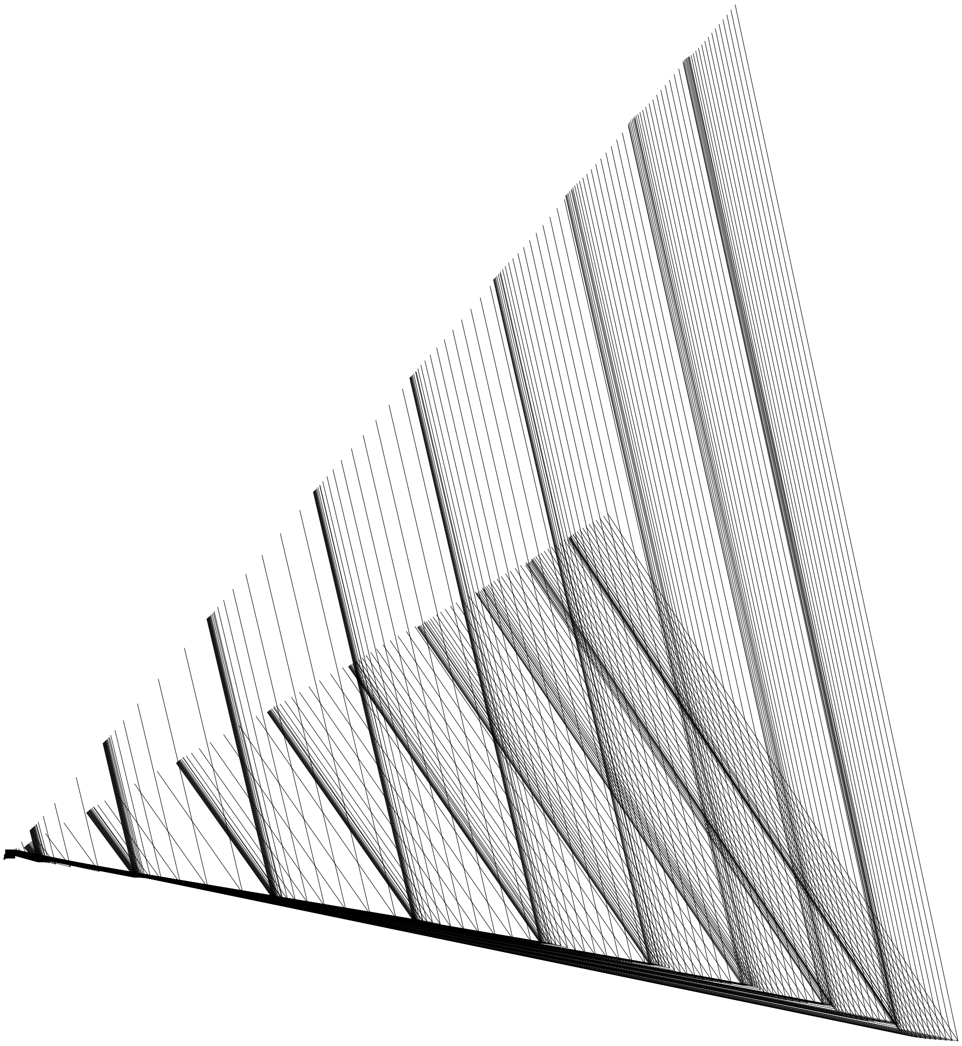


Backpropagation und Neurocontrol

Sebastian Thrun
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Email: thrun@cmu.edu

Backpropagation und Neurocontrol

Diplomarbeit
im Fachbereich Informatik

vorgelegt von
Sebastian Thrun

Rheinische Friedrich-Wilhelms-Universität Bonn

November 1992

ERKLÄRUNG: Hiermit erkläre ich an Eides statt, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Meinen Eltern gewidmet.

Vorwort

Künstliche Neuronale Netze, oft als Abbilder der „echten“ Neuronalen Netze im Menschlichen Gehirn glorifiziert, erfreuen sich seit Mitte der Achtziger Jahre eines neuen Forschungsbooms. Ausgehend von der Beobachtung, daß die leistungsfähigsten derzeit verfügbaren Computer zwar um Zehnerpotenzen schneller schalten können als Nervenzellen im Gehirn, aber dennoch völlig scheitern, wenn es um primitiv intelligentes Verhalten wie zum Beispiel das Erkennen von Gesichtern geht, versuchen Wissenschaftler aller Welt, die Prinzipien des Gehirns zu erforschen, um sie auf den Computern von morgen zu implementieren.

Anno domino 1992, inmitten eines nun knapp zehn Jahre andauernden weltweiten Booms, ist die Forschung im Bereich *Neuronaler Netze* weit von ihren zum Teil recht ehrgeizigen Zielen entfernt. Dabei spaltet sich dieses interdisziplinäre Feld in diverse Richtungen auf: Philosophen und so manche Kognitionswissenschaftler versuchen, die Plausibilität von verteilten, parallelen Konstrukten mit philosophischen und geisteswissenschaftlichen Methoden herauszuarbeiten. Desweiteren gibt es viele Biologen, die an der möglichst naturgetreuen Modellierungen von Nervenzellen und kleineren Zellverbänden arbeiten. Die Simulation menschlichen Verhaltens mit einfachen Netzmodellen fällt in die Sparte der Psychologie, während Physiker und Mathematiker Neuronale Netze im Rahmen von oszillierenden und chaotischen Systemen studieren. Diese Arbeit fällt in keine dieser Kategorien.

Die vorliegende Arbeit ist eine pragmatische Beschreibung von Forschungsarbeiten im Bereich Informatik und Ingenieurswissenschaften, Untergebiet Künstliche Intelligenz (KI) und Maschinelles Lernen. Dabei werden Neuronale Netze als Funktionsapproximatoren aufgefaßt, die eine gegebene Menge von Eingabe-Ausgabe-Tupeln approximieren. Parallelität, Selbstorganisation und verteilte Berechnung innerhalb solcher Netze, um nur einige Schlagworte der Branche zu benennen, spielen dabei eine untergeordnete Rolle. Diese Arbeit bezieht sich ausschließlich auf den Algorithmus Backpropagation, jenem Neuronalen Lernalgorithmus, der zur Zeit die Fachliteratur im Bereich Neurocontrol mit mehr als 80% aller Publikationen dominiert. Dabei werden wir uns insbesondere Anwendungen dieses Algorithmus im Bereich *„control“* widmen. Beispiele hierfür sind Robotersysteme jeder Art, Denkspiele wie Schach und Backgammon, heuristische Suche in Suchbäumen, Autopiloten in Flugzeugen, kurz alle Domänen, wo ein System mit Aktionen den Zustand seiner „Welt“ gezielt beeinflussen kann, und dabei lernen soll, Aktionen zu optimieren.

Wie liest man diese Arbeit? Natürlich kann man diese Arbeit von der ersten bis zur letzten Seite lesen, hat man die erforderliche Zeit und Kurzweil. Diese Arbeit ist jedoch modular aufgebaut; je nach Vorwissen und Interesse bietet es sich an, gewisse Teile zu überspringen.

Kapitel 1 stellt anschaulich mit vielen einfachen Beispielen das Grundkonzept von Künstlichen Neuronalen Netzen und Backpropagation vor. Dem Neuling auf diesem Gebiet sei die Lektüre dieses Kapitels empfohlen – Ist man mit diesem Algorithmus bereits vertraut und hat man Backpropagation bereits selbst hergeleitet und implementiert, wird einem dieses Kapitel allerdings nicht viel Neues verraten. Kapitel 2 behandelt Backpropagation deutlich formaler und erweitert diesen Algorithmus auf komplexere Netze, sogenannte rekurrente Netze. In diesem Kapitel stellen wir oft verwendete Netzarchitekturen und Heuristiken vor. Dieses Kapitel ist zum einen als Vertiefung des Stoffes von Kapitel 1 gedacht, wendet sich zum anderen aber besonders an solche Leser, die Backpropagation implementieren wollen.

Ist man hauptsächlich an Kontrollprinzipien interessiert, kann man auch direkt in Kapitel 3 einsteigen. Dieses Kapitel beschreibt grundlegende Konzepte zur Aktionssuche und Optimierung in Neurocontrol, baut dabei aber kaum auf der Lektüre der beiden ersten Kapitel auf. Kapitel 4 beschreibt eine spezielle Kontrollarchitektur, die im Rahmen dieser Diplomarbeit entwickelt und getestet wurde. Dieses Kapitel ist eine Fortsetzung von Kapitel 3. Schließlich, in Kapitel 5, wenden wir uns einem Thema zu, das in der Neurocontrolliteratur gerade erst Aufmerksamkeit findet: Exploration. Auch dieses Kapitel beruht auf Arbeiten, die während meines Studiums entstanden sind, und die Lektüre von Kapitel 3 ist zum Verständnis dieses Kapitels ebenfalls erforderlich. Die Arbeit endet in Kapitel 6, dem Ausblick.

Am Ende eines jeden Kapitels findet sich eine ausführliche Literaturübersicht, die zum weiterführenden Studium gedacht ist.

Inhaltsverzeichnis

Danksagungen

Mein allererster Dank gilt meinen Betreuern an der Universität Bonn, den Herren Prof. Dr. Gerd Veenker und Dr. Knut Möller, sowie gleichermaßen Alexander Linden und Dr. Jörg Kindermann, meinen Betreuern an der Gesellschaft für Mathematik und Datenverarbeitung (GMD) in Sankt Augustin. Knut Möller, mit dem ich im Hauptstudium in Bonn wohl am engsten zusammengearbeitet habe, hat mir immer mit wertvollem Rat und Tat beiseite gestanden. Neben vielen anderen wissenschaftlichen Dingen hat er mir beigebracht, wie man Wissenschaft kritisch betreibt. Alexander Linden, Jörg Kindermann und Dr. Heinz Mühlenbein, wie auch Knut Möller, haben mich nicht nur mit tatkräftiger und sachkundiger Hilfe unterstützt und inspiriert, sondern mir auch die nötige Forschungsfreiheit und finanzielle Unterstützung zukommen lassen, für die ich ihnen sehr dankbar bin. Ohne die Loyalität und die wissenschaftliche Inspiration meiner Betreuer und Vorgesetzten wären die hier beschriebenen Arbeiten nicht entstanden. Schließlich hat Herr Prof. Veenker zusammen mit Knut Möller mir die Möglichkeit gegeben, schon zu Studentenzeiten an der Forschung und der Lehre an der Universität Bonn mitzuarbeiten. Ich habe dadurch viel nützliche Erfahrung gewonnen, die ich nicht missen möchte.

Des weiteren danke ich Uwe Schnepf für viele offene und inspirierende Diskussionen, die sich deutlich auf mein Verständnis der Dinge ausgewirkt haben. Die GMD-Arbeitsgruppe „Adaptive Systeme“ stand mir immer mit Rat und Tat und Sachverstand beiseite; insbesondere danke ich Christoph Windheuser, der mir auch vielen guten persönlichen Rat gegeben hat, Jörg Kindermann, Alexander Linden, Heinz Mühlenbein, Peter Hendricks und Frank Śmieja. Knut Möller, Michael Contzen, Michael Eichhard und Michael Faßbender haben zusammen mit mir an der Universität Bonn tatkräftig mit und gegen den Roboterarm gefochten und dabei auch Nachteinsätze nicht gescheut. Schließlich möchte ich mich bei meinem Freund Roman Englert, Stefan Hoffrichter und Frank Kimmlingen bedanken, mit denen ich im Rahmen eines Praktikums die in dieser Arbeit vorgestellten Ansätze auf dem Roboterarm implementiert habe. Roman Englert hat mir auch bei vielen Entscheidungen beiseite gestanden.

Diese Danksagung wäre nicht komplett, würde ich nicht zwei Personen danken, die in meiner bisherigen wissenschaftlichen Ausbildung die wohl zentralste Rolle gespielt haben: Prof. Tom Mitchell und Prof. Alex Waibel. Alex Waibel gab mir die Möglichkeit, an der Carnegie Mellon University (CMU) in Pittsburgh, USA, zu arbeiten. Zusammen mit Tom Mitchell hat er mich Forschung gelehrt. Tom Mitchell, mit dem ich hier am engsten zusammengearbeitet habe, hat mich wie kein Anderer in der Sichtweise der Dinge beeinflusst und hat mir vielfältige Möglichkeiten und Unterstützung gegeben, in der Forschung an der Carnegie Mellon University mitzuwirken.

Des weiteren bin ich zutiefst dankbar für den wissenschaftlichen Sachverstand und das Feedback von Tom Mitchells „Reinforcement Learning Group“. Endlose Diskussionen mit Lonnie Chrisman, Long-Ji Lin, Sven König und Rich Caruana haben mich sehr bereichert. Des weiteren möchte ich mich bei Dr. Steve Suddarth für Freundschaft und Rat, bei Dr. Don Sofge für wichtige Diskussionsnächte während zweier Konferenzen, bei Prof. Michael Jordan für viele augenöffnende Tips, desgleichen bei Prof. Chris Atkeson und Dr. Andrew W. Moore, und ganz besonders bei Prof. Andy Barto und Satinder Singh bedanken. Letztere haben mir immer wieder neue wissenschaftliche Anstöße gegeben.

Ich bedanke mich bei Knut Möller, Rainer Holve, Michael Berthold und Thomas Thielke für das Korrekturlesen früherer Versionen dieser Arbeit.

Wer noch? Die wichtigste Person während meines Studiums war und ist meine langjährige Freundin Petra Dierkes. Es gibt so viele Dinge, für die ich dankbar bin, daß ich gar nicht erst den Versuch machen möchte, sie aufzuzählen. In Hinblick auf meinen bisherigen wissenschaftlichen Fortgang möchte ich ihr aber ausdrücklich für das viele Verständnis danken, das sie einem „Workaholic“ hat angedeihen lassen. Ich möchte natürlich auch ganz besonders meinen Eltern danken, die mich in all meinen Entscheidungen sehr unterstützt haben – Meine Mutter hat das Ende meines Studiums leider nicht mehr miterleben können.

Keine Forschung ohne Funding! Diese Arbeiten sind von der Gesellschaft für Mathematik und Datenverarbeitung, der Universität Bonn, Siemens Corp. in Princeton und dem Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 unter Vertrag Nummer F33615-90-C-1465, Arpa Order No. 7597 (DARPA), und nicht zuletzt von meinem Vater finanziell unterstützt worden. Die Ansichten und Schlußfolgerungen in diesem Dokument sind die des Autors und sollten nicht als die offizielle Politik, explizit oder implizit, der Universität Bonn, der GMD, Siemens Corp, der US-Regierung oder meines Vaters interpretiert werden. Nichtsdestotrotz bin ich für diese Unterstützung sehr dankbar.

Kapitel 1

Einleitung

1.1 Künstliche Neuronale Netze – die Idee der parallelen verteilten Berechnung

Das Forschungsgebiet der Anwendung massiv paralleler Prinzipien zum Design von Computern mit bisher unbekannten Möglichkeiten erfreut sich seit Mitte der Achtziger Jahre eines neuen Booms. Es zeichnet sich ab, daß massiv parallele Hardware mit 10^6 oder auch deutlich mehr Prozessoren nach anderen Prinzipien als heute verfügbare Rechner gebaut und daher auch programmiert werden müssen.

Interessanterweise gibt es in der Natur ein beeindruckend funktionierendes Vielprozessorsystem, das seit wenigen Jahren verstärkt zum Vorbild für massiv parallele Programme gereicht: das Gehirn. Diese durch seine vielseitigen Möglichkeiten bestechende „wet ware" besteht beim Menschen aus ca. 10^{11} Nervenzellen, den Neuronen, die über durchschnittlich etwa 10^4 Verbindungen (Axonen, Dendriten und Synapsen) pro Neuron miteinander verbunden sind und in massiv-paralleler Manier Daten in Form von Aktionspotentialen austauschen und verarbeiten.

Nun gibt es eine Vielzahl mathematischer Beschreibungen eines Neurons. Biologen, auf der einen Seite, versuchen, beobachtete dynamische Effekte einzelner Nervenzellen mathematisch zu modellieren und zu simulieren. Auf der anderen Seite arbeiten Computerwissenschaftler und Physiker mit einer Vielzahl rudimentärer Neuronenmodellen, die mathematisch so simplifiziert sind, daß sie mit einem Neuron bestenfalls noch die Idee der verteilten Berechnung gemein haben, aber dafür leicht auf einem Digitalcomputer simuliert werden können. Der Backpropagation-Algorithmus, der diesem Text zugrundeliegt, gehört zu dieser Kategorie. Basierend auf einer extrem simplifizierten Darstellung eines Neurons realisiert er einen nichtlinearen Approximationsalgorithmus, der einfach zu implementieren und für nichtlineare Approximationsprobleme mannigfaltiger Art anwendbar ist, und nicht zuletzt deshalb sich einer großen Beliebtheit erfreut.

Wir werden im folgenden Grundkonstrukte dieses Algorithmus anhand einfacher Beispiele vorstellen, ohne weiter auf die ohnehin rudimentäre biologische Motivation einzugehen. Dabei gliedert sich die Einführung im wesentlichen in drei Teile. Zuerst werden Künstliche Neuronale Netze vorgestellt und anhand einfacher Beispiele ihre Funktionsweise erläutert. Dann wird eine *Lernregel*, *Backpropagation*, eingeführt, die aus dem Netz einen Funktionsapproximator macht. Schließlich werden übersichtsartig verschiedenste Anwendungsmöglichkeiten des Backpropagation-Algorithmus skizziert.

1.1.1 Einfache Künstliche Neuronale Netze am Beispiel NICHT, UND und ODER

Ein Künstliches Neuronales Netz besteht aus einer Reihe von <u>Knoten</u>, die über gerichtete <u>Kanten</u> miteinander verbunden sind, und die sich über diese Kanten gegenseitig Daten in Form reeller Zahlen schicken und diese verarbeiten[1]. Dabei stellt ein Knoten eine Funktionseinheit dar, die lediglich multiplizieren, addieren und vergleichen kann. In einem Backpropagation-Netz gibt es zwei Arten von festen, reellwertigen <u>Parametern</u>: Zum einen ist jedem Knoten ein sogenannter <u>Schwellwert</u> zugeordnet, zum anderen ist jeder Kante ein Wert zugeordnet, der als <u>Gewicht</u> bezeichnet wird.

Mit jeder Wahl von Werten für diese Parameter stellt ein Künstliches Neuronales Netz eine Funktion dar. Die Menge aller Parameter repräsentiert also eine Klasse von Funktionen, die bestimmte Abschlußeigenschaften besitzt. Bei der Berechnung der durch die Parameter festgelegten Funktion ist jedem Knoten einen sogenannter <u>Aktivationswert</u> in Form einer reellen Zahl assoziiert. Knoten können multiplizieren, addieren und vergleichen, und zwar nach einem festen Zweischritt-Schema: Zuerst (Schritt 1) werden die Aktivationswerte der Vorgänger, so diese evaluiert sind, jeweils mit dem Gewicht der entsprechenden Kante (die den Vorgänger mit dem betrachteten Knoten verbindet) multipliziert und die Produkte aufsummiert. Dies ist eine Linearkombination: Ein Gewicht bestimmt also, wie stark die Aktivation des entsprechenden *sendenden* Knotens die Summe des *empfangenden* beeinflusst, da sie diese als Faktor *gewichtet*. Jeder Knoten vergleicht danach (Schritt 2) diese gewichtete Summe, auch <u>Netzeingabe</u> genannt, mit seinem internen Schwellwert. Ist die Summe größer als der Schwellwert, wird der Knoten *aktiv* und nimmt den Aktivationswert 1 an – Ansonsten wird er *passiv* mit der Aktivation 0.[2]

Um diese Entscheidungsregel mathematisch auszudrücken, sei folgendes definiert: w_{ij} ($w_{ij} \in I\!R$) bezeichne das <u>Gewicht</u> der <u>Kante</u> vom j-ten Knoten zum i-ten Knoten (man beachte die Reihenfolge der Indizes) und θ_i ($\theta_i \in I\!R$) sei der <u>Schwellwert</u> vom i-ten Knoten. Damit ergibt sich für die <u>Netzeingabe</u> net_i ($net_i \in I\!R$) und den <u>Aktivationswert</u> x_i ($x_i \in \{0, 1\}$) von Knoten i folgende Gleichungen, die in der Literatur oft als <u>Aktivationsgleichungen</u> bezeichnet werden:

[1] wir wählen bewußt nicht die in Teilen der Literatur gängige Bezeichnung *Neuronen* und *Synapsen*.

[2] Um die Funktionsweise Künstlicher Neuronaler Netze möglichst einleuchtend zu illustrieren, gehen wir vorerst von dieser, wie sich herausstellen wird, etwas vereinfachten Verarbeitungsweise aus. Die Restriktion auf binäre Aktivationswerte wird später zugunsten reellwertiger im Intervall [0, 1] aufgehoben.

Abbildung 1.1: Funktionsweise eines Knotens v_i: Die Vorgängeraktivationen – hier mit x_1, x_2 und x_3 bezeichnet – werden mit den entsprechenden Kantengewichten w_{i1}, w_{i2} und w_{i3} multipliziert. Die Summe dieser Produkte minus dem Schwellwert θ_i ergibt die Netzeingabe net_i. Ist dieser größer 0, ist die Aktivation des Knotens 1, und sonst 0.

$$
\begin{aligned}
net_i &\equiv \left(\sum_{j \in P(i)} w_{ij} \cdot x_j \right) - \theta_i \\
x_i &\equiv \begin{cases} 1 & \text{wenn } net_i > 0 \\ 0 & \text{wenn } net_i \leq 0 \end{cases}
\end{aligned}
\tag{1.1}
$$

Dabei bezeichnet $P(i)$ die Menge der Vorgängerknoten von Knoten i, also diejenigen j, für die eine Kante zum Knoten i existiert. Die Aktivationsgleichungen sind in Abbildung 1.1 graphisch zusammengefaßt.

Mit Gleichung (1.1) wissen wir nun, wie innerhalb eines Künstlichen Neuronalen Netzes reelle Zahlen, die Aktivationen, berechnet werden[3]. Um dies als mathematische Funktion vom $I\!R^m$ in $\{0,1\}^n \subset I\!R^n$ zu interpretieren, fehlt noch eine Anbindung von Eingabe und Ausgabe. Zur Eingabe: Es hat sich eingebürgert, den m-dimensionalen reellwertigen Eingabevektor direkt als Aktivationswerte von m gesondert gekennzeichneten Knoten des Netzes, den sogenannten Eingabeknoten, zu kopieren. Dieser Vektor ist gewöhnlich auf das Intervall $[0,1]^m$ beschränkt. Die Eingabeknoten wenden also nicht Regel (1.1) an und haben weder Gewichte noch Schwellwerte, weshalb sie oft als *Pseudoknoten* betrachtet werden. Die Ausgabe des Netzes sind dann die resultierenden n Aktivationswerte ebenfalls gesondert gekennzeichneter Ausgabeknoten. Diese

[3]Daraus folgt unmittelbar, daß in der Graphenstruktur kein Zyklus vorkommen darf, da zur Aktivationsberechnung in Knoten i alle Vorgänger bereits ihre Aktivation berechnet haben müssen.

Abbildung 1.2: (a) Ein Netz, das die boolsche Funktion NICHT realisiert. In (b) wird die Eingabe 0 ausgewertet, in (c) 1.

Knoten berechnen ihre Aktivationen gemäß (1.1). Im übrigen werden Knoten, die weder Eingabe- noch Ausgabeknoten sind, als interne Knoten (*hidden*, verborgen) bezeichnet.

Wir werden nun zeigen, wie ein Netz eine Funktion berechnet, indem wir für einige elementare Funktionen Netze designen. Bisher läßt die Aktivationsfunktion (1.1) nur binäre Aktivationswerte, nämlich 0 oder 1 zu. Es liegt daher nahe, auch für die Eingabe nur solche zu betrachten. Damit realisiert ein Netz eine *boolsche Funktion*.

In Abbildung 1.2 ist ein Netz gezeigt, das die boolsche Funktion NICHT berechnet. Gibt man als Eingabe den Wert 0 (Abb 1.2 (b)), so ergibt sich mit dem Kantengewicht -1 das Produkt -1, und mit dem Schwellwert -0.5 die Netzeingabe 0.5 (vgl. Gleichung (1.1)). Ein Vergleich mit 0 liefert die Ausgabeaktivation 1. Genauso zeigt man, daß die Eingabe 1 die Netzeingabe -0.5 bedingt; der Vergleich mit 0 liefert dann 0. Damit ist gezeigt, daß dieses Netz die boolsche Funktion NICHT realisiert.

Interessanter sind die Künstliche Neuronale Netze für die zweistelligen boolschen Funktionen UND und ODER. In Abbildung 1.3 sind mögliche Realisierungen für diese beiden Funktionen dargestellt. Folgende Tabellen geben die entsprechenden Berechnungen an:

Abbildung 1.3: Zwei Künstliche Neuronale Netze für die boolschen Funktionen ODER und UND.

UND						
Eingabe Knoten #1	Eingabe Knoten # 2	$w_{31} \cdot x_1$	$w_{32} \cdot x_1$	θ_3	Netzeingabe net_3	Ausgabe Knoten # 3
0	0	0	0	1.5	-1.5	0
0	1	0	1	1.5	-0.5	0
1	0	1	0	1.5	-0.5	0
1	1	1	1	1.5	0.5	1

ODER						
Eingabe Knoten #1	Eingabe Knoten # 2	$w_{31} \cdot x_1$	$w_{32} \cdot x_1$	θ_3	Netzeingabe net_3	Ausgabe Knoten # 3
0	0	0	0	0.5	-0.5	0
0	1	0	1	0.5	0.5	1
1	0	1	0	0.5	0.5	1
1	1	1	1	0.5	1.5	1

Die explizite Darstellung der drei Funktionen NICHT, UND und ODER als Künstliches Neuronales Netz zeigt übrigens, daß, unter Benutzung eines aus der Mengenalgebra bekannten Satzes, jede boolsche Funktion durch ein Künstliches Neuronales Netz realisiert werden kann. Prinzipiell ist also die Klasse der Künstlichen Neuronalen Netze sehr mächtig.[4]

[4]Im folgenden Kapitel werden wir Künstliche Neuronale Netze um Rekurrenzen erweitern, die die Darstellung beliebiger endlicher Automaten ermöglicht. Diese Ergebnisse sind allerdings von fraglicher praktischer Relevanz für die derzeitige Forschung im Bereich Neuronaler Netze.

Abbildung 1.4: Bei der UND- und ODER-Funktion trennt eine Gerade den Eingaberaum in zwei Halbräume. Dabei markiert der graue Bereich den Ausgabewert 1, der weiße den Wert 0.

1.1.2 Mehrschichtige Netze am Beispiel XOR

Sucht man nun nach diesen einleitenden Beispielen die XOR-Funktion, also die boolsche Funktion $XOR : \{0,1\}^2 \longrightarrow \{0,1\}$, die genau dann 1 ist, wenn genau eine Eingabe 1 ist, durch ein Künstliches Neuronales Netz gleicher Architektur zu realisieren, wird man scheitern. Warum?

Wir werden diese Frage geometrisch beantworten. In den beiden letzten Beispielen war die Netzeingabe durch

$$net_3 = \left(\sum_{j=1,2} w_{3j} \cdot x_j \right) - \theta_3 \tag{1.2}$$

gegeben. Der Test $net_3 > 0$ definiert geometrisch einen Halbraum, dessen Rand durch die Gleichung

$$net_3 = 0 \tag{1.3}$$

beschrieben wird. Da net_3 Linearkombination über x_1 und x_2 ist, definiert Gleichung (1.3) eine Gerade im x_1-x_2-Raum. Diese ist für die Beispiele UND und ODER in Abbildung 1.4 gezeigt.

Die Erkenntnis, daß es sich bei der Lösung von (1.3) um eine Gerade handelt, ist unabhängig von den konkreten Werten für die Gewichte und den Schwellwert – sie ist also eine direkte Konsequenz der Architektur. Man überlegt sich leicht, daß sich dieses Argument auf beliebige

Abbildung 1.5: Ein zweischichtiges Netz löst das XOR-Problem. Der interne Knoten 3 berechnet UND, Knoten 4 ODER. XOR läßt sich dann auf der zweiten Schicht linear separieren.

Künstliche Neuronale Netze mit m Eingabeknoten und n Ausgabeknoten verallgemeinern läßt, sofern die Ausgabeknoten direkt mit den Eingabeknoten verbunden sind, d.h., die Netzeingabe aller Ausgabeknoten eine lineare Kombination des Eingabevektors ist.

Der Grund, warum ein solches Netz das XOR-Problem nicht lernen kann, liegt nun auf der Hand: Man kann den Eingaberaum beim XOR-Problem nicht linear separieren, d.h., es gibt keine Gerade, die alle 0-Lösungen von allen 1-Lösungen trennt. Dies ändert sich mit Einführen einer weiteren Schicht von Knoten, den internen Knoten, wie in Abbildung 1.5 gezeigt. Für dieses Netz ergibt sich:

XOR							
Eingabe		Netzeingabe		Aktivation		Netzeingabe	Ausgabe
x_1	x_2	net_3	net_4	x_3	x_4	net_5	x_5
0	0	-1.5	-0.5	0	0	-0.5	0
0	1	-0.5	0.5	0	1	0.5	1
1	0	-0.5	0.5	0	1	0.5	1
1	1	0.5	1.5	1	0	-0.5	0

Tabelle 1.1: Berechnung der XOR-Funktion.

In der Tat ist die Klasse der Funktionen, die ein Netz mit einer internen Schicht repräsentieren kann, deutlich größer als die eines Netzes ohne interne Knoten. Wie bereits oben gesagt, können Netze ohne interne Knoten nur genau eine Hyperebene zur Klassifikation repräsentieren, und folglich wird der Eingaberaum linear separiert. Durch das Hinzufügen von internen Knoten kann ein Netz auch nichtlinear separieren, da jeder der internen Knoten eine eigene Hyperebene repräsentiert. Graphisch entspricht dies einer Klassifikation, die mit n Hyperebenen beschrieben werden kann, wobei n die Anzahl der internen Knoten ist. So ist das XOR-Beispiel graphisch mit zwei Geraden, und daher mit zwei internen Knoten repräsentierbar.

Oft ist es üblich, Netze *schichtweise* aufzubauen. Dabei stellen die Eingabeknoten die 0-te Schicht dar, gefolgt von einigen internen Schichten. Die Ausgabeknoten bilden dann die letzte Schicht. Dabei sind nur aufeinanderfolgende Schichten durch Kanten verbunden, und zwar vollverbunden, d.h., von jedem Knoten der k-ten Schicht führt eine Kante zu jedem Knoten der $(k+1)$-ten Schicht. Demzufolge sind die in Abbildung 1.2 und 1.3 gezeigten Netze Einschichtnetze, und das Netz in Abbildung 1.5 ein Zweischichtnetz. Ein traditioneller Sammelbegriff für geschichtete Netze ist Multilayerperceptrons.

1.2 Lernen mit Gradientenabstieg

In den vorangegangenen Abschnitten haben wir elementare boolsche Funktionen durch explizite Angabe der Gewichts- und Schwellwerte realisiert. Nun ist das Finden solcher Gewichte gerade bei komplexeren Zielfunktionen und dadurch bedingt größeren Netzen oft ein komplizierter Vorgang. Wünschenswert wäre ein Algorithmus, der aus Eingabe-Ausgabe-Tupeln (sog. Trainingsbeispielen oder Trainingsmustern) einen Gewichtssatz extrahiert, der die gewünschte Funktion realisiert. Im Allgemeinen ist diese Suche derart aufwendig (abhängig vom verwendeten Netz kann sie $\mathcal{NP}$-vollständig sein), daß sich iterative, unvollständige Approximationsalgorithmen als vorteilhaft erwiesen haben. Hier startet man mit z.B. zufällig initialisierten Gewichten und Schwellwerten und verbessert diese iterativ mit dem Ziel, die Abweichung von der tatsächlichen Funktion und der durch die Beispiele manifestierten zu minimieren.

Um dies zu erreichen, muß das bisherige Modell für Künstliche Neuronale Netze erweitert werden. Die wichtigste Erweiterung ist, als Aktivationen reelle Werte im Intervall $(0, 1)$ zuzulassen, indem die Schwellwertfunktion in (1.1) durch eine differenzierbare Funktion ersetzt wird. Üblicherweise bedient man sich der logistischen Funktion, die folgendermaßen definiert ist:

$$x_i \quad = \quad \sigma(net_i) \equiv \frac{1}{1 + e^{-net_i}} \tag{1.4}$$

Diese differenzierbare Funktion (Abbildung 1.6) hat eine besondere Eigenschaft, die sie der Schwellwertfunktion ähnlich macht: Betreibt man sie im mittleren Eingabebereich (so etwa zwischen -2.0 und 2.0), dem Linearbereich, so verhält sie sich annähernd linear. Im Randbereich dagegen, dem Sättigungsbereich, approximiert sie die binäre Schwellwertfunktion. Darüber hinaus ist sie bijektiv und somit umkehrbar.

Diese Erweiterung, die dem binären Denken erst einmal zu widersprechen scheint, ermöglicht die Anwendung einer ganzen Reihe von Parameterschätzverfahren auf das Problem der Gewichts-

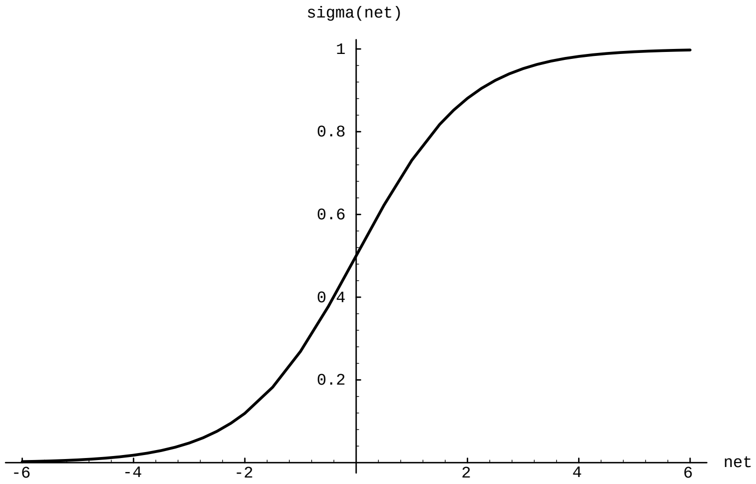


Abbildung 1.6: Logistische Funktion

und Schwellwertsuche. Das einfachste und in der Neuro-Literatur bekannteste solche Schätzverfahren ist wohl die Methode des steilsten Abstieges. Dazu definiert man eine differenzierbare Fehlerfunktion, die die quadratische Abweichung zwischen gewünschter und tatsächlicher Ausgabe mißt:

$$\begin{aligned} E &\equiv \sum_{p=1}^{P} E^p \\ \text{mit } E^p &\equiv \frac{1}{2} \sum_{i \text{ Ausgabe}} (d_i^p - x_i^p)^2 \;=\; \frac{1}{2} \sum_{i \text{ Ausgabe}} (d_i - x_i)^2 \end{aligned} \tag{1.5}$$

Hier indiziert p die Menge der Trainingsbeispiele, die Trainingsmenge $\mathcal{P}$. Dabei ist d_i^p als die erwünschte Ausgabe (*target*, *desired output*) der i-ten Komponente des p-ten Trainingsbeispiels (*pattern*, Muster) definiert, und x_i^p ist die entsprechende tatsächliche Ausgabe, eine Funktion der p-ten Eingabe. Mit dieser Fehlerfunktion wird das Lernproblem zu einem Optimierungsproblem reduziert. Ziel der Optimierung ist es, E zu minimieren.

Im folgenden werden wir anschaulich eine Regel für Gradientenabstieg für Gewichte und Schwellwerte herleiten. Ohne Beschränkung der Allgemeinheit betrachte man ein beliebiges Gewicht w_{ij} (analoges gilt für jeden Schwellwert θ_i). Der Fehler E kann als eine Funktion dieses Gewichtes $E = E(w_{ij})$ verstanden werden, indem man alle anderen Parameter – Gewichte und Schwellwerte – für einen Moment als konstant annimmt. Eine denkbare Situation ist in Abbildung 1.7 illustriert. Hier ist der Fehler Funktion eines Gewichtes. Man entnimmt diesem Graphen leicht, wie man das Gewicht ändern muß, um den Fehler zu minimieren: Mathematisch kann dies über die Steigung, also dem Gradienten der Fehlerfunktion bezüglich des Gewichtes, an der Stelle des momentanen Gewichtswertes ausgedrückt werden: Steigt der Graph, ist der Gradient positiv und das Gewicht muß verkleinert werden, also ein negativer Wert muß aufaddiert werden. Im reziproken Fall eines fallenden Graphen ist der Gradient negativ, und ein positiver Wert muß

Abbildung 1.7: Beispiel einer Fehlerfunktion: Bei momentanen Wert des Parameters (a) gibt die Steigung der Tangente an, in welche Richtung der Parameter verändert werden muß, um den Fehler zu verkleinern. Iteratives Anwenden dieser Regel in hinreichend kleinen Schritten kann zum globalen Minimum führen (a), aber auch zu schlechteren lokalen (b). Man kann sich Gradientenabstieg als eine massenlose Kugel in der Fehlerlandschaft vorstellen, die bergab zum nächsten Tal rollt.

---

aufaddiert werden. Nimmt man an, daß die Steilheit des Graphen anzeigt, wie weit man den Parameter ändern sollte, macht es Sinn, einfach ein Vielfaches der Steigung (des Gradienten) vom Parameter abzuziehen. Diese in der Mathematik seit geraumer Zeit bekannte Methode ist als Gradientenabstieg oder Methode des steilsten Abstieges in die Literatur eingegangen und soll nun auf Künstliche Neuronale Netze angewendet werde.

An dieser Stelle macht sich die Einführung einer differenzierbaren Aktivationsfunktion bezahlt. Mit der logistischen Funktion (1.4) als Squashingfunktion ist es möglich, den gewünschten Steigungsgradienten zu berechnen:

1. Die Fehlerfunktion E ist differenzierbar. Somit kann man die Ableitung des Fehlers nach der Ausgabeaktivation (jedes Ausgabeknotens i) berechnen, vermöge:

$$\begin{aligned} \frac{\partial E}{\partial x_i} &= \frac{\partial \frac{1}{2} \sum_{p=1}^{P} E^p}{\partial x_i} = \frac{1}{2} \sum_{p=1}^{P} \frac{\partial E^p}{\partial x_i} \\ &= \frac{1}{2} \sum_{p=1}^{P} \frac{\partial (d_i - x_i)^2}{\partial x_i} = - \sum_{p=1}^{P} (d_i - x_i) \end{aligned} \tag{1.6}$$

2. Die Aktivationsfunktion (1.4) ist ebenfalls differenzierbar:

$$
\begin{aligned}
\frac{\partial x_i}{\partial net_i} &= \sigma'(net_i) = \left(\frac{1}{1+e^{-net_i}}\right)' \\
&= \frac{e^{-net_i}}{(1+e^{-net_i})^2} = \frac{1+e^{-net_i}}{(1+e^{-net_i})^2} - \frac{1}{(1+e^{-net_i})^2} \\
&= \frac{1}{1+e^{-net_i}}\left(1 - \frac{1}{1+e^{-net_i}}\right) = \sigma(net_i)\cdot(1-\sigma(net_i)) \\
&= x_i \cdot (1-x_i)
\end{aligned}
\tag{1.7}
$$

3. Schließlich ist net_i nach dem Gewicht w_{ij} ableitbar:

$$
\frac{\partial net_i}{\partial w_{ij}} = \frac{\partial \sum\limits_{k\in\mathcal{P}(i)} w_{ik}\cdot x_k - \theta_i}{\partial w_{ij}} = \frac{\partial w_{ij}\cdot x_j}{\partial w_{ij}} = x_j \tag{1.8}
$$

   Hierbei wird ausgenutzt, daß für alle $k \in P(i)$ mit $k \neq j$ das Gewicht w_{ij} weder die Aktivation x_k, noch das Gewicht w_{ik} oder den Schwellwert θ_i beeinflusst, und die entsprechenden Gradienten folglich 0 sind.

   Analog läßt sich der Gradient von net_i nach dem Schwellwert θ_i ableiten:

$$
\frac{\partial net_i}{\partial \theta_i} = \frac{\partial \sum\limits_{k\in\mathcal{P}(i)} w_{ik}\cdot x_k - \theta_i}{\partial \theta_i} = -1 \tag{1.9}
$$

4. Faßt man obige Herleitungen mithilfe der Kettenregel zusammen, ergibt sich für die gesuchten Gradienten der Fehlerfunktion nach den freien, *internen Parametern* eines Künstlichen Neuronalen Netzes – den Gewichten und Schwellwerten:

$$
\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_i}\cdot\frac{\partial x_i}{\partial net_i}\cdot\frac{\partial net_i}{\partial w_{ij}} = -\sum_{p=1}^{P}(d_i - x_i)\cdot x_i\cdot(1-x_i)\cdot x_j \tag{1.10}
$$

$$
\frac{\partial E}{\partial \theta_i} = \frac{\partial E}{\partial x_i}\cdot\frac{\partial x_i}{\partial net_i}\cdot\frac{\partial net_i}{\partial \theta_i} = \sum_{p=1}^{P}(d_i - x_i)\cdot x_i\cdot(1-x_i) \tag{1.11}
$$

Diese Gradienten geben nun an, wie eine infinitesimal kleine Änderung des Parameters die Fehlerfunktion beeinflusst. Wie bereits oben motiviert, ändert man nun Parameter iterativ so, daß sie die Fehlerfunktion verkleinern, indem man ein Vielfaches des Gradienten subtrahiert:

$$
w_{ij} \longleftarrow w_{ij} - \eta\frac{\partial E}{\partial w_{ij}} \tag{1.12}
$$

$$
\theta_i \longleftarrow \theta_i - \eta\frac{\partial E}{\theta_i} \tag{1.13}
$$

Dabei ist η ($\eta > 0$) eine sog. Schrittweite, eine *geeignet* zu wählende heuristische Größe, auf deren Bestimmung wir hier nicht weiter eingehen werden.

1.2.1 Lernen der UND-Funktion mit Gradientenabstieg

Bevor wir die Lernregel auf Mehrschichtnetze erweitern, und damit den eigentlichen Backpropagation-Algorithmus vorstellen, werden wir am Beispiel der UND-Funktion die Arbeitsweise von Gradientenabstieg erläutern. Diese Funktion ist vollständig durch 4 Trainingsbeispiele charakterisiert:

Eingabe x_1	Eingabe x_2	$\longrightarrow$	Target d_3
0	0		0
0	1		0
1	0		0
1	1		1

1. Man initialisiere Gewichte und Schwellwert mit kleinen Zufallswerten, z.B.:

   $$w_{31} = -0.1432 \quad w_{32} = 0.0130 \quad \theta_3 = -0.1920$$

2. Nun propagiere man nacheinander alle Trainingsbeispiele durch das Netz. Dafür ergibt sich:

x_1	x_2	net_3	x_3	d_3	$E^p = \frac{1}{2}(d_3 - x_3)^2$
0.0	0.0	0.1920	0.5479	0	0.1501
0.0	1.0	0.2050	0.5511	0	0.1519
1.0	0.0	0.0488	0.5122	0	0.1312
1.0	1.0	0.0618	0.5154	1	0.1174

   Der quadratische Fehler ist nach (1.5) ist 0.5506.

3. Nun berechne man die Gradienten für die Gewichte und den Schwellwert gemäß (1.10) und (1.11):

   $$\frac{\partial E}{\partial w_{31}} = 0.00695 \quad \frac{\partial E}{\partial w_{32}} = 0.01531 \quad \frac{\partial E}{\partial \theta_3} = -0.27899$$

4. Ein Ändern der Parameter mit Schrittweite $\eta = 0.5$ ergibt:

   $$w_{31} = -0.14668 \quad w_{32} = 0.00535 \quad \theta_3 = -0.05251$$

   Berechnet man nun für diese Gewichte den Fehler, ergibt sich ein kleinerer Gesamtwert: $E = 0.5138$ (< 0.5506). Dieses Beispiel demonstriert die Arbeitsweise von Gradientenabstieg. Nach jedem Schritt wird bei geeigneter Wahl der Schrittweite η der Fehler kleiner oder, falls man sich in einem lokalem Minimum (oder einem Sattelpunkt) befindet, bleibt gleich.

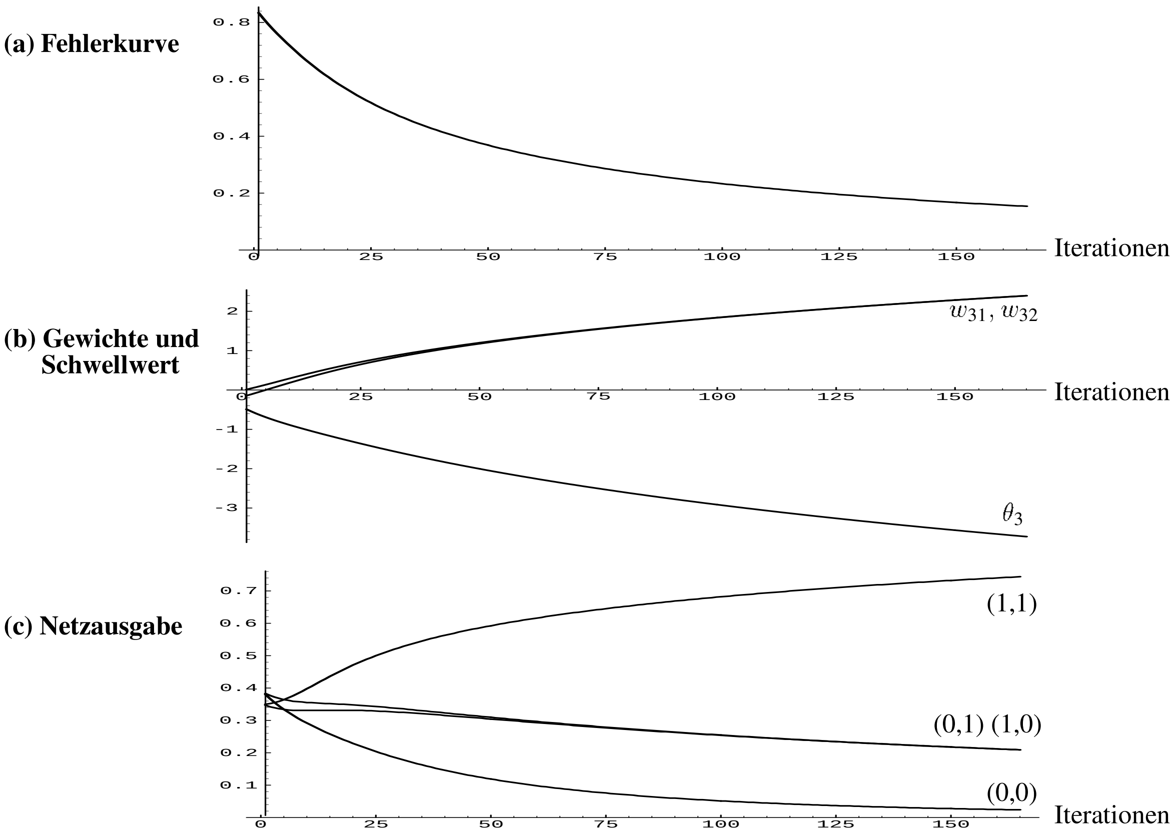


Abbildung 1.8: Training der UND-Funktion. (a) Fehlerkurve. Der Fehler ist streng monoton fallend mit der Anzahl der Trainingsiterationen. (b) Gewichte und Schwellwert. Obwohl die Netzklassifikation konvergiert, divergieren die Gewichte und der Schwellwert. (c) Netzausgabe. Ausgaben > 0.5 werden als Klassifikation 1, und Ausgaben ≤ 0.5 werden als Klassifikation 0 gewertet.

Iteriert man Schritt 2-4, erhält man die in Abbildung 1.8 gezeigten Trainingskurven. Der Fehler sinkt dabei streng monoton, während die Gewichte und der Schwellwert langsam divergieren.

Man beachte, daß der Fehler eine streng monoton fallende Funktion ist, was der Grundidee der Methode des steilsten Abstieges entspricht. Weiterhin zeigt sich, daß Gewichte und Schwellwerte keineswegs konvergieren, da die logistische Funktion die gewünschten Werte 0 und 1 nur asymptotisch annimmt, der Fehler nie exakt 0 erreicht, und die Gewichte in Wirklichkeit gegen $\pm\infty$ divergieren. Wir haben daher die Berechnung nach einer festen Anzahl von Schritten abgebrochen[5]. Das nach 165 Iterationen resultierende Netz liefert dann folgende Ergebnisse:

[5]Ein weiteres beliebtes Abbruchkriterium ist das Sinken des Fehlers unter ein bestimmtes, vorgegebenes Toleranzniveau.

Eingabe x_1	Eingabe x_2	$\longrightarrow$	Ausgabe x_3	Target t_3
0	0		0.024	0
0	1		0.209	0
1	0		0.209	0
1	1		0.744	1

Entscheidet man sich nun im binären Fall für 1, wenn die Ausgabe größer als 0.5 ist, und 0 sonst, dann hat das Netz die UND-Funktion *gelernt*.

1.2.2 Backpropagation

Am Beispiel XOR haben wir festgestellt, daß ein Netz ohne interne Knoten nicht in der Lage ist, beliebige boolsche Funktionen zu lernen. Der Backpropagation-Algorithmus ist die Erweiterung von Gradientenabstieg auf Netze mit internen Knoten. Dabei berechnet man Gradienten nicht nur für die Schwellwerte und Gewichte der Ausgabeschicht, sondern auch für tieferliegende (interne) Schwellwerte und Gewichte.

Wir beginnen die Herleitung von Backpropagation, indem wir ein einzelnes Trainingsbeispiel betrachten, also den Gradienten bezüglich E^p für ein beliebiges $p \in \mathcal{P}$ berechnen. Als Hilfsgradient führt man (für jeden Knoten i) ein:

$$\delta_i \quad \equiv \quad \frac{\partial E^p}{\partial net_i} \tag{1.14}$$

Für die Knoten der Ausgabeschicht haben wir diesen Gradienten bereits in den Gleichungen (1.6) und (1.7) hergeleitet:

$$\boxed{\delta_i \quad = \frac{\partial E^p}{\partial x_i} \cdot \frac{\partial x_i}{\partial net_i} \quad = -(d_i - x_i) \cdot x_i \cdot (1 - x_i)} \tag{1.15}$$

Für alle internen Knoten ergibt sich der Gradient aus den Gradienten der <u>Nachfolger</u> im Netz, definiert als $S(i)$:

$$\boxed{\delta_i \quad = \sum_{j \in S(i)} \frac{\partial E^p}{\partial net_j} \cdot \frac{\partial net_j}{\partial x_i} \cdot \frac{\partial x_i}{\partial net_i} \quad = \left(\sum_{j \in S(i)} \delta_j \cdot w_{ji} \right) x_i \cdot (1 - x_i)} \tag{1.16}$$

Aus (1.1) und (1.16) wird ersichtlich, daß diese Hilfsgradienten entgegen der Propagationsrichtung für Aktivationen durch das Netz propagiert werden. Diese Beobachtung gibt dem Backpropagation-Algorithmus (zu deutsch: „Rückwärtspropagation“) seinen Namen.

Mit der Kettenregel lassen sich nun leicht die Gradienten für alle Gewichte w_{ij} und Schwellwerte θ_i herleiten:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{p=1}^{P} \frac{\partial E^p}{\partial net_i} \cdot \frac{\partial net_i}{\partial w_{ij}} = \sum_{p=1}^{P} \delta_i \cdot x_j \tag{1.17}$$

$$\frac{\partial E}{\partial \theta_i} = \sum_{p=1}^{P} \frac{\partial E^p}{\partial net_i} \cdot \frac{\partial net_i}{\partial \theta_i} = -\delta_i \tag{1.18}$$

Die Verallgemeinerte Deltaregel (1.15), (1.16) zusammen mit (1.17) und (1.18) und den folgenden Adaptionsregeln (vgl. (1.12) und (1.13))

$$w_{ij} \longleftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} \tag{1.19}$$

$$\theta_i \longleftarrow \theta_i - \eta \frac{\partial E}{\theta_i} \tag{1.20}$$

bezeichnet man als Backpropagation.

1.2.3 Der Momentum-Term

An dieser Stelle werden wir den Lernalgorithmus um eine wichtige Heuristik erweitern, die in der Regel die Konvergenz des Fehlers erheblich beschleunigt und deshalb inzwischen fest mit dem Begriff Backpropagation verknüpft ist: das Momentum.

Das Momentum läßt sich gut mithilfe der Abbildung 1.7 motivieren. Interpretiert man die Energiefunktion wie in Abbildung 1.7, stellt man fest, daß die dort gezeigte Kugel gewisse Eigenschaften einer echten Kugel besitzt, da ihre Geschwindigkeit von der Steilheit der Funktion abhängt. Eine weitere Eigenschaft einer Kugel, nämlich die massenbedingte *Trägheit*, ist aber noch nicht berücksichtigt. Trägheit bedeutet, daß die Geschwindigkeit der Kugel sich nur langsam ändert und somit eine Funktion ihrer selbst ist. Nun scheinen die meisten Fehlerkurven von Backpropagation-Netzen so beschaffen zu sein, daß eine gewisse Trägheit bei der Änderung der Parameter die Suche

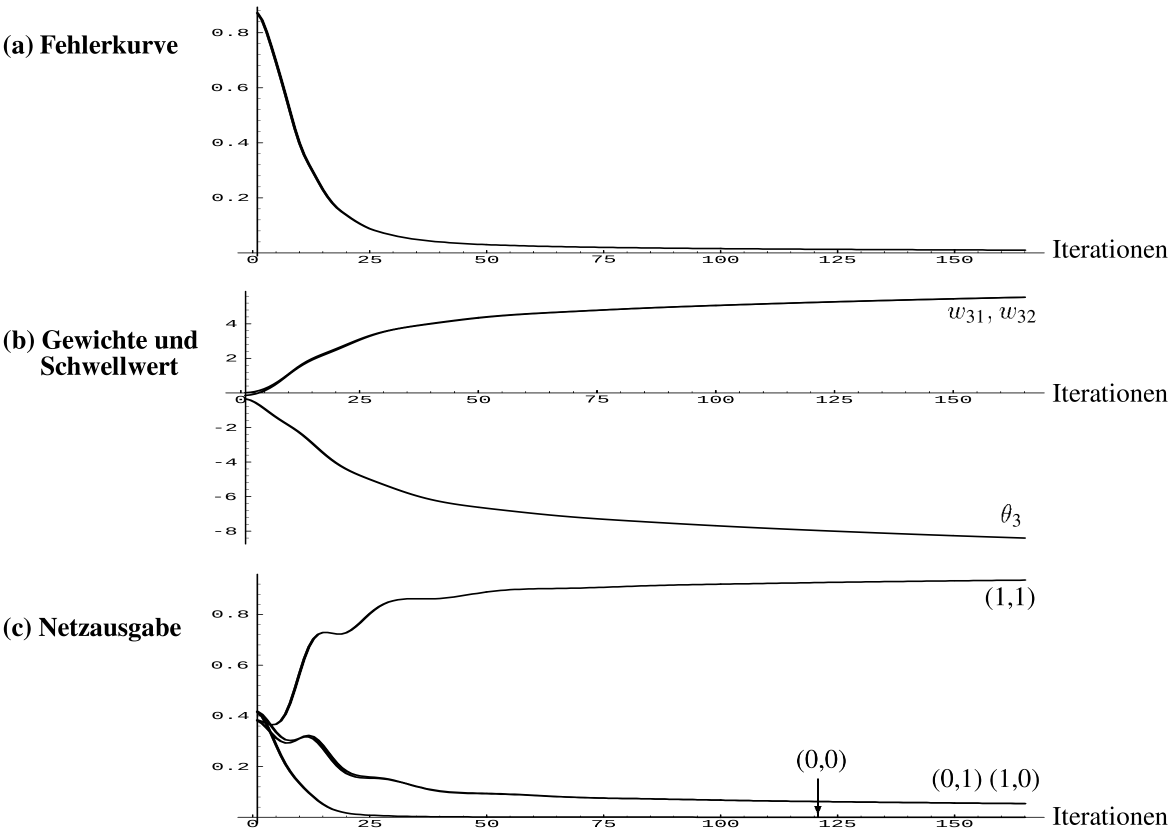


Abbildung 1.9: Training der UND-Funktion mit Momentum $\mu = 0.9$. Bereits nach etwa 10 Schritten ist die UND-Funktion gelernt, da alle Muster richtig klassifiziert werden.

beschleunigt. Dabei interpretieren wir *Trägheit* so, daß die Änderung der Parameter im iterativen Gradientenabstiegsverfahren nicht allein von der momentanen Steilheit der Fehlerkurve, sondern auch von der *Änderung im letzten Schritt* (falls ein solcher existiert) abhängt. Dies wird erreicht durch:

$$\begin{aligned}
\Delta w_{ij} &\longleftarrow \frac{\partial E}{\partial w_{ij}} + \mu \cdot \Delta w_{ij} && (1.21)\\
\Delta \theta_i &\longleftarrow \frac{\partial E}{\partial \theta_i} + \mu \cdot \Delta \theta_i && (1.22)\\
w_{ij} &\longleftarrow -\eta \cdot \Delta w_{ij} && (1.23)\\
\theta_i &\longleftarrow -\eta \cdot \Delta \theta_i && (1.24)
\end{aligned}$$

Dabei setzt sich die Parameteränderung Δw_{ij} bzw. $\Delta\theta_i$ aus zwei Summanden zusammen: dem aktuellen Gradienten plus der letzten Änderung multipliziert mit der Momentum-Rate μ. (vgl.(1.19) und (1.20)). μ sollte dabei im Bereich $[0, 1]$ liegen: Mit $\mu = 0$ sind diese Gleichungen identisch mit (1.19) und (1.20).

Wendet man ein Momentum $\mu = 0.9$ auf das UND-Beispiel in Kapitel 1.2.1 an, dann ergibt sich das in Abbildung 1.9 gezeigte Bild. Nach nur 19 Iterationen mit $\mu = 0.9$ ist bereits der gleiche Lernerfolg erzielt wie nach 165 Schritten ohne Momentum (mit Momentum $\mu = 0.0$). Nichtsdestotrotz ist das Momentum eine Heuristik, die von bestimmten Eigenschaften der Fehlerfunktion ausgeht. Wir haben das Momentum wegen seiner starken Bedeutung, die sich in vielen Anwendungen widerspiegelt, hier aufgenommen – andere Heuristiken werden wir in Kapitel 2 diskutieren.

1.2.4 Lernen der XOR-Funktion mit Backpropagation

Wir wollen uns nun dem letzten Zahlenbeispiel dieses Kapitels zuwenden, der XOR-Funktion. Diese Funktion ist deutlich schwerer zu erlernen als das UND, was darauf zurückzuführen ist, daß die beiden internen Knoten erst eine geeignete Funktion zur Verfügung stellen müssen, bevor der Ausgabeknoten die gewünschte Funktion XOR realisieren kann. Man spricht in diesem Kontext oft davon, daß sich eine *interne Repräsentation entwickelt*.

Bei Verwendung der Lernrate $\eta = 0.5$ und dem Momentum $\mu = 0.9$ ergibt sich der in Abbildung 1.10 gezeigte Trainingsverlauf. Auffallend ist, daß für etwa 180-200 Schritte die Änderungen minimal bleiben, sowohl die der Gewichte und Schwellwerte, als auch die des Fehlers. Das Netz liefert in dieser Phase auf alle vier Muster als Ausgabe in etwa den Durchschnittswert 0.5, und die Gewichte bleiben relativ klein. Dies zeigt an, daß noch keine geeignete Repräsentation in der internen Schicht gefunden ist. Erst ab etwa dem 200. Schritt sinkt der Fehler deutlich und schnell. Die beiden internen Knoten agieren nun als UND und ODER, und wie in Kapitel 1.1.2 explizit gezeigt, ist nun eine Lösung des Lernproblems möglich. Die Zweiphasigkeit dieses Experimentes kennzeichnet eine besondere Problematik: Selbst wenn der Fehler gleichmäßig auf einem schlechten Niveau bleibt, liegt nicht unbedingt ein lokales Minimum vor.

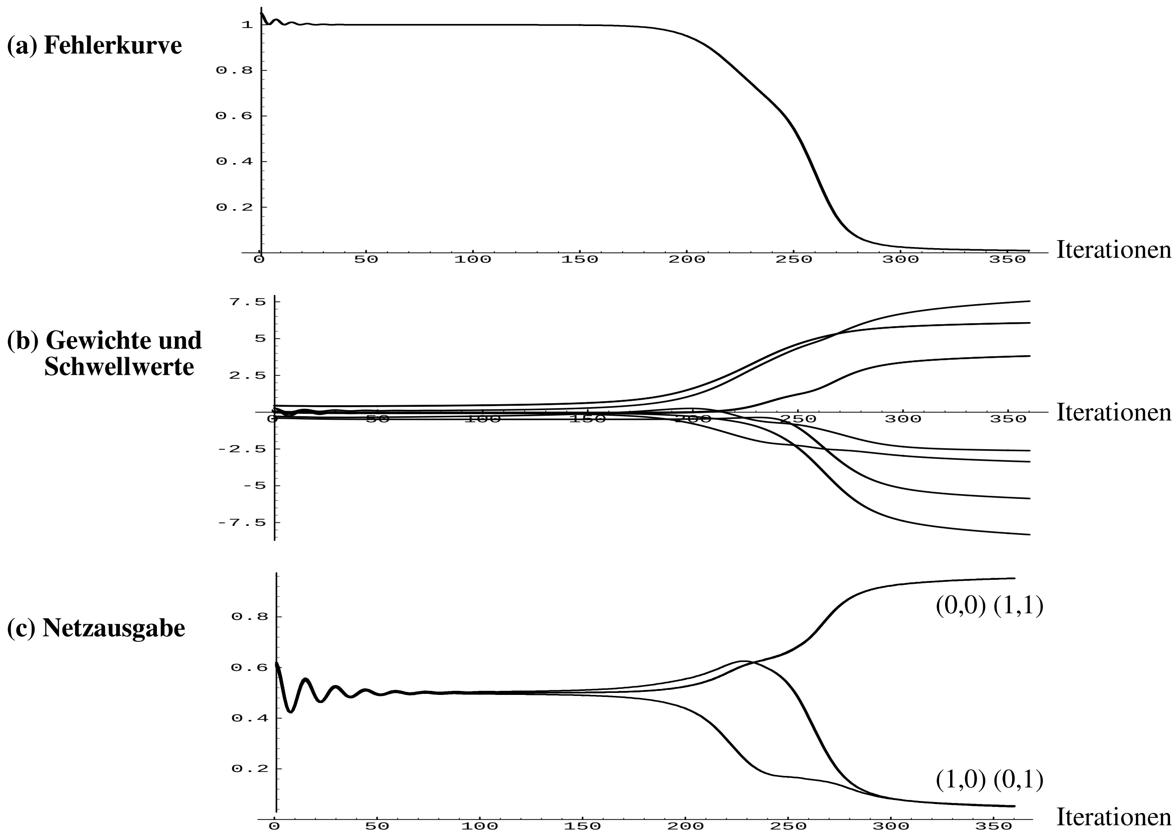


Abbildung 1.10: Training der XOR-Funktion mit dem Backpropagationalgorithmus.

Damit sei das einleitende XOR-Beispiel abgeschlossen.

1.3 Anwendungsgebiete Künstlicher Neuronaler Netze

Wir haben nun eine Reihe einfacher Netze vorgestellt, die boolsche Funktionen realisierten. Dabei haben wir die Gewichte und Schwellwerte erst manuell gesucht, später durch den Backpropagation-Algorithmus „lernen“ lassen.

Natürlich ist die Klasse der Funktionen, die durch Backpropagation approximiert werden können, nicht auf boolsche Funktionen beschränkt. Vielmehr realisiert ein Künstliches Neuronales Netz eine differenzierbare und somit stetige Funktion $f : I\!R^m \longrightarrow (0,1)^n$, die sich auch für allgemeine Approximationsprobleme eignet. Dabei findet man je nach Vor- und Nachbereich die in Tabelle 1.2 gezeigten Anwendungsgebiete Künstlicher Neuronaler Netze:

Eine im Bereich *Maschinelles Lernen* gefragte Eigenschaft Künstlicher Neuronaler Netze ist *Ge-*

	Nachbereich diskret	Nachbereich kontinuierlich
Vorbereich diskret	boolsche Funktionen, diskrete Klassifikation, symbolische KI	Mustergenerierung, Trajektoriengenerierung
Vorbereich kontinuierlich	Musterklassifikation, Trajektorienklassifikation	Nichtlineare Approximation, Modellierung stetiger Welten, Adaptive Controller

Tabelle 1.2: Anwendungsdomänen für Künstliche Neuronale Netze.

neralisierung. Unter Generalisierung – das Pendant der Künstlichen Intelligenz (KI) zum Begriff „Approximation“ oder „Interpolation“ in der Praktischen Mathematik und „Regression“ in der Statistik – versteht man die Fortsetzung von endlich vielen Trainingsbeispielen zu einer Funktion. Backpropagationnetze erzeugen typischerweise sehr glatte Generalisierungen. Wir werden im weiteren Verlauf dieser Arbeit, insbesondere in Kapitel 3 bis 5, Künstliche Neuronale Netze als allgemeine nichtlineare, generalisierende Approximatoren verwenden.

1.4 Literatur

Die Geschichte von Künstlichen Neuronalen Netzen läßt sich bis in der Vierziger Jahre zurückverfolgen, als McCulloch und Pitts zum ersten Male ein formales Neuron definierten [McCulloch und Pitts, 1943]. In den Fünfzigern und Sechzigern gab es dann einige wenige Pionierarbeiten zum Thema Künstliche Neuronale Netze, von denen die berühmtesten wohl die Promotion von Marvin Minsky [Minsky, 1954] (siehe auch [Minsky und Papert, 1988]), Oliver Selfridge [Selfridge, 1958], Bernard Widrow [Widrow und Hoff, 1960], und Rosenblatt [Rosenblatt, 1962] sind. Nahezu alle historisch bedeutenden Veröffentlichungen finden sich in einer gelungen zusammengestellten Kollektion von Anderson [Anderson, 1988].

Nachdem in den Siebzigern weltweit die Forschung im Bereich Künstlicher Neuronaler Netze fast völlig stagnierte, erfreut sie sich seit den frühen Achtzigern eines neuen und gewaltigen Booms. 1986 brachten Dave Rumelhart, Jay McClellend, Jeff Hinton und andere maßgebliche Forscher auf dem Gebiet Neuronaler Netze die berühmten PDP-Bände (*parallel distributed processing*) heraus [Rumelhart *et al.*, 1986], [McClelland und Rumelhart, 1988], in denen Backpropagation zum ersten Male unter diesem Namen publiziert wurde. Diese Bücher sind ein Muß für den Forscher im Bereich Neuronaler Netze: Auf begeisternde Weise werden psychologische und kognitive Beobachtungen und einfache Netzmodelle miteinander in Beziehung gebracht. Die PDP-Bücher werden in Insiderkreisen bisweilen als „Bibel“ bezeichnet, was den nicht ganz ungefährlichen Charakter dieses Schriftguts unterstreicht. Seit Erscheinen dieser Bücher, aber nicht allein dadurch ausgelöst, erfreuen sich Künstliche Neuronale Netze einer breiten Beliebtheit, ja eines ganzen Forschungsbooms.

In den darauffolgenden drei Jahren sind einige gute Übersichten entstanden, die noch heute ihre Gültigkeit haben, und deren Lektüre wir empfehlen: [Lippmann, 1987], [Kohonen, 1988], [Knight, 1989], [Hinton, 1989]. Seit Ende der Achtziger ist darüber hinaus eine Flut von mehr oder weniger guten Büchern auf dem Markt erschienen. Eines der bedeutendsten Bücher, das auch an der Carnegie Mellon University, einer der Geburtsstätten Neuronaler Netze, als Kursunterlagen dient, ist Hertz/Krogh/Palmers Buch [Hertz *et al.*, 1991].

Kapitel 2

Gradientenabstieg in Künstlichen Neuronalen Netzen

Nachdem wir nun für nicht-rekurrente Künstliche Neuronale Netze eine Lernregel hergeleitet und ihre Arbeitsweise demonstriert haben, werden wir im folgenden Künstliche Neuronale Netze und Gradientenabstieg erweitern und ein wenig formaler betrachten. Dazu beginnen wir mit der Definition eines Künstlichen Neuronalen Netzes als Graph. Wir haben bereits gesehen, daß man Künstliche Neuronale Netze als gerichtete Graphen interpretieren kann, entlang deren Kanten Aktivationswerte propagiert werden. Bei obigen Beispielen hing der Aktivationszustand eines Künstlichen Neuronalen Netzes nur von der gerade angelegten Eingabe ab – legt man nacheinander mehrere solche an, beeinflusst natürlich keine zu einem früheren Zeitpunkt evaluierte Eingabe die laufende Berechnung.

Nun werden im allgemeinen Künstliche Neuronale Netze auch als zeitliche Systeme benutzt, bei denen zeitlich vorangegangene Eingaben die laufenden Aktivationen beeinflussen. Realisiert wird dies durch zwei fundamentale Erweiterungen:

- Kanten werden mit einem *zeitlichen Delay* versehen. Legt man zum Beispiel zu einem bestimmten Zeitpunkt t einen Wert an, verzögert eine Delay-Kante die Propagation dieses Wertes um n Zeiteinheiten – n ist dabei das ihr zugeordnete Delay. Delays werden gerne benutzt, um ein zeitliches Fenster auf einer Folge von Eingabemustern zu realisieren – dabei sind Eingabeknoten mit mehreren unterschiedlichen zeitlichen Delays versehen. Wird die feed-forward-Struktur (Zyklenfreiheit) des Netzes nicht verletzt, können allerdings keine Informationen jenseits des Zeitfensters die Berechnung beeinflussen. Dies wird möglich durch die Einführung von
- *Rekurrenzen*: In Verbindung mit Delay-Kanten werden Zyklen im Graph zugelassen, bei denen Aktivationen, die das Netz zu einem früheren Zeitpunkt berechnet hat, in das Netz so wieder eingespeist werden, daß ein Zyklus entsteht. In diesem Zyklus besteht dann die prinzipielle Möglichkeit, daß Informationen unbeschränkt lange erhalten bleibt. Oft ist

davon die Rede, daß die an einem Zyklus beteiligten Knoten eine *interne Repräsentation* für die Darstellung von Information entwickeln, nämlich für genau die Information, die für das erfolgreiche Lernen von zeitlichen Sequenzen gespeichert werden muß (vorausgesetzt natürlich, das Lernen konvergiert erfolgreich).

Die einleitende Definiton eines Netzes läßt die geforderte Erweiterung auf zeitliche Strukturen nicht zu, da nur einfache *zeitlose* Multilayerperceptrons betrachtet wurden. Daher beginnen wir erneut mit der formalen Definition allgemeiner Künstlicher Neuronaler Netze, die eine Art Oberbegriff des bisherigen Netzbegriffes darstellt. Dieses Kapitel ist folgendermaßen gegliedert:

Nach der graphentheoretischen Definition des Künstlichen Neuronalen Netzes und der formalen Einführung von Aktivationen, Gewichten und Schwellwerten werden die Aktivationsfunktion und zwei Squashingfunktionen – die bereits vorgestellten *logistischen* Funktionen, und die *Radial-Basis* Funktionen – beschrieben. Im folgenden werden dann zwei Algorithmen für Gradientenabstieg im Gewichtsraum hergeleitet: BPTT (*Backpropagation through time*) und BPFF (*feed-forward Backpropagation*). Da beide Algorithmen Gradientenabstieg realisieren, ist ihr Ergebnis identisch, doch impliziert die unterschiedliche Art der Berechnung unterschiedliche Eigenschaften und Komplexitäten:

- Der erste Algorithmus, BPTT (*Backpropagation through time*), erweitert Backpropagation in der oben eingeführten Form auf rekurrente Netze. Dabei ist die Beobachtung von zentraler Bedeutung, daß jedes rekurrente Netz in ein nichtrekurrentes Netz überführt werden kann, indem man die Zeit als eigene Achse einführt. Diese Technik, *Unfolding-in-Time* genannt, macht sich die einfache Tatsache zunutze, daß entlang der Zeit keine Rekurrenzen auftreten können: keine Aktivation wird durch eine zeitlich spätere beeinflusst. Dies wissend kann man nun für jedes rekurrentes ein äquivalentes nichtrekurrentes Netz finden, indem man für jeden Zeitschritt eine Kopie des ursprünglichen Netzes anlegt. Das Nichtvorhandensein von zeitlichen Rekurrenzen impliziert dann, daß dieses neue Netz völlig frei von Rekurrenzen, also im Prinzip ein einfaches Multilayerperceptron ist. Unter der Berücksichtigung, daß Kanten durch das Kopieren von Netzen nun mehrfach vorhanden sind (jede Kopie muß das gleiche Gewicht haben, da im äquivalenten Ursprungsnetz nur ein Gewicht dafür existiert), ist es ein Leichtes, den Backpropagation-Algorithmus auf diese Multilayerperceptrons zu erweitern.

  Natürlich behält der Lernalgorithmus BPTT die Zwei-Phasen-Eigenschaft von Backpropagation bei. In der Vorwärtsphase werden alle Aktivationen durch das aufgefaltete Netz propagiert, und in der darauf folgenden Rückwärtsphase die Gradienten. Dies bedingt die unangenehme Eigenschaft, daß alle Aktivationen während der Vorwärtsphase gespeichert werden müssen, da sie später zur Rückpropagation der Gradienten benötigt werden.

- Der zweite Algorithmus, BPFF (*feed-forward Backpropagation*), unterscheidet sich von BPTT, indem er eine Rückwärtsphase überflüssig macht. Durch ein geändertes Berechnen von Gradienten wird es möglich, alle zum Gradientenabstieg erforderlichen Werte vorwärts in der Zeit zusammen mit den Aktivationen zu propagieren. Dies macht den Algorithmus universeller, da nun nicht die ganze zeitliche Vergangenheit des Netzes zwischengespeichert werden muß, erhöht aber auf der anderen Seite seine Komplexität[1].

Nach der Diskussion der Heuristiken *Teacher Forcing* und *Online Learning* werden wir die am häufigsten verwendeten architektonischen Grundtypen Künstlicher Neuronaler Netze vorstellen. Im darauf folgenden Teil werden zwei weitere Algorithmen für Gradientenabstieg hergeleitet, die statt der Gewichte und Schwellwerte andere, externe Größen zu optimieren suchen. Externe Größen sind dabei Initialaktivationen von Knoten in rekurrenten Netzen und Eingabewerte. Die Gleichungen und Gradienten zur Suche in diesen externen Räumen ähnelt dabei sehr den Lernregeln der beiden Backpropagationalgorithmen. Diese Techniken finden insbesondere im zweiten Teil dieser Arbeit ihre Anwendung.

Den letzten Teil bildet eine Übersicht über die zugrundeliegende und weiterführende Literatur.

2.1 Formale Definition eines Künstlichen Neuronalen Netzes

2.1.1 Der Graph

Wie wir im einführenden Kapitel gesehen haben, kann man ein Künstliches Neuronales Netz als einen gerichteten Graphen interpretieren, dessen gewichtete Kanten Knoten miteinander verbinden. Will man zwischen zwei beliebigen Knoten auch Kanten mit Delay zulassen, muß die Notation aus Kapitel 1 erweitert werden, da nun zwei Knoten mit mehreren Kanten unterschiedlichen Delays verbunden sein können.

Definition (Künstliches Neuronales Netz):

Ein <u>Künstliches Neuronales Netz</u> (im folgenden oft: <u>Netz</u>) ist ein *gerichteter Graph* $\mathcal{G} \equiv (\mathcal{V}, \mathcal{E})$ mit <u>Knotenmenge</u> $\mathcal{V}$ und <u>Kantenmenge</u> $\mathcal{E}$. Die Knotenmenge ist endlich: $\mathcal{V} \equiv \{v_1, v_2, ..., v_N\}$ und $\mathcal{N} \equiv \{1, 2, ..., N\}$ ist die <u>Indexmenge der Knotenmenge</u> $\mathcal{V}$. Die <u>Kantenmenge</u> $\mathcal{E}$ ist Untermenge von $\{\langle v_i, v_j\rangle_n |\ v_i, v_j \in \mathcal{V} \wedge n \in \mathcal{S}\}$ für ein $\mathcal{S} \subset \{0, 1, ..., \hat{n}\}$. $\mathcal{S}$ ist die <u>Menge der zeitlichen Delays</u> des betrachteten Netzes und $\hat{n} \equiv \max \mathcal{S}$ das <u>größte Delay</u>, eine netzspezifische Konstante.

Jede Kante ist also formal ein Tripel $\langle v_i, v_j\rangle_n$, das Knoten v_j mit Knoten v_i mit einem Delay von n verbindet. Das größte Delay $\hat{n}$ legt dann fest, daß bei der Vorwärtspropagation die Aktivationsvergangenheit von $\hat{n}$ Zeiteinheiten jeweils zwischengespeichert werden muß.

[1] zumindest in der hier vorgestellten Version – siehe die Literaturübersicht.

Weiter sei definiert:

Definition:

Die Untermenge $\mathcal{I} \in \mathcal{N}$ von Knotenindizes zählt die Menge der Eingabeknoten $\{v_k | \; k \in \mathcal{I}\}$ auf. Entsprechend zählt die Untermenge $\mathcal{O} \in \mathcal{N}$ von Knotenindizes die Menge der Ausgabeknoten $\{v_k | \; k \in \mathcal{O}\}$ auf.

Um die Funktionalität des Netzes zu beschreiben, führen wir im folgenden *Aktivationen* und *Gewichte* ein. Dabei wird der *Schwellwert* im Gewichtsbegriff subsumiert.

2.1.2 Aktivationen, Gewichte und Schwellwerte

Jedem Knoten $v_i \in \mathcal{V}$ ist zu jedem Zeitpunkt t eine reellwertige Aktivation $x_i(t)$ zugeordnet.

Im Falle eines Multilayerperceptrons, wie zum Beispiel in den einleitenden Beispielen, macht der Zeitindex t wenig Sinn. Im allgemeinen Fall dagegen, in dem ganze Eingabesequenzen anstelle von singulären Eingabemustern verarbeitet werden, ist die Aktivation auch eine Funktion der Zeit. Dabei gilt, daß t sich in einem Arbeitsintervall $\{t_0, t_0 + 1, ..., t_1\}$ bewegt: $t_0 \in I\!R$ ist der Anfangszeitpunkt und $t_1 \in \overline{I\!R} = I\!R \cup \{\pm\infty\}, t_1 \geq t_0$, der Endzeitpunkt der Berechnung.

Analog ist jeder Kante $\langle v_i, v_j \rangle_n \in \mathcal{E}$ ein Gewicht $w_{ijn} \in I\!R$ zugeordnet, das, bis auf später explizit diskutierte Ausnahmen, im Intervall $\{t_0, t_0 + 1, ..., t_1\}$ konstant ist.

Dabei gilt:

- Gewöhnlich ist die Menge der Kanten $\mathcal{E}$ eine echte Untermenge der Menge aller möglichen Kanten $\{\langle v_i, v_j \rangle_n | \; v_i, v_j \in \mathcal{V} \wedge n \in \mathcal{S}\}$. $\mathcal{E}$ bestimmt die Topologie des Netzes. Struktur, in Form von partieller Konnektivität, kann, wie vielfach experimentell in der Literatur gezeigt, von erheblichem Vorteil für das Konvergenzverhalten und die Generalisierung sein. Im Laufe dieses Kapitel werden wir eine Reihe in der Literatur beliebter Netzarchitekturen vorstellen.

- Um die Mathematik weiter zu vereinfachen, werden die Schwellwerte θ_i in den Gewichtstensor subsumiert. Dies geschieht durch die Auszeichnung eines speziellen Knotens $v_* \in \mathcal{V}$, der immer die maximale Aktivation (fester Wert, zum Beispiel 1.0) hat und Vorgänger aller anderen Knoten $v_i \in \mathcal{V} \backslash \{v_*\}$ ist. Dann sind die Gewichte $\langle v_i, v_* \rangle_0$ identisch mit dem negativem Schwellwert $-\theta_i$, wie man leicht aus 1.1 sieht. Im folgenden lassen wir daher den Schwellwert einfach weg.

2.1.3 Aktivationsfunktionen

Die Aktivationsfunktion σ bestimmt, wie Aktivationen berechnet, sprich durch das Netz propagiert werden. In den einleitenden Beispielen hatten die Eingabeknoten die *identische Funktion* als Aktivationsfunktion, weshalb Eingabeknoten oft nicht als richtige Knoten betrachtet werden. Alle anderen Funktionen hatten als Aktivationsfunktion die Funktion:

$$x_i \equiv \sigma\left(\sum_{j\in\mathcal{N}} w_{ij} - \theta_j\right) \quad \text{mit} \quad \sigma(net_i) \equiv \frac{1}{1+e^{-net_i}} \tag{2.1}$$

Obwohl in der Literatur unterschiedlichste stetige Aktivationsfunktionen diskutiert werden, beschränken wir uns hier auf solche, deren Eingabe *summarischen* Charakter haben und somit Aktivationsfunktionen des folgenden Typs sind:

$$x_i(t) \;\equiv\; \sigma_i\left(\sum_{j\in\mathcal{N}} \sum_{n=0}^{\min\{t-t_0,\hat{n}\}} w_{ijn} x_j(t-n)\right) + I_i(t) \quad \forall i \in \mathcal{N} \tag{2.2}$$

wobei

- σ_i ist die Squashing-Funktion, eine differenzierbare Funktion, die hauptsächlich dem Beschränken von Aktivationen dient und in Kapitel 2.1.4 weiter untersucht wird.
- $I_i(t)$ ist die externe Eingabe, die dem Netz zum Zeitpunkt t über den Knoten v_i übergeben wird.
- Abkürzend definiert man die netzinterne Eingabe eines Knotens als
$$net_i(t) \equiv \sum_{j\in\mathcal{N}} \sum_{n=0}^{\min\{t-t_0,\hat{n}\}} w_{ijn} x_j(t-n) \tag{2.3}$$
- Die initialen Aktivationen $\tilde{x}_i$ sind ebenfalls *externe* Größen, die die Berechnung beeinflussen. Sie initialisieren die Aktivationen des Netzes:
$$x_i(t_0) \equiv \tilde{x}_i \tag{2.4}$$

2.1.4 Squashing-Funktionen

In der Aktivationsfunktion blieb die Squashing-Funktion bisher unspezifiziert. Wesentliche Forderung für Gradientenabstieg ist üblicherweise, daß sie (zumindest stückweise) *differenzierbar* ist. Zwei weitere oft verlangte Eigenschaften sind *Beschränktheit* und *Monotonie*.

Wir werden im folgenden die zwei in der Literatur meistverwendeten Familien von Squashing-Funktionen vorstellen: die logistische Funktionen und die Radial-Basis Funktionen. Beide Funktionsklassen sind differenzierbar und beschränkt. Dabei bezeichne x^- das Infimum und das x^+ Supremum der Aktivation.

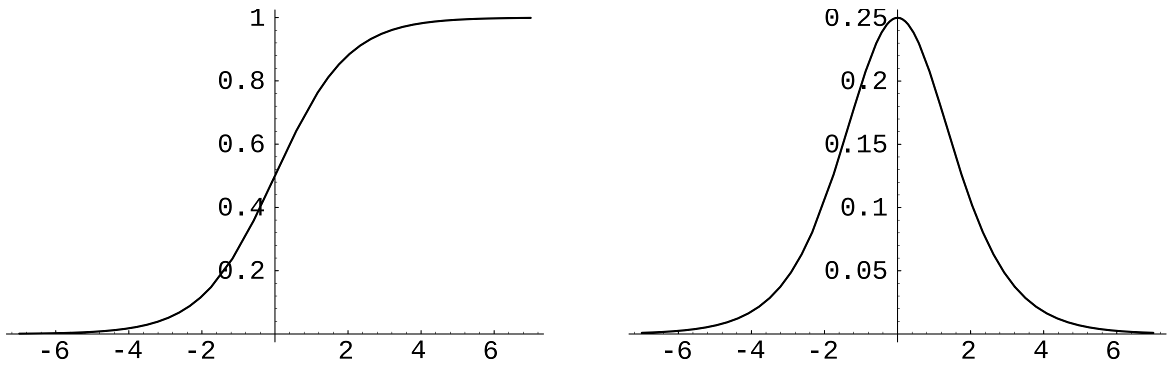


Abbildung 2.1: Eine logistische Funktion ($a = 0$, $b = c = 1$) und ihre Ableitung.

Logistische Funktionen

Logistische Squashing-Funktionen haben wir schon in den einführenden Beispielen kennengelernt. Sie erfüllen alle oben genannten Forderungen und stellen eine Art differenzierbare Mixtur aus linearen Funktionen (Betrieb im mittleren Bereich) und binären Schwellwertfunktionen (Betrieb im Sättigungsbereich) dar (Abbildung 2.1). Mit ihrer Hilfe werden Aktivationen monoton beschränkt:

$$\sigma_i : I\!R \longrightarrow (x^-, x^+) \quad \text{mit} \quad \sigma_i(net_i) \equiv \frac{b}{1 + e^{-c \cdot net_i}} - a \tag{2.5}$$

Dabei definieren die reellwertigen Skalare $a, b, c \in I\!R$, $b, c \neq 0$ den Symmetriepunkt, den Wertebereich und die maximale Steilheit der Funktion:

- Der Symmetriepunkt ist $P_{sym} = (0|\frac{b}{2} - a)$,
- der Wertebereich (x^-, x^+) ist $(\min\{-a, b - a\}, \max\{-a, b - a\})$ und
- der steilste Punkt ist der Symmetriepunkt mit der Steigung $\frac{1}{4}bc$.

Die Ableitung der Squashing-Funktion nach net_i ist ein entscheidender Bestandteil der Gradientenberechnung. Sie ist gegeben durch (vergleiche (1.4) und (1.7)):

$$\begin{aligned}\sigma_i'(net_i) &= \frac{\partial \sigma_i(net_i)}{\partial net_i} = bc\,\frac{1}{1 + e^{-c \cdot net_i}} \left(1 - \frac{1}{1 + e^{-c \cdot net_i}}\right) \\ &= \frac{c}{b}\,(\sigma_i(net_i) + a)\,(b - a - \sigma_i(net_i)) \end{aligned} \tag{2.6}$$

Im folgenden werden wir in nahezu allen Beispielen, wie bereits bei den einleitenden, die Parameter $a = 0$, $b = 1$ und $c = 1$ verwenden; dies ergibt für die logistische Squashing-Funktion:

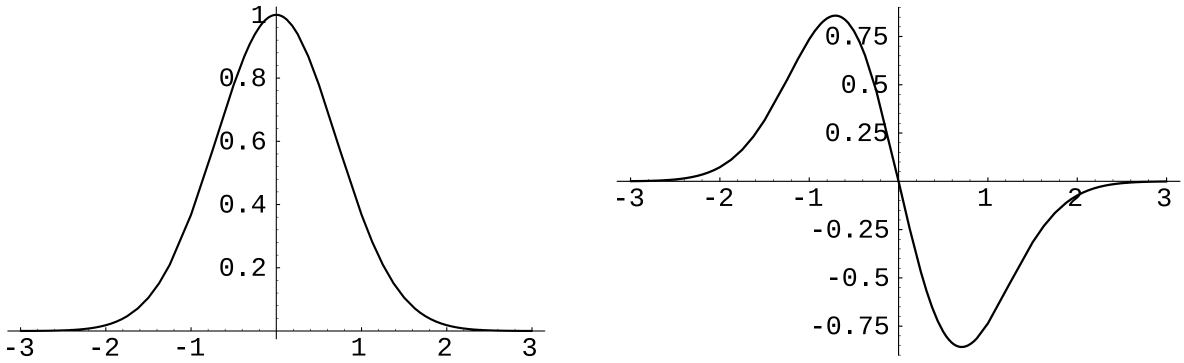


Abbildung 2.2: Eine Radial-Basis Funktion, die Gaussche Glockenkurve ($a = 0$, $b = c = 1$), und ihre Ableitung.

$$\sigma_i(net_i) = \frac{1}{1 + e^{-net_i}} \tag{2.7}$$

$$\sigma_i'(net_i) = \sigma_i(net_i) \cdot (1 - \sigma_i(net_i)) \tag{2.8}$$

Zwar beeinflusst die Wahl der Parameter a, b, c die Mächtigkeit der Netze nicht, da es zu jedem Netz mit Parametern a, b, c ein gleichgroßes Netz mit Parametern a', b', c' gibt, das die selbe mathematische Funktion realisiert. Das gilt allerdings nicht für Gradientenabstieg. Verschiedene Autoren berichten empirisch, daß die Wahl dieser Parameter die Lerngeschwindigkeit deutlich beeinflusst. Zum Beispiel scheint die ursprungssymmetrische Squashing-Funktion mit $a = 1$, $b = 2$ und $c = 1$ in vielen Anwendungen von Vorteil zu sein.

Radial-Basis Funktionen

Eine weitere in der Literatur nicht ganz so oft verwendete Squashing-Funktion ist die Familie der Radial-Basis Funktionen. Die Gaussche Glockenkurve, gezeigt in Abbildung 2.2, ist ein bekannter Vertreter dieser Klasse. Diese Funktion ähnelt der logistischen Funktion, doch läuft sie in beide Richtungen asymptotisch gegen 0 ($x \longrightarrow \pm\infty$).

$$\sigma_i : I\!R \longrightarrow (x^-, x^+]$$

mit

$$\sigma_i(net_i) \equiv b \cdot e^{-(cx)^2} - a \tag{2.9}$$

Wie die logistischen Funktionen sind die Radial-Basis Funktionen beschränkt und überall differenzierbar. Die Skalare $a, b, c \in I\!R$, $c \neq 0$ legen den Wertebereich und die Steilheit der Funktion fest:

- der Wertebereich $(x^-, x^+]$ ist $(\min\{-a, b-a\}, \max\{-a, b-a\}]$,
- die steilsten Punkte sind $(-(\sqrt{2}c)^{-1}|b \cdot e^{-0.5} - a)$ und $((\sqrt{2}c)^{-}1|b \cdot e^{-0.5} - a)$ mit den Steigungen $b \cdot e^{-0.5}$ und $-b \cdot e^{-0.5}$, und
- die Symmetrieachse ist die y-Achse ($x = 0$).

Leitet man die Radial-Basis Funktionen ab, ergibt sich:

$$\sigma_i'(net_i) \quad = \quad \frac{\partial \sigma_i(net_i)}{\partial net_i} \quad = \quad -2bc \cdot e^{-(cx)^2} \tag{2.10}$$

Obige Bemerkungen für logistische Squashing-Funktionen lassen sich direkt auf Radial-Basis Funktionen übertragen. Typische Parameterwerte sind: $a = 0$ und $b = c = 1$.

2.1.5 Restriktionen für die Netzarchitektur

Bisher haben wir beliebige Graphen für Künstliche Neuronale Netze zugelassen. Da entlang der Kanten aber Aktivationswerte propagiert werden, muß mit der Einführung von Aktivationsfunktionen die Konnektivität von Netzen eingeschränkt werden. Diese Einschränkung beruht auf der Beobachtung, daß in Künstlichen Neuronalen Netzen Aktivationen zum Zeitpunkt t Aktivationen zum selben Zeitpunkt beeinflussen können, und jeder Knoten seine Aktivation nur berechnen kann, wenn all seine Vorgänger im Graphen dies bereits getan haben:

Bedingung $(*)$:

Es gibt keinen Zyklus $\{\langle v_{i_2}, v_{i_1}\rangle_0, \langle v_{i_3}, v_{i_2}\rangle_0, ..., \langle v_{i_c}, v_{i_{c-1}}\rangle_0, \langle v_{i_1}, v_{i_c}\rangle_0\} \neq \emptyset$ in $\mathcal{G}$.

Generell ist diese Bedingung von Bedeutung, wenn die Aktivationen eines Knotens zum Zeitpunkt t die Aktivationen anderer Knoten zum selben Zeitpunkt benötigt. Ohne $(*)$ besteht die Gefahr eines *Deadlocks*: Jeder Knoten einer Untermenge $\mathcal{V}' \subseteq \mathcal{V}$ kann seine eigene Aktivation erst berechnen, wenn mindestens ein anderer Knoten in $\mathcal{V}'$ seine Aktivation bereits berechnet hat.

Wie wir im folgenden zeigen werden, kann Bedingung $(*)$ in die Bedingung überführt werden, daß die Gewichtsmatrix der Gewichte mit dem Delay 0 in eine obere Dreiecksmatrix überführt werden kann:

Theorem:

Die Bedingung $(*)$ ist gilt genau dann, wenn es möglich ist, für die Knotenmenge $\mathcal{V}$ eine Aufzählung zu finden, die der Bedingung $\forall i, j \in \mathcal{N}, i \leq j : \langle v_i, v_j \rangle_0 \notin \mathcal{E}$ $(**)$ genügt.

Beweis:

„$\Longleftarrow$" Trivial.

„$\Longrightarrow$" (konstruktiver) Beweis: Man entferne schrittweise von der Menge der Knoten $\mathcal{V}$ einen Knoten v, zu dem keine Kante $\langle v, u \rangle_0 \in \mathcal{E}$ von einem beliebigen Knoten u im Rest von $\mathcal{V}$ führt. Solch ein Knoten v muß existieren, da ja nach Voraussetzung $(*)$ kein Zyklus mit Delay 0 im Graph $\mathcal{G}$ vorhanden ist. Da es nur endlich viele Knoten in $\mathcal{V}$ gibt, muß der Algorithmus mit einer leeren Menge $\mathcal{V}$ terminieren.

Die neue Aufzählung ergibt sich nun durch Indizierung der Knoten in der Reihenfolge ihrer Entnahme: Der n-te entnommene Knoten erhält Index n. Damit folgt aus $i < j$, daß Knoten v_i vor v_j entnommen wurde, also v_j bei der Entnahme von v_i noch in $\mathcal{V}$ vorhanden war, und somit $\langle v_i, v_j \rangle_0 \notin \mathcal{E}$. □

Die neue Indizierung bestimmt nun die Reihenfolge der Aktivationsberechnung, da für sie sichergestellt ist, daß bei der Berechnung einer Aktivation alle Vorgängerknoten (im Graph $\mathcal{G}$, Kanten mit Delay 0) ihre Aktivation bereits evaluiert haben.

2.2 Gradientenabstieg im Gewichtsraum

Wie schon in den einleitenden Beispielen gilt es nun, die Approximationsgüte eines Künstlichen Neuronalen Netzes in Form eines differenzierbaren Fehlermaßes E auszudrücken: Dieses Maß mißt die Abweichung zwischen gewünschter und tatsächlicher Ausgabe – Minimierung des Fehlers ist äquivalent zum Lernen der vorgegebenen gewünschten Ausgabe.

Das Grundprinzip von Gradientenabstieg basiert auf der Tatsache, daß der Gradient des Fehlers bezüglich der internen Netzparameter, zum Beispiel eines Gewichtes, anzeigt, wie dieser Fehler sich bei infinitesimal kleiner Änderung des Parameters verhält, ob er steigt oder sinkt. Ändert man dann den Parameter in Richtung des negativen Gradienten, bewirkt das im infinitesimalen Fall eine infinitesimale Verkleinerung des Fehlers. Nun kann man infinitesimal kleine Parameteränderungen auf digitalen Computern nicht realisieren. Daher werden die Parameter um einen endlichen Bruchteil dieses negativen Gradienten geändert; die dabei verwendete *Schrittweite* η_E sollte dabei so beschaffen sein, daß sie nicht zu klein ist, um das Lernen nicht unnötig zu verlangsamen, aber auch nicht zu groß ist, um Vorzeichenänderungen des (nichtlinearen) Gradienten hinreichend berücksichtigen zu können.

2.2.1 Das Fehlermaß

Obwohl man die zu optimierende Funktion, das Fehlermaß, recht allgemein als differenzierbare Funktion über alle Ausgabewerte aller Trainingsbeispiele definieren kann, haben sich in der Literatur Funktionen durchgesetzt, die über besondere additive Eigenschaften verfügen:

- Der Fehler soll eine *summarische* Funktion über alle Trainingsbeispiele sein. In den beiden einleitenden Lernbeispielen war der Fehler definiert als Summe der Einzelfehler über alle Trainingsmuster $p \in \mathcal{P}$.
- Der Fehler soll ebenfalls eine *summarische* Funktion über die Zeit sein: Die Einzelfehler zu jedem Zeitpunkt werden über das Arbeitsintervall $\{t_0, t_0 + 1, ..., t_1\}$ summiert. Diese Einschränkung ist insbesondere wichtig für den zweiten hier vorgestellten Algorithmus BPFF.
- Schließlich soll der Fehler *Summe* der Einzelfehler der einzelnen Ausgabeknoten sein.

Somit ergibt sich für den Fehler im allgemeinen summarischen Fall:

$$E \equiv \int \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} \zeta_k(p,t)\ \mu(dp) \tag{2.11}$$

Dabei summiert das erste Integral über die Verteilung der Eingabemuster μ, das erste Summenzeichen über die Zeit t im Arbeitsintervall $\{t_0, t_0 + 1, ..., t_1\}$, und das zweite Summenzeichen über alle Ausgabeknoten k. $\zeta_k(p,t)$ mißt den individuellen Fehler des Ausgabeknotens k beim Muster p zum Zeitpunkt t und ist nach der Aktivation des entsprechenden Ausgabeknotens $x_k(t)$ ableitbar.

In den weitaus meisten Anwendungen von Backpropagation, die man in der Literatur findet, ist die gewünschte Funktion eines Netzes durch eine Trainingsmenge definiert. Üblicherweise mißt dann die Fehlerfunktion die quadratische Abweichung (*total square sum*) der tatsächlichen Ausgabe $x_k(t)$ von der sogenannten gewünschten Ausgabe $d_k(t)$.

Das Fehlerfunktional hat dann folgendes Aussehen (vgl. (1.5)):

$$E = \frac{1}{2} \sum_{p \in \mathcal{P}} \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} (d_k(t) - x_k(t))^2 \tag{2.12}$$

Wenn E die quadratische Abweichung zwischen gewünschter und tatsächlicher Trajektorie mißt, ergibt sich für den momentanen Fehlergradienten:

$$e_k(t) = \underbrace{(x_k - d_k)(t)}_{(***)} \tag{2.13}$$

Da die lineare Abweichung $(***)$ zu jeder Zeit meßbar ist, ist $e_k(t)$ für alle p, t und k verfügbar. In Zukunft können wir also $e_k(t)$ als bekannt voraussetzen. Für alle Nicht-Ausgabeknoten v_i ($i \in \mathcal{N} - \mathcal{O}$) sei definiert: $e_i(t) \equiv 0$ ($\forall t$).

Vereinbarung: Um die Notation in Gleichungen ein wenig zu vereinfachen, werden wir in Zukunft gelegentlich den Index p weglassen, ohne dies explizit zu erwähnen. Zur Vervollständigung denke man sich einfach alle Aktivationen x und Gradienten ($\delta, \beta, \gamma, e, ...$, siehe unten) mit dem Musterindex p versehen.

2.2.2 BPTT: Backpropagation durch die Zeit

Nach der Definition des Fehlermaßes werden wir nun zwei Algorithmen zum Training von Künstlichen Neuronalen Netzen vorstellen: Der erste, BPTT (*Backpropagation through time*) macht sich die bereits informell beschriebene Technik des *Unfolding-in-Time* zunutze und erweitert so Backpropagation auf rekurrente Netze. Der zweite, BPFF (*feed forward gradient descent*), ist ein reiner Vorwärtsalgorithmus, in dem das Speichern von Aktivationen aus der Vergangenheit dank des Wegfalls einer Rückwärtsphase überflüssig wird.

Die einfachste Technik der Anwendung von Backpropagation auf ein rekurrentes Netz bildet ein rekurrentes Netz auf ein äquivalentes nichtrekurrentes Netz ab, auf das dann leicht die Backpropagation-Regel angewendet werden kann. Dafür *entfaltet* man das Netz in der Zeit, wie im folgenden beschrieben.

Unfolding-in-Time

Unfolding-in-Time repräsentiert die Zeit als eine eigene Koordinatenachse in dem entfalteten Netz. Dies ist für jedes rekurrente Netz möglich, da Aktivationen grundsätzlich vorwärts in der Zeit berechnet werden: Sei $x(t_0)$, $x(t_0 + 1)$, $x(t_0 + 2)$,...,$x(t_1)$ eine Sequenz von Aktivationen. Es gibt somit eine Berechnungsreihenfolge, in der die einzelnen $x_k(t)$ berechnet werden können. Folglich verbirgt sich dahinter eine zyklenfreie Struktur.

Diese Struktur wird durch Unfolding-in-Time explizit erzeugt. Man kopiert für jeden Zeitschritt $t \in \{t+0, t_0+1, ..., t_1\}$ die Knoten $\mathcal{V}$ des ursprünglichen rekurrenten Netzes $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ und erhält so für jedes t eine Kopie aller Knoten, $\mathcal{V}^t$ genannt. Die Aktivationen von $\mathcal{V}^t$ repräsentieren die Aktivationen in $\mathcal{V}$ *zum Zeitpunkt* t. Daher verbindet jede Kante $\langle v_i, v_j \rangle_n \in \mathcal{E}$ des ursprünglichen Netzes nun $\forall t \in \{t_0, t_0 + 1, ..., t_1 - n\}$ den Knoten v_j in $\mathcal{V}^t$ mit Knoten v_i in $\mathcal{V}^{t+n}$ (mit dem neuen Delay 0, versteht sich). Hatte nun eine Kante das Delay n, verbindet sie je zwei Knoten zweier

Abbildung 2.3: Unfolding-in-Time: Das rekurrente Netz (a) wird in ein äquivalentes nichtrekurrentes Netz (b) durch Einführung der Zeit als Koordinatenachse überführt. Unfolding-in-Time ist hier an einem einfachen Beispielnetz mit $\mathcal{S} = \{0, 1, 2\}, t_0 = 0$ und $t_1 = 3$ demonstriert. Schwarze Kanten sind delaylos, und die Zahlen an den grauen Kanten symbolisieren Delays (1, 1 und 2) von Delaykanten. Das resultierende Netz ist sehr viel größer, aber dafür zyklenfrei.

Kopien, die n Schritte voneinander entfernt sind. Unfolding-in-Time ist also nichts anderes als das Darstellen der Dimension Zeit als Achse im Koordinatensystem. Das resultierende Netz heißt $\mathcal{G}'$.

Abbildung 2.3 zeigt exemplarisch den Vorgang an einem einfachen Beispielnetz. Hier ist das Netz für 4 Zeitschritte aufgefaltet. Man beachte, daß diese Technik nur anwendbar ist, wenn die Berechnung in endlich vielen Zeitschritten erfolgt, also $t_1 \in I\!R$ ist. BPTT ist nicht in der Lage, Gradientenabstieg in unendlichen Trajektorien zu verwirklichen.

Die Rückwärtsphase

Wir können nun für jedes rekurrente Netz mit beliebigen Delays für endlich viele Berechnungszeitschritte ein äquivalentes nichtrekurrentes Netz konstruieren. Backpropagation, wie in der Einleitung vorgestellt, kann nun auf dieses Netz direkt angewendet werden. Dabei ist jedoch zu beachten, daß jeder Knoten $v_i \in \mathcal{V}$ und jede Kante $\langle v_i, v_j \rangle_n \in \mathcal{E}$ des ursprünglichen Netzes $\mathcal{G}$ mehrfach (genauer: Knoten $(t_1 - t_0 + 1)$-fach und Kanten $(t_1 - t_0 - n + 1)$-fach) in $\mathcal{G}'$ vorliegt. Wir werden das daraus resultierende Problem der *Eindeutigkeit von internen Parametern* am Beispiel der Gewichte verdeutlichen: Da zu jeder Kante in $\mathcal{G}$ genau ein Gewicht assoziiert ist, folgt, daß jede der entsprechenden Kanten in $\mathcal{G}'$ einen identischen Gewichtswert haben muß. Die resultierenden Gewichtsänderungen beim Gradientenabstieg führen aber bei diesen Kanten im allgemeinen zu unterschiedlichen Gradienten. Verwendet man diese individuellen Gradienten zur Gewichtsadaption, ist die Gleichheitsbedingung verletzt. Daher wird stattdessen der über alle Kopien arithmetisch gemittelte Gradient verwendet. Wegen der Produktregel der Differentiation ist dies der exakte Gradient.

Definiert man gemäß Gleichung (1.14) im einleitenden Beispiel die Hilfsgradienten $\delta_i(t)\ \forall i \in \mathcal{N}, t \in [t_0, t_1]$ mit:

$$\delta_i(t) \equiv \frac{\partial E}{\partial net_i}(t) \tag{2.14}$$

dann ergibt sich daraus folgende Propagationsregel (vgl. (1.15) und (1.16)):

$$\begin{aligned}
\delta_i(t) &= \frac{\partial E}{\partial net_i}(t) = \frac{\partial \zeta_i(t)}{\partial net_i(t)} + \sum_{n=0}^{\min\{t_1-1,\hat{n}\}} \sum_{j\in\mathcal{N}} \frac{\partial E}{\partial net_j}(t+n)\frac{\partial net_j(t+n)}{\partial net_i(t)} \\
&= \frac{\partial \zeta_i(t)}{\partial x_i(t)}\frac{\partial x_i(t)}{\partial net_i(t)} + \sum_{n=0}^{\min\{t_1-1,\hat{n}\}} \sum_{j\in\mathcal{N}} \delta_j(t+n)\frac{\partial net_j(t+n)}{\partial x_i(t)}\frac{\partial x_i(t)}{\partial net_i(t)} \\
&\overset{(2.13)}{=} \sigma'(net_i(t)) \cdot \left(e_i(t) + \sum_{n=0}^{\min\{t_1-1,\hat{n}\}} \sum_{j\in\mathcal{N}} w_{ijn}\delta_j(t+n)\right)
\end{aligned} \tag{2.15}$$

Man sieht klar, daß die Propagation *rückwärts in der Zeit* erfolgt. Die in der Rückwärtsphase verwendeten Netzeingaben $net_i(t)$ müssen während der Vorwärtsphase zwischengespeichert werden. Da die Rückwärtsphase separat für jedes Trainingsbeispiel (Sequenz) aus $\mathcal{P}$ durchgeführt werden kann (dies ergibt sich aus der Additivität der Fehlerfunktion und wird deutlich in den im folgenden hergeleiteten Gleichungen), hängt die Speicherkomplexität von der Sequenzlänge der längsten Sequenz in $\mathcal{P}$ ab. Man sagt daher, BPTT sei nicht lokal in der Zeit. Dieses Problem hat der zweite hier vorgestellte Algorithmus, BPFF, (Kapitel 2.2.3) nicht.

Gradientenabstieg mit Backpropagation

Wir haben nun für allgemeine Künstliche Neuronale Netze eine Rückwärtspropagationsregel für die Hilfsgradienten $\delta_i(t)$. Diese können nun direkt zum Gradientenabstieg verwendet werden. Dabei definiere man:

$$\begin{aligned}
\Delta_{w_{ijn}} &\equiv \frac{\partial E}{\partial w_{ijn}} = \sum_{p\in\mathcal{P}} \sum_{t=t_0}^{t_1} \frac{\partial E}{\partial net_i(t)}\frac{\partial net_i(t)}{\partial w_{ijn}} \\
&= \sum_{p\in\mathcal{P}} \sum_{t=t_0+n}^{t_1} \delta_i(t) x_j(t-n)
\end{aligned} \tag{2.16}$$

(Anmerkung: Hier und im folgenden wird davon ausgegangen, daß die Trainingsbeispiele in einer endlichen Mustermenge $\mathcal{P}$ vorliegen (vgl. Kapitel 2.2.1). Ist die nicht der Fall, ersetze man die Summe über alle p in (2.16) durch das Integral über $\mu(dp)$ gemäß (2.11).)

Anschaulich mißt $\delta_{w_{ijn}}$ die punktweise Abhängigkeit des Fehlers E bezüglich einer Änderung des Gewichts w_{ijn}. Gradientenabstieg bedeutet nun, das Gewicht einen kleinen Schritt in die entgegengesetzte Richtung dieses Gradienten zu ändern. Dies führt zu der Lernregel Backpropagation für rekurrente Netze:

$$\boxed{\begin{aligned} w_{ijn} &\longleftarrow w_{ijn} - \eta_E \frac{\partial E}{\partial w_{ijn}} \\ &\overset{(2.16)}{=} w_{ijn} - \eta_E \sum_{p \in \mathcal{P}} \sum_{t=t_0+n}^{t_1} \delta_i(t) x_j(t-n) \end{aligned} \qquad (2.17)}$$

Dabei ist $\eta_E > 0$ die *Schrittweite* (*Lernrate*). Die Korrektheit der Gradienten gilt nur für infinitesimal kleine Änderungen, und schon geringe „echte" Änderungen der Gewichte kann eine Vorzeichenänderung der Gradienten verursachen. Ist dies der Fall, kann Gradientenabstieg zur Vergrößerung des Fehlers E führen. Daher ist es von zentraler Wichtigkeit, die *Schrittweite* η_E richtig zu wählen. Ohne dieses Problem hier weiter zu elaborieren, gehen wir künftig davon aus, daß η_E brauchbar gewählt ist.

Dies vervollständigt die Lernregel Backpropagation für das Adaptieren von Gewichten. Es sei erneut darauf hingewiesen, daß *Schwellwerte*, wie in Kapitel 1.1.1 eingeführt, in dem hier verwendeten erweiterten Gewichtsbegriff subsumiert sind (vgl. auch Kapitel 2.1.1). Der komplette Algorithmus BPTT ist in Abbildung 2.4 zusammengefaßt.

2.2.3 BPFF: Ein feed-forward-Lernalgorithmus

Das Prinzip von Backpropagation als Gradientenabstieg läßt sich mit wenigen Worten so umschreiben: Man berechne für jedes Eingabemuster die Aktivationen $x_i(t)$, natürlich vorwärts durch die Zeit. Nach der vollständigen Berechnung der Aktivationen und Ermitteln des Fehlers ist die Vorwärtsphase abgeschlossen, und es beginnt die Rückwärtsphase. Dabei werden, beginnend mit den letzten Ausgabeknoten, die Gradienten $\delta_k(t_1)$ berechnet. Diese messen, wie die entsprechenden Netzeingaben $net_k(t_1)$ den Fehler E beeinflussen.[2] Für reine Ausgabeknoten ist die Berechnung des Gradienten trivial. Die Rückwärtspropagationsregel beruht auf der Kettenregel: Der Effekt der Netzeingabe eines beliebigen Knotens $net_i(t)$ auf seine Nachfolgerknoten ist durch die Ableitung

[2] Man hätte die gleiche Mathematik auch bezogen auf Aktivationen statt auf Netzeingaben herleiten können; ist die Squashing-Funktion bijektiv, sind Netzeingaben und Aktivationen jerderzeit ineinander überführbar.

initialisiere w_{ijn} zufällig $\forall i, j \in \mathcal{N},\ n \in \mathcal{S}$

wiederhole

$\Delta w_{ijn} := 0\ \forall i, j \in \mathcal{N},\ n \in \mathcal{S}$

für alle $p \in \mathcal{P}$ iteriere
$E := 0$;
$x_i(t_0) := \tilde{x}_i\ \ \forall i \in \mathcal{N}$;

für $t := t_0$ bis t_1 mit Inkrement 1 iteriere
berechne $I_i^p(t)\ \ \forall i \in \mathcal{N}$;
berechne $net_i(t)$ und $x_i(t)\ \ \forall i \in \mathcal{N}$; mit (2.2) und (2.3)
$E := E + E_t$;
Ende Schleife t

für $t := t_1$ bis t_0 mit Inkrement -1 iteriere
berechne $e_k^p(t)\ \ \forall k \in \mathcal{O}$;
berechne $\delta_i^p(t)\ \ \forall i \in \mathcal{N}$; mit (2.15)
$\Delta w_{ijn} := \Delta w_{ijn} - \eta_E \delta_i^p(t) x_j(t-n)$
$\forall i, j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$;
Ende Schleife t

Ende Schleife p

$w_{ijn} := w_{ijn} + \Delta w_{ijn}\ \ \forall i, j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$;
bis $E < \varepsilon$

Abbildung 2.4: BPTT: Der Backpropagation through time-Algorithmus.

der Aktivationsfunktion gegeben. Kennt man nun $\delta_j(t+n)$ für alle Nachfolger dieses Knotens – das sind alle Knoten, die mit Knoten i durch eine Kante $\langle v_j, v_i \rangle_n$ verbunden sind – kann man kraft der Kettenregel den Gradienten berechnen, der angibt, wie eine infinitesimal kleine Änderung der Netzeingabe $net_i(t)$ die Fehlerfunktion ändert: $\delta_i(t)$. Die Kettenregel macht sich die einfache Tatsache zunutze, daß Einflüsse auf den Fehler nur entlang der Kanten im Netz weitergereicht werden. In diesem Sinne ist Backpropagation *lokal im Raum*.

Der im folgenden beschriebene Algorithmus, BPFF, realisiert ebenfalls Gradientenabstieg, vermeidet aber die Rückwärtsphase und ist lokal in der Zeit. Dies geschieht dadurch, daß während der Vorwärtsphase bereits Gradienten $\beta_{ijn}^k(t)$ propagiert werden, die messen, wie ($\forall i, j, n, k$) das Gewicht w_{ijn} die Aktivation $x_k(t)$ beeinflusst. Trivialerweise beeinflusst kein Gewicht initiale Aktivationen – diese sind a priori gegeben. Kennt man nun für alle Vorgänger (bzgl. des Graphen $\mathcal{G}$)

v_l eines Knotens v_k die Gradienten $\beta^l_{ijn}(t-n)$, kann man mithilfe der Aktivationsfunktion und der Kettenregel berechnen, wie die Aktivation $x_k(t)$ durch das Gewicht w_{ijn} beeinflusst wird: $\beta^k_{ijn}(t)$. Hier wird die Tatsache ausgenutzt, daß derartige Einflüsse nur über die hinführenden Kanten von v_k weitergereicht werden können.

Es ist leicht zu sehen, daß man durch die simultane Vorwärtspropagation der Hilfsgradienten $\beta^k_{ijn}(t)$ zusammen mit den Aktivationen Gradienten aller Aktivationen bezüglich aller Gewichte erhält. Damit kann ohne weitere Propagation berechnet werden, wie Gewichtsänderungen den Fehler beeinflussen und die Gewichte adaptieren. Im Prinzip scheint dieser Algorithmus auch nicht lokal in der Zeit zu sein, da man offensichtlich für Ausgabeknoten die Gradienten $\beta^k_{ijn}(t)$ für alle Zeitschritte zwischenspeichern muß. Wegen der zeitlichen Additivität der Fehlerfunktion E aber – eine fundamentale Voraussetzung – können die Einzelgradienten über die Zeit kumuliert werden, was die angenehme Eigenschaft *zeitlicher Lokalität* sichert.

Auf der anderen Seite erkauft man sich dies durch eine erhöhte Komplexität. Im Gegensatz zu Backpropagation, bei dem entlang einer Kante (im entfalteten Netz) nur genau ein Gradient zurückpropagiert wird, wird hier eine ganzer Tensor von Gradienten weitergereicht, nämlich ein Gradient für jede Kante $w_{ijn} \in \mathcal{E}$. Der Rechenaufwand hängt bei sequentiellen Realisierungen also von der Mächtigkeit der Kantenmenge $\mathcal{E}$ ab. Obwohl man den Gradientenfluss als lokal bezüglich der Topologie des Netze bezeichnen kann, stufen manche Autoren einen solchen Algorithmus als *nicht lokal im Raum* ein.

Vorwärtspropagation von Hilfsgradienten

Wir werden nun formal eine Vorwärtspropagationsregel für die Gradienten herleiten, die die Abhängigkeit der Aktivationen von den Gewichten messen. Diese Gradienten sind folgendermaßen definiert:

$$\beta^k_{ijn}(t) \equiv \frac{\partial x_k}{\partial w_{ijn}}(t) \tag{2.18}$$

Die Propagationsregel lautet dann:[3]

[3] Hier ist δ_{ik} das Kronecker-Symbol

$$\delta_{ik} \begin{cases} 1 & \text{wenn } i = k \\ 0 & \text{sonst} \end{cases}$$

und nicht zu verwechseln mit dem δ_i der Backpropagation-Gleichungen (2.14).

$$
\begin{aligned}
\beta^k_{ijn}(t_0) &= 0 \\
\beta^k_{ijn}(t) &= \frac{\partial x_k}{\partial w_{ijn}}(t) \;=\; \frac{\partial x_k}{\partial net_k}(t)\frac{\partial}{\partial w_{ijn}}\left(\sum_{l\in\mathcal{N}}\sum_{m=0}^{\min\{t-t_0,\hat{n}\}} w_{klm}x_l(t-m)\right) \\
&= \sigma'_k(net_k(t))\cdot\left(\delta_{ik}x_j(t-n)+\sum_{l\in\mathcal{N}}\sum_{m=0}^{\min\{t-t_0,\hat{n}\}} w_{klm}\beta^l_{ijn}(t-m)\right) \\
&\quad \forall\;\; t\in\{t_0+1,t_0+2,...,t_1\} \qquad (2.19)
\end{aligned}
$$

Gradientenabstieg im Gewichtsraum

Die folgende Regel beschreibt Gradientenabstieg:

$$
\begin{aligned}
w_{ijn} &= w_{ijn}\underbrace{-\eta_E\frac{\partial E}{\partial w_{ijn}}}_{\Delta w_{ijn}} \\
&= w_{ijn}-\eta_E\sum_{t=t_0}^{t_1}\sum_{k\in\mathcal{O}} e_k(t)\beta^k_{ijn}(t) \qquad (2.20)
\end{aligned}
$$

mit einer geeigneten Schrittweite $\eta_E > 0$.

Abbildung 2.5 skizziert den resultierenden Algorithmus für feed-forward Gradientenabstieg in Künstlichen Neuronalen Netzen BPFF.

2.2.4 Zur Berechenbarkeit der Gradienten

Da bei der Berechnung der einzelnen Gradienten $\beta^k_{ijn}(t)$ auf andere Gradienten desselben Typs zurückgegriffen wird, muß sichergestellt werden, daß es eine Berechnungsreihenfolge gibt. Wir werden zeigen, daß aus der Existenz einer Berechnungsreihenfolge für Aktivationen konstruktiv eine ebensolche für die Gradienten $\beta^k_{ijn}(t)$ folgt.

Da wir die Initialwerte $\beta^k_{ijn}(t_0)$ kennen, reicht es aus, durch Induktion über die Zeit t zu zeigen, daß unter Kenntnis aller $\beta^k_{ijn}(t')$ früherer Zeitschritte, d.h. $t' < t$, alle $\beta^k_{ijn}(t)$ berechnet werden können. Es ist sogar ausreichend, die Verfügbarkeit der Gradienten im Zeitfenster $\{\max\{t_0, t -$

`initialisiere` w_{ijn} `zufällig` $\forall i,j \in \mathcal{N},\ n \in \mathcal{S}$

`wiederhole`
  $\Delta w_{ijn} := 0\ \ \forall i,j \in \mathcal{N},\ n \in \mathcal{S}$;

  `für alle` $p \in \mathcal{P}$ `iteriere`
    $E := 0$;
    $x_i(t_0) := \tilde{x}_i\ \ \forall i \in \mathcal{N}$;
    $\beta^k_{ijn}(t_0) := 0\ \ \forall i,j \in \mathcal{N}, k \in \mathcal{O},\ n \in \mathcal{S}$;

    `für` $t := t_0$ `bis` t_1 `mit Inkrement` 1 `iteriere`
      `berechne` $I^p_i(t)\ \ \forall i \in \mathcal{N}$;
      `berechne` $net_i(t)$ `und` $x_i(t)\ \ \forall i \in \mathcal{N}$; mit (2.2) und (2.3)
      `berechne` $e^p_k(t)\ \ \forall k \in \mathcal{O}$;
      `berechne` $\beta^k_{ijn}(t)\ \ \forall i,j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$; mit (2.19)
      $E := E + E_t$;
      $\Delta w_{ijn} := \Delta w_{ijn} - \eta_E \sum_{k \in \mathcal{O}} e^p_k(t) \cdot \beta^k_{ijn}(t)$;
        $\forall i,j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$;

    `Ende Schleife` t
  `Ende Schleife` p

  $w_{ijn} := w_{ijn} + \Delta w_{ijn}\ \ \forall i,j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$;

`bis` $E < \varepsilon$

Abbildung 2.5: BPFF: Der feed-forward Backpropagation Algorithmus.

$\hat{n}\}, \ldots, t-1\} = \{t - \min\{t-t_0, \hat{n}\}, \ldots, t-1\} \subseteq \{t-\hat{n}, \ldots, t\}$ zu fordern, so daß Gradienten nur in einem $\hat{n}$-Zeitfenster gespeichert werden, und die Speicherkapazität dieses Algorithmus nicht von der Größe des Arbeitsintervalls $\{t_0, t_0+1, ..., t_1\}$ abhängt.

Nach Induktionsvoraussetzung taucht der kritische Fall nur für Kanten mit Delay 0 auf – für alle anderen wird in (2.19) auf bereits bekannte Gradienten zurückgegriffen.

Zur Vereinfachung nehme man an, daß die Knoten in $\mathcal{V}$ bereits gemäß $(**)$ numeriert sind (vergleiche Kapitel 2.1.5), d.h., $\forall k,l \in \mathcal{N}$ mit $k < l$ gilt: $\langle v_j, v_i\rangle_n \notin \mathcal{E}$. Damit kann (2.19) reduziert werden zu

$$\beta^k_{ijn}(t) \;=\; \sigma'_k(net_k(t)) \cdot \left[\delta_{ik} x_j(t-n) + \underbrace{\sum_{l<k} w_{kl0}\beta^l_{ijn}(t)}_{(****)} + \sum_{l \in \mathcal{N}} \sum_{m=1}^{\min\{t-t_0,\hat{n}\}} w_{klm}\beta^l_{ijn}(t-m) \right]$$

Alle Terme in dieser Gleichung sind unkritisch, mit der Ausnahme des mit $(****)$ gekennzeichneten Ausdrucks. Der kritische Term $(****)$ zeigt, daß nur $\beta^l_{ijn}(t)$ mit $l < k$ für die Berechnung der $\beta^k_{ijn}(t)$ benötigt werden. Daraus folgt eine mögliche Berechnungsreihenfolge:

$$(\beta^1_{ijn}(t)), (\beta^2_{ijn}(t)), ..., (\beta^N_{ijn}(t)) \tag{2.21}$$

und somit die Berechenbarkeit. □

2.3 Heuristische Modifikationen

Wir haben nun Künstliche Neuronale Netze formal definiert und zwei Algorithmen für Lernen mit Gradientenabstieg hergeleitet. Gradientenabstieg ist eine Suchheuristik, die insbesondere davon ausgeht, daß die lineare Approximation eines Gradienten im Bereich eines Adaptionsschritts hinreichend genau ist. Des weiteren wird bei weitem nicht der gesamte Parameterraum durchsucht, was oft suboptimale Lösungen zur Folge hat.

In der Literatur findet sich eine Vielzahl heuristischer Erweiterungen für Gradientenabstieg. Diese haben meist eine recht anschauliche Motivation und führen in empirischen Untersuchungen oft zu schnellerer oder besserer Konvergenz und/oder Generalisierung. Wir wollen hier einige wenige aufgreifen, ohne in irgendeiner Form den Anspruch auf Vollständigkeit zu erheben. Es findet sich Evidenz, daß die Bekanntheit von Backpropagation, zumindest der hohe Citation-Index, auf die Tatsache zurückzuführen ist, daß Backpropagation in vielen Anwendungen derart langsam ist, daß jede halbwegs gelungene Heuristik Backpropagation in bestimmten Anwendungen schlägt. Insbesondere das Fehlen einer Konvergenztheorie dieses nichtlinearen Approximators trägt dazu bei, daß empirische Untersuchungen Aussagekraft haben. Leider gibt es allerdings kaum einen weithin verwendeten Benchmark-Test.

Wir werden im folgenden recht unterschiedliche Heuristiken beschreiben, die sich in einer Vielzahl von Publikationen wiederfinden.

2.3.1 Online-Lernen

Beim *Online-Lernen* handelt es sich um ein Verfahren, das Parameter nicht erst nach Präsentation der ganzen Trainingsmenge, sondern öfter, zum Beispiel nach jeder Trainingssequenz oder sogar nach jedem Zeitschritt modifiziert. Diese Heuristik hat eindeutig den Charakter sequentieller Simulation: Alle Muster und alle Sequenzen werden sequentiell verarbeitet. Das Adaptieren der Gewichte nach jedem Schritt (anstatt nach der gesamten Trainingsmenge) ist verhältnismäßig günstig und von geringerer Komplexität als die eigentliche Gradientenabstiegsberechnung. Bei starken Redundanzen in der Trainingsmenge kann dies in der Tat die Konvergenz deutlich beschleunigen.

Online-Learning ist auch bedeutend, wenn bei einer zugrundeliegenden Verteilung μ einer unendlichen Menge $\mathcal{P}$ pro Zeittakt nur ein Beispiel $p \in \mathcal{P}$ verfügbar ist. Dieses wird zum Beispiel bei den in Kapitel 3 beschriebenen Kontrollexperimenten der Fall sein. Das korrekte Erfassen einer Trainingsmenge ist, insbesondere, wenn die zu lernende Funktion zeitlich variiert, unpraktikabel oder sogar unmöglich – das Adaptieren an einzelne Bespiele geschieht daher gewöhnlich online.

Online-Lernen nach jeder Musterpräsentation

Eine in der Literatur häufig im Zusammenhang mit nichtrekurrenten Netzen diskutierte Form des Online-Lernens ist Online-Lernen nach jeder Musterpräsentation. Dieses Verfahren adaptiert nach jeder Präsentation eines Trainingsbeispiels, und nicht erst nach der Präsentation der gesamten Trainingsmenge, die internen Parameter des Netzes. Dafür trennt man die Fehlerfunktion E in disjunkte Teile E_p auf (vergleiche Gleichungen (2.11) und (2.12)):

$$E_p \quad = \quad \tfrac{1}{2} \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} \zeta_k(p,t) \tag{2.22}$$

und adaptiert nach jedem Muster p die Gewichte bezüglich E_p:

$$\begin{aligned} w_{ijn} &\longleftarrow w_{ijn} + \Delta w_{ijn} \\ \Delta w_{ijn} &= -\eta'_E \frac{\partial E_p}{\partial w_{ijn}} = -\eta'_E \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} \zeta_k(p,t) \end{aligned} \tag{2.23}$$

Dabei wird die neue Schrittweite $\eta'_E > 0$ auf $\eta_E \cdot |\mathcal{P}|^{-1}$ (bei unendlicher Patternmenge ist dieser Ausdruck $\eta_E \cdot \mu(p)$) gesetzt.

Online-Lernen nach jedem Zeitschritt

Eine weitere Verfeinerung der Lernzyklen wird erreicht, wenn nach jedem Schritt innerhalb einer Sequenz (eines Musters) adaptiert wird. Die Fehlerfunktion für diesen Fall ist

$$E_p(t) = \sum_{k \in \mathcal{O}} \zeta_k(p,t). \tag{2.24}$$

Die Gewichtsänderung ist dann

$$\begin{aligned} \Delta w_{ijn} &= -\eta''_E \frac{\partial E_p(t)}{\partial w_{ijn}} = -\eta''_E \sum_{k \in \mathcal{O}} e_k(t) \beta^k_{ijn}(t) \\ \text{mit } -\eta''_E &= \frac{\mu(p)}{(t_1 - t_0 + 1)} = \frac{1}{|\mathcal{P}| \cdot (t_1 - t_0 + 1)} \end{aligned} \tag{2.25}$$

Wir haben diese zweite Online-Technik durch den feed-forward Algorithmus realisiert, obwohl BPTT dies auch zugelassen hätte. Es ist jedoch anzumerken, daß bei BPTT jeder Online-Lernschritt eine komplette Rückpropagationsphase erfordert – dies vergrößert die Komplexität um den Faktor t. Dagegen bleibt die Komplexität beim feed-forward-Algorithmus unberührt. Dies macht BPTT bei rekurrenten Netzen für Online-Lernen in der Regel unpraktikabel.

2.3.2 Teacher Forcing

Teacher Forcing ist eine Technik, die nur für Sequenzverarbeitung in bestimmten rekurrenten Netzen und in Verbindung mit Online-Lernen von Interesse ist. Zentrale Idee ist, daß von Netzen, deren Ausgabe mit rekurrenten Kanten zurückgeleitet wird, bei fehlerhafter Ausgabe zum Zeitpunkt t keine korrekte Ausgabe zum nächsten Zeitpunkt $t + \Delta t$ erwartet werden kann, selbst wenn die durch das Netz realisierte Funktion korrekt wäre. Der Grund liegt darin, daß falsche Ausgabewerte die Folgeaktivationen beeinflussen, und der Fehler sich fortpflanzt. Dieses Problem – so die Idee von Teacher Forcing – wird eingedämmt, indem man *nach* der Anwendung eines Gradientenabstiegschritts (Online-Lernen) die entsprechenden Aktivationen von Hand so setzt, daß sie nun nicht mehr fehlerhaft sind. Falls Zielausgaben $d_k(t)$ existieren, setzt man die Aktivationen der Ausgabeknoten ($\forall k \in \mathcal{O}$) genau auf diese Zielausgaben:

$$x_k(t) \quad \longleftarrow \quad d_k(t) \tag{2.26}$$

Teacher Forcing hat den Nachteil, daß man ausschließlich die Aktivationen der Ausgabeknoten korrigiert. Fehlerhafte Ausgabeaktivationen bedeuten aber in der Regel auch fehlerhafte Aktivationen bei den internen Knoten, die ebenfalls korrigiert werden müßten. Die Wechselwirkung zwischen korrigierten und nicht korrigierten Aktivationen kann in ungünstigen Fällen das Lernen verlangsamen.

2.4 Spezielle Architekturen Künstlicher Neuronaler Netze

Bisher haben wir Künstliche Neuronale Netze allgemeiner definiert als gemeinhin in der Literatur üblich. Kanten sind grundsätzlich zwischen je zwei Knoten in beiden Richtungen zugelassen, wobei beliebige Delays verwendet werden können. Diese Menge von Netzen ist sehr mächtig – in der Tat ist sie eine Obermenge der meisten in der Literatur verwendeten Netze – aber zugleich auch sehr unstrukturiert. Wir wollen hier einige wenige prinzipielle Architekturen vorstellen, die in der Literatur vermehrt zitiert werden: Nicht-rekurrente Multilayerperceptrons, sowie vollverbundene und bestimmte teilverbundene geschichtete rekurrente Netze. Zum einem dient dies als Einführung in die Welt der Netztopologien; zum anderen wird demonstriert, daß die hier verwendete Definition all diese Netzformen subsumiert. In allen Abbildungen ist aus Gründen der Übersichtlichkeit der Schwellwertknoten v_* nicht gezeigt.

2.4.1 Nicht-rekurrente Multilayerperceptrons

Nicht-rekurrente Multilayerperceptrons, die wohl die Literatur dominierende Netzarchitektur, haben wir schon in der Einleitung kennengelernt. Die Trainingsbeispiele sind dabei statisch und enthalten keine Sequenzen, es gibt weder Delays noch Rekurrenzen. Multilayerperceptrons (Mehrschichtnetze) sind in Schichten organisierte feed-forward Netze, wobei die Knoten jeder

Abbildung 2.6: Zwei Multilayerperceptrons (Mehrschichtnetze). Die meisten in der Literatur verwendeten Backpropagationnetze sind geschichtete Netze mit einer internen Schicht. Selten findet man Netze mit zwei internen Schichten, und fast nie Netze mit mehr als zwei Schichten. In dieser und den folgenden Abbildungen ist der Schwellwertknoten v_* nicht gezeigt.

Schicht vollverbunden mit der Vorgängerschicht sind. In Abbildung 2.6 sind zwei prototypische Multilayerperceptrons dargestellt.

Formal gilt:

- Das maximale Delay ist $\hat{n} = 0 \Longrightarrow \mathcal{S} = \{0\}$. Das Netz arbeitet ohne zeitlichen Kontext.
- Die Menge der Knotenindizes $\mathcal{N}$ ist in $\rho + 2$ disjunkte Mengen zerlegt: $\mathcal{N} = \mathcal{I} \uplus \mathcal{H}_1 \uplus \mathcal{H}_2 \uplus ... \uplus \mathcal{H}_{\rho-1} \uplus \mathcal{O} \uplus \{v_*\}$. Dabei sind $\mathcal{I}$ die Indizes der Eingabeknoten, $\mathcal{H}_i$ die der i-ten internen Schicht und $\mathcal{O}$ die der Ausgabeknoten. v_* ist der Schwellwertknoten (vgl. Kapitel 2.1.1). Man sagt, das Netz habe ρ Schichten (die Eingabeschicht und der Schwellwertknoten werden nicht mitgezählt).
- Zwei aufeinanderfolgende Schichten $\rho' - 1$ und ρ' sind vollverbunden, d.h. es gilt $\forall \rho' = 1, 2, ..., \rho, \forall i \in \mathcal{H}_{\rho'}, j \in \mathcal{H}_{\rho'-1}$: $\langle v_i, v_j \rangle_0 \in \mathcal{E}$. Hier gilt zur Vereinfachung der Schreibweise $\mathcal{H}_0 \equiv \mathcal{I}$ und $\mathcal{H}_\rho \equiv \mathcal{O}$.
- Der Schwellwertknoten v_* ist mit allen Knoten, bis auf denen der Eingabeschicht, verbunden: $\forall i \in \mathcal{N} - \mathcal{I}$: $\langle v_i, v_* \rangle_0 \in \mathcal{E}$.
- Es ist keine andere Kante in $\mathcal{E}$.
- Die Eingabeknoten v_i, $i \in \mathcal{I}$, haben keine Squashing-Funktion; Aktivation ist die externe Eingabe I_i. Für alle anderen Knoten gilt: $a = 0$, $b = c = 1$ (vgl. Kapitel 2.1.4).

Man vereinbart dann noch zur Vereinfachung:

- Die Knoten seien gemäß $(**)$ geordnet (Kapitel 2.1.5).

Abbildung 2.7: Beispiel für ein vollverbundenes, rekurrentes Netz. Alle Kanten haben Delay 1. Vollverbundene Netze sind strukturlos, und zeigen in der Praxis häufig schlechtes Konvergenz- und Lernverhalten.

- Der Delayindex 0 der Kanten und Gewichte wird weggelassen
- Der Zeitindex (t) wird weggelassen

Man erhält damit ein Standard-Multilayerperceptron resp. nichtrekurrentes Backpropagationnetz. Beispiele für zwei Multilayerperceptrons sind in Abbildung 2.6 gezeigt.

2.4.2 Vollverbundene rekurrente Netze

Diese Netze sind vollverbundene Netze, wobei jede Kante mit einem Delay von 1 versehen ist ($\mathcal{S} = \{1\}$). Eine Untermenge der Knoten wird zu Eingabeknoten, eine andere nichtnotwendigerweise disjunkte Untermenge zu Ausgabeknoten erklärt. Charakteristisch ist, daß jede Aktivationspropagation hier Zeit benötigt – in einem Schritt kann Information nur entlang einer Kante weitergereicht werden. Ein Prototyp ist in Abbildung 2.7 gezeigt. Wegen der fehlenden Struktur stellen vollverbundene rekurrente Netze mathematisch zwar eine Obermenge aller strukturierten Netze dar, doch haben sie sich in der Praxis wegen Konvergenzschwierigkeiten für größere Probleme oft als nachteilhaft erwiesen.

2.4.3 Jordan-Netze

Diese nach Michael Jordan benannte Architektur ist strukturierter als die vollverbundenen rekurrenten Netze. Man kann sich ein solches Netz als ein um einige, wenige Rekurrenzen erweitertes Multilayerperceptron vorstellen: Neben den gewöhnlichen Eingabeknoten existieren sog. *Kontextknoten*, die ebenfalls vollverbunden mit der ersten internen Schicht sind. Die Ausgabe dieses Multilayerperceptrons wird über rekurrente Kanten mit dem Delay 1 in die Kontextschicht 1:1

Abbildung 2.8: Jordan-Netze erweitern nichtrekurrente Mehrschichtnetze durch rekurrente Kanten mit Delay 1. Diese Kanten verbinden die Ausgabeschicht mit einer sogenannten Kontextschicht, die wie ein Teil der Eingabe agiert. Die Kontextschicht und die Ausgabeschicht haben gleichviele Knoten, und nach jeder Vorwärtsphase werden die Ausgabeaktivationen eins-zu-eins in die Kontextknoten kopiert. Dies ermöglicht den Zugriff auf die Ausgabe des letzten Zeitschritts.

---

kopiert, die ihrerseits genauso viele Knoten hat wie die Ausgabeschicht. Damit kann Information, zeitlicher Kontext, im Prinzip beliebig lange im Netz gespeichert werden, vorausgesetzt die Repräsentation der Ausgabe gestattet dies, denn nur die Ausgabe wird rekurrent in das Netz gespeist. Abbildung 2.8 zeigt ein Jordan-Netz.

Formal muß man die Restriktionen für ein Multilayerperceptron ein wenig ändern:

- Die Menge der Delays ist $\mathcal{S} = \{0, 1\}$.
- Die Menge der Knotenindizes ist zerlegt in: $\mathcal{N} = \mathcal{I} \uplus \mathcal{C} \uplus \mathcal{H}_1 \uplus \mathcal{H}_2 \uplus ... \uplus \mathcal{H}_{\rho-1} \uplus H_\rho = \mathcal{O} \uplus \{v_*\}$. Dabei gilt: $\mathcal{H}_0 = \mathcal{I} \uplus \mathcal{C}$ und $|\mathcal{C}| = |\mathcal{O}|$. $\mathcal{C}$ indiziert die Menge der *Kontextknoten*.
- Neben den für ein Multilayerperceptron definierten Kanten mit Delay 0 gibt es noch $|\mathcal{O}|$ Kanten mit Delay 1, die jeden Ausgabeknoten mit dem entsprechenden Kontextknoten verbindet.

2.4.4 Elman-Netze

Elman-Netze sind den Jordan-Netzen strukturell ähnlich, besitzen aber einen fundamentalen Unterschied: Anstelle der Ausgabeaktivationen werden die Aktivationen der internen Knoten in die Kontextschicht kopiert (siehe Abbildung 2.9) . Die formale Definition von Elman-Netzen ist der der Jordan-Netze sehr ähnlich. Der kleine Unterschied in der Konnektivität bringt aber einige bedeutende funktionelle Unterschiede mit sich. So haben Elman-Netze die Fähigkeit, eine eigene *interne Repräsentation* der zeitlich relevanten Information zu entwickeln. Dies liegt daran, daß

Abbildung 2.9: Ein Elman-Netz. Im Gegensatz zu den zuvor diskutierten Jordan Netzen werden in Elman-Netzen die Aktivationen der verborgenen Schicht in die Kontextschicht kopiert.

---

die Repräsentation von Information in der internen Schicht nicht a priori vorgegeben ist. In Jordan-Netzen dagegen ist die Repräsentation dessen, was zeitlich weitergegeben wird, durch die Trainingsmenge bestimmt, da diese ja die Funktion der Ausgabeknoten bestimmt. Elman-Netze werden in der Regel unter anderem dann den Jordan-Netzen vorgezogen, wenn die optimale Kontextrepräsentation und -information nicht bekannt ist.

Es sei angemerkt, daß in den Originalpublikationen für Jordan- und Elman-Netze Backpropagation nicht, wie hier geschehen, rekurrent angewendet wurde. Stattdessen wurde Backpropagation zu jedem Zeitschritt wie in Kapitel 1 beschrieben benutzt. Im Falle von Jordan-Netzen ist dieser Unterschied weniger bedeutend, für Elman-Netze hingegen macht dies wegen der unspezifizierten Repräsentation in der Kontextschicht einen deutlichen Unterschied. Elman-Netze werden inzwischen meist rekurrent trainiert.

2.4.5 Time-Delay Neural Networks (TDNN)

Diese Netze sind ebenfalls mehrschichtige feed-forward Netze, doch ist jeder Knoten mit seinem Vorgänger über $\hat{n}+1$ Kanten mit den Delays $0, 1, ..., \hat{n}$ verbunden. TDNNs werden viel in der Sprachverarbeitung eingesetzt, wo es gilt, Informationen unter Einbeziehung eines endlichen Kontext (Zeitfenster) zu klassifizieren. Abbildung 2.10 zeigt ein Beispiel.

Wegen der starken Ähnlichkeit mit Multilayerperceptrons gelten alle dortigen Restriktionen, ausgenommen der zweiten und der vierten. Es gilt nun:

- Die Menge der Delays ist $\mathcal{S} = \{0, 1, ..., \hat{n}\}$.
- Zwei aufeinanderfolgende Schichten $\rho' - 1$ und ρ' sind vollverbunden, d.h., es gilt $\forall n \in \mathcal{S}$, $\forall \rho' = 1, 2, ..., \rho, \forall i \in \mathcal{H}_{\rho'}, j \in \mathcal{H}_{\rho'-1}$: $\langle v_i, v_j \rangle_n \in \mathcal{E}$.

Abbildung 2.10: Ein Time Delay Neural Network (TDNN). Man erhält ein TDNN, indem man bei einem nichtrekurrenten, schichtweise vollverbundenen Mehrschichtnetz jede Kante durch $\hat{n}+1$ Kanten mit den Delays $0, 1, \ldots, \hat{n}$ substituiert.

Delays $n > 1$ finden sich gewöhnlich nur bei TDNNs, nicht aber bei den anderen hier vorgestellten Netztopologien. Es sei angemerkt, daß in der Originalliteratur TDNNs für jede Gewichtsschicht ein unterschiedliches maximales Delay besitzen.

2.5 Gradientenabstieg in externen Räumen

Nach der Herleitung von Gradientenabstieg in Gewichtsräumen allgemeiner rekurrenter Netze und nach der Betrachtung von „typischen“ Netzarchitekturen werden wir dieses Kapitel mit der Beschreibung von zwei weiteren Suchalgorithmen basierend auf Gradientenabstieg abschließen.

Backpropagation ist ein Beispiel für einen Suchalgorithmus im Gewichtsraum (und Schwellwertsraum) eines Künstlichen Neuronalen Netzes. Dieser Suchraum ist ein interner Suchraum, da es sich um netzinterne Parameter handelt. Für die in den nächsten Kapiteln diskutierten Kontrollalgorithmen ist die Erweiterung von Gradientenabstieg auf externe Räume von Bedeutung. Externe Räume sind zum Beispiel die Initialaktivationen des Netzes oder die Eingabe. Gradientenabstieg zur Suche in externen Räumen ist völlig analog zu BPTT oder BPFF. Wir werden kurz zwei Algorithmen vorstellen, die Initialaktivationen und Eingabe adaptieren. Dabei stellen wir hier nur die zu BPFF duale Version vor, d.h., in den beiden Algorithmen werden gesonderte Hilfsgradienten vorwärts in der Zeit propagiert, und eine Rückwärtsphase existiert nicht.

2.5.1 Adaption der Initialaktivationen

Die Initialaktivationen $\tilde{x}_i$ wurden bereits in Kapitel 2.1.3 vorgestellt. Zur Wiederholung: Diese Aktivationen initialisieren in rekurrenten Netzen die Aktivationswerte zum Zeitpunkt t_0 und beeinflussen somit die spätere Berechnung. Initialaktivationen $\tilde{x}_i$ können, analog zu Gewichten, mit Gradientenabstieg adaptiert werden, um den Fehler zu minimieren. Dies kann in der Regel initialisiere w_{ijn}, $\tilde{x}_i$ zufällig $\forall i, j \in \mathcal{N},\ n \in \mathcal{S}$

wiederhole
  $\Delta w_{ijn} := 0\ \ \forall i, j \in \mathcal{N},\ n \in \mathcal{S}$;
  $\Delta \tilde{x}_i := 0\ \ \forall i \in \mathcal{N}$;

  für alle $p \in \mathcal{P}$ iteriere
    $E := 0$;
    $x_i(t_0) := \tilde{x}_i\ \ \forall i \in \mathcal{N}$;
    $\beta^k_{ijn}(t_0) := 0\ \ \forall i, j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$;
    $\gamma^k_i(t_0) := \delta_{ik}\ \ \forall i \in \mathcal{N},\ k \in \mathcal{O}$;

    für $t := t_0$ bis t_1 mit Inkrement 1 iteriere
      berechne $I^p_i(t)\ \ \forall i \in \mathcal{N}$;
      berechne $net_i(t)$ und $x_i(t)\ \ \forall i \in \mathcal{N}$; mit (2.2) und (2.3)
      berechne $e^p_k(t)\ \ \forall k \in \mathcal{O}$;
      berechne $\beta^k_{ijn}(t)\ \ \forall i, j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$; mit (2.19)
      berechne $\gamma^k_i(t)\ \ \forall i \in \mathcal{N},\ k \in \mathcal{O}$;
      $E := E + E_t$;
      $\Delta \tilde{x}_i := \Delta \tilde{x}_i - \eta_E \sum_{k \in \mathcal{O}} e^p_k(t) \cdot \gamma^k_i(t)\ \ \forall i \in \mathcal{N}$;
      $\Delta w_{ijn} := \Delta w_{ijn} - \eta_E \sum_{k \in \mathcal{O}} e^p_k(t) \cdot \beta^k_{ijn}(t)$
        $\forall i, j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$;

      wenn online-learning dann
        $w_{ijn} := w_{ijn} + \Delta w_{ijn}\ \ \forall i, j \in \mathcal{N},\ n \in \mathcal{S}$;
        $\Delta w_{ijn} := 0\ \ \forall i, j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$;
      Ende

      wenn teacher-forcing dann
        $x_k(t) = x_k(t) - e^p_k(t)\ \ \forall k \in \mathcal{O}$; wenn (2.12) gilt
      Ende
    Ende Schleife t
  Ende Schleife p

  wenn nicht online-learning dann
    $w_{ijn} := w_{ijn} + \Delta w_{ijn}\ \ \forall i, j \in \mathcal{N},\ k \in \mathcal{O},\ n \in \mathcal{S}$;
  Ende

  $\tilde{x}_i = \tilde{x}_i + \Delta \tilde{x}_i\ \ \forall i \in \mathcal{N}$;
bis $E < \varepsilon$

Abbildung 2.11: BPFF: Der erweiterte feed-forward Backpropagation Algorithmus. Diese Version ist um a) Online-Learning, b) Teacher forcing und c) Adaption der Initialaktivationen erweitert. Suche im Eingaberaum ist nicht eingeschlossen, da dies zu Konflikten mit dem Lernen im Netz führt.

unerwünschte Initialkonfigurationen verhindern. Dazu beginnt man den Gradientenabstieg mit einer initialen Schätzung für $\tilde{x}_i$ (zum Beispiel per Zufallswerten). Die Änderung ergibt sich $\forall i \in \mathcal{N}$ als

$$\begin{aligned}\Delta\tilde{x}_i &= -\eta\frac{\partial E}{\partial \tilde{x}_i} = -\eta\frac{\partial}{\partial \tilde{x}_i}\sum_{t=t_0}^{t_1}\sum_{k\in\mathcal{O}}\zeta_k(t) \\ &= -\eta\sum_{t=t_0}^{t_1}\sum_{k\in\mathcal{O}} e_k(t)\frac{\partial x_k(t)}{\partial \tilde{x}_i} = -\eta\sum_{t=t_0}^{t_1}\sum_{k\in\mathcal{O}} e_k(t)\gamma_i^k(t) \end{aligned}\tag{2.27}$$

Definiert man die Hilfsgradienten γ_i^k ($\forall i, k \in \mathcal{N}$) als

$$\gamma_i^k(t) \equiv \frac{\partial x_k(t)}{\partial \tilde{x}_i} = \frac{\partial x_k(t)}{\partial x_i(t_0)},\tag{2.28}$$

dann gilt

$$\begin{aligned}\gamma_i^k(t_0) &= \delta_{ik} = \begin{cases} 1, & \text{wenn } i = k \\ 0, & \text{sonst}\end{cases} \\ \gamma_i^k(t) &= \frac{\partial x_k(t)}{\partial \tilde{x}_i} = \frac{\partial}{\partial \tilde{x}_i}\sigma\left(\sum_{l\in\mathcal{N}}\sum_{m=0}^{\min\{t-t_0,\hat{n}\}} w_{klm}x_l(t-m)\right) \\ &= \sigma'(net_k(t))\left(\sum_{l\in\mathcal{N}}\sum_{m=0}^{\min\{t-t_0,\hat{n}\}} w_{klm}\gamma_i^l(t-m)\right) \\ &\quad \forall\ t \in \{t_0+1, t_0+2, ..., t_1\}\end{aligned}\tag{2.29}$$

Die Hilfsgradienten γ_i^k werden völlig analog zu den Hilfsgradienten β_{ijn}^k vorwärts propagiert. Gleichungen (2.27) und (2.29) definieren einen feed-forward Gradientenabstiegsalgorithmus für die Initialaktivationen $\tilde{x}_i$.

Abbildung 2.11 skizziert den Algorithmus BPFF (vergleiche Abbildung 2.5) mit allen bisher vorgestellten Erweiterungen: a) Online-Learning, b) Teacher forcing und c) Adaption der Initialaktivationen.

2.5.2 Suche im Eingaberaum

Völlig analog kann man eine Gradientenabstiegssuche im Eingaberaum eines Netzes durchführen. Man verwendet diese Technik zum Beispiel zum Generieren von Eingabevektoren, die eine bestimmte Ausgabe haben sollen.[4] Anwendungen finden sich zum Beispiel in Finden einer Pseudoinversen

[4]Logischerweise dient Suche im Eingaberaum im allgemeinen nicht dazu, den Fehler auf der Trainingsmenge E zu minimieren.

eines trainierten Netzes, also einer Antwort auf die Frage *„Welche Eingabe generiert eine gegebene Ausgabe?“* Die Relevanz dieser Frage leuchtet ein, wenn man ein Künstliches Neuronales Netz zur Klassifikation verwendet. Gradientenabstieg im Eingaberaum kann man zum Beispiel zur Suche von Mißklassifikationen verwenden. Eine zweite Anwendung der Suche im Eingaberaum wird in Kapitel 3.2.3 beschrieben.

Im folgenden werden wir das Gradientenabstiegsverfahren beschreiben. $I_i(t)$ bezeichnet die externe Eingabe des Knotens $v_i \in \mathcal{N}$ zum Zeitpunkt $t \in \{t_0, t_0+1, ..., t_1\}$, und $d_i(t)$ bezeichnet die gewünschte Ausgabe des Ausgabeknotens $k \in \mathcal{O}$ zur Zeit t.

$$E_{\text{Eingabe}} = \frac{1}{2} \sum_{t=t_0}^{t_1} \sum_{k \in \mathcal{O}} (d_k(t) - x_k(t))^2 \tag{2.30}$$

sei die quadratische Abweichung der gewünschten und der tatsächlichen Ausgabeaktivationen. Für die Gradientensuche benötigen wir eine initiale externe Eingabe $I_i(t)$, die zum Beispiel zufällig generiert werden kann. Diese wird dann schrittweise durch die folgenden Gleichungen geändert: $\forall s \in \{t_0, t_0+1, ..., t_1\}$ und $\forall i$

$$\Delta I_i(s) = -\eta \frac{\partial E_{\text{Eingabe}}}{\partial I_i(s)} = -\eta \sum_{t=s}^{t_1} \sum_{k \in \mathcal{O}} e_k(t) \frac{\partial x_k(t)}{\partial I_i(s)} \tag{2.31}$$

Definiert man die Hilfsgradienten $\xi_{is}^k(t)$ als

$$\xi_{is}^k(t) \equiv \frac{\partial x_k(t)}{\partial I_i(s)} = \sigma'(net_k(t)) \left(\sum_{l \in \mathcal{N}} \sum_{m=0}^{\min\{t-s, \hat{n}\}} w_{klm} \frac{\partial x_l(t-m)}{\partial I_i(s)} \right) + \underbrace{\frac{\partial I_k(t)}{\partial I_i(s)}}_{=\delta_{ik} \cdot \delta_{st}}, \tag{2.32}$$

dann gilt folgende Propagationsregel für ξ_{is}^k:

$$\begin{aligned}
\xi_{is}^k(s) &= \sigma'(net_k(s)) \left(\sum_{l \in \mathcal{N}} w_{kl0} \xi_{is}^l(s) \right) + \delta_{ik} \\
&\overset{\text{wenn } 0 \notin \mathcal{S}}{=} \quad \delta_{ik} = \begin{cases} 1, & \text{wenn } i = k \\ 0, & \text{sonst} \end{cases} \\
\xi_{is}^k(t) &= \frac{\partial x_k(t)}{\partial I_i(s)} = \sigma'(net_k(t)) \left(\sum_{l \in \mathcal{N}} \sum_{m=0}^{\min\{t-s, \hat{n}\}} w_{klm} \xi_{is}^l(t-m) \right) \\
& \forall \; t \in \{s+1, s+2, ..., t_1\}
\end{aligned} \tag{2.33}$$

Die Hilfsgradienten $\xi_{is}^k(t)$ werden zusammen mit den Aktivationen vorwärtspropagiert. In dieser Adaptionsregel sind Werte für die externe Eingabe unbeschränkt. Soll die Eingabe in einem Hyperwürfel liegen, bietet sich an, die Eingabewerte nachträglich zu beschränken.

$$\begin{aligned}
I_i(s) &\longleftarrow \max\{I_i(s), M_U\} \\
I_i(s) &\longleftarrow \min\{I_i(s), M_O\}
\end{aligned} \tag{2.34}$$

Hierbei bezeichnen M_U die untere und M_O die obere Schranke für Eingabewerte.

Damit wollen wir die Beschreibung der Gradientenabstiegsverfahren in Künstlichen Neuronalen Netzen abschließen. Literatur zur Anwendung dieses Verfahrens findet sich im Literaturverzeichnis.

2.6 Literatur

Wie bereits im Literaturverzeichnis zum ersten Kapitel bemerkt, geht der Name „Backpropagation" auf [Rumelhart *et al.*, 1986] zurück. Die Verallgemeinerung von Backpropagation auf rekurrente Netze geschah historisch unmittelbar nach Erscheinen der PDP-Bände [Rumelhart *et al.*, 1986]. Erste Arbeiten von Fernando Pineda [Pineda, 1987], [Pineda, 1989] und Luis Almeida [Almeida, 1988] leiteten Gradientenabstieg für im Aktivationsraum konvergente rekurrente Netze her. Den hier vorgestellten Algorithmus BPTT (Backpropagation through time) findet man zum Beispiel in [Mozer, 1988] und [Werbos, 1990a]. Da es sich hierbei um die direkte Erweiterung von Backpropagation für rekurrente Netze handelt, ist er von vielen Autoren unabhängig voneinander vorgeschlagen worden. BPFF, feed-forward Backpropagation, taucht vermutlich erstmals in den Werken von Robinson/Fallside [Robinson, 1989] und Williams/Zipser [Williams und Zipser, 1989] auf. Mike Mozer [Mozer, 1988], Rich Rohwer/Forrest [Rohwer und Forrest, 1987] und Sato [Sato, 1990] haben ebenfalls unterschiedliche Gradientenabstiegsalgorithmen beschrieben, die zum Teil nur für besondere Netzarchitekturen anwendbar sind. Jürgen Schmidhuber stellte einen allgemeinen rekurrenten Algorithmus vor, der rein vorwärts in der Zeit arbeitet, aber zeitlich weniger komplex als der hier vorgestellte Algorithmus BPFF ist [Schmidhuber, 1992]. Unabhängig voneinander erweiterten dann Barak Pearlmutter [Pearlmutter, 1989], [Pearlmutter, 1990] und Mike Gherrity [Gherrity, 1989] Gradientenabstieg auf *continuous networks* (Netze mit in der Zeit stetigen Aktivationsfunktionen). Andere Versionen von Gradientenabstieg für zeit-stetige Netze finden sich in [Doya und Yoshizawa, 1989] und [Doya und Shuji, 1989]. Die in Kapitel 2.4 vorgestellten Architekturen für rekurrente Netze gehen im Falle der Jordan-Netze auf Mike Jordan [Jordan, 1986] und im Falle der Elman-Netze auf Jeff Elman [Elman, 1988] zurück. Time Delay Neural Networks, ursprünglich von Kevin Lang und Alex Waibel entworfen [Lang und Waibel, 1990] [Waibel *et al.*, 1988], sind von Alex Waibel und seiner Gruppe intensiv im Bereich Sprachverarbeitung untersucht worden [Waibel und Lee, 1990]. Die diesem Text zugrundeliegende graphentheoretische Beschreibung Neuronaler Netze wurde ursprünglich in [Thrun und Śmieja, 1990] publiziert. In diesem Artikel findet sich auch eine komplette Beschreibung zeitlich stetiger Netze mit den dazugehörigen Differentialgleichungen.

In der Literatur der letzten Jahre finden sich eine Vielzahl von Aktivations- und Squashingfunktionen. Radial-Basis-Funktionen sind in Moody/Darkens Arbeiten [Moody und Darken, 1989], wie auch in Nowlans Arbeiten [Nowlan, 1990a] zu finden. Als weiteres in diesem Text allerdings nicht vorgestelltes Beispiel seien die Sigma-Pi Aktivationsfunktionen vermerkt [Rumelhart *et al.*, 1986] [Mel, 1990], die insbesondere in der biologischen Neuroforschung Anklang gefunden haben, da sie das starr *summarische* Eingabeschema für Knoten um *Multiplikationen* erweitern.

Teacher forcing, wie in Kapitel 2.3.2 vorgestellt, ist unter anderem in [Mozer, 1988], [Williams und Zipser, 1989], [Pearlmutter, 1989] und [Doya und Yoshizawa, 1989] zu finden. Die in Kapitel 2.5.2 vorgestellten Suchtechniken zur Suche in Eingaberäumen gehen ursprünglich auf Ideen von Williams [Williams, 1986] zurück. Unabhängig davon widmeten sich Alexander Linden/Jörg Kindermann diesem Thema im Kontext von Mustererkennung [Linden und Kindermann, 1989] [Kindermann und Linden, 1990]. Linden/Kindermanns Technik, der sie den Namen „Inversion" gaben, wurde später auf rekurrente Netze erweitert [Thrun und Linden, 1990], [Thrun *et al.*, 1990]. Eine gute deutschsprachige Übersicht der Inversionstechnik ist in Alexander Lindens Diplomarbeit [Linden, 1990] zu finden.

Seit Mitte der Achtziger Jahre hat der Boom im Bereich Künstlicher Neuronaler Netze zu einer Flut von Modifikationen und Erweiterungen des ursprünglichen Backpropagationalgorithmus geführt. Wir werden im folgenden wesentliche Forschungsrichtungen skizzieren, und dabei nur wenige repräsentative Arbeiten zitieren.

Eine der offensichtlichen Schwächen von Backpropagation ist die Lerngeschwindigkeit des Algorithmus. Wir haben in Kapitel 1 bereits das auf [Rumelhart *et al.*, 1986] zurückgehende Momentum vorgestellt, das das Lernen erheblich beschleunigen kann. In [Jacobs, 1988] und [Fahlman, 1988] finden sich ähnliche Techniken, die die Schrittweite bei Gewichtsänderungen zu optimieren suchen. Moller stellt ein Verfahren vor, das sukkzessive Gradienten verbindet, um die Konvergenzgeschwindigkeit zu erhöhen [Moller, 1990]. Während sich all diese Verfahren in der Praxis als vorteilhaft erwiesen haben, fehlt ein theoretischer Beweis ihrer Wirksamkeit meist völlig. Adaptive Schrittweiten bringen allerdings den angenehmen Vorteil mit sich, daß der Benutzer die Schrittweite nicht mehr von Hand vorgeben muß. Yann LeCun arbeitet derzeit an einem quasi-optimalen Verfahren der Schrittweitenbestimmung, das auf der Schätzung der Hessischen Matrix Neuronaler Netze beruht.

Andere Forscher haben sich mit der Struktur Neuronaler Netze beschäftigt. In dieser Arbeit wird stets angenommen, daß der Benutzer die Topologie des Netzes selbst geeignet festlegt. Zweifelsohne wirkt sich die Größe eines Netzes auf die Generalisierung und die Lernbarkeit von Trainingsmengen aus. In [Mozer und Smolensky, 1989], [Chauvin, 1989], [Fahlman und Lebiere, 1990] und [Fahlman, 1989] sind eine Vielzahl von Algorithmen zur Automatisierung der Architekturbestimmung beschrieben. Dabei starten solche Algorithmen in der Regel erst mit kleinen Netzen, die im Training wachsen, oder mit großen Netzen, die schrumpfen. Eine zunehmend beliebter werdende Technik zum Einschränken der Freiheitsgrade eines Netzes ist das in Kapitel 4.4 beschriebene *weight decay* [Chauvin, 1989]. Hier werden Gewichte künstlich klein gehalten, was in vielen Untersuchungen einen positiven Effekt auf die Generalisierung hat [Chauvin, 1989] [Thrun *et al.*, 1991a]. Nowlan erweiterte dieses Schema auf eine Menge von konkurrierenden Gewichtsbeschränkungen [Nowlan und Hinton, 1992].

Ein andere Trend ist die Modularisierung von Netzen. Das in die Informatik bekannte Prinzip *„divide and conquer"* (*„divide et impera"* in Latein und zu deutsch: *„teile und herrsche"*) wird derzeit viel im Bereich Künstlicher Neuronaler Netze untersucht. Dabei werden Funktionen von mehreren Netzmoduln approximiert, die sich sozusagen die Arbeit aufteilen [Waibel, 1989], [Waibel *et al.*, 1988], [Jacobs und Jordan, 1991a], [Nowlan, 1990b], [Nowlan, 1991], [Fox *et al.*, 1991], [Śmieja, 1991] und [Jordan und Jacobs, 1992]. Es ist nach wie vor eine offene Frage, inwieweit solche Verfahren geeignet sind, Funktionen sinnvoll und effizient zu partitionieren. Simard/Victorri/LeCun/Denker [Simard *et al.*, 1992] sowie Mitchell/Thrun [Mitchell und Thrun, 1993] arbeiten an Verfahren, mit Backpropagationnetzen nicht nur Trainingspunkte, sondern auch *Trainingsgradienten* zu approximieren. Integrationen von symbolischen (AI) und neuronalen Lerntechniken finden sich in [McMillan und Smolensky, 1988], [Shavlik und Towell, 1989b] [Shavlik und Towell, 1989a], [Mahoney und Mooney, 1992] und [Mitchell und Thrun, 1993], [Thrun und Mitchell, 1992]. Vergleiche von symbolischen und neuronalen Algorithmen findet man in [Shavlik *et al.*, 1989] und [Thrun und Mitchell, 1992].

Auch die theoretische Forschung hat im Bereich Neuronaler Netze nicht haltgemacht. Ausgehend von der Annahme, Knoten seien lineare Separatoren (binäre Schwellwertknoten), haben u.a. Avrim Blum und David Haussler Valiants PAC-Learning Modell auf Neuronale Netze übertragen. Dem an der Frage „Wie viele Trainingsbeispiele benötigt ein Netz, damit es wie generalisiert?" interessierten Leser sei die Lektüre von [Valiant, 1984], [Blum, 1989], [Baum, 1988] [Baum und Haussler, 1989] und [Haussler, 1990] empfohlen. Darüber hinaus haben Stephen Judd, Avrim Blum und Ron Rivest gezeigt, daß das generelle Finden von Gewichten in Künstlichen Neuronalen Netzen NP-vollständig sein kann [Judd, 1988], [Blum und Rivest, 1992].

Für den statistisch interessierten Leser sei der exzellente Artikel von Geman/Bienenstock/Doursat empfohlen [Geman *et al.*, 1992]. Dieser Artikel betrachtet Lernen in Neuronale Netze vom Standpunkt eines Statistikers und leitet gleichzeitig in grundlegende statistische Methoden ein.

Diese kurze Übersicht erhebt wegen der Fülle der Literatur in keiner Form irgendeinen Anspruch auf Vollständigkeit. Sie soll vielmehr Anhaltspunkte vermitteln, was aktuelle Forschungsrichtungen sind und wo nach weiterer Literatur gesucht werden kann. Wir empfehlen dem interessierten Leser auch das Durchblättern von Zeitschriften und Konferenzbänden. So sind zum Beispiel die Publikationen der jährlich in Denver (Kolorado) abgehaltenen Konferenz *„Neural Information Processing Systems" (NIPS)* exzellent geeignet, um sich ein Bild vom Stand der Forschung und aktuellen Forschungstrends zu machen.

Kapitel 3

Grundschemata in Neurocontrol

Nach der Einführung von Künstlichen Neuronalen Netzen in den beiden ersten Kapiteln wenden wir uns nun der Anwendung solcher Systeme im Bereich Adaptive Neurokontrolle[1] (oder kurz: *Adaptive Kontrolle*) zu. Unter diesem Oberbegriff verbirgt sich folgende Problemstellung: Ein lernendes System, oft als Adaptiver Kontroller oder Agent bezeichnet, interagiert mit einem dynamischen System. Anfangs ist diese Dynamik dieses Systems unbekannt. So kann zum Beispiel ein Roboter sich in einem unbekannten Gebäude befinden, oder ein automatisches Flugsystem steuert ein ihm unbekanntes Flugzeug. Das dynamische System, oft mit *world*, *plant* oder *environment* und im folgenden mit Welt bezeichnet, befindet sich zu jedem Zeitpunkt in einem Zustand (Konfiguration). Ein Zustand im Roboterszenario kann zum Beispiel die Position des Roboters und der Hindernisse sein, oder die Höhe, Geschwindigkeit, Winkel etc. eines Flugzeuges im Flugzeugszenario. Der Kontroller kann nun den Zustand der Welt mit seinen Sensoren messen. Des weiteren kann er mit Aktionen den Zustand der Welt beeinflussen. Da die Welt zu Lernbeginn dem Lerner unbekannt ist, kann er die Regeln der Welt nur per Ausprobieren von Aktionen lernen. Das eigentliche Lernproblem in Neurocontrol ist allerdings, in seiner allgemeinsten Formulierung, die Welt in einen Zielzustand (*goal state*, *desired state*) zu überführen und/oder zu halten. Dabei ist die Menge der Zielzustände bekannt, nicht aber die Folge von Aktionen, die diese herbeiführt. Man sucht daher nach einem *optimalen Kontroller*, der Zustände in eine Aktion bzw. eine Folge von Aktionen abzubilden vermag, die in möglichst wenig Schritten die Welt in einen Zielzustand überführen.

Dieses Problem ist in Abbildung 3.1, einem sogenannten „Mitchogramm“, graphisch dargestellt: Die Welt ist eine rekursive Zustandsübergangsfunktion f, die Zustände s und Aktionen a in Folgezustände s' überführt. Um die Welt je nach Definition in einen oder eine Menge von gewünschten Zielzuständen zu überführen oder zu bewahren, sucht man eine Funktion g, die den aktuellen Zustand in eine Aktion oder eine Aktionssequenz abbildet. Zur Vereinfachung der

[1] Dies ist eine vage Übersetzung des Englischen Begriffs *Neurocontrol*, hinter dem sich Prinzipien der *adaptive control*, *repetitive control* und *optimal control* mit Neuronalen Netzen verbergen. Neuerdings werden Forschungsarbeiten in diesem Bereich auch unter dem Begriff *intelligent control* zusammengefaßt.

Abbildung 3.1: Die *Welt* f ist eine Funktion, die Aktionen und Zustände in Folgezustände überführt. Der Kontroller ist eine Funktion g, die Zustände in Aktionen abbildet mit dem Ziel, das *Reward* R zu maximieren. Neurocontrol beschäftigt sich mit der Suche nach einem g realisierenden *Kontrollnetz*, das das Reward maximiert.

Notation vereinbaren wir, daß der Kontroller über eine Bewertungsfunktion verfügt, die wir als Reward-Funktion[2] bezeichnen werden. Diese Funktion *bewertet* Zustände, wobei das Reward widerspiegelt, wie gut (wünschenswert) ein Zustand ist. Ziel des Lerners ist also, das Reward zu maximieren. Des weiteren vereinbaren wir, daß die Sensoren den Zustand der Welt zu jedem Zeitpunkt akkurat messen können. Ist dies nicht der Fall, ist das Lernproblem erheblich schwieriger.

Ein klassisches und vielzitiertes Problem dieser Art ist das Pole Balancing-Problem (Abbildung 3.2): Auf einem Wagen ist ein Stab befestigt, den es zu balancieren gilt, indem man auf den Wagen eine geeignete Kraft (=Aktionen) ausübt. Das Problem ist nun, daß man ohne explizite Kenntnis der Dynamik des Stabes Sequenzen von Aktionen, d.h. Kräften, generieren soll, die den Stab möglichst lange in der Balance, dem wünschenswerten Zustand, halten. Im Pole Balancing-Problem kann der Zustand z.B. durch die horizontale Position des Wagens x, seine Geschwindigkeit $\dot{x}$, den Winkel des Stabes zum Wagen θ und dessen Winkelgeschwindigkeit $\dot{\theta}$ beschrieben werden. Im einfachsten Fall (*bang-bang control*) besteht der Aktionsraum lediglich aus zwei Aktionen: Ausüben einer Kraft festen Betrages von links (a_1) und von rechts (a_2). Definiert man nun die Reward-Funktion als 1, wenn der Stab den Boden nicht berührt, und 0, wenn er umgefallen ist, dann ist das Kontrollproblem ein Optimierungsproblem, das das Reward zu maximieren sucht.

Adaptive Kontroll-Probleme sind mannigfaltiger Natur – das Pole Balancing-Problem dient nur zur Motivation (wir werden später auf dieses Beispiel zurückkommen). Insgesamt kennzeichnet die Klasse von Problemen, die wir hier betrachten wollen, folgende Eigenschaften:

- Es handelt sich um zeitliche Prozesse, d.h., es wird eine oft unendliche Folge von Aktionen gesucht, die entweder das System in einem Zielzustand halten oder in einen solchen überführen (je nach Problem).
- Die Funktion, die sich hinter der Welt verbirgt, ist nicht oder nur teilweise bekannt. Im

[2]Dieser in der Kontroll-Literatur weniger übliche Begriff ist gängig in der Reinforcement Learning Literatur.

Abbildung 3.2: Das Pole Balancing-Problem: Auf einen in horizontaler Richtung beweglichen Wagen ist über ein Gelenk ein Stab montiert, den es zu balancieren gilt. Aktionen sind z.B. das Ausüben einer Kraft festen Betrages F von links oder von rechts.

Gegensatz zu weiten Teilen der Kontroll-Literatur nehmen wir an, daß die Welt nichtlinear sein kann.

- Es gibt einen zu optimierenden Wert, das Reward R, das die Qualität des aktuellen Zustandes bewertet (z.B. die Abweichung des aktuellen Zustandes von dem gewünschten). Damit ist das Adaptive Kontroll-Problem ein Optimierungsproblem, bei dem dieses R maximiert wird.
- Durch Ausprobieren kann der Effekt von Aktionen auf das Reward R geschätzt werden. Der Kontroller soll von Beispielen generalisieren, um geeignete Aktionssequenzen zu finden. Das bedingt, daß Generalisierung möglich ist. Im allgemeinen sind die betrachteten Welten daher stückweise stetig, oft auch differenzierbar.
- Die beste Aktion ist i.a. nicht eindeutig.
- In den meisten in der Neuroliteratur diskutierten Anwendungen im Bereich Adaptive Kontrolle ist die Welt strikt deterministisch. Deterministische Welten sind relativ leicht modellierbar und ermöglichen so z.B. den Einsatz von Planen. Viele Resultate lassen sich auf stochastische Welten übertragen, wobei wir im Rahmen dieser Arbeit eine entscheidende Restriktion fordern: Die Welt f sei *konvex*, d.h., mit zwei möglichen Funktionswerten der Welt ist auch jeder Wert dazwischen ein gültiger Funktionswert. Solche Welten sind z.B. deterministische Welten, die mit geringem Rauschen (Meßfehlern etc.) überlagert sind, und so dem Beobachter stochastisch erscheinen. Das Modellieren beliebiger stochastischer Welten wird in dieser Arbeit nicht adressiert.

Wir werden in diesem Kapitel gängige Techniken zur Identifikation der Kontrollfunktion g vorstellen. Dabei unterscheiden wir zwischen

- <u>Einschrittkontrolle:</u> Bewertet das Reward R die Qualität von Aktionen separat, indem jedes Reward nur die unmittelbar vorangegangene Aktion evaluiert, spricht man von Einschrittkontrolle. Dies ist der weniger komplexe Fall, da zeitliche Abhängigkeiten zwischen Aktionen und Zuständen nicht berücksichtigt werden müssen.

- Mehrschrittkontrolle: Im allgemeinen Fall dagegen ist das Reward eine Funktion des Initialzustandes sowie *aller* Aktionen. Gute Aktionen wirken sich erst unbekannt viele Zeitschritte später auf R aus (*delayed reward*); R bewertet somit nicht mehr jede Aktionen separat, sondern ganze Aktionsfolgen. Man sieht leicht ein, daß dies die Suchkomplexität für optimale Aktionen erheblich erhöhen kann.

Die zweite in der Literatur gängige Unterscheidung bezieht sich auf die Realisierung von g. Generell unterscheidet man zwischen direkten Kontrollern und indirekten Kontrollern:

- Findet man eine Funktion g, die unter Kenntnis der bisherigen Zustände der Welt und einer Beschreibung der gewünschten Zielzustände *direkt* geeignete Aktionen generiert, bezeichnet man diese als direkten Kontroller. Solche Kontroller haben den Vorteil, daß sie relativ schnell und daher für Echtzeitanwendungen gut geeignet sind. Auf der anderen Seite ist das Finden eines solchen Kontrollers oft sehr aufwendig. Außerdem wird die Menge der möglichen optimalen Aktionen gewöhnlich auf eine einzige reduziert, nämlich die, die der Kontroller gerade erzeugt.

- Indirekte Kontroller dagegen werden durch eine schwächere Funktion realisiert, nämlich durch ein Modell, das unter Kenntnis von bisherigen Zuständen und der Zielfunktion den Wert einer Aktion evaluieren kann. Mit diesem Modell kann man optimale Aktionen zwar nicht direkt erzeugen, aber man kann in ihrem Eingaberaum die optimale Aktion suchen. Bei unendlichen Räumen wird dabei oft mit Techniken wie Gradientenabstieg gesucht, worauf wir uns im folgenden auch beschränken wollen.

Dieses Kapitel stellt die in der Neuroliteratur gängigsten Ansätze zur Identifikation eines Kontrollers g vor. Wir beginnen mit dem einfachsten Fall, in dem g mithilfe eines Lehrers gelernt wird, der die optimalen Aktionen kennt und vormacht. Da Lehrer in adaptiven Kontrollproblemen in der Regel nicht verfügbar sind, werden wir zwei direkte sowie eine indirekte Technik vorstellen, bei dem der Kontroller selbst die optimalen Aktionen sucht. Anschließend werden wir uns dem komplexeren Fall der Mehrschrittkontrolle widmen und zwei Algorithmen vorstellen, die in der Lage sind, solche Probleme zu lösen: Lookahead Planen und Reinforcement Learning.

3.1 Adaptive Kontroller mit Lehrer

In manchen Kontrollanwendungen ist ein externer Lehrer verfügbar, ein System, das die gewünschten optimalen Aktionen exemplarisch generieren kann. Beispiele für externe Lehrer sind z.B. Menschen (man stelle sich ein Problem vor, bei dem der Kontroller lernen soll, ein Auto zu steuern). Lehrer können aber auch durch Computer realisiert werden. So kann z.B. ein sehr aufwendiges, mit eine Vielzahl von Sensoren ausgestattetes Kontrollsystem durch ein adaptives, billigeres und möglicherweise schnelleres ersetzt werden, indem letzteres ersteres kopiert. Dieses wohl einfachste Kontrollszenario ist in Abbildung 3.3 skizziert.

Abbildung 3.3: Lernen mit Lehrer: Der externe Lehrer generiert Beispiele von optimalen Aktionen, die durch das Kontrollnetz approximiert werden.

Das Erlernen von g reduziert sich auf das Kopieren des verfügbaren Lehrers, zum Beispiel mit dem Backpropagationalgorithmus. Dabei agiert der Zustand als Eingabe eines Netzes, und die vom Lehrer generierte Aktion als gewünschte Ausgabe. Die Algorithmen aus Kapitel 1 und 2 sind somit direkt anwendbar, wobei sich die Trainingsmenge aus allen beim Lehrer beobachteten Zustand-Aktionstupel ergibt.

3.2 Einschrittkontrolle

Lernen mit Lehrer unterscheidet sich in einem Punkt wesentlich von Lernen ohne Lehrer. Während im ersten Szenario der Lerner für jeden beobachteten Zustand die beste Aktion mitgeteilt bekommt und somit diese direkt approximieren kann (*supervised learning*), ist im zweiten Fall nur das Reward verfügbar, das Aktionen post-facto evaluiert. Der Kontroller muß selbst herausfinden, welches optimale Aktionen sind (*self-supervised learning*). Die hier vorgestellten Methoden, direkte und indirekte, basieren im wesentlichen auf sukkzessives Ausprobieren und Vergleichen. Dabei beschränken wir uns vorerst auf Einschrittkontrolle, also Szenarios, wo das Reward direkt eine einzelne Aktion bewertet.

3.2.1 Direkte Kontrolle: Lernen der Inversen Funktion

Das Ziel von Optimaler Adaptiver Kontrolle ist die Identifikation einer Funktion g, die das Reward maximiert. Diese Funktion, die häufig als inverses Modell der Welt bezeichnet wird, kann man in besonderen Fällen einfach durch geeignete Definition von Eingabe und Ausgabe finden. Dies ist in Abbildung 3.4a illustriert: Nachdem der Kontroller im Zustand s eine beliebige, z.B. zufällig generierte Aktion a ausgeführt hat, wird der resultierende Rewardwert R zum Training eines

Abbildung 3.4: Adaptive Kontrolle: Lernen eines inversen Modells und direkte Kontrolle. (a) Das Netz lernt, für Zustände und Folgerewards die entsprechende Aktion auszugeben. (b) Ist das Netz trainiert, kann man es für optimale Kontrolle verwenden, indem man es Aktionen mit maximalem Reward generieren läßt.

---

inversen Modells benutzt. Dabei dient der Zustand s und das resultierende Reward R als Eingabe, und die zugehörige Aktion als Ausgabe.

Wie man sieht, ist diese ungewöhnliche Definition von Eingabe und Ausgabe nicht immer unproblematisch. Was passiert, wenn mehrere Aktionen in einem Zustand s zum gleichen Reward führen? Ist dies der Fall – und dies ist bei vielen Kontrollproblemen gegeben – so würde das bedeuten, daß die Trainingsmenge für g nicht eindeutig ist. Generell muß man für diese Technik also fordern, daß die "inverse Welt", die Abbildung von Zuständen und resultierenden Rewards auf Aktionen, eindeutig ist. In vielen Anwendungen kann man dies erreichen, indem man neben dem Reward auch den Nachfolgezustand als Eingabe verwendet. Darüber hinaus gibt es Techniken, mehrdeutige Funktionen in mehrere eindeutige aufzuteilen, und diese dann getrennt zu lernen.

Hat man ein inverses Modell g gelernt, kann dieses leicht zur Generierung einer optimalen Aktion verwendet werden. Dabei verwendet man einen höchstmöglichen Reward-wert (das Reward-Ziel) und den aktuellen Zustand als Eingabe, und g generiert die zugehörige Aktion, wie in Abbildung 3.5b gezeigt.

3.2.2 Lernen eines Modells der Welt

Ein beliebtes und allgemeineres Verfahren zur Generierung optimaler Aktionen ist *indirekte Kontrolle*. Dieses Verfahren basiert auf der Modellierung der Welt. Dabei versteht man unter Modellierung, daß ein adaptiver Mechanismus, hier ein Backpropagationnetz, die Zustandsübergangsfunktion der Welt f approximiert. Nach jeder Aktion wird das beobachtete Verhalten der Welt – der Ausgangszustand, die Aktion, der Folgezustand und das Reward – als Trainingsbeispiel für das Modellnetz verwendet. Dieses Netz lernt somit, Folgezustände und Rewards *vorherzusagen* (vgl. Abbildung 3.5).

Abbildung 3.5: (a) Systemidentifikation. Das Modellnetz modelliert die Welt f. (b) Suche im Eingaberaum des Modellnetzes ermöglicht das Finden optimaler Aktionen, optimal bezüglich des erwarteten Rewards.

3.2.3 Indirekte Kontrolle durch Suche im Aktionsraum

Das Modell kann nun verwendet werden, um optimale Aktionen zu finden. Generell beinhaltet dies Suche im Aktionsraum: Für den aktuellen Zustand sucht man diejenige Aktion, die das vorhergesagte Reward maximiert. Hierbei können im Prinzip beliebige Suchtechniken eingesetzt werden. Da Backpropagationnetze oft sehr glatte Approximationen liefern, bietet sich Gradientenabstieg als Suchverfahren an. Das spezielle Verfahren zur Suche im Eingaberaum eines Netzes wurde in Kapitel 2.5.2 beschrieben: Da das Modellnetz differenzierbar ist, kann man berechnen, wie man Aktionen ändern muß, um das vorhergesagte Reward zu vergrößern. Startet man mit einer beliebigen initialen Aktion, führt iterative Gradientenabstiegssuche zur lokal besten Aktion. Abbildung 3.5b veranschaulicht diesen Prozeß der Suche im Eingaberaum des Modellnetzes.

Im Gegensatz zu direkter Kontrolle kann indirekte Kontrolle auch dann verwendet werden, wenn die Welt nicht bijektiv ist, d.h., wenn das inverse Modell der Welt nicht eindeutig ist. Im Gegensatz zum direkten Ansatz sind indirekte Systeme wegen der erforderlichen Suche allerdings vergleichsweise langsam.

3.2.4 Learning with a Distal Teacher

Ein Modell kann nicht nur zur Suche im Aktionsraum, sondern auch zur Identifikation eines direkten Kontrollers verwendet werden. Das im folgende beschriebene Verfahren *„learning with a distal teacher“* verwendet das Modell zum Lernen einer inversen Funktion g: diese Funktion (Kontrollnetz) bildet Zustände auf Aktionen ab, und ist somit ein direkter Kontroller.

Abbildung 3.6: Das Modell kann benutzt werden, um einen direkten Kontroller zu trainieren. Dabei werden Gradienten (grau), die die Abweichung zwischen dem erwarteten und dem optimalen Reward messen, durch das Modellnetz propagiert, um dann im Kontrollnetz Gewichte und Schwellwerte zu optimieren.

Abbildung 3.6 skizziert das Verfahren: In Kapitel 3.2.2 und Abbildung 3.5 ist die Modellidentifikation bereits beschrieben; dieses Modell wird zum Training des Kontrollnetzes verwendet. Die grundlegende Idee ist dabei die Umwandlung der Reward-Optimierung in ein Trainingssignal für das Kontrollnetz (daher der Name „distal teacher“). Dazu läßt man das Kontrollnetz eine Aktion generieren, für die das Modellnetz das entsprechende Reward voraussagt. Dieses Reward wird durch die Adaption der Gewichte und Schwellwerte des Kontrollnetzes mit Backpropagation optimiert: Die quadratische Abweichung des optimalen Reward und des entsprechenden vorausgesagten Rewards sei J. Da das Modellnetz eine differenzierbare Funktion darstellt, kann man mit den in Kapitel 2.5.2 beschriebenen Techniken den Gradienten der Funktion J bezüglich der externen Eingabe des Modellnetzes, der Aktion, berechnen. Dieser Gradient mißt, wie die Aktion geändert werden muß, um das vorausgesagte Reward zu maximieren. Er dient damit als Basis für die Gewichts- und Schwellwertsänderungen im Kontrollnetz. Man kann sich diesen kompliziert anmutende Vorgang leicht veranschaulichen, wenn man sich das Kontroll- und das Modellnetz als ein einziges Netz vorstellt. In diesem Netz werden nur die Gewichte und Schwellwerte des Kontrollnetzes, nicht aber die des Modellnetzes adaptiert. „Learning with a distal teacher“ ist dann äquivalent mit Backpropagation, gibt man als gewünschte Ausgabe immer das maximale Reward, und adaptiert man nur eine Teilmenge aller Gewichte und Schwellwerte.

Dieser direkte Kontrollalgorithmus hat einige Vorzüge, die ihn für Anwendungen im Bereich Adaptive Kontrolle sehr interessant machen. So ist er in der Lage, optimale Aktionen selbst dann zu generieren, wenn das inverse Modell nicht eindeutig ist, also mehrere optimale Aktionen existieren. Das liegt daran, daß optimale Aktionen nicht direkt gelernt werden, sondern stattdessen Gradienten Grundlage des Trainings bilden. Diese Gradienten dienen dem iterativen Verbessern des Kontrollnetzes und ermöglichen somit, lokal optimale Aktionen zu finden. Diese Methode hat außerdem den Vorteil, daß sie eine optimale Kontrollfunktion g finden kann, selbst wenn das Modell nicht oder nur teilweise korrekt ist. Dieses nicht intuitive Resultat bedingt allerdings eine kleine algorithmische Modifikation: Anstelle des *vorausgesagten* Rewards ist das *beobachtete* Reward Grundlage für die Gradientenberechnung. Wenn nun die 1. Ableitung des Modells korrekte Vorzeichen hat (d.h. nicht, daß sie korrekt sein muß!), ermöglicht der resultierende Algorithmus das Finden eines optimalen Kontrollnetzes, selbst wenn das Modell falsche Voraussagen liefert. Obwohl in der Praxis die Bedeutung der Annahme richtiger Vorzeichen für die 1. Ableitung des Modells unklar ist, ist das Potential, selbst mit einem ungenauen Modell optimale Aktionen zu lernen, bemerkenswert.

3.3 Reinforcement Learning und das Temporal Credit Assignment Problem

Die bisher beschriebenen Kontrollverfahren bezogen sich alle auf Einschrittkontrolle, d.h. auf Anwendungen, in denen das Reward direkt die Güte der gewählten Aktion mißt. Im allgemeinen Fall aber kann das Reward *verzögert* sein; eine Aktion kann ihre Auswirkung möglicherweise erst einige Schritte später entfalten. Dies ist zweifelsohne wahr für das Pole Balancing-Problem: Es gibt Aktionen, die das sichere Umfallen des Stabes verursachen, obwohl direkt nach solch einer Aktion der Stab noch in der Balance zu sein scheint. Im Pole Balancing-Problem kann die zeitliche Verzögerung leicht 50 Zeitschritte betragen. Das allgemeine Problem zeitlicher Verzögerung des Rewards wird als temporal credit assignment problem bezeichnet.

Wir werden im folgenden zwei algorithmische Prinzipien vorstellen, die solche Probleme lösen können. Diese Algorithmen – Lookahead Planen und Reinforcement Learning – sind im wesentlichen Erweiterungen des bereits vorgestellten indirekten Kontrollschemas.

3.3.1 Lookahead Planen

In indirekter modellbasierter Einschrittkontrolle (siehe Kapitel 3.2.3) wird das Reward des unmittelbar nächsten Zeitschritts vorausgesagt und optimiert. Diese Technik ist bereits eine Form von Lookahead Planen mit einem Schritt Vorausschau. N-Schritt Lookahead Planen erweitert dieses Kontrollschema auf die Voraussage und Optimierung der nächsten N Rewards. Dazu nehme man an, daß der Zeitraum, in dem sich eine Aktion auf das Reward auswirken kann, durch eine ganzzahlige und im Voraus bekannte Konstante $N \geq 1$ beschränkt ist; ist dies nicht der Fall, kann Lookahead Planen zwar keine optimalen Aktionen, oft allerdings immer noch hinreichend gute Aktionen finden.

Lookahead Planen mit Gradientenabstieg ist ein iterativer Prozeß. Dabei startet man mit einem initialen Plan der Länge N, einer Folge von N Aktionen. Dieser initiale Plan kann aus beliebigen

Abbildung 3.7: Lookahead Planen. Das Modellnetz wird benutzt, um für einen N Schritte langen Plan das Reward vorherzusagen. Dabei dient der jeweilige vorhergesagte Zustand als Eingabe für den nächsten Schritt. Suche im Aktionsraum ermöglicht die Maximierung des vorhergesagten Rewards. Schließlich wird die erste Aktion des Plans ausgeführt.

Aktionen, zum Beispiel zufällig erzeugten Aktionen, bestehen. Durch iterative Anwendung des Modellnetzes können die nächsten N Zustände und Rewards für beliebige Pläne voraussagt werden: Beginnend mit dem aktuellen Zustand und der ersten Aktion im Plan ermöglicht das Modellnetz, den resultierenden Folgezustand vorauszusagen. Dieser wird zusammen mit der zweiten Aktion im Plan zur Voraussage des nächsten Folgezustandes verwendet, und so weiter. Das Mitchogramm in Abbildung 3.7 skizziert Lookahead Planen. Da das Modellnetz, und somit auch die Kette von N Modellnetzen, differenzierbar ist, kann man völlig analog zur indirekten Einschrittkontrolle die Gradienten der Rewardfunktion bezüglich der Parameter des Planes berechnen. Mit diesen Gradienten kann man den Plan modifizieren, um so das Reward der nächsten N Zeitschritte zu maximieren. Iterative Anwendung dieser Gradientensuche optimiert den Plan bezüglich des Rewards.

Planen ist in den letzten 35 Jahren im Rahmen des Forschungsgebietes „Künstliche Intelligenz“

(KI) intensiv untersucht worden. Es sei vermerkt, daß sich der hier präsentierte Ansatz von den meisten in der KI bekannt gewordenen Ansätze u.a. darin unterscheidet, daß die verwendeten Modelle approximativ und daher gewöhnlich nicht korrekt sind. Dies kann Schwierigkeiten mit sich bringen, die Planungsalgorithmen basierend auf korrekten Modellen nicht haben. Wie bereits erwähnt, sind alle Zustände während des Planungsvorgangs hypothetisch, da vom Modell vorhergesagt. Schlechte, d.h. ungenaue Modelle generieren ungenaue Voraussagen, die in der Regel zu noch ungenaueren Folgevoraussagen führen. Voraussagefehler sind daher kumulativ, und gute Planungsresultate hängen entscheidend von der Qualität des Modellnetzes ab.

Unter bestimmten Bedingungen kann Planen mit ungenauen Modellen zu noch extremeren Voraussagefehlern führen. Ist zum Beispiel der Zustandsraum durch nicht bekannte *Constraints* beschränkt (mathematisch gesprochen: Ist der Raum der gültigen Zustände eine Untermannigfaltigkeit des durch den Zustandsvektor aufgespannten Hyperwürfels), kann die Verwendung von approximativen Modellen zu Voraussagen von nicht definierten Zuständen führen. Für solche Zustände ist das Modell aber nie trainiert worden. Somit ist die Voraussage des Modells gewissermaßen dem Zufall überlassen, und Planen kann in die Irre führen. Ein Beispiel hierfür sei das folgende: Der Zustand eines Pendels in einem zweidimensionalem Raum wird durch die x- und y-Abweichung vom Ruhepunkt angegeben. Obwohl der durch x und y repräsentierbare Raum zweidimensional ist, hat das Pendel nur einen Freiheitsgrad. Die Schnur des Pendels ist ein solches *Constraint*, das x und y in eine gewisse Relation setzt, und das dem Lerner nicht bekannt sein mag. Das Modell wird also auch für physikalisch nicht mögliche Zustände Voraussagen liefern. Benutzt man Planungstechniken zur optimalen Kontrolle des Pendels, kann es passieren, daß der vorausgesagte Zustand kein physikalisch definierter Zustand ist. Jedes Planungsergebnis, das auf diesen Zustand beruht, ist daher fragwürdig.

Die Anwendbarkeit von Planen mit unkorrekten Weltmodellen ist ein offenes Thema aktiver Forschung. Das zuletzt geschilderte Problem der Constraints im Zustandsraum hat glücklicherweise eine zumindest teilweise befriedigende Lösung. Typischerweise werden in solchen Fällen Modelle mit geringem Rauschen (*noise*) trainiert, d.h., der Zustandsbeschreibung wird eine Zufallsvariable mit Erwartungswert 0 und geringer Streuung beigemischt. Somit ist das Modell in der ganzen Umgebung der Untermannigfaltigkeit trainiert, und iterative Voraussagen fallen mit großer Wahrscheinlichkeit in diese Umgebung.

3.3.2 Reinforcement Learning

Für Kontrollprobleme mit verzögertem Reward erfreuen sich Techniken bekannt als *„Reinforcement Learning“* der derzeit wohl größten Beliebtheit in der Neuroliteratur. Die Grundidee von Reinforcement Learning, insbesondere *„AHC-Learning“* und *„Q-Learning“*, ist das Erlernen einer Bewertungsfunktion (*evaluation function, utility function*), die für jeden Zustand (im Falle von AHC-Learning) oder für jedes Zustand-Aktionstupel (im Falle von Q-Learning) die *Güte* angibt, d.h. die *Nützlichkeit* dieses Zustandes oder dieses Zustand-Aktion-Paares für optimale Kontrolle. Was bedeutet das, und wie ist das möglich?

Reinforcement Learning bedingt eine besonders Definition der zu optimierenden Zielfunktion. Erklärtes Ziel ist es, diejenige Aktion zu wählen, die die Summe alles zukünftigen Rewards maximiert. Dabei darf das Reward über die Zeit diskontiert sein, d.h. das Reward in Δt Zeitschritten kann mit einem *discount factor* $\gamma^{\Delta t}$ mit $0 \leq \gamma \leq 1$ gewichtet sein. Die zu maximierende Zielfunktion sieht folglich folgendermaßen aus:

$$V(s) \equiv \sum_{t=t_0}^{\infty} \gamma^{t-t_0} R(t) \quad \Bigg| \quad \begin{array}{l} s \text{ ist der aktuelle Zustand} \\ \text{und man wählt ausschließlich bzgl } V \text{ optimale Aktionen} \end{array} \tag{3.1}$$

Die Bewertungsfunktion, die es zu erlernen gilt, gibt eben dieses erwartete kumulierte Reward an, indem sie $V(\cdot)$ approximiert (wenn man Aktionen optimal bezüglich der derzeitigen Bewertungsfunktion wählt). Hat man eine korrekte Bewertungsfunktion, so weiß man, welcher Zustand und/oder welche Aktion welches erwartete kumulierte Reward mit sich bringt. Optimale Aktionen können dann durch Maximierung dieser Bewertungsfunktion generiert werden. Dieser Suchvorgang entspricht der oben beschriebenen indirekten Kontrolle im Aktionsraum eines Modells. An die Stelle eines Modells tritt im Falle von Reinforcement Learning die Bewertungsfunktion.

Im folgenden werden wir eine Methode beschreiben, iterativ diese Bewertungsfunktion zu approximieren. Dabei werden wir uns an die Notation eines Algorithmus anlehnen, der in die Literatur als „Q-Learning“ eingegangen ist. Q sei definiert als:

$$Q(s,a) \equiv \sum_{t=t_0}^{\infty} \gamma^{t-t_0} R(t) \;\big|\; s \text{ ist aktueller Zustand, } a \text{ ist beliebige Aktion} \tag{3.2}$$

Ziel von Q-Learning ist die Identifikation von Q durch sukkzessive Experimentation in der Welt. Q wird durch das Bewertungsnetz, $\hat{Q}$ genannt, approximiert: Nehmen wir an, folgende (Teil-) Sequenz ist beobachtet worden: Ausgehend vom Zustand s wurde Aktion a angewendet, was den Zustand s' mit dem zugehörigen Reward R verursachte. Diese Information ist Grundlage für ein Trainingsbeispiel für $\hat{Q}$:

$$\hat{Q}(\underbrace{s,a}_{\text{Eingabe}}) \longrightarrow \underbrace{R + \gamma \max_{a'} \hat{Q}(s',a)}_{\text{Ausgabe}} \tag{3.3}$$

Diese Trainingsregel geht aus der Definition (3.2) hervor: Bezeichnet $\hat{Q}(s,a)$ das beste zu erwartende zukünftige kumulierte Reward, kann dieses auch ausgedrückt werden als die Summe des Rewards R und dem besten zu erwartenden zukünftigen kumulierten Reward im Zustand s' einen Zeitschritt später. Die iterative Definition in (3.2) ist also in eine rekursive Trainingsregel für die Bewertungsfunktion $\hat{Q}$ umgeformt worden. Es ist leicht zu sehen, daß die Konvergenz der Trainingsregel 3.3 sichert, daß die Maximierung von $\hat{Q}$ optimale Aktionen generiert. $\hat{Q}$ agiert dann wie ein Modell, ein Modell des zukünftigen kumulierten Rewards. Aktionen sind optimal, wenn sie eben dieses kumulierte Reward maximieren, also $\hat{Q}$. Es ist weniger leicht zu sehen, daß sukkzessives Anwenden von Regel 3.3 konvergiert. Wir verzichten auf den Beweis, da er den Rahmen dieser Arbeit sprengen würde.

Es sei darauf hingewiesen, daß Reinforcement-Learning eine bestimmte Form von Asynchroner Dynamischer Programmierung ist. Zustands-Aktionstupel werden durch ihre Nachfolger aktualisiert. In Zustandsräumen mit expliziten Zielzuständen (goal states) führt dies zur Propagierung von Kontrollinformation in Rückwärtsrichtung: Beginnend mit dem Zielzustand ermöglicht Reinforcement Learning, iterativ Kontrollinformationen zu den jeweiligen Vorgängern zu propagiern. Diese „Rückwärtspropagation" ist jedoch nicht mit der Rückwärtsphase in Backpropagation zu verwechseln.

3.3.3 Welches Verfahren leistet was?

Indirekte und direkte Verfahren, insbesondere Lookahead Planen und Reinforcement Learning, haben jeweils entscheidende Vor- und Nachteile. Wir werden diese hier kurz skizzieren, da sie für den Anwender, sowie für das Verständnis des nächsten Kapitels, von Relevanz sind.

Lerngeschwindigkeit: In vielen Welten ist die Identifikation eines Modells deutlich einfacher als das Finden der optimalen $\hat{Q}$-Funktion. Das liegt daran, daß nach jeder Aktion ein Trainingsbeispiel für das Modell generiert werden kann, da nach jeder Aktion ein neuer Zustand beobachtet wird. In den meisten Szenarios für Reinforcement Learning hingegen ist Reward selten, z.B. nur am Ende einer ganzen Sequenz. So erfährt z.B. im Pole Balancing-Problem der Lerner nur am Ende einer ganzen Sequenz ein aussagekräftiges Reward.

In manchen Anwendungen, insbesondere Anwendungen mit großen Eingabe- und Ausgaberäumen, kann das Finden eines Modells und somit der indirekte Ansatz dennoch vergleichsweise langsam sein. Das liegt daran, daß das Modell *alle* Aspekte der Welt modelliert, die Reinforcementfunktion aber nur die relevanten, nämlich das Reward. Ist die Welt durch viele schwer modellierbare und irrelevante Faktoren gekennzeichnet, wird die Modellidentifikation schwierig und langwierig.

Konvergenz: Benutzt man statt Neuronaler Netze Table-lookup-Methoden zur Repräsentation des Gelernten, konvergiert Q-Learning sowie die gängigen anderen Reinforcement Learning Methoden. Das gleiche gilt auch für Planen. Durch die Benutzung von Approximatoren wie Backpropagation ist in beiden Fällen die Konvergenz gefährdet. Aus weiter oben beschriebenen Gründen ist Planen allerdings anfälliger für schlechte Approximationen, da iteratives Anwenden eines ungenauen Modells zu erheblichen Fehlern in der Voraussage führen kann.

Reaktionsgeschwindigkeit: Unter diesem Begriff versteht man die Zeit, die das Generieren einer Aktion benötigt. Hier gewinnen klar die Reinforcement Learning Algorithmen. Während man für diese Methoden nur jeweils eine einzelne Aktion zu optimieren hat, muß man beim Planen eine ganze Aktionssequenz gleichzeitig optimieren, was deutlich komplexer ist. Ist man auf schnelle Reaktion angewiesen, erweist sich Planen als unvorteilhaft. Generell skaliert die Suche von optimalen Aktionen exponentiell in der Länge der Lookaheads N. Für große Werte N ist Lookahead Planen, wie hier vorgestellt, daher unpraktikabel, da es entweder schlechte Ergebnisse liefert, oder zu zeitaufwendig ist, oder beides.

Zielvariabilität: Was geschieht, wenn man die Zielfunktion (das Reward) eines bereits trainierten Kontrollers ändern möchte? Planen erlaubt leicht die Änderung der Zielfunktion, sofern die erforderlichen Zustandsmerkmale vom Modell bereits gelernt sind. Modelle sind unabhängig von der Zielfunktion der Suche, für die sie im Planen verwendet werden. Reinforcement Learning Methoden dagegen erlauben keine Änderungen der Zielfunktion nach dem Training. Wir werden diesen Aspekt ausführlicher im nächsten Kapitel diskutieren.

Horizont: Planer sind begrenzt auf Optimierung von Reward in einem Zeitfenster, dem Horizont. Dieses Zeitfenster muß in der Regel vorgegeben werden. Reinforcement Learning Algorithmen arbeiten dagegen im Prinzip mit unbegrenztem Horizont. Dies liegt an der speziellen Definition der Zielfunktion, die die Summe alles zukünftigen Rewards maximiert. In der Praxis ist allerdings nicht klar, ob dies als Vorteil oder als Nachteil zu werten ist.

Wir werden im nächsten Kapitel diese Fragen aufgreifen. Wir werden einen adaptiven Kontrollalgorithmus vorstellen, der zum Ziel hat, die Vorteile indirekter und direkter Ansätze zu verbinden unter Elimination der jeweiligen Nachteile.

3.4 Literatur

Es gibt ein breites Spektrum relevanter Literatur, die verschiedenste Aspekte verschiedenster Kontrollarchitekturen untersucht. Dabei sind nahezu alle Arbeiten im Bereich Neurocontrol und Reinforcement Learning in den letzten 10 Jahren entstanden.

Control ist eine Wissenschaft mit Jahrzehnte alter Tradition. Zahlreiche Werke sind erschienen, von denen die meisten sich mit verschiedensten Kontrollarten bekannter Systeme, nicht aber mit Lernen beschäftigen. Wichtige Aspekte in dieser Literatur sind *Konvergenz*, *Stabilität* und *Robustheit* der untersuchten Kontroller. Da die „klassische Kontrolltheorie“ meist die Robustheit der verfolgten Ansätze zu beweisen sucht, haben sich Wissenschaftler in diesem Gebiet mit sehr vereinfachten und mathematisch handhabbaren Welten beschäftigt. Der überwiegende Teil der Literatur im Bereich *„adaptive control“* und *„repetitive control“* beschäftigt sich z.B. ausschließlich mit der Kontrolle *linearer* oder *stückweise linearer* dynamischen, meist deterministischen und meßfehlerfreien Welten. Eine gute historische Übersicht über adaptive control mit ausgiebigen und wertvollen Literaturhinweisen findet sich in Kumpati Narendra [Narendra, 1991], geschrieben von einem der Väter des Feldes.

Neurocontrol unterscheidet sich von dieser Art Kontrolltheorie in der Regel dadurch, daß die betrachteten Welten meist hochgradig nichtlinear sein können. Stabilität und Robustheit spielen infolgedessen eine sehr untergeordnete Rolle, während empirische Untersuchungen einen entscheidenden Anteil in der Literatur haben. Dies liegt an dem Dilemma, daß weder für die meisten neuronalen Lernalgorithmen (wie z.B. Backpropagation), noch für die verwendeten nichtlinearen Welten Robustheit- und Konvergenztheorien verfügbar sind. Neurocontrol ist in diesem Sinne gleichermaßen ein Schritt vorwärts – durch das Betrachten komplexerer Probleme – und ein Schritt zurück – durch das Überbordwerfen wichtiger Kontrollprinzipien.

Das Werk zum Einstieg in die Neurocontrolliteratur ist zweifelsohne [Miller *et al.*, 1990]. Dieses Buch stellt eine gelungene Übersicht verschiedenster Arbeiten im jungen Gebiet Neurocontrol dar.

In diesem Buch sei besonders der einleitende Artikel von Andy Barto [Barto, 1989] empfohlen, in dem die grundlegenden Architekturen vorgestellt und erläutert werden. Für Kontrollanwendungen im Roboterbereich seien die Arbeiten von Mike Jordan [Jordan und Rosenbaum, 1988], [Jordan, 1989], [Jordan und Jacobs, 1990], [Jordan und Rumelhart, 1990], [Jacobs und Jordan, 1991b], [Jordan und Jacobs, 1992], Mitsuo Kawato [Kawato *et al.*, 1990] [Kawato, 1990], Chris Atkeson [An *et al.*, 1988], [Atkeson, 1991], Andy Moore [Moore, 1990b], [Moore, 1990a], [Moore, 1991b], [Moore, 1991a] und Narendra/Parthasarathy [Narendra and Parthasarathy, 1990] empfohlen. Während Jordans und Kawatos Gruppen entscheidende Beiträge im Bereich Roboterarmkontrolle geleistet haben, arbeiten Atkeson und Moore im Bereich Kalibrierung, Modellierung und Steuerung komplexer Roboter, wie z.B. Joggler und Tischtennisspieler. Als weiterführende Literatur ist auch David Whites und Don Sofges Handbuch der adaptiven Kontrolle [White und Sofge, 1990] zu empfehlen. Des weiteren ist Robinsons Promotion empfehlenswert [Robinson, 1989]. Charles Schley et al. [Schley *et al.*, 1991] beschreibt eine Implementation eines Neurokontroller für die Automatische Landung von Flugzeugen.

Die wohl einflußreichste Implementation, die auf dem Prinzip „Lernen durch Lehrer“ basiert, ist Dean Pomerleaus ALVINN [Pomerleau, 1989], [Pomerleau, 1991]. ALVINN ist ein autonomes Vehikel, das, ausgerüstet mit einer Kamera und einem Backpropagation-Netz (zwei Schichten, 4 interne Knoten), in der Lage ist, in einer Geschwindigkeit von bis zu 55 mph (88 km/h) zu fahren. Diese Implementation besticht durch ihre Einfachheit und ist wahrscheinlich die beeindruckenste im Bereich Neurocontrol und Neurovision.

Der Begriff Reinforcement Learning, ursprünglich durch Marvin Minsky in die Informatik importiert, hat durch Andy Bartos Gruppe in UMASS seit den späten 70ern deutliche Beachtung gefunden. Obwohl Reinforcement Learning im Prinzip erst einmal nichts mit Neuronalen Netzen zu tun hat, sind viele der entwickelten Ideen mit neuronalen Funktionsapproximatoren getestet worden. Dies liegt nicht zuletzt an der Offenheit der „Neuro community“. Bartos Studenten, insbesondere Rich Sutton, Charles Anderson, Jonathan Bachrach und jüngst Satinder Singh, haben zur Verbreitung des Gedankenguts entscheidend beigetragen. Dem Leser seien vor allem Bartos jüngste Beiträge [Barto *et al.*, 1990], [Barto *et al.*, 1991] empfohlen, die eine exzellente Einführung in das Feld vermitteln. Rich Sutton promovierte 1984 in Bartos Gruppe. Seine Promotion [Sutton, 1984] ist die erste, die Reinforcement Learning und Temporal Credit Assignment untersuchte. In dieser Promotion wurde der AHC-Algorithmus vorgestellt, einer der beiden meist beachtetsten Algorithmen und vielleicht der Grundstein im Gebiet Reinforcement Learning. Rich Suttons Artikel [Sutton, 1988] ist ebenfalls ein entscheidender Beitrag und sehr empfehlenswert. Hier definiert Sutton den Lernalgorithmus $TD(\lambda)$, der die wesentliche Idee des Reinforcement Learnings birgt und erweitert, und beweist seine Konvergenz für den Spezialfall $\lambda = 0$. Ein Jahr später veröffentlichte Chris Watkins seine Promotion [Watkins, 1989] (siehe auch [Watkins, 1992]), in der er Q-Learning vorstellte. Seitdem ist Q-Learning der wohl beachtetste Algorithmus im Bereich Reinforcement Learning, nicht zuletzt, weil er einige Vorzüge gegenüber Suttons AHC besitzt. Peter Dayans Promotion [Dayan, 1991], die sich radikal mit Reinforcement Learning beschäftigt, enthält einen Beweis der Konvergenz von $TD(\lambda)$ für beliebige λ. Dieser Beweis, zusammen mit Suttons Originalbeweis, sei dem Leser empfohlen, der die in Kapitel 3.3.2 angesprochenen

Konvergenzergebnisse nachlesen möchte.

Zahlreiche Forscher haben zur Maturation von Reinforcement Learning beigetragen. Wichtige Arbeiten sind die von Ron Williams [Williams, 1987], Paul Werbos [Werbos, 1990b] über *adaptive heuristische Kritiker*, Leslie Kaelbling [Kaelbling, 1990] über die Anwendung von statistischen Methoden in Reinforcement Learning, Long-Ji Lin [Lin, 1991b], [Lin, 1991a] über die effiziente Kombination hybrider Lerntechniken, Satinder Singh [Singh, 1992b] [Singh, 1992a] im Bereich hierarchischen Planens. Jonathan Bachrach [Bachrach, 1991] war einer der ersten, die Reinforcement Learning für simulierte Roboter untersucht haben. Shridar Mahadevan/Jonathan !!Connell [Mahadevan und Connell, 1990] haben Reinforcement Learning im Kontext von Rod Brooks *behavior-based robotics* studiert. Lernen bei unvollständiger Wahrnehmung – wie sich herausstellt ein wesentlich komplexeres Problem – ist zum Beispiel von Jon Bachrach/Michael Mozer [Bachrach und Mozer, 1991], Lonnie Chrisman [Chrisman, 1992], Long-Ji Lin/Tom Mitchell [Lin und Mitchell, 1992], Rivest/Shapire [Rivest und Schapire, 1987], Jürgen Schmidhuber [Schmidhuber, 1991b], Ming Tan [Tan, 1991] und Steve Whitehead [Whitehead und Ballard, 1991] mit zum Teil äußerst unterschiedlichen Methoden untersucht worden. Vor etwa einem Jahr stellte Gerald Tesauro ein Programm vor, das völlig autonom lernen kann, Backgammon zu spielen [Tesauro, 1991], [Tesauro, 1992]. Dieses Programm, das Großmeistern paroli bieten kann, basiert auf eine Kombination von Suttons $TD(\lambda)$ und Backpropagation, und wird allgemein als die erste nichttriviale Implementation auf dem Gebiet des Reinforcement Learnings angesehen. Eine deutschsprachige Einführung in Neurocontrol findet sich in Knut Möllers Promotion [Möller, 1991].

Kapitel 4

Ein kombiniert direkt-indirekter Adaptiver Kontroller

In diesem Kapitel werden wir die im letzten Kapitel vorgestellten Prinzipien für Adaptive Kontrolle aufgreifen, erweitern, und schließlich einige Anwendungen beschreiben. Wir werden einen indirekt-direkten Kontroller vorstellen, der gewissermaßen Vorteile beider Methoden nutzt und deren Nachteile eliminiert. Das resultierende System arbeitet in der frühen Phase des Lernens strikt indirekt, indem es ein internes Modell der Welt konstruiert und Aktionen mit Lookahead Planen maximiert. Nach jedem Planen werden generierte Aktionen in einen direkten Kontroller, in das sogenannte *Erfahrungsnetz* (*experience network*) kompiliert. Mit gradueller Verbesserung der Qualität dieses Netzes wird die Kontrolle schrittweise dem indirekten, planbasierten Kontroller entzogen, was eine Steigerung der Reaktionsgeschwindigkeit durch die wegfallende Suche mit sich bringt. Schließlich, wenn das Erfahrungsnetz vollständig trainiert ist, realisiert das System einen direkten Controller. Die Kombination von indirekter und direkter Kontrolle birgt nicht nur den Vorteil kurzer Lernzeiten bei verhältnismäßig schneller Reaktion. Darüber hinaus ist das System in der Lage, sich an bestimmte unvorhersehbare Änderungen in der Welt anzupassen. So führt zum Beispiel eine Änderung der Rewardfunktion zu längeren Reaktionszeiten, nicht aber zu ungeeigneten Aktionen. Unvorhersehbare Änderungen in der Welt können in das Modell absorbiert werden. Da Modelle in der Regel vergleichsweise schnell gefunden werden können, adaptiert der Kontroller schneller als die meisten rein direkten Systeme.

4.1 Der Planer

Bevor wir die Details des kombinierten Kontrollers explizit vorstellen und ihre Interaktion beschreiben, wollen wir den Leser mit einem einleitenden Beispiel motivieren. Dieses Beispiel ist als ein roter Faden gedacht, der nicht zuletzt der Veranschaulichung von Problematiken und Lösungen dient.

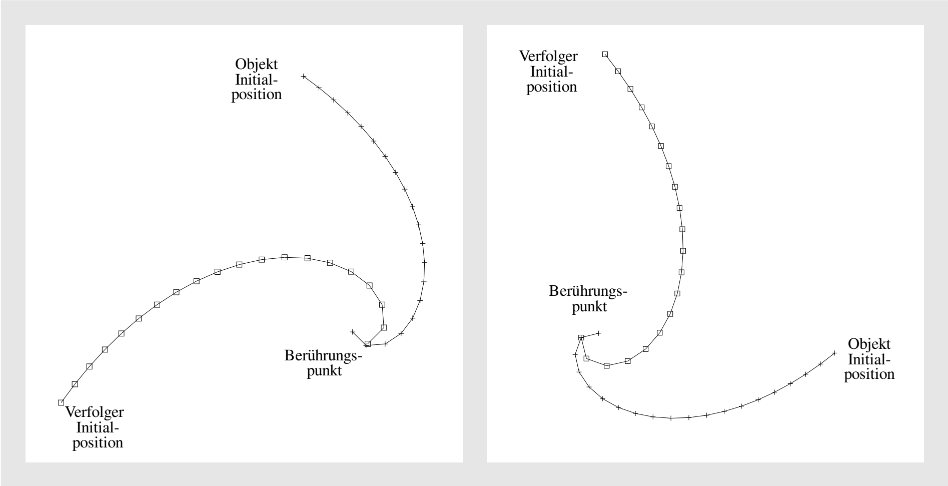


Abbildung 4.1: Target Tracking – Trivialstrategie: Bewegt sich der Verfolger (Quadrate) unmittelbar in die Richtung des Objektes (Kreuze), ist die resultierende Lösung nicht optimal: Planen ist angezeigt.

4.1.1 Target Tracking – ein einleitendes Beispiel

In Abbildung 4.1 ist ein einfache Welt illustriert, anhand derer wir im folgenden einen Kontroller designen werden. Diese „Welt" ist ein Target Tracking-Szenario, in dem ein Agent (Quadrat) ein sich bewegendes Objekt (Kreuz) fangen muß. Damit der Verfolger eine gute Chance hat, ist er ein wenig schneller als das Objekt. Was der Verfolger allerdings nicht weiß, ist die Tatsache, daß das Objekt mit einer Strategie dem Verfolger zu entkommen versucht. Das Objekt bewegt sich immer rechtwinklig (nach rechts) zum Verfolger.

Zustände in diesem Target Tracking Szenario sind definiert als die Koordinaten des Verfolgers (x_V, y_V) und die des Objektes (x_O, y_O). Aktionen sind Koordinatenänderungen $(\Delta x_V, \Delta y_V)$. Die Dynamik der Welt ist somit gegeben durch:

$$\begin{aligned} (x_V, y_V) &\longleftarrow (x_V + \Delta x_V, y_V + \Delta y_V) \\ (x_O, y_O) &\longleftarrow (x_O + 0.9\Delta y_V, y_O - 0.9\Delta x_V) \end{aligned} \tag{4.1}$$

und das Reward ist definiert als komponentenweise Distanz zwischen Objekt und Verfolger.[1]

Da der Verfolger schneller als das Objekt ist, gibt es eine triviale Strategie, die immer zum Ziel führt: *„Gehe immer in die Richtung der aktuellen Objektposition!"* Wie aber in Abbildung 4.1 gezeigt, führt dies nicht zu optimalen Lösungen. An diesem Beispiel sieht man den Nutzen von Planungsalgorithmen: Wenn ein Kontroller die Strategie des Objektes vorherzusagen lernt, kann dieses in weniger Schritten gefangen werden, wie in Abbildung 4.2 illustriert.

[1] Das Reward ist somit ein Tupel, was die Modellierung des Rewards mit Künstlichen Neuronalen Netzen erleichtert und die betrachteten Kontrollprinzipien nicht weiter beeinträchtigt. Ziel des Kontrollers ist die Minimierung der Distanz zwischen Objekt und Verfolger oder, in unserer Terminologie, die Minimierung der Abweichung zwischen tatsächlichem und optimalem Reward.

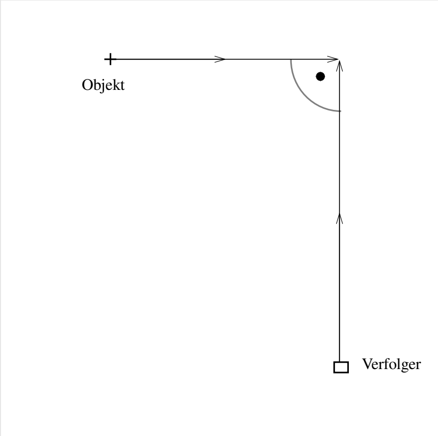


Abbildung 4.2: Target Tracking – analytisch optimale Strategie.

4.1.2 Aktionssuche mit Lookahead Planen

Wie haben die grundlegende Idee des Lookahead Planens bereits im vorangegangenen Kapitel beschrieben und diskutiert. Wir werden im folgenden den verwendeten Algorithmus präzisieren:

- Die zu maximierende Reward-Zielfunktion ist gegeben als:

  $$J = \frac{1}{2} \sum_{\tau=1}^{N} G(\tau) \cdot |R^{*} - R(t)|^{2} \tag{4.2}$$

  Dabei R^{*} das optimale Reward, und $G(\cdot)$ ist eine Rewardgewichtungsfunktion, die das Gewichten der vorausgesagten Rewards untereinander ermöglicht.

  Im Target Tracking Beispiel sei $G(\tau) = 2^{\tau}$. Diese exponentiell steigende Funktion stellt sicher, daß zeitlich entfernte Rewards die Aktionssuche stärker beeinflussen als zeitlich nähere. Dies reflektiert genau das Verhalten, das wir uns im Target Tracking wünschen: Kurzfristiges Reward soll zugunsten einer guten Gesamtlösung aufgegeben werden. Gewichtungsfunktionen, wie die hier vorgestellte, haben zwei Nachteile: Zum einem muß man eine brauchbare Funktion vorgeben, was sich in vielen Anwendungen als schwierig erweist. Zum anderen gewichtet die hier verwendete exponentielle Funktion spätere Vorhersagen stärker als frühere, obwohl wegen der kumulativen Eigenschaft von Vorhersagefehlern spätere Vorhersagen in der Regel ungenauer sind. Wir werden später andere Gewichtungsfunktionen beschreiben.

- Geplant wird mit einem maximalen Horizont N.

- In jedem Schritt wird komplett neu geplant. Dieses ineffizient erscheinende Verfahren wird effizienter durch die Wiederbenutzung von Teilplänen: Wenn eine Aktion ausgeführt ist,

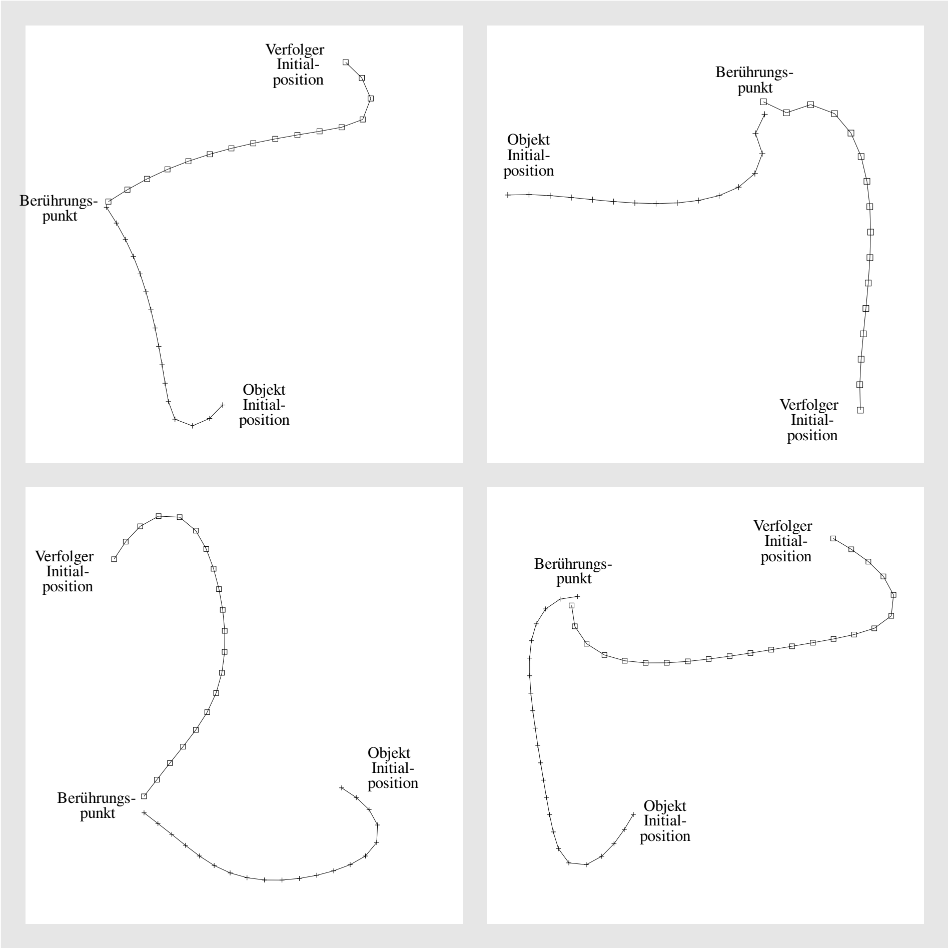


Abbildung 4.3: Kontrolle nach 800 Trainingsschritten.

werden die letzten $N-1$ Aktionen des jüngsten Plans als Initialaktionen für den neuen Plan verwendet. Somit muß nur eine, nämlich die letzte Aktion eines Planes neu (zum Beispiel zufällig) initialisiert werden.

- Das Suchverfahren im Planraum ist Gradientenabstieg. Im ersten Schritt einer Episode (Sequenz) wird der Plan mit zufälligen Werten initialisiert. Da die Modellnetze, wie auch die Reward-Zielfunktion J, differenzierbar sind, kann man die zum Gradientenabstieg erforderlichen Gradienten berechnen, wie in Kapitel 3.3.1 skizziert.

4.1.3 Ergebnisse

Wie arbeitet Planen für unser Target Tracking Problem? Zweifelsohne ist die Zustandsübergangsfunktion der Welt leicht zu modellieren, da sie linear und somit für das Modellnetz leicht lernbar

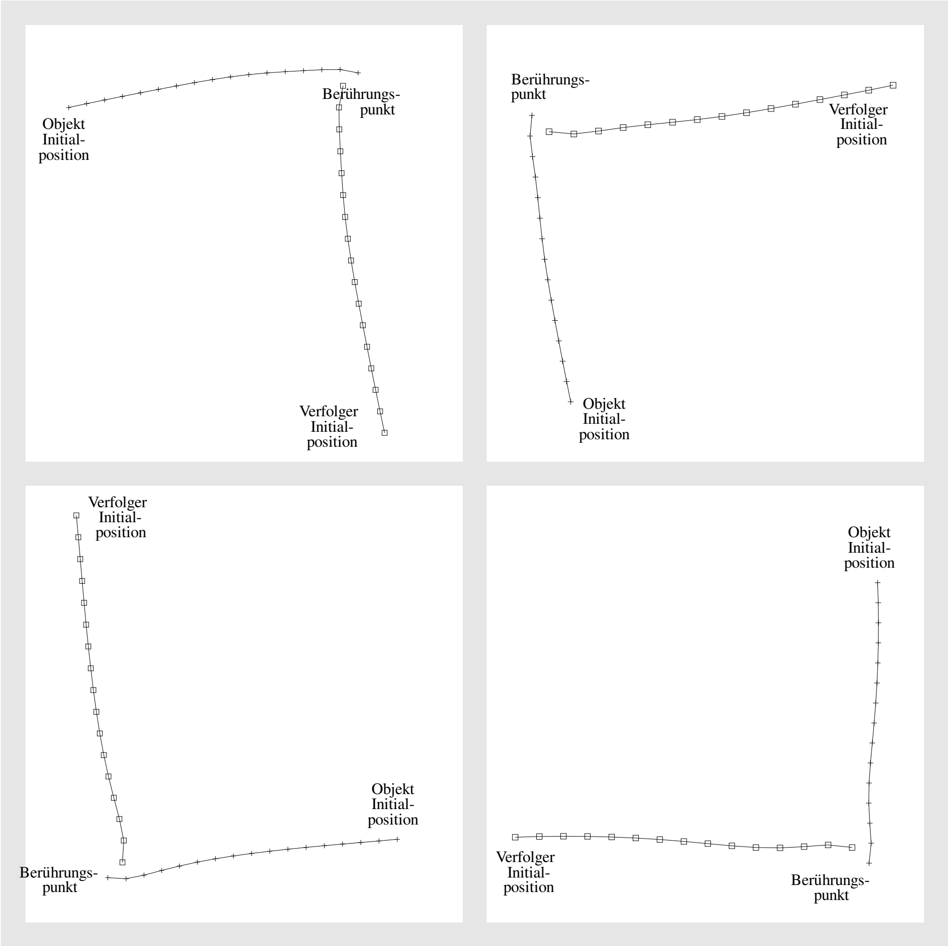


Abbildung 4.4: Kontrolle nach 3 500 Schritten. Die generierten Aktionen sind nahezu optimal.

ist. Die eigentliche Rewardfunktion, die euklidische Distanz zwischen Objekt und Verfolger, ist verhältnismäßig schwer modellierbar. Aus diesem Grunde haben wir die Koordinatendifferenz zwischen Objekt und Verfolger

$$\frac{x_V - x_O}{2} + \tfrac{1}{2} \quad \text{und} \quad \frac{y_V - y_O}{2} + \tfrac{1}{2} \tag{4.3}$$

komponentenweise lernen lassen. Mit der optimalen Distanz „0" (in der Skalierung von Gleichung (4.3): 0.5) mißt dann die Reward-Zielfunktion J (vergleiche Gleichung (4.2)) die gewünschte euklidische Distanz, während die modellierte Distanz eine lineare Funktion ist. Um die Linearität der Welt wissend, haben wir in unseren Versuchen ein Netz ohne interne Schichten zur Modellierung verwendet, was das Lernen weiter beschleunigte. Die Abbildungen 4.3 und 4.4 zeigen jeweils vier Beispiele generierter Trajektorien nach 800 und nach 3 500 Lernschritten mit Planungshorizont $N = 10$. Wie zu erwarten, findet der Planalgorithmus nahezu optimale Trajektorien nach 3 500 Schritten. Das Target Tracking Problem kann leicht und schnell mit Planen gelöst werden. Es sei angemerkt, daß dieses Problem insbesondere leicht lernbar ist, da die verwendeten Modelle approximativ lineare Funktionen realisieren. In linearen Funktionen führt Gradientenabstieg sicher zur optimalen Lösung, was für die weiter unten beschriebenen nichtlinearen Welten nicht gilt. Des weiteren wurde der Planungshorizont N dynamisch gewählt: Wenn das Modellnetz maximales Reward voraussagte, d.h. die Distanz von Objekt und Verfolger kleiner als ein kleines ε war (ε entsprach der Schrittweite des Verfolgers), wurde weiteres Lookahead verhindert. Diese Technik, die auch in den weiter unten beschriebenen Roboterarmexperimenten verwendet wurde, kann die Effizienz von Planen steigern.

4.2 Kompilieren von Erfahrung: die direkte Komponente

Nun erfordert Planen, wie bereits erwähnt, vergleichsweise viel Zeit zur Aktionsfindung (Reaktionszeit). Wir werden deshalb den Kontroller um eine direkte, schnelle Komponente erweitern, die mit wachsender Praxis die Planungszeiten drastisch reduzieren kann: das *Erfahrungsnetz* (*experience network*).

Die Idee dieses Netzes ist einfach: Ein Planer ist eine Funktion, die den aktuellen Zustand in eine Aktion überführt. Diese Aktion ist bezüglich des Rewards in einem endlichen Suchhorizont optimiert. Die Aufgabe des Erfahrungsnetzes ist es, diese Abbildung direkt zu lernen; der Planer ist damit der *Lehrer* des Erfahrungsnetzes. Dies geschieht, indem nach jedem Planen Trainingsbeispiele aus dem aktuellen Zustand als Eingabe, und die vom Planer generierte Aktion als gewünschte Ausgabe erzeugt werden. Das Erfahrungsnetz lernt also, den Planer zu simulieren. Damit fällt die hier verwendete Technik in die in Kapitel 3.1 beschriebene Technik des „Lernens mit Lehrer“.

Dieses Verfahren wirft zwei Probleme bzw. offene Fragen auf:

1. Was geschieht mit Mehrdeutigkeiten? Der Planer kann z.B. für ein und denselben Zustand mehrere Aktionen generieren, und welche soll dann gelernt werden?

2. Wie wird das Erfahrungsnetz, das ja einen zweiten Kontroller darstellt, in der Aktionsgenerierung eingesetzt?

Zuerst zur zweiten Frage: Das Erfahrungsnetz wird benutzt, um *initiale Pläne* zu erzeugen. Diese initialen Aktionen werden dann vom Planer weiter optimiert. Approximiert das Erfahrungsnetz den Planer gut, ist der erforderliche Planungsaufwand gering. Die Verwendung des Erfahrungsnetzes zur Initialisierung des Planers beantwortet im übrigen auch die erste Frage: Da der Planer Gradientenabstieg anwendet, findet er die der initialen Aktion nächste lokal optimale Aktion. Damit konvergiert die Ausgabe des Erfahrungsnetzes gegen diese eine Aktion – andere lokal optimale Aktionen werden nicht gelernt, da sie gar nicht erst generiert werden.

Dies wirft die dritte und eigentlich entscheidende Frage auf, die Frage nach *globaler* Optimierung. Wie schon erwähnt ist Gradientenabstieg durch lokal optimale, nicht aber global optimale Aktionen charakterisiert. Dieser im allgemeinen unerwünschte Effekt überträgt sich auf das hier vorgestellte

Abbildung 4.5: Das Roboterarmexperiment: Ein Roboterarm soll einen langsam rollenden Ball in minimaler Zeit berühren. Weder die Dynamik des Balles, noch die Kinematik des Armes sind dem Kontroller beim Versuchsbeginn bekannt. Durch Experimentation werden diese gelernt, und mit Lookahead Planen werden Aktionen optimiert.

Kontrollschema. Dies charakterisiert den wohl stärksten Nachteil von Gradientenverfahren in Planungsdomänen: Je komplexer die Welt, desto unwahrscheinlich ist es, mit Gradientenabstieg optimale Aktionen zu generieren. Wo dies ein Problem ist, müssen andere Suchalgorithmen (z.B. *grid search, genetic algorithms*) eingesetzt werden. Die Anwendungen anderer Suchtechniken eröffnet eine interessante Perspektive zukünftiger Forschung in diesem Gebiet.

4.3 Berühren eines rollenden Balles mit einem Roboterarm

Wir werden das Zusammenspiel von indirekter und direkter Kontrolle, von Planen und Erfahrungsnetz, anhand eines komplexeren Beispieles untersuchen. Es geht darum, mit einem Roboterarm einen rollenden Ball zu berühren. Der Aufbau des Systems ist in Abbildung 4.5 gezeigt: Ein Kamerasystem beobachtet einen rollenden Ball und einen Roboterarm und berechnet die Koordinaten von beiden Systemen in einem globalen Koordinatenraum. Diese Daten werden dem adaptiven Neurokontroller übergeben, der zum einen lernt, Beobachtungen vorherzusagen, und der zum anderen mithilfe dieser Modelle Aktionen generiert.

Da wir uns für das Lernen von Kontrolle interessieren, nehmen wir an, daß das Kamerasystem akkurate Informationen in Echtzeit liefert. Die Ballposition wird dabei durch (x_{Ball}, y_{Ball}) und die Armposition durch die Position der Hand $(x_{Hand}, y_{Hand}, z_{Hand}, \theta_{Hand})$ beschrieben. Hierbei beziehen sich x_{Hand}, y_{Hand} und z_{Hand} auf die absolute Position eines speziell markierten Punktes zwischen den beiden Fingern des Greifers, und θ_{Hand} mißt den Winkel zwischen dem Greifer und dem Tisch. Die Angabe dieses Winkels ist aus technischen Gründen erforderlich: Ließe man diesen weg, wäre die Konfiguration (die Winkel der Gelenke) nicht eindeutig bestimmt; die

Abbildung 4.6: Modellnetze und Erfahrungsnetz im Roboterarmexperiment. Die Eingabe ist in der Mitte der Abbildung zu finden. Ein rekurrentes Ballmodell (links oben) und ein Roboterarmmodell (rechts oben) sagen die Trajektorie von Ball und Roboter vorher. Das Distanznetz evaluiert das Reward. Im unteren Teil der Abbildung findet sich das Kontrollnetz, das zur Generierung von initialen Plänen eingesetzt wird.

---

inverse Kinematik und somit das zu lernende Modell *wäre nicht eindeutig*. Im Gegensatz zu den Greiferkoordinaten, die im Arbeitsraum (*workspace*, x-y-z-Raum) gemessen werden, beziehen sich die Aktionen auf den Konfigurationsraum (*configuration space*, Raum der Armkonfigurationen). Aktionen sind vierelementige Vektoren $(\Delta\theta_1, \Delta\theta 2, \Delta\theta_3, \Delta\theta_4)$, die die Änderungen der Winkel der vier verfügbaren Gelenke messen. Die unterschiedlichen Repräsentationen von Zustands- und Aktionsraum gestalten das Modellierungsproblem interessant: Zur analytischen Lösung dieser Gleichungen muß erst die inverse Kinematik und dann die Vorwärtskinematik des Arms berechnet werden.

Reward ist, analog zum Target Tracking Experiment, als der Abstandsvektor zwischen den Ball- und den Greiferkoordinaten definiert. Abbildung 4.7 zeigt, was passiert, wenn in jedem Schritt das Reward strikt maximiert wird (*greedy optimization*): Der Roboterarm folgt dem Ball. Ziel des Experimentes ist es, einen Kontroller zu finden, der ohne Vorwissen über Ball, Arm oder Reward

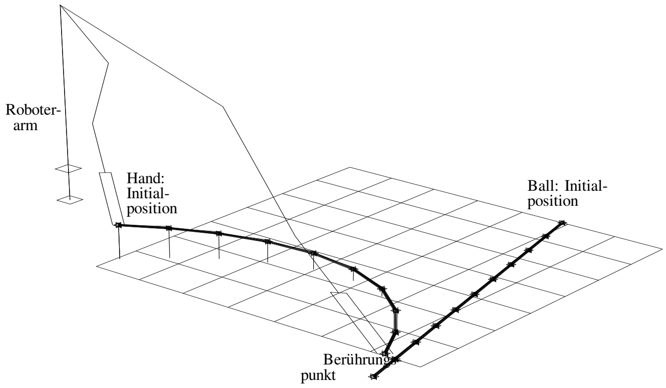


Abbildung 4.7: Das Roboterarmexperiment: Unmittelbare Optimierung des Rewards, das die Distanz zwischen Ball und Arm mißt, ist keine optimale Strategie. Der Arm „verfolgt“ den Ball.

den Ball in kurzmöglichster Zeit berührt.

Die verwendeten Netze – drei Modellnetze für den Roboterarm, den Ball und die Distanz zwischen beiden, sowie ein Kontrollnetz – sind in Abbildung 4.6 gezeigt. Gemäß unserer Definition liefert das Kamerasystem nur die aktuellen Ballkoordinaten als Zustandsvektor. Diese reichen aber für eine Vorhersage des nächsten Zustandes nicht aus, da man von einer Momentanaufnahme die Bewegungsrichtung und Geschwindigkeit eines Balles nicht bestimmen kann. Daher ist das Ballmodellnetz rekursiv – wir haben ein in Kapitel 2.4.4 beschriebenes Elman-Netz eingesetzt. Andere Zeitnetze, wie zum Beispiel Time Delay Neural Networks oder Jordan-Netze, hätten ebenfalls eingesetzt werden können. Da die Kontextschicht des Ballnetzes nun ebenfalls Zustandsinformationen speichert (dort ist nach dem Lernen die Bewegungsrichtung und die Geschwindigkeit des Balles repräsentiert), sind die Aktivationen dieser Knoten ebenfalls Eingabe des Erfahrungsnetzes. Die Modularisierung des Modells erleichtert das Lernen des Modells. In unseren Experimenten wurden die einzelnen Komponenten des Modells getrennt trainiert. Es sei angemerkt, daß es sich als vorteilhaft erwiesen hat, die reellwertigen Zustandsvariablen x_{Hand}, y_{Hand}, z_{Hand} und θ_{Hand} jeweils im Intervall $[0.2, 0.8]$ (anstatt in $[0, 1]$) zu repräsentieren. In diesem Bereich agieren die Squashingfunktionen in ihrem linearen Bereich (vergleiche Kapitel 1.2). Viele Wissenschaftler berichten, daß der Sättigungsbereich der logistischen Squashingfunktion, in etwa die Intervalle $[0, 0.2]$ und $[0.8, 1]$, zum Lernen solcher Funktionen weniger geeignet ist.

Abbildung 4.8 demonstriert Lookahead Planen nach rund 20 000 Schritten Lernerfahrung[2]. Folgende Parameter wurden benutzt: Der Horizont war $N = 5$, Gradientenabstieg wurde 9 mal zur iterativen Planoptimierung angewendet, die Rewardgewichtungsfunktion war exponentiell $G(\tau) = 1.3^{\tau}$, und die Planadaptionsrate war die ebenfalls exponentiell gewichtet Funktion $0.05 * 0.9^{\tau}$. Die verschiedenen Grafiken in 4.8 geben einzelne Planungsprozesse wider. Man sieht deutlich, wie der Planungsprozeß Aktionen generiert, die die Zeit bis zur Berührung mini-

[2]Dieser Zeitpunkt wurde willkürlich gewählt. Nach 20 000 Schritten Lernerfahrung schien das Training des Modells zu konvergieren.

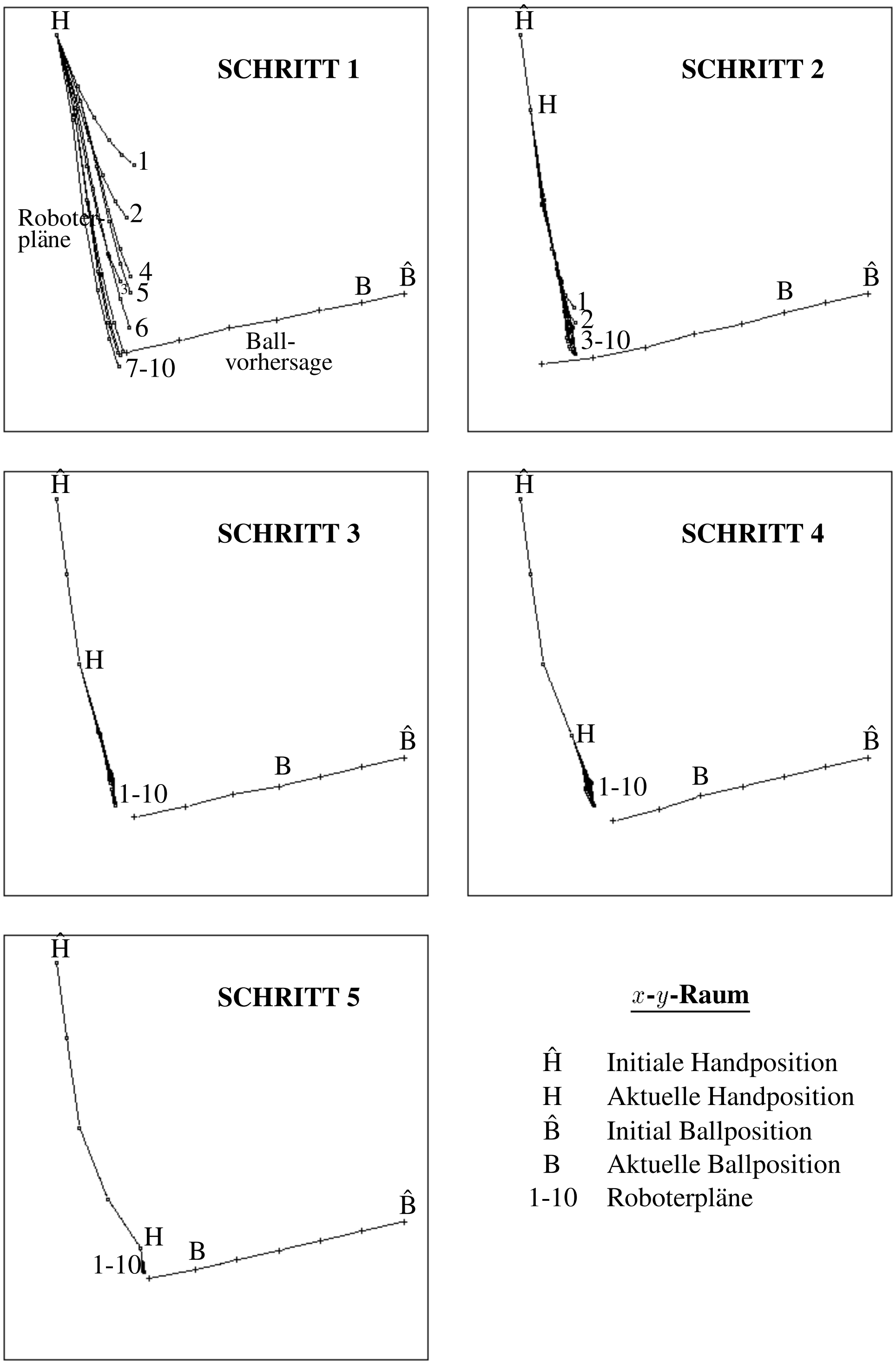


Abbildung 4.8: Zweidimensionale Projektion des Planungsvorganges. Jede Abbildung spiegelt einen Planungsvorgang wider. Nach dem Planen wird die jeweils erste Aktion des Plans ausgeführt.

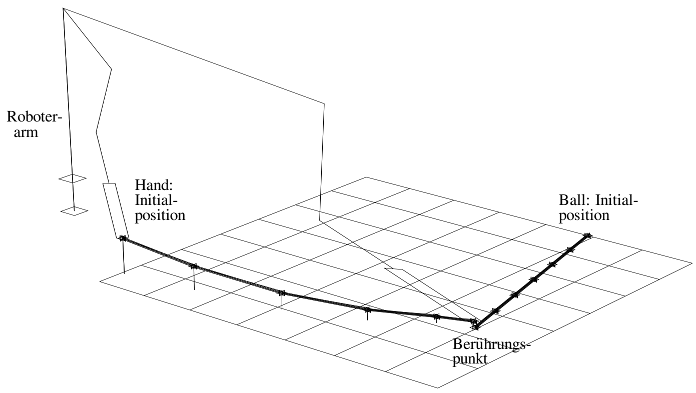


Abbildung 4.9: Durch Planen generierte Trajektorie im Arbeitsraum: Der Roboterarm findet die kürzeste Aktionssequenz in Echtzeit.

mieren. Eine resultierende Trajektorie ist in Abbildung 4.9 gezeigt. Es sei darauf hingewiesen, daß nur in ca. 60% aller Versuche optimale Aktionssequenzen generiert werden. In 40% der Fälle findet der Planer nur deutlich schlechtere, suboptimale Lösungen. Dies liegt insbesondere daran, daß Gradientenabstieg keine vollständige Suchmethode ist.

4.3.1 Übergang zur direkten Kontrolle

Wie man Abbildung 4.9 entnimmt, hat das Modellnetz die Welt gut genug approximiert, um das Generieren von quasi-optimalen Aktionen zu erlauben. Hand in Hand mit dem Training des Modells wird das Kontrollnetz trainiert. Wie bereits gesagt, lernt dieses, das Ergebnis des Planungsprozesses vorauszusagen.

Testet man das Kontrollnetz nach 20 000 Schritten, wird man enttäuscht sein. Die vom Kontrollnetz unter gleichen Bedingungen wie Abbildung 4.9 generierten Aktionsvorschläge führen zu der in Abbildung 4.10 gezeigten Aktionssequenz. Hier wird die zeitliche Diskrepanz zwischen dem direkten und dem indirektem Ansatz deutlich. Da das Erfahrungsnetz ausschließlich mit vom Modellnetz generierten Aktionen trainiert wird, hinkt bei gleichzeitigem Training beider Netze das Erfahrungsnetz dem Modellnetz zwangsläufig hinterher. Nach weiteren 20 000 Schritten sieht die Situation besser aus. Die vom Erfahrungsnetz vorgeschlagenen Aktionen sind gut genug, den Ball zu berühren, wie in Abbildung 4.11a illustriert. Weiteres Planen verändert diese Aktionen nicht nennenswert (Abbildung 4.11b). Dieses Ergebnis demonstriert, wie die Integration von indirekter und direkter Kontrolle in der Praxis arbeitet: Während anfangs die indirekte Kontrolle dominiert, gewinnt mit zunehmender Genauigkeit des hinterherhinkenden Erfahrungsnetzes die direkte Kontrolle immer mehr an Gewicht. Schließlich ist das Erfahrungsnetz gut genug, um meist ohne weiters Planen quasi-optimale Aktionen zu generieren.

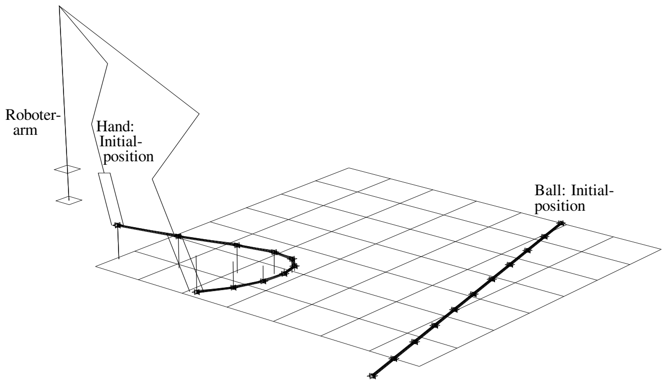


Abbildung 4.10: Nach 20 000 Trainingsschritten liefert das Erfahrungsnetz alleine keine optimale Aktionen. In der hier gezeigten Sequenz hätte der Greifer den Tisch berührt.

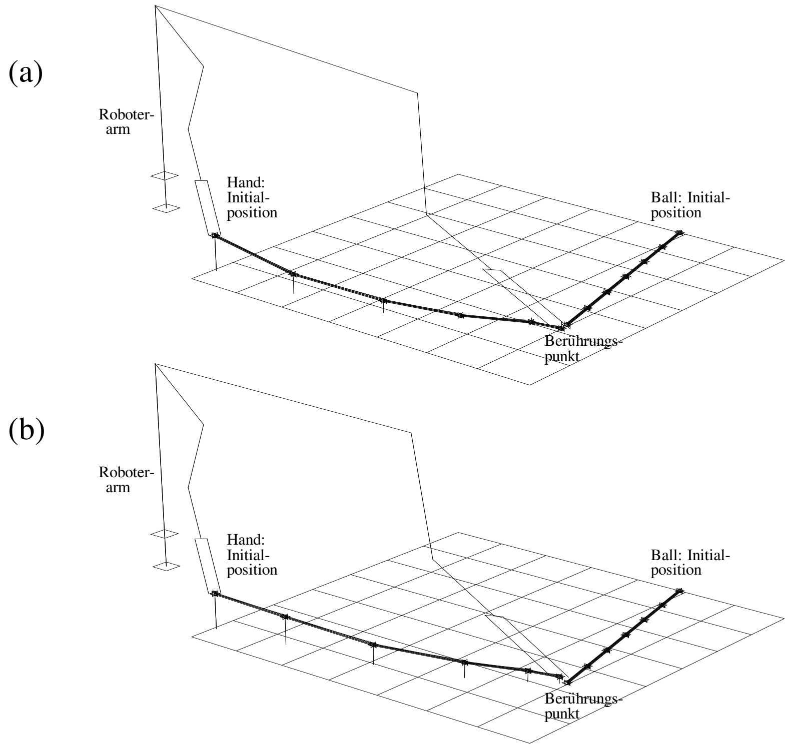


Abbildung 4.11: (a) Nach weiteren 20 000 Schritten simuliert das Erfahrungsnetz erfolgreich den Planer. Dies reduziert die Planungszeit drastisch, da das Erfahrungsnetz als direkter Kontroller deutlich kürzere Reaktionszeiten hat. (b) Weiteres Planen kann das Ergebnis nicht sichtbar verbessern.

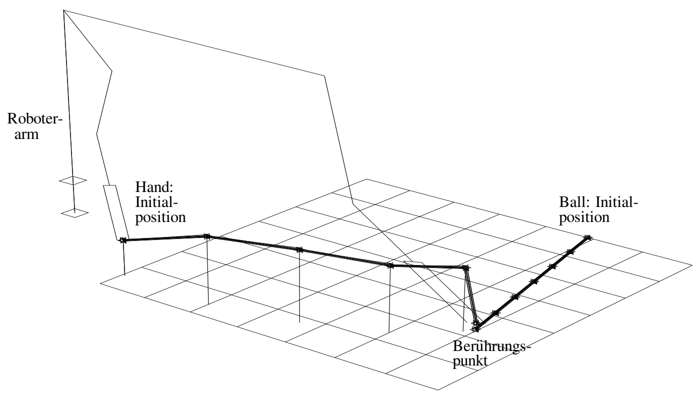


Abbildung 4.12: Geänderte Zielfunktion: Der Arm nähert sich von oben.

4.3.2 Ändern der Zielfunktion zur Laufzeit

Eine der immanenten Schwächen von Reinforcement Learning Algorithmen, wie das in Kapitel 3.3.2 vorgestellte Q-Learning, ist die Tatsache, daß bereits geringfügige Änderungen der zu optimierenden Zielfunktion weiteres Training bedingen. Dies liegt daran, daß diese Methoden ausschließlich das kumulative Reward modellieren. Eine Änderung der Rewardfunktion ist somit fatal.

Planungsmethoden sind universeller, da das Modell nicht notwendigerweise vom Reward abhängt. In dem folgenden Beispiel haben wir die Rewardfunktion – hier eine Funktion des Zustandes, die dem Lerner bekannt sei – geringfügig geändert. Anstatt den Ball auf dem kurzmöglichsten Wege zu berühren, ist das geänderte Reward ausschließlich dann maximal, wenn der Ball *von oben* berührt wird, d.h., wenn die letzten beiden Schritt von oben nach unten erfolgen[3]. Um das in Abbildung 4.12 gezeigte Resultat zu erhalten mußten wir allerdings die Anzahl der Gradientenabstiegsiterationen von 9 auf 18 verdoppeln. Dies liegt daran, daß das Erfahrungsnetz Aktionen vorschlägt, die die alte Zielfunktion optimieren. Diese Aktionen sind ähnlich genug, um in der neuen Situation als geeignete Initialpunkte zu dienen. Weiteres Planen ist dann in der Lage, die Aktionen bezüglich der geänderten Zielfunktion zu optimieren.

4.4 Von stetigen zu diskreten Räumen: Das „Pole Balancing-Problem“

Als abschließendes Beispiel werden wir uns schließlich dem im Kapitel 3 vorgestellten Pole Balancing-Problem widmen. Zur Wiederholung: Auf einem horizontal in einer Dimension be-

[3]Genau genommen wird eine Funktion des Winkels der letzten Aktion dem bisherigen Rewards hinzuaddiert. Diese Funktion ist maximal, wenn die Bewegung von oben erfolgt.

weglichen Wagen ist ein Stab mittels eines passiven Gelenks befestigt, wie in Abbildung 3.2 gezeigt. Diesen Stab gilt es durch das Ausüben von Kräften auf den Wagen so lange wie möglich zu balancieren. Erschwert wird dies dadurch, daß die Bewegungsfreiheit des Wagens eingeschränkt ist. In der in der Neurocontrol-Literatur meist studierten Version des pole-balancing problems wird der Zustand durch die Größen $x(t) \in [3.0\text{m}, 3.0\text{m}]$ (Position des Wagens), $\dot{x}(t) \in [$-$1.5\frac{\text{m}}{\text{s}}, 1.5\frac{\text{m}}{\text{s}}]$ (Geschwindigkeit des Wagens), $\theta(t) \in [0.30, 0.30]$ (Winkel des Stabes) und $\dot{\theta}(t) \in [$-$2, 2]$ (Winkelgeschwindigkeit des Stabes) mit Zeitparameter t beschrieben. Aktionen sind das Ausüben einer Kraft $F \in \{10\text{nt}, 10\text{nt}\}$ fixen Betrages entweder von rechts oder von links in Zeitabständen von $\tau = 0.02$ Sekunden. Die Dynamik des Systems ist gegeben durch:

$$
\begin{aligned}
\ddot{\theta} &= \frac{-g\sin\theta + \cos\theta\left[\frac{-F - m_p l \dot{\theta}^2 \sin\theta + \mu_c \operatorname{sgn}(\dot{x})}{m_c + m_p}\right] - \frac{\mu_p \dot{\theta}}{m_p l}}{l\left[\frac{4}{3} - \frac{m_p \cos^2\theta}{m_c + m_p}\right]} \\
\ddot{x} &= \frac{F + m_p l\left[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta\right] - \mu_c \operatorname{sgn}(\dot{x})}{m_c + m_p}
\end{aligned}
\qquad (4.4)
$$

mit

m_c	=	1.0 kg	=	Masse des Wagens,
m_p	=	0.1 kg	=	Masse des Stabes,
l	=	0.5 m	=	Abstand des Massenmittelpunkts des Stabes zum Gelenk,
μ_c	=	0.0005	=	Reibungskoeffizient der Räder,
μ_p	=	0.000002	=	Reibungskoeffizient des Gelenks
g	=	-9.81 $\frac{\text{m}}{\text{s}^2}$	=	Beschleunigung durch Gravitation.

Diese Differentialgleichungen werden in der Regel nicht gelöst, sondern durch stückweise Linearisierung approximiert, indem die Beschleunigungen $\ddot{\theta}$ und $\ddot{x}$ in Zeitintervallen von $\tau = 0.02$ als konstant angesehen werden. Somit definieren die Gleichungen in (4.4) eine Zustandsübergangsfunktion der Welt.

Ziel des Kontrollers ist es, den Stab so lange wie möglich zu balancieren, ohne daß dabei der Wagen seinen vorgegebenen horizontalen Bewegungsfreiraum überschreitet. Um das Lernen zu erleichtern, benutzen wir ein zweidimensionales Reward (R_θ, R_x):

$$
\begin{aligned}
R_\theta &= \begin{cases} -1 & \text{wenn } \theta \notin [0.21, 0.21] \\ 1 & \text{sonst} \end{cases} \\
R_x &= \begin{cases} -1 & \text{wenn } x \notin [2.4\text{m}, 2.4\text{m}] \\ 1 & \text{sonst} \end{cases}
\end{aligned}
\qquad (4.5)
$$

Nun birgt dieses Problem eine Schwierigkeit, die den vorangehenden nicht gemein ist: das Reward ist binär. Lernt man mit einem Backpropagationnetz ein Modell des Rewards, ergibt sich bereits nach rund 1 200 Lernschritten das in Abbildung 4.13a gezeigt Bild: Die Rewardfunktion ist sehr

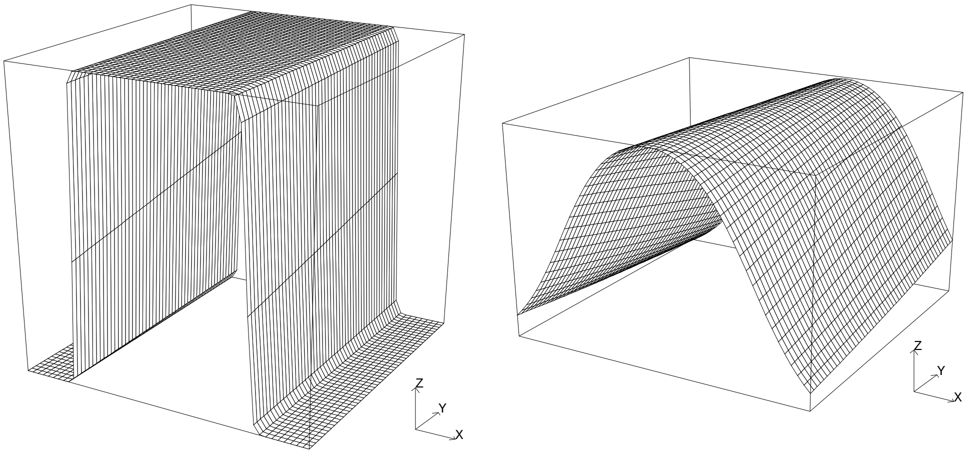


Abbildung 4.13: Beispiel für eine gelernte Rewardfunktion im Pole Balancing-Problem, (a) ohne und (b) mit weight decay. Die x-Achse mißt den Winkel des Stabes θ, die y-Achse die Position des Wagens x, und die z-Achse das erwartete Reward R. Die ursprüngliche Rewardfunktion ist binär. Zwar approximiert (a) diese binäre Funktion deutlich besser, doch ist (b) besser geeignet für Optimierung mit Gradientenabstieg, da hier die Gradienten nicht fast ausschließlich 0 sind. Die gezeigte Funktion modelliert die Rewardkomponente R_θ; ähnliches gilt für das Modell von R_x.

gut approximiert, und nur in einem minimalen Randbereich ist die Ausgabe des Netzes nicht approximativ -1 oder 1. Dies bedeutet aber, daß der Gradient dieser Funktion, wie auch der der ursprünglichen Rewardfunktion, fast überall 0 ist. Sucht man also mit Gradientenabstieg das Reward zu maximieren, wird man vom Gradienten keine Information erwarten können und scheitern: Nahezu alle Gradienten sind faktisch 0, was keine Änderung des Planes zur Folge hätte. Diese Eigenschaft ist aber nicht spezifisch für die verwendeten Approximatoren, sondern für das Problem binären bzw. diskreten Rewards.

4.4.1 Glätten der Rewardfunktion mit Weight Decay

Man kann das Problem diskreten Rewards im Falle des Pole Balancing-Problems lösen, wenn man nach einem ungenauen Modell mit informativeren Gradienten sucht. Eine in der Literatur oft anzufindende Technik zum Glätten der von Künstlichen Neuronalen Netzen erzeugen Funktionen ist weight decay. Unter weight decay versteht man eine Technik, die Gewichte und Schwellwerte klein hält. Große Steigungen, wie die in Abbildung 4.13a gezeigten, sind eine direkte Folge großer Gewichte, während kleine Gewichte verhältnismäßig glatte Funktionen liefern. Realisiert wird weight decay in Künstlichen Neuronalen Netzen durch die Erweiterung der Fehlerfunktion E um einen differenzierbaren Term, der die quadratische Abweichung der Gewichte und Schwellwerte zu 0 mißt:

$$E_{wd} \equiv \sum |w_{ijn}|^p \quad (4.6)$$

Abbildung 4.14: Die beiden verwendeten Modellnetze im Pole Balancing-Problem: Das *Aktionsmodell* (links) modelliert die Zustandsübergangsfunktion der Welt. Das *Rewardnetz* (rechts) sagt das Reward für einen Zustand voraus.

---

mit einem beliebigen Wert für den Exponenten p. Die resultierende, erweiterte Version von Backpropagation minimiert dann eine gewichtete Summe des Fehlers der Trainingsmenge und E_{wd} gleichzeitig. Wendet man kubisches ($p = 3$) weight decay auf die Rewardvorhersage im Pole Balancing-Problem an, erhält man exemplarisch das in Abbildung 4.13b gezeigte Modell von R_θ (diese Argumentation läßt sich direkt auf das Modell von R_x übertragen). Dieses Modell ist zwar deutlich ungenauer, doch erlaubt es das Finden von guten Aktionen mit Gradientenabstiegssuche. Diese Beobachtung führt zu dem nicht intuitiven Resultat, daß Planen mit genaueren Modellen nicht immer bessere Aktionen zur Folge hat, falls man Suchheuristiken wie Gradientenabstieg verwendet. Ohne weight decay ist das Pole Balancing-Problem mit Gradientenabstieg schlicht nicht lösbar.

4.4.2 Simulationsergebnisse

In unseren Simulationen des Pole Balancing-Problems wurden die in Abbildung 4.14 gezeigten Netze verwendet. Dabei ist das Modell, analog zu den weiter oben präsentierten Roboterarmexperimenten, in zwei Netze unterteilt: Das Aktionsmodell modelliert die Zustandstransition der Welt, wobei die Eingabe der vierdimensionale Zustandsvektor $(x(t), \dot{x}(t), \theta(t), \dot{\theta}(t))$ zusammen mit der Aktion F, und die Ausgabe der Folgezustand $(x(t+\tau), \dot{x}(t+\tau), \theta(t+\tau), \dot{\theta}(t+\tau))$ ist. Das Rewardmodell sagt für den Folgezustand das zu erwartende Reward (R_θ, R_x) voraus. Dabei wurde das Aktionsmodell mit dem unmodifizierten Backpropagationalgorithmus, das Rewardmodell jedoch mit Backpropagation und weight decay trainiert. Ein Erwartungsnetz gab es in diesem Experiment nicht – Pläne wurden grundsätzlich mit der Nullaktion $F = 0$ initialisiert. In unseren

Pole Balancing-Problem – Resultate für verschiedene Planlängen (Horizonte)								
Initialkonfiguration		Horizont N						
x $\in [-2.4; 2.4]$	θ $\in [-0.21; 0.21]$	1	2	3	4	5	7	10
$\dot{x} = 0$	$\dot{\theta} = 0$							
-2.076	0.022	1334	706	811	>20 000	>10 000	>10 000	>10 000
-0.993	0.011	276	301	1266	>10 000	>10 000	>10 000	>10 000
-0.652	-0.035	258	269	335	1547	582	>10 000	>10 000
0.000	0.000	313	330	367	367	662	>10 000	>10 000
-0.872	0.053	286	311	389	1579	>10 000	952	884
-0.892	0.114	307	361	409	617	912	617	475
-0.149	0.055	313	342	391	499	670	499	710
0.413	-0.131	296	287	302	347	357	726	434
2.145	-0.112	100	157	207	256	336	256	1531
-0.676	-0.183	190	202	190	216	223	231	227
1.843	0.179	111	82	87	77	77	77	61
Durchschnitt		344	304	432	–	–	–	–

Tabelle 4.1: Ergebnisse für Pole Balancing-Problem für verschiedene initiale Konfigurationen und Planhorizonte. Der Effekt des Planens ist deutlich zu sehen. Zum Vergleich: Zufallsaktionen halten den Stab für durchschnittlich 10 Schritte in der Balance.

Experimenten waren wir insbesondere daran interessiert, wie weit man in die Zukunft vorausplanen muß, um den Stab perfekt balancieren zu können. Tabelle 4.1 faßt eine Studie des simulierten Pole Balancing-Problem für verschiedene Planlängen zusammen. In dieser Testreihe war die Rewardgewichtungsfunktion $G(\tau) = 0.9^{\tau}$ (vergleiche Gleichung (4.2)) eine exponentiell fallende Funktion. Beide Netze, das Aktionsmodell und das Rewardmodell, wurden dabei jeweils mit 8 192 synthetisch erzeugten Trainingsbeispielen im Voraus trainiert. Mit diesen verhältnismäßig großen Trainingsmengen wurde sichergestellt, daß die von den Netzen gelernten Funktionen gute Approximationen der Welt repräsentierten. Für 11 verschiedene Startpositionen x und θ (mit $\dot{x} = 0$ und $\dot{\theta} = 0$) wurde die Zeit bis zum Umfallen des Stabes gemessen. Wir haben in Tabelle 4.1 die zufällig generierten Startpositionen danach geordnet, wie geeignet sie für das erfolgreiche Balancieren sind. Dabei kommt es bereits bei einem Horizont $N = 4$ vor, daß der Stab länger als 10 000 Schritte in der Luft bleibt. Zum Vergleich: Zufällige Aktionen balancieren den Stab für durchschnittlich rund

10 Schritte.

Diese Resultate demonstrieren, daß das simulierte Pole Balancing-Problem mit reinem Lookahead Planen lösbar ist, ist das Modell hinreichend genau. Will man den Stab unendlich lange balancieren, ist es von entscheidender Bedeutung, daß das Modell nicht (oder nicht ausschließlich) während des Balanciervorganges mit den dabei gemachten Beobachtungen trainiert wird. Dies liegt daran, daß mit steigender Performanz die Trainingsbeispiele mit negativem Reward immer seltener werden. Im Limit, in dem der Stab beliebig lange balanciert wird, haben alle Trainingsbeispiele für das Rewardnetz ausschließlich positives Reward. Trainiert man das Rewardnetz mit diesen Trainingsbeispielen, verliert es früher oder später die Fähigkeit, negatives Reward vorauszusagen. Dies bedeutet aber, das der Planer keine geeigneten Aktionssequenzen mehr generieren kann! In unserem Experiment wurden daher diese Netze mit einer statischen Trainingsmenge trainiert und während der Aktionsgenerierung nicht mehr verändert.

Planen vermag, beliebige Aktionen (Kräfte) im Intervall $[-10\mathrm{nt}, 10\mathrm{nt}]$ zu generieren. In dem hier vorgestellten Pole Balancing-Problem waren allerdings nur zwei Aktionen definiert: $-10\mathrm{nt}$ und $10\mathrm{nt}$ (*bang-bang-control*). Dabei wurde während der Suche reellwertige Aktionen in $[-10\mathrm{nt}, 10\mathrm{nt}]$ zugelassen, und anschließend die resultierende Aktion zu $-10\mathrm{nt}$ oder $10\mathrm{nt}$ gerundet. Grundsätzlich wird das Pole Balancing-Problem durch die Reduktion des Suchraums auf zwei Aktionen erleichtert. Es sei angemerkt, daß wegen des endlichen Aktionsraums leicht andere Suchtechniken, z.B. erschöpfendes Aufzählen, hätten verwendet werden können, und dies nicht notwendigerweise eine Erweiterung der Rewardfunktion erfordert hätte. Wir haben in unseren Versuchen Gradientenabstieg den Vorzug gegeben, da wir prinzipiell an der Erweiterung von Gradientenabstiegssuche auf diskrete Rewardfunktionen interessiert sind, und das Pole Balancing-Problem dabei nur die Rolle eines Beispiels spielt.

4.5 Literatur

Planen ist traditionell ein Begriff des Fachgebietes „Künstliche Intelligenz" (KI). Dort ist er in den letzten 35 Jahren intensiv untersucht worden. Hinter Planen verbirgt sich heute ein Sammelsurium unterschiedlichster Plantechniken. Dabei seien neben vielen anderen Arbeiten die Werke von Nils Nilsson [Nilsson, 1982] und das exzellente deutschsprachige Einführungswerk von Joachim Hertzberg [Hertzberg, 1989] empfohlen. Planen ist auch im Bereich der Kontrolliteratur unter dem Namen *online design* bekannt [Mayne und Michalska, 1990].

Typischerweise beschränkt sich die Planen-Literatur der KI hauptsächlich auf Anwendungen, in denen vollständiges und korrektes Weltwissen verfügbar sind, d.h. die Modelle perfekt sind. Anwendungen mit induktiv erlernten Weltmodellen finden sich vergleichsweise selten, da Modellbildung (Lernen) und Planen als orthogonal angesehen werden. Dabei untersucht die Literatur meist einen hier völlig vernachlässigten Aspekt von Planen: Die Komplexität des Suchraums. Gradientenabstieg oder andere Hill-climbing Algorithmen sind in der Regel nicht in der Lage, interessante Probleme der KI zu lösen.

Kawato war, nach Wissen des Autors, der erste, der Planen mit Neuronalen Modellen im Bereich Roboterarmsteuerung systematisch untersucht hat [Kawato *et al.*, 1990] [Kawato, 1990]. Sutton hat Planen im Kontext von Reinforcement Learning untersucht. Seine DYNA-Architektur integriert Reinforcement Learning und Planen [Sutton, 1990].

Die in diesem Kapitel geschilderten Techniken und Experimente gehen auf Thrun/Möller/Linden zurück [Thrun *et al.*, 1990], [Thrun *et al.*, 1991b]. Dabei basiert die Motivation, Gradientensuche zum Planen zu verwenden, auf Linden/Kindermanns Technik der „Netzinversion“ [Linden und Kindermann, 1989] [Kindermann und Linden, 1990], [Thrun und Linden, 1990]. Linden und Kindermann untersuchten die Gradientensuche im Eingaberaum eines Netzes im Kontext von Mustererkennung. Die Roboterarmexperimente wurden nach Entstehen dieser Arbeiten mit dem Roboterarm der Universität Bonn zusammen mit Knut Möller, Roman Englert, Stefan Hoffrichter und Frank Kimmlingen getestet. Eine deutschsprachige Beschreibung findet sich in [Möller und Thrun, 1991]. Das Pole Balancing-Problemist zum Beispiel in [Anderson, 1989], [Wieland, 1990] beschrieben. Des weiteren kann man die hier beschriebenen Roboterarmexperimente auch in Knut Möllers Promotion [Möller, 1991] nachlesen.

Die in diesem Kapitel geschilderten Techniken und Experimente sind nahezu vollständig in [Thrun und Möller, 1991] enthalten. In diesem Bericht wird überdies eine Technik beschrieben, Planhorizonte dynamisch während des Planungsvorganges zu bestimmen.

Kapitel 5

Exploration

In diesem Kapitel wollen wir uns einer Problemstellung widmen, die allen Adaptiven Kontrollproblemen gemein ist: *Exploration*. Was ist Exploration? Wie wichtig ist Exploration?

Backpropagation, so wie wir es in den beiden ersten Kapiteln vorgestellt haben, ist ein Algorithmus, der eine fest vorgegebene Trainingsmenge bezüglich eines Fehlermaßes approximiert. Dies ist kohärent mit vielen im Bereich „Maschinelles Lernen“ studierten induktiven Lernproblemen: Ziel des Lerners ist es, eine statische Menge zu approximieren und die bestmögliche Generalisierung zu finden. Die Verteilung der Trainingsmenge und der Generalisierungsmenge werden dabei als identisch angenommen. Solche Approximationsprobleme werden oft als *passives Lernen* bezeichnet. Unterschiedliche induktive Lernprogramme, sei es Backpropagation, seien es Entscheidungsbäume (*decision trees, ID3*), unterscheiden sich im wesentlichen in der Art, wie sie die beobachteten Trainingspunkte zu einer vollständigen Funktion fortsetzen. Man spricht dabei von unterschiedlichen *induktiven Biases* – hinter diesem Begriff verbergen sich die „Regeln“, die das Fortsetzen von Trainingspunkten zu einer Funktion definieren.

In Kontrollproblemen ist die Situation komplexer als in statischen, passiven Lernproblemen: Der Lerner hat selbst Einfluß auf die Selektion und Verteilungsfunktion der Trainingsbeispiele. Je nach generierten Aktionen tragen die resultierenden Zustandsübergänge mehr oder weniger Information. Darüber hinaus ist das Ziel von Optimaler Kontrolle die Maximierung des Rewards und nicht die beste Generalisierung. Als Beispiel betrachte man einen autonomen Roboter, der sich von einem Raum A in einen Raum B manövrieren soll, ohne dabei an Hindernisse zu stoßen. Man sieht leicht, daß die Wahl der Aktionen den potentiellen Informationszuwachs nachhaltig beeinflussen kann: Bewegt sich der Roboter nicht (oder nur auf der Stelle), sieht er nach kurzer Zeit nichts Neues und kann zwar optimale Voraussagen treffen, aber keinen geeigneten Kontroller zum Lösen des Navigationsproblems finden. Nur das Explorieren unbekannter Teile des Zustandsraums sichert den Erfolg, auch wenn dies in diesem Beispiel das Risiko von Kollisionen mit sich bringt.

Es gibt in der Literatur eine Vielzahl von Explorationsstrategien, die alle mehr oder weniger die asymptotisch erschöpfende Exploration unbekannter Zustands-Aktionsräume sichern. Das wohl beliebteste, bei weitem aber nicht effizienteste Schema ist Zufallsexploration. Hier werden

Aktionen per Zufall generiert, um aus den beobachteten Folgen zu lernen. Natürlich kann das zufällige Generieren von Aktionen teuer sein, versucht man, das Reward bereits zur Lernzeit zu optimieren. In solchen Szenarios spricht man vom *Tradeoff zwischen Exploration und Exploitation*: Zum einem führt reine Exploration zu geringem Reward, zum anderen führt reines Optimieren von Reward zu schlechten Lernerfolgen, was sich letztlich auch negativ auf das Reward auswirkt. Will man Langzeitreward optimieren, muß man teilweise explorieren, teilweise optimieren, aber wie?

Wir werden in diesem Kapitel zuerst einige theoretische Resultate zum Thema Exploration wiedergeben. Wir werden zeigen, daß bestimmte Algorithmen, die unter dem Sammelbegriff *gerichtete Explorationstrategien* (*directed exploration*) zusammengefaßt sind, signifikant schneller sein können als z.B. Zufallsexploration (*undirected exploration*). Danach werden wir eine neuronale Architektur, die zielgerichtete Exploration ermöglicht, präsentieren. Diese Architektur bedient sich eines *Konfidenznetzes*, das modelliert, wie gut der Zustandsraum bereits exploriert ist. Mit diesem Metamodell werden dann systematisch unexplorierte Zustände exploriert, was zur Maximierung des Wissenszuwachses führt. Um negatives Reward zu vermeiden, werden wir einen Aufmerksamkeitsmechanismus (*selective attention*) einführen, der es erlaubt, zwischen Exploration und Exploitation hin- und herzuschalten. Der resultierende gerichtet explorierende Kontrollmechanismus wird schließlich anhand eines einfachen Roboterkontrollproblems evaluiert und und mit anderen Explorationstechniken verglichen.

5.1 Exploration in diskreten Räumen: Komplexitätsergebnisse

Die Gretchenfrage für die Bedeutung von Exploration in Adaptiver Kontrolle ist: *„Wie wirkt sich die Explorationsstrategie auf die Lernzeit und die Lernkosten aus?“* Wie werden im folgenden entscheidende theoretische Resultate im Bereich Exploration zusammenfassen.

5.1.1 Ungerichtete und gerichtete Exploration

Die beiden zentralen Prinzipien für Exploration sind: *ungerichtete Exploration* und *gerichtete Exploration*. Unter ungerichteter Exploration versteht man alle Strategien, bei denen die Exploration ausschließlich per Zufall durchgeführt wird. Dabei gibt es ein ganzes Spektrum solcher Strategien. So kann man zum Beispiel, wie im einleitenden Roboterbeispiel, alle Aktionen gleichwahrscheinlich machen. Nicht gleichförmige Wahrscheinlichkeitsverteilungen sind ebenfalls zugelassen, sofern die Wahrscheinlichkeiten (Dichte), mit der Aktionen gewählt werden, eine monotone Funktion des erwarteten Rewards darstellen. So gewichten viele ungerichtete Explorationsstrategien die Wahrscheinlichkeiten, mit der Aktionen ausgewählt werden, wobei viel Reward versprechende Aktionen mehr wahrscheinlich sind als wenig Reward versprechende Aktionen. Solche Verteilungen sind in der Regel effizienter. Im wesentlichen sind ungerichtete Explorationstechniken aber dadurch charakterisiert, daß Exploration auf der Basis von Zufall erfolgt.

Gerichtete Exploration umfaßt eine Klasse von Algorithmen, die, im Gegensatz zu ungerichteter Exploration, spezielles Metawissen verwenden, um die Exploration meist deterministisch zu steuern. Ein einfaches Beispiel hierfür sei die Regel *„Wähle die am wenigsten explorierte Aktion!“*. Eine solche Regel vermag die Welt zu explorieren, ohne daß Aktionen per Zufall generiert werden müssen. Grundsätzlich bedingt gerichtete Exploration ein Explorationsmodell (Metamodell), das bewerten kann, wie gut Aktionen und/oder Zustände exploriert sind. Gerichtete Explorationsstrategien unterscheiden sich untereinander u.a. in den verwendeten Explorationsmodellen, der Definition des Maßes der Exploriertheit und der Bewertung des erwarteten Wissenszuwachses.

5.1.2 Die Zeitkomplexität von ungerichteter und gerichteter Exploration

Steve Whitehead hat für das in Kapitel 3.3.2 vorgestellte Q-Learning bewiesen, daß unter bestimmten Bedingungen bei Verwendung von Zufallsexploration die Lernzeit eine *exponentielle* Funktion der Anzahl der Zustände ist.[1] Dieses Theorem läßt sich auf beliebige Lerntechniken und ungerichtete Exploration übertragen. Whiteheads Theorem lautet, übersetzt, verkürzt, und auf unsere Sprachweise zugeschnitten:

Theorem (Whitehead):

Unter den folgenden Bedingungen ist die Zeitkomplexität von ungerichteter Exploration exponentiell in der Anzahl der Zustände.

1. Reward ist überall 0, außer für einen einzelnen *Zielzustand*,
2. alle Aktionen ändern die *Distanz zum Zielzustand* (die minimale Anzahl von Aktionen, derer es bedarf, die Welt in den Zielzustand zu versetzen) nur um $+1$, 0 oder -1, und Zufallsaktionen vergrößern im Mittel diese Distanz, und
3. die Größe des Zustandsraums ist polynomial in seiner *Tiefe*, die maximale Distanz aller Zustände zum Zielzustand.

In Whiteheads Beweis wird lediglich die Explorationskomplexität von Zufallsexploration evaluiert, die eine untere Schranke für das dort betrachtete Q-Learning darstellt. Zufallsexploration, so zeigt Whitehead, kann in ungünstigen Welten exponentiell mit der Anzahl der Zustände wachsen. Da beliebige ungerichtete Explorationstrategien in den betrachteten Welten ebenfalls mit Zufallsexploration starten, läßt sich das Theorem auf beliebige ungerichtete Techniken übertragen, und ist somit ein allgemeines Komplexitätstheorem für ungerichtete Exploration. Dieses ist im übrigen unabhängig von der verwendeten Lerntechnik des Kontrollers.

[1] Dieses Theorem gilt nur in endlichen Zustandsräumen in deterministischen Welten, da andere Welten schwerer zu analysieren sind. Außerdem müssen die Zustandsräume *ergodisch* sein, d.h., jeder Zustand muß von jedem anderen prinzipiell erreichbar sein.

Die Bedeutung von gerichteter Exploration in Adaptiver Kontrolle wird durch das folgende duale Theorem klar: Die geeignete Explorationsregel sichert worst-case polynomiale Lernzeit (einschließlich in all den Welten, für die Whiteheads Theorem gültig ist).

Theorem:

Verwendet man die Explorationsregel *„Wähle den am wenigsten explorierten Nachbarzustand*[2]*!“* an, kann man immer in polynomialer Zeit $O(\#\text{states}^3)$ den Zustandsraum komplett explorieren. Das Erlernen eines Kontrollers ist ebenfalls in polynomialer Zeit möglich.

Der Beweis dieses Theorems findet sich im Anhang dieser Arbeit. Hinweise auf die Originalpublikationen sind in der Literaturübersicht zu finden.

5.1.3 Wie wichtig sind die Resultate?

Dieses Theorem, zusammen mit Whiteheads Theorem, verdeutlichen die entscheidende Rolle der Explorationsstrategie beim Lernen. Ungerichtete Explorationsstrategien können lernbare Probleme praktisch in unlernbare verwandeln. Eine typische Situation, für die diese Komplexitätsergebnisse relevant sind, ist in Abbildung 5.1 gezeigt. Der Zustandsraum besteht aus n diskreten Zuständen, und die Aufgabe des Roboters ist, eine Funktion zu finden, die ihn vom linken Zustand in den rechten manövriert. Dabei hat der Roboter – dies ist ein extrem simplifiziertes Beispiel – drei mögliche Aktionen: „gehe nach links“, „gehe nach links“ und „gehe nach rechts“. Man sieht leicht, daß zufällig gewählte Aktionen im Mittel die Distanz zum Ziel vergrößern. Zufallsexploration benötigt in diesem Beispiel im Mittel $3(2^n - n - 1)$ Schritte, um zum ersten Male Reward im Zielzustand rechts außen zu erlangen. Gerichtete Exploration, zum Beispiel die einfache Explorationsregel *„Wähle den am wenigsten explorierten Nachbarzustand“*, kommt dagegen mit $n - 1$ Schritten aus. Diese Regel ist gerichtet, da sie modelliert, wie oft Zustände aufgetreten sind. Für einen Zustandsraum mit nur $n = 50$ Zuständen ergeben sich im Mittel $3.38 \cdot 10^{15}$ Explorationsschritte für ungerichtete und 49 Explorationsschritte für gerichtete Exploration.

Sowohl das Beispiel, als auch die Theoreme unterstreichen die wichtige Rolle, die Exploration in Adaptiver Kontrolle spielt: Je nach Situation kann die Lernkomplexität polynomial (gerichtete Exploration) oder exponentiell (ungerichtete Exploration) in der Zahl der Zustände der Welt sein. Wie im obigen Beispiel gezeigt, kann dies bereits in einfachen Welten der Fall sein. Empirische Untersuchungen haben gezeigt, daß ungerichtete Explorationstechniken auch in Welten, die nicht Whiteheads Theorem genügen (z.B. in Roboternavigationsproblemen), als deutlich nachteilhaft gegenüber gerichteten Techniken erweisen. Es sei darauf hingewiesen, daß sich in einem überwiegenden Teil der Literatur im Bereich Neurocontrol und Reinforcement Learning

[2] mit einem Schritt erreichbarer Zustand

Abbildung 5.1: Eine eindimensionale Roboterwelt demonstriert den Komplexitätsunterschied zwischen *Zufallsexploration* (ungerichtete Exploration) und *gerichteter* Exploration. Drei Aktionen sind möglich: „gehe nach links", „gehe nach links", und „gehe nach rechts". Die *Distanz* zum Zielzustand (Zustand rechts außen) entspricht der Entfernung des Roboters vom rechen Rand. In diesem Szenario vergrößern zwei Aktionen diese Distanz, während nur eine Aktion diese verkleinert. Zufallsexploration benötigt im Mittel $O(2^n)$ Schritte, während die Regel „gehe zum am wenigsten explorierten Nachbarzustand" (gerichtete Exploration) mit $O(n)$ auskommt, um den Zielzustand zu finden.

---

ungerichtete Explorationsstrategien finden. In vielen dieser Arbeiten kann man zeigen, daß das Kontrollschema daher nicht für komplexere Probleme geeignet ist. Diese Beobachtung war Ausgangspunkt für die in diesem Kapitel zusammengefaßte Forschung im Bereich Exploration.

5.2 Neuronale Exploration

5.2.1 Wie gut ist was exploriert? – Das Konfidenznetz

Nachdem wir die Wichtigkeit der Explorationskomponente aufgezeigt haben, werden wir einen neuronalen Ansatz zur gerichteten Exploration vorstellen. Wir werden der Einfachheit halber Exploration im Kontext von indirekter Einschrittkontrolle untersuchen – der verwendete Kontrollalgorithmus ist in Kapitel 3.2.3 beschrieben.

Gerichtete Exploration beruht auf der Modellierung der Genauigkeit des Weltwissens. Dabei eignen sich Künstliche Neuronale Netze, wie im folgenden demonstriert wird. Zentrale Komponente gerichteter Exploration ist das Konfidenznetz. Dieses Netz stellt ein Metamodell dar, das modelliert, wie gut die Welt exploriert ist. Hat man ein solches Netz, kann man dort explorieren, wo der erwartete Fehler des Weltmodells groß, und somit der erwartete Wissenszuwachs ebenfalls groß ist. Ein solches Explorationsschema ist *gerichtet*, da Exploration nicht zufällig, sondern auf der Basis eines Metamodells geschieht. Wie erhält man ein geeignetes Konfidenznetz? In dem hier beschriebenen Ansatz wird das Konfidenznetz trainiert, den *erwarteten Fehler* des Modells der Welt vorauszusagen. Dies geschieht im wesentlichen durch Beobachtung und Modellieren von Voraussagefehlern des Modellnetzes. Das Mitchogramm 5.2 illustriert die Arbeitsweise des Konfidenznetzes: Wenn das Modellnetz eine Vorhersage trifft, wird diese mit der Welt verglichen und der Fehler E extrahiert. Dieser Fehler, zusammen mit den Eingaben des Modells (Zustand und Aktion), bildet nun ein Trainingsbeispiel für das Konfidenznetz. Während das Modellnetz lernt, den nächsten Zustand besser vorauszusagen, lernt das Konfidenznetz, den Fehler dieser Vorhersage vorherzusagen. In diesem Sinne approximiert das Modell den Erwartungswert und das Konfidenznetz die Varianz der Welt.

Abbildung 5.2: Das *Konfidenznetz* lernt, Fehler des Modells vorauszusagen. Die Eingabe ist identisch mit der des Modells, die Ausgabe ist jedoch die quadratische Abweichung der Modellausgabe und dem entsprechenden beobachteten Zustand. Das Konfidenznetz wird für gerichtete Exploration eingesetzt.

Im Falle von Backpropagation muß das Trainingsverfahren des Konfidenznetzes um eine weitere Komponente erweitert werden: Sei E^* der größtmögliche Fehlerwert des Modells. Für jedes Trainingsbeispiel des Konfidenznetzes wird eine Menge von $\lambda > 0$ zufälligen Trainingsbeispielen generiert, die jeweils die Zielausgabe (*target*) E^* haben. Der Grund hierfür liegt in der Tatsache, daß die Unsicherheit für selten beobachtete Zustände stetig steigt. Da folglich aber solche Zustände selten Trainingsbeispiele liefern, müssen künstlich Trainingsbeispiele generiert werden, die sichern, daß das Konfidenznetz einen hohen Fehler voraussagt. Dieser akademisch erscheinende Punkt ist besonders wichtig, wenn Backpropagationnetze angewendet werden, da Backpropagation kein lokaler, sondern ein stark extrapolierender Funktionsapproximator ist. Dies bedeutet, daß eine Änderung des Funktionswert in einer Region des Eingaberaums die Ausgabe im gesamten Eingaberaum ändern kann, ohne daß dafür ein explizites Trainingsbeispiel vorliegt.

5.2.2 Exploration und Exploitation

In Kapitel 3.2.3 haben wir beschrieben, wie Reward-maximierende Aktionen durch Suche im Eingaberaum eines Modellnetzes generiert werden können. Sei $E_{\text{Exploitation}}$ die Optimierungsfunktion, die die Distanz zwischen vorausgesagtem Reward und dem bestmöglichen mißt. Der Begriff Exploitation bezeichnet dann die Aktionsgenerierung unter Maximierung von $E_{\text{Exploitation}}$. Analog läßt sich die Exploration maximieren. Sei $E_{\text{Exploration}}$ die quadratische Abweichung vom vorausgesagtem Fehler und dem größtmöglichen Fehler. Exploration und Exploitation wird kombiniert, wenn man die Linearkombination

$$J \;=\; \Gamma\, E_{\text{Exploitation}} \;+\; (1-\Gamma)\, E_{\text{Exploration}} \tag{5.1}$$

minimiert. Dabei gibt der tradeoff-Faktor Γ ($\leq \Gamma \leq 1$) das relative Gewicht von Exploitation und Exploration an.

Es liegt nun nahe, einen festen Wert für Γ zu wählen und somit gleichzeitig zu explorieren und Reward zu maximieren. Prinzipiell ist dies eine gute Strategie, doch birgt eine solche Technik ihre Nachteile. Was passiert zum Beispiel, wenn Exploration und Exploitation genau entgegengesetzt sind? In diesem Falle heben sich Exploration und Exploitation gegenseitig auf, und die resultierende Aktion erzielt weder einen Explorations-, noch einen Exploitationseffekt. Das Problem läßt sich am besten durch ein einfaches Beispiel veranschaulichen: Ein Zoologe in der Wüste, der seinen ersten Löwen in freier Wildbahn sieht, mag geneigt sein, wegzurennen (Exploitation), aber auch, den Löwen näher zu untersuchen (Exploration). Eine fixe Linearkombination beider Prinzipien kann zu keiner Entscheidung führen. Der Forscher würde dann stehen bleiben, wo er ist, was sicher nicht die optimale Lösung des Dilemmas darstellt. Ausgehend von dieser Beobachtung macht es Sinn, Exploration und Exploitation dynamisch zu kombinieren und zu jedem Zeitpunkt entweder zu explorieren oder Reward zu maximieren.

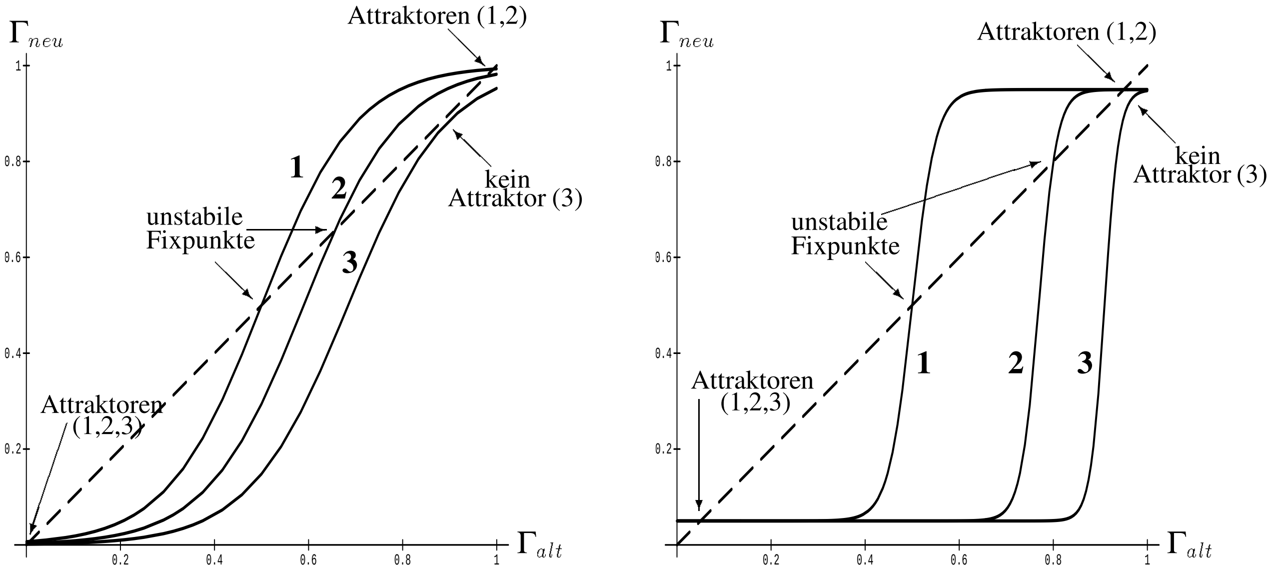
5.2.3 Selektive Aufmerksamkeit

Selektive Aufmerksamkeit (*selective attention*) ist eine Lösung für dieses Entscheidungs- bzw. Gewichtungsproblem. Die Grundidee hierbei ist folgende: Der Kontroller hat im wesentlichen zwei Zustände: entweder Γ ist groß (nahe 1), und der Kontroller maximiert das Reward, oder Γ ist klein (nahe 0), und der Kontroller exploriert. Übersteigen die erwarteten Kosten während der Exploration eine bestimmte Grenze, ändert der Kontroller seinen Zustand und maximiert Reward. Ist dagegen der erwartete Wissenszuwachs während der Exploitation größer als eine bestimmte Grenze, schaltet der Kontroller in den Explorationsmodus um.

Wir werden im folgenden einen Ansatz vorstellen, der diese Idee aufgreift und verallgemeinert. Seien $e_{\text{Exploration}}$ der *erwartete Wissenszuwachs* und $e_{\text{Exploitation}}$ das *erwartete Reward*. Dabei wird $e_{\text{Exploration}}$ mit dem Konfidenznetz und $e_{\text{Exploitation}}$ mit dem Modell vorausgesagt. Der Wert κ mit

$$\kappa \;\longleftarrow\; \Gamma \cdot f(e_{\text{Exploitation}}) \;-\; (1-\Gamma) \cdot f(e_{\text{Exploration}}) \tag{5.2}$$

vergleicht den Exploitations- mit dem Explorationseffekt *unter dem derzeitigen Fokus der Aufmerksamkeit* Γ. Hier ist $f(\cdot)$ eine strikt monotone positive Funktion (in unseren Experimenten wurde stets $f(z) = e^z$ gewählt). Ist κ positiv, dominiert Exploitation Exploration, und der Kontroller maximiert Reward. Ist κ negativ, liegt die umgekehrte Situation vor, und der Kontroller exploriert. Folglich kann κ zur Berechnung von Γ verwendet werden, indem wir die aus den ersten beiden



$c = 3$

Kurve	$f(e_{\text{Exploitation}})$	$f(e_{\text{Exploration}})$
1	1.0	1.0
2	0.8	1.15
3	0.6	1.3

$c = 20$

Kurve	$f(e_{\text{Exploitation}})$	$f(e_{\text{Exploration}})$
1	1.0	1.0
2	0.6	2.0
3	0.3	3.0

Abbildung 5.3: Beispiele für die Übergangsfunktion für den Aufmerksamkeitsparameter Γ. Der Stabilitätsfaktor c ist (a) $c = 3$ und (b) $c = 20$. Fixpunkte befinden sich auf der gestrichelten Linie. In beiden Diagrammen haben Graph 1 und 2 jeweils zwei Attraktoren, was bedeutet, daß Γ stabil ist. Wird der Unterschied zwischen $e_{\text{Exploration}}$ und $e_{\text{Exploitation}}$ allerdings zu groß, geht ein Attraktor verloren, und Γ wandert, falls nicht schon dort, zum anderen Fixpunkt.

Kapiteln bekannte logistische Squashingfunktion auf κ anwenden:

$$\Gamma \longleftarrow \frac{1}{1 + e^{-c \cdot \kappa}} \tag{5.3}$$

$c > 0$ ist dabei eine Konstante (die Stabilitätskonstante), die die Stabilität des Systems regelt. Abbildung 5.3 zeigt die Übergangsfunktion von Γ über κ zum neuen Γ. Diese Funktion hat, solange $e_{\text{Exploration}}$ und $e_{\text{Exploitation}}$ in einem ausgewogenen Verhältnis sind, zwei Attraktoren (stabile Fixpunkte). Geraten $e_{\text{Exploration}}$ und $e_{\text{Exploitation}}$ allerdings aus dem Lot, geht einer der Attraktoren verloren, und das System ändert seine Aufmerksamkeit. Der Initialwert für Γ ist 0.5.

Die Definition der Übergangsfunktion für Γ (5.2) und (5.3) erscheint ein wenig ad hoc. In der Tat manifestieren diese Funktionen nur ein Beispiel aus vielen denkbaren. Die hier vorgestellten Funktionen sind komplexer als zum Beispiel einfache Schwellwerte, doch bieten sie einige Vorteile, da sie reellwertig und träge sind.

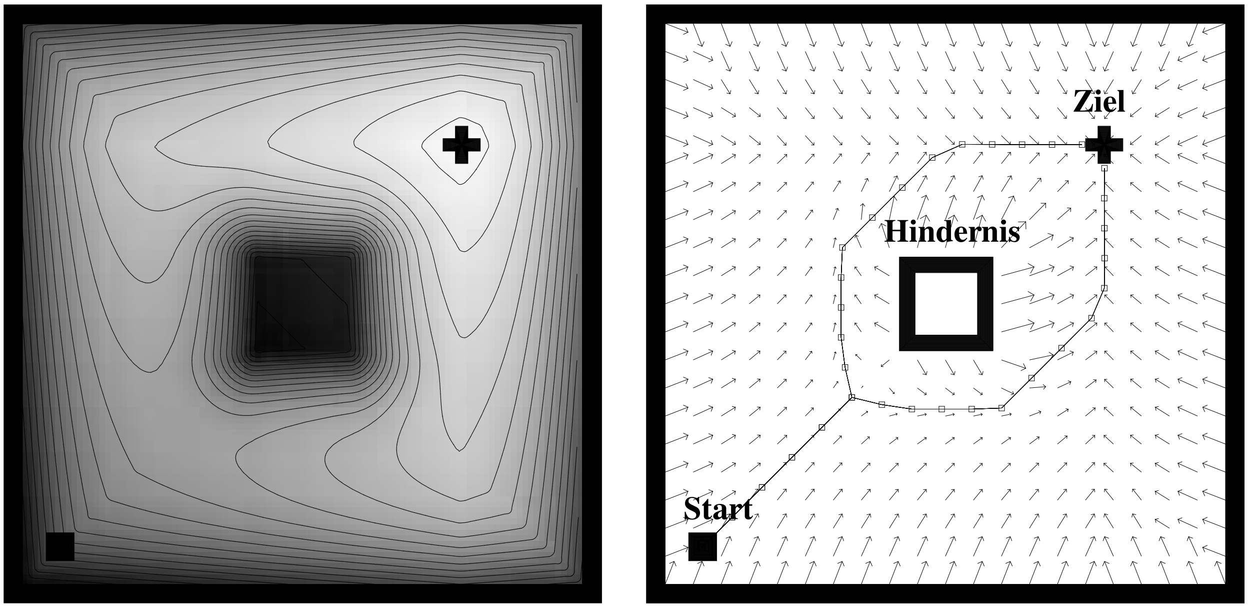


Abbildung 5.4: (a) Die simulierte Roboterwelt: Ziel des Roboters ist es, von der Startposition zur Zielposition zu manövrieren und dabei weder mit dem Hindernis in der Mitte, noch mit den Wänden zu kollidieren. Der Graulevel symbolisiert das Reward (*potential field*): Je heller das Grau, desto größer ist das Reward. Das Erlernen der Rewardfunktion ist dabei Teil der Aufgabe. (b) Gradientenabstieg in der Rewardfunktion liefert optimale Pfade zum Zielpunkt. Die Pfeile zeigen die Richtung des Steilsten Abstieges an. Zusätzlich sind die beiden optimalen Pfade gezeigt.

5.3 Beispiel: Autonome Roboternavigation

Wir wollen im letzten Teil dieser Arbeit das Zusammenspiel von Exploration und Kontrolle, das Zusammenspiel von Konfidenznetz und Modellnetz, im Rahmen einer simulierten Roboternavigation untersuchen. Die Robotertestwelt ist in Abbildung 5.4 gezeigt. Ziel ist es, den Roboter von der Startposition in der linken unteren Ecke in die Zielposition in der rechten oberen Ecke zu manövrieren, ohne dabei das Hindernis in der Mitte oder die Außenwände zu berühren. Geschieht dies trotzdem, wird die Sequenz beendet und der Roboter neu gestartet.

Zustände $s = (x, y)$ sind als x-y-Koordinaten in $[0, 1]^2$, und Aktionen $a = (\Delta x, \Delta y)$ sind als Koordinatenänderungen repräsentiert. Die Zustandsübergangsfunktion ist die Summe aus dem vorangehenden Zustand und der Aktion (und somit trivial). Reward R dagegen ist eine nicht-lineare *Potentialfeldfunktion* (*potential field function*), die in Abbildung 5.4a gezeigt ist. Dieses Potentialfeld mißt die Distanz zum nächsten Hindernis/Wand, sowie die Distanz zum Ziel. Maximierung des Rewards führt den Roboter von beliebigen Positionen zum Ziel. Aufgabe des Kontroller ist es, ein akkurates Modell der Zustandsübergangsfunktion und der Rewardfunktion zu erlernen, um so optimale Kontrolle zu ermöglichen.

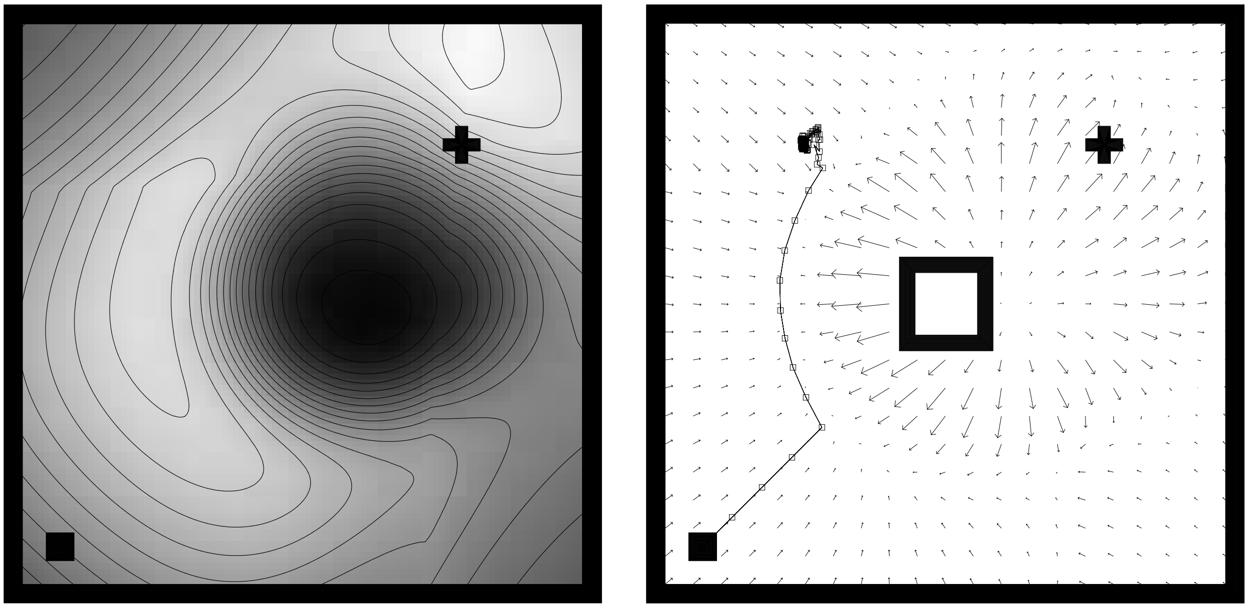


Abbildung 5.5: Zufallsexploration: (a) Das resultierend Modell der Rewardfunktion. Die vergleichsweise dunkle Farbe des Hindernisses rührt von zahlreichen Kollisionen seitens des Roboters. (b) Benutzt man den resultierenden Kontroller zur Rewardmaximierung, findet man keinen der beiden Wege zum Ziel.

Folgende Netze wurden verwendet: Das Modellnetz war in zwei Teile aufgeteilt, je ein Netz für die Zustandsübergangsfunktion und das Reward. Das Zustandsübergangsmodell war ein Einschichtnetz mit vier Eingabe- und zwei Ausgabeknoten. Das kompliziertere Rewardnetz hatte zwei Eingabeknoten (der vorhergesagte Zustand war Eingabe für dieses Netz), eine interne Schicht mit zehn Knoten mit Radial-Basis Aktivationsfunktionen (vergleiche Kapitel 2.1.4), eine zweite interne Schicht mit acht Knoten, und einem Ausgabeknoten für das vorauszusagende Reward. Schließlich hatte das verwendete Konfidenznetz vier Eingabeknoten (Zustand und Aktion), zwei interne Schichten mit jeweils sechs Knoten und einem Ausgabeknoten für den vorauszusagenden Fehler. Ziel des im folgenden beschriebenen Experimentes ist es, drei Explorationstechniken zu vergleichen: Zufallsexploration, reine Exploitation und gerichtete Exploration mit einem Konfidenznetz. Die im folgenden beschriebenen Resultate sind in Tabelle 5.1 zusammengefaßt:

Zufallsexploration (mit gleichverteilten Wahrscheinlichkeiten für alle Aktionen): Diese Explorationsform schneidet extrem schlecht ab. Das Modell ist ungenau und unbrauchbar für das Problem. Dies liegt nicht zuletzt daran, daß wir ein für Zufallsexploration schlecht geeignetes Problem gewählt haben. Schließlich macht das Zurücksetzen der Roboters zum Startpunkt bei Kollisionen das Finden des Zieles äußerst unwahrscheinlich. Ergo stammen die meisten Trainingsbeispiele für das Modell von Positionen in der Nähe des Startpunktes. Dieses Beispiel demonstriert, daß Zufallsexploration keine Gleichverteilung der Zustände impliziert. Abbildung 5.4 zeigt das Modell nach 15 000 Trainingsschritten. Wie dort gezeigt, ist dieses Modell unbrauchbar für Kontrolle.

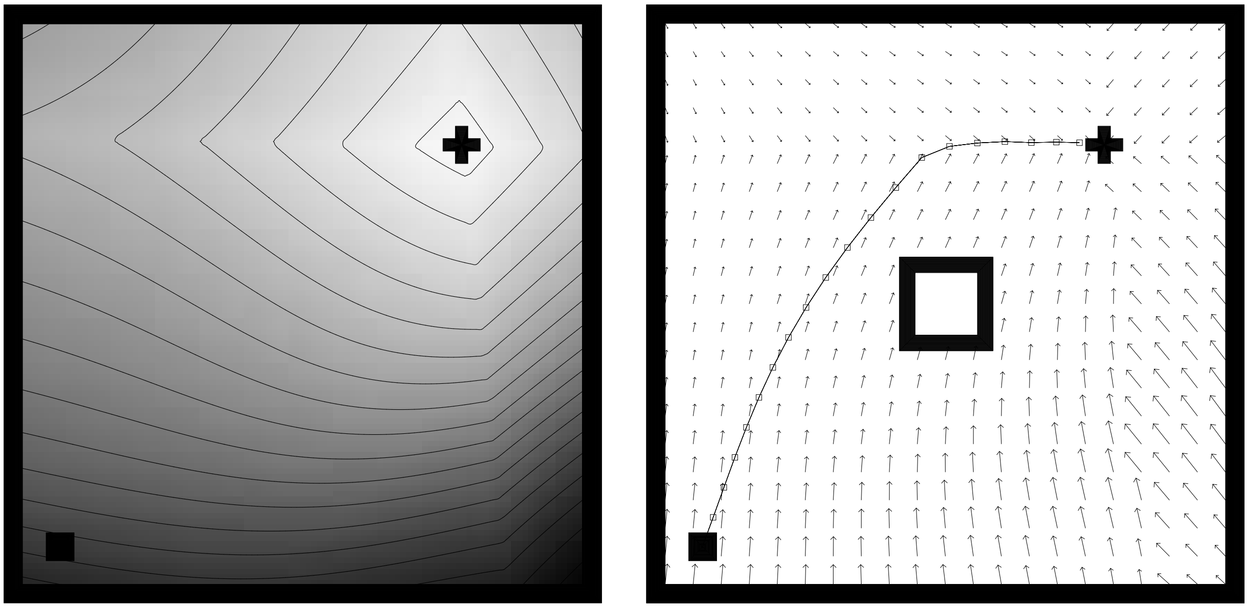


Abbildung 5.6: Reine Exploitation (Rewardmaximierung): (a) Zwar ist das resultierende Modell äußerst ungenau, doch (b) erlaubt es die Generierung von zumindest einem Pfad zum Ziel. Der andere Pfad ist allerdings völlig unbekannt.

Exploitation: Das nächste untersuchte Schema ist eine konservative Kombination aus Exploitation und Zufallsexploration. Dabei wurde im wesentlichen reine Exploitation zur Aktionsfindung angewendet – die einzige Ausnahme, in der per Zufallsaktionen exploriert wurde, war gegeben, wenn Exploitation die Nullaktion $a = (0, 0)$ lieferte[3]. Dann nämlich, und das ist wichtig für die ersten Lernschritte, würde sich der Roboter nicht von der Stelle bewegen, und nichts lernen. Abbildung 5.6 zeigt das gelernte Modell nach ebenfalls 15 000 Schritten. Dieses Ergebnis ist bereits erfreulicher: Ein Weg zum Ziel kann generiert werden, und das Modell ist korrekt entlang dieses Weges. Man beachte, daß nach Tabelle 5.1 diese Methode die Anzahl der Kollisionen nicht minimiert, obwohl erklärtes Ziel von Exploitation die Maximierung des Rewards ist. Dies demonstriert, daß ein Adaptiver Kontroller zum Maximieren des Rewards auch explorieren muß.

Exploration mit Konfidenznetz und selektiver Aufmerksamkeit: In unseren Versuchen verwendeten wir den Stabilitätsfaktor $c = 100$, $\lambda = 5$ und $f(z) = e^z$ (vergleiche Gleichungen (5.2) und (5.3)) und die bereits beschriebenen Netze. Dies führte zu dem in Abbildung 5.7 gezeigtem Modell. Dieses Modell ist das genaueste unter den drei getesteten Algorithmen, und beide Pfade zum Ziel können generiert werden. Abbildung 5.8 demonstriert ein „typisches“ Konfidenznetz, sowie einige während des Lernens generierte Trajektorien. Es ist deutlich zu sehen, wie das System zwischen Exploration und Exploitation wechselt, um zum einem Kollisionen zu vermeiden und zum anderen Wissen zu maximieren.

[3]genauer: $|a| < \varepsilon$ für ein sehr keines $\varepsilon > 0$

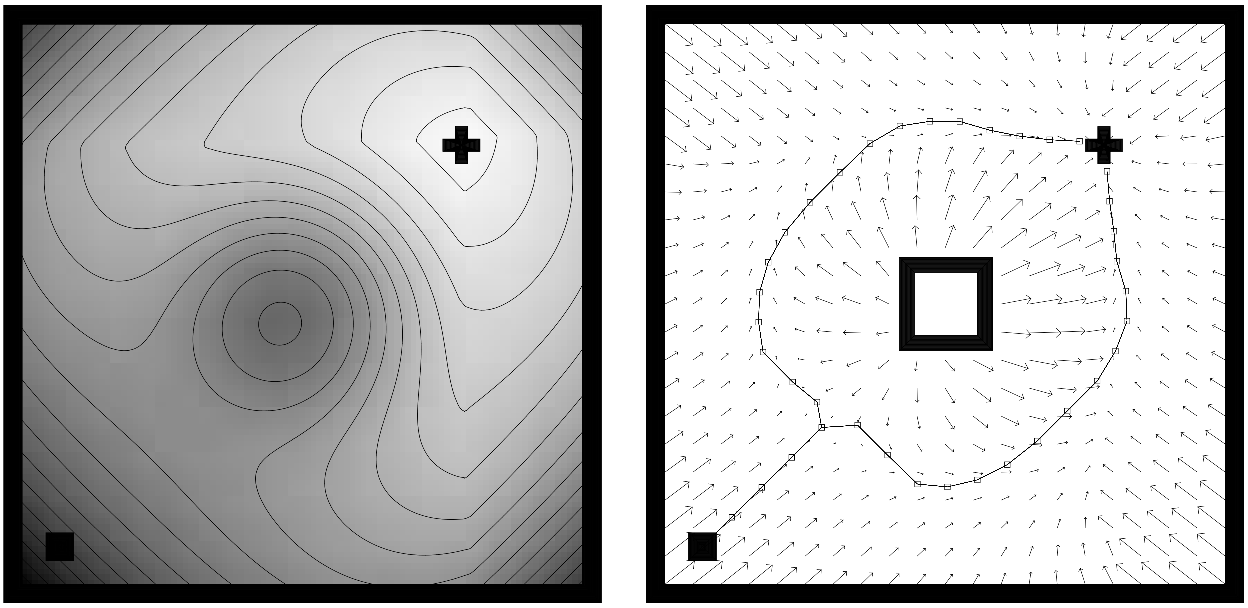


Abbildung 5.7: Exploration und Exploitation mit Konfidenznetz und selektiver Aufmerksamkeit, wie im Text beschrieben: (a) Das resultierende Modell ist deutlich genauer als das durch die beiden anderen Techniken gefundene, und (b) beide Pfade können generiert werden. Da der selektive Aufmerksamkeitsmechanismus zwischen Exploitation und Exploration wechselt, werden beide Pfade ungefähr gleich oft begangen.

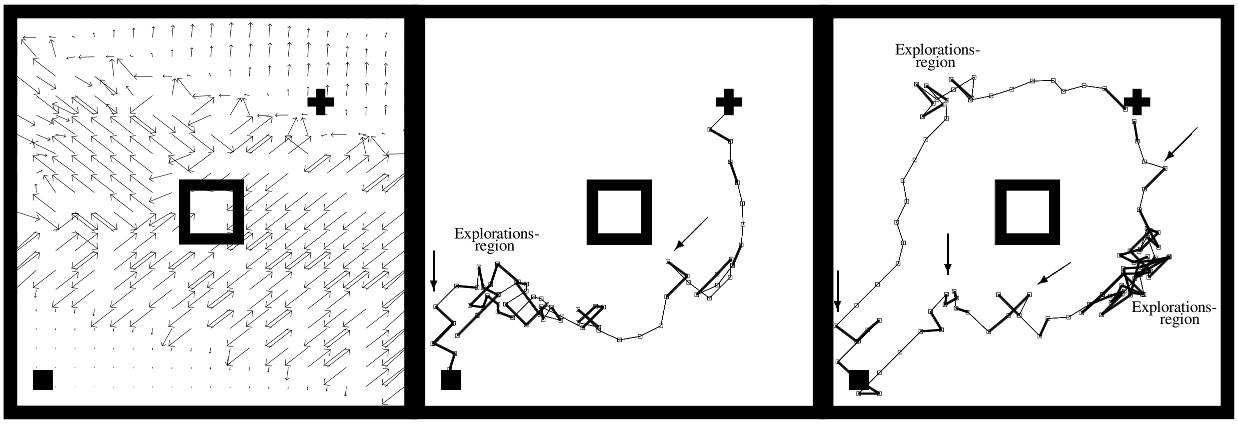


Abbildung 5.8: Ein „typisches" Konfidenznetz während des Lernvorgangs. Die Pfeile repräsentieren die optimale Explorationsrichtungen, d.h. diejenigen Aktionen, für die der erwartete Wissenszuwachs maximal ist. (b) und (c): Drei Beispielstrajektorien während des Lernens. Dicke Linien symbolisieren Explorationsaktionen und dünne Linien Exploitationsaktionen. Es ist klar zu sehen, wie das System zwischen beiden Modi wechselt (Pfeile).

	# Sequenzen	# Kollisionen	Anzahl gefundene Pfade	Modellfehler
Zufallsexploration	15,000	14,991	0	2.5 %
Exploitation	15,000	11,000	1	0.7 %
Gerichtete Exploration	15,000	4,000	2	0.4 %

Tabelle 5.1: Resultate des Vergleichs verschiedener Explorationsmethoden (gemittelt über 20 komplette Simulationen). Der quadratische Fehler ist in Relation zum Anfangsfehler ($= 100\%$) angegeben.

Diese Ergebnisse demonstrieren, daß die Explorationstechnik einen nicht unerheblichen Einfluß auf Lerngeschwindigkeit und -kosten haben kann. Der hier vorgestellte Mechanismus ist dabei nur einer unter vielen denkbaren Realisierungen. Er beruht auf der Annahme, daß die Maximierung des Wissenszuwachses eine geeignete Explorationsregel ist. Diese Annahme, auf der alle bekannten gerichteten Explorationsschemata beruhen, ist fraglich in großen Welten, in denen nur ein kleiner Teil exploriert werden muß, um eine optimale Kontrollfunktion zu finden. Wird während der Exploration auch Reward optimiert, kann man zumindest teilweise das Explorieren völlig inrelevanter Teile des Zustandsraums vermeiden. Es kann gezeigt werden, daß Kombinationen aus Exploration und Exploitation oft besser explorieren als reine Exploration.

Prinzipiell ist es unmöglich, Wissenszuwachs zu berechnen. Daher beruhen gerichtete Explorationsverfahren ausschließlich auf heuristischen Schätzverfahren. Typische Beispiele hierfür sind Systeme, die zählen, wie oft und/oder wann welcher Zustand beobachtet wurde. Das hier vorgestellte System versucht, den Voraussagefehler direkt zu schätzen. Dabei sind diese Schätzungen völlig ungenau – wie kann man bei einem Backpropagation-Modellnetz den Fehler in einer nie beobachteten Region des Zustandsraums kennen? Der Grund, warum Konfidenznetze trotzdem erfolgreich sind hängt mit der Tatsache zusammen, daß der genaue Fehler gar nicht von belang ist. Stattdessen reicht es, eine geeignete obere Schranke zu schätzen, solange gut explorierte Regionen einen geringeren Wert haben als schlecht explorierte. Der resultierende Kontroller konvergiert nicht, da Exploration nie abgeschlossen ist. Dieser Effekt ist nachteilhaft, ist man daran interessiert, daß nach einer Zeit des Lernens der Kontroller gegen eine optimale Lösung konvergiert. Je nach Problem kann die Divergenz des Kontrollers aber auch zum Vorteil gereichen. In dynamischen Welten, die sich mit der Zeit langsam ändern (zum Beispiel Roboterarme, deren Reibungskoeffizienten mit Erwärmung variieren), kann die nie endende Exploration dazu führen, daß Änderungen der Welt exploriert und modelliert werden.

5.4 Literatur

Zufallsexploration ist die wohl am häufigsten verwendete Explorationsart im Bereich Neurocontrol und Reinforcement Learning. Autoren wie Chuck Anderson [Anderson, 1986], Andy Barto [Barto *et al.*, 1989], Jonathan Bachrach/Mike Mozer [Bachrach und Mozer, 1991], [Mozer und

Bachrach, 1989], Mike Jordan [Jordan, 1989], Bartlett Mel [Mel, 1989], Paul Munro [Munro, 1987], Nguyen und Bernhard Widrow [Nguyen und Widrow, 1989], und Sebastian Thrun/Knut Möller/Alexander Linden [Thrun *et al.*, 1991b] berichten von Experimenten mit Zufallsexploration mit gleichverteilten Aktionswahrscheinlichkeiten. Im Bereich „Reinforcement Learning" hat sich dagegen durchgesetzt, bei Zufallsexplorationen die Wahrscheinlichkeiten für Aktionen dynamisch zu gewichten, um das erwartete Reward gleichzeitig zu maximieren und die Exploration auf relevante Zustände zu konzentrieren. Arbeiten, in denen solche Prinzipien definiert und angewandt werden, sind zum Beispiel Andy Barto/Rich Sutton/Chris Watkins [Barto *et al.*, 1989], David Chapman/Leslie Kaelbling [Chapman und Kaelbling, 1991], Lonnie Chrisman [Chrisman, 1992] Shridar Mahadevan/Jonathan Connell [Mahadevan und Connell, 1990], [Mahadevan und Connell, 1991], Long-Ji Lin [Lin, 1992], Satinder Singh [Singh, 1992b], Rich Sutton [Sutton, 1984], Steve Whitehead/Dana Ballard [Whitehead und Ballard, 1991] und Chris Watkins [Watkins, 1989].

Im Bereich gerichteter Explorationstechniken ist das Feld weniger dicht gesäht. Dies liegt vermutlich daran, daß nahezu alle im Bereich Neurocontrol und Reinforcement Learning bisher studierten Probleme hinreichend einfach und klein sind, so daß der durch die Explorationstechnik verursachte Komplexitätsunterschied unbedeutend ist. In Barto/Singh [Barto und Singh, 1990], Barto/Steve Bratke/Singh [Barto *et al.*, 1991] und Leslie Kaelbling [Kaelbling, 1990] finden sich Techniken, die auf „Zählen" beruhen: Ausgehend von diskreten Zustandsräumen zählt der Kontroller, welcher Zustand wie oft beobachtet wurde, und favorisiert dann selten beobachtete Zustände. Ein ähnliches Prinzip findet sich in Rich Suttons Arbeiten [Sutton, 1990], [Sutton, 1991]. Hier werden Zustände vorgezogen die für eine lange Zeit nicht aufgetaucht sind. Die dem Konfidenznetz wohl am verwandtesten Explorationsstrategien finden sich in Jürgen Schmidhubers Arbeiten [Schmidhuber, 1991a] und in Linden/Webers Papier [Linden und Weber, 1992]. Schmidhubers Explorationsalgorithmus [Schmidhuber, 1991a] läßt dabei auch gewisse nichtdeterministische Welten zu. Andy Moore hat ein ähnliches Explorationsschema basierend auf *memory-based learning* untersucht [Moore, 1990b]. In [Thrun, 1992a] und [Thrun, 1993] ist eine Erweiterung von direkter Exploration für einen mobilen Roboter beschrieben. Die im letzten Versuch vorgestellten Potentialfelder für Roboternavigation gehen auf Oussama Khatib zurück [Khatib, 1986a], [Khatib, 1986b].

Obwohl in jeder zweiten Neurocontrolpublikation der Begriff „Exploration" zu finden ist, gibt es keinen einheitlichen Gebrauch dieses Begriffes. Die hier verkürzt vorgestellte Taxonomie von Explorationstechniken geht auf Thrun [Thrun, 1992b], [Thrun, 1992c] zurück, und stellt einen Versuch der Standardisierung dieses Begriffes dar. Des weiteren sind die Komplexitätsergebnisse von Steve Whitehead (exponentielle Schranke) in [Whitehead, 1991a], [Whitehead, 1991b], und die von Sebastian Thrun (polynomiale Schranke) in [Thrun, 1992b] zu finden, wobei die hier vorgestellte polynomiale Schranke unter Abwandlung des Explorationsalgorithmus später von Sven König verbessert wurde [Koenig, 1992]. Sven König zeigt in dieser Arbeit auch, daß bessere Schranken nicht gefunden werden können. Die neuronale Explorationsarchitektur wurde erstmals in [Thrun und Möller, 1992] und [Thrun, 1992c] publiziert. Als Übersichtsliteratur zum Thema Exploration empfehlen wir [Thrun, 1992c].

Kapitel 6

Ausblick und zukünftige Forschung

Ziel des Fachbereiches „Künstliche Intelligenz“ (KI), in das diese Arbeit einzuordnen ist, ist das Bauen von Systemen, die in komplexen Situationen Entscheidungen fällen können. Um Entscheidungen zu fällen, müssen diese Systeme Wissen akquirieren, repräsentieren und verarbeiten. Ziel von „Maschinellem Lernen“, einem Untergebiet der KI, in das diese Arbeit ebenfalls einzuordnen ist, ist es, Wissen aus der Interaktion mit der Welt zu extrahieren, und dieses, zusammen mit anderem, vorgegebenen Wissen, in den Entscheidungsprozeß einfließen zu lassen.

Die in dieser Arbeit vorgestellten Techniken basieren auf dem *induktiven Lernen* von Wissen, d.h., dem Kompilieren von Wissen durch sukkzessives Experimentieren, Beobachten und Generalisieren. Da wir dabei zur Repräsentation gelernten Wissens Künstliche Neuronale Netze verwenden, wird dieses Wissen ausschließlich in den Gewichten und Schwellwerten akkumuliert. Die Interaktion mit der Welt spielt dabei eine zentrale Rolle. Dabei starten induktive Lernverfahren mit zwei Arten von Vorwissen: domänenunabhängiges Wissen und domänenabhängiges Wissen. Domänenunabhängiges Wissen sind zum Beispiel die Bausteine Künstlicher Neuronaler Netze, die Knoten und Kanten etc., sowie zum Beispiel die Suchheuristik Gradientenabstieg zur Suche von Gewichten und Schwellwerten. Dieses Wissen ist unabhängig von dem zu lösenden Problem. Domänenabhängiges Wissen dagegen ist zum Beispiel die spezielle Repräsentation des Problems, sprich zum Beispiel die Definition von Eingabe, Ausgabe und Reward. Weiteres domänenabhängiges Wissen sind die Anzahl der internen Schichten und Knoten eines Netzes, die Lernrate, der Planungshorizont, etc. Diese Parameter werden in der Regel unter Kenntnis des zu lösenden Problems gewählt.

Betrachtet man die in dieser Arbeit vorgestellten Ansätze aus dem Blickwinkel der KI, so kann man das initiale Wissen als *schwach* (*weak*) bezeichnen. Schwaches Wissen ist dadurch gekennzeichnet, daß es zur Lösung eines Problems alleine nicht ausreichend ist, und daher ohne Experimentation und Interaktion mit der Welt Probleme nicht gelöst werden können.

Was ist der Vorteil von schwachem Wissen? Systeme mit schwachem Wissen bieten den entscheidenden Vorteil, daß sie eine ganze Klasse von Problemen und nicht nur ein einzelnes lösen können. Im schwachen Initialwissen ist das spezielle zu lösende Problem nicht vollständig spezifiziert. So wissen zum Beispiel die in Kapitel 1 beschriebenen Mehrschichtnetze nicht, ob sie UND, XOR, oder vielleicht eine völlig andere Funktion realisieren sollen. Dem in Kapitel 4 beschriebenen Planer ist anfangs nicht bekannt, daß seine spezielle Welt aus einem Roboterarm und einem rollenden Ball besteht, und nur sukkzessives Ausprobieren vermittelt ihm dieses Wissen. Vielmehr ist die Klasse der lösbaren Probleme indirekt durch die *Architektur* des Systems (Backpropagationnetze lösen andere Probleme als Adaptive Kontroller) sowie durch *Vorgaben* und *Parameter* wie Größe der Netze, Aktivationsfunktionen, Planungshorizont (Lookahead) etc. bestimmt. Schwache Systeme stellen daher eine vielversprechende Perspektive für den Bau von Problemlösern für Probleme dar, deren Details zur Designzeit nicht oder nicht vollständig bekannt sind. Schwaches Wissen, zusammen mit induktivem Lernen, ermöglicht die Anpassung des Problemlösers an das Problem. In komplexen, nicht modellierbaren Welten erscheint daher die Verwendung von induktiven Lernalgorithmen zum Lösen von Problemen unbedingt notwendig.

Was ist der Nachteil von schwachem Wissen? Grundsätzlich stellt sich die Frage, wie *effizient* Systeme mit schwachem Wissen komplexe Probleme lösen können. In solchen Systemen muß nämlich ein Teil des Problemwissen interaktiv in Form von Trainingsbeispielen aus der Welt extrahiert werden. Ist das Wissen zu schwach, kann dies für komplexe Probleme sehr aufwendig und somit ineffizient sein. Die in dieser Arbeit vorgestellten Probleme, zweistellige boolsche Funktionen, Target Tracking, Roboterarmkontrolle und zweidimensionale Roboternavigation, sind alle mehr oder minder einfache Probleme. Wie verhalten sich Backpropagation und die vorgestellten Ansätze zur Kontrolle und Exploration in komplexeren Situationen? Was passiert zum Beispiel,. . .

- . . . wenn Eingabe- und Ausgaberäume 1 000 oder mehr Dimensionen haben? Nach Kenntnis des Autors lassen die meisten empirischen Untersuchungen von Backpropagation im Bereich Neurocontrol zwischen 2 und schätzungsweise 200 Eingabevariablen zu. Bis heute ist nicht klar, wie sich Künstliche Neuronale Netze sowie der Backpropagationalgorithmus in größeren Problemen bewähren, und wie die Anzahl der zur erfolgreichen Generalisierung erforderlichen Trainingsbeispiele mit der Dimensionalität und der Komplexität des Problems wächst. Viele komplexe in der KI studierten Probleme weisen solch große Zustandsräume auf.

  Die Anwendung von Backpropagation auf solche Probleme, und als direkte Folge die Anwendung der hier beschriebenen Kontrollverfahren, stellen offene Fragen dar, die einer Antwort bedürfen. Generell ist es unwahrscheinlich, daß Lookahead Planen in der hier vorgestellten einfachen Version auf deutlich komplexere Probleme angewendet werden kann. Große Aktionsräume und besonders große Planungshorizonte vergrößern den Suchraum beim Planen, und mit größeren Suchräumen steigt die Wahrscheinlichkeit, daß Gradientenabstieg ungeeignete Lösungen generiert. Um Planen auf komplexere Situationen anzuwenden, macht es Sinn, alternativ andere Suchheuristiken zu verwenden. Suchräume können darüber hinaus durch das Einführen von *Teilzielen* (*subgoals*) und somit Planhierarchien verkleinert werden. Dieses hat sich in der symbolischen KI als hilfreich erwiesen. Exploration mit Konfidenznetzen, wie in Kapitel 5 beschrieben, beruht mehr oder minder auf der erschöpfenden Exploration des gesamten Zustandsraums. Exploration wird dabei nicht auf relevante Teile des Zustandsraums fokussiert. In großen Zustandsräumen ist erschöpfende Exploration in der Regel ineffizient oder schlicht unmöglich. Dies ist zum Beispiel wahr für viele im Bereich symbolischer KI studierte Zustandsräume. Exploration muß dann mit Exploitation so kombiniert werden, daß nur relevante und für das Problem vielversprechende Teile des Zustandsraums exploriert werden.

- . . . wenn die Welt stochastisch ist? Stochastische Welten haben nicht-eindeutige Klassifikationen, nicht-eindeutige Zustandsübergangsfunktionen, verrauschte Beobachtungen etc.

  Backpropagation ist zwar in der Lage, bestimmte Formen von stochastischen Rauschen (Meßfehler) zu tolerieren. Zweifelsohne aber sind deterministische Funktionsapproximatoren zum Modellieren von allgemeinen stochastischen und damit nichtdeterministischen Funktionen wenig geeignet. Im Falle von Künstlichen Neuronalen Netzen liegt dies daran, daß Backpropagation den quadratischen Voraussagefehler minimiert und das resultierende Netz vereinfacht gesagt den Mittelwert aller Beobachtungen vorhersagt. Im Falle des in Kapitel 5 beschriebenen, gerichteten Explorationsalgorithmus kann Nichtdeterminismus besonders kritisch sein, da der Explorationsalgorithmus Regionen mit großem Voraussagefehler zu explorieren sucht. In deterministischen Welten führt sukkzessives Explorieren solcher Regionen zum Sinken des Voraussagefehlers, nicht jedoch so in stochastischen Welten.

  Grundsätzlich ist das Modellieren von stochastischen Welten im Bereich Künstlicher Neuronaler Netze wenig erforscht. Kann man parametrische Annahmen über die Verteilungsfunktionen der stochastischen Zufallsvariablen machen, besteht die Möglichkeit, diese Parameter mit Künstlichen Neuronalen Netzen lernen zu lassen. So kann zum Beispiel im Falle Gausscher Verteilungen ein Netz den Mittelwert, und ein zweites Netz die Streuung lernen. Solche Techniken sind im Bereich Neurocontrol bisher wenig oder gar nicht systematisch evaluiert worden.

- . . . wenn die Welt nichtsichtbare Zustandsinformation besitzt? Alle hier geschilderten Techniken beruhen mehr oder minder auf der Annahme, daß der Zustand der Welt beobachtbar ist. Im einfachsten Fall, den sogenannten Typ-0 Problemen, kann der Kontroller jederzeit den Zustand der Welt komplett erfassen. Im allgemeinen wird man jedoch davon ausgehen, daß ein „Schnappschuß“ der Welt nicht alle zur Entscheidungsfindung erforderlichen Informationen liefert. Grundsätzlich gibt es daher vier weitere Typen von Kontrollproblemen, die in diese Kategorie fallen:

  - Typ-1 und Typ-2 Probleme sind Probleme, bei denen die relevante Information irgendwann in der Vergangenheit beobachtet wurde. In Typ-1 Problemen geschah dies irgendwann in den letzten n Zeitschritten für ein bekanntes n, während dies in Typ-2 Problemen beliebig weit in der Vergangenheit liegen kann.
  - Typ-3 Probleme sind Typ-2 Probleme, bei denen man die relevante Information nur beobachten kann, wenn man die richtigen Aktionen angewendet. Es besteht also die Möglichkeit, daß die zur Entscheidung relevante Information nie beobachtet worden ist.

- Typ-4 Probleme sind die allgemeinsten Probleme. Hier ist es möglich, daß die erforderliche Information prinzipiell nicht beobachtbar ist, unabhängig von der Wahl der Aktionen.

Ein Beispiel für Typ-1 Welten ist in dem Roboterarmexperiment gegeben: Vom momentanen Ort des Balles kann man dessen nächste Position nicht voraussagen, kennt man nicht zusätzlich die letzte Position des Balles. Time Delay Neural Networks, die als Input ein Zeitfenster der letzten n Beobachtungen verwenden, bieten eine attraktive Möglichkeit, die in dieser Arbeit vorgestellten Techniken auf Typ-1 Probleme zu erweitern. Rekurrente Netze (Elman-Netze) bieten sich für Typ-2 Probleme an. Typ-3 Probleme sind dagegen deutlich schwerer zu lösen. Keine der hier geschilderten Kontrolltechniken erlaubt das Erlernen einer Kontrollstrategie, die systematisch relevante Information durch geeignete Aktionen zutage fördert. Die hier vorgestellten Explorationstechniken sind ebenfalls nicht ohne weiteres für solche Probleme anwendbar. Dies liegt im wesentlichen daran, das gerichtete Explorationssysteme wissen, wo sie nichts wissen. In Typ-3 Problemen weiß man allerdings nicht, wo man nichts weiß. Dieser subtile Unterschied ist entscheidend für die Exploration unbekannter Welten. Type-4 Probleme lassen sich als Mixtur von Typ-3 Problemen und den weiter oben diskutierten stochastischen Problemen auffassen.

- . . . wenn die Anzahl der Trainingsbeispiele begrenzt ist? In vielen Applikationen von Lernsystemen in der *real world* ist Trainingsinformation der eigentlich teure und schwer zu erhaltende Faktor. Setzt man zum Beispiel Künstliche Neuronale Netze zur Klassifikation von Krebspatienten ein (ein vielstudiertes Problem), ist die Anzahl der Patienten begrenzt und kann nur schwer erweitert werden. Baut man zum Beispiel ein Kontrollsystem zum Landen von Flugzeugen, ist es offensichtlich, daß Trainingsinformation teuer ist und nur begrenzt zur Verfügung steht. Wie viele Flugzeuge kann man schließlich opfern, bevor eine Kontrollstrategie erfolgreich gelernt ist?

  Probleme mit beschränkter Trainingsinformation spielen eine wichtige, wenn nicht sogar die wichtigste Rolle im Bereich Maschinellen Lernens. In vielen komplexen Anwendungen von Lernalgorithmen jeglicher Art ist es unmöglich, eine dem Performanzziel adäquate Menge von Trainingsbeispielen zu extrahieren. Dies steht im Gegensatz zu allen in dieser Arbeit geschilderten Simulationen und Experimenten, bei denen hinreichend viele Trainingsmuster synthetisch generiert wurden. Um die hier vorgestellten Verfahren in komplexen Problemen in der Praxis einsetzen zu können, muß Trainingsinformation daher graduell durch Vorwissen ersetzt werden. Dieses Vorwissen kann mannigfaltiger Natur sein. In vielen Systemen im Bereich Maschinellen Lernens wird dieses Wissen durch Fakten, Regeln, oder anderen regelähnlichen Suchraumbeschränkungen zum Beispiel in Form von prädikatenlogischen Ausdrücken repräsentiert. Solches Wissen ist in Künstlichen Neuronalen Netzen allerdings nur begrenzt brauchbar, da man bestenfalls die Größe des Netzes und eine Handvoll Parameter optimieren kann. Folglich ist eine der zentralen offenen Fragen bezüglich der hier vorgestellten Lernverfahren, wie man generell Domänenwissen in benutzerfreundlicher Form in Netze oder Kontrollsysteme injizieren kann, um die Generalisierung zu verbessern und die Anzahl der erforderlichen Trainingsbeispiele zu verringern. Kann man hybride Systeme bauen, die solches Wissen verwenden können und gleichzeitig die Vorteile von Künstlichen Neuronalen Netzen nutzen?

Wie man dieser nicht vollständigen Liste entnimmt, gibt es viele zu beantwortende Fragen, bevor die dieser Arbeit zugrundeliegenden Techniken erfolgreich auf komplexe Probleme angewendet werden können. Leider fehlen darüber hinaus für Künstliche Neuronale Netze, Backpropagation und Neurocontrol noch vielfach theoretische Resultate. Forschung im Bereich Lerntheorie, insbesondere im Bereich *PAC-Learning* (*PAC=probably approximately correct*, zu deutsch *wahrscheinlich annähernd korrekt*) haben bereits wichtige Resultate für das induktive Lernen von Funktionen erzielt. Dabei analysiert PAC-Learning die Größe der Trainingsmenge, die man für erfolgreiche Generalisierung benötigt. Theoretische Resultate im Bereich Künstlicher Neuronaler Netze beziehen sich nahezu ausschließlich auf die in Kapitel 1 eingeführten binären Schwellwertknoten, nicht aber auf kontinuierliche Aktivationsfunktionen. Darüber hinaus ist die Verteilung der Trainingsbeispiele eines Neurocontollers in der Regel unterschiedlich von der Verteilung, mit der dieser nach dem Training verwendet wird. In der PAC-Lerntheorie nimmt man dagegen an, daß diese Verteilungen identisch sind. Aus diesen und anderen Gründen lassen sich diese Ergebnisse nicht direkt auf die hier vorgestellten Probleme und Techniken übertragen. Es ist nicht bekannt, unter welchen Bedingungen und mit welcher Geschwindigkeit Backpropagation konvergiert. Des weiteren sind keine generellen Rezepte bekannt, wie man unter Kenntnis eines Problems das geeignete, optimal generalisierende Netz wählt. Es ist auch nicht bekannt, unter welchen Bedingungen adaptive Kontroller konvergieren. Theoretische Resultate für die Geschwindigkeit von Planen mit Gradientenabstieg fehlen. Des weiteren ist nicht bekannt, wie sich die Ungenauigkeit der Modellnetze auf die Qualität der Aktionen und das erwartete Reward auswirkt. Im Bereich Exploration gibt es zwar theoretische Resultate für Table-lookup-Repräsentationen, und diese Arbeit leistet dazu einen Beitrag, doch lassen sich diese Resultate nur begrenzt auf Künstliche Neuronale Netze übertragen. Ein besseres theoretisches Verständnis der verwendeten Methoden erscheint, nicht nur im Hinblick auf diese Arbeit, sondern auch im Hinblick auf weite Teile der Forschung im Bereich Künstlicher Neuronaler Netze, Backpropagation und Neurocontrol wünschenswert, um nicht zu sagen erforderlich.

Anhang

Die Zeitkomplexität von gerichteter Exploration

Die Zeitkomplexität von Exploration läßt sich am besten in deterministischen Welten mit endlichen Zustandsmengen untersuchen. Solche Welten sind Spezialfälle von *Makov-Welten*, die allgemeine probabilistische Zustandsübergänge zulassen. In Makov-Welten läßt sich der Zustand der Welt beobachten. Formal kann man die Welt als Funktion von Zuständen s und Aktionen a zu Folgezuständen s' beschreiben. Dabei sind die Zustandsräume endlich (d.h., es gibt endlich viele Zustände und Aktionen) und ergodisch (d.h., jeder Zustand ist mit einer geeigneten Aktionsfolge von jedem anderen Zustand aus erreichbar.)

Sei n die Anzahl der Zustände und l die Tiefe des Zustandsraums, definiert als die längste kürzeste Distanz zwischen zwei Zuständen (damit meinen wir eine obere Schranke für die minimale Anzahl von Aktionen, derer es bedarf, zwei beliebige Zustände s_1 und s_2 ineinander zu überführen). Im ungünstigsten Fall ist $l = n - 1$, aber in vielen Welten ist l deutlich kleiner als n (in d-dimensionale Gitterwelten ist l in $O(d\sqrt[d]{n})$). Ein Aktionsmodell der Welt (im folgenden gelegentlich kurz mit Modell bezeichnet) ist eine Funktion, die Zustände s und Aktionen a in Folgezuständen s' überführt. Ein vollständiges Modell ist perfekt, wenn die vom Modell vorhergesagten Folgezuständen mit denen der Welt für beliebige Zustände und Aktionen übereinstimmen.

Wir werden im folgenden den Komplexitätsbeweis in zwei Teile unterteilen. Im ersten Teil beweisen wir, daß eine gerichtete Explorationsregel der Art *„wähle die Aktion, die zu dem am wenigsten explorierten Folgezustand führt“* in $O(ln^2)$ Schritten das Auftreten aller Zustände sichert. Diese Regel bedingt allerdings ein korrektes Modell. Im zweiten Teil werden wir beweisen, daß ein solches Modell während der Exploration gelernt werden kann, und die resultierende Komplexität in $O(dln^2)$ liegt. Dabei ist in beiden Fällen die Explorationsregel lokal, d.h., die Zeit zur Aktionssuche ist in $O(d)$ und hängt weder von der Größe des Zustandsraums n, noch seiner Tiefe l ab.

Exploration mit einem perfektem Aktionsmodell

Theorem 1 (Komplexität von Exploration unter Verwendung eines perfekten Aktionsmodells):

Sei die Welt deterministisch, endlich und ergodisch. Dann gibt es eine lokale, gerichtete Explorationsregel, mit der jeder Zustand unter Verwendung eines perfekten Aktionsmodells nach maximal $O(ln^2)$ Aktionen mindestens einmal aufgetreten ist.

Bevor wir dieses Theorem beweisen, werden wir die Explorationsregel definieren und zwei Lemmata beweisen. Die Explorationsregel, die diesem Theorem zugrunde liegt, ist eine gerichtete Explorationsregel basierend auf *Zählern*. Zu jedem Zustand s sei ein Zähler $c(s)$ assoziiert, der zählt, wie oft Zustand s beobachtet wurde (bzw. der eine obere Schranke dieser Zahl darstellt). Wir benutzen die Schreibweise $c^t(s)$ zur Bezeichnung des Wertes von $c(s)$ zum Zeitpunkt t ($t = 0, 1, \ldots$), d.h., nach der Ausführung von t Aktionen. Der Zustand der Welt nach t Aktionen sei s^t, und $Succ(s)$ bezeichne die Menge der von einem beliebigen Zustand s mit einer Einzelaktion erreichbaren Zustände s', die Menge der Folgezustände. Die Explorationsregel ist die folgende:

Explorationsregel:
Zum Explorationsbeginn wird die Zeit t mit 0 initialisiert, und alle Zähler $c(s)$ auf 0 gesetzt: $(\forall s)$ $c^0(s) \longleftarrow 0$. In jedem Explorationsschritt $t = 0, 1, \ldots$, wähle eine Aktion nach folgender Regel:

1. Sei s_{min} derjenige Folgezustand in $Succ(s^t)$, der $c(s)$ unter allen Folgezuständen minimiert. Existieren mehrere Minima, wähle einen Folgezustand zufällig. Man beachte, daß die Berechnung von s_{min} und der zugehörigen Aktion in $O(d)$ Zeit geschieht und somit lokal ist.

2. Wenn $c(s^t) \leq c(s_{min})$, der Zähler des derzeitigen Zustandes also kleiner oder gleich dem Zähler des Folgezustandes s_{min} ist, setze $c(s^t) \longleftarrow c(s_{min}) + 1$.

3. (Nun gilt $c(s^t) > c(s_{min})$.) Wähle und führe diejenige Aktion aus, die zum Zustand s_{min} führt. Nach Ausführen der Aktion wird der Zähler des Folgezustandes $c(s_{min})$ um 1 inkrementiert. Die Zeit t wird ebenfalls um 1 erhöht.

4. Gehe zu Schritt 1.

Man beachte, daß t die Anzahl der Aktionen zählt, und daß $\sum_s c^t(s)$ eine obere Schranke für t ist. Darüber hinaus ist $c^t(s)$ eine obere Schranke für die Anzahl der Vorkommnisse von Zustand s bis zum Zeitpunkt t. Für alle Zustände s ist $c^t(s)$ monoton steigend in t.

Das folgende Lemma zeigt, daß während einer Aktionsfolge, die von einem beliebigen Zustand mit Zähler m zu einem beliebigen anderen Zustand mit Zähler $\geq m + 1$ führt, Schritt 2 der

Explorationsregel mindestens einmal ausgeführt wurde, und daß zu diesem Zeitpunkt der Zähler des aktuellen Zustandes dabei von einem Wert $\leq m$ zu einem Wert $\geq m+1$ geändert wurde.

Lemma 1: Sei für ein beliebiges $m \geq 0, t' > t \geq 0$: $c^t(s^t) = m$ und $c^{t'}(s^{t'}) \geq m+1$. Dann gibt es ein $\hat{t} \in \{t, t+1, \ldots, t'\}$ mit $c^t(s^{\hat{t}}) \leq m$ und $c^{t'}(s^{\hat{t}}) \geq m+1$.

Beweis: Grundsätzlich gibt es zwei Schritte in der Explorationsregel, die einen Zähler inkrementieren können: Schritt 2 und Schritt 3. Zuerst betrachte Schritt 3: Die Ausführung von Schritt 3 ändert den aktuellen Zustand der Welt, s_1 genannt, zu einem Folgezustand, s_2 genannt. Dabei gilt $c(s_2) < c(s_1)$ vor der Ausführung der Aktion, und der Zähler $c(s_2)$ wird anschließend um 1 hochgesetzt. Wenn nun der Zähler $c(s_1)$ kleiner oder gleich m ist, folgt, daß der Zähler des neuen Zustandes $c(s_2)$ ebenfalls kleiner oder gleich m ist. Startet man mit einem aktuellen Zustand mit Zähler $\leq m$, kann Schritt 3 nicht ändern, daß der Zähler des folgenden aktuelle Zustand ebenfalls $\leq m$ ist. Da Zähler aber generell größer als m werden können, wird dies durch Schritt 2 erreicht.

Der Rest des Beweises ist einfach. Wegen $c^t(s^t) = m$ gibt es einen Zustand $s^{\hat{t}}$ in der Folge von Zuständen, bei dem Schritt 2 den Zähler $c^{\hat{t}}(s^{\hat{t}})$ von einem Wert $\leq m$ auf einen Wert $\geq m+1$ inkrementierte. Das Lemma folgt dann unter Beachtung der Tatsache, daß $c(\cdot)$ monoton in t steigt. □

Das nächste Lemma zeigt, daß es nicht beliebig große „Sprünge“ in den Zählern zweier benachbarter Zustände gibt.

Lemma 2: Für alle t und Zustände s und $\hat{s}$ mit $s \in Succ(\hat{s})$ gilt: $c^t(\hat{s}) - c^t(s) \leq n$.

Beweis: Seien s und $\hat{s}$ zwei beliebige Zustände mit $s \in Succ(\hat{s})$. Sei $m \equiv c^t(s)$ und $\hat{m} \equiv c^t(\hat{s})$.

Indirekter Beweis: Man nehme an, daß $\hat{m} - m > n$. Zuerst wird gezeigt, daß es einen Zeitpunkt $\hat{t}$ gibt, ab dem der Zähler von $\hat{s}$ nur noch durch Schritt 3, nicht aber Schritt 2 obiger Explorationsregel inkrementiert wurde. Daraus folgt, daß das Inkrement von $c(\hat{s})$ im Zeitraum $\{\hat{t}+1, \ldots, t\}$ angibt, wie oft sich die Welt im Zustand $\hat{s}$ während dieses Zeitintervalls befunden hat. Dies wird folgendermaßen gezeigt: Wegen der Annahme gilt $\hat{m} - m > n$. Da $\hat{m} > m$, folgt, daß es einen Zeitpunkt $\hat{t} < t$ gibt, ab dem $c(\hat{s})$ größer als m wurde. Dies geschah entweder mit Schritt 2, oder Schritt 3 der Explorationsregel, doch in beiden Fällen war der genaue Wert $c^{\hat{t}}(\hat{s}) = m+1$, da es einen Folgezustand s mit Zähler $c^{\hat{t}}(s) \leq m$ gab, und somit $c^{\hat{t}}(s_{min}) \leq m$, und da folglich Schritt 2 den Wert von $c^{\hat{t}}(\hat{s})$ nicht auf einen Wert $> m+1$ inkrementieren konnte. Diese Beobachtung läßt sich auf die nächsten $t - \hat{t}$ Aktionen und somit auf das gesamte Zeitintervall $\{\hat{t}+1, \ldots, t\}$ verallgemeinern: Für alle $\tau \in \{\hat{t}+1, \ldots, t\}$ gilt:

$c^\tau(\hat{s}) > m$ und $c^\tau(s) \leq m$. Da $s \in Succ(\hat{s})$, folgt für alle Zeitpunkte τ, bei denen der aktuelle Zustand $\hat{s}$ war: $c^\tau(s_{min}) \leq c^\tau(s) \leq m$. Daraus folgt, daß auch hier Schritt 2 nicht im Zustand $\hat{s}$ ausgeführt wurde, und der Zähler $c(\hat{s})$ nur durch durch Schritt 3 hochgesetzt wurde. Folglich trat $\hat{s}$ als aktueller Zustand der Welt genau $(\hat{m} - m - 1)$ mal im Zeitraum $\{\hat{t} + 1, \ldots, t\}$ auf.

Da per Annahme $\hat{m} - m - 1 \geq n$ gilt, war $\hat{s}$ mindestens n mal aktueller Zustand in $\{\hat{t} + 1, \ldots, t\}$. Der von Schritt 1 bestimmte Folgezustand ist immer der Zustand mit dem kleinsten Zählerwert. Daraus kann man folgern, daß der Folgezustand von $\hat{s}$ innerhalb dieses Zeitintervalls immer einen Zähler $\leq m$ hatte. Es gilt allerdings auch, daß nach Ausführen einer Aktion im Zustand $\hat{s}$ dieser Zustand mindestens $n - 1$ weitere Male beobachtet wurde, der ja selbst einen Zähler $\geq m + 1$ hat. Dann folgt nach Lemma 1, daß für jede erneute Beobachtung von $\hat{s}$ ein anderer Zustand seinen Zähler von einem Wert $\leq m$ zu einem Wert $\geq m + 1$ änderte. Da c monoton steigend ist, kann jeder Zähler nur einmal seinen Wert so ändern. Es folgt, daß nach obigen $n - 1$ erneuten Beobachtungen von $\hat{s}$ alle $n - 1$ Zähler von allen anderen Zuständen, d.h., allen Zuständen $\neq \hat{s}$, einen Wert $\geq m + 1$ haben. Dies gilt wegen der Monotonie der Zähler auch zum Zeitpunkt t, und widerspricht somit der Annahme $c^t(s) = m$. □

Der Beweis des Theorems ist nun einfach:

<u>Beweis des Theorems:</u> Wenn der letzte Zustand s_{last} (und damit jeder andere Zustand) erstmalig aufgetreten ist, ist sein Zähler $c(s_{last}) = 1$. Nach Lemma 2 und Induktion über alle Aktionsfolgen, die zu diesem Zustand führen, folgt, daß kein anderer Zustand s einen Zähler $c(s) > 1 + (n - 1) * l = ln - l + 1$ besitzt. Daraus folgt, daß jeder Zustand maximal ln mal beobachtet wurde, und $\sum_s c(s) < ln^2$. Dies ist aber aber, wie unmittelbar nach Theorem 1 bemerkt, eine obere Schranke für die Anzahl der Aktionen t. Damit ist das Theorem bewiesen. □

Anmerkung: Aus Theorem 1 folgt, daß Zustände, die nach ln^2 nicht gefunden worden sind, vom aktuellen Zustand nicht erreicht werden können. In diesem Fall ist die Welt nicht, wie angenommen, ergodisch.

Exploration ohne Aktionsmodell

Theorem 1 basiert auf der Annahme, daß ein perfektes Aktionsmodell zum Zeitpunkt der Exploration verfügbar ist. Diese Annahme ist unrealistisch im Zusammenhang mit Adaptiver Kontrolle, da hier die Welt anfangs unbekannt ist. Wir werden nun Theorem 1 erweitern auf Szenarios, in denen das Aktionsmodell anfangs unbekannt ist und während der Exploration gelernt wird. Dafür muß obige Explorationsregel um einen Schritt erweitert werden, der sicherstellt, daß das Modell systematisch gelernt wird. Wenn während der Exploration für einen Zustand s eine Aktion a ausgeführt wird, dies in den Zustand s' führt, und diese Aktion bisher noch nicht in diesem Zustand ausgeführt wurde, wird das Tripel (s, a, s') als Teil des Aktionsmodells gespeichert. Da die Welt deterministisch und endlich ist, führt iteratives Anwenden zur erschöpfenden *Identifikation* der Welt, d.h., zum inkrementellen *Erlernen* eines Aktionsmodells. Basierend auf diesem einfachen „Lernmechanismus“ werden wir nun eine Komplexitätsschranke für Exploration in Welten herleiten, für die kein Aktionsmodell vorgegeben ist.

Theorem 2 (Komplexität von Exploration):

Sei die Welt deterministisch, endlich und ergodisch. Dann gibt es eine lokale, gerichtete Explorationsregel, mit der jeder Zustand nach maximal $O(dln^2)$ Aktionen mindestens einmal aufgetreten ist.

Um diese generellere Komplexitätsschranke herzuleiten, muß die bisherige Explorationsregel modifiziert werden. Die im folgenden beschriebene Änderung besagt, daß die Identifikation der Welt höhere Priorität hat als die Suche nach Zuständen.

0. Wenn es eine Aktion a gibt, die im aktuellen Zustand noch nie ausgeführt wurden, wähle a (unabhängig von der Zählertabelle c). Führe a aus und inkrementiere den Zähler des Folgezustandes. Inkrementiere t und speichere den beobachteten Zustandsübergang in dem Aktionsmodell. Gehe zu Schritt 0.

   ⋮

4. Gehe zu Schritt 0.

Man beachte, daß die Ausführung von Schritt 1, 2 oder 3 bedingt, daß die Vorbedingung von Schritt 0 im aktuellen Zustand nicht erfüllt ist, und somit alle Aktionen im aktuellen Zustand modelliert sind. In diesem Fall ist das Modell *vollständig* im aktuellen Zustand, und die im Rahmen von Theorem 1 beschriebene Explorationstechnik ist in der Tat anwendbar. Man beachte überdies, daß alle der obigen Explorationsregel folgenden Bemerkungen nach wie vor gültig sind.

> Beweis von Theorem 2: Mit der erweiterten Explorationsregel ist Lemma 1 ungültig, da Schritt 0 eine Zustandstransition in einen Zustand mit größerem Zähler als der Zähler des vorangehenden Zustandes verursachen kann. Da aber die Anzahl der anfangs unbekannten Zustands-Aktionstupel durch dn beschränkt ist, kann dies nur dn während des gesamten Explorationsvorganges geschehen. Daher kann man Lemma 1 folgendermaßen erweitern:

<u>*Lemma 3:*</u> *Sei für beliebige $m \geq 0, t' > t \geq 0$: $c^t(s^t) = m$ und $c^{t'}(s^{t'}) \geq m+1$. Dann gibt es einen Zeitpunkt $\hat{t} \in \{t, t+1, \ldots, t'\}$ in dem Schritt 0 angewendet wurde, oder Lemma 1 gilt.*

Lemma 3 folgt direkt aus Lemma 1 und den obigen Überlegungen. Aus Lemma 3 folgt nun, analog zum Beweis von Lemma 2, daß die Anzahl der Vorkommnisse eines beliebigen Zustandes $\hat{s}$ mit $c^t(\hat{s}) = \hat{m}$ im Zeitintervall $\{\hat{t}+1, \ldots, t\}$ durch $n + nd$ nach oben beschränkt ist. Die zusätzliche Komplexität nd stammt von Schritt 0 in der erweiterten Explorationsregel, die, wie bereits vermerkt, die Welt in Zustände mit größerem Zähler versetzen kann. Da per definitionem gilt $c^\tau(\hat{s}) > m = c^t(s)$ für alle $\tau \in \{\hat{t}+1, \ldots, t\}$, folgt, daß $\hat{s}$ genau $m' - m + 1$ mal in diesem Zeitintervall vorgekommen ist. Für jede dieser $m' - m + 1$ Beobachtungen des Zustandes $\hat{s}$ gibt es zwei mögliche Ursachen: a) Lemma 2 gilt und irgendein anderer Zustand inkrementierte seinen Zähler von einem Wert $\leq m$ zu einem Wert $\geq m+1$, oder b) es wurde mindestens eine „neue" Aktion mit Schritt 0 gelernt. Diese Aktion führte von einem Zustand mir Zähler $\leq m$ zu einem Zustand mit Zähler $\geq m+1$. Da es maximal $n-2$ Zähler auf einen Wert $\geq m+1$ inkrementiert werden können (alle Zustände bis auf s und $\hat{s}$), und da die Anzahl der „neuen" Aktionen und somit Anwendungen von Schritt 0 durch dn beschränkt ist, folgt die obere Schranke $\hat{m} - m - 1 \leq n - 2 + dn < n + dn$. Dies führt zu der folgenden Erweiterung von Lemma 2:

<u>*Lemma 4:*</u> *Für alle t und Zustände s und $\hat{s}$ mit $s \in Succ(\hat{s})$ gilt: $c^t(\hat{s}) - c^t(s) \leq n(d+1)$.*

Der Rest des Beweises von Theorem 2 ist äquivalent zum Beweis von Theorem 1 unter Einbeziehung des Faktors d in der Komplexität. Daraus folgt, daß die Anzahl der Aktionen in $O(dln^2)$ liegt, wenn die erweiterte Explorationsregel angewendet wird und gleichzeitig ein Aktionsmodell akquiriert wird. □

Nach d Vorkommnissen eines jeden Zustandes s sind alle legalen Aktionen mindestens einmal ausprobiert worden, und das Aktionsmodell ist somit vollständig. Da die Welt deterministisch ist, ist es auch korrekt und somit perfekt. Daraus folgt direkt eine obere Schranke für die Komplexität für <u>Systemidentifikation</u>, d.h., des Erlernens eines Modells:

Korollar (Komplexität von Systemidentifikation):

Sei die Welt deterministisch, endlich und ergodisch. Dann gibt es eine lokale, gerichtete Explorationsregel, mit der nach maximal $O(d^2ln^2)$ Aktionen ein perfektes Modell der Welt gelernt werden kann.

Es sei darauf hingewiesen, daß Theoreme 1 und 2 in der Originalpublikation auf eine Komplexitätsschranke für Reinforcement Learning und Q-Learning in deterministischen Welten erweitert wird.

Diese Erweiterung kombiniert die Komplexität von gerichteter Exploration mit der Komplexität von Dynamischer Programmierung. Da die Komplexität von Dynamischer Programmierung ebenfalls polynomial in der Anzahl der Zustände ist (sie ist in $O(ln)$), folgt daraus worst-case polynomiale Lernzeit für Reinforcement Learning und Q-Learning. Dies ist allerdings nur gültig in deterministischen, endlichen und ergodischen Welten, wenn man Table-lookup-Methoden zur Repräsentation von Zählern und Modell verwendet. Einen Komplexitätsbeweis für Reinforcement Learning mit Künstlichen Neuronalen Netzen gibt es wegen der fehlenden Konvergenztheorie für Künstliche Neuronale Netze bisher nicht.

Für allgemeine stochastische Welten hingegen kann man zeigen, daß beliebige Explorationsregeln, ob gerichtet oder nicht, exponentielle Explorationszeit erfordern können. Das Spektrum stochastischer Welten, für die gerichtet Explorationstechniken beweisbar effizienter sind als ungerichtete, ist nach wie vor unerforscht.

Literaturverzeichnis

[Almeida, 1988] L. B. Almeida. Backpropagation in perceptrons with feedback. In R. Eckmiller and C. von der Malsburg, editors, *Neural Computers*, Berlin, 1988. Springer-Verlag.

[An *et al.*, 1988] C.H. An, Ch.G. Atkeson, and J.M. Hollerbach. *Model-Based Control of a Robot Manipulator*. The MIT Press, Cambridge, MA, 1988.

[Anderson, 1986] Charles W. Anderson. Learning and problem solving with multilayer connectionist systems. Technical Report COINS TR 86-50, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, 1986.

[Anderson, 1988] James A. Anderson. *Neurocomputing: foundations of research*. MIT Press, Cambridge, Massachusetts, 1988.

[Anderson, 1989] C.W. Anderson. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37, 1989.

[Atkeson, 1991] Christopher A. Atkeson. Using locally weighted regression for robot learning. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pages 958–962, Sacramento, CA, April 1991.

[Bachrach und Mozer, 1991] Jonathan R. Bachrach and Michael C. Mozer. Connectionist modeling and control of finite state systems given partial state information. 1991.

[Bachrach, 1991] Jonathan R. Bachrach. A connectionist learning control architecture for navigation. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 457–463, San Mateo, 1991. Morgan Kaufmann.

[Barto und Singh, 1990] Andy G. Barto and Satinder P. Singh. On the computational economics of reinforcement learning. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Connectionist Models, Proceedings of the 1990 Summer School*, pages 35–44, San Mateo, CA, 1990. Morgan Kaufmann.

[Barto *et al.*, 1989] Andy G. Barto, Richard S. Sutton, and Chris J. C. H. Watkins. Learning and sequential decision making. Technical Report COINS 89-95, Department of Computer Science, University of Massachusetts, MA, September 1989.

[Barto *et al.*, 1990] Andy G. Barto, Richard S. Sutton, and Chris J. C. H. Watkins. Learning and sequential decision making. In M. Gabriel and J.W. Moore, editors, *Learning and Computational Neuroscience*, pages 539–602, Cambridge, Massachusetts, 1990. MIT Press.

[Barto *et al.*, 1991] Andy G. Barto, Steven J. Bradtke, and Satinder P. Singh. Real-time learning and control using asynchronous dynamic programming. Technical Report COINS 91-57, Department of Computer Science, University of Massachusetts, MA, August 1991.

[Barto, 1989] A. G. Barto. Connectionist learning for control: An overview. Technical Report COINS TR 89-89, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, September 1989.

[Baum und Haussler, 1989] E. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.

[Baum, 1988] E. B. Baum. On the capabilities of multilayer perceptrons. *Journal of Complexity*, 4:193–215, 1988.

[Blum und Rivest, 1992] Avrim L. Blum and Ronald L. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5:117–127, 1992.

[Blum, 1989] Avrim Blum. On the computational complexity of training simply neural networks. Technical Report MIT/LCS/TR-445, MIT, Cambridge, Massachusetts, 1989.

[Chapman und Kaelbling, 1991] David Chapman and Leslie P. Kaelbling. Input generalization in delayed reinforcement learning: an algorithm and performance comparisons. In *Proceedings of IJCAI-91*, 1991.

[Chauvin, 1989] Y. Chauvin. A back-propagation algorithm with optimal use of hidden units. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, pages 519–526, San Mateo, CA, 1989. IEEE, Morgan Kaufmann.

[Chrisman, 1992] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinction approach. In *Proceedings of 1992 AAAI Conference*, Menlo Park, CA, July 1992. AAAI Press / The MIT Press.

[Dayan, 1991] Peter Dayan. *Reinforceing Connectionism: Learning the Statistical Way*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, 1991.

[Doya und Shuji, 1989] K. Doya and Y. Shuji. Adaptive neural oscillator using continuous-time backpropagation learning. *Neural Networks*, 2(5):375–385, 1989.

[Doya und Yoshizawa, 1989] K. Doya and S. Yoshizawa. Memorizing oscillatory patterns in the analog neuron network. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[Elman, 1988] J. L. Elman. Finding structure in time. Technical Report CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

[Fahlman und Lebiere, 1990] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, February 1990.

[Fahlman, 1988] S. E. Fahlman. An empirical sudy of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, CMU, 1988.

[Fahlman, 1989] Scott E. Fahlman. The recurrent cascade-correlation architecture. Technical Report CMU-CS-91-100, Carnegie Mellon University, 1989.

[Fox *et al.*, 1991] Dieter Fox, Volker Heinze, Knut Möller, Sebastian B. Thrun, and Gerd Veenker. Learning by error-driven decomposition. In Simula/Kohonen, editor, *Proceedings of International Conference on Artificial Neural Networks*, Amsterdam, 1991. Elsevier Publisher.

[Geman *et al.*, 1992] Stuart Geman, Elie Bienenstock, and Rene Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[Gherrity, 1989] M. Gherrity. A learning algorithm for analog, fully recurrent neural networks. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[Haussler, 1990] David Haussler. Decision theoretic generalizations of the pac model for neural net and other learning applications. Technical Report UCSC-CRL-91-02, Baskin Center for Computer Engineering and Information Sciences, UCSC, Santa Cruz, CA 95064, Dec. 1990.

[Hertz *et al.*, 1991] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the theory of neural computation*. Addison-Wesley Pub. Co., Redwood City, California, 1991.

[Hertzberg, 1989] Joachim Hertzberg. *Planen – Einführung in die Planerstellungsmethoden der Künstlichen Intelligenz*. BI-Wissenschafts-Verlag, Mannheim, 1989. (auf Deutsch).

[Hinton, 1989] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989. also appeared as: Technical Report CMU-CS-87-115, Computer Science Department, Carnegie-Mellon University, Pittsburgh PA, 1987.

[Jacobs und Jordan, 1991a] R.A. Jacobs and M.I. Jordan. Adaptive mixtures of local experts. *Neural Computation*, 3(1), 1991.

[Jacobs und Jordan, 1991b] Robert A. Jacobs and Michael J. Jordan. A modular connectionist architecture for learning piecewise control strategies. In *Proceedings of the American Control Conference*. Dept. of Brain and Cognitive Sciences, MIT, 1991.

[Jacobs, 1988] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[Jordan und Jacobs, 1990] Michael I. Jordan and Robert A. Jacobs. Learning to control an unstable system with forward modeling. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, San Mateo, 1990. Morgan Kaufmann Publishers.

[Jordan und Jacobs, 1992] Michael I. Jordan and Robert A. Jacobs. Hierarchies of adaptive experts. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 985–992, San Mateo, CA, 1992. Morgan Kaufmann.

[Jordan und Rosenbaum, 1988] M. I. Jordan and D. A. Rosenbaum. Action. Technical Report COINS Technical Report 88-26, MIT, Massachusetts, 1988.

[Jordan und Rumelhart, 1990] Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. submitted to Cognitive Science, 1990.

[Jordan, 1986] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Conference on Cognitive Science*, 1986.

[Jordan, 1989] Michael I. Jordan. Generic constraints on underspecified target trajectories. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE TAB Neural Network Committee.

[Judd, 1988] S. J. Judd. Neural network design and the complexity of learning. Technical Report Caltech-CS-TR-88-20, Computer Science Dept, California Institute of Technology, Pasadena, CA, September 1988. Phdthesis.

[Kaelbling, 1990] Leslie P. Kaelbling. *Learning in Embedded Systems*. PhD thesis, Department of Computer Science, Stanford University, 1990.

[Kawato *et al.*, 1990] M. Kawato, Y. Maeda, Y. Uno, and R. Suzuki. Trajectory formation of arm movement by cascade neural network model based on minimum torque-change criterion. *Biological Cybernetics*, 62:275 – 288, 1990.

[Kawato, 1990] Mitsuo Kawato. Computational schemes and neural network models for formation and control of multijoint arm trajectory. In W. Thomas III Miller, Richard S. Sutton, and Paul J. Werbos, editors, *Neural Networks for Control*, pages 197–228. MIT Press, 1990.

[Khatib, 1986a] O. Khatib. The potential field approach and operational space formulation in robot control. In K. S. Narendra, editor, *Adaptive and Learning Systems - Thaory and Applications*, pages 367–378, New York and London, 1986. Plenum Press.

[Khatib, 1986b] Oussama Khatib. Real-time obstacle avoidance for robot manipulator and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[Kindermann und Linden, 1990] J. Kindermann and J. Linden. Inversion of neural networks by gradient descent. *Parallel Computing*, 14(3):277–286, 1990.

[Knight, 1989] Kevin Knight. A gentle introduction to subsymbolic computation: connectionism for the a.i. researcher. Technical Report CMU-CS-89-150, Carnegie Mellon University, 1989.

[Koenig, 1992] Sven Koenig. The complexity of real-time search. Technical Report CMU-CS-92-145, Carnegie Mellon University, April 1992.

[Kohonen, 1988] T. Kohonen. An introduction to neural computing. *Neural Networks*, 1:3–16, 1988.

[Lang und Waibel, 1990] K. J. Lang and A. H. Waibel. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:23–43, 1990.

[Lin und Mitchell, 1992] Long-Ji Lin and Tom M. Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Submitted for publication, 1992.

[Lin, 1991a] Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of AAAI-91*, Menlo Park, CA, July 1991. AAAI Press / The MIT Press.

[Lin, 1991b] Long-Ji Lin. Self-improving reactive agents: Case studies of reinforcement learning frameworks. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 297–305, 1991.

[Lin, 1992] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning Journal*, 8, 1992.

[Linden und Kindermann, 1989] A. Linden and J. Kindermann. Inversion of multilayer nets. In *Proceedings of the First International Joint Conference on Neural Networks, Washington DC*, San Diego, 1989. IEEE.

[Linden und Weber, 1992] Alexander Linden and Frank Weber. Exploring the kinematics of a robot arm by competence modeling. Gesellschaft für Mathematik und Datenverarbeitung, St. Sugustin, Germany, 1992.

[Linden, 1990] A. Linden. Untersuchung von Backpropagation in konnektionischen Systemen. Diplomarbeit, Universität Bonn, Informatik-Institutsbericht Nr. 80, 1990.

[Lippmann, 1987] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, 1987.

[Mahadevan und Connell, 1990] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. December 1990.

[Mahadevan und Connell, 1991] Sridhar Mahadevan and Jonathan Connell. Scaling reinforcement learning to robotics by exploiting the subsumption architecture. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 328–332, 1991.

[Mahoney und Mooney, 1992] J. Jeffrey Mahoney and Raymond J. Mooney. Combining symbolic and neural learning to revise probabilistic theories. 1992.

[Mayne und Michalska, 1990] David Q. Mayne and Hannah Michalska. Receding horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, 35(7):814–824, July 1990.

[McClelland und Rumelhart, 1988] J. L. McClelland and D. E. Rumelhart. *Explorations in Parallel Distributed Processing. A Handbook of Models, Programs, and Exercises*. MIT Press, Cambridge, 1988.

[McCulloch und Pitts, 1943] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115 – 133, 1943.

[McMillan und Smolensky, 1988] C. McMillan and P. Smolensky. Analyzing a connectionist model as a system of soft rules. Technical Report CU-CS-393-88, Dept. of Computer Science, University of Colorado, Boulder, Colorado, 1988.

[Mel, 1989] Bartlett W. Mel. Murphy: A neurally-inspired connectionist approach to learning and performance in vision-based robot motion planning. Technical Report CCSR-89-17A, Center for Complex Systems Research Beckman Institute, University of Illinois, 1989.

[Mel, 1990] B. W. Mel. The sigma-pi column. Technical Report CNS 216-76, California Institute of Technology, Pasadena, April 1990.

[Miller *et al.*, 1990] W. Thomas III Miller, Richard S. Sutton, and Paul J. Werbos. *Neural Networks for Control*. MIT Press, 1990.

[Minsky und Papert, 1988] M. Minsky and S. Papert. *Perceptrons*. The MIT Press, Cambridge, Massachusetts, 1988.

[Minsky, 1954] Marvin Lee Minsky. *Theory of neural-analog reinforcement systems and its application to the brain-model problem*. Pinceton University, 1954.

[Mitchell und Thrun, 1993] Tom M. Mitchell and Sebastian B. Thrun. Explanation-based neural network learning for robot control. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann. (wird erscheinen).

[Möller und Thrun, 1990] Knut Möller and Sebastian B. Thrun. Task modularization by network modulation. In *Neuro-Nimes '90, Third International Workshop Neural Networks & their applications, Proceedings & exhibition Catalog*, November 1990.

[Möller und Thrun, 1991] Knut Möller and Sebastian B. Thrun. ARC: Adaptive Roboterkontrolle mit Künstlichen Neuronalen Netzen. *Wirtschaftsinformatik*, april, 1991. (auf Deutsch).

[Moller, 1990] Martin F. Moller. A scaled conjugate gradient algorithm for fast supervised learning. Technical Report PB-339, Aarhus Universitet, Denmark, Computer Science Department, 1990.

[Möller, 1991] Knut Möller. *Adaptive Roboterkontrolle mit Konnektionistischen Systemen*. PhD thesis, Universität Bonn, Sankt Augustin, FRG, 1991. ISBN 3-929037-07-6, ISSN 0941-5769 (auf Deutsch).

[Moody und Darken, 1989] John Moody and Chris Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[Moore, 1990a] Andrew W. Moore. Acquisition of dynamic control knowledge for a robotic manipulator. In Bruce Porter and Ray Mooney, editors, *Machine learning: proceedings of the seventh international conference*, pages 244–252. Morgan Kaufmann Publishers, Inc., June 1990.

[Moore, 1990b] Andrew W. Moore. *Efficient Memory-based Learning for Robot Control*. PhD thesis, Trinity Hall, University of Cambridge, England, 1990.

[Moore, 1991a] Andrew W. Moore. Knowledge of knowledge and intelligent experimentation for learning control. In *Proceedings of IJCNN-91*, 1991.

[Moore, 1991b] Andrew W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 333–337, 1991.

[Mozer und Bachrach, 1989] Michael C. Mozer and Jonathan R. Bachrach. Discovering the structure of a reactive environment by exploration. Technical Report CU-CS-451-89, Dept. of Computer Science, University of Colorado, Boulder, November 1989.

[Mozer und Smolensky, 1989] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. Technical Report CU-CS-421-89, University of Colorado, Boulder CO, 1989.

[Mozer, 1988] M. C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. Technical Report CRG-TR-88-3, Depts. of Psychology and Computer Science, University of Toronto, Toronto, Jun 1988.

[Munro, 1987] Paul Munro. A dual backpropagation scheme for scalar-reward learning. In *Ninth Annual Conference of the Cognitive Science Society*, pages 165–176, Hillsdale, NJ, 1987. Cognitive Science Society, Lawrence Erlbaum.

[Narandra und Thathachar, 1989] K. Narandra and M.A.L. Thathachar. *Learning automata: an introduction*. Prentice Hall, Englewood Cliffs, NJ, 1989, 1989.

[Narendra and Parthasarathy, 1990] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

[Narendra, 1991] Kumpati S. Narendra. The maturing of adaptive control. In P.V. Kokotovic, editor, *Foundations of Adaptive Control*, pages 3–36. Springer Publisher, 1991.

[Nguyen und Widrow, 1989] D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning in neural networks. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE TAB Neural Network Committee.

[Nilsson, 1982] Nils J. Nilsson. *Principles of Artificial Intelligence*. Springer Verlag, Berlin, 1982.

[Nowlan und Hinton, 1992] Steven J. Nowlan and Geoffrey E. Hinton. Adaptive soft weight tying using gaussian mixtures. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 993–1000, San Mateo, CA, 1992. Morgan Kaufmann.

[Nowlan, 1990a] S. J. Nowlan. Max likelihood competition in rbf networks. Technical Report CRG-TR-90-2, Dept. of Computer Science, University of Toronto, Toronto, February 1990.

[Nowlan, 1990b] S.J. Nowlan. Competing experts: An experimental investigation of associative mixture models. Technical Report CRG-TR-90-5, Dept. of Computer Science, University of Toronto, Canada, September 1990.

[Nowlan, 1991] Steven J. Nowlan. *Soft competitive adaption: neural network learning based on fitting statistical mixtures*. PhD thesis, Carnegie Mellon University, 1991.

[Pearlmutter, 1989] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989. also appeared as: Technical Report CMU-CS-88-191, Carnegie Mellon University, 1988.

[Pearlmutter, 1990] Barak A. Pearlmutter. Dynamic recurrent neural networks. Technical Report CMU-CS-90-196, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Pittsburgh, PA, December 1990.

[Pineda, 1987] F. J. Pineda. Generalization of backpropagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, 1987.

[Pineda, 1989] F. J. Pineda. Recurrent backpropagation and the dynamical approach to adaptive neural computation. *Neural Computation*, 1(2):161–172, 1989.

[Pomerleau, 1989] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical Report CMU-CS-89-107, Computer Science Dept. Carnegie Mellon University, Pittsburgh PA, 1989.

[Pomerleau, 1991] Dean A. Pomerleau. Rapidly adapting neural networks for autonomous navigation. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 429–435, San Mateo, 1991. Morgan Kaufmann.

[Rivest und Schapire, 1987] Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *Proceedings of Foundations of Computer Science*, 1987.

[Robinson, 1989] A. J. Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Cambridge University Engineering Dept., Cambridge, UK, February 1989.

[Rohwer und Forrest, 1987] R. Rohwer and B. Forrest. Training time-dependence in neural networks. In M. Caudill and C. Butler, editors, *Proceedings of IEEE Conference on Neural Networks*, San Diego, 1987. IEEE.

[Rosenblatt, 1962] F. Rosenblatt. *Principles of Neurodynamics*. Spartan Books, Washington, 1962.

[Rumelhart *et al.*, 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing. Vol. I + II*. MIT Press, 1986.

[Sato, 1990] M. Sato. A real time learning algorithm for recurrent analog neural networks. *Biological Cybernetics*, 62:237–241, 1990.

[Schley *et al.*, 1991] Charles Schley, Yves Chauvin, Van Henkle, and Richard Golden. Neural networks structured for control application to aircraft landing. In R. P. Lippmann, J. E. Moody, und D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 415–421, San Mateo, 1991. Morgan Kaufmann.

[Schmidhuber, 1991a] Jürgen H. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Technische Universität München, 1991.

[Schmidhuber, 1991b] Jürgen H. Schmidhuber. Reinforcement learning in markovian and non-markovian environments. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 500–506, San Mateo, 1991. Morgan Kaufmann.

[Schmidhuber, 1992] Jürgen H. Schmidhuber. A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248, 1992.

[Selfridge, 1958] O. G. Selfridge. Pandemonium: A paradigm for learning. In *Mechanisation of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory, November 1958*, London: HMSO, 1958.

[Shavlik und Towell, 1989a] Jude W. Shavlik and Geoffrey G. Towell. Combining explanation-based learning: an algorithm and empirical results. Technical Report 859, University of Wisconsin-Madison, Computer Science, 1989.

[Shavlik und Towell, 1989b] Jude W. Shavlik and G.G. Towell. An approach to combining explanation-based and neural learning algorithms. *Connection Science*, 1(3):231–253, 1989.

[Shavlik *et al.*, 1989] Jude W. Shavlik, Raymond J. Mooney, and Geoffrey G. Towell. Symbolic and neural learning algorithms: An experimental comparison. Technical report, University of Wisconsin, 1989.

[Simard *et al.*, 1992] Patrice Simard, Bernard Victorri, Yann LeCun, and John Denker. Tangent prop – a formalism for specifying selected invariances in an adaptive network. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 895–903, San Mateo, CA, 1992. Morgan Kaufmann.

[Singh, 1992a] Satinder P. Singh. The efficient learning of multiple task sequences. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 251–258, San Mateo, CA, 1992. Morgan Kaufmann.

[Singh, 1992b] Satinder P. Singh. Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning Journal*, 8, 1992.

[Śmieja, 1991] F.J. Śmieja. Multiple network systems (MINOS) modules: Task division and module discrimination. In *Proceedings of the 8th AISB conference on Artificial Intelligence, Leeds, 16–19 April, 1991*, 1991.

[Sutton, 1984] Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, Department of Computer and Information Science, University of Massachusetts, 1984.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 1988.

[Sutton, 1990] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning, June 1990*, pages 216–224, 1990.

[Sutton, 1991] Richard S. Sutton. Integrated modeling and control based on reinforcement learning and dynamic programming. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 471–478, San Mateo, 1991. Morgan Kaufmann.

[Tan, 1991] Ming Tan. Learning a cost-sensitive internal representation for reinforcement learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 358–362, 1991.

[Tesauro, 1991] Gerald J. Tesauro. Practical issues in temporal difference learning. Technical Report RC 17223 (#76307) 9/30/91, IBM Research Division, 1991.

[Tesauro, 1992] Gerald J. Tesauro. Practical issues in temporal difference learning. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 259–266, San Mateo, CA, 1992. Morgan Kaufmann.

[Thrun und Linden, 1990] S. Thrun and A. Linden. Inversion in time. In *Proceedings of the EURASIP Workshop on Neural Networks*, Sesimbra, Portugal, Feb 1990. EURASIP.

[Thrun und Mitchell, 1992] Sebastian B. Thrun and Tom M. Mitchell. Integrating inductive neural network learning and explanation-based learning. submitted for publication, 1992.

[Thrun und Möller, 1991] Sebastian B. Thrun and Knut Möller. On planning and exploration in non-discrete environments. Technical Report 528, GMD, Sankt Augustin, FRG, 1991.

[Thrun und Möller, 1992] Sebastian B. Thrun and Knut Möller. Active exploration in dynamic environments. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 531–538, San Mateo, CA, 1992. Morgan Kaufmann.

[Thrun und Śmieja, 1990] Sebastian B. Thrun and Frank Śmieja. A general feed-forward algorithm for gradient-descent in connectionist networks. Technical Report 483, GMD, Sankt Augustin, FRG, November 1990.

[Thrun *et al.*, 1990] Sebastian B. Thrun, Alexander Linden, und Knut Möller. Adaptive look-ahead planning. In J.C. Rault, editor, *Proceedings Neuro-Nimes*, November 1990.

[Thrun *et al.*, 1991a] Sebastian B. Thrun, Jerzy Bala, Eric Bloedorn, Ivan Bratko, Bojan Cestnik, John Cheng, Kenneth De Jong, Saso Džeroski, Douglas Fisher, Scott E. Fahlman, Rainer Hamann, Kenneth Kaufman, Stefan Keller, Igor Kononenko, Juergen Kreuziger, Ryszard S. Michalski, Tom Mitchell, Peter Pachowicz, Yoram Reich, Haleh Vafaie, Walter Van de Welde, Walter Wenzel, Janusz Wnek, and Jianping Zhang. The monk's problems - a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, Pittsburgh, PA, December 1991.

[Thrun *et al.*, 1991b] Sebastian B. Thrun, Knut Möller, and Alexander Linden. Planning with an adaptive world model. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 450–456, San Mateo, 1991. Morgan Kaufmann.

[Thrun, 1992a] Sebastian B. Thrun. Columbus: An autonomously exploring mobile robot. submitted for publication, 1992.

[Thrun, 1992b] Sebastian B. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, PA 15213, January 1992.

[Thrun, 1992c] Sebastian B. Thrun. The role of exploration in learning control. In David A. White and Donald A. Sofge, editors, *Handbook of intelligent control: neural, fuzzy and adaptive approaches*, Florence, Kentucky 41022, 1992. Van Nostrand Reinhold.

[Thrun, 1993] Sebastian B. Thrun. Exploration and model building in mobile robot domains. In Enrique Ruspini, editor, *Proceedings of the IEEE International Conference on Neural Networks*, March 1993.

[Valiant, 1984] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[Waibel und Lee, 1990] Alex Waibel and Kai-Fu Lee. *Readings in speech recognition*. Morgan Kaufmann Publishers, San Mateo, Californien, 1990. (eds.).

[Waibel *et al.*, 1988] A. H. Waibel, H. Sawai, and K. Shikano. Modularity and scaling in large phonemic networks. Technical Report TR-I-0034, ATR Interpreting Telephony Research Laboratories, 1988.

[Waibel, 1989] A. H. Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1:39–46, 1989.

[Wasserman, 1989] Philip D. Wasserman. *Neural computing: theory and practice*. Von Nostrand Reinhold, New York, 1989.

[Watkins, 1989] Chris J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

[Watkins, 1992] Christopher J.C.H. Watkins. Q-learning. *Machine Learning Journal*, 8, 1992.

[Werbos, 1990a] Paul J. Werbos. Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE, Vol. 78, No. 10*, pages 1550 – 1560. IEEE, October 1990.

[Werbos, 1990b] Paul J. Werbos. A menu of designs for reinforcment learning over time. In W. Thomas III Miller, Richard S. Sutton, and Paul J. Werbos, editors, *Neural Networks for Control*, pages 67–95. MIT Press, 1990.

[White und Sofge, 1990] D. White and D. Sofge. *Handbook of intelligent control: neural, fuzzy and adaptive approaches*. Van Nostrand Reinhold, 1990.

[Whitehead und Ballard, 1991] Steven D. Whitehead and Dana H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7:45–83, 1991.

[Whitehead, 1991a] Steven D. Whitehead. Complexity and cooperation in Q-learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 363–367, 1991.

[Whitehead, 1991b] Steven D. Whitehead. A study of cooperative mechanisms for faster reinforcement learning. Technical Report 365, University of Rochester, Computer Science Department, Rochester, NY, March 1991.

[Widrow und Hoff, 1960] G. Widrow and M. E. Hoff. Adaptive switching circuits. Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part4, 1960.

[Wieland, 1990] Andreas P. Wieland. Evolving control for unstable systems. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Connectionist Models, Proceedings of the 1990 Summer School*, pages 91–102, San Mateo, CA, 1990. Morgan Kaufmann.

[Williams und Zipser, 1989] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. also appeared as: Technical Report ICS Report 8805, Institute for Cognitive Science, University of California, San Diego, CA, 1988.

[Williams, 1986] R. J. Williams. Inverting a connectionist network mapping by backpropagation of error. In *8th Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 1986. Lawrence Erlbaum.

[Williams, 1987] R. J. Williams. Reinforcement-learning connectionist systems. Technical Report NU-CCS-87-3, College of Computer Science, Northeastern University, Boston, 1987.