

FORTGESCHRITTENENPRAKTIKUM:  
Implementierung und Anwendung eines  
'neuronalen' Echtzeit-Lernalgorithmus für reaktive  
Umgebungen

Josef Hochreiter, Institut für Informatik  
Technische Universität München  
Arcisstr. 21, 8000 München 2, Germany  
hochreit@kiss.informatik.tu-muenchen.de

Dezember 1990

# Kapitel 1

## Einleitung und Motivation

Bei Lernmethoden für neuronale Netze kann man zwischen ‘überwachtem’ und ‘unüberwachtem’ Lernen unterscheiden. Beim überwachten Lernen ist zu jeden Zeitpunkt bei bestimmten Eingabe extern ein gewünschter Ausgabe des Netzes bekannt, d. h. es muß nichts über die Umgebung, welche die Eingaben liefert, gewußt werden, denn der Lehrer gibt explizit den Fehler an. Somit muß das Netz auch nicht selbst Ausgabeaktionen suchen, um den Gesamtfehler des Netzes im Laufe der Zeit zu minimieren.

In der letzten Zeit ist bei der überwachten Lernmethode Backpropagation (BP) populär geworden (siehe z. B. in [21] [6] [10] [15] [24] [11]). Andere Methoden wurden in [12] [13] beschrieben. BP ist ein Verfahren von ‘gradient descent’ (Gradientenabstieg), d. h. es wird der Fehler bzw. die Fehlerfunktion (z. B.: Fehlerquadratsumme) zwischen gewünschten Ausgabe und dem aktuellen Ausgabe minimiert, indem man im ‘Gebirge der Fehlerfunktion’ den Gradienten des steilsten Abstiegs verfolgt. Und dies geschieht durch entsprechende Änderung der Gewichtsmatrix nach folgender Vorschrift:

$$E = \sum_t \|d_t - x_t\|^2,$$

$$\Delta w = -\eta \frac{\partial \sum_t \|d_t - x_t\|^2}{\partial w},$$

wo  $E$  der Fehler des Netzes,  $w$  die Gewichtsmatrix des Netzes,  $\Delta w$  die Änderung der Gewichtsmatrix nach einem Trainingszeitabschnitt ist. Ein Trainingszeitabschnitt ist eine festgelegte Zeitspanne, während der ein festes Netz Ausgaben liefert, zu welchen dann der Fehler berechnet wird.

Während dieses Trainings werden laufend Eingaben und die gewünschten Ausgaben vorgegeben. Die Variable  $t$  geht über alle Zeitschritte des Trainingsabschnitts,  $d_t$  ist der gewünschte Ausgabevektor zum Zeitpunkt  $t$ ,  $x_t$  ist der aktuelle Ausgabevektor zur Zeit  $t$ , und  $\eta$  ist eine positive Konstante. Hier nun der grundlegende Algorithmus:

Notation:

$y(t)$  ist das  $n$ -Tupel der Ausgabe der Units im Netzwerk zum Zeitpunkt  $t$ ,  
 $x(t)$  ist das  $m$ -Tupel der externen Eingabe des Netzwerkes zum Zeitpunkt  $t$ ,  
 $z(t)$  ist das  $(m+n)$ -Tupel, welches  $y(t)$  und  $x(t)$  zusammenfaßt,  
 $U$  ist eine Menge von Indizes  $k$ , so daß  $z_k$  die Ausgabe einer Unit im Netz ist,  
 $I$  ist eine Menge von Indizes  $k$ , so daß  $z_k$  eine externe Eingabe ist,  
 $W$  ist die Gewichtsmatrix des Netzes, wobei  $w_{ij}$  die Verbindung von der  $j$ -ten zur  $i$ -ten Unit darstellt,  
 $s_k(t)$  ist die Netzeingabe für die  $k$ -te Unit zur Zeit  $t$  und  $k \in U$ ,  
 $f_k$  ist die Auswertfunktion der Unit,  
 $T(t)$  ist die Menge von Indizes  $k \in U$ , für die ein gewünschter Wert  $d_k(t)$  existiert, welcher mit der  $k$ -ten Unit übereinstimmen sollte,  
 $e_k(t)$  ist der  $k$ -te Fehler für  $k \in T(t)$ ,  
 $E(\tau)$  ist der Netzwerkfehler zum Zeitpunkt  $\tau$ ,  
 $E_{total}(t_0, t_1)$  ist der Gesamtfehler zwischen Startzeitpunkt  $t_0$  und Endzeitpunkt  $t_1$ ,  
 $\delta_{ik}$  ist das Kronecker Delta,  
 $\alpha$  ist eine feste positive Lernrate.

Es ist also

$$z_k(t) = \begin{cases} x_k(t), & k \in I \\ y_k(t), & k \in U \end{cases}$$

und weiter

$$s_k(t) = \sum_{l \in U} w_{kl} y_l + \sum_{l \in I} w_{kl} x_l = \sum_{l \in U \cup I} w_{kl} z_l(t).$$

Die Aktivierung sieht wie folgt aus

$$y_k(t+1) = f_k(s_k(t)),$$

und der Fehler ist

$$e_k(t) = \begin{cases} d_k(t) - y_k(t), & k \in T(t) \\ 0, & k \notin T(t) \end{cases}$$

hieraus ergibt sich die Fehlerquadratsumme

$$E(\tau) = \frac{1}{2} \sum_{k \in U} [e_k(\tau)]^2.$$

Nun ergibt sich der Gesamtfehler zu

$$E_{total}(t_0, t+1) = \sum_{\tau=t_0+1}^{t+1} E(\tau) = E_{total}(t_0, t) + E(t+1)$$

hieraus ist ersichtlich, daß es genügt,  $E(t)$  während jedes Schrittes zu minimieren und am Ende des Trainingsabschnitts die einzelnen Gewichtsänderungen aufzusummieren. Wie in [15] auf Seite 323 ff hergeleitet wurde, soll gelten

$$\Delta w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}},$$

womit sich am Ende eines Trainingsabschnittes also folgende Gewichtsänderung ergibt

$$\sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t).$$

Da  $e_k(t)$  zu jedem Zeitpunkt und für  $k \in U$  bekannt ist, muß nur noch  $\partial y_k(t)/\partial w_{ij}$  gefunden werden, um

$$-\frac{\partial E(t)}{\partial w_{ij}} = \sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}}$$

zu erhalten. Mit der Kettenregel ergibt sich nun

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f'_k(s_k(t)) \left[ \sum_{l \in U \cup I} w_{kl} \frac{\partial z_l(t)}{\partial w_{ij}} + \delta_{ik} z_j(t) \right],$$

dies kann wegen  $\frac{\partial z_l(t)}{\partial w_{ij}} = 0$  für  $l \in I$  vereinfacht werden zu

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f'_k(s_k(t)) \left[ \sum_{l \in U} w_{kl} \frac{\partial y_l(t)}{\partial w_{ij}} + \delta_{ik} z_j(t) \right].$$

Der Anfangszustand ist unabhängig von den Gewichten, so daß gilt  $\frac{\partial y_k(t_0)}{\partial w_{ij}} = 0$  für  $k \in U, i \in U, j \in U \cup I$ . Also kann man die partiellen Ableitungen  $\frac{\partial y_k(t)}{\partial w_{ij}}$  inkrementell berechnen und durch die Variablen  $p_{ij}^k$  ersetzen, mit  $p_{ij}^k(t_0) = 0$ , somit ergibt sich

$$p_{ij}^k(t+1) = f'_k(s_k(t)) \left[ \sum_{l \in U} w_{kl} p_{ij}^l(t) + \delta_{ik} z_j(t) \right].$$

Mit der aktuellen Gewichtsänderung

$$\Delta w_{ij}(t) = \alpha \sum_{k \in U} e_k(t) p_{ij}^k$$

erhält man eine Gesamtgewichtsänderung über den Trainingszeitraum von  $t_0$  bis  $t_1$

$$\Delta w_{ij} = \sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t).$$

Verwendet man die ‘Logistic-Squashing’-Funktion  $f(x) = \frac{1}{1+e^{-x}}$ , so ergibt sich nun

$$f'_k(s_k(t)) = y_k(t+1)[1 - y_k(t+1)].$$

Der Fehler wird durch das Netzwerk durchpropagiert und nützt somit die Kenntnis der Struktur des Netzes vorteilhaft aus. Aus diesem Grund ist Backpropagation bei überwachtem Lernen sehr effizient.

Dieser Algorithmus wurde von Williams und Zipser [24] (siehe auch [11] und [7]) modifiziert, so daß keine Episoden-Grenzen mehr notwendig sind und das Netzwerk kontinuierlich lernen kann (Real-Time Backpropagation). Erreicht wird dies, indem man die Gewichtsmatrix zu jedem Zeittick neu berechnet. Der so gewonnene Algorithmus ist kein exakter Gradientenabstieg mehr, aber bei kleiner Lernrate wird dieser gut angenähert, und führt in den meisten Fällen zum Ziel. Bei dieser Version ergibt sich ein Speicherbedarf

von  $O(n^3)$  und eine Rechenzeit von  $O(n^4)$  pro Zyklus, wenn  $n$  die Anzahl der Units ist.

Die zweite Erweiterung ist die ‘Teacher-Forced’-Version des Backpropagation, d.h. die aktuelle Ausgabe wird durch die gewünschte Ausgabe ersetzt. Es muß dabei auch die angenäherte Ableitung  $p_{ij}^k$  von  $\frac{\partial y_k}{\partial w_{ij}}$  auf 0 gesetzt werden, wenn die  $k$ -te Unit eine teacher-forced Ausgabeneinheit ist. Das Teacher-Forcing ist nach Williams und Zipser bei manchen Lernaufgaben wie z. B. ‘stable oscillation’ sehr hilfreich. Nun zur Teacher-Forced Real-Time Version:

$$z_k(t) = \begin{cases} x_k(t) & \text{für } k \in I \\ d_k(t) & \text{für } k \in T(t) \\ y_k(t) & \text{für } k \in U - T(t) \end{cases}$$

$$\frac{\partial z_k(t)}{\partial w_{ij}} = \begin{cases} 0 & \text{für } k \in I \\ 0 & \text{für } k \in T(t) \\ \frac{\partial y_k(t)}{\partial w_{ij}} & \text{für } k \in U - T(t) \end{cases}$$

Nun ergibt sich folgende Berechnungsformel:

$$p_{ij}^k(t+1) = f_k(s_k(t)) \left[ \sum_{l \in U - T(t)} w_{kl} p_{ij}^l(t) + \delta_{ik} z_j(t) \right].$$

Dieser Algorithmus und der vorherige Algorithmus sind nahezu identisch, nur daß nun die Gewichtsmatrix bei jedem Zeitschritt durch  $\Delta w_{ij}(t)$  neu berechnet wird, die aktuelle Ausgabe  $y_k(t)$  einer Unit durch das Teacher-Signal  $d_k(t)$  ersetzt wird und die entsprechenden  $p_{ij}^k$  Werte auf 0 gesetzt werden.

Im Gegensatz zum überwachten Lernen ist das ‘Reinforcement’ Lernen (Fachausdruck beim Lernvorgang von Tieren) ein unüberwachtes Lernen. Hier sagt der Lehrer erst nach einiger Zeit, ob das Netzwerk samt Umgebung in einem fehlerhaften oder gewünschten Zustand ist, d. h. das Netz die richtigen oder die falschen Ausgaben geliefert hat. Die Angabe von fehlerhaften und gewünschten Zuständen kann auch kontinuierlich sein (und dies auch während des Lernvorgangs). Der Lehrer sagt aber nicht, *wie* das Netz bzw. die Umgebung den richtigen Zustand erreicht, beim überwachten Lernen hingegen wird dem System die richtige Ausgabe gegeben. Bei der Reinforcement Version des Lernens gibt der Lehrer nur eine Eingabe vor, nämlich das Reinforcement (Wert für Erfolg oder Mißerfolg), welches

skaliert angibt, ob man den gewünschten Zustand erreicht hat. Hier muß das Netzwerk etwas über seine Umgebung lernen, um diese durch Ausgaben evtl. zu beeinflussen und so den richtigen Zustand zu erreichen, fehlerhafte Ausgaben können aber zu unerwünschten künftigen Zuständen führen. Man muß also das Reinforcement, das den ungewünschten Zustand anzeigt, minimieren.

Reinforcement-Lernverfahren müssen verschiedene Ausgaben ‘durchprobieren’, um den besten Ausgabewert zu finden, während man sich beim überwachten Lernen der gewünschten Ausgabe direkt nähern kann. Aus diesem Grund benötigt das Reinforcement Learning wesentlich mehr Zeit. Dennoch gibt es viele Fälle, in denen man die gewünschte Ausgabe nicht explizit weiß, d.h. man benötigt Reinforcement Learning z. B. bei Spielen wie Schach, Dame, . . . , Robotersteuerung, Maschinensteuerung mit leicht unterschiedlichen Anfangsbedingungen, Erkennen von kausalen Zusammenhängen, Aufgaben mit gestörten Eingaben usw. . . . Einige Methoden des Reinforcement Learning werden in [1], [2], [16], [24], [19], [18] beschrieben.

Man kann aber auch Überwachtes Lernen für das Reinforcement Learning, das andernfalls sehr langsam ist, verwenden. Diese Methode wird ‘backpropagating through a model’ genannt und wurde von Munro 1987 [8] untersucht. Hierbei werden zwei Netzwerke (bei Munro sind beide ‘feedforward’, d. h. ohne rekurrente Verbindungen) durch Backpropagation trainiert. Das eine Netzwerk wird Controll-Netzwerk (kurz Controller oder C) genannt und steuert durch die Ausgabe die Umgebung. Das andere Netzwerk heißt Modell-Netzwerk (kurz Modell oder M) und soll die Beziehung zwischen Eingabe incl. Reinforcement und Ausgabe des Controllers modellieren. Bei Munro lernt erst das Modell diese Beziehung zu beschreiben, indem zufällige Eingaben angelegt werden. Danach werden die Gewichte des Modells eingefroren und der Controller lernt, indem Fehler durch das Modell in den Controller zurückpropagiert werden. Der Fehler ist dabei der Unterschied zwischen aktuellem Reinforcement und gewünschtem Reinforcement.

Robinson und Fallside [14] [12] erweiterten diesen Algorithmus, indem sie rekurrente Netzwerke verwendeten und beide parallel lernen lassen. Hier wurden die Fehler von Modell und Controller gemischt, um ein einziges Netz zu erhalten.

Modelliert das Modell nur die Reinforcement-Eingaben wie bisher, so kann nicht durch die Umgebung zurückpropagiert werden. Der Fehler für das Modell ist der Unterschied zwischen beobachtetem Reinforcement und vorhergesagtem (durch Modell) Reinforcement und der Fehler für den Controller ist der Unterschied zwischen gewünschtem Reinforcement und vor-

hergesagtem Reinforcement.

Jordan [4] hat ein ähnliches System untersucht, wobei die Ausgabe-Units des Controllers Eingabe-Units des Modells sind. Außerdem modelliert das Modell alle Eingabe-Units des Controllers, wodurch es extern vorgegebene Werte der vom Modell vorhergesagten Eingaben gibt. Der Fehler des Controllers wird wie oben beschrieben durch das Modell zurückpropagiert (siehe hierzu auch [9] [22]).



## Kapitel 2

# Beschreibung des Algorithmus

Der hier untersuchte Algorithmus wurde von Jürgen Schmidhuber 1989 entwickelt. Eine genaue Beschreibung des Algorithmus und weitere Einzelheiten findet man in [17]. Beim hier betrachteten Algorithmus wird das Reinforcement als spezielle Eingabe gesehen, wobei diese Eingabe-Units ‘Schmerzunits’ (‘Painunits’) heißen. Im Gegensatz zum normalen überwachten Lernen, wo nur die Ausgabe-Einheiten einen gewünschten Wert zu einem bestimmten Zeitpunkt besitzen, haben hier die Schmerzunits auch einen gewünschten Wert zu jedem Zeitpunkt. Bei diesem Algorithmus ist dieser Wert  $c$ , wobei meist  $c = 0$  ist, da die Schmerzunits einem negativen Reinforcement entsprechen. Ist  $r_i(t)$  die Aktivierung der  $i$ -ten Schmerzunit zum Zeitpunkt  $t$ , so soll  $\sum_{t,i} (c_i - r_i(t))^2$  minimiert werden. Die Schmerzunits können auch gewichtet werden, so daß  $\sum_{t,i} \alpha_i (c_i - r_i(t))^2$  minimiert werden muß, oder man verwendet nur die Fehlersumme  $\sum_{t,i} r_i(t)$  für  $c_i = 0$ .

Da das Modell nicht nur die Schmerzunits sondern auch alle anderen Eingabeunits vorhersagen soll (ähnlich den Taskunits bei Jordan), kann sich das Reinforcement auf ‘Credit-Assignment-Wege’ stützen, die rückwärts über die Umgebung führen. So kann der Lernalgorithmus bei bestimmten Eingaben eine Ausgabe erzeugen, so daß einen Schritt später eine gewünschte neue Eingabe entsteht, diese wird aber evtl. zu Schmerz führen, da das Netz noch nicht gelernt hatte auf diese zwar gewünschte, aber noch neue Eingabe zu reagieren. So minimiert der Algorithmus stückweise den Schmerz während der Zeit.

Hier gibt es (im Gegensatz zum Algorithmus von Robinson und Fallside)

Credit-Assignment-Wege von den Schmerzeinheiten des Modells über die Ausgabeinheiten des Controllers zu den Eingabeinheiten beim Lernen des Controllers. Die anderen Credit-Assignment-Wege beim Lernen des Modells führen von den Eingabeinheiten, die das Modell vorhersagt, zurück zu den Eingabeinheiten. Die Hauptaufgabe des Modells ist es, die Umgebungsdynamik, evtl. unter Vereinfachungen, zu simulieren und sie somit differenzierbar zu machen. Somit kann man durch die Umgebung zurückpropagieren und man weiß den Einfluß von Gewichtsänderungen des Controllers auf die Umgebung.

Das Modell erhält als Eingabe die aktuelle Eingabe des Controllers (incl. Reinforcement), sowie die aktuelle Ausgabe des Controllers (die durch die aktuelle Controllereingabe erzeugte Ausgabe) und liefert als Ausgabe die gesamte Eingabe (incl. Reinforcement) des Controllers zum nächsten Zeittick.

Beim Trainieren des Modells kann ein beliebiger überwachter Lernalgorithmus für rekurrente Netze verwendet werden, doch hier wird aus Gründen, die unten noch erläutert werden, der ‘Real-Time Teacher-Forced Backpropagation’ Algorithmus von Williams und Zipser benutzt. Durch die Eingaben der Umgebung erhält man für das Modell die gewünschten Eingaben fürs Teacher-Forcing.

Betrachtet man nun das gesamte Netzwerk (Controller und Modell) und sieht die vorhergesagten Schmerzeinheiten als Ausgabeinheiten an, welche den gewünschten Wert  $c$  haben sollen, so kann von den Schmerzeinheiten des Modells durch das Modell zurückpropagiert werden zu den Ausgabeinheiten des Controllers zurück zu den Eingaben (incl. Reinforcement). Auch hier kann jeder beliebiger überwachter Lernalgorithmus verwendet werden, wobei aber nur die Gewichte des Controllers verändert werden. Auch hier wird der ‘Real-Time’ Algorithmus von Williams und Zipser ohne Teacher-Forcing verwendet.

Da Modell und Controller rekurrent sind, kann der Algorithmus auch in Nicht-Markov-Umgebungen eingesetzt werden. D. h. vergangene Ausgaben von C und vergangene Zustände der Umgebung sind auch bestimmend für das jetzige Verhalten der Umgebung.

Der unten angeführte Algorithmus kann in zwei verschiedenen Varianten benutzt werden: die sequentielle Version und die parallele Version. In der sequentiellen Version lernt erst das Modell, wobei Trainingsbeispiele (d. h. zufällige Umgebungszustände) angelegt werden müssen und die Controllerausgabe zufällig erzeugt wird. Dadurch lernt das Modell die Controllerausgabe-Umgebungs-Dynamik, nach dem Lernen des Modells werden die Modellgewichte fixiert, und nun lernt der Controller. Hingegen lernen bei der parallelen Version Controller und Modell gleichzeitig, was nur möglich ist, da in

Abbildung 2.1: Hier ist das oben beschriebenes System skizziert. Der Einfachheit halber ist nur eine Eingabeunit (IN), nur eine Reinforcementunit (R), nur eine Hiddenunit und nur eine Ausgabeunit (OUT) dargestellt. Das Modell soll die Umgebung simulieren, indem es die Eingabe des Controllers ( $PRED_{IN}$  und  $PRED_R$ ) vorhersagt. Diese Abbildung ist aus oben erwähntem Papier von Schmidhuber.

beiden Fällen der Real-Time Algorithmus verwendet wird. Der unten angegebene Algorithmus ist die parallele Version des Algorithmus, doch erhält man die sequentielle Version hieraus, indem man Controller-Lernen und Modell-Lernen trennt und beim letzteren eine zufällige Controllerausgabe erzeugt.

Verzichtet man für obiges Lernen auf den parallelen Fall, so muß aber jede Kombination von Eingabe und Ausgabe des Controllers gelernt werden, was viel zu aufwendig sein kann. Es kann nämlich sein, daß einige Eingaben völlig irrelevant für das richtige Verhalten des Controllers sind, oder daß manche Kombinationen nicht in der Realität vorkommen. Lernt man hingegen dem Modell extern durch den Lehrer nur die wichtigsten Fälle dieser Kombinationen, so muß der Lehrer sehr viel über die Umgebung wissen. Aus diesen Gründen verwendet man den Real-Time Algorithmus von Williams und Zipser, wobei Controller und Modell parallel lernen können, außerdem kann sich das Modell evtl. nur auf die wichtigsten wirklich vorkommenden Kombinationen von Eingabe und Ausgabe des Controllers konzentrieren und muß somit nicht alles lernen. Das Modell kann sogar sehr fehlerhaft sein und muß nur die kausalen Zusammenhänge für den Schmerz modellieren, vorausgesetzt, daß die Umgebung solche kausalen Zusammenhänge erlaubt und nicht chaotisch reagiert.

Es wird kein Teacher-Forcing für das Lernen des Controllers verwendet, da man sonst die Schmerzeinheiten auf  $c$  setzen müßte. Es ist aber evtl. sinnvoll, wenn die Schmerzeinheiten etwas aktiviert sind, da das Netz so mehr Information erhält, d. h. das Netzwerk merkt z. B., daß es kurz vor einem Schmerz steht. Auch wird dadurch der sprunghafte Anstieg der Schmerzeinheiten abgeschwächt, denn dies ist wegen der fehlenden Differenzierbarkeit von Nachteil.

Im 1. Schritt der Hauptschleife werden die Ausgaben von  $C$  berechnet, die die Aktionen für die Umgebung liefern. Außerdem werden alle Ableitungen der Controllerunits nach den Controllergewichten berechnet. Nun werden im 2. Schritt die Umgebungsaktionen ausgeführt und die neuen Eingaben werden sichtbar. Im 3. Schritt versucht das Modell diese neuen Eingaben vorherzusagen, sieht sie aber noch nicht. Hier werden auch die partiellen Ableitungen der Modellunits nach den Modellgewichten berechnet. Im 4. Schritt wird das Modell so verbessert, daß es exaktere Vorhersagen macht. Die Gewichte von  $C$  werden so abgeändert, daß sich die Reinforcementunits den gewünschten Wert annähern. Für  $M$  findet noch Teacher-Forcing statt.

Der nachfolgende Algorithmus ist nur geeignet, wenn sich die Umgebung annäherungsweise linear verhält. Ist dies nicht der Fall, so müssen mehrere Zeitschritte des Netzwerkes während eines Zeitschritts der Umgebung erfol-

gen, auf diese Weise ergibt sich eine lineare Approximation der Umgebung. Führt man den 1. Schritt mehrmals aus, so erhält man mehrere Controlerticks pro Umgebungsttick, was bei komplexen Aufgaben für C gemacht werden sollte. Führt man den 3. Schritt mehrmals aus, so erhält man mehrere Modellticks pro Umgebungsttick, was für eine nichtlineare oder komplexe Umgebung notwendig ist.

## 2.1 Der Algorithmus

Hier die parallele Version des Algorithmus:

Notation :

$C$  ist die Menge aller Nichteingabeunits des Controllers,

$A$  ist die Menge aller Ausgabeunits des Controllers,

$I$  ist die Menge aller 'normalen' Eingabeunits des Controllers,

$P$  ist die Menge aller Schmerzunits des Controllers,

$M$  ist die Menge aller Units des Modells,

$O$  ist die Menge aller Ausgabeunits des Modells,

$O_P \subset O$  ist die Menge aller Units, die Schmerz vorhersagen,

$W_M$  ist die Gewichtsmatrix des Modells,

$W_C$  ist die Gewichtsmatrix des Controllers,

$y_{k_{new}}$  ist die Variable für die neu ausgewertete Aktivierung der  $k$ -ten Unit von  $M \cup C \cup I \cup P$ ,

$y_{k_{old}}$  ist die Variable für den letzten Wert von  $y_{k_{new}}$ ,

$w_{ij}$  ist der Wert des Gewichtes der gerichteten Verbindung von Unit  $j$  zu Unit  $i$ ,

$p_{ij_{new}}^k$  ist die Variable, die den aktuellen (angenäherten) Wert von  $\frac{\partial y_{k_{new}}}{\partial w_{ij}}$  angibt,

$p_{ij_{old}}^k$  ist die Variable, die den letzten Wert von  $p_{ij_{new}}^k$  angibt,

ist  $k \in P$ , dann ist  $c_k$  die gewünschte Aktivierung der  $k$ -ten Unit zu jedem Zeitpunkt,

ist  $k \in I \cup P$ , dann ist  $k_{pred}$  die Unit aus  $O$ , welche die  $k$ -te Unit vorhersagt,

$\alpha_C$  ist eine positive Konstante, die Lernrate des Controllers,

$\alpha_M$  ist eine positive Konstante, die Lernrate des Modells,

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

$|O_P| = |P|$ .

Jede Unit von  $I \cup P \cup A$  hat eine vorwärts gerichtete Verbindung zu jeder Unit aus  $M \cup C$ .

Jede Unit aus  $M$  ist mit jeder anderen Unit aus  $M$  verbunden. Jede Unit aus  $C$  ist mit jeder anderen Unit aus  $C$  verbunden.

Jedes Gewicht, das zu einer Verbindung gehört, die zu einer Unit in  $M$  führt, ist Teil von  $W_M$ .

Jedes Gewicht, das zu einer Verbindung gehört, die zu einer Unit in  $C$  führt, ist Teil von  $W_C$ .

Jedes Gewicht  $w_{ij} \in W_M$  muß  $p_{ij}^k$ -Werte haben für alle  $k \in M$ .

Jedes Gewicht  $w_{ij} \in W_C$  muß  $p_{ij}^k$ -Werte haben für alle  $k \in M \cup C \cup I \cup P$ .

**INITIALISIERUNG:**

Für alle  $w_{ij} \in W_M \cup W_C$ :

begin  $w_{ij} \leftarrow$  zufällig,

für alle möglichen  $k$ :  $p_{ij_{old}}^k \leftarrow 0, p_{ij_{new}}^k \leftarrow 0$

end,

für alle  $k \in M \cup C$ :  $y_{k_{old}} \leftarrow 0, y_{k_{new}} \leftarrow 0$ .

Für alle  $k \in I \cup P$ :

Setze  $y_{k_{old}}$  durch die Umgebungsbeobachtungen,  $y_{k_{new}} \leftarrow 0$ .

**ENDLOSSCHLEIFE:**

1. Für alle  $i \in C$ :  $y_{i_{new}} \leftarrow \frac{1}{1+e^{-\sum_j w_{ij} y_{j_{old}}}}$ ,

Für alle  $w_{ij} \in W_C, k \in C$ :

$p_{ij_{new}}^k \leftarrow y_{k_{new}}(1 - y_{k_{new}})(\sum_l w_{kl} p_{ij_{old}}^l + \delta_{ik} y_{j_{old}})$ .

Für alle  $k \in C$ :

begin  $y_{k_{old}} \leftarrow y_{k_{new}}$ ,

für alle  $w_{ij} \in W_C$ :  $p_{ij_{old}}^k \leftarrow p_{ij_{new}}^k$  end.

2. Führe alle Umgebungsaktionen aus, die auf Aktivationen der Units in  $A$  basieren.

Für alle  $i \in I \cup P$ :

Setze  $y_{i_{new}}$  durch die Umgebungsbeobachtungen.

3. Für alle  $i \in M$ :  $y_{i_{new}} \leftarrow \frac{1}{1+e^{-\sum_j w_{ij} y_{j_{old}}}}$ .

Für alle  $w_{ij} \in W_M \cup W_C, k \in M$ :

$p_{ij_{new}}^k \leftarrow y_{k_{new}}(1 - y_{k_{new}})(\sum_l w_{kl} p_{ij_{old}}^l + \delta_{ik} y_{j_{old}})$ .

Für alle  $k \in M$ :

begin  $y_{k_{old}} \leftarrow y_{k_{new}}$ ,

für alle  $w_{ij} \in W_C \cup W_M$ :  $p_{ij_{old}}^k \leftarrow p_{ij_{new}}^k$  end.

4. Für alle  $w_{ij} \in W_M$ :

$w_{ij} \leftarrow w_{ij} + \alpha_M \sum_{k \in I \cup P} (y_{k_{new}} - y_{k_{pred_{old}}}) p_{ij_{old}}^{k_{pred}}$ .

Für alle  $w_{ij} \in W_C$ :

$$w_{ij} \leftarrow w_{ij} + \alpha_C \sum_{k \in P} (c_k - y_{k_{new}}) p_{ij_{old}}^{k_{pred}}.$$

Für alle  $k \in I \cup P$ :

$$\text{begin } y_{k_{old}} \leftarrow y_{k_{new}}, y_{k_{pred_{old}}} \leftarrow y_{k_{new}},$$

$$\text{für alle } w_{ij} \in W_M : p_{ij_{old}}^{k_{pred}} \leftarrow 0,$$

$$\text{für alle } w_{ij} \in W_C : p_{ij_{old}}^k \leftarrow p_{ij_{old}}^{k_{pred}} \text{ end.}$$

## 2.2 Probleme der parallelen Version

a. Da für das Lernen beider Netzwerke der Real-Time Algorithmus verwendet wird, muß eine kleine Lernrate verwendet werden, um Instabilitäten zu vermeiden (siehe hierzu [25]).

b. Das Modell kann in ein lokales Minimum gelangen ohne das globale zu finden. In diesem Fall kann C natürlich nicht korrekt lernen.

c. Zu Anfang des Lernvorgangs ist M sehr schlecht in seiner Vorhersage, deshalb sind auch die Gewichtsänderungen in C willkürlich. Wie aber Jordan [4] und Schmidhuber und Huber [20] feststellten kann mit einem schlechten Modell auch eine Leistungsverbesserung von C erreicht werden. Dies funktioniert vor allem, wenn man als Fehler von C nicht die Differenz von gewünschter Eingabe und Modelloutput verwendet sondern die Differenz von gewünschter Eingabe und aktueller Eingabe, welche zum Teacher-Forcing im Modell benützt wird.

Damit sich C aber zu Anfang bei einem schlechten Modell nicht zu stark ändert, so daß später eine weitere Gewichtsänderung von C immer schwieriger wird, muß die Lernrate für C kleiner gewählt werden als die Lernrate von M. Wählt man die Lernrate von C zu groß, so sind zu Anfang evtl. die Gewichte von C schon sehr groß, aber C ist wegen des schlechten Modells miserabel. In diesem Fall liefert C immer dieselben fehlerhaften Ausgaben und das Modell lernt nicht alle möglichen Ausgaben von C darunter die idealen.

d. Im vorhin erwähnten Fall ist schon das größte Problem der parallelen Version angedeutet. Dies ist das 'Festfahren' ('deadlock'), das z. B. im vorhin erwähnten Fall leicht eintreten kann, wo C festgefahren ist. Hier liefert C immer fehlerhafte Ausgaben und M stellt sich auf diese ein, doch die Änderung von C geht nur sehr zäh voran, da die Gewichte von C im Betrag sehr groß sind. Während der langsamen Verbesserung von C lernt M nur immer dieselben C-Ausgaben und hat, nachdem C sich geändert hat, sehr

hohe Gewichte, die aber nur für die alten C-Ausgaben nützlich sind. Nun ist M festgefahren, hier kann man M nur sehr langsam abändern und währenddessen wird auch C laufend geändert, weg von den alten C-Ausgaben, zu denen die hohen M-Gewichte gehören. Hat sich aber nun M auf die neue C-Ausgabe eingestellt, so ist aber C schon wieder festgefahren auf die neuen Ausgaben. Während C sich die neuen Ausgaben abgewöhnt, was eben wieder sehr zäh voran geht, fährt sich M auf die neuen Ausgaben fest und vergißt dabei aber alle früher gelernten Fälle der Ausgaben von C, usw. ...

Dieses Festfahren geschieht vor allem dadurch, daß C immer dieselbe evtl. falsche Ausgabe liefert und M sich auf diese Ausgabe spezialisiert. Man kann dieses Problem nun vermeiden oder zumindest vermindern, wenn man für C *probabilistische Ausgabeunits* einführt. In den Experimenten führte in vielen Fällen erst diese Idee der Random-Units, welche Williams [23] vorschlägt, zum Erfolg der parallelen Version. Eine probabilistische Ausgabeunit  $k$  besteht aus einer gewöhnlichen Unit  $k_\mu$ , welche den Mittelwert angibt und einer anderen gewöhnlichen Unit  $k_\sigma$ , welche die Varianz erzeugt. Zu einem gegebenen Zeitpunkt berechnet sich die probabilistische Ausgabeunit  $y_{k_{new}}$  wie folgt:

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

wobei  $z$  logistisch- oder normalverteilt ist. Die entsprechenden  $p_{ij_{new}}^k$  werden wie folgt berechnet:

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

Außerdem kann der Zufall auch von außen importiert werden, indem man zu Beginn *zufällige Anfangsbedingungen* der Umgebung wählt. So sieht M auch manchmal Situationen, zu denen man sonst nicht gelangt, da C evtl. immer dieselben Ausgaben liefert. Diese Methode ist aber nicht so effektiv wie der durch probabilistische Units intern erzeugte Zufall.



## Kapitel 3

# Experimente und Untersuchungen zum Algorithmus

### 3.1 Das Flip-Flop-Experiment

Zum Testen des Algorithmus wurde das Beispiel ‘Learning Internal State’ aus [25] verwendet. C hat außer der Schmerzeingabe 3 Eingaben, welche a,b,c repräsentieren, wobei jeweils nur eine der 3 Eingaben aktiv (gleich 1.0) ist. C soll immer 0 ausgeben außer nach dem ersten Auftreten eines b’s nach einem a. Dieses b kann durch eine Folge von c’s verzögert werden, so daß C sich das Vorkommen eines a’s merken muß. Die Umgebung ist also eine *Nicht-Markov-Umgebung*.

M hat auch die 3 Eingaben, die Schmerzeingabe und zusätzlich die aktuelle Ausgabe von C als Eingaben und muß den Fehlerbetrag (Schmerz) der Sollausgabe von C zur tatsächlichen Ausgabe von C vorhersagen. Abb. 3.1 zeigt die Topologie von M und C. Da die Eingaben zufällig gewählt werden, wurde darauf verzichtet, daß M auch die Eingaben a,b,c vorhersagen muß, was nur die Lernzeit von M verlängert.

Mit dem obigen Algorithmus die Aufgabe zu lösen ist wesentlich schwieriger als bei Williams und Zipser, da das Modell sowohl die Aufgabe von C als auch die Fehlerberechnung lernen muß. Außerdem hängt die Güte von C entscheidend von der Güte von M ab.

Das Reinforcement wurde wie folgt berechnet:

$$r = 0.5 \|\text{Sollausgabe} - \text{Istausgabe}\|,$$

d. h. es wurde auf 0.5 beschränkt. In den Versuchen hat sich herausgestellt, daß dies erheblich besser ist, da somit maximale Fehler zu maximalen Gewichtsänderungen führen, denn der Faktor  $y(1 - y)$  von  $p_{ij}^k$  hat seinen maximalen Wert von 0.25 bei 0.5.

C hat eine probabilistische Ausgabeunit, die sehr hilfreich ist, wenn man M und C parallel lernen läßt. Diese Ausgabeunit ist auf  $[0 \dots 1]$  beschränkt, denn durch die Varianz, die zu  $\mu$  addiert wird kann dieser Bereich verlassen werden. C macht nur einen Zeittick (ein Zeittick von C entspricht einem Durchlauf von Schritt 1 des Algorithmus) pro Umgebungsttick und M macht 2 Zeitticks (ein Zeittick von M entspricht einem Durchlauf von Schritt 3 des Algorithmus) pro Umgebungsttick. M und C besitzen nur logistische Units und sind voll rekurrent mit Ausnahme der probabilistischen Ausgabeunit von C. Diese hat Gewichte zu allen anderen Units, bekommt aber ihre Eingabe nur von der  $\sigma$ - und  $\mu$ -Unit, wobei die  $\sigma$ -Unit Eingaben von allen anderen Units, außer der  $\sigma$ - und  $\mu$ -Unit, erhält und nur ein zufällig gewähltes Gewicht  $z$  zur Ausgabeunit besitzt. Die  $\mu$ -Unit besitzt dieselben Verbindungen wie die  $\sigma$ -Unit (die Werte sind natürlich nicht gleich), außer daß die Verbindung zur Ausgabeunit immer 1.0 ist. Das Random-Gewicht  $z$  von der  $\sigma$ -Unit zur Ausgabeunit, welches logistisch verteilt ist, wurde wie folgt berechnet:

$$y \in [0.2 \dots 0.8] \text{ gleichverteilt}$$

$$z = \ln\left(\frac{y}{1 - y}\right),$$

also

$$\text{out} = \mu + z\sigma.$$

C besitzt nur eine Hidden-Unit, dann wurden M als auch C von einer 1-Unit versorgt.

Bei allen Versuchen wurde eine Lernrate von  $\alpha_M = \alpha_C = 1.0$  für sequentielles Lernen und eine Lernrate von  $\alpha_M = 1.0$  und  $\alpha_C = 0.1$  für paralleles Lernen verwendet. Lernraten für  $\alpha_M$  von 2.0 oder 3.0 führten zu Instabilitäten, doch die Verwendung von kleineren Lernraten war auch erfolgreich,

Abbildung 3.1: *Hier die Topologie der Kombination von  $M$  und  $C$  für das Flip-Flop Experiment. Der Balken unten repräsentiert die Umgebung. Die gestrichelte Linie beinhaltet die Unterknoten für den Mittelwert und die Varianz zur Ausgabeunit von  $C$ . Die Abbildung wurde aus dem Papier von Schmidhuber entnommen.*

nahmen aber mehr Zeit in Anspruch. Beim parallelen Lernen hat sich auch eine Lernrate von  $\alpha_C = 0.2$  bewährt und führte zu schnellerem Lernen.

Beide Gewichtsmatrizen wurden zu Beginn aus dem Intervall  $[-0.1 \dots 0.1]$  vorbesetzt. Erfolgreiches Lernen wurde aus der Gewichtsmatrix von C abgelesen, denn bei erfolgreichem Lernen, d. h. C lieferte in einem Testprogramm die richtigen, gut unterscheidbaren Werte für 0 und 1, hatte C qualitativ immer dieselbe Gewichtsmatrix. Diese sieht qualitativ wie in Abb. 3.2 aus, wenn man die  $\sigma$ -Unit und die Ausgabeunit wegläßt, da  $\sigma$  nahe an 0 liegt und somit die Ausgabeunit den Wert von  $\mu$  übernimmt.

Wenn man Abb. 3.2 betrachtet, kann man folgendes Verhalten erkennen. Die Hidden-Unit ist aktiv, wenn ein b vor einer c-Folge war, und drückt so die  $\mu$ -Unit auf 0. War jedoch ein a vor einer c-Folge, so wird die  $\mu$ -Unit beim ersten Auftauchen eines b's aktiv.

Beim sequentiellen Lernen von C und M wurde M 150 000 Zyklen trainiert, bevor die Gewichte von M eingefroren wurden, wobei bei den erfolgreichen Versuchen beim Lernen des Modells eine entscheidende Verbesserung bei 60 000 - 110 000 Zyklen eintrat, die wohl hinreichend ist, um C zu trainieren. M hatte hier 3 Hidden-Units. In 6 von 10 Fällen war M gut genug, um bei beliebiger Initialisierung C ins globale Maximum zu leiten. In 2 der schlechten Fällen beim Lernen von M wurde jedesmal ein lokales Minimum bei C erreicht, d. h. es fehlte nur noch das Merkgewicht der Hidden-Unit auf sich. In den letzten beiden schlechten Fällen vom M-Lernen erreichte C einen Zustand, in dem die  $\sigma$ -Unit bei einem Auftreten eines b's auf 0.98 stieg, d. h. C suchte nach dem richtigen Verhalten beim Vorkommen eines b's.

C bekam 50 000 Trainingsbeispiele und wurde mit gutem Modell zwischen 15 000 und 25 000 Beispielen hinreichend gut. Bei beiden Lernfällen waren keine Episodengrenzen notwendig. Die Zeitdauer des Lernens von C hängt entscheidend von der Güte M's ab.

Nun zum parallelen Fall: hier wurde C und M 1 Million Zyklen trainiert.

Die Werte für den Zeitpunkt des Erfolgs bei den geglückten Versuchen in 100 000 sind nachfolgen aufgelistet.

1. Test: Es wurden 6 Hidden-Units verwendet und 40 Versuche durchgeführt, wobei 21 mal nicht erfolgreich und 19 mal erfolgreich gelernt wurde. Bei den letzteren lag der Durchschnitt des Erfolgszeitpunkts bei 700 000 Zyklen:

4.5 , 7.5 , 7.5 , 6.0 , 5.5 , 9.5 , 7.5 , 6.0 , 4.5 , 6.5 , 4.5 , 7.5 , 6.0 , 10.0 , 10.0 , 8.5 , 7.5 , 4.0 , 10.0 .

2. Test: Es wurden 5 Hidden-Units verwendet und 20 Versuche durch-

Abbildung 3.2: Gezeigt ist hier die Gewichtsmatrix von  $C$  nach erfolgreichem Lernen. Es wurde auf die  $\sigma$ -Unit und auf die Ausgabeunit verzichtet, da  $\sigma$  fast 0 ist und somit die Ausgabeunit den Wert von  $\mu$  annimmt. Die Matrix, die bei Williams und Zipser nach erfolgreichem Lernen gezeigt wird, hat auch dieses Aussehen.

geführt, wobei 10 mal nicht erfolgreich und 10 mal erfolgreich gelernt wurde. Bei den letzteren lag der Durchschnitt des Erfolgszeitpunkts bei 480 000 Zyklen:

5.0 , 5.0 , 9.5 , 4.0 , 9.0 , 5.0 , 5.0 , 6.5 , 5.0 , 4.0 .

3. Test: Es wurden 3 Hidden-Units verwendet und 50 Versuche durchgeführt, wobei 16 mal nicht erfolgreich und 34 mal erfolgreich gelernt wurde. Bei den letzteren lag der Durchschnitt des Erfolgszeitpunkts bei 480 000 Zyklen:

4.5 , 4.0 , 7.5 , 4.0 , 9.5 , 4.5 , 3.5 , 3.5 , 5.5 , 5.5 , 6.0 , 5.5 , 4.0 , 8.5 , 3.5 , 4.0 , 6.0 , 6.5 , 4.0 , 5.0 , 4.0 , 10.0 , 4.0 , 6.0 , 4.5 , 4.0 , 4.0 , 3.5 , 4.0 , 4.0 , 4.0 , 3.5 , 9.5 , 5.0 .

Die Gewichtsmatrix von C bei 1 Million Zyklen, wenn C nicht erfolgreich gelernt hat, hatte fast immer dasselbe Aussehen wie die Gewichtsmatrix von C bei geglückten Versuchen einige 100 000 Zyklen vor dem Erfolg. Es ist also sehr wahrscheinlich, daß viele der erfolglosen Versuche nach einiger Zeit doch noch zum Ziel führen würden. Bei vielen dieser Fälle fehlt nur das starke positive Gewicht der Hidden-Unit auf sich selbst, d. h. C muß nur noch lernen sich etwas zu ‘merken’.

Zu Beginn des Lernens wird die  $\mu$ -Unit von C gegen 0 gedrückt, was in  $\frac{5}{6}$  der Fälle auch richtig wäre und  $\sigma$  geht an dieser Stelle weit zurück. Als nächstes wird nur noch ein Fehler bei der Folge ‘cb’ gemacht, d. h. bei dieser Folge wird nur immer 0 oder immer 1 ausgegeben. Dies ist der Fall, da das Merkgewicht der Hidden-Unit auf sich selbst fehlt und somit bei ‘cb’ nicht unterschieden wird, ob vor einer c-Folge ein a oder ein b war. Im Fall, daß bei ‘cb’ immer 0 ausgegeben wird, liefert C in  $\frac{17}{18}$  der Fälle das richtige Ergebnis. Es wird hier die Ausgabeunit von der Hidden-Unit nicht auf 0 gedrückt, falls ein b vor einer c-Folge auftrat. Hier ist  $\sigma$  oft im Durchschnitt bei 0.01, also sehr klein und wird nur groß bei der Folge ‘cb’, wodurch noch eine Verbesserung des durchschnittlichen Fehlers erreicht wird.

Ein anderes lokales Minimum entsteht, wenn c die Hidden-Unit aktiviert und a sie löscht und dies nur für einen Zeitschritt, wobei alle anderen Gewichte wie oben sind. D. h. bei der Folge ‘cb’ entsteht immer 0, aber die  $\sigma$ -Unit steigt hier von 0.01 auf 0.2 bis 0.3 und verbessert so den mittleren Fehler.

Die  $\sigma$ -Unit erkennt also in diesen (und auch anderen, wie unten noch beschrieben wird) Fällen genau die Situationen, in denen noch Fehler auftraten und versucht durch höheres  $\sigma$  den richtigen Wert zu finden und den mittleren Fehler zu minimieren.

Abbildungen 3.3 bis 3.5 zeigen typische Verläufe der  $\sigma$ -Kurve im Falle

### KAPITEL 3. EXPERIMENTE UND UNTERSUCHUNGEN ZUM ALGORITHMUS22

erfolgreichen Lernens und Abb. 3.6 zeigt den Verlauf des maximalen Fehlers des Modells.

Noch deutlicher wurde obiger Effekt der Vergrößerung der  $\sigma$ -Unit bei fehlerhafter Ausgabe von C bei folgendem Versuch: Es wurde das Beispiel von oben verwendet, nur daß jetzt eine 1 ausgegeben wird, wenn *direkt* vor einem b ein a war. Oben war die Häufigkeit der Einsen  $\frac{1}{6}$ , so ist sie jetzt  $\frac{1}{9}$ , was dazu führte, daß C seine  $\mu$ -Unit auf 0 drückte. Die  $\sigma$ -Unit stieg jedesmal, wenn die Sollausgabe eine 1 war, stark an (0.1 auf 0.95 bei 8000 Versuchen und 0.05 auf 0.99 bei 40000 Versuchen). Da die Ausgabeunit nach unten und oben beschränkt war, führte dies dazu, daß die Hälfte der Einsen wesentlich besser vorhergesagt wurde, denn es ergibt sich für die Ausgabeunit

$$out = \begin{cases} 0, & \mu + z\sigma < 0 \\ \mu + z\sigma, & \text{sonst} \end{cases} = \begin{cases} 0, & 0 + z < 0 \\ 0 + z\sigma, & \text{sonst} \end{cases} = \begin{cases} 0, & z < 0 \\ z, & \text{sonst} \end{cases} ,$$

d. h. für  $z > 0$  ergab sich bei einer 1 als Sollausgabe eine Verbesserung des Fehlers und bei  $z \leq 0$  blieb der Fehler derselbe. C konnte dieses lokale aber nichtglobale Minimum in den meisten Fällen nicht mehr verlassen, aber  $\sigma$  zeigte deutlich, wo der Fehler zu suchen war.

Wie diese Ergebnisse zeigen ist hier die sequentielle Version oft besser und schneller als die parallele.

Abbildung 3.3: Verlauf der  $\sigma$ -Kurve beim Lernen von  $C$  im Flip-Flop Experiment. Die Varianz  $\sigma$  wurde jeweils über 20 Werte gemittelt. Gelernt wurde der perfekte Controller zwischen 500 000 und 700 000 Zyklen. Vorher befand sich  $C$  im lokalen Minimum und gab bei der Folge 'cb' immer eine 1 aus.



Abbildung 3.4: Verlauf der  $\sigma$ -Kurve beim Lernen von  $C$  im Flip-Flop Experiment. Die Varianz  $\sigma$  wurde jeweils über 20 Werte gemittelt. Gelernt wurde der perfekte Controller zwischen 400 000 und 600 000 Zyklen. Vorher befand sich  $C$  im lokalen Minimum und gab bei der Folge 'cb' immer eine 1 aus.

Abbildung 3.5: Verlauf der  $\sigma$ -Kurve beim Lernen von  $C$  im Flip-Flop Experiment. Die Varianz  $\sigma$  wurde jeweils über 20 Werte gemittelt. Gelernt wurde der perfekte Controller zwischen 650 000 und 750 000 Zyklen. Bis 400 000 Zyklen befand sich  $C$  im lokalen Minimum und gab bei der Folge 'cb' immer eine 0 aus. Der Anstieg der  $\sigma$ -Kurve bei 450 000 Zyklen entstand dadurch, daß  $\sigma$  bei den 'cb'-Folgen, bei denen eine 1 ausgegeben werden sollte, sehr stark anstieg.

Abbildung 3.6: *Es ist der Verlauf des maximalen Fehlers des Modell beim separaten Lernen im Flip-Flop Experiment dargestellt. Werte unter 0.04 sind auf 0.04 gesetzt. Es gab nach 90 000 Zyklen noch Fehler bis 0.08, aber ab 140 000 Zyklen war er nicht größer als 0.07. Ein anderes Ergebnis ist: unter 0.3 ab 55 000, unter 0.2 ab 100 000, unter 0.1 ab 130 000 (ab 100 000 sind pos. Werte unter 0.1).*

Abbildung 3.7: *Hier ist das beschriebenes System skizziert.*

## 3.2 Pole Balancing

C soll hier einen Stab auf einem Wagen balancieren, indem die Ausgabeunit von C in die Kraft umgerechnet wird, die auf den Wagen einwirkt. Es gibt zwei Begrenzungen, so daß sich der Wagen nur auf der festgelegten Strecke bewegen kann.

Wird der Stab bis zu einem bestimmten Winkel ausgelenkt (hier meistens  $12^\circ$ ) oder stößt der Wagen an die Randbegrenzungen, so wird der Versuch als fehlerhaft betrachtet. Stab und Wagen bewegen sich nur eindimensional, d. h. der Wagen kann nur nach links oder rechts bewegt werden und der Stab nur nach links oder rechts ausgelenkt werden. Das Beispiel wurde aus [2] genommen, siehe auch [5].

Das System wird durch folgende 4 Variablen vollständig beschrieben:

$x$  - Position des Wagens (Abstand von der Mitte)

$\dot{x}$  - Wagengeschwindigkeit

$\theta$  - Winkel zwischen dem Stab und der Mittelsenkrechten

auf den Wagen

$\dot{\theta}$  - Winkelgeschwindigkeit des Stabes.

Weiter die Differentialgleichung und die Variablen des Systems:

$$\ddot{\theta} = \frac{g \sin \theta + \cos \theta \frac{-F - ml\dot{\theta}^2 \sin \theta + \mu_c \operatorname{sgn}(\dot{x})}{m_c + m} - \frac{\mu_p \dot{\theta}}{ml}}{l \left( \frac{4}{3} - \frac{m \cos^2 \theta}{m_c + m} \right)},$$

$$\ddot{x} = \frac{F + ml(\dot{\theta}^2 \sin \theta - \ddot{\theta} \cos \theta) - \mu_c \operatorname{sgn}(\dot{x})}{m_c + m}$$

wobei

$-12^\circ < \theta < 12^\circ$  (Winkel des Stabes mit der Vertikalen),

$-2.4m < x < 2.4m$  (Position des Wagens auf der Spur),

$g = 9.8 \frac{m}{s^2}$  (Gravitationskonstante),

$m_c = 1kg$  (Masse des Wagens),

$m = 0.1kg$  (Masse des Stabes),

$l = 0.5m$  (halbe Stablänge),

$\mu_c = 0.0005$  (Reibungskoeffizient des Wagens mit der Spur),

$\mu_p = 0.000002$  (Reibungskoeffizient des Stabes mit dem Wagen),

$F \in [25N, -25N]$  (Kraft die auf den Schwerpunkt des Wagens angewendet wird).

(Es sei darauf hingewiesen, daß hier ein Tippfehler in [2] ist: die Gravitation ist dort gegeben als  $g = -9.8 \frac{m}{s^2}$ . Auch in [5] ist ein Fehler in der dort gegebenen Differentialgleichung).

Die beiden skalierten Eingabevariablen sind  $\bar{x} = 0.5 \left( \frac{x}{2.4} + 1 \right)$  und  $\bar{\theta} = 0.5 \left( \frac{\theta}{0.21} + 1 \right)$ .

Die Differentialgleichung wurde durch das Euler-Verfahren angenähert:

$$\theta(t) = \theta(t-1) + h\dot{\theta}(t-1)$$

$$x(t) = x(t-1) + h\dot{x}(t-1)$$

$$\dot{\theta}(t) = \dot{\theta}(t-1) + h\ddot{\theta}(t-1)$$

$$\dot{x}(t) = \dot{x}(t-1) + h\ddot{x}(t-1)$$

Man kann  $\ddot{x}(t-1)$  und  $\ddot{\theta}(t-1)$  mit der gegebenen Differentialgleichung berechnen aus den anderen Variablen zum Zeitpunkt  $t-1$ . Es ist hier zu bemerken, daß  $F(t-1)$  nur in  $\ddot{\theta}(t-1)$  und  $\ddot{x}(t-1)$  eingeht und somit auch in

Abbildung 3.8: *Der Kreis als Kurve im Phasenraum beim gleichmäßigen Pendeln.*

$\dot{\theta}(t)$  und  $\dot{x}(t)$ , d. h.  $F(t-1)$  bestimmt  $\theta(t)$  und  $x(t)$  nicht. Die Kraft hat also beim Euler-Verfahren erst im übernächsten Schritt einen Einfluß auf  $x$  und  $\theta$ . Aus diesem Grund wurden bei den Versuchen meistens 2 Eulerschritte manchmal auch bis zu 10 verwendet.

Eine weitere Beobachtung zum Eulerverfahren ist folgende. Würde man den Stab gleichmäßig zwischen  $\pm a$  hin und her pendeln lassen, so daß der Stab immer dieselbe Energie besitzt, deshalb ergibt sich im Phasenraum ein Kreis (siehe Abb. 3.8). Denn bei  $\theta = 0$  ist die Geschwindigkeit am größten und bei  $\theta = \pm a$  ist  $\theta$  betragsmäßig am größten, so daß dies im Phasenraum einen Kreis gibt. Durch das Euler-Verfahren wird aber dieser Kreis verlassen und das Verfahren tendiert dazu  $\theta$  bzw.  $\dot{\theta}$  zu vergrößern, d. h. die Energie wird vergrößert (siehe Abb. 3.9). Dieser Instabilität des Euler-Verfahrens kann durch entsprechende Krafteinwirkung aber gut entgegengewirkt werden, daher ist dieser nachteilige Effekt nicht bedeutsam.

Die Tabellen 3.1 bis 3.8 zeigen einige Versuche zum Verhalten des Systems.

Abbildung 3.9: Der Kreis als Kurve im Phasenraum wird angenähert durch das Euler-Verfahren, was zu unerwünschten Engergiezuwachs des Systems führt, aufgrund des Verfahrenfehlers.

Schrittnr.	$\theta$	$\dot{\theta}$ in $\frac{1}{s}$	$x$ in m	$\dot{x}$ in $\frac{m}{s}$	$F$ in N
1.	-0.003	-0.29	0.002	0.20	+10
2.	-0.011	-0.59	0.007	0.39	+10
3.	-0.026	-0.89	0.017	0.59	+10
4.	-0.046	-1.19	0.030	0.78	+10
5.	-0.073	-1.50	0.048	0.98	+10
6.	-0.105	-1.82	0.069	1.17	+10
7.	-0.145	-2.15	0.094	1.37	+10
8.	-0.191	-2.48	0.124	1.57	+10
9.	-0.243	-2.83	0.156	1.76	+10

Tabelle 3.1: Bei 10 Eulerschritten pro Umgebungstuck fällt der Stab bei einer konstanten Kraft von 10 N in 9 Zyklen um.

KAPITEL 3. EXPERIMENTE UND UNTERSUCHUNGEN ZUM ALGORITHMUS31

Schrittnr.	$\theta$	$\dot{\theta}$ in $\frac{1}{s}$	$x$ in m	$\dot{x}$ in $\frac{m}{s}$	$F$ in N
7.	-0.139	-1.56	0.091	0.98	-10
8.	-0.169	-1.32	0.109	0.79	-10
9.	-0.193	-1.09	0.123	0.59	-10
10.	-0.213	-0.87	0.133	0.40	-10

Tabelle 3.2: Bei 10 Eulerschritten pro Umgebungstuck fällt der Stab um, wenn man nach Anwendung von einer Kraft von 10 N eine Kraft von -10 N erst im 7. Schritt auf den Wagen einwirken läßt. Die ersten 6 Schritte sind wie vorher.

Schrittnr.	$\theta$	$\dot{\theta}$ in $\frac{1}{s}$	$x$ in m	$\dot{x}$ in $\frac{m}{s}$	$F$ in N
6.	-0.100	-1.23	0.066	0.78	-10
7.	-0.122	-0.98	0.080	0.59	-10
8.	-0.140	-0.73	0.090	0.40	-10
9.	-0.152	-0.49	0.096	0.20	-10
10.	-0.160	-0.25	0.098	0.01	-10
11.	-0.162	-0.001	0.097	-0.18	-10
12.	-0.161	0.229	0.091	-0.37	-10
13.	-0.154	0.468	0.082	-0.57	-10

Tabelle 3.3: Bei 10 Eulerschritten pro Umgebungstuck fällt der Stab NICHT um, wenn man nach Anwendung von einer Kraft von 10 N eine Kraft von -10 N im 6. Schritt auf den Wagen einwirken läßt. Die ersten 5 Schritte sind wie vorher.



KAPITEL 3. EXPERIMENTE UND UNTERSUCHUNGEN ZUM ALGORITHMUS32

Schrittnr.	$\theta$	$\dot{\theta}$ in $\frac{1}{s}$	$x$ in m	$\dot{x}$ in $\frac{m}{s}$	$F$ in N
1.	-0.003	-0.29	0.002	0.20	+10
2.	-0.006	0.0	0.004	0.0	-10
3.	-0.009	-0.30	0.006	0.20	+10
4.	-0.012	-0.01	0.008	0.0	-10
5.	-0.015	-0.30	0.009	0.20	+10
6.	-0.018	-0.02	0.012	0.0	-10
7.	-0.021	-0.31	0.014	0.20	+10
8.	-0.025	-0.03	0.016	0.0	-10
9.	-0.028	-0.33	0.017	0.20	+10
10.	-0.032	-0.05	0.020	0.0	-10
11.	-0.036	-0.35	0.021	0.20	+10
12.	-0.040	-0.07	0.024	0.0	-10
13.	-0.044	-0.38	0.025	0.20	+10
14.	-0.050	-0.10	0.028	0.0	-10
15.	-0.054	-0.41	0.030	0.20	+10
16.	-0.060	-0.13	0.032	0.01	-10
17.	-0.065	-0.44	0.034	0.20	+10
18.	-0.072	-0.17	0.036	0.01	-10
19.	-0.078	-0.49	0.039	0.21	+10
20.	-0.086	-0.22	0.040	0.01	-10
21.	-0.093	-0.54	0.042	0.21	+10
22.	-0.101	-0.28	0.045	0.01	-10
23.	-0.110	-0.61	0.047	0.21	+10
24.	-0.120	-0.35	0.049	0.02	-10
25.	-0.130	-0.68	0.051	0.21	+10
26.	-0.141	-0.43	0.054	0.02	-10
27.	-0.153	-0.77	0.055	0.22	+10
28.	-0.166	-0.53	0.058	0.02	-10
29.	-0.180	-0.87	0.061	0.22	+10
30.	-0.195	-0.64	0.063	0.03	-10
31.	-0.211	-0.99	0.066	0.23	+10

Tabelle 3.4: Bei 10 Eulerschritten pro Umgebungstuck hält sich der Stab 31 Schritte, wenn man abwechselnd +10 N und -10 N anwirken läßt.

KAPITEL 3. EXPERIMENTE UND UNTERSUCHUNGEN ZUM ALGORITHMUS33

Schrittnr.	$\theta$	$\dot{\theta}$ in $\frac{1}{s}$	$x$ in m	$\dot{x}$ in $\frac{m}{s}$	$F$ in N
1.	-0.005	-0.59	0.003	0.39	+20
2.	-0.022	-1.17	0.014	0.78	+20
3.	-0.051	-1.77	0.034	1.17	+20
4.	-0.091	-2.37	0.061	1.56	+20
5.	-0.145	-2.99	0.096	1.95	+20
6.	-0.210	-3.62	0.138	2.34	+20

Tabelle 3.5: Bei 10 Eulerschritten pro Umgebungstuck fällt der Stab bei einer Kraft von 20 N in 6 Zyklen um.

Schrittnr.	$\theta$	$\dot{\theta}$ in $\frac{1}{s}$	$x$ in m	$\dot{x}$ in $\frac{m}{s}$	$F$ in N
5.	-0.134	-1.83	0.089	1.17	-20
6.	-0.166	-1.30	0.109	0.79	-20
7.	-0.188	-0.78	0.121	0.40	-20
8.	-0.198	-0.27	0.125	0.01	-20
9.	-0.199	0.24	0.122	-0.37	-20
10.	-0.190	0.76	0.111	-0.76	-20
11.	-0.170	1.27	0.092	-1.14	-20

Tabelle 3.6: Bei 10 Eulerschritten pro Umgebungstuck fällt der Stab NICHT um, wenn man nach Anwendung von einer Kraft von 20 N eine Kraft von -20 N im 5. Schritt auf den Wagen einwirken läßt. Die ersten 4 Schritte sind wie vorher.

Schrittnr.	$\theta$	$\dot{\theta}$ in $\frac{1}{s}$	$x$ in m	$\dot{x}$ in $\frac{m}{s}$	$F$ in N
1.	-0.011	-1.17	0.007	0.78	+40
2.	-0.045	-2.35	0.030	1.56	+40
3.	-0.102	-3.54	0.068	2.34	+40
4.	-0.184	-4.73	0.122	3.12	+40
5.	-0.289	-5.93	0.191	3.90	+40

Tabelle 3.7: Bei 10 Eulerschritten pro Umgebungstuck fällt der Stab bei einer Kraft von 40 N in 5 Zyklen um.

Schrittnr.	$\theta$	$\dot{\theta}$ in $\frac{1}{s}$	$x$ in m	$\dot{x}$ in $\frac{m}{s}$	$F$ in N
4.	-0.163	-2.42	0.108	1.56	-40
5.	-0.201	-1.32	0.132	0.79	-40
6.	-0.269	-0.25	0.141	0.01	-40

Tabelle 3.8: Bei 10 Eulerschritten pro Umgebungstuck fällt der Stab um, wenn man nach Anwendung von einer Kraft von 40 N eine Kraft von -40 N erst im 4. Schritt auf den Wagen einwirken läßt. Die ersten 3 Schritte sind wie vorher.

Desweiteren fällt der Stab in 15 Schritten um, wenn man abwechselnd  $\pm 40$  N anwirken läßt. Man kann den Stab mit +10 N und -2 N balancieren. Hier kann 13 Schritte -2 N angelegt werden und trotzdem kann man mit +10 N den Stab wieder aufrichten.

Hier noch die Differenz der Variablen im 9. Schritt bei einer konstanten Kraft von +10 N, wenn man 5 Eulerschritte und 10 Eulerschritte pro Umgebungstuck vergleicht:

$$\Delta\theta = 0.0032$$

$$\Delta\dot{\theta} = 0.0065$$

$$\Delta x = 0.0018$$

$$\Delta\dot{x} = 0.0001$$

$$\Delta\ddot{\theta} = 0.0901$$

Die durchschnittliche Balancierzeit bei zufälligen Stößen aus  $\{+10N, -10N\}$  ist 28 Zyklen, hierbei wurden 1000 Versuche durchgeführt. Die Aufgabe ist nun, daß C den Stab balancieren soll und C bekommt als Eingabe nur  $x$  und  $\theta$  (und natürlich die Reinforcement-Eingaben). Dadurch wird das System für C zu einem Nicht-Markov-System, da C sich alte  $x$  und  $\theta$  ‘merken’ muß, um die Geschwindigkeiten  $\dot{x}$  und  $\dot{\theta}$  zu erkennen.

Die Ausgabe von C, hier mit  $y_{out}$  bezeichnet, liefert die Kraft wie folgt:  $F = (y_{out} - 0.5) * a$ , wobei meist  $a = 50$  gesetzt wurde, da  $y_{out}$  im linearen Bereich sein sollte, um schnelle Änderungen in ihrer Aktivierung machen zu können ( $y_{out} = 0.8$  führt zu 15 N). Außerdem zeigen obige Tabellen, daß  $\pm 20$  N ein guter Bereich ist, um den Stab kurz vor dem Umfallen noch zu retten. Die Kraft ist kontinuierlich gewählt, um die Umgebung besser differenzieren zu können, was für das Modell wichtig ist, da es die Umgebung differenzierbar approximiert. Anfängliche Versuche mit diskreten  $\pm 10$  N haben sich nicht bewährt. In einigen Fällen wurde die Ausgabeinheit linear gewählt, wobei die Aktivierung der Unit gleich der Kraft in Newton entsprach. Die Steigung wurde hierbei auf 0.2 gesetzt, um konform zur Steigung der logistischen Units im linearen Bereich zu bleiben (siehe auch [3]).

Die Eingaben wurden auf den Bereich  $[0 \dots 1]$  skaliert ( $y_{in1} = \left(\frac{\theta}{max.Winkel} + 1\right) * 0.5$ ), was zu einer sehr nachteiligen Asymmetrie in der Eingabe führt. So kann die Auslenkung des Stabes nach rechts, welche zu einer Eingabe nahe an 1 führt, direkt über die Ausgabeinheit in eine Kraft umgesetzt werden, die den Stab wieder nach links bewegt. Bei einer Auslenkung nach links hingegen, muß dies indirekt über andere Units (z. B. der 1-Unit) geschehen, da die Eingabe nahe an 0 liegt. Als Verbesserung kann man lineare Eingabeunits verwenden, die auch negativ werden können, womit die Symmetrie erhalten bleibt. Außerdem kann die Aufgabe den Stab von links aufzustellen schon gelöst sein, wenn man den Stab von rechts aufstellen kann. Eine andere Möglichkeit wurde in [5] verwendet, wo eine eigene Vorzeichenunit verwendet wird und ansonsten die Eingabe im Betrag angegeben wird.

Für das Reinforcement wurden verschiedene Wege gewählt, es zu berechnen. Der einfachste Fall ist ein Reinforcement, welches auf 0.5 gesetzt wird, falls der Stab den maximalen Winkel überschreitet oder der Wagen die Randbegrenzungen berührt. Man kann diesen Fall auf zwei Reinforcementunits aufteilen, wobei eine für das Umfallen des Stabes und eine für das

Anstoßen an den Rändern verantwortlich ist. Diesen Fall wiederum kann man auf vier Reinforcementunits aufteilen, wobei je eine Unit für Stab nach links umfallen, Stab nach rechts umfallen, Wagen an linken Rand stoßen und Wagen an rechten Rand stoßen zuständig ist. Dieser Fall führt zu sprunghaften Anstieg des Reinforcements im Fehlerfalle, hierbei hat das Modell Schwierigkeiten dies zu simulieren. Die obigen Fälle kann man nun für kontinuierliches Reinforcement verwenden, d. h. je weiter der Stab ausgelenkt wird, umso stärker ist der Schmerz bzw. je weiter der Wagen von der Mitte wegplaziert ist, umso stärker ist der Schmerz. Für diese Fälle wurden als Fehler  $(\theta, x)$ ,  $(\theta^2, x^2)$ ,  $(\theta^3, x^3)$  betrachtet. Es wurde auch experimentiert mit Reinforcementfunktionen wie z. B.  $r_1 * (\theta + \alpha\dot{\theta})^2 + r_2 * (x + \beta\dot{x})^2$  oder für das  $\theta$ -Reinforcement  $\begin{cases} \theta, & \theta, \dot{\theta} \text{ gleiches Vorzeichen} \\ 0, & \text{sonst} \end{cases}$ .

Als letztes wurde noch eine Funktion benutzt, welche im Intervall  $[-d \dots + d]$  den Wert 0 besitzt und dann stetig bis zu dem maximalen Winkel auf 0.5 ansteigt, für  $x$  wurde eine ähnliche Funktion gewählt.

Das kontinuierliche Reinforcement kann noch verbessert werden, indem man statt  $\theta(t)$  und  $x(t)$  die Werte  $\theta(t + 1)$  und  $x(t + 1)$  verwendet, da diese Werte bei 2 Eulerschritten durch die vom Controller gelieferten Kräfte schon zum Zeitpunkt  $t$  determiniert sind. Bei mehr Eulerschritten kann man auch schon die Werte von  $\theta$  und  $x$  des ersten Eulerschrittes des nächsten Umgebungsticks verwenden, da diese Werte determiniert sind.

Für den Fall des nichtkontinuierlichen Reinforcements bekommt C nur einmal pro Versuch Schmerz und muß diesen auf frühere Fehler zurückführen. Außerdem darf die Gewichtsänderung auch nicht zu stark sein, damit ein spezieller Fehler nicht übergewichtet wird.

Bei einer oder zwei Reinforcementunits gibt es wegen der Asymmetrie in der Eingabe auch eine Asymmetrie bei der Vorhersage des Modells, d. h. links oder rechts Umfallen bzw. Anstoßen wird gut vorhergesagt und der entgegengesetzte Fall schlecht. In fast allen Fällen führte eine Näherung der Eingabe an 1 zum Anstieg der entsprechenden Reinforcementunit, diese sollte aber auch bei einer Näherung der Eingabe an 0 ansteigen, was aber nicht zu beobachten war. Es sollten also bei nichtkontinuierlichen Reinforcementunits und asymmetrischer Eingabe 4 Reinforcementunits verwendet werden. Dieser Effekt der Asymmetrie ist bei kontinuierlichen Reinforcement bei weitem nicht so ausgeprägt. Beim Modell haben sich bei nichtkontinuierlichem Reinforcement logistische Reinforcementunits besser bewährt als lineare.

Doch auch die kontinuierlichen Reinforcementunits haben einen ent-

scheidenden Nachteil. Angenommen der Controller wäre perfekt, so würde während des Balancierens die Gewichte von C laufend so verändert, daß C nicht mehr perfekt wäre. Dieser Effekt ist nicht nur störend, falls C perfekt ist, sondern er ist in der gesamten Lernphase hemmend. Beispielsweise war es oft der Fall, daß C in der Mitte der Spur eine gewisse Zeit den Stab relativ ruhig in der Vertikalen halten konnte und dann nach rechts oder links abdriftete. Nun war C schon so gut, daß er den Stab auch einige Zeit am Rande balancieren konnte, aber dort mit anderen Balanciermethoden (z. B. den Stab schräg halten und nicht senkrecht oder einen starken Stoß in die eine Richtung und viele schwache Stöße in die andere Richtung geben), da der Rand gefährlich nahe war. Aber durch das Balancieren am Rande verlernte C das Balancieren in der Mitte, was C durch mehrere Versuche wieder erlernen mußte.

Bei linearem Fehler gibt es noch den Nachteil, daß der Zuwachs des Fehlers von  $0^\circ$  auf  $1^\circ$  genauso gewichtet ist, wie der Zuwachs von  $11^\circ$  auf  $12^\circ$ , obwohl man beim Balancieren den Stab oft ein wenig auslenken muß.

Wählt man als Anfangsbedingung die Wagenposition zufällig zwischen den Rändern, um auch zu Anfang Situationen zu schaffen, die nahe an den Rändern liegen, so übt C oft eine Kraft aus, die den Wagen weiter in die Mitte rollt. Doch dies bringt den Stab aus dem Gleichgewicht und die Balancierzeit verkürzt sich erheblich. Besser wäre es, wenn C den Stab mit Wagen langsam in die Mitte der Bahn balancieren würde.

Was obige Fehlerfunktionen auch nicht berücksichtigen, ist der Fall einer Auslenkung  $\theta_0$  mit einer Winkelgeschwindigkeit  $\dot{\theta}_0$ , welche  $\theta$  in Richtung  $0^\circ$  bringt. Dieser Fall sollte wenig oder keinen Schmerz erzeugen, ist hingegen  $\dot{\theta}_0$ , so daß  $\theta$  in Richtung  $\pm 12^\circ$  verändert wird, so sollte größerer Schmerz erzeugt werden evtl. abhängig von der Größe von  $\dot{\theta}_0$ . Bei obigen Fehlerfunktionen ist aber beides gleich bewertet.

Um diese Probleme zu verkleinern, könnte man versuchen das Reinforcement während der Zeit abnehmen zu lassen, um so langes Balancieren zu belohnen. Besser ist das Reinforcement als Vorhersage des zu erwartenden akkumulativen Schmerzes zu sehen, ähnlich wie in [5] mit der Methode der Temporalen Differenzen von Sutton, wie Klaus Bergner für Feedforwardnetze herausfand. Doch bei rekurrenten Netzen und nichtkontinuierlichen Reinforcement wurden die Reinforcementunits auf 0 gedrückt, da sie am Ende eines Versuchs meist bei 0 sind (z. B. die Schmerzunits der Randbegrenzung) und somit diese 0 auch zurückreichen. Außerdem gab es Gewichtsexplosionen aufgrund des festen perfekten Modells, durch welches die Vergangenheit des Systems berücksichtigt wurde.

Die Versuchsergebnisse sind weiter unten in Tabellenform aufgelistet.

### 3.2.1 Lernen des Modells

Hier wurden meist zwischen 3 und 7 Hidden-Units verwendet. Als Eingabe bekam das Modell die Controller-Ausgabe, die skalierten Werte  $x$  und  $\theta$  und dann noch das Reinforcement. Als Ausgabe liefert M den nächsten Wert von  $x$  und  $\theta$  und das nächste Reinforcement. Außerdem besitzt M eine 1-Unit und macht pro Umgebungstick 2 eigene Ticks. Es wurde eine Lernrate von  $\alpha_M = 1.0$  verwendet und meist 80 000 Versuche gelernt.

Um die Anfangsbedingung zu schaffen wurde zufällig zwischen 1 und 15 Schläge von  $\pm 10$  N auf den Wagen ausgeübt, welcher zuerst mit senkrechtem Stab mit Winkelgeschwindigkeit 0 und Geschwindigkeit 0 beliebig zwischen den Rändern stand. Stieß hierbei der Wagen an den Rand oder überschritt der Stab die  $12^\circ$  Grenze, so wurde erneut gestartet. Nach diesen 10 Schlägen bekam das Modell das System zu Gesicht. Das Modell machte auch nach sehr langer Lernphase bei den ersten beiden Umgebungsticks einen Fehler von bis zu 0.5 bei der Vorhersage von  $\theta$ , dann bewegte sich der Fehler zwischen 0.08 und 0.02. Gegen Ende des Versuchs, wenn der Stab maximal ausgelenkt war, ergab sich nochmals ein Fehler von bis zu 0.2 bei der Vorhersage von  $\theta$ . Der große Fehler zu Anfang ergibt sich, da das Modell die Vorgeschichte nicht sah und sich auf die neue Situation einstellen muß. Der größere Fehler am Ende des Versuchs oder bei starker Auslenkung des Stabes rührt daher, daß bei einer starken Auslenkung der Sollwert der Vorhersageunit von  $\theta$  nahe bei 1 oder bei 0 liegt, dieser Wert wird aber nicht erreicht. Wegen der schnellen Änderungen bewegt sich die  $\theta$ -Unit nur im linearen Bereich, also maximal zwischen 0.2 und 0.8.

Wie die Abb. 3.10 zeigt wird der tatsächliche Wert von  $\theta$  auf 0.2 bis 0.8 transformiert. Aus diesem Grund wurde bei einigen Versuchen die Eingabe auf  $[0.3 \dots 0.7]$  transformiert, was zu besserem Verhalten an den Rändern führte.

An den Umkehrpunkten sieht man oft eine Verzögerung um einige Ticks (bis zu 8 Ticks), d. h.  $\theta$  wird in der Realität wieder kleiner, nachdem es zugenommen hatte, doch das vorhergesagte  $\theta$  wächst noch immer. Man kann oft auch an einem Umkehrpunkt ein ‘Hinterherhinken’ und am andern Umkehrpunkt ein ‘Vorauslaufen’ des vorhergesagten  $\theta$ 's beobachten, wodurch die Frequenz eingehalten wird und die Vorhersage um den Nullpunkt gut ist.

Bei  $x$  kann man diesen Effekt nicht so deutlich sehen, da sich  $x$  nicht so

Abbildung 3.10: *Es ist skizziert, wie sich der Stab tatsächlich bewegt und was das Modell vorhersagt.*



Abbildung 3.11: *Es ist der Verlauf der Aktivierung der Reinforcementunit gezeigt, welche den Fehler anzeigen soll, den das System begann.*

schnell ändert und nicht so oft den Rand erreicht.

Bei nichtkontinuierlichen Reinforcementunits ergibt sich am Ende des Versuchs immer ein relativ großer Fehler. Die Reinforcementunits sollen während des Balancierens 0 sein und bei einem Fehler sollte eine Unit auf 0.5 steigen. Doch durch die lange 0-Phase werden die Reinforcementunits auf 0 gedrückt und steigen am Ende höchstens auf Werte zwischen 0.15 bis 0.2.

Den frühzeitigen Anstieg der Reinforcementunit, der anzeigt, daß bald ein Fehler zu erwarten ist, könnte man unter Umständen zum Lernen von C verwenden. Dieser Anstieg zeigt nämlich an, daß das System sich in einer ungünstigen Situation befindet, in der bisher in Kürze ein Fehler zu erwarten war. Diese Situation könnte nun C lernen zu vermeiden, wenn man beim Lernen von C nicht den tatsächlichen Schmerz verwendet (Teacher-Forcing), sondern die Vorhersage vom M.

Der große Fehler zu Anfang jedes Versuchs kann vermieden werden, indem man das Modell einige Ticks mitlaufen läßt, ohne daß es eine korrekte Vorhersage machen muß (kein Error). Deshalb wurde nach Justierung der Anfangsbedingung dem Modell 2 Ticks kein Error gegeben. Am Ende des

Lernens ergab sich hier ein maximaler Fehler von 0.06 - 0.08, wenn man die maximale Auslenkung von  $\theta$  nicht berücksichtigt.

Der andere große Fehler von  $\theta$  bei starker Auslenkung des Stabes kann vermieden werden, wenn man eine lineare Unit verwendet. Es wurde deshalb eine lineare Unit mit Steigung 0.2 verwendet, in diesen Fall wurde  $x$ , welches sich zwischen  $\pm 2.4$  bewegt durch 10 geteilt und zur Vorhersage von  $x$  auch eine lineare Unit benützt. Bei kontinuierlichen Reinforcement wurden für das Reinforcement auch lineare Units benützt. Dies führt bei der Vorhersage von  $\theta$  zu einem maximalen Fehler von 0.007 - 0.02 und zu einem durchschnittlichen Fehler von 0.003 - 0.007 pro Versuch.

Hieraus wird auch ersichtlich, daß am Ende des Lernvorgangs das Modell nicht nur die aktuelle Umgebung (hier das aktuelle  $\theta$ ) vorhersagt sondern die zukünftige Umgebung. Zwischendurch lernt das Modell nur die aktuelle Umgebung, welche  $M$  als Eingabe erhält, als zukünftige Umgebung vorherzusagen, und erst später die Umgebung im nächsten Schritt vorherzusagen.

Bei linearen Units führt eine zu hohe Lernrate z. B. 3.0 zu Instabilitäten. Die Umgebung ändert sich nur sehr wenig, d. h. der relative Unterschied von einem Umgebungstuck zum nächsten ist sehr gering. Aus diesen Grund darf man auf eine entscheidende Verbesserung des Algorithmus hoffen, falls man das Modell die Änderung der Umgebung, statt die Umgebung selbst, vorhersagen läßt, wie in [3] vorgeschlagen wird.

‘Aufblähen’ der Eingabeunits auf größere Werte brachte auch keine entscheidende Verbesserung.

### 3.2.2 Probleme mit Modell mit festen Gewichten

Sind die Gewichte des Modells betragsmäßig zu groß, so ergeben sich beim Lernen von C Instabilitäten. Die  $p_{ij}^k$  sind in diesen Fällen unbeschränkt. Im folgenden wird nur der lineare Fall mit Steigung 1 betrachtet.

$$p_{ij}^k(t+1) - p_{ij}^k(t) = \left( \sum_{l \in U} w_{kl}(t) p_{ij}^l(t) + \delta_{ik} z_j(t) \right) - \left( \sum_{l \in U} w_{kl}(t-1) p_{ij}^l(t-1) + \delta_{ik} z_j(t-1) \right) = \left( \sum_{l \in U} w_{kl}(t) p_{ij}^l(t) - w_{kl}(t-1) p_{ij}^l(t-1) \right) + \delta_{ik} (z_j(t) - z_j(t-1))$$

Bei normalem Backpropagation können also große Schwankungen der Units, vor allem sind dies die Eingabe- und Ausgabeunits bzw. die probabilistischen Units, zu Instabilitäten führen.  $w_{kl}$  wird erst groß, wenn  $p_{kl}^s$  groß ist, also  $\Delta p_{kl}^s$  groß ist. Damit aber  $\Delta p_{kl}^s$  immer dasselbe Vorzeichen hat, müßte  $z_j$  monoton wachsen (fallen), was meistens nicht der Fall ist. Alterniert hingegen  $\Delta p_{kl}^s$ , so werden die Gewichte nicht so schnell wachsen. Dieses Problem ist aus diesem Grund nicht sehr ausgeprägt und kann vermutlich vernachlässigt werden. Hierzu wurde aber doch ein Versuch gestartet. Ein Netz hat 3 Eingaben und soll 1 ausgeben, wenn genau 2 Eingaben aktiv sind, dabei besitzt das Netz 2 Hidden-Units. Die Lernrate war 1.0 und es wurde abgebrochen, falls ein  $p_{ij}^k > 0.25 = \max_{y \in [0..1]} y(1-y)$  war. Diese Abbruchgrenze wurde gewählt, da logistische Units verwendet wurden. Gelernt wurde mit dem Real-Time Teacher-Forcing Algorithmus von Williams und Zipser. Verwendet man bei 20 Versuchen als Einswert für die Eingaben  $e = 1.0$ , so ergibt sich durchschnittlich ein Abbruch nach  $A = 4.55$  Zyklen, ist  $e = 0.3$  so ist  $A = 44$  und bei  $e = 0.7$  ist  $A = 25$ . Bei  $e = 1.0$  ergibt sich die Instabilität durch die Eingabeunits und bei den letzten beiden Fällen ergibt sich die Instabilität durch die Ausgabeunit. Bemerkenswert ist, daß ohne Teacher-Forcing kein Abbruch mehr zu beobachten ist. Betrachtet man  $e = 0.7$  und statt Einsausgabe  $out = 1.0$  die Einsausgabe  $out = 0.8$ , so gibt es keinen Abbruch und bei  $out = 0.97$  werden 50 % der Fälle abgebrochen. Wählt man  $out = 0.99$ , so liefert dies dasselbe Resultat wie  $out = 1.0$ . Dies zeigt, daß bei üblichen Lernalgorithmen die hier genannten Instabilitäten nicht besonders relevant sind und wohl selten auftreten, wobei man evtl. bei probabilistischen Units vorsichtig sein sollte.

Nun zu den hier betrachteten Algorithmus, wobei nun  $k$  eine von  $M$  vorhergesagte Reinforcementunit sein soll. Beim Lernen von C ist  $w_{kl}(t) = w_{kl}(t-1)$  und  $\delta_{ik} = 0$ , da  $w_{ij} \in W_C$ . Es ist hier also

Abbildung 3.12: *Theoretisches Beispiel:  $w_1, w_2$  fest;  $w_3$  variabel.*

$$p_{ij}^k(t+1) - p_{ij}^k(t) = \sum_{l \in U} w_{kl} (p_{ij}^l(t) - p_{ij}^l(t-1)).$$

Betrachtet man nur das Gewicht  $w_{kk}$ , so ergibt sich nach  $n$  Schritten:

$$p_{ij}^k(t+n) - p_{ij}^k(t+n-1) \approx w_{kk}^n (p_{ij}^k(t) - p_{ij}^k(t-1)),$$

exakter ergibt sich

$$p_{ij}^k(t+n) - p_{ij}^k(t+n-1) = (w_{kk}^n - (n-1)w_{kk}^{n-1} \pm \dots) p_{ij}^k(t) - (w_{kk}^n - (n-2)w_{kk}^{n-1} \pm \dots) p_{ij}^k(t-1).$$

Es kann also zu einer Aufschaukelung bei großen festen Gewichten im Modell kommen.

Nun noch ein einfaches recurrentes Netzwerk mit zum Teil festen Gewichten in theoretischer Betrachtung, um eine Gewichtsexplosion zu verdeutlichen.

Beispiel 1 siehe Abb. 3.12:

$$\begin{aligned} p_3^a &= w_1 p_{3old}^a + w_2 p_{3old}^b \\ p_3^b &= i n_{old} \\ p_3^a(t) &= \end{aligned}$$

$$\begin{aligned}
 w_1 p_3^a(t-1) + w_2 p_3^b(t-1) &= w_1 [w_1 p_3^a(t-2) + w_2 p_3^b(t-2)] + w_2 in(t-2) = \\
 &w_1^2 p_3^a(t-2) + w_1 w_2 p_3^b(t-2) + w_2 in(t-2) = \\
 w_1^2 [w_1 p_3^a(t-3) + w_2 p_3^b(t-3)] + w_1 w_2 in(t-3) + w_2 in(t-2) &= \\
 w_1^3 p_3^a(t-3) + w_1^2 w_2 p_3^b(t-3) + w_1 w_2 in(t-3) + w_2 in(t-2) &= \\
 &\vdots \\
 = w_1^n p_3^a(t-n) + w_1^{n-1} w_2 p_3^b(t-n) + w_1^{n-2} w_2 in(t-n) + \dots + w_1 w_2 in(t-3) + w_2 in(t-2) &= \\
 w_1^n p_3^a(t-n) + w_1^{n-1} w_2 p_3^b(t-n) + w_2 \sum_{k=2}^n w_1^{k-2} in(t-k) &
 \end{aligned}$$

Die letzte Formel kann leicht durch Induktion bewiesen werden. Das System ist also instabil für  $|w_1| > 1$ .

Tickt das Modell, welches nur aus der Unit  $a$  besteht,  $u$  mal, so ergibt sich:

$$\begin{aligned}
 p_3^a(t) &= \\
 w_1 p_3^a(t-1) + w_2 p_3^b(t-1) &= w_1 [w_1 p_3^a(t-2) + w_2 p_3^b(t-2)] + w_2 p_3^b(t-2) = \\
 &w_1^2 p_3^a(t-2) + w_1 w_2 p_3^b(t-2) + w_2 p_3^b(t-2) = \\
 w_1^2 [w_1 p_3^a(t-3) + w_2 p_3^b(t-3)] + w_1 w_2 p_3^b(t-3) + w_2 p_3^b(t-2) &= \\
 w_1^3 p_3^a(t-3) + w_1^2 w_2 p_3^b(t-3) + w_1 w_2 p_3^b(t-3) + w_2 p_3^b(t-2) &= \\
 &\vdots \\
 = w_1^u p_3^a(t-u) + w_2 p_3^b(t-u) \sum_{k=0}^{u-1} w_1^k &= \\
 w_1^u p_3^a(t-u) + w_2 p_3^b(t-u) \frac{w_1^u}{w_1 - 1} &= \\
 w_1^u p_3^a(t-u) + p_3^b(t-u) \frac{w_2 (w_1^u)}{w_1 - 1} &
 \end{aligned}$$

Auch hier ergibt sich die Instabilität für  $|w_1| > 1$ .

Wie dieser Fall zeigt, kann es bei großen, festen Modellgewichten zu Aufschaukelungen kommen, und die  $p_{ij}^k$ 's können explodieren. Bei hohen Modellgewichten können frühere Fehler mehr Einfluß auf die Gewichtsänderung haben, als die aktuellen Fehler.

Es sollte also beim Modell die Gewichte  $w_{ij}$  betragsmäßig auf 1.0 beschränkt werden und bei logistischen Units genügt eine Beschränkung auf 4.0, da bei den  $p_{ij}^k$ 's noch der Faktor  $|y(1-y)| \leq 0.25$  zu berücksichtigen ist.

Durch weitere Beschränkung der Gewichte auf kleinere Werte kann der Einfluß der früheren Fehler weiter vermindert werden. Um weiter zurückliegende Fehler, die im Netz kursieren, zu vergessen, kann man die  $p_{ij}^k$  explizit auf 0 setzen oder man verzichtet auf Rekurrenz im Netz.

### 3.2.3 Verwendung eines perfekten Modells

Um die Probleme mit einem neuronalen Modell zu vermeiden, wird ab jetzt durch die Gleichungen propagiert, d. h. das Modell ist *perfekt*. Dadurch erhält man ein die Controller-Umgebungs-Dynamik ideal beschreibendes Modell und man kann sich auf das Lernen von C konzentrieren.

Hier nun die akkumulativen Gleichungen für dieses Modell:

Die Differentialgleichungen lauten:

$$\ddot{\theta} = \frac{g \sin \theta + \frac{-F - ml\dot{\theta}^2 \sin \theta + \mu_c \operatorname{sgn}(\dot{x})}{m_c + m} \cos \theta - \frac{\mu_p \dot{\theta}}{ml}}{l\left(\frac{4}{3} - \frac{m \cos^2 \theta}{m_c + m}\right)} =: \frac{\text{numerator}}{\text{denominator}} =: \frac{\text{numerator}}{\text{denominator}}$$

$$\ddot{x} = \frac{F + ml(\dot{\theta}^2 \sin \theta - \ddot{\theta} \cos \theta) - \mu_c \operatorname{sgn}(\dot{x})}{m_c + m}$$

Das Eulerverfahren liefert:

$$\theta(t) = \theta(t-1) + h\dot{\theta}(t-1)$$

$$x(t) = x(t-1) + h\dot{x}(t-1)$$

$$\dot{\theta}(t) = \dot{\theta}(t-1) + h\ddot{\theta}(t-1)$$

$$\dot{x}(t) = \dot{x}(t-1) + h\ddot{x}(t-1)$$

Damit ergibt sich:

$$\frac{\partial \theta(t)}{\partial w_{ij}} = \frac{\partial \theta(t-1)}{\partial w_{ij}} + h \frac{\partial \dot{\theta}(t-1)}{\partial w_{ij}}$$

$$\frac{\partial x(t)}{\partial w_{ij}} = \frac{\partial x(t-1)}{\partial w_{ij}} + h \frac{\partial \dot{x}(t-1)}{\partial w_{ij}}$$

$$\frac{\partial \dot{\theta}(t)}{\partial w_{ij}} = \frac{\partial \dot{\theta}(t-1)}{\partial w_{ij}} + h \frac{\partial \ddot{\theta}(t-1)}{\partial w_{ij}}$$

$$\frac{\partial \dot{x}(t)}{\partial w_{ij}} = \frac{\partial \dot{x}(t-1)}{\partial w_{ij}} + h \frac{\partial \ddot{x}(t-1)}{\partial w_{ij}}$$

$$\frac{\partial F(t)}{\partial w_{ij}} = a \frac{\partial y_{out}}{\partial w_{ij}}, \text{ für } F(t) = a(y_{out}(t) - c)$$

Die Kettenregel liefert hier:

$$\frac{\partial \ddot{\theta}(t)}{\partial w_{ij}} = \frac{\partial \ddot{\theta}(t)}{\partial \theta(t)} \frac{\partial \theta(t)}{\partial w_{ij}} + \frac{\partial \dot{\theta}(t)}{\partial \dot{\theta}(t)} \frac{\partial \dot{\theta}(t)}{\partial w_{ij}} + \frac{\partial \ddot{\theta}(t)}{\partial F(t)} \frac{\partial F(t)}{\partial w_{ij}}$$

$$\frac{\partial \ddot{\theta}(t)}{\partial \theta(t)} = \frac{\frac{2lm \cos \theta(t) \sin \theta(t)}{m_c + m} \text{nenner} - \left( g \cos \theta(t) - \text{kla1} \sin \theta(t) + \cos \theta(t) \frac{-ml \dot{\theta}^2(t) \cos \theta(t)}{m_c + m} \right) \text{teiler}}{\text{teiler}^2}$$

$$\frac{\partial \ddot{\theta}(t)}{\partial \dot{\theta}(t)} = \frac{-2ml \dot{\theta}(t) \cos \theta(t) \sin \theta(t)}{m_c + m} - \frac{\mu_p}{ml}$$

$$\frac{\partial \ddot{\theta}(t)}{\partial F(t)} = \frac{-\frac{\cos \theta}{m_c + m}}{\text{teiler}}$$

$$\frac{\partial \ddot{x}(t)}{\partial w_{ij}} = \frac{\partial \ddot{x}(t)}{\partial \theta(t)} \frac{\partial \theta(t)}{\partial w_{ij}} + \frac{\partial \dot{x}(t)}{\partial \dot{\theta}(t)} \frac{\partial \dot{\theta}(t)}{\partial w_{ij}} + \frac{\partial \ddot{x}(t)}{\partial \ddot{\theta}(t)} \frac{\partial \ddot{\theta}(t)}{\partial w_{ij}} + \frac{\partial \ddot{x}(t)}{\partial F(t)} \frac{\partial F(t)}{\partial w_{ij}}$$

$$\frac{\partial \ddot{x}(t)}{\partial \theta(t)} = \frac{ml \left( \dot{\theta}^2(t) \cos \theta(t) + \sin \ddot{\theta}(t) \right)}{m_c + m}$$

$$\frac{\partial \ddot{x}(t)}{\partial \dot{\theta}(t)} = \frac{2ml \dot{\theta}(t) \sin \theta(t)}{m_c + m}$$

$$\frac{\partial \ddot{x}(t)}{\partial \ddot{\theta}(t)} = \frac{-ml \cos \theta(t)}{m_c + m}$$

$$\frac{\partial \ddot{x}(t)}{\partial F(t)} = \frac{1}{m_c + m}$$



Für 2 Eulerschritte ergibt sich, zur direkten Berechnung:

$$\frac{\partial \theta(t)}{\partial w_{ij}} = h^2 \frac{\frac{-\cos \theta(t-2)}{m_c+m}}{l \left( \frac{3}{4} - \frac{m \cos^2 \theta(t-2)}{m_c+m} \right)} a \frac{\partial y_{out}(t-2)}{\partial w_{ij}} = h^2 \frac{\partial \ddot{\theta}(t-2)}{\partial w_{ij}}$$

$$\frac{\partial x(t)}{\partial w_{ij}} = h^2 \left[ \frac{1}{m_c+m} + \frac{\frac{ml \cos^2 \theta(t-2)}{(m_c+m)^2}}{l \left( \frac{3}{4} - \frac{m \cos^2 \theta(t-2)}{m_c+m} \right)} a \frac{\partial y_{out}(t-2)}{\partial w_{ij}} \right] =$$

$$h^2 \frac{1}{m_c+m} \frac{\partial F(t)}{\partial w_{ij}} - h^2 \frac{ml \cos \theta(t-2)}{m_c+m} \frac{\partial \ddot{\theta}(t-2)}{\partial w_{ij}}$$

Hierbei wurde vorausgesetzt, daß gilt:

$$\frac{\partial \theta(t-2)}{\partial w_{ij}} = 0$$

$$\frac{\partial x(t-2)}{\partial w_{ij}} = 0$$

$$\frac{\partial \dot{\theta}(t-2)}{\partial w_{ij}} = 0$$

$$\frac{\partial \dot{x}(t-2)}{\partial w_{ij}} = 0$$

Es wurden zuerst Versuche gestartet ohne Randbegrenzungen und mit allen 4 Variablen  $x, \dot{x}, \theta, \dot{\theta}$  als Eingabe von C, d. h. es handelte sich hier um eine Markov-Umgebung.

Bei allen Versuchen hatte C 6 oder 7 Hidden-Units, 1 Ausgabeunit (und eine 1-Unit wie immer) bei einer Lernrate von  $\alpha_C = 1.0$ . Der Startwinkel lag beliebig zwischen  $\pm 6.4^\circ$  und die Wagenposition war beliebig zwischen  $\pm 2.4m$ , außerdem wurde die Gewichtsmatrix von C zufällig aus dem Intervall  $[-0.1; +0.1]$  vorbesetzt.

Nach 10 - 50 Versuchen konnte C den Stab einige hundert Zyklen in der Mitte der fiktiven Bahn balancieren. Doch dann bekam eine Richtung das Übergewicht und der Wagen fuhr mit leicht wachsender Geschwindigkeit in diese Richtung, wobei trotz der Wagengeschwindigkeit der Stab balanciert wurde. Bei sehr großer Geschwindigkeit und bei oft einigen tausend Zyklen fiel der Stab dann um. Nach einem solchen Versuch hatte C das Balancieren des Stabes *ohne* hohe Geschwindigkeit in die Richtung wie im vorigen Versuch fast vollkommen verlernt. C brauchte bis zu 10 Versuche, um dies wieder zu lernen, wonach obiger Effekt abermals auftrat. Während der Lernphase erreichte man oft eine Zyklenzahl von 20 000 - 60 000, gefror man jedoch die Gewichtsmatrix ein, so stellte sich heraus, daß C nur diese spezielle Situation, in der der Wagen in eine bestimmte Richtung fuhr, gut konnte. Bei niedrigeren Lernraten ergaben sich ähnliche Effekte oder C lernte nur wenige hundert Zyklen zu balancieren. Die Lernrate war also hoch genug, daß C sich auf die spezielle aktuelle Situation einstellen konnte, obwohl die globale Güte von C abnahm. Verzichtete man auf  $x$  und  $\dot{x}$  als Eingabe, so erreichte man nur eine maximale Balancierzeit von 1000 Zyklen, doch die durchschnittliche Balancierzeit war deutlich besser.

Hieraus ist also ersichtlich, daß Randbegrenzungen vernünftig sind, um den Umgebungsraum (Phasenraum) zu begrenzen. D. h. die Eingaben sind begrenzt, daher gibt es nicht mehr viele Situationen mit hoher Geschwindigkeit in eine Richtung. Ab jetzt sind wieder Ränder bei  $\pm 2.4m$  vorhanden und C bekommt als Nicht-Markov-Eingabe nur  $x$  und  $\theta$  hat aber weiterhin eine Lernrate  $\alpha_C = 1.0$ .

Nach einigen hundert Versuchen (auch bei anderen Lernraten) hatte C nichts gelernt und hatte nur eine durchschnittliche Zyklenzahl von 7 erreicht. In Tabelle 3.9 sieht man hierfür die Ursache, wobei  $i$  die Ausgabeunit und  $j$  die 1-Unit ist.

Man sieht deutlich in den Tabellen 3.9, 3.10, daß  $\frac{\partial \theta(t)}{\partial w_{ij}}$  sich zu langsam ändert. Ist z. B. das Gewicht  $w_{ij}$  dafür verantwortlich, daß der Stab nach

KAPITEL 3. EXPERIMENTE UND UNTERSUCHUNGEN ZUM ALGORITHMUS50

Zyklennr.	$\frac{\partial\theta(t)}{\partial w_{ij}}$	$\frac{\partial\dot{\theta}(t)}{\partial w_{ij}}$	$\theta$ in rad	$F$ in N
7.	-0.04	-1.29	0.025	0.70
8.	-0.05	-1.47	0.025	0.70
9.	-0.07	-1.65	0.024	0.98
10.	-0.08	-1.83	0.024	0.98
11.	-0.10	-2.00	0.023	1.39
12.	-0.12	-2.18	0.023	1.39
13.	-0.14	-2.34	0.022	1.95
14.	-0.17	-2.51	0.021	1.95
15.	-0.19	-2.67	0.019	2.66
16.	-0.22	-2.83	0.018	2.66
17.	-0.25	-2.98	0.016	3.48
18.	-0.28	-3.12	0.013	3.48
19.	-0.31	-3.26	0.010	4.34
20.	-0.34	-3.40	0.006	4.34
21.	-0.37	-3.53	0.002	5.09
22.	-0.41	-3.65	-0.003	5.09
23.	-0.45	-3.76	-0.008	5.53
24.	-0.48	-3.87	-0.015	5.53
25.	-0.52	-3.98	-0.022	5.34
26.	-0.56	-4.08	-0.031	5.34
27.	-0.60	-4.18	-0.040	4.08
28.	-0.64	-4.27	-0.049	4.08
29.	-0.69	-4.35	-0.059	1.23
30.	-0.73	-4.43	-0.070	1.23
31.	-0.78	-4.50	-0.081	-3.70
32.	-0.82	-4.56	-0.091	-3.70
33.	-0.87	-4.57	-0.101	-10.34
34.	-0.91	-4.57	-0.110	-10.34
35.	-0.96	-4.50	-0.117	-16.83
36.	-1.00	-4.42	-0.122	-16.83
37.	-1.05	-4.27	-0.125	-21.26
38.	-1.09	-4.12	-0.125	-21.26
39.	-1.13	-3.92	-0.122	-23.48
40.	-1.17	-3.72	-0.115	-23.48
41.	-1.21	-3.49	-0.106	-24.38
42.	-1.24	-3.25	-0.093	-24.38
43.	-1.27	-3.01	-0.077	-24.72
44.	-1.30	-2.78	-0.057	-24.72
45.	-1.33	-2.54	-0.034	-24.82
46.	-1.36	-2.32	-0.007	-24.82

Tabelle 3.9: Verlauf der partiellen Ableitung der Umgebungsvariablen während der Zeit. Hierbei ist  $i$  die Ausgabeunit und  $j$  die 1-Unit. Es wurden 2 Eulerschritte gemacht, daher 2 mal dieselbe Kraft in der Tabelle.

Zyklennr.	$\frac{\partial\theta(t)}{\partial w_{ij}}$	$\frac{\partial\dot{\theta}(t)}{\partial w_{ij}}$	$\theta$ in rad	$F$ in N
1.	0.0	-0.18	0.029	-2.51
2.	0.0	-0.36	0.029	-2.51
3.	-0.01	-0.54	0.030	-2.57
4.	-0.01	-0.72	0.031	-2.57
5.	-0.02	-0.89	0.033	-2.55
6.	-0.03	-1.07	0.035	-2.55
7.	-0.04	-1.24	0.038	-2.47
8.	-0.05	-1.41	0.041	-2.47
9.	-0.06	-1.58	0.044	-2.28
10.	-0.08	-1.75	0.048	-2.28
11.	-0.10	-1.91	0.052	-1.88
12.	-0.12	-2.08	0.056	-1.88
13.	-0.14	-2.24	0.061	-1.16
14.	-0.16	-2.39	0.066	-1.16
15.	-0.18	-2.55	0.072	0.06
16.	-0.21	-2.70	0.077	0.06
17.	-0.24	-2.85	0.083	1.99
18.	-0.26	-2.99	0.088	1.99
19.	-0.29	-3.12	0.094	4.82
20.	-0.33	-3.24	0.098	4.82
21.	-0.36	-3.35	0.102	8.57
22.	-0.39	-3.45	0.106	8.57
23.	-0.43	-3.52	0.107	12.91
24.	-0.46	-3.58	0.108	12.91
25.	-0.50	-3.59	0.106	17.02
26.	-0.53	-3.60	0.103	17.02
27.	-0.57	-3.57	0.096	20.14
28.	-0.60	-3.53	0.088	20.14
29.	-0.64	-3.46	0.076	22.01
30.	-0.67	-3.39	0.061	22.01
31.	-0.71	-3.31	0.044	22.87
32.	-0.74	-3.22	0.022	22.87
33.	-0.77	-3.13	-0.002	22.96
34.	-0.80	-3.04	-0.029	22.96
35.	-0.84	-2.97	-0.060	21.90
36.	-0.86	-2.89	-0.095	21.90
37.	-0.89	-2.88	-0.132	16.82
38.	-0.92	-2.86	-0.172	16.82
39.	-0.95	-2.90	-0.215	-2.71
40.	-0.98	-2.93	-0.258	-2.71

Tabelle 3.10: Verlauf der partiellen Ableitung der Umgebungsvariablen während der Zeit. Hierbei ist  $i$  die Ausgabeunit und  $j$  die 1-Unit. Es wurden 2 Eulerschritte gemacht, daher 2 mal dieselbe Kraft in der Tabelle.

links ausgelenkt wurde, so wird  $\frac{\partial \theta(t)}{\partial w_{ij}}$  aufsummiert und ist sehr groß. Ist nun C weiter schon in der Lage bei großer Auslenkung nach links eine Kraft anzuwenden, um den Stab wieder in die senkrechte Lage zu bringen, so nimmt aber  $\frac{\partial \theta(t)}{\partial w_{ij}}$  nicht schnell genug ab und das System macht das Gewicht  $w_{ij}$  dafür verantwortlich, daß der Stab zu weit nach rechts ausgelenkt wurde, wobei evtl. genau das Gegenteil der Fall ist. So kann es zu sehr nachteiligen Gewichtsänderungen kommen. Das entstehende Problem ist anders ausgedrückt das, daß das Modell vergangene Ursachen nicht schnell genug vergißt und diese als Erzeuger für die jetzige Situation sieht, was aber nicht der Fall ist.

Setzt man die Variablen  $\frac{\partial \theta(t)}{\partial w_{ij}}, \frac{\partial \dot{\theta}(t)}{\partial w_{ij}}, \frac{\partial x(t)}{\partial w_{ij}}, \frac{\partial \dot{x}(t)}{\partial w_{ij}}, \frac{\partial \ddot{\theta}(t)}{\partial w_{ij}}, \frac{\partial \ddot{x}(t)}{\partial w_{ij}}, \frac{\partial F(t)}{\partial w_{ij}}$  explizit auf 0, so hat damit das Modell alles Vergangene vergessen. Außer zu Beginn jedes Versuchs, wo diese Variablen immer schon auf 0 gesetzt wurden, werden diese nun nach einer bestimmten Anzahl  $v$  von Zyklen auf 0 gesetzt, d. h. das Modell kann höchstens  $v$  Schritte in die Vergangenheit sehen. Ein großer Nachteil ist der, daß das Modell oft nicht sehen kann, was nun tatsächlich für den aktuellen Zustand verantwortlich ist, da evtl. partielle Ableitungen der Modellvariablen nach den Gewichten gerade erst auf 0 gesetzt wurden.

Schritte in die Vergangenheit	$\bar{x}_{max}$ maximaler Durchschnitt	$\bar{x}_{max}$ bei Versuch	$x_{max}$ maximale Zyklenzahl	$x_{max}$ bei Versuch
1	227	41	400	42, 44
2	256	54, 56	700	33
4	189	55	900	23, 57
5	193	58	600	30, 38
8	188	56	500	16, 17
10	163	57	600	25
10	118	52	900	43
12	7	1 - 60	7	1 -60
12	90	125	400	109, 118
15	7	1 - 60	7	1 -60
20	7	1 - 60	7	1 -60

Tabelle 3.11: *Es wurden 60 Versuche pro Zeile durchgeführt, außer bei Zeile "12, 90, 125, ...", wo 125 Schritte durchgeführt wurden und es war bis Schritt 90 wie in 12, 7, ... Bei der Durchschnittsbildung wurden 100 Versuche mit gleichen Anfangsbedingungen wie beim Lernen genommen.*

Tabelle 3.11 zeigt einige Werte bei nur 60 Versuchen. Bei der Durchschnittsbildung wurden 100 Versuche mit gleichen zufälligen Anfangsbedingungen wie während des Lernens durchgeführt.

Die Ergebnisse in Tab. 3.11 sind statistisch nicht sehr aufschlußreich, da aufgrund von Rechenzeitbeschränkungen nur wenig Experimente durchgeführt wurden. In den Tab. 3.12 - 3.14 sind noch Tests zusammengefaßt in denen das Modell höchstens 8 bzw. 5 Schritte bei einer Nicht-Markov-Umgebung in die Vergangenheit schauen konnte.

Test Nummer	zu Beginn			$\bar{x}_{max}$ maximaler Durchschnitt bei	$x_{max}$ maximale Zyklenzahl bei	bei $\bar{x}_{max}$				
	$\bar{x}_{max}$	$x_{max}$	$x_{min}$			$\bar{x}_{max}$	$x_{max}$	$x_{min}$		
1.	31	55	19	559	115	791	571	614	2429	113
2.	38	66	26	360	220	1063	541	346	877	68
3.	20	27	16	359	132	826	219	347	967	67
4.	29	65	18	521	122	1052	589	518	3784	88
5.	34	91	20	473	116	578	127	507	2428	87
6.	37	65	23	390	120	821	139	344	848	54
7.	18	25	15	360	492	1174	695	396	1576	51
8.	35	81	22	501	107	828	167	430	1437	92
9.	33	74	20	398	132	919	322	384	1314	77
10.	33	112	20	29667	318	920	633	35651	211453	80
11.	28	58	18	377	121	974	657	347	1166	65
12.	35	75	20	609	11	861	285	586	1387	80

Tabelle 3.12: *Das Modell konnte höchstens 5 Schritte in die Vergangenheit sehen. Es wurden immer 4 Hidden-Units verwendet, außer bei den letzten beiden Test, wo 7 Hidden-Units verwendet wurden. Bei Testnr. 10 waren 5 Versuche bei der Durchschnittsbildung über 150 000 Zyklen und nur 22 von 100 Versuchen waren unter 10 000 Zyklen.*

Test Nummer	zu Beginn $\bar{x}_{max}$ $x_{max}$ $x_{min}$	$\bar{x}_{max}$ maximaler Durchschnitt bei	$x_{max}$ maximale Zyklenzahl bei	bei $\bar{x}_{max}$ $\bar{x}_{max}$ $x_{max}$ $x_{min}$
1.	29 65 18	324 644	1048 786	269 1047 54
2.	36 65 20	827 55	1100 671	760 4700 15
3.	27 43 20	914 10	1266 507	1025 5019 20
4.	36 123 22	412 223	1146 778	414 1361 49
5.	25 47 18	376 223	1072 597	317 978 49
6.	28 47 20	355 203	1101 162	294 1063 50
7.	24 33 19	895 83	1323 285	788 3197 15
8.	33 77 20	458 33	1261 647	477 1251 274
9.	20 28 16	795 20	1090 432	766 1139 299
10.	39 126 22	286 17	1023 433	360 8039 12
11.	33 54 22	516 14	1086 893	419 1081 78
12.	33 66 20	446 456	1155 547	478 2105 50
13.	31 55 19	986 15	1051 988	782 3913 17
14.	21 30 17	540 16	759 354	490 1575 16
15.	24 35 18	301 513	1593 937	257 824 66
16.	35 70 22	346 994	1297 602	380 1697 65
17.	43 93 26	382 172	932 377	366 2392 58
18.	21 30 16	402 401	1327 506	358 1136 49

Tabelle 3.13: *Das Modell konnte höchstens 8 Schritte in die Vergangenheit sehen. Es wurden immer 4 Hidden-Units verwendet.*



KAPITEL 3. EXPERIMENTE UND UNTERSUCHUNGEN ZUM ALGORITHMUS56

Test Nummer	zu Beginn			$\bar{x}_{max}$ maximaler Durchschnitt bei	$x_{max}$ maximale Zyklenzahl bei	bei $\bar{x}_{max}$				
	$\bar{x}_{max}$	$x_{max}$	$x_{min}$			$\bar{x}_{max}$	$x_{max}$	$x_{min}$		
1.	25	51	18	1119	59	1508	940	1059	8272	15
2.	45	112	23	968	9	917	587	871	2924	247
3.	34	62	23	504	266	916	135	439	1932	62
4.	15	20	13	1505	10	955	500	1223	4561	24
5.	37	109	23	961	21	958	110	738	2403	18
6.	28	68	18	301	514	993	376	256	766	51
7.	18	23	15	362	303	1094	893	372	957	73
8.	23	34	18	606	176	1193	877	577	1690	93
9.	24	35	18	311	227	1027	673	284	1385	53
10.	18	22	16	420	168	997	917	392	1366	55
11.	31	79	18	416	13	1271	279	388	3195	15
12.	25	35	19	1218	16	1090	265	1204	3020	195
13.	47	108	29	153	128	97	91	152	320	42
14.	14	16	12	414	136	1255	397	337	792	57
15.	25	35	19	324	261	272	265	342	1323	68
16.	23	31	18	304	188	1092	548	240	698	58
17.	34	75	22	465	12	1310	416	384	1685	15
18.	17	23	14	2768	14	1056	339	2811	11145	16
19.	26	35	20	298	651	1110	674	244	920	50
20.	23	27	17	338	186	1414	490	298	1039	67
21.	19	24	16	1031	16	1401	863	996	2730	24
22.	21	26	17	908	215	1377	850	721	3343	60
23.	35	95	22	444	229	1101	635	428	1909	62
24.	25	36	19	379	350	1118	720	323	1115	83
25.	32	90	20	376	122	1144	758	311	876	56
26.	32	45	23	672	37	1009	814	621	2159	15
27.	21	30	15	377	620	1121	210	329	1513	69
28.	34	62	23	504	266	916	135	439	1932	50
29.	43	109	27	519	440	1107	318	430	1121	49
30.	14	18	12	2082	201	940	180	1918	7256	103
31.	17	20	14	399	230	1069	939	364	1485	66
32.	19	24	15	496	19	1279	984	597	2247	18
33.	36	107	23	346	267	1149	314	312	1507	48
34.	44	82	30	584	24	1039	141	586	749	362

Tabelle 3.14: Das Modell konnte höchstens 8 Schritte in die Vergangenheit sehen. Es wurden immer 7 Hidden-Units verwendet.

Die Abbildungen 3.13 bis 3.16 zeigen einige Kurven des Lernverlaufs.

Es wurden auch noch Tests gestartet mit kleineren maximalen Winkeln, um die Propagierzeit rückwärts in der Zeit zu verkürzen, was aber auch zu keinem besseren Ergebnis führte. Kleinere Kräfte brachten auch keine Performanceverbesserung.

### 3.3 Die Neugierunit

Zur Verbesserung des Lernvorgangs des Modells in der parallelen Version des Algorithmus für ein neuronales Modell kann eine *‘Neugierunit’* eingeführt werden. Diese Unit soll das Modell in Situationen leiten, die es noch nicht oder nicht gut gelernt hat, d. h. hier sind die Vorhersagen des Modells schlecht.

Die Aktivierung der Neugierunit ist bestimmt durch den Fehler des Modells (Fehlerquadratsumme oder Fehlerbetragssumme). Je größer der Fehler des Modells um so kleiner ist die Aktivierung der Neugierunit und umgekehrt. Ziel ist es nun eine möglichst geringe Aktivierung der Neugierunit zu erreichen, d. h. das Modell macht hier schlechte Voraussagen, was heißt, daß es sich in einer ungelernten oder schlecht gelernten Situation befindet. Der Sollwert der Neugierunit ist ein relativ niedriger Wert und die Abweichung vom Sollwert, nur wenn sie nach oben geht, wird durch das Modell in den Controller propagiert. So wird im Laufe der Zeit ein niedriger Wert erreicht, d. h. Situationen, in denen das Modell schlecht ist. Dieses Zurückpropagieren geschieht mit niedriger Priorität, womit gemeint ist, daß die Lernrate wesentlich kleiner als die anderen Lernraten ist. Dies führt auch dazu, daß die Neugierunit, falls das Modell alles Lernbare kann, sich auf einen hohen Wert einpendelt. Es gibt aber negative Rückkoppelungen mit C, denn das Lernen von C kann beim Minimieren der Schmerzunits entscheidend gestört werden. Deshalb ist es auch schwierig die richtige Lernrate zur Minimierung der Neugierunit zu finden. Hier wurden Werte von  $\frac{1}{10}$  bis  $\frac{1}{100}$  der Lernrate von C verwendet.

Probleme entstehen, falls das Modell nicht alle Situationen gleich gut vorhersagen kann, da bei einigen Umgebungszuständen ein komplexerer Sachverhalt zugrunde liegt, C würde aber versuchen genau diese Situationen zu erreichen.

Die Ergebnisse zeigen aber, daß zumindest der Lernvorgang beim Modell beschleunigt und verbessert werden kann. Abb. 3.17 - 3.19 zeigen den Verlauf des Fehlers mit und ohne Neugierunit beim Lernen des Modells, es wurde

Abbildung 3.13: *Hier ist der Versuch mit Nummer 12 der Tabelle 3.14 dargestellt.*

Abbildung 3.14: *Hier ist der Versuch mit Nummer 4 der Tabelle 3.14 dargestellt.*

KAPITEL 3. EXPERIMENTE UND UNTERSUCHUNGEN ZUM ALGORITHMUS60

Abbildung 3.15: *Hier ist der Versuch mit Nummer 29 der Tabelle 3.14 dargestellt.*

Abbildung 3.16: *Hier ist der Versuch mit Nummer 7 der Tabelle 3.14 dargestellt.*

ein Modell fürs Pole Balancing gelernt. Der exponentielle Abfall des Fehlers rührt daher, daß der durchschnittliche Fehler während eines Versuchs in der Kurve dargestellt wurde, denn es gibt einen sehr großen Fehler am Ende jedes Versuchs (z. B. beim Umfallen des Stabes), da nichtkontinuierliche Schmerzunits verwendet wurden.

### 3.4 Weitere Experimente

C sollte eine Eisenkugel zwischen 2 Magneten halten, indem er auf die Kugel eine Kraft anwendete. Es handelte sich um eine Markov-Umgebung, da C die vollständige Eingabe (d. h. alle Variablen der Umgebung) erhält. In den meisten Fällen lernte C die Kugel in der Mitte zu halten, wobei er die Kugel oft bis kurz vor einen Magneten rollen ließ und sie dann mit einem kräftigen Stoß in die Mitte schleuderte.

Ein sehr einfach scheinender Versuch ist der, daß C nur einen konstanten Wert ausgeben soll, den er auch als Eingabe erhält. Hat nun M nur eine Reinforcementunit, so muß M den Betrag der Differenz von den konstanten Wert und der Ausgabe von C lernen. An dieser Stelle tritt jedoch ein großes Problem zutage, denn *logistische und lineare Units sind nicht in der Lage, eine Betragsfunktion zu simulieren bzw. anzunähern*. M kann diese Betragsfunktion also nicht lernen und somit kann auch C nichts lernen. Nimmt man nun, um die Betragsfunktion zu approximieren, 2 Units, so steht man vor dem Problem, daß die eine Unit an einer Stelle (z. B. 0) plötzlich stehen bleiben muß und die andere Unit aktiv werden muß. Dies gelingt noch am besten mit logistischen Units, obwohl auch sie Probleme mit dem plötzlichen Gefrieren ihrer Aktivierung haben. Hier in diesen speziellen Fall traten aber nun andere Schwierigkeiten auf, falls man 2 logistische Reinforcementunits für die zu berechnende Betragsfunktion benutzte. Sollte C beispielsweise lernen, einen konstanten Wert von 0.7 auszugeben und bekam M in seiner Lernphase als Controllerausgabe eine Gleichverteilung im Intervall  $[0; 1]$  als Eingabe, so konnte die eine Reinforcementunit bis zu 0.7 und die andere nur bis zu 0.3 aktiv werden. Außerdem war die erstere viel öfters aktiv, da es wahrscheinlicher ist einen Wert unter 0.7 zu bekommen. Diese Asymmetrie führte dazu, daß die erste Reinforcementunit immer eine relativ hohe Aktivierung hatte und die 0.7-Grenze schlecht erkannte. Die zweite Reinforcementunit hingegen war während des gesamten Versuchs nur sehr wenig aktiv und überschritt eine Grenze von 0.17 niemals. Beim Lernen von C führte dies nun dazu, daß die eine Reinforcementunit überbewertet

Abbildung 3.17: Fehler der Modells beim Pole-Balancing mit und ohne Neugierunit nach 400 Versuchen.



Abbildung 3.18: Fehler der Modells beim Pole-Balancing mit und ohne Neugierunit nach 1200 Versuchen.

Abbildung 3.19: Fehler der Modells beim Pole-Balancing mit und ohne Neugierunit nach 2000 Versuchen.

### *KAPITEL 3. EXPERIMENTE UND UNTERSUCHUNGEN ZUM ALGORITHMUS66*

wurde und die Ausgabe von C auf 1.0 answoll. Es war also mit dem hier vorgestellten Algorithmus nicht möglich, diese 'einfache' Aufgabe zu lösen.

## Kapitel 4

# Abschließende Bemerkungen

In der parallelen Online-Version des Algorithmus gibt es verschiedene Probleme. So muß das beschriebene Festfahren des Modells bzw. des Controllers verhindert werden, indem man kleine Lernraten und probabilistische Ausgabeneinheiten von C verwendet, welche aber einige Zeit brauchen, um sich zu justieren, da sich die  $\sigma$ -Unit immer wieder anpassen muß. Desweiteren muß in der Online-Version die Lernrate von C mindestens um einen Faktor 5, meist um einen Faktor 10 und in schwierigen Fällen (C braucht exakte Werte von M) um einen Faktor 100 kleiner als die Lernrate von M gewählt werden. Aus diesen Gründen kann die *sequentielle Version oft sehr viel schneller als die parallele* zum Ziel führen.

Die Güte des Algorithmus hängt sehr *entscheidend von der Wahl des Reinforcements* ab, denn es ist wichtig, daß zur Lösung der gestellten Aufgabe einerseits M dies gut vorhersagen und andererseits C es gut und effektiv minimieren kann. Bei den Reinforcementunits sollte man auf Symmetrieeigenschaften achten, denn sonst wird nur eine Art von Schmerz minimiert.

Wie schon erwähnt könnte das *Modell nur die Änderung der Umgebung statt der Umgebung vorhersagen*, wie in [3] vorgeschlagen wird.

Das Pole Balancing ist vermutlich nicht der beste Versuch die Stärken des Algorithmus zu zeigen, da es hier viele, vom Algorithmus unabhängige, Probleme gibt. Das Flip-Flop-Experiment zeigt, daß der Algorithmus in einer Nicht-Markov-Umgebung gut arbeiten kann. Und da in der realen Welt die meisten Problem, die zu lösen sind, Nicht-Markov-Probleme sind, verdient der Algorithmus Beachtung.

# Literaturverzeichnis

- [1] C. W. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems*. PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1986.
- [2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846, 1983.
- [3] J. Jameson. A neurocontroller based on model feedback and the adaptive heuristic critic. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 2, pages 37–43, 1990.
- [4] M. I. Jordan. Supervised learning and systems with excess degrees of freedom. Technical Report COINS TR 88-27, Massachusetts Institute of Technology, 1988.
- [5] M. I. Jordan and R. A. Jacobs. Learning to control an unstable system with forward modeling. In *Proc. of the 1990 Connectionist Models Summer School, in press*. San Mateo, CA: Morgan Kaufmann, 1990.
- [6] Y. LeCun. Une procédure d'apprentissage pour réseau à seuil asymétrique. *Proceedings of Cognitiva 85, Paris*, pages 599–604, 1985.
- [7] M. C. Mozer. On the interaction of selective attention and lexical knowledge: A connectionist account of neglect dyslexia. Technical Report CU-CS-441-89, University of Colorado at Boulder, 1989.
- [8] P. W. Munro. A dual back-propagation scheme for scalar reinforcement learning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA*, pages 165–176, 1987.

- [9] Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *IEEE/INNS International Joint Conference on Neural Networks, Washington, D.C.*, volume 1, pages 357–364, 1989.
- [10] D. B. Parker. Learning-logic. Technical Report TR-47, Center for Comp. Research in Economics and Management Sci., MIT, 1985.
- [11] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. Technical report, Dept. of Comp. Sci., Carnegie-Mellon Univ., Pittsburgh, 1988.
- [12] A. J. Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Trinity Hall and Cambridge University Engineering Department, 1989.
- [13] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.
- [14] T. Robinson and F. Fallside. Dynamic reinforcement driven error propagation networks with application to game playing. In *Proceedings of the 11th Conference of the Cognitive Science Society, Ann Arbor*, pages 836–843, 1989.
- [15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.
- [16] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM journal on Research and Development*, 3:210–229, 1959.
- [17] J. Schmidhuber. Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report FKI-126-90 (revised), Institut für Informatik, Technische Universität München, November 1990. (Revised and extended version of an earlier report from February.).
- [18] J. Schmidhuber. Networks adjusting networks. In J. Kindermann and A. Linden, editors, *Proceedings of ‘Distributed Adaptive Neural Information Processing’, St. Augustin, 24.-25.5. 1989*, pages 197–208. Oldenbourg, 1990. In November 1990 a revised and extended version appeared.

red as FKI-Report FKI-125-90 (revised) at the Institut für Informatik, Technische Universität München.

- [19] J. Schmidhuber. Recurrent networks adjusted by adaptive critics. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, Washington, D. C.*, volume 1, pages 719–722, 1990.
- [20] J. Schmidhuber and R. Huber. Learning to generate focus trajectories for attentive vision. Technical Report FKI-128-90, Institut für Informatik, Technische Universität München, 1990.
- [21] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.
- [22] P. J. Werbos. Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, 17, 1987.
- [23] R. J. Williams. On the use of backpropagation in associative reinforcement learning. In *IEEE International Conference on Neural Networks, San Diego*, volume 2, pages 263–270, 1988.
- [24] R. J. Williams. Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Comp. Sci., Northeastern University, Boston, MA, 1988.
- [25] R. J. Williams and D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.