur zweckmäßig für *'On-line'*-Lernsituationen, bei denen in irgendeiner Form dynamische selektive Aufmerksamkeit ins Spiel gebracht werden muß.

Die Vorbedingung von Neugier ist also so etwas wie der 'On-line'-Algorithmus A2, oder die Dreinetzwerkversion von A3, oder irgendein anderer modellbildender Algorithmus (z.B. auch Sutton's DYNA-Architektur [68]). Neugier und Langeweile sind Ausdruck *selektiver Aufmerksamkeit* auf bestimmte Eigenheiten der Umgebung. A2 stellt durch die Möglichkeit der externen Rückkopplung ein Potential für dynamische selektive Aufmerksamkeit zur Verfügung. Weiterhin baut A2 ein 'Weltmodell', um es für zielgerichtetes Lernen auszunützen. Das *direkte* Ziel von Neugier ist, das Weltmodell zu verbessern. Das *indirekte* Ziel ist, das Erlernen neuer zielgerichteter Aktionssequenzen zu erleichtern. Der Beitrag dieses Abschnitts besteht darin, zu zeigen, wie A2 (oder ähnliche Algorithmen) um Neugier und ihr Gegenstück *Langeweile* erweitert werden können.

Die zentrale Idee ist einfach [63]: Wir führen einen zusätzlichen Reinforcement-Knoten für C ein. Dieser Knoten, von nun an der *Neugierknoten* genannt, wird durch einen Prozeß aktiviert, welcher zu jedem Zeitpunkt die Distanz (z.B. die euklidische) zwischen Realität und Modellvorhersage mißt. Die Aktivation des Neugierknotens ist eine Funktion dieser Distanz. Ihr *gewünschter vordefinierter Wert* ist für alle Zeiten eine zur (weiter unten diskutierten) *idealen Diskrepanz zwischen Glauben und Tatsachen* korrespondierende relle Zahl. Damit wird Reinforcement unter Umständen gerade dann vergeben, wenn M *kein* guter Prophet der Umgebungsentwicklung ist. Der im letzten Kapitel detailliert ausgeführte Lernprozeß für C ermutigt demzufolge Aktionssequenzen, die zu einer Wiederholung der Diskrepanz-Situation führen.

Dabei verbessert sich M notgedrungen aufgrund seines eigenen Lernprozesses. Sobald M gelernt hat, in bestimmten Situationen die Umgebungsdynamik korrekt vorherzusagen, werden die zu solchen Situationen führenden Aktionen auch wieder *entmutigt*. Das liegt natürlich daran, daß die Aktivation des Neugierknotens auf Null zurück geht. *Langeweile* wird mit den entsprechenden Situationen assoziiert.

Wichtig ist dabei: Derselbe komplexe Mechanismus, der für 'normales'

Abbildung 7.8: Die Zeichnung stimmt weitgehend mit Abbildung 5.1 überein. Die Erweiterung bezieht sich auf den 'Neugierknoten' (CUR), welcher durch *Diskrepanzen zwischen Erwartungen des Modellnetzes und der Realität* aktiviert wird. CUR soll seinerseits durch $PRED_{CUR}$ vorhergesagt werden. Das Modellnetzwerk modelliert also unter anderem seine eigene Ignoranz und zeigt damit eine rudimentäre Form *introspektiven* Verhaltens.

zielgerichtetes Lernen verantwortlich ist, ist auch für adaptives Neugier- bzw. Langeweileverhalten verantwortlich. Es besteht kein Grund, ein separates System zur Verbesserung des Modellnetzes einzuführen. Solch ein Vorgehen steht in starkem Kontrast zu allen nicht-adaptiven Strategien zur Erforschung der Umgebung (wie zum Beispiel exhaustiver Suche, oder dem in DYNA verwendeten fixen Schema, welches sich einfach auf die vergangenen Zeitspannen seit dem letzten Auftreten jedes lokal repräsentierten Zustandes abstützt [68]).

C's Lernprozeß zielt auf den wiederholten Eintritt in Situationen, in denen M's Performanz nicht optimal ist. *Man beachte, daß dieser Prozeß selbst auf M angewiesen ist!* M muß lernen, unter anderem auch die zeitabhängigen Aktivationen des Neugierknotens vorherzusehen. *Damit ist M gezwungen, seine eigene Unwissenheit zu modellieren.* M muß lernen *zu wissen*, daß es gewisse Dinge *nicht weiß*.

Was ist die oben erwähnte *ideale Diskrepanz*? In der konventionellen AI gibt es ein geflügeltes Wort, welches besagt, daß ein System nichts lernen kann, was es nicht schon *beinahe weiß*. Will man sich dieser Ansicht anschließen, so sollte die Diskrepanzen in Reinforcement übersetzende Funktion konsequenterweise *keine lineare* Funktion sein. Kein Reinforcement sollte im Falle perfekter Voraussagen, hohes Reinforcement im Falle von *'near-misses'*, und wiederum niedriges Reinforcement im Falle großer Diskrepanzen vergeben werden. Diese Idee korrespondiert zu einem Gedanken aus der sogenannten *ästhetischen Informationstheorie* [32], welche sich bemüht, den Begriff der 'Schönheit' zu formalisieren. Ein Ansatz der ästhetischen Informationstheorie erklärt Schönheit durch einen Quotienten aus 'Unbekanntem' und 'Bekanntem', jeweils auf informationstheoretische Weise gemessen. Dieser Quotient sollte einen gewissen *idealen* Wert annehmen. Interessanterweise spielt in wenigstens einem dieser Ansätze der Kehrwert der Basis des natürlichen Logarithmus $\frac{1}{e}$ eine entscheidende Rolle als Kandidat für diesen Quotienten. Die genaue Natur einer guten Abbildung von Unterschieden zwischen Erwartung und Realität auf Reinforcement ist im Moment allerdings noch unklar.

Zukünftige Forschungen sollten sich unter anderem folgenden Fragen widmen: Gibt der um Neugierverhalten erweiterte Algorithmus Anlaß zu irgendwelchen dynamischen Instabilitäten? Wie sehen brauchbare Lernraten aus (es wird angenommen, daß M wesentlich schneller als C lernen sollte)? Wie groß sollte die relative Stärke von Neugierreinforcement im Vergleich zum rein zielgerichteten Reinforcement sein? Und was sind die genauen Eigenschaften einer 'guten' Abbildung von Diskrepanzen zwischen Erwartung und Realität auf Reinforcement?

Wenn auch eine nicht-lineare derartige Abbildung aus obigen Erwägungen heraus wünschenswert erscheinen mag, so heißt das nicht, daß eine

einfache *lineare* Abbildung sich nicht bereits als vorteilhaft gegenüber *gar keiner* Abbildung erweisen kann. Tatsächlich wurde in einigen wenigen Experimenten mit einer linearen Abbildung durch Josef Hochreiter bereits demonstriert, daß M's Fehler durch die *'On-line'*-Generierung von Neugierzielen reduziert werden können.

7.5.1 Schlußbemerkungen

Wie schon erwähnt, ist die Idee zur Implementierung von Neugier und Langeweile nicht auf A2 beschränkt. *Jeder* modellbildende Algorithmus kann von demselben Prinzip Gebrauch machen. Eine Hauptmotivation hierfür ist: Statt einen separaten Ad-Hoc-Mechanismus zur Verbesserung der Umweltmodellierung einzusetzen, wollen wir uns die wachsenden Fähigkeiten des zielgerichtet lernenden Systems selbst zunutze machen.

Der interessante Seiteneffekt für Algorithmen wie A2 ist der folgende: Da der Lernalgorithmus sich auf das Modellnetz abstützt, muß dieses eine Vorhersage über seine eigenen gegenwärtigen Vorhersagefähigkeiten treffen. Die *Aktivationen* des Modellnetzes werden (teilweise) interpretiert als eine Aussage über den Zustand seiner *Gewichte*. Man beachte, daß dies bereits eine rudimentäre Form von *introspektivem* Verhalten ist. Erweiterungen solcher introspektiver neuronaler Algorithmen könnten den Schlüssel zu Lernsystemen darstellen, die bis zu einem gewissen Grade lernen, wie man lernt. Hier könnte möglicherweise ein entscheidender Schritt in der Entwicklung von NN-Algorithmen getan werden.

Kapitel 8

Kompositionelles hierarchisches Lernen

In den bisherigen Kapiteln wurden einige im Prinzip sehr allgemeine Ansätze zum Lernen mit interner und externer Rückkopplung vorgestellt.

Dieses Kapitel kritisiert alle vorangegangenen Kapitel. Trotz ihrer Allgemeinheit sind die dort vorgestellten Algorithmen nämlich in verschiedener Hinsicht immer noch unbefriedigend. Keiner dieser Algorithmen (und auch kein Algorithmus eines anderen Autors) versucht, *hierarchische Komposition von Aktionssequenzen* ins Spiel zu brigen.

Keiner der existierenden Lernalgorithmen für dynamische Umgebungen wendet sich dem Problem des Lernens durch Zusammenfügen von Unterprogrammen, des Erlernens von 'Teile und herrsche'-Strategien zu. Im diesem Kapitel wird argumentiert, daß weder einfache adaptive Kritiker noch reine Gradientenabstiegsmethoden für großmaßstäbliche dynamische Steuerprobleme tauglich sein werden. Es besteht ein Bedarf für *kompositionelle Lernmethoden* und für *dynamische selektive zeitliche Aufmerksamkeit*. Im folgenden wird erklärt, was unter selektiver *zeitlicher* Aufmerksamkeit zu verstehen ist, und warum sie so wichtig ist. Einige mit kompositionellem Lernen assoziierte Probleme werden identifiziert, und ein System wird beschrieben, welches wenigstens eines dieser Probleme angreift.

Mit zwei Beiträgen zielt das 8. Kapitel in die Zukunft:

Im ersten Beitrag wird in konstruktiver Weise gezeigt, daß das *Erlernen* des hierarchischen Aufstellens von Subzielen und von *'divide and conquer'* Strategien möglich ist. Am Beispiel des ersten *adaptiven neuronalen Subzielgenerators* wird experimentell gezeigt, daß NN's das hierarchische Aufstellen von Subzielen wenigstens im Prinzip *erlernen* können [62].

Damit ergibt sich zum ersten Mal eine Möglichkeit, einen sogenannten *'higher-level-process'* (zeitüberbrückende Planung) auf 'neuronale' Weise adaptiv zu machen.

Der zweite Beitrag führt ein einfaches Prinzip zur unüberwachten 'Kausalitätsdetektion' aus einem kontinuierlichen Ereignisstrom ein. Darauf basierend werden die Grundzüge eines Verfahrens zur hierarchischen Abstraktion von Ereignisfolgen angegeben.

Der abschließende Ausblick weist auf perspektivenreiche Möglichkeiten für *introspektive* neuronale Lernalgorithmen hin. Wesentliche Grundzüge für 'neuronales Meta-Lernen' werden skizziert, im Rahmen der Arbeiten zur Dissertation allerdings nicht mehr implementiert.

8.1 Wozu selektive zeitliche Aufmerksamkeit ?

Im Rahmen der vorliegenden Arbeit wurde bereits klar, daß es zwei sehr unterschiedliche Klassen von Algorithmen für das Erlernen von raumzeitlichen Steuervorgängen gibt: Die Ansätze der adaptiven Kritiker, und die Systemidentifikationsansätze.

All diesen Algorithmen ist jedoch mindestens eine Eigenschaft gemein: Sie zeigen signifikante Schwächen, wenn der Lernprozeß große zeitliche Lücken zwischen vergangenen Aktionen und späteren Konsequenzen überbrücken muß. Man betrachte das folgende (utopische) Beispiel (welches lediglich der Illustration einiger Probleme dienen soll).

Ein von dynamischen neuronalen Netzen gesteuerter Roboter kommt zu Hause an und sieht sich außerstande, die Tür zu öffnen, weil er seinen Haustürschlüssel nicht dabei hat. Er hat den Schlüssel auf einem Pult der Technischen Universität vergessen. Eine weise (nun vom Lernprozeß zu entdeckende) Aktion hätte darin bestanden, den Schlüssel vor dem Verlassen der Universität einzustecken.

Was würde passieren, wenn der Roboter eine Gradientenabstiegsmethode (z.B. nach dem Systemidentifikationsansatz) für die Schuldzuweisung an vergangene Aktionen verwendete?

Alle vergangenen Aktivationen *aller* Netzknoten würden zur Berechnung eines Fehlergradienten für den Gewichtsvektor des Roboters beitragen. (Der Fehler könnte durch negatives Reinforcement oder durch die Differenz einer gewünschten finalen Eingabe - sagen wir, dem Anblick einer offenen Tür - und der tatsächlichen finalen Eingabe definiert werden.) Spuren *jeder* vergangenen Aktion des Roboters, *jeden* Schrittes auf seinem Heimweg würden für den Lernprozeß in Betracht gezogen werden. *Nahezu alle* die-

ser vergangenen Schritte sind jedoch *völlig irrelevant* im Kontext der gegenwärtigen Aufgabe (welche darin besteht, das Netzwerk so zu verändern, daß so etwas wie die gegenwärtige unliebsame Erfahrung sich in Zukunft nicht wiederholt). Im allgemeinen Fall werden nur *einige wenige* vergangene Ereignisse potentiell wesentlich für den augenblicklichen Mißerfolg des Roboters gewesen sein, einschließlich der 'Entscheidung', die Universität ohne den Schlüssel zu verlassen.

Was geschähe, wenn der Roboter einen adaptiven Kritiker für seinen Lernprozeß beschäftigen würde?

Zunächst würden *nur* die kürzlichst durchlaufenen internen Zustände des Roboters mit einer modifizierten Erwartung des unerwünschten Ereignisses versehen werden. Der Roboter müßte denselben Fehler *immer wieder* aufs Neue wiederholen, um relevante Entscheidungen zu Beginn jedes vom Mißerfolg gekrönten Versuches mit in den Lernprozeß einzubeziehen.

Für den Fall, daß der Roboter dank früherem Training schon eine Menge nützlicher Handlungssequenzen beherrscht (wie zum Beispiel das Greifen nach Schlüsseln, oder das Nach-Hause-Marschieren), hinterlassen beide Ansätze den Eindruck überwältigender Verkorkstheit. Bei beiden Ansätzen schreitet das *'credit assignment'* von 'Zeitschritt zu Zeitschritt' voran, anstatt auf einem höheren, abstrakteren Niveau *'Sprünge durch die Zeit'* zuzulassen. Beide Ansätze tendieren dazu, *Unterprogramme* statt *Aufrufbedingungen für Unterprogramme* zu modifizieren. Keiner der Ansätze besitzt auch nur das *Konzept* eines 'Unterprogramms'. Reine Gradientenabstiegsmethoden ziehen ohne Rücksicht auf vergangene Erfahrungen *immer* alle vergangenen Ereignisse für den Lernprozeß in Betracht. Adaptive Kritiker ziehen ohne Rücksicht auf vergangene Erfahrungen *immer* nur die kürzlichst vergangenen Ereignisse in Betracht. *Beide* tendieren dazu, in solchen Situationen wie oben zunächst die *falschen* vergangenen Ereignisse zu betrachten.

Es gibt also ein offensichtliches Bedürfnis nach Lernverfahren, die nur solche vergangenen Ereignisse berücksichtigen, welche wahrscheinlich *relevant* für den Lernprozeß sind. Dies erfordert *dynamische zeitliche Aufmerksamkeit. Adaptive* dynamische zeitliche Aufmerksamkeit zielt darauf, zu lernen, im Kontext bestimmter Zielvorgaben relevante vergangene Ereignisse zu isolieren.

8.2 Kompositionelles Lernen: Das 'Teile und herrsche'-Problem

Gibt es kein *a priori* Wissen über typische Konsequenzen bestimmter Aktionssequenzen, können wir von unserem Roboter nicht erwarten, daß er die Anzahl der für den Lernprozeß zu beachtenden Ereignisse in vernünftiger Weise reduziert. Nehmen wir jedoch an, daß der Roboter schon gelernt hat, gewisse Aktionssequenzen in zufriedenstellender Weise durchzuführen, dann sollte ein intelligenter Adaptionsvorgang schon vorhandene 'Unterprogramme' zur Erleichterung des Lernens neuer Aufgaben ausnützen. Auf inkrementelle Weise sollte der Roboter Information über Startbedingungen und Effekte von Unterprogrammen zur Komposition von komplizierteren (Unter-)Programmen heranziehen.

Kompositionelles Lernen bedeutet zu lernen, Lösungen für neue Aufgaben durch sequentielle Aneinanderreihung von Lösungen für ältere Probleme zu generieren. *Kompositionelles Lernen* heißt zu lernen, wie man 'teilt und herrscht'. Es bedeutet auch zu lernen, wie man das Problem, eine Aktionssequenz von einem Startzustand in einen Zielzustand zu finden, in mehrere Unterprobleme zerlegt, für die schon Unterprogramme existieren. Es bedeutet das Erlernen des Ignorierens irrelevanter Details von Unterprogrammen. Die *Schnittstellen zwischen Unterprogrammen* treten dabei in den Vordergrund. Kompositionen von Unterprogrammen können ihrerseits als Unterprogramme für noch kompliziertere Aufgaben dienen.

Erneut sei auf die Symmetrie zwischen Planen und Lernen hingewiesen: In beiden Fällen müssen Ereignissubsequenzen kombiniert werden, um eine Brücke von Anfangszuständen zu Endzuständen zu bauen. Derselbe Prozeß, der einen gegenwärtigen Zustand durch die Zusammensetzung von Repräsentationen vergangener Subprozesse zu erklären vermag, kann auch zur Erreichung zukünftiger Ziele durch Zusammensetzung von Subprozessen dienen. Da Lernen nur dann Sinn macht, wenn Aufgaben, die man in der Vergangenheit hätte lösen sollen, denjenigen ähneln, die in der Zukunft zu lösen sein werden, kann man das Planen im wesentlichen dem Lernen gleichsetzen.

Wir gehen im folgenden davon aus, daß das 'Teile und herrsche'-Problem in zwei Teile zerlegt und dann (besser) beherrscht werden kann, nämlich in das 'Teile'-Problem und das 'Herrsche'-Problem.

Das 'Teile'-Problem besteht darin, alle Arten von Ereignissequenzen in *zusammengehörige* Untersequenzen zu zerlegen. Es besteht also darin zu entscheiden, *was* ein gutes Unterprogramm ist, und welche Anfangs- und Endzustände den Unterprogrammen zu eigen sind. Offensichtlich hat das Teile-Problem mit *unüberwachtem* Lernen zu tun.

Das 'Herrsche'-Problem besteht darin, aus vielen vorhandenen Unterprogrammen bestimmte auszuwählen und sie in einer Weise zu kombinieren, die das Erreichen eines Zielzustandes zur Folge hat.

Im folgenden wird das Herrsche-Problem zunächst isoliert unter der Annahme betrachtet werden, daß das Teile-Problem bereits gelöst ist. (Das Teile-Problem wird im Ausblick beim Thema *Kausalitätsdetektoren* angeschnitten werden.) Wir konzentrieren uns also auf das Problem der Generierung adäquater Unterziele für nicht-triviale Aufgaben unter der Voraussetzung des Vorhandenseins funktionierender Unterprogramme für einfacherere Aufgaben. Der nächste Unterabschnitt beschreibt einen adaptiven Subzielgenerator. Letzterer benutzt adaptive Modellnetzwerke zur Vorhersage der Effekte der dem Gesamtsystem eigenen Unterprogramme.

8.3 Ein adaptiver Subzielgenerator

Abbildung 7.8 zeigt die Hauptkomponenten eines aus einer Anzahl interagierender neuronaler Netze bestehenden Systems. Das Herz des Systems ist ein neuronales Steuernetz C mit interner und externer Rückkopplung. C dient als Programmausführer. C's Eingabe besteht aus der Repräsentation eines Startzustandes, der Repräsentation eines gewünschten Zielzustandes und zeitlich variierenden Eingaben aus der Umgebung. Eine Kombination aus Start und Ziel dient als *Programmname*. Es wird angenommen, daß C schon gelernt hat, eine Anzahl von Programmen auszuführen. Das bedeutet nichts anderes, als daß bereits einige funktionierende Unterprogramme existieren, die tatsächlich von den jeweiligen Start- zu den jeweiligen Zielzuständen führen, durch die die Programme indiziert sind. Solche Unterprogramme mögen durch einen der in den vorangegangenen Kapiteln dieser Arbeit beschriebenen Algorithmen gelernt worden sein *oder auch durch eine rekursive Anwendung des im folgenden ausgeführten Prinzips.*

Ein zweites wichtiges Modul ist das *statische Evaluatornetzwerk* E. E's Eingabe besteht aus der Repräsentation eines Startzustandes und der Repräsentation eines gewünschten Zielzustandes. E's Ausgabe soll anzeigen, ob C ein vom Startzustand zum Zielzustand (oder nahe an den Zielzustand heran-) führendes Programm kennt. Eine Ausgabe von 1 bedeutet, daß ein adäquates Unterprogramm existiert. Eine Ausgabe von 0 bedeutet, daß kein adäquates Unterprogramm existiert. Eine Ausgabe zwischen 0 und 1 bedeutet, daß es ein Unterprogramm gibt, das vom Startzustand in einen Zustand führt, welcher dem Zielzustand in einem gewissen Sinne *nahe* ist. Das *Maß der Nähe* muß durch einen evaluativen Prozeß gegeben sein, welcher adaptiv sein kann oder aber auch nicht, und der in diesem Unterabschnitt nicht detailliert spezifiziert zu werden braucht. E repräsentiert

ein Modell der dem System eigenen Fähigkeiten.

Wir nehmen im folgenden an, daß E schon für jede Start-Ziel Kombination die Güte des entsprechenden Unterprogramms angeben kann. E kann in einer explorativen Phase trainiert werden, während der vorhandene Unterprogramme bzw. Start-Ziel Kombinationen getestet werden. (Natürlich ergibt sich langfristig gesehen eine interessantere Situation, wenn E parallel zu den anderen Systemkomponenten lernt.)

Schließlich enthält das System ein als Subzielgenerator dienendes weiteres *statisches* Netzwerk S. S' Eingabe besteht aus C's Starteingabe sowie aus der am Ende der auszuführenden Aktionssequenz für C *gewünschten* Eingabe.

Die Ausgabe des Subzielgenerators S ist natürlich ein Subziel. Geradeso wie das Ziel ist auch das Subziel ein die gewünschte Eingabe am Ende eines auszuführenden Unterprogramms beschreibendes Aktivationsmuster, welches *gleichzeitig* die Starteingabe für ein weiteres Unterprogramm darstellt. Wir konzentrieren uns auf den einfachsten Fall: Wir nehmen an, daß Lösungen für gegebene Aufgaben durch die Generierung eines einzigen Subziels gefunden werden können (Generalisierungen zu ganzen Subzielsequenzen sind allerdings trivial). S sollte ein Subziel mit der Eigenschaft generieren, daß sowohl ein vom Startzustand zum Subziel als auch ein vom Subziel zum gewünschten Endzustand führendes Unterprogramm existiert.

Wie lernt der Subzielgenerator, der zunächst eine *tabula rasa* ist, der Problemstellung angemessene Subziele zu generieren? Wir nehmen 2 Kopien des Evaluatornetzes E und verbinden sie mit S wie in Abbildung 7.9 dargestellt. Die gewünschte Ausgabe beider Kopien ist 1. Ist eine der beiden tatsächlichen Ausgaben niedriger als 1, so wird der entsprechende Fehler durch E's Kopien und durch S hindurchpropagiert. *Von der im Anschluß stattfindenden Gewichtsänderung bleiben die Gewichte der Kopien von E sowie E's Gewichte selbst unberührt.* Nur S' Gewichte ändern sich. Für ein gegebenes Problem wird die Prozedur iteriert, bis der Gesamtfehler an den Ausgaben von E's Kopien gleich Null ist (dann wurde eine durch die Komposition der beiden entsprechenden Unterprogramme zu erhaltende Lösung gefunden), oder bis ein lokales Minimum erreicht wurde (keine Lösung). *Die Gradientenabstiegsprozedur dient also einer informierten Suche im Raum aller Subziele.*

In Kapitel 5 wurde gesehen, wie man nicht nur Programm*ausgaben*, sondern auch Programm*eingaben* in Bezug auf Programme differenzierbar machen kann. (Die Gewichtsmatrix eines Netzwerkes mit fixer Topologie wurde dabei als sein Programm angesehen.) Da man eine Kombination von Repräsentationen bestimmter Start und Zielzustände als einen Programmnamen ansehen kann, geht der in diesem Abschnitt vorgestellte Ansatz noch einen Schritt weiter: Er macht Eingaben differenzierbar in Bezug auf *Pro-*

grammnamen.

8.4 Ein illustratives Experiment mit dem Subzielgenerator

Zur Demonstration der prinzipiellen Funktionstüchtigkeit des oben beschriebenen Ansatzes wurde ein einfaches Experiment ausgeführt. Implementierung und Tests oblagen Rudolf Huber, der gegenwärtig an der TUM seine Diplomarbeit anfertigt.

Es wurde eine zweidimensionale 'Miniwelt' konstruiert. Die Miniwelt fand Platz im reellen Einheitsquadrat. Sowohl Start- als auch Zielzustände wurden durch die entsprechenden Koordinatenpaare repräsentiert. Programmausführer, Evaluierer, und Subzielgenerator waren einfache Standard-BP-Netzwerke mit jeweils drei Netzlagen und 20 versteckten Knoten. Alle Nicht-Eingabeknoten bedienten sich der logistischen Aktivierungsfunktion $f(x) = \frac{1}{1+e^{-x}}$.

Der Programmausführer steuerte die Bewegungen eines 'künstlichen Tierchens', welches durch die Miniwelt marschierte. Die stationäre Eingabe des Programmausführers war vierdimensional und bestand aus einer Start/Ziel-Kombination. Seine Ausgabe war ebenfalls vierdimensional (es gab jeweils einen Ausgabeknoten für jede der Richtungen 'Nord', 'Süd', 'Ost' und 'West'). Zu jedem Zeitschritt wurde die Aktivation jedes Ausgabeknotens durch 20 dividiert, das Resultat wurde als die Länge eines Vektors in der dem jeweiligen Ausgabeknoten entsprechenden Richtung interpretiert. Schließlich wurde die Bewegung des Tierchens durch Addition der vier entsprechenden Vektoren errechnet. Damit war die maximale Schrittweite pro Zeitschritt in jeder der vier Richtungen durch 0.05 beschränkt.

Im Zentrum der Miniwelt befand sich ein Hindernis (symbolisiert durch das schwarze Quadrat in Abbildung 3). Traf das Tierchen im Laufe seines Marsches auf das Hindernis, so wurde es gestoppt.

Konventionelles BP wurde verwendet, um dem Programmausführer beizubringen, das Tierchen in gerader Linie von bestimmten Punkten der Miniwelt zu gewissen anderen Punkten zu manövrieren. Sowohl Start- als auch Zielzustände wurden durch Koordinaten korrespondierender Punkte angezeigt. Der Ausführer sah während seiner Trainingsphase 100.000 Aktionssequenzen. Keine der dem Tierchen beigebrachten Aktionssequenzen dauerte länger als 20 Zeitschritte.

Nach dem Training gab es immer noch viele Start-Ziel Kombinationen, für die der Programmausführer kein zugehöriges Programm kennen konnte.

In einer zweiten Trainigsphase wurde als nächstes der Evaluator darauf-

Abbildung 8.1: Die drei wesentlichen Bestandteile des Subziele lernenden Systems sind ein Steuernetzwerk, ein Evaluatornetz und ein Subzielgenerator.

Abbildung 8.2: Kopien des Evaluatornetzwerkes dienen der Berechnung von Gradienten für den Subzielgenerator.

Abbildung 8.3: Dargestellt ist eine zweidimensionale Welt mit einem 'Hindernis' (dem schwarzen Quadrat in der Mitte). Die mehr oder weniger geraden Linien stellen die Spuren von einigen wenigen Programmen (aus der sehr großen Menge der dem Programmausführer bekannten, jeweils von einem Anfangszustand zu einem Endzustand führenden Programme) dar.

Abbildung 8.4: Eine Start/Ziel-Kombination, für die der Programmausführer alleine keine korrekte Aktionssequenz kennen konnte. Mit Hilfe des adaptiven Subzielgenerators fand das System jedoch sehr schnell eine Lösung des Problems.

hin trainiert, für gegebene Start-Ziel-Kombinationen die Güte des entsprechenden Programms vorherzusagen. Die Ausgabe des Evaluierers wurde auf einen Wert von 1.0 trainiert für den Fall, daß die finale Position des Tierchens sich betragsmäßig um mehr als 0.1 von der Zielposition unterschied. Die gewünschte Ausgabe des Evaluierers betrug 0.0 im Fall der Deckungsgleichheit von finaler Position und Zielposition. Dazwischen wurde lineare Interpolation verwendet. Der Evaluierer sah 1.000.000 Trainingsbeispiele während seiner Lernphase. Sowohl Programmausführer als auch Evaluierer lernten mit einer Lernrate von 0.05.

Der Hauptgrund für die Wahl einer einfachen Umgebung war, den Subzielgenerierugsprozeß von Effekten zu isolieren, die durch eine *adaptive* Evaluierungsfunktion (z.B. einem adaptiven Kritiker) hätten eingeführt werden können. Für unsere einfache Umgebung war es leicht, eine *vorverdrahtete* Evaluierungsfunktion zu definieren. (Zukünftige Arbeiten werden sich natürlich auf paralleles Lernen aller Systemkomponenten konzentrieren. Wie immer ist es allerdings vorzuziehen, inkrementell von kleineren Problemen in Richtung größere Probleme voranzuschreiten.)

In der finalen Phase wurde der Subzielgenerator (20 versteckte Knoten) trainiert. Kombinationen von Start- und Zielzuständen ohne zugehöriges funktionierendes Programm wurden dem im letzten Abschnitt beschriebenen Subzielgenerierungsprozeß überantwortet. *Für eine gegebene Start/Ziel-Kombination lernte der Subzielgenerator dabei tatsächlich innerhalb von ca. 10 Iterationen (bei einer Lernrate von 1.0), angemessene Subziele für das Tierchen aufzustellen* (siehe Abbildung 4).

Das Beispiel zeigt, daß die Methode im Prinzip nicht nur theoretisch, sondern auch praktisch für adaptive Subzielgenerierung geeignet ist. Weiterführende (im Rahmen dieser Arbeit nicht mehr in Angriff genommene) Experimente wären nötig, um z.B. die Konsequenzen verschiedenartiger Hindernisse systematisch zu erfassen. Ebenso sollte die adaptive Subzielgenerierung in Fällen, bei denen mehr als ein Subziel erforderlich ist, experimentell untersucht werden.

8.5 Das 'Teile'-Problem und adaptive Kausalitätsdetektoren

Eine sehr wichtige Frage, der im letzten Abschnitt aus dem Weg gegangen wurde, lautet: Was *ist* denn ein gutes Unterprogramm, das es auch wirklich wert ist, durch eine Anfang/Ende-Kombination in abgekürzter Form memoriert zu werden? Damit eng verwandt ist die Frage: Welche in der Umgebung sichtbaren Ereignissequenzen 'gehören zusammen'? Diese Fra-

gen haben offensichtlich mit *unüberwachter Regularitätsentdeckung* zu tun. Sie zielen ab auf die in Ereignis- und Aktionssequenzen enthaltene *kausale Struktur*.

Um die kausale Struktur einer dynamischen Umgebung in effizienter Weise zu repräsentieren, sei hier folgende grundlegende Idee zur Messung kausaler Abhängigkeiten vorgeschlagen:

Als eine zusammengehörige Untersequenz soll eine Ereignissequenz dann betrachtet werden, wenn es während ihres Ablaufs für einen adaptiven Vorhersager einfach ist, die Eingabe zu einem gegebenen Zeitpunkt aus Eingaben vergangener Zeitpunkte vorherzusagen [62]. Der adaptive Prophet kann durch irgendeinen überwachten Algorithmus für dynamische rekurrente Netze trainiert werden. *Wann immer es eine Diskrepanz zwischen Erwartung und Realität gibt, existiert auch ein Grund, für die Untersequenz, welche die Zeit* zwischen *den letzten beiden unerwarteten Ereignissen überbrückt, einen neuen Namen zu generieren* (wir nehmen einfach die Kombination aus Anfangs- und Endzustand der Untersequenz).

Dieses Vorgehen ist in einem gewissen Sinne sehr natürlich und effizient: *Erwartete* Ereignisse braucht sich ein lernendes System ja nicht zu merken, sie können aus dem *deduziert* werden, was das System schon weiß.

Kausalitätsaufdeckung zielt auf die Reduktion der Repräsentation der externen Dynamik. Es wird versucht, eine 'minimale' Beschreibung der kausalen temporalen Struktur der Umgebung zu erhalten. Natürlich ist 'Minimalität' in diesem Kontext *relativ* zum gegewärtigen Wissen des lernenden Systems zu sehen: Einer Umgebung mag eine tiefe Kausalstruktur zugrundeliegen, das Lernsystem mag jedoch unter Umständen weit davon entfernt sein, sie zu entdecken.

Der Leser beachte, daß *Kausalitätsdetektion* sehr verwandt ist mit dem, was er selbst ständig tut. Man erinnert sich bevorzugt an die unerwarteten, *merk*würdigen Ereignisse. Diejenigen Vorfälle, die sich ständig wiederholen, fallen mit der Zeit der Nichtbeachtung anheim.

Es sollte hier erwähnt werden, daß es wichtig ist, zwischen zwei Arten von unerwarteten Ereignisse zu unterscheiden: Solchen, die in Situationen auftreten, für die das System bereits gelernt hat, daß es keine zuverlässige Voraussage treffen kann, und solchen, die tatsächlich den vom System für zuverlässig eingeschätzten Voraussagen widersprechen. Es ist also notwendig, das Vertrauen des neuronalen Systems in seine eigenen Vorhersagen zu modellieren, was durch ein geeignetes adaptives Sub-netzwerk erreicht werden kann. Experimente mit derartigen Systemen seien jedoch auf weiterführende Arbeiten verschoben.

Natürlich gibt es beim zielgerichteten Lernen noch weitere interessante Kandidaten für *merk*würdige Situationen. Regularitätsdetektion sollte in hohem Maße durch adaptive selektive Aufmerksamkeit [64] und durch

augenblickliche Subziele beeinflußt sein. Es ist beabsichtigt, durch weitere Untersuchungen Kausalitätsdetektoren und Subzielgeneratoren für die Generierung von *Subzielhierarchien* zu einem kohärenten Ganzen zu verschmelzen.

8.6 Schlußwort und Ausblick

Die vorliegende Arbeit lieferte Beiträge für die Lösung gewisser Aspekte des *'fundamentalen raumzeitlichen Lernproblems'* in zeitlich variierenden reaktiven Umgebungen.

Nicht angesprochen wurde das Problem des *'Meta*-Lernens'. 'Meta-Lernen' sollte darauf abzielen, die Art und Weise, *wie* gelernt wird, *selbst* weitgehend adaptiv zu machen. Langfristig will man nicht auf einem bestimmten vorgegebenen Lernverfahren (Gradientenabstieg mit Systemidentifikation, oder adaptive Kritiker mit TD-Methoden etc.) beharren, sondern das Lernverfahren *selbst* abhängig vom Umgebungskontext sinnvoller Modifikation zugänglich machen.

In diesem Zusammenhang sei ein sehr interessanter Aspekt des Konzepts der 'Modellnetzwerke' erwähnt. Ein Modellnetzwerk kann nicht nur (wie bei A2) dazu benützt werden, die *Eingaben* eines Steuernetzwerkes vorherzusagen. Es kann auch zur Modellierung der zu erwartenden *Ausgaben* herangezogen werden. Ein *perfektes* Modellnetzwerk dieser Sorte modelliert auf indirekte Weise unter anderem auch die internen Gewichtsänderungen des Steuernetzwerkes. Es modelliert die Evolution des Steuernetzwerkes, und damit auch die Effekte der Gradientenabstiegsprozedur selbst. Der *Aktivations*fluß in solch einem Modellnetzwerk beschreibt also die *Gewichts*änderungen im Steuernetz.

Das hat viel mit *Meta*-Lernen zu tun: Beim Meta-Lernen geht es unter anderem darum, zu *lernen*, Aussagen über die Effekte von Lernprozeduren *selbst* zu machen. Besitzt ein lernendes System ein gutes Modell der Effekte seiner eigenen Lernprozeduren, so liegt der Gedanke an den nächsten Schritt nahe: Solch ein System sollte sein Modell auch dazu verwenden, zu lernen, in sinnvoller Weise Einfluß zu nehmen auf die Art und Weise, in der es unter bestimmten Umständen überhaupt erst gewisse Lernmechanismen in Gang setzt. Es wurden bereits Arbeiten an einer Architektur begonnen, die es dem lernenden System selbst erlauben soll, mit Hilfe *interner Aktionen* die Art und Weise zu manipulieren, in der es Assoziationen kreiert, Aufmerksamkeit lenkt und Subzielgeneration triggert. Introspektive Module sollen dabei lernen, Vorhersagen über die Effekte der Lernprozeduren selbst zu machen. Diese Vorhersagen sollen mit Hilfe der oben beschriebenen Kausalitätsdetektoren zur adaptiven Adjustierung der internen Aktionen dienen.

Obwohl neuronale Systeme mit *introspektiven* Fähigkeiten schon für sich selbst genommen interessant sind und langfristig vielleicht geradezu *notwendig* sein werden [48], sei ihre Untersuchung auf weiterführende Arbeiten verschoben.

Anhang A

Mathematische Details

Details der Wagen/Stab Simulation

Das in den Kapiteln 4, 5, und 6 verwendete physikalische Wagen/Stab-System wurde durch folgende Differentialgleichungen modelliert:

$$\ddot{\theta} = \frac{gsin\theta + cos\theta \frac{-F - ml\dot{\theta}^2 sin\theta + \mu_c sgn(\dot{z})}{m_c+m} - \frac{\mu_p \dot{\theta}}{ml}}{l(\frac{4}{3} - \frac{mcos^2\theta}{m_c+m})},$$

$$\ddot{z} = \frac{F + ml(\dot{\theta}^2 sin\theta - \ddot{\theta}cos\theta) - \mu_c sgn(\dot{z})}{m_c + m}.$$

Dabei ist $-0.21 < \theta < 0.21$ der Winkel des Stabes mit der Vertikalen, $-2.4m < z < 2.4m$ die Position des Wagens auf der Spur, $g = 9.8\frac{m}{s^2}$ die Gravitationsbeschleunigung, $m_c = 1kg$ die Masse des Wagens, $m = 0.1kg$ die Masse des Stabes, $l = 0.5m$ die halbe Stablänge, $\mu_c = 0.0005$ der Reibungskoeffizient des Wagens auf der Bahn, $\mu_p = 0.000002$ der Reibungskoeffizient des Stabes auf dem Wagen, $F \in \{-10N, 10N\}$ bzw. $F \in [-25N, 25N]$ für A1 bzw. für A2 die auf den Schwerpunkt des Wagens parallel zur Spur ausgeübte Kraft. (Man beachte, daß die in [5], [66] und [2] angegebenen Gleichungen einen Tippfehler enthalten: Dort wurde die Gravitationsbeschleunigung jeweils als $g = -9.8\frac{m}{s^2}$ definiert.)

Für A1 und A3 wurden u. a. folgende skalierte Eingabevariablen verwendet: $\bar{z} = \frac{z+2.4}{4.8}$, $\bar{\dot{z}} = \frac{\dot{z}+1.5}{3}$, $\bar{\theta} = \frac{\theta+0.21}{0.42}$, $\bar{\dot{\theta}} = \frac{\dot{\theta}+2}{4}$.

Für den modifizierten A2 gab es nur die beiden skalierten Eingabevariablen $\bar{z}$ und $\bar{\theta}$.

Weite Sprünge durch den Gewichtsraum im Falle spärlicher Codierung

Was uns beim Gradientenabstieg (siehe das Kapitel zum überwachten Lernen) wirklich interessiert, sind nicht so sehr die Minima, sondern die Nullstellen der Fehlerfunktion

$$E = \sum_p E_p,$$

wobei E_p der zu einem Eingabemusterpaar p gehörige Fehler ist. BP liefert uns für alle p

$$\frac{\partial E_p}{\partial \vec{w},}$$

wobei $\vec{w} = (w_1, .., w_n)^T$ der komplette Gewichtsvektor des Netzes ist. Gradientenabstieg erfordert eine Gewichtsänderung

$$\triangle \vec{w} = -\eta \left(\frac{\partial E_p}{\partial \vec{w}} \right)^T,$$

wobei η eine positive Lernrate ist. Was wir brauchen, ist eine gute Wahl für η. Wir berechnen η für jede Musterpräsentation neu, so daß der geänderte Gewichtsvektor

$$\hat{\vec{w}} = \vec{w} + \triangle \vec{w}$$

auf den Schnitt der n-dimensionalen Gewichtshyperebene (im $n+1$-dimensionalen Gewichts-Fehler-Raum) mit der durch den gegenwärtigen Fehler und den gegenwärtigen Gradienten definierten Geraden zeigt. Die Grundannahme dabei ist, daß die E_p lokal durch die tangentialen Hyperebenen approximiert werden können. (Man betrachte Abbildung A.1 für eine Illustration des eindimensionalen Falles.)

Etwas elementare Geometrie ergibt, daß für ein gegebenes p

$$\eta = \frac{E_p}{\sum_k (\frac{\partial E_p}{\partial w_k})^2}$$

gelten muß, wobei w_i das i-te Gewicht ist, und k alle Gewichtsindizes durchläuft. Falls der Gradient verschwindet, so wird η gleich Null gesetzt. In [49] wird experimentell an einigen Beispielen gezeigt, daß die Methode bei spärlicher Codierung zu einer bedeutenden Verringerung der Anzahl der Lernzyklen führen kann.

Abbildung A.1: Illustration eines Iterationsschrittes für den Fall eines eindimensionalen Gewichtsvektors.

Literaturverzeichnis

[1] L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 609–618, 1987.

[2] C. W. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1986.

[3] A. G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[4] A. G. Barto and M. I. Jordan. Gradient following without back propagation in layered networks. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 629–636, 1987.

[5] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.

[6] R. Bellman. *Adaptive Control Processes.* Princeton University Press, 1961.

[7] M. Compiani, D. Montanari, R. Serra, and G. Valastro. Classifier systems and neural networks. In E. R. Caianello, editor, *1st Workshop on Parallel Architectures and Neural Nets*, 1989.

[8] J. L. Elman. Finding structure in time. Technical Report CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

[9] S. E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie-Mellon Univ., 1988.

[10] M. Gherrity. A learning algorithm for analog fully recurrent neural networks. In *IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 1, pages 643–644, 1989.

[11] S. Grossberg. Adaptive pattern classification and universal recoding, 1: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, 23:187–202, 1976.

[12] D. O. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[13] A. Herz, B. Sulzer, R. Kühn, and J.L. van Hemmen. Hebbian learning reconsidered: Representation of static and dynamic objects in associative neural nets, 1988. Technischer Report 463, Sonderforschungsbereich 123, Universität Heidelberg.

[14] G. E. Hinton and T. E. Sejnowski. Learning and relearning in Boltzmann machines. In Rumelhart and McClelland [45], pages 282–317.

[15] Josef Hochreiter. Implementierung und Anwendung eines 'neuronalen' Echtzeit-Lernalgorithmus für reaktive Umgebungen , 1990. Fortgeschrittenenpraktikum, Institut für Informatik, Technische Universität München.

[16] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[17] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. of the National Academy of Sciences*, 79:2554–2558, 1982.

[18] R. Huber. Selektive visuelle Aufmerksamkeit: Untersuchungen zum Erlernen von Fokustrajektorien durch neuronale Netze, 1990. Diplomarbeit, Institut für Informatik, Technische Universität München.

[19] J. Jameson. A neurocontroller based on model feedback and the adaptive heuristic critic. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 2, pages 37–43, 1990.

[20] M. I. Jordan. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, San Diego, 1986.

[21] M. I. Jordan. Supervised learning and systems with excess degrees of freedom. Technical Report COINS TR 88-27, Massachusetts Institute of Technology, 1988.

[22] M. I. Jordan and R. A. Jacobs. Learning to control an unstable system with forward modeling. In *Proc. of the 1990 Connectionist Models Summer School, in press.* San Mateo, CA: Morgan Kaufmann, 1990.

[23] T. Kohonen. *Self-Organization and Associative Memory.* Springer, second edition, 1988.

[24] A. Lapedes and R. Faber. How neural nets work. In D. Z. Anderson, editor, *'Neural Information Processing Systems: Natural and Synthetic' (NIPS).* NY, American Institute of Physics, 1987.

[25] Y. LeCun. Une procédure d'apprentissage pour réseau à seuil asymétrique. *Proceedings of Cognitiva 85, Paris*, pages 599–604, 1985.

[26] G. Lukes. Review of Schmidhuber's paper 'Recurrent networks adjusted by adaptive critics'. *Neural Network Reviews*, 4(1):41–42, 1990.

[27] G. Lukes, B. Thompson, and P. Werbos. Expectation driven learning with an associative memory. In *Proc. IEEE International Joint Conference on Neural Networks, Washington, D. C.*, volume 1, pages 521–524, 1990.

[28] M. Minsky. Steps toward artificial intelligence. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 406–450. McGraw-Hill, New York, 1963.

[29] M. Minsky and S. Papert. *Perceptrons.* Cambridge, MA: MIT Press, 1969.

[30] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. Technical report, BBN Systems and Technologies, Inc., Cambridge, MA, 1989.

[31] P. W. Munro. A dual back-propagation scheme for scalar reinforcement learning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA*, pages 165–176, 1987.

[32] F. Nake. *Ästhetik als Informationsverarbeitung.* Springer, 1974.

[33] K. S. Narendra and M. A. L. Thathatchar. Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 4:323–334, 1974.

[34] Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *IEEE/INNS International Joint Conference on Neural Networks, Washington, D.C.*, volume 1, pages 357–364, 1989.

[35] D. B. Parker. Learning-logic. Technical Report TR-47, Center for Comp. Research in Economics and Management Sci., MIT, 1985.

[36] D. B. Parker. Optimal algorithms for adaptive networks: Second order back propagation, second order direct propagation, and second order hebbian learning. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 593–600, 1987.

[37] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269, 1989.

[38] F. J. Pineda. Dynamics and architecture for neural computation. *Journal of Complexity*, 4:216–245, 1988.

[39] A. J. Robinson. *Dynamic Error Propagation Networks.* PhD thesis, Trinity Hall and Cambridge University Engineering Department, 1989.

[40] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

[41] T. Robinson and F. Fallside. Dynamic reinforcement driven error propagation networks with application to game playing. In *Proceedings of the 11th Conference of the Cognitive Science Society, Ann Arbor*, pages 836–843, 1989.

[42] R. Rohwer. The 'moving targets' training method. In J. Kindermann and A. Linden, editors, *Proceedings of 'Distributed Adaptive Neural Information Processing', St.Augustin, 24.-25.5,*. Oldenbourg, 1989.

[43] R. Rohwer and B. Forrest. Training time-dependence in neural networks. In *IEEE 1st International Conference on Neural Networks, San Diego*, volume 2, pages 701–708, 1987.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In Rumelhart and McClelland [45], pages 318–362.

[45] D. E. Rumelhart and J. L. McClelland, editors. *Parallel Distributed Processing*, volume 1. MIT Press, 1986.

[46] D. E. Rumelhart and D. Zipser. Feature discovery by competitive learning. In Rumelhart and McClelland [45], pages 151–193.

[47] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:210–229, 1959.

[48] J. H. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook, 1987. Report, Institut für Informatik, Technische Universität München.

[49] J. H. Schmidhuber. Accelerated learning in back-propagation nets. In R. Pfeifer, Z. Schreter, Z. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, pages 429 – 438. Amsterdam: Elsevier, North-Holland, 1989.

[50] J. H. Schmidhuber. The neural bucket brigade. In R. Pfeifer, Z. Schreter, Z. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, pages 439–446. Amsterdam: Elsevier, North-Holland, 1989.

[51] J. H. Schmidhuber. Applying temporal difference methods to fully recurrent reinforcement learning networks. *In preparation*, 1990.

[52] J. H. Schmidhuber. Learning algorithms for networks with internal and external feedback. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 52–61. San Mateo, CA: Morgan Kaufmann, 1990.

[53] J. H. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1(4):403–412, 1990.

[54] J. H. Schmidhuber. Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report FKI-126-90 (revised), Institut für Informatik, Technische Universität München, November 1990. (Revised and extended version of an earlier report from February.).

[55] J. H. Schmidhuber. Networks adjusting networks. In J. Kindermann and A. Linden, editors, *Proceedings of 'Distributed Adaptive Neural Information Processing', St.Augustin, 24.-25.5. 1989*, pages 197–208. Oldenbourg, 1990. In November 1990 a revised and extended version appeared as FKI-Report FKI-125-90 (revised) at the Institut für Informatik, Technische Universität München.

[56] J. H. Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 2, pages 253–258, 1990.

[57] J. H. Schmidhuber. Recurrent networks adjusted by adaptive critics. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, Washington, D. C.*, volume 1, pages 719–722, 1990.

[58] J. H. Schmidhuber. Reinforcement learning with interacting continually running fully recurrent networks. In *Proc. INNC International Neural Network Conference, Paris*, volume 2, pages 817–820, 1990.

[59] J. H. Schmidhuber. Reinforcement-Lernen und adaptive Steuerung. *Nachrichten Neuronale Netze*, 2:1–3, 1990.

[60] J. H. Schmidhuber. Response to G. Lukes' review of 'Recurrent networks adjusted by adaptive critics'. *Neural Network Reviews*, 4(1), 1990.

[61] J. H. Schmidhuber. Temporal-difference-driven learning in recurrent networks. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 209–212. North-Holland, 1990.

[62] J. H. Schmidhuber. Towards compositional learning with dynamic neural networks. Technical Report FKI-129-90, Institut für Informatik, Technische Universität München, 1990.

[63] J. H. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In J. A. Meyer and S. W. Wilson, editors, *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 222–227. MIT Press/Bradford Books, 1991.

[64] J. H. Schmidhuber and R. Huber. Learning to generate focus trajectories for attentive vision. Technical Report FKI-128-90, Institut für Informatik, Technische Universität München, 1990.

[65] B. Schürmann. Stability and adaptation in artificial neural systems. *Physical Review*, A 40(50):2681–2688, 1989.

[66] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1984.

[67] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[68] R. S. Sutton. First results with DYNA, an integrated architecture for learning, planning and reacting. In *Proceedings of the AAAI Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*, 1990.

[69] R. S. Sutton and B. Pinette. The learning of world models by connectionist networks. *Proceedings of the 7th Annual Conference of the Cognitive Science Society*, pages 54–64, 1985.

[70] C. v.d. Malsburg. Technical Report Internal Report 81-2, Abteilung für Neurobiologie, Max-Planck Institut für Biophysik und Chemie, Göttingen, 1981.

[71] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

[72] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[73] P. J. Werbos. Advanced forecasting methods for global crisis warning and models of intelligence. In *General Systems*, volume XXII, pages 25–38, 1977.

[74] P. J. Werbos. Backpropagation and neurocontrol: A review and prospectus. In *IEEE/INNS International Joint Conference on Neural Networks, Washington, D.C.*, volume 1, pages 209–216, 1989.

[75] P. J. Werbos. Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, 2:179–189, 1990.

[76] S.D. Whitehead and D. H. Ballard. Active perception and reinforcement learning. Technical Report 331, University of Rochester, Dept. of Comp. Sci., 1990.

[77] R. J. Williams. On the use of backpropagation in associative reinforcement learning. In *IEEE International Conference on Neural Networks, San Diego*, volume 2, pages 263–270, 1988.

[78] R. J. Williams. Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Comp. Sci., Northeastern University, Boston, MA, 1988.

[79] R. J. Williams and Leemon C. Baird. Draft: A mathematical analysis of actor-critic architectures for learning optimal controls through incremental dynamic programming. Technical report, College of Comp. Sci., Northeastern University, Boston, MA, 1990.

[80] R. J. Williams and D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.