DIPLOMARBEIT
IM FACH INFORMATIK

Untersuchungen zu dynamischen neuronalen Netzen

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

Aufgabensteller: Professor Dr. W. Brauer

Betreuer: Dr. Jürgen Schmidhuber

15 Juni 1991

Ich versichere, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe.

Datum Josef Hochreiter

Inhalt

Einleitung

Seit dem grundlegenden Artikel von Williams, Hinton und Rumelhart [RHW86] ist Backpropagation (BP) als Lernmethode für neuronale Netze mit und ohne Rückkopplung sehr populär geworden. Im Gegensatz zu vielen anderen Lernmethoden für neuronale Netze berücksichtigt BP die Netzstruktur und verbessert das Netz aufgrund dieser Kenntnis.

Soll eine weit in der Vergangenheit liegende Netzeingabe Einfluß auf die jetzt gewünschte Ausgabe des Netzes haben, so hat in der Regel bei einem zufällig ausgewählten Netz diese zurückliegende Eingabe sehr geringen Einfluß auf den jetzigen Netzzustand. Deshalb wird auch von BP-Lernalgorithmen in der Praxis nicht erkannt, daß diese Eingabe für die gewünschte Ausgabe mitverantwortlich zu machen ist. Mit BP-Lernalgorithmen ist es also sehr schwer einem Netz beizubringen, sich Eingaben zu 'merken', bis diese zur Erzeugung einer später gewünschten Ausgabe gebraucht werden. Weiterhin nehmen die üblichen BP-Lernalgorithmen für rekurrente Netze sehr viel Rechenzeit in Anspruch.

In vielen Fällen braucht man aber eine zurückliegende Eingabe, so beispielsweise, wenn ein Netz, wie bei Mozer [Moz90], lernen soll zu komponieren, da sich dort Notenfolgen wiederholen und spätere Tonhöhen von den vorhergehenden Tonhöhen bestimmt sind. Steuert ein Netz ein Fahrzeug in einem Labyrinth, und erhält das Netz die Fehlerinformation erst, falls das Fahrzeug in einer Sackgasse ist, so sind hier auch zurückliegende Entscheidungen über den eingeschlagenen Weg von Bedeutung. Soll ein von einem neuronalen Netz gesteuerter Roboter eine Aufgabe verrichten, so sind evtl. vorbereitende Tätigkeiten notwendig, deren Ausführung sich das System merken muß.

In dieser Arbeit wird untersucht, wie man an das Problem der langen Lernzeit bei Netzeingaben, welche später noch Einfluß auf die gewünschte Ausgabe haben sollen, herangehen kann. Dies kann geschehen durch die Netzarchitektur oder unter Nutzung der Struktur der Eingabesequenzen. In Kapitel 4 wird ein Netz so aufgebaut, daß weit zurückliegende Eingaben besser berücksichtigt werden, als bei der herkömmlichen Netzarchitektur. Es werden hier 'Speicherknoten' eingführt, welche Information über einen beliebig langen Zeitraum tragen können. Die Verkürzung der Eingabesequenzen, unter Erhaltung aller relevanten Information, wird in Kapitel 3 untersucht. Bei einer verkürzten Eingabesequenz muß man innerhalb dieser nicht so weit in die Vergangenheit sehen, um die relevanten Eingaben zu erkennen. In Kapitel 1 werden die verwendeten BP-Lernalgorithmen vorgestellt, welche dann in Kapitel 2 analysiert werden, um die Ursache der langen Lernzeit zum Erlernen des Speicherns vergangener Eingaben zu ermitteln. Zu den Lernalgorithmen ist zu sagen, daß in manchen Fällen diese etwas modifiziert wurden, um Rechenzeit einzusparen. Das bei den Methoden von Kapitel 3 und 4 auftretende Problem der Ressourcenbeschaffung wird in Kapitel 5 behandelt.

Die in der Arbeit beschriebenen Versuche wurden auf Sparc-Stations von SUN durchgeführt. Aufgrund von Rechenzeitbeschränkungen konnten algorithmenvergleichende Versuche nicht im

gewünschten Umfang durchgeführt werden. Es gab Testläufe, welche bis zu einer Woche auf diesen Rechnern liefen, wobei aber auch andere Prozesse mit höherer Priorität auf diesen Maschinen vorhanden waren.

Die Definitionen und Notationen in dieser Arbeit sind keine bei Untersuchungen über neuronale Netze üblicherweise gebrauchten Bezeichnungen, sondern werden nur für die hier vorliegende Arbeit eingeführt. Der Grund ist, es gibt bisher keine einheitlichen, grundlegende Definitionen für neuronale Netze, auf welche sich andere Autoren berufen hätten. Deshalb ist nicht gewährleistet, daß in den Definitionen und Notationen nicht Unstimmigkeiten mit anderen Arbeiten auftreten.

Es wurden nicht alle Aussagen mathematisch exakt bewiesen, da die Arbeit nicht den Anspruch einer mathematischen Analyse von neuronalen Netzen erhebt, außerdem ist es sehr schwierig einfache mathematische Formalismen für neuronale Netze zu finden. Die Arbeit will vielmehr Ideen und Ansätze beschreiben und testen, um oben beschriebenes Problem der langen Lernzeit bei wichtigen in der Vergangenheit liegenden Eingaben besser in den Griff zu bekommen.

Außer den hier beschriebenen Methoden zum Lernen in nichtstatischen Umgebungen gibt es noch den Ansatz des 'Adaptiven Kritikers', wie in [Sch90a] und [Sch90c] beschrieben. Der Ansatz der 'fast weights' von Schmidhuber [Sch91b] führt zu einer Speicherfunktion, allerdings mit einem völlig anderen Ansatz als in Kapitel 4, wo auch ein Speicher konstruiert wird.

Kapitel 1

Die verwendeten BP-Lernalgorithmen

1.1 Notationen und Definitionen

Ein **diskretes neuronales Netz** N ist ein Tupel (W, f), wobei W eine $(m+n, m+n)$-Matrix, die **Gewichtsmatrix** von N, ist. Sei $w_{ij} := W(i,j) \in \mathrm{R} \cup \xi$, R ist die Menge der reellen Zahlen und $\xi \notin \mathrm{R}$ wird gebraucht, falls an einer bestimmten Stelle kein Gewicht existiert, dies wird später rigoros eingeführt. Man nennt $f : \mathrm{R}^n \to \mathrm{R}^n$ die **Auswertfunktion** oder **Aktivierungsfunktion** von N.

Sei

$$M = \{E_j \mid \ 1 \leq j \leq r \text{ und } E_j : \mathrm{N}_p \to \mathrm{R}^m\}$$

die Menge der **Eingabesequenzen** bzw. **Trainingssequenzen** der Länge p mit $\mid M \mid = r$. $\mathrm{N}_p = \{1, 2, 3, \ldots, p\}$ ist eine Teilmenge der natürlichen Zahlen N. Eine Eingabesequenz $E_j \in M$ wird von N in p Schritten bzw. **Zeitschritten** verarbeitet, wie unten noch genauer beschrieben wird. Wird gerade die Eingabesequenz E_j verarbeitet, so **liegt** E_j **an** N **an**, und zum Zeitpunkt $1 \leq q \leq p$ wird die Eingabe $x(q) := E_j(q)$ verarbeitet.

Sei $g(q+1) : \mathrm{R}^{qm} \to (\mathrm{R} \cup \zeta)^n, 1 \leq q \leq p$ und $g(1) \in (\mathrm{R} \cup \zeta)^n$ die Funktion für die **gewünschte Netzausgabe** zum Zeitpunkt $q+1$ (allgemeiner ist $g(q+1) : \mathrm{R}^{qm} \to \wp[(\mathrm{R} \cup \zeta)^n], 1 \leq q \leq p$, mit $\wp(A)$ als Potenzmenge von A, wobei es hier eine Menge gewünschter Netzausgaben gibt). $\zeta \notin \mathrm{R}$ wird gebraucht, falls keine gewünschte Netzausgabe zum Zeitpunkt $q+1$ existiert. Zur Vereinfachung der Schreibweise wird $d(q+1) := g(q+1)\,(x(1), x(2), \ldots, x(q))\,, 0 \leq q \leq p$ gesetzt.

Die **Aktivierung des Netzes** N ist $y : \mathrm{N}_p \to \mathrm{R}^n$, wobei $y(q), 2 \leq q \leq p+1$, die Aktivierung des Netzes zum Zeitpunkt q ist mit der **Anfangsaktivierung** $y(1) \in \mathrm{R}^n$, d. h. N ist mit diesem Wert initialisiert.

Das Tupel $(f_i, i+m), 1 \leq i \leq n$, heißt der $(i+m)$-te **Knoten** von N oder die $(i+m)$-te **Einheit** von N, wobei f_i die Aktivierungsfunktion des $(i+m)$-ten Knotens und $y_i(q)$ die Aktivierung des $(i+m)$-ten Knotens zum Zeitpunkt q ist. Als i-ten **Eingabeknoten** oder i-te **Eingabeeinheit** bezeichnet man das Tupel (x, i) für $1 \leq i \leq m$, welches auch oft verkürzt nur als i-ter Knoten oder i-te Einheit bezeichnet wird, falls Verwechslungen ausgeschlossen sind. Ist $d_i(q) \neq \zeta$, so wird der Knoten $(i+m)$ **Ausgabeknoten zum Zeitpunkt** q genannt. Der Knoten $(i+m)$ heißt **Ausgabeknoten**, falls $d_i(q) \neq \zeta$ für alle $1 \leq q \leq p+1$ ist. Der Vektor der

Ausgabeknoten Zeitpunkt q heißt **Netzausgabe** zum Zeitpunkt q. Knoten, die weder Eingabe- noch Ausgabeknoten sind, bezeichnet man als **versteckte Knoten**.

Um die Notation zu vereinfachen, wird der Vektor z wie folgt definiert:

$$z_k(q) = \begin{cases} x_k(q), & 1 \le k \le m \\ y_{k-m}(q), & m+1 \le k \le n+m \end{cases}$$

und hieraus ergibt sich die **Netzeingabe** von N

$$s : \mathrm{N}_p \to \mathrm{R}^n,\ s_k(q) = \sum_{l=1, w_{kl} \neq \xi}^{m+n} w_{kl} z_l(q), \quad 1 \le q \le p,\ 1 \le k \le n,$$

dabei ist $s_k(q)$ die Netzeingabe für die k-te Einheit.

Der Wert w_{kl} wird **Gewicht** oder **Verbindung** vom Knoten l zum Knoten k genannt. Ist $w_{kl} = \xi$, so existiert **keine Verbindung** vom Knoten l zum Knoten k. Es gilt $w_{kl} = \xi$ für $k \le m$, d. h. zu den Eingabeknoten führen keine Verbindungen.

Eine endliche Folge $(w_{j_1 j_0}, w_{j_2 j_1}, \ldots, w_{j_u j_{u-1}})$ heißt **Weg** der Länge u vom Knoten j_0 zum Knoten j_u im Netz N.

Das neuronale Netz N heißt **rekurrent** oder **zyklisch**, falls ein Weg

$$(w_{j_1 j_0}, w_{j_2 j_1}, \ldots, w_{j_u j_{u-1}}), \quad 1 \le u \le n,$$

mit $w_{j_o j_{o-1}} \neq \xi, 1 \le o \le u$ und $j_0 = j_u$ existiert, andernfalls heißt das Netz N **azyklisch**.

Eine Verbindung $w_{jj} \neq \xi$ heißt **rekurrente Verbindung des Knotens** j oder **Verbindung des j-ten Knotens auf sich**. Ein solcher Knoten j heißt **rekurrenter** oder **zyklischer Knoten**.

Ein Knoten i eines azyklischen Netzes gehört zur **Verzögerungsstufe** V_u, falls mindestens ein Weg $(w_{j_1 j_0}, w_{j_2 j_1}, \ldots, w_{j_u j_{u-1}})$ existiert mit $j_0 \le m$, d. h. j_0 ist Eingabeknoten, und $j_u = i$ und $w_{j_o j_{o-1}} \neq \xi, 1 \le o \le u$.

Ein Knoten i eines azyklischen Netzes gehört zur **u-ten Lage**, falls $i \in V_u$ und $\forall_{\tilde{u}>u} i \notin V_{\tilde{u}}$. Die Eingabeknoten werden als die erste Lage betrachtet.

Die Aktivierung des Netzes zum Zeitpunkt q berechnet sich wie folgt:

$$y_k(q) = f_k(s_k(q-1)).$$

Der **Fehler für den k-ten Knoten** von N zum Zeitpunkt q ist:

$$e_k(q) = \begin{cases} d_k(q) - y_k(q), & d_k(q) \neq \zeta \\ 0, & d_k(q) = \zeta \end{cases}.$$

Der **Netzwerkfehler** zum Zeitpunkt τ ist die Fehlerquadratsumme (es könnte auch eine andere Norm verwendet werden)

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

Nun ergibt sich der **Gesamtfehler** zwischen Startzeitpunkt 1 und Zeitpunkt $q \le p+1$ zu

$$E_{total}(1, q) = \sum_{\tau=1}^{q} E(\tau) = E_{total}(1, q-1) + E(q).$$

Die **Lernaufgabe** ist das 3-Tupel (N, M, d), für die eine endliche Folge N_i von neuronalen Netzen gefunden werden soll, so daß die Summe des Gesamtfehlers der Eingabesequenzen

$$E_g = \sum_{j=0, E_j \in M \text{ liegt an } N_l \text{ an}}^{r} E_{total}(1, p+1),$$

wenn man jede Eingabesequenz aus M genau einmal an das letzte Folgenglied N_l anlegt, minimiert wird. Man versucht die Funktion g durch ein neuronales Netz N_l zu approximieren, indem man die Fehlerquadratsumme minimiert und so eine Folge von neuronalen Netzen N_i, mit $N = N_0$ und N_l als letztem Folgenglied, konstruiert.

Sei $N_i = (W^i, f^i)$, so geht N_{i+1} aus N_i hervor mit $f^{i+1} = f^i$ und $W^{i+1} = W^i + \Delta W^i$, d. h. es werden die Werte der Komponenten von W^i abgeändert, wobei gelten soll $W^i(k,l) = \xi \Leftrightarrow W^{i+1}(k,l) = \xi$.

Die Lernaufgabe ist **gelöst**, falls das globale Minimum gefunden wurde, bzw. wenn der Gesamtfehler eine feste Schranke unterschritten hat.

Ab jetzt werden nur existierende Verbindungen betrachtet, d. h. $w_{kl} \neq \xi$. Würde in einer Summe ein $w_{kl} = \xi$ vorkommen, so wird der Summand, der w_{kl} enthält, weggelassen, und auch Ableitungen $\frac{\partial}{\partial w_{kl}}$ mit $w_{kl} = \xi$ werden vernachlässigt, d. h. sie werden gleich null gesetzt.

Betrachtet man N_i als Funktion mit der Parametermatrix W^i, so kann man die aus der Numerischen Mathematik bekannte Methode des Gradientenabstiegs verwenden, um den Fehler von N_i zu minimieren und so N_{i+1} zu erhalten. Die Lernmethode mit Gradientenabstieg nach Newton ist als **Backpropagation** (BP) bekannt und wurde von Williams, Hinton und Rumelhart [RHW86] beschrieben, weitere Erläuterungen und Modifikationen dieses Lernalgorithmus findet man in [Jor86], [Rob89], [RF87], [MP69], [Roh89], [Ghe89], [Pea89].

Mit der Methode des Newtonschen Gradientenabstiegs ist hier

$$\Delta W^i = -\alpha \sum_{j=0, E_j \in M \text{ liegt an } N_i \text{ an}}^{r} \sum_{\tau=1}^{p+1} \frac{\partial E(\tau)}{\partial W^i}$$

mit einer postiven Konstanten α als **Lernrate**.

Zur Berechnung von ΔW^i müssen $r = \mid M \mid$ Eingabesequenzen an N_i angelegt werden. Diese Zeitspanne der Erzeugung von ΔW^i nennt man **Trainingszeitraum** oder **Trainingsabschnitt**. Bei praktischen Verfahren werden die Eingabesequenzen aus M beliebig gewählt, zum Trainieren von N_i. Hier werden zur Berechnung von ΔW^i mehr als r Eingabesequenzen benötigt, um die Wahrscheinlichkeit zu erhöhen, daß jede Eingabesequenz mindestens einmal angelegt worden ist.

1.2 Der Algorithmus von Robinson und Fallside bzw. Williams und Zipser

Der Gesamtfehler zum Zeitpunkt q während dem Anliegen einer Eingabesequenz ist

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

hieraus ist ersichtlich, daß es genügt, $E(q)$ während jedes Schrittes zu minimieren, um am Ende des Trainingabschnittes die einzelnen Gewichtsänderungen aufzusummieren. Die Variable t soll ab jetzt die Anzahl der Schritte seit Beginn des Lernens für N_0 angeben. Es sei

$$\triangle w_{ij}(t_0 + kp + q) = -\alpha \frac{\partial E(q)}{\partial w_{ij}},$$

wenn t_0 das Ende des letzten Trainingsabschnittes war und nun die $(k+1)$-te Eingabesequenz des laufenden Trainingszeitraums anliegt. Bei dieser Eingabesequenz liegt gerade die q-te Eingabe an. Am Ende des Trainingsabschnittes ergibt sich also folgende Gewichtsänderung, wenn t_1 das Ende des Trainingszeitraumes ist,

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

Der Fehler $e_k(q)$ ist zu jedem Zeitpunkt bekannt, da die Funktion d explizit bekannt ist, deshalb muß nur noch $\partial y_k(q)/\partial w_{ij}$ berechnet werden, um

$$-\frac{\partial E(q)}{\partial w_{ij}} = \sum_{k=m+1}^{m+n} e_k(q) \frac{\partial y_{k-m}(q)}{\partial w_{ij}}$$

zu erhalten. Die Kettenregel liefert nun mit dem Kronecker Delta δ

$$\frac{\partial y_k(q+1)}{\partial w_{ij}} = f_k'(s_k(q)) \left[\sum_{l=1}^{m+n} w_{kl} \frac{\partial z_l(q)}{\partial w_{ij}} + \delta_{ik} z_j(q) \right],$$

dies kann wegen $\frac{\partial z_l(q)}{\partial w_{ij}} = 0$ für $l \leq m$ vereinfacht werden zu

$$\frac{\partial y_k(q+1)}{\partial w_{ij}} = f_k'(s_k(q)) \left[\sum_{l=m+1}^{n+m} w_{kl} \frac{\partial y_{l-m}(q)}{\partial w_{ij}} + \delta_{ik} z_j(q) \right].$$

Der Anfangszustand ist unabhängig von den Gewichten, so daß gilt $\frac{\partial y_k(1)}{\partial w_{ij}} = 0$ für $1 \leq k \leq n, m+1 \leq i \leq m+n, 1 \leq j \leq m+n$. Also kann man die partiellen Ableitungen $\frac{\partial y_k(q)}{\partial w_{ij}}$ inkrementell berechnen und durch die Variablen p_{ij}^k ersetzen, mit $p_{ij}^k(1) = 0$, somit ergibt sich

$$p_{ij}^k(q+1) = f_k'(s_k(q)) \left[\sum_{l=m+1}^{m+n} w_{kl} p_{ij}^{l-m}(q) + \delta_{ik} z_j(q) \right].$$

Mit der aktuellen Gewichtsänderung

$$\triangle w_{ij}(q) = \alpha \sum_{k=m+1}^{m+n} e_k(q) p_{ij}^{k-m}$$

erhält man eine Gesamtgewichtsänderung über den Trainingszeitraum von t_0 bis t_1

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

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

$$f'_k(s_k(q)) = y_k(q+1)[1 - y_k(q+1)].$$

Dieser Algorithmus wurde von Robinson und Fallside [RF87] später auch von Williams und Zipser [WZ88] (siehe auch [Pea88] und [Moz89]) modifiziert, so daß keine Grenzen für den Trainingszeitraum mehr notwendig sind und das Netzwerk kontinuierlich lernen kann ('Real-Time' Backpropagation), somit erhält man zu jedem Zeitschritt ein neues N_i. Erreicht wird dies, indem man die Gewichtsmatrix zu jedem Zeitschritt neu berechnet. Außerdem werden am Anfang einer Eingabesequenz die Variablen p_{ij}^k nicht auf 0 gesetzt und die Aktivierung y wird nicht initialisiert, dadurch wird über die Grenzen der Eingabesequenzen hinweg zurückpropagiert. Der so gewonnene Algorithmus ist kein exakter Gradientenabstieg mehr, aber bei kleiner Lernrate wird dieser gut angenähert und der neue Algorithmus 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.

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 Einheit eine 'teacher-forced' Ausgabeeinheit 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:

$$z_k(q) = \begin{cases} x_k(q) & für \quad k \leq m \\ d_k(q) & für \quad d_k(q) \neq \zeta \\ y_k(q) & für \quad d_k(q) = \zeta \end{cases}$$

$$\frac{\partial z_k(q)}{\partial w_{ij}} = \begin{cases} 0 & für \quad k \leq m \\ 0 & für \quad d_k(q) \neq \zeta \\ \frac{\partial y_k(q)}{\partial w_{ij}} & für \quad d_k(q) = \zeta \end{cases}$$

Nun ergibt sich folgende Berechnungsformel:

$$p_{ij}^k(q+1) = f_k(s_k(q)) \left[\sum_{l=m+1, d_l(q)=\zeta}^{m+n} w_{kl} p_{ij}^{l-m}(q) + \delta_{ik} z_j(q) \right].$$

Dieser Algorithmus und der vorherige Algorithmus sind nahezu identisch, nur daß nun die Gewichtsmatrix bei jedem Zeitschritt durch $\triangle w_{ij}(q)$ neu berechnet wird, die aktuelle Ausgabe $y_k(q)$ einer Einheit durch das Teacher-Signal $d_k(q)$ ersetzt wird und die entsprechenden p_{ij}^k Werte auf 0 gesetzt werden. Durch das Abändern der Gewichtsmatrix bei jedem Zeitschritt wird ständig ein neues N_i erzeugt. Ist $|W^i - W^{i+1}|$ klein, so kann $W^i = W^{i+1}$ gesetzt werden und es kann weiter zurückpropagiert werden.

1.3 'Unfolding in time' für zyklische Netze

Wie Minsky und Papert [MP69] zeigten, existiert zu jedem zyklischen Netz ein äquivalentes azyklisches Netz, wenn man nur enlich viele Eingaben in der Eingabesequenz zuläßt, es wird also höchstens p Schritte zurückpropagiert.

Das zum zyklischen Netz äquivalente azyklische Netz entsteht, indem man ein Netz mit $p+1$ Lagen konstruiert, wobei jede Lage eine Kopie aller Knoten (incl. Eingabeknoten) des zyklischen Netzes ist. Die Knoten der ersten Lage enthalten die Anfangsaktivierung und die q-te Lage für $1 \leq q \leq p$ enthält die Eingabe $x(q)$. Ist j_u die Kopie des Knotens j des zyklischen Netzes im azyklischen Netz in der u-ten Lage und ist i_{u+1} die Kopie des Knotens i des zyklischen Netzes im azyklischen Netz in der $(u+1)$-ten Lage, dann ist $\tilde{w}_{i_{u+1}j_u} = w_{ij}$ mit $\tilde{W}$ als Gewichtsmatrix für das azyklische Netz und W als Gewichtsmatrix für das zyklische Netz. Andere Verbindungen als die gerade beschriebenen existieren nicht (siehe hierzu auch Abb. 1.1).

Man muß also nur die letzten p Aktivierungen des zyklischen Netzes abspeichern, damit man, wenn man p Schritte zurückpropagieren möchte, Lernalgorithmen für azyklische Netze verwenden kann, um zyklische Netze zu trainieren. Hier wird der Algorithmus von Rumelhart, Williams und Hinton [RHW86] verwendet in einer 'Real-Time' Version ähnlich dem oben beschriebenen Algorithmus von Williams und Zipser. Zur Erklärung des Algorithmus siehe auch [Wer88] und [Pea89], wo auch 'Unfolding in time' Algorithmen behandelt werden.

Ab jetzt gibt das Argument q nur an, in welcher Lage man sich befindet und die Indizierung gibt an, um welche Knotenkopie es sich in dieser Lage handelt. Hier ist $w_{ij}(\tau)$ die Verbindung von der Kopie des Knotens j in der Lage τ zu der Kopie des Knotens i in der Lage $\tau+1$, weiter ist $w_{ij}(\tau) = w_{ij}(\tau+1)$ für $1 \leq \tau \leq p-1$ und $s_i(\tau) = \sum_{j=1}^{m+n} w_{ij}(\tau) z_j(\tau)$ für $1 \leq \tau \leq p$. Es ist

$$\frac{\partial E(q)}{\partial w_{ij}(\tau)} = \frac{\partial E(q)}{\partial s_i(\tau)} \frac{\partial s_i(\tau)}{\partial w_{ij}(\tau)},$$

mit $2 \leq q \leq p+1$ und weiter

$$\frac{\partial s_i(\tau)}{\partial w_{ij}(\tau)} = z_j(\tau).$$

Sei $\vartheta_i(\tau) = -\frac{\partial E(q)}{\partial s_i(\tau)}$, der Wert für die Abhängigkeit des Fehlers von der Netzeingabe des Knotens i in der Lage $\tau+1$, so ist

$$\vartheta_i(\tau) = -\frac{\partial E(q)}{\partial z_i(\tau+1)} \frac{\partial z_i(\tau+1)}{\partial s_i(\tau)}.$$

Ist $\tau+1 = q$ und ist i eine Ausgabeeinheit zum Zeitpunkt q, dann folgt

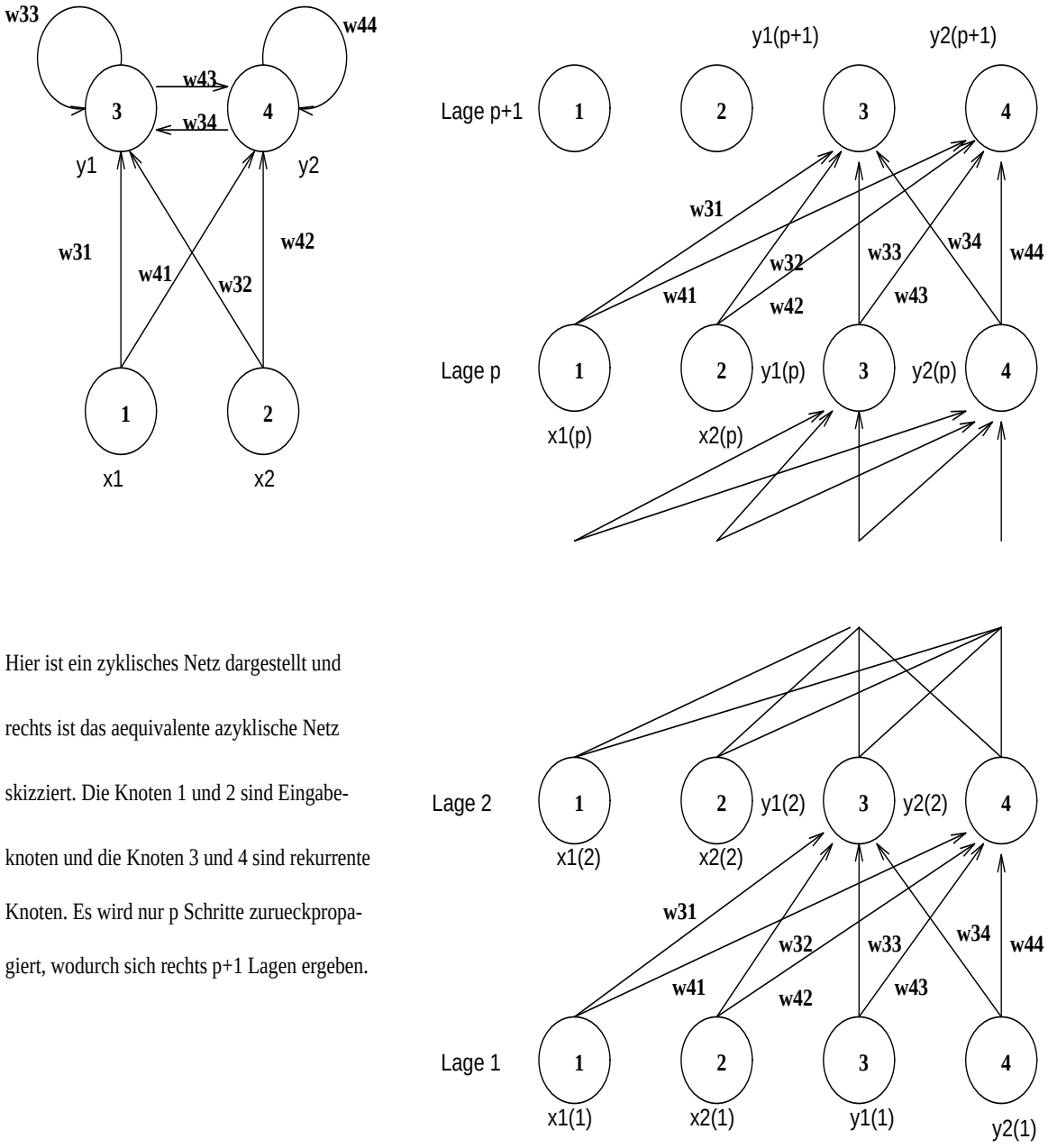
$$\vartheta_i(q-1) = (d_i(q) - y_i(q))\, f_i'(s_i(q-1)) = e_i(q) f_i'(s_i(q-1)).$$

Ist i hingegen keine Ausgabeeinheit zum Zeitpunkt q, so folgt $\frac{\partial E(q)}{\partial z_i(q)} = 0$. Für $\tau+1 < q$ erhält man mit der Kettenregel

$$\frac{\partial E(q)}{\partial z_i(\tau+1)} = \sum_{k, w_{ki}(\tau+1) \text{ existiert}} \frac{\partial E(q)}{\partial s_k(\tau+1)} \frac{\partial s_k(\tau+1)}{\partial z_i(\tau+1)} =$$

$$\sum_{k, w_{ki}(\tau+1) \text{ existiert}} \frac{\partial E(q)}{\partial s_k(\tau+1)} w_{ki}(\tau+1) = - \sum_{k, w_{ki}(\tau+1) \text{ existiert}} \vartheta_k(\tau+1) w_{ki}(\tau+1),$$

was zu



Hier ist ein zyklisches Netz dargestellt und rechts ist das aequivalente azyklische Netz skizziert. Die Knoten 1 und 2 sind Eingabeknoten und die Knoten 3 und 4 sind rekurrente Knoten. Es wird nur p Schritte zurueckpropagiert, wodurch sich rechts p+1 Lagen ergeben.

Abbildung 1.1: *Links ist ein zyklisches Netz dargestellt, welches zu einem azyklichen Netz entfaltet wird, welches rechts dargestellt ist. Es werden p Schritte zurückpropagiert, wodurch man $p+1$ Lagen im azyklischen Netz erhält. Die Knoten 1 und 2 sind Eingabeknoten und die Knoten 3 und 4 sind rekurrent. In der Lage $p+1$ ist die Aktivierung der Eingabeknoten beliebig, da diese Aktivierung nicht verwendet wird.*

$$\vartheta_i(\tau) = f_i'(s_i(\tau)) \sum_{k, w_{ki}(\tau+1) \text{ existiert}} \vartheta_k(\tau+1) w_{ki}(\tau+1) =$$

$$f_i'(s_i(\tau)) \sum_{k, w_{ki}(\tau+1) \text{ existiert}} \vartheta_k(\tau+1) w_{ki}$$

führt, d. h. die Werte ϑ können sukzessiv berechnet werden, wenn man von der q-ten Lage ausgeht.

Die Gewichtsänderung für den Fehler $E(q)$ ist also

$$\Delta w_{ij}(q) = \alpha \sum_{l=1}^{q-1} \vartheta_j(l) z_i(l).$$

Bei der hier verwendeten Version wurde nach Berechnung des Fehlers einer Lage sofort die Gewichtsmatrix abgeändert, was der 'Real-Time' Version von Williams und Zipser des vorherigen Algorithmus entspricht.

Die sukzessive Berechnung der ϑ's beginnend bei $\vartheta_i(q-1) = e_i(q) f_i'(s_i(q-1))$, mit i als Ausgabeeinheit zum Zeitpunkt q, bezeichnet man als **Fehlerrückfluß** durch das Netz. Der Wert $\vartheta_j(q)$ ist der Fehler, der von dem Knoten j zurückfließt.

Betrachtet man in jeder Lage $l, 1 \leq l \leq k$ nur eine Einheit und wird nun ein Fehlerrückfluß durchgeführt ausgehend von $\vartheta_j(k)$ bis zu $\vartheta_i(k-l), 1 \leq l \leq k-1$, so ergeben die zum Fehlerrückfluß benötigten Gewichte einen Weg von j in der $(k+1)$-ten Lage zu i in der $(k-l+1)$-ten Lage. Dies bezeichnet man als einen **Weg, entlang dem der Fehler von j zu i fließt**. Die ϑ's können durch Summation über alle Wege, die der Fehler zurückfließt, berechnet werden.

Beim Algorithmus von Williams und Zipser steckt der Weg, entlang dem der Fehler $e_k(q)$ mit k als Ausgabeknoten zum Zeitpunkt q zurückfließt zum Knoten j, in den Variablen $p_{ij}^k(q+1)$. Der Wert $p_{ij}^k(q+1)$ enthält die Summe aller Wege, die der Fehler vom Knoten k zum Knoten j zurückfließen kann, multipliziert mit der Aktivierung $z_i(q-l-1)$ des Knotens i zum Zeitpunkt $q-l-1$, wenn l die Länge des Weges angibt.

Kapitel 2

Untersuchungen zum Fehlerrückfluß

2.1 Die Methode von Williams und Zipser

Interessante Effekte des Fehlerrückflusses ergeben sich, falls einige der Gewichte des Netzes konstant gehalten werden. Dies geschieht bei der oft verwendeten Methode der Systemidentifikation (siehe hierzu z. B. [Sch90b], [And86], [BSA83], [Sam59], [Wil88], [SH91], [Mun87], [RF89], [Jor88], [NW89], [Wer87]). Hier modelliert ein Teil des gesamten Netzes die Umgebung, um diese differenzierbar zu machen, wobei dieser Teil des Netzes später festgehalten wird, d. h. die Gewichte sind fixiert. Nun kann man durch den Teil des Netzes, welcher die Umgebung repräsentiert, zurückpropagieren zu einem Teil des Netzes, welcher die Umgebung steuert und kontrolliert. Wie in [Hoc90] näher erläutert ist, gibt es Instabilitäten, falls Teile des Netzes festgehalten werden. Hier ist mit Instabilitäten gemeint, daß Gewichte oft so groß wurden, daß es zu einem Überlauf der Float-Zahlen am Rechner kam. Dies war auch bei sehr kleiner Lernrate der Fall, nur daß dann der Überlauf erst später zu beobachten war.

Hier nun einige theoretische Betrachtungen einfachster Netze mit konstanten Gewichten bei Verwendung des Lernalgorithmus von Williams und Zipser. Es werden nur Knoten mit linearer Aktivierungsfunktion betrachtet, mit logistischen Knoten erhält man ein ähnliches Ergebnis.

Beispiel 1 siehe Abb. 2.1, wo die Gewichte $w_{32}, w_{23}, w_{21}, w_{31}$ fest sind und das Gewicht w_{1i} variabel ist:

$$p^2_{1i} = w_{23} {p^3_{1i}}_{\text{old}} + w_{21} {p^1_{1i}}_{\text{old}}$$

$$p^3_{1i} = w_{32} {p^2_{1i}}_{\text{old}} + w_{31} {p^1_{1i}}_{\text{old}}$$

$$p^1_{1i} = {y_i}_{\text{old}}$$

$$p^2_{1i}(t) =$$

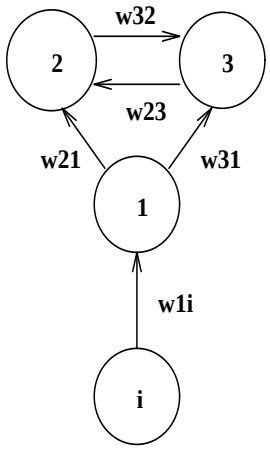
$$w_{23}p^3_{1i}(t-1) + w_{21}p^1_{1i}(t-1) = w_{23}[w_{32}p^2_{1i}(t-2) + w_{31}p^1_{1i}(t-2)] + w_{21}y_i(t-2) =$$

$$w_{32}w_{23}p^2_{1i}(t-2) + w_{23}w_{31}p^1_{1i}(t-2) + w_{21}y_i(t-2) =$$

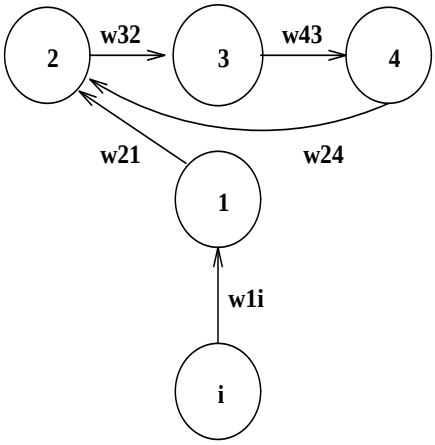
$$w_{32}w_{23}[w_{23}p^3_{1i}(t-3) + w_{21}p^1_{1i}(t-3)] + w_{23}w_{31}y_i(t-3) + w_{21}y_i(t-2) = w_{32}w^2_{23}p^3_{1i}(t-3) +$$

$$w_{32}w_{23}w_{21}p^1_{1i}(t-3) + w_{23}w_{31}y_i(t-3) + w_{21}y_i(t-2) =$$

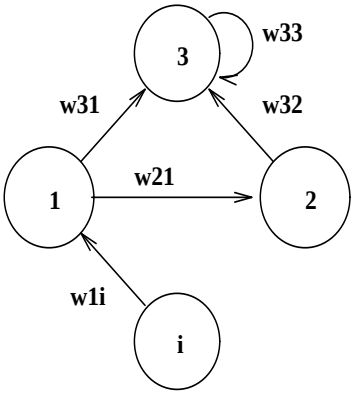
$$w_{32}w^2_{23}[w_{32}p^2_{1i}(t-4) + w_{31}p^1_{1i}(t-4)] + w_{32}w_{23}w_{21}y_i(t-4) + w_{23}w_{31}y_i(t-3) + w_{21}y_i(t-2) =$$



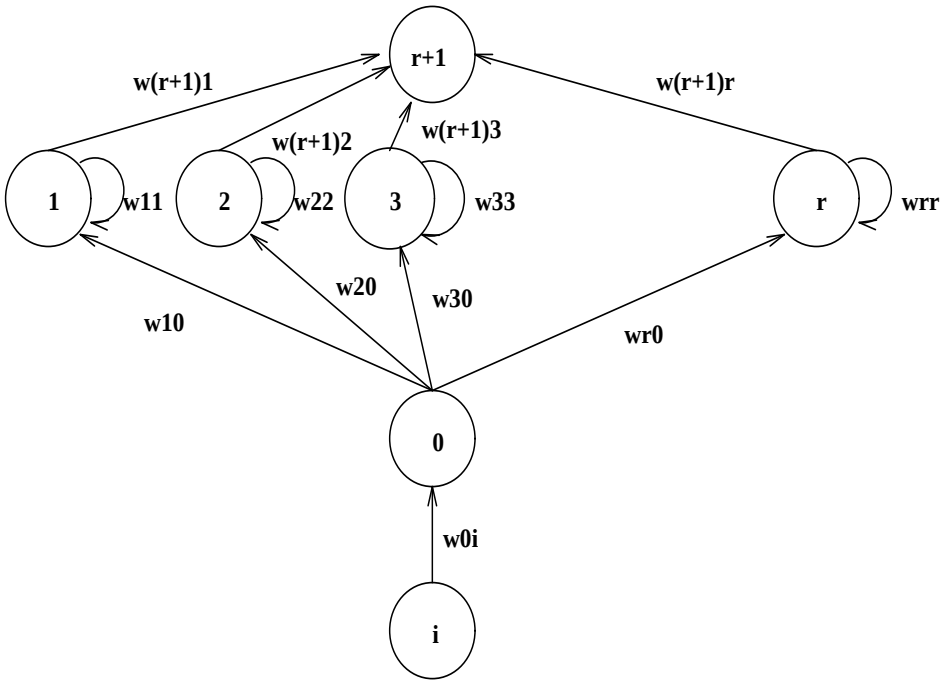
1. theoretisches Beispiel



2. theoretisches Beispiel



4. theoretisches Beispiel



3. theoretisches Beispiel

Abbildung 2.1: *Hier sind einige theoretische Beispielnetze skizziert, an welchen die Probleme gezeigt werden, die bei festgehaltenen Gewichten auftreten. Im 1. Beispiel ist nur das Gewicht w_{1i} variabel, ebenso im 2. Beispiel und im 3. Beispiel ist nur das Gewicht w_{0i} variabel. Im 4. Beispiel sind die beiden Gewichte w_{21} und w_{1i} variabel, hier sieht man auch, wie verschieden stark zurückliegende Fehler Einfluß auf die aktuelle Gewichtsänderung haben.*

$$w_{32}^2 w_{23}^2 p_{1i}^2(t-4) + w_{32} w_{23}^2 w_{31} p_{1i}^1(t-4) + w_{32} w_{23} w_{21} y_i(t-4) + w_{23} w_{31} y_i(t-3) + w_{21} y_i(t-2) =$$

$$w_{32}^2 w_{23}^2 [w_{23} p_{1i}^3(t-5) + w_{21} p_{1i}^1(t-5)] +$$

$$w_{32} w_{23}^2 w_{31} y_i(t-5) + w_{32} w_{23} w_{21} y_i(t-4) + w_{23} w_{31} y_i(t-3) + w_{21} y_i(t-2) =$$

$$w_{32}^2 w_{23}^3 p_{1i}^3(t-5) + w_{32}^2 w_{23}^2 w_{21} p_{1i}^1(t-5) +$$

$$w_{32} w_{23}^2 w_{31} y_i(t-5) + w_{32} w_{23} w_{21} y_i(t-4) + w_{23} w_{31} y_i(t-3) + w_{21} y_i(t-2) =$$

$$\vdots$$

$$= w_{23} (w_{32} w_{23})^{\frac{n-1}{2}} p_{1i}^3(t-n) + w_{21} (w_{32} w_{23})^{\frac{n-1}{2}} p_{1i}^1(t-n) +$$

$$w_{23} w_{31} \sum_{k=1}^{\frac{n-1}{2}} (w_{32} w_{23})^{k-1} y_i(t-2k-1) + w_{21} \sum_{k=1}^{\frac{n-1}{2}} (w_{32} w_{23})^{k-1} y_i(t-2k),$$

für n ungerade und

$$= (w_{32} w_{23})^{\frac{n}{2}} p_{1i}^2(t-n) + w_{23} w_{31} (w_{32} w_{23})^{\frac{n}{2}-1} p_{1i}^1(t-n) +$$

$$w_{23} w_{31} \sum_{k=1}^{\frac{n}{2}-1} (w_{32} w_{23})^{k-1} y_i(t-2k-1) + w_{21} \sum_{k=1}^{\frac{n}{2}} (w_{32} w_{23})^{k-1} y_i(t-2k),$$

für n ungerade. Der Induktionsbeweis ist leicht durchzuführen. Es ergibt sich Instabilität für $|\, w_{32} w_{23} \,| > 1$. Tickt hier der festgehaltene Teil u Mal, so ergibt sich folgende Gleichung, wie man leicht nachrechnet:

$$p_{1i}^2(t) =$$

$$w_{23} (w_{32} w_{23})^{\frac{u-1}{2}} p_{1i}^3(t-u) + p_{1i}^1(t-u) \left(w_{21} (w_{32} w_{23})^{\frac{u-1}{2}} + [w_{21} + w_{23} w_{31}] \sum_{k=0}^{\frac{u-1}{2}-1} (w_{32} w_{23})^k \right) =$$

$$w_{23} (w_{32} w_{23})^{\frac{u-1}{2}} p_{1i}^3(t-u) + p_{1i}^1(t-u) \left(w_{21} (w_{32} w_{23})^{\frac{u-1}{2}} + \frac{(w_{21} + w_{23} w_{31})((w_{32} w_{23})^{\frac{u-1}{2}} - 1)}{w_{32} w_{23} - 1} \right),$$

für u ungerade und

$$= (w_{32} w_{23})^{\frac{u}{2}} p_{1i}^2(t-u) + p_{1i}^1(t-u) \left([w_{21} + w_{23} w_{31}] \sum_{k=0}^{\frac{u}{2}-1} (w_{32} w_{23})^k \right) =$$

$$(w_{32} w_{23})^{\frac{u}{2}} p_{1i}^2(t-u) + p_{1i}^1(t-u) \frac{(w_{21} + w_{23} w_{31})((w_{32} w_{23})^{\frac{u}{2}} - 1)}{w_{32} w_{23} - 1},$$

für u gerade. Hier ergibt sich Instabilität für $|\, w_{32} w_{23} \,| > 1$.

Beispiel 2 siehe Abb. 2.1, wo die Gewichte $w_{32}, w_{43}, w_{24}, w_{21}$ fest sind und das Gewicht w_{1i} variabel ist:

$$p_{1i}^2 = w_{24} {p_{1i}^4}_{\text{old}} + w_{21} {p_{1i}^1}_{\text{old}}$$

$$p_{1i}^3 = w_{32} {p_{1i}^2}_{\text{old}}$$

$$p_{1i}^4 = w_{43} {p_{1i}^3}_{\text{old}}$$

$$p_{1i}^1 = {y_i}_{\text{old}}$$

$$p_{1i}^2(t) =$$

$$= w_{24} p_{1i}^4(t-1) + w_{21} p_{1i}^1(t-1) = w_{43} w_{24} p_{1i}^3(t-2) + w_{21} y_i(t-2) =$$

$$w_{32} w_{43} w_{24} p_{1i}^2(t-3) + w_{21} y_i(t-2) =$$

$$w_{32} w_{43} w_{24}^2 p_{1i}^4(t-4) + w_{32} w_{43} w_{24} w_{21} p_{1i}^1(t-4) + w_{21} y_i(t-2) =$$

$$w_{32} w_{43}^2 w_{24}^2 p_{1i}^3(t-5) + w_{32} w_{43} w_{24} w_{21} y_i(t-5) + w_{21} y_i(t-2) =$$

$$w_{32}^2 w_{43}^2 w_{24}^2 p_{1i}^2(t-6) + w_{32} w_{43} w_{24} w_{21} y_i(t-5) + w_{21} y_i(t-2) =$$

$$\vdots$$

$$= (w_{32} w_{43} w_{24})^l p_{1i}^2(t-n) + w_{21} \sum_{k=0}^{l-1} (w_{32} w_{43} w_{24})^k y_i(t-3k-2),$$

für $n = 3l$ und $l \in \mathrm{N}$ und

$$= w_{24} (w_{32} w_{43} w_{24})^l p_{1i}^4(t-n) + w_{21} (w_{32} w_{43} w_{24})^l p_{1i}^1(t-n) + w_{21} \sum_{k=0}^{l-1} (w_{32} w_{43} w_{24})^k y_i(t-3k-2),$$

für $n = 3l+1$ und $l \in \mathrm{N}$ und zuletzt

$$w_{43} w_{24} (w_{32} w_{43} w_{24})^l p_{1i}^3(t-n) + w_{21} \sum_{k=0}^{l} (w_{32} w_{43} w_{24})^k y_i(t-3k-2),$$

für $n = 3l+2$ und $l \in \mathrm{N}$. Dies ist auch leicht durch Induktion zu beweisen. Instabilitäten ergeben sich hier für $|\, w_{32} w_{43} w_{24} \,| > 1$.

Beispiel 3 siehe Abb. 2.1, hier sind die Gewichte w_{i0}, w_{r+1i}, w_{ii} fest und das Gewicht w_{0i} ist variabel:

$$p_{0i}^{r+1} = \sum_{i=1}^{r} w_{r+1i} {p_{0i}^i}_{\text{old}}$$

$$p_{0i}^{l} = w_{ll} {p_{0i}^{l}}_{\text{old}} + w_{l0} {p_{0i}^{0}}_{\text{old}},\ 1 \le l \le r$$

$$p_{0i}^{0} = {y_i}_{\text{old}}$$

$$p_{0i}^{r+1}(t) =$$

$$\sum_{l=1}^{r} w_{r+1l} p_{0i}^{l}(t-1) = \sum_{l=1}^{r} w_{r+1l} w_{ll} p_{0i}^{l}(t-2) + \sum_{l=1}^{r} w_{r+1l} w_{l0} p_{0i}^{0}(t-2) =$$

$$\sum_{l=1}^{r} w_{r+1l} w_{ll}^{2} p_{0i}^{l}(t-3) + \sum_{l=1}^{r} w_{r+1l} w_{ll} w_{l0} p_{0i}^{0}(t-3) + y_i(t-3) \sum_{l=1}^{r} w_{r+1l} w_{l0} =$$

$$\vdots$$

$$= \sum_{l=1}^{r} w_{r+1l} w_{ll}^{n-1} p_{0i}^{l}(t-n) + \sum_{l=1}^{r} w_{r+1l} w_{ll}^{n-1} w_{l0} p_{0i}^{0}(t-n) +$$

$$\sum_{k=0}^{n-3} y_i(t-k-3) \sum_{l=1}^{r} w_{r+1l} w_{ll}^{k} w_{l0} =$$

$$\sum_{l=1}^{r} w_{r+1l} w_{ll}^{n-1} p_{0i}^{l}(t-n) + \sum_{l=1}^{r} w_{r+1l} w_{ll}^{n-1} w_{l0} p_{0i}^{0}(t-n) +$$

$$\sum_{l=1}^{r} w_{r+1l} w_{l0} \sum_{k=0}^{n-3} w_{ll}^{k} y_i(t-k-3) =$$

$$\sum_{l=1}^{r} (w_{r+1l} w_{ll}^{n-1} p_{0i}^{l}(t-n) + w_{r+1l} w_{ll}^{n-1} w_{l0} p_{0i}^{0}(t-n) +$$

$$w_{r+1l} w_{l0} \sum_{k=0}^{n-3} w_{ll}^{k} y_i(t-k-3)) =$$

$$\sum_{l=1}^{r} w_{r+1l} \left(w_{ll}^{n-1} p_{0i}^{l}(t-n) + w_{ll}^{n-1} w_{l0} p_{0i}^{0}(t-n) + w_{l0} \sum_{k=0}^{n-3} w_{ll}^{k} y_i(t-k-3) \right)$$

Wieder ist der Induktionsbeweis einfach auszuführen. Hier ist die rekurrente Einheit auf r Einheiten aufgeteilt, um evtl. ein starkes rekurrentes Gewicht auf viele schwache rekurrente Gewichte aufzuteilen. Instabil ist dieses System, falls ein $|w_{ii}| > 1$ ist für $i \in \{1, 2, \ldots, r\}$.

Beispiel 4 siehe Abb. 2.1, die Gewichte w_{33}, w_{31}, w_{32} sind fest und die Gewichte w_{21}, w_{1i} sind variabel, es soll $y_3(t) = \tilde{a}(t)$ gelten. Der Rückfluß von früheren Fehlern wird hier ersichtlich:

$$p_{1i}^{3} = w_{33} {p_{1i}^{3}}_{\text{old}} + w_{31} {p_{1i}^{1}}_{\text{old}} + w_{32} {p_{1i}^{2}}_{\text{old}}$$

$$p_{1i}^2 = w_{21} p_{1i}^1{}_{\text{old}}$$

$$p_{1i}^1 = y_i{}_{\text{old}}$$

$$p_{21}^3(t) = w_{33} p_{21}^3{}_{\text{old}} + w_{32} p_{21}^2{}_{\text{old}}$$

$$p_{21}^2 = y_1{}_{\text{old}}$$

$$e(t) = \alpha(y_3(t) - \tilde{a}(t))$$

$$y_1 = w_{1i} y_i{}_{\text{old}}$$

$$w_{21} = w_{21}{}_{\text{old}} + e p_{21}^3{}_{\text{old}}$$

$$w_{1i} = w_{1i}{}_{\text{old}} + e p_{1i}^3{}_{\text{old}}$$

Für den Beweis der weiter unten aufgeführten Formel erhält man folgenden Induktionsanfang:

$$p_{1i}^3(t) =$$

$$w_{33} p_{1i}^3(t-1) + w_{31} p_{1i}^1(t-1) + w_{32} p_{1i}^2(t-1) =$$

$$w_{33}[w_{33} p_{1i}^3(t-2) + w_{31} p_{1i}^1(t-2) + w_{32} p_{1i}^2(t-2)] + w_{31} y_i(t-2) + w_{32} w_{21}(t-2) p_{1i}^1(t-2) =$$

$$w_{33}^2 p_{1i}^3(t-2) + w_{33} w_{31} p_{1i}^1(t-2) + w_{33} w_{32} p_{1i}^2(t-2) + w_{31} y_i(t-2) + w_{32} w_{21}(t-2) p_{1i}^1(t-2) =$$

$$w_{33}^2[w_{33} p_{1i}^3(t-3) + w_{31} p_{1i}^1(t-3) + w_{32} p_{1i}^2(t-3)] + w_{33} w_{31} y_i(t-3) + w_{31} y_i(t-2) +$$

$$w_{33} w_{32} w_{21}(t-3) p_{1i}^1(t-3) + w_{32}[w_{21}(t-3) + e(t-3) p_{21}^3(t-3)] y_i(t-3) =$$

$$w_{33}^3 p_{1i}^3(t-3) + w_{33}^2 w_{31} y_i(t-4) + w_{33}^2 w_{32} p_{1i}^2(t-3) + w_{33} w_{31} y_i(t-3) + w_{31} y_i(t-2) +$$

$$w_{33} w_{32} w_{21}(t-3) p_{1i}^1(t-3) + w_{32} w_{21}(t-3) y_i(t-3) + w_{32} e(t-3) p_{21}^3(t-3) y_i(t-3) =$$

$$w_{33}^3[w_{33} p_{1i}^3(t-4) + w_{31} p_{1i}^1(t-4) + w_{32} p_{1i}^2(t-4)] + w_{33}^2 w_{31} y_i(t-4) + w_{33} w_{31} y_i(t-3) + w_{31} y_i(t-2) +$$

$$w_{33}^2 w_{32} w_{21}(t-4) p_{1i}^1(t-4) + w_{33} w_{32}[w_{21}(t-4) + e(t-4) p_{21}^3(t-4)] y_i(t-4) +$$

$$w_{32}[w_{21}(t-4) + e(t-4) p_{21}^3(t-4)] y_i(t-3) + w_{32} e(t-3)[w_{33} p_{21}^3(t-4) + w_{32} y_1(t-5)] y_i(t-3) =$$

$$w_{33}^4 p_{1i}^3(t-4) + w_{33}^3 w_{32} p_{1i}^2(t-4) + w_{33}^3 w_{31} y_i(t-5) +$$

$$w_{33}^2 w_{31} y_i(t-4) + w_{33} w_{31} y_i(t-3) + w_{31} y_i(t-2) +$$

$$w_{33}^2 w_{32} w_{21}(t-4) p_{1i}^1(t-4) + w_{33} w_{32} w_{21}(t-4) y_i(t-4) + w_{32} w_{21}(t-4) y_i(t-3) +$$

$$w_{33} w_{32} e(t-4) p_{21}^3(t-4) y_i(t-4) + w_{32} e(t-4) p_{21}^3(t-4) y_i(t-3) +$$

$$w_{33} w_{32} e(t-3) p_{21}^3(t-4) y_i(t-3) + w_{32}^2 e(t-3) y_1(t-5) y_i(t-3) =$$

$$w_{33}^4[w_{33} p_{1i}^3(t-5) + w_{31} p_{1i}^1(t-5) + w_{32} p_{1i}^2(t-5)] + w_{33}^3 w_{32} w_{21}(t-5) p_{1i}^1(t-5) +$$

$$w_{33}^3 w_{31} y_i(t-5) + w_{33}^2 w_{31} y_i(t-4) + w_{33} w_{31} y_i(t-3) +$$

$$w_{31} y_i(t-2) + w_{33}^2 w_{32}[w_{21}(t-5) + e(t-5) p_{21}^3(t-5)] p_{1i}^1(t-4) +$$

$$w_{33} w_{32}[w_{21}(t-5) + e(t-5) p_{21}^3(t-5)] y_i(t-4) + w_{32}[w_{21}(t-5) + e(t-5) p_{21}^3(t-5)] y_i(t-3) +$$

$$w_{33} w_{32} e(t-4)[w_{33} p_{21}^3(t-5) + w_{32} p_{21}^2(t-5)] y_i(t-4) +$$

$$w_{32}e(t-4)[w_{33}p_{21}^3(t-5)+w_{32}p_{21}^2(t-5)]y_i(t-3)+$$
$$w_{33}w_{32}e(t-3)[w_{33}p_{21}^3(t-5)+w_{32}p_{21}^2(t-5)]y_i(t-3)+w_{32}^2e(t-3)y_1(t-5)y_i(t-3)=$$
$$w_{33}^5p_{1i}^3(t-5)+w_{33}^4w_{32}p_{1i}^2(t-5)+w_{33}^4w_{31}y_i(t-6)+$$
$$w_{33}^3w_{31}y_i(t-5)+w_{33}^2w_{31}y_i(t-4)+w_{33}w_{31}y_i(t-3)+w_{31}y_i(t-2)+$$
$$w_{33}^3w_{32}w_{21}(t-5)p_{1i}^1(t-5)+w_{33}^2w_{32}w_{21}(t-5)y_i(t-5)+$$
$$w_{33}w_{32}w_{21}(t-5)y_i(t-4)+w_{32}w_{21}(t-5)y_i(t-3)+$$
$$w_{33}^2w_{32}e(t-5)p_{21}^3(t-5)y_i(t-5)+w_{33}w_{32}e(t-5)p_{21}^3(t-5)y_i(t-4)+$$
$$w_{32}e(t-5)p_{21}^3(t-5)y_i(t-3)+w_{33}^2w_{32}e(t-4)p_{21}^3(t-5)y_i(t-4)+$$
$$w_{33}w_{32}e(t-4)p_{21}^3(t-5)y_i(t-3)+w_{33}w_{32}^2e(t-4)p_{21}^2(t-5)y_i(t-4)+w_{32}^2e(t-4)p_{21}^2(t-5)y_i(t-3)+$$
$$w_{33}^2w_{32}e(t-3)p_{21}^3(t-5)y_i(t-3)+w_{33}w_{32}^2e(t-3)p_{21}^2(t-5)y_i(t-3)+w_{32}^2e(t-3)y_1(t-5)y_i(t-3)$$

Hier die Induktionsaussage:

$$p_{1i}^3(t)=$$
$$w_{33}^np_{1i}^3(t-n)+w_{33}^{n-1}w_{31}p_{1i}^1(t-n)+w_{33}^{n-1}w_{32}p_{1i}^2(t-n)+$$
$$w_{31}\sum_{k=2}^{n}w_{33}^{k-2}y_i(t-k)+w_{33}^{n-2}w_{32}w_{21}(t-n)p_{1i}^1(t-n)+$$
$$w_{32}w_{21}(t-n)\sum_{k=3}^{n}w_{33}^{k-3}y_i(t-k)+$$
$$w_{32}p_{21}^3(t-n)w_{33}^{n-3}\sum_{s=3}^{n}e(t-s)\sum_{k=3}^{s}w_{33}^{k-s}y_i(t-k)+$$
$$w_{32}^2p_{21}^2(t-n)w_{33}^{n-4}\sum_{s=3}^{n-1}e(t-s)\sum_{k=3}^{s}$$
$$w_{33}^{k-s}y_i(t-k)+w_{32}^2\sum_{r=5}^{n}w_{33}^{r-5}y_1(t-r)\sum_{s=3}^{r-2}e(t-s)\sum_{k=3}^{s}w_{33}^{k-s}y_i(t-k)$$

Nun zum Induktionsschluß, wobei die Variablen der Induktionsaussage ersetzt wurden:

$$p_{1i}^3(t)=$$
$$w_{33}^n[w_{33}p_{1i}^3(t-(n+1))+w_{31}p_{1i}^1(t-(n+1))+w_{32}p_{1i}^2(t-(n+1))]+w_{31}\sum_{k=2}^{n+1}w_{33}^{k-2}y_i(t-k)+$$
$$w_{33}^{(n+1)-2}w_{32}w_{21}(t-(n+1))p_{1i}^1(t-n)+$$
$$w_{33}^{n-2}w_{32}[w_{21}(t-(n+1))+e(t-(n+1))p_{21}^3(t-(n+1))]y_i(t-(n+1))+$$
$$w_{32}[w_{21}(t-(n+1))+e(t-(n+1))p_{21}^3(t-(n+1))]\sum_{k=3}^{n}w_{33}^{k-3}y_i(t-k)+$$

$$w_{32}[w_{33}p_{21}^3(t-(n+1))+w_{32}p_{21}^2(t-(n+1))]w_{33}^{n-3}\sum_{s=3}^{n}e(t-s)\sum_{k=3}^{s}w_{33}^{k-s}y_i(t-k)+$$

$$w_{32}^2y_1(t-(n+1))w_{33}^{n-4}\sum_{s=3}^{n-1}e(t-s)\sum_{k=3}^{s}w_{33}^{k-s}y_i(t-k)+$$

$$w_{32}^2\sum_{r=5}^{n}w_{33}^{r-5}y_1(t-r)\sum_{s=3}^{r-2}e(t-s)\sum_{k=3}^{s}w_{33}^{k-s}y_i(t-k)=$$

$$w_{33}^{(n+1)}p_{1i}^3(t-(n+1))+w_{33}^nw_{31}p_{1i}^1(t-(n+1))+w_{33}^nw_{32}p_{1i}^2(t-(n+1))+$$

$$w_{31}\sum_{k=2}^{(n+1)}w_{33}^{k-2}y_i(t-k)+w_{33}^{(n+1)-2}w_{32}w_{21}(t-(n+1))p_{1i}^1(t-(n+1))+$$

$$w_{32}w_{21}(t-(n+1))\sum_{k=3}^{(n+1)}w_{33}^{k-3}y_i(t-k)+w_{32}p_{21}^3(t-(n+1))w_{33}^{n-2}\sum_{s=3}^{(n+1)}e(t-s)\sum_{k=3}^{s}$$

$$w_{33}^{k-s}y_i(t-k)+w_{32}^2p_{21}^2(t-(n+1))w_{33}^{n-3}\sum_{s=3}^{(n+1)-1}e(t-s)\sum_{k=3}^{s}w_{33}^{k-s}y_i(t-k)+$$

$$w_{32}^2\sum_{r=5}^{(n+1)}w_{33}^{r-5}y_1(t-r)\sum_{s=3}^{r-2}e(t-s)\sum_{k=3}^{s}w_{33}^{k-s}y_i(t-k).$$

Zur letzten Umformung ist anzumerken, daß mit $s=n+1$ gilt:

$$w_{32}e(t-(n+1))p_{21}^3(t-(n+1))\sum_{k=3}^{n}w_{33}^{k-3}y_i(t-k)=$$

$$w_{32}e(t-(n+1))p_{21}^3(t-(n+1))w_{33}^{n-2}\sum_{k=3}^{n}w_{33}^{k-(n+1)}y_i(t-k)=$$

$$w_{32}e(t-(n+1))p_{21}^3(t-(n+1))w_{33}^{n-2}\sum_{k=3}^{s-1}w_{33}^{k-s}y_i(t-k)$$

Für u Zeitticks des festgehaltenen Teils des Netzes gilt:

$$p_{1i}^3(t)=$$

$$w_{33}^up_{1i}^3(t-u)+[w_{31}p_{1i}^1(t-u)+w_{32}p_{1i}^2(t-u)]\sum_{k=0}^{u-1}w_{33}^k=$$

$$w_{33}^up_{1i}^3(t-u)+\frac{w_2(w_{33}^u-1)}{w_{33}-1}p_{1i}^1(t-u)+\frac{w_{32}(w_{33}^u-1)}{w_{33}-1}p_{1i}^2(t-u)$$

$$p_{21}^3(t)=$$

$$w_{33}^u p_{21}^3(t-u) + \frac{w_{32}(w_{33}^u - 1)}{w_{33} - 1} p_{21}^2(t-u).$$

Hier wurde angenommen, daß $y_1(t)$ und $e(t)$ unabhängig von früheren Zuständen sind, was aber in der Realität nicht der Fall ist.
Es ist

$$\frac{\partial p_{1i}^3(t)}{\partial e(t-l)} =$$

$$w_{32} p_{21}^3(t-l) w_{33}^{l-3} \sum_{k=3}^{l} w_{33}^{k-l} y_i(t-k) =$$

$$w_{32} p_{21}^3(t-l) \sum_{k=3}^{l} w_{33}^{k-3} y_i(t-k)$$

und

$$\frac{\partial p_{1i}^3(t)}{\partial e(t-3)} = w_{32} p_{21}^3(t-3) y_i(t-3)$$

Hieraus sieht man, daß sich unter Umständen ein früherer Fehler aufschaukeln kann, wenn $\mid w_{33} \mid > 1$ ist. Je kleiner die Gewichte sind, entlang denen der Fehler zurückfließt, um so weniger geht der sich im Netz bewegende Fehler in die aktuelle Gewichtsänderung ein. Bei kleinen rekurrenten Gewichten wird ein früherer Fehler leichter 'vergessen'.

2.2 Der 'Unfolding in Time' Algorithmus

Bei der Lernmethode des 'Unfolding in Time' wird nun quantitativ der Fehlerrückfluß untersucht. Soll ein Netz Informationen über große Zeiträume speichern können, um diese Informationen später zur Erzeugung einer gewünschten Ausgabe zu gebrauchen, so muß der Fehler über diesen Zeitraum im Netz umherfließen. Dies kann durch zyklische Wege in einem zyklischen Netz oder durch entsprechend viele Lagen in einem azyklischen Netz erreicht werden.

Betrachtet man den Fehler $E(q)$, so gilt

$$\vartheta_i(q-1) = \begin{cases} 0 & i \text{ ist keine Ausgabeeinheit zum Zeitpunkt } q \\ e_i(q) f_i'(s_i(q-1)) & i \text{ ist Ausgabeeinheit zum Zeitpunkt } q \end{cases}.$$

Es wird nur die Ausgabeeinheit s zum Zeitpunkt q betrachtet und es wird der Fehlerrückfluß analysiert, die der Fehler $e_s(q)$ zur Einheit j in der $(q-u)$-ten Lage zurückfließt, um dann das Gewicht w_{ij} abzuändern. Bei den beim Fehlerrückfluß verwendeten Knoten kann höchstens der Knoten i ein Eingabeknoten sein, da zu den Eingabeknoten keine Verbindungen bestehen und somit dort alle Wege enden.

Es ist mit $l_u = j$ und $l_0 = s$

$$\frac{\partial^2 \Delta w_{ij}(q)}{\partial z_i(q-u-1) \partial \vartheta_s(q)} = \alpha \frac{\partial \vartheta_j(q-u)}{\partial \vartheta_s(q)} =$$

$$\sum_{l_0=m+1}^{m+n} \cdots \sum_{l_u=m+1}^{m+n} \prod_{k=1}^{u} f'_{l_k}(s_{l_k}(q-k)) w_{l_k l_{k-1}},$$

wobei man letzte Umformung induktiv erhält, wegen

$$\frac{\partial \vartheta_v(q-k)}{\partial \vartheta_s(q)} = \begin{cases} w_{sv} & k=1 \\ f'_v(s_v(q-k)) \sum_{l=m+1}^{m+n} \frac{\partial \vartheta_l(q-k+1)}{\partial \vartheta_s(q)} w_{lj} & k>1 \end{cases}.$$

Nimmt man an, daß die Gewichte aus dem Intervall $[-2\gamma, 2\gamma]$ gleichverteilt sind, so ergibt sich ein positiver Erwartungswert γ für $|\, w_{ij}\,|$.

Sei $S := n^{u+1}$ die Anzahl der Summanden in obiger Formel, wobei die Wahrscheinlichkeit, daß ein beliebig ausgewählter Summand positiv ist, $\frac{1}{2}$ ist. Diese Wahrscheinlichkeit ergibt sich auch, falls erheblich mehr postive (negative) Gewichte als negative (positive) vorhanden sind, denn für das Vorzeichen ist nur entscheidend, ob eine gerade oder ungerade Anzahl von negativen Gewichten im Weg vorkommen, entlang dem der Fehler zurückfließt. Somit hat man eine Wahrscheinlichkeit von $\binom{S}{k}\left(\frac{1}{2}\right)^S$, daß k Summanden positiv sind.

Nimmt man noch einen durchschnittlichen Wert β von $f_l(s_l(k))$ für den Knoten l in der Lage k an, so erhält man folgenden Erwartungswert

$$\frac{1}{\alpha} E\left(\left| \frac{\partial^2 \Delta w_{ij}(q)}{\partial \vartheta_s(q) \partial z_i(q-u-1)} \right| \right) =$$

$$(\gamma\beta)^u \left(\frac{1}{2}\right)^S \sum_{k=[\frac{S+2}{2}]}^{S} \binom{S}{k} (2k-S),$$

wobei der Faktor 2 bei k in $2k-S$ entsteht, da man auch negativen Fehlerrückfluß berücksichtigen muß.

Verwendet man für die Aktivierungsfunktionen der einzelnen Knoten die logistische Funktion, so hat man $f'_l(s_l(k-1)) = y_l(k)(1-y_l(k))$. Wie viele Versuche zeigten, bewegt sich die Aktivierung der Knoten bei dieser Aktivierungsfunktion zwischen 0.2 und 0.8, deshalb wurde, wegen dem Mittelwert $\frac{1}{0.6}\int_{0.2}^{0.8} y(1-y)dy = 0.22$, der Wert $\beta = 0.22$ verwendet. Wegen $\int_0^1 y(1-y)dy = \frac{1}{6}$, könnte sich bei ungünstigen Gewichten der Wert β noch verkleinern.

Zur Berechnung von

$$\binom{S}{k}\left(\frac{1}{2}\right)^S = \frac{S!}{k!(S-k)!2^S}$$

wurde für großes S die Stirling-Formel

$$n! = n^{n+\frac{1}{2}} e^{-n+\varepsilon_n}\sqrt{2\pi}, \quad \frac{1}{12n+1} < \varepsilon_n < \frac{1}{12n}$$

verwendet. Somit ist

$$\binom{S}{k}\left(\frac{1}{2}\right)^S = \frac{1}{\sqrt{2\pi}} \exp(\varepsilon_S - \varepsilon_k - \varepsilon_{S-k} +$$

$$S(\ln S - \ln(S-k) - \ln 2) + k(\ln(S-k) - \ln k) + \frac{1}{2}(\ln S - \ln(S-k) - \ln k)),$$

wobei $\ln S = (u+1)\ln n$ und $\frac{1}{12S+1} - \frac{1}{12k} - \frac{1}{12(S-k)} < \varepsilon_S - \varepsilon_k - \varepsilon_{S-k} < \frac{1}{12S} - \frac{1}{12k+1} - \frac{1}{12(S-k)+1} < 0$ ist.

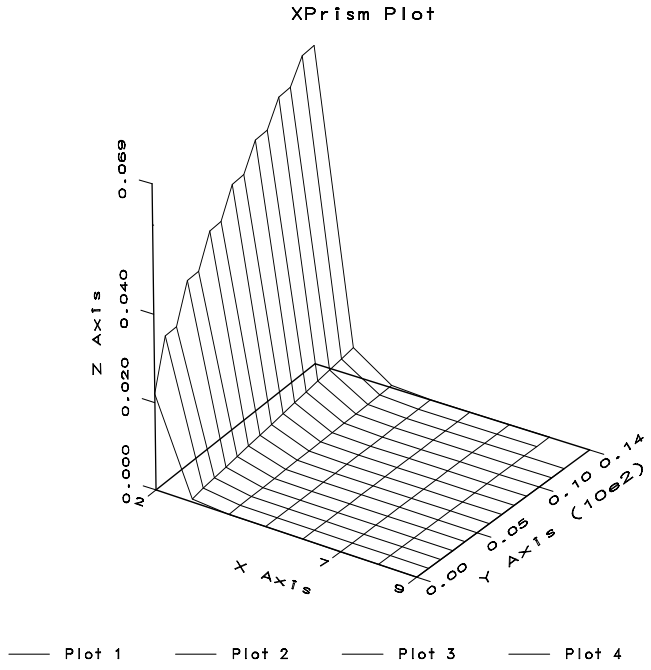
Die Abbildungen 2.2 bis 2.4 zeigen den Wert von $\frac{1}{\alpha} E\left(\left| \frac{\partial^2 \Delta w_{ij}(q)}{\partial \vartheta_s(q) \partial z_i(q-u-1)} \right|\right)$ für verschiedene Werte von γ. Bei der Initialisierung wurde bei den Versuchen $\gamma = 0.1$ gewählt, wonach der Wert für γ in den meisten Fällen ständig, aber sehr langsam wächst. Bei größeren Werten für γ ist die Aktivierung der Knoten oft nahe an 1.0 oder 0, so daß man β dort kleiner wählen sollte, was aber hier nicht berücksichtigt wurde.

2.3 Zusammenfassung der Untersuchungen zum Fehlerrückfluß

Wie die vorangegangenen Analysen zeigen, sind die Gewichte eines neuronalen Netzes nicht alle gleichbewertet, d. h. die Abhängigkeit der Netzausgabe von den einzelen Gewichten w_{ij} ist von dem b-fachen Produkt $P = \prod_{k=1}^{b} f'_{i_k}(s_{i_k}(q-k)) w_{i_k j_k}$ anderer Gewichte $w_{i_k j_k}, 1 \leq k \leq b$, abhängig. Dies führt zu Problemen bei der Einstellung von w_{ij} auf einen idealen Wert durch das Gradientenabstiegsverfahren, da entweder w_{ij} zu stark oder zu schwach in den Gradienten der Fehlerfunktion eingeht. Geht w_{ij} zu stark ein, so ergeben sich Instabilitäten, da dieses Gewicht bei jeder Gewichtsänderung so stark abgeändert wird, daß der ideale Wert kaum gefunden werden kann. Geht andererseits w_{ij} zu schwach in den Gradienten des Fehlers ein, so wird w_{ij} kaum verändert. Da aber P in Abhängigkeit von den Gewichten exponentiell fällt bzw. steigt, ist eine Einstellung von P auf einen Wert, welcher um 1.0 liegt, sehr schwierig und langwierig, da eine sehr kleine Lernrate hierzu benötigt wird. Sind die Gewichte fest, so läßt sich P erst gar nicht einstellen. Bei Lernbeginn werden die Gewichte um 0 initialisiert, da man das Netz nicht auf extreme Gewichtswerte einstellen möchte, welche dann meist wieder abgebaut werden müssen. Die Initialisierung führt nun aber zu einem sehr kleinen Wert für P, was eine lange Lernphase zur Folge hat.

Die Gewichte sind die Parameter der Funktionen, die durch das neuronale Netz repräsentiert werden, doch bei dieser Art von Funktionen hat die Änderung der verschiedenen Parameter eine sehr unterschiedliche Auswirkung auf die Funktionswerte. Wird zum Lösen der Aufgabe jedoch eine Einstellung aller Parameter benötigt, so kann dies zu Instabilitäten beim Lernen oder zu einer extrem langen Lernzeit führen.

Zur Beseitigung des beschriebenen Problems gibt es zwei verschiedene Ansätze. Durch eine Reduzierung der Eingabesequenzen kann ein neuronales Netz gefunden werden, welches weniger Eingaben verarbeiten muß, um die Aufgabe zu lösen. Deshalb kann dieses Netz eine einfachere Struktur besitzen, d. h. es werden zur Aufgabenlösung weniger auf einen idealen Wert eingestellte Parameter benötigt. Diese Methode wird nun im anschließenden Kapitel näher untersucht. Die zweite Möglichkeit dem Problem zu begegnen, ist die Konstruktion eines Netzes, so daß die Parameter in annäherungsweise gleicher Größenordnung in den Gradienten der Fehlerfunktion eingehen. Eine solche Konstruktion wird in Kapitel 4 durchgeführt und analysiert.



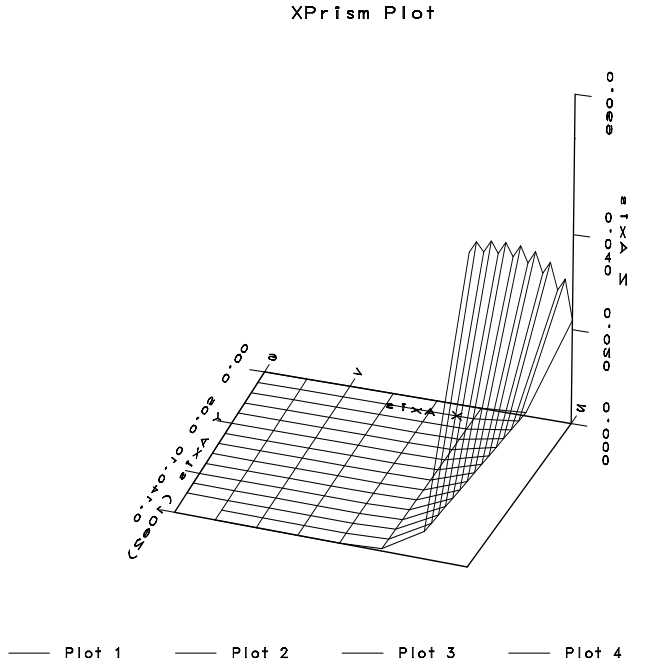
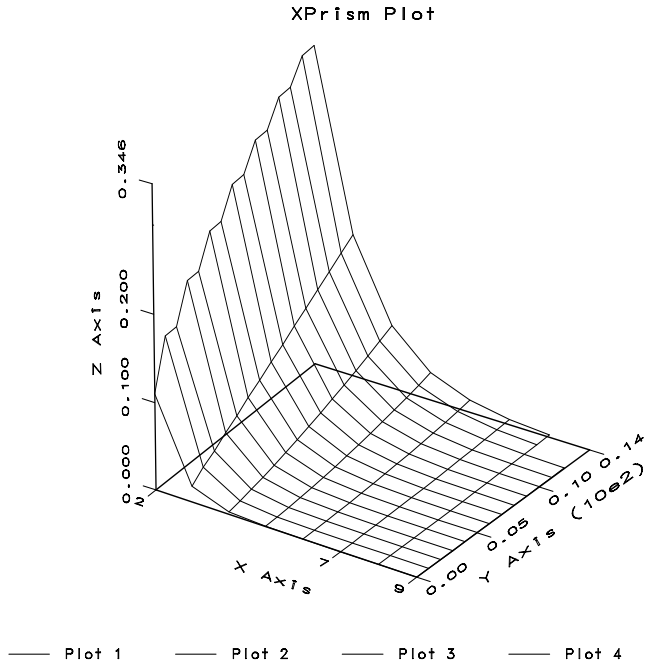


Abbildung 2.2: *Gezeigt ist hier der Wert von* $\frac{1}{\alpha}E\left(\left|\frac{\partial^2 \Delta w_{ij}(q)}{\partial \vartheta_s(q)\partial z_i(q-u-1)}\right|\right)$ *für* $\gamma = 0.1$. *Die y-Achse gibt die Anzahl der Knoten an, welche von 2 - 16 laufen. Die x-Achse gibt die Anzahl der zurückpropagierten Schritte an, welche von 2 - 9 laufen.*



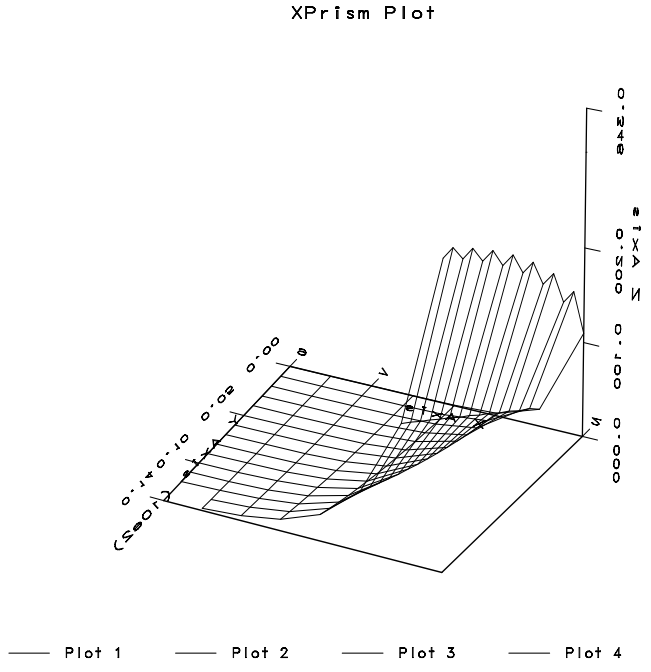
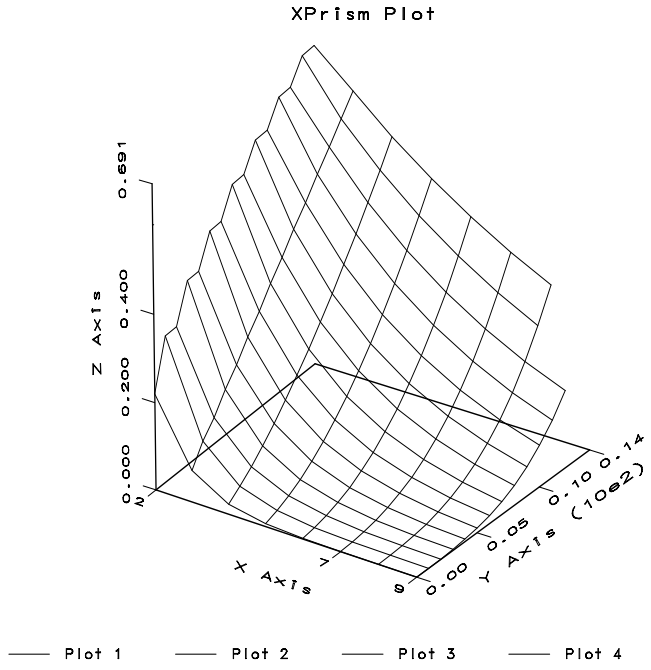


Abbildung 2.3: *Gezeigt ist hier der Wert von* $\frac{1}{\alpha}E\left(\left|\frac{\partial^2\Delta w_{ij}(q)}{\partial\vartheta_s(q)\partial z_i(q-u-1)}\right|\right)$ *für* $\gamma = 0.5$. *Die y-Achse gibt die Anzahl der Knoten an, welche von 2 - 16 laufen. Die x-Achse gibt die Anzahl der zurückpropagierten Schritte an, welche von 2 - 9 laufen.*



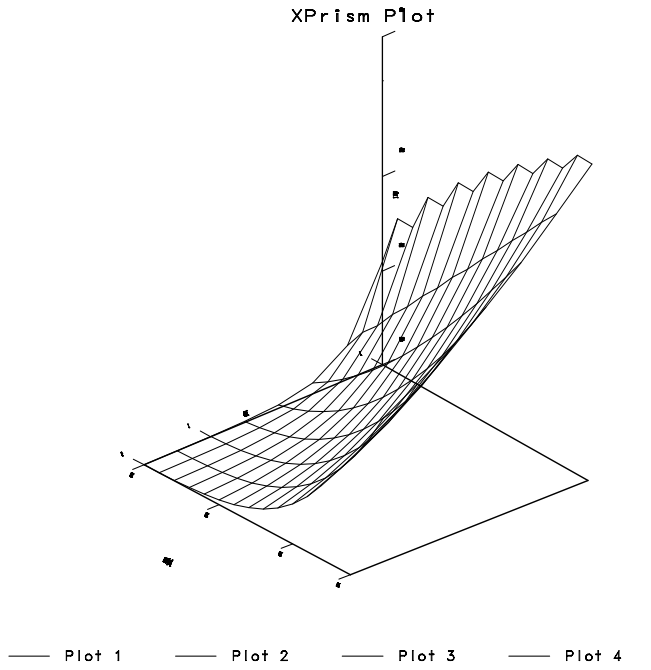


Abbildung 2.4: *Gezeigt ist hier der Wert von* $\frac{1}{\alpha}E\left(\left|\frac{\partial^2\Delta w_{ij}(q)}{\partial\vartheta_s(q)\partial z_i(q-u-1)}\right|\right)$ *für* $\gamma = 1.0$. *Die y-Achse gibt die Anzahl der Knoten an, welche von 2 - 16 laufen. Die x-Achse gibt die Anzahl der zurückpropagierten Schritte an, welche von 2 - 9 laufen.*

Kapitel 3

Reduktion der Eingabesequenz

Um einen Fehlerrückfluß über weite Zeitspannen zu vermeiden, sollte die Eingabesequenz des Netzes auf die wesentliche Information zusammengerafft werden, d. h. es sollen einzig die Eingaben der Sequenz an das Netz angelegt werden, die auch wesentliche Information enthalten. Dies ist nur dann möglich, wenn die Eingabesequenz Strukturen besitzt, d. h. wenn Regelmäßigkeiten in der Eingabesequenz vorhanden sind.

Diese Regelmäßigkeiten können als kausale Abhängigkeit der einzelnen Eingaben betrachtet werden. Hier ist gemeint, daß bestimmte Eingaben durch vorherige Eingaben schon festgelegt sind, diese determinierten Eingaben hängen somit kausal von den vorherigen ab. Die festgelegten Eingaben beinhalten also nur redundante Information und können durch eine geeignete Funktion berechnet werden. Ziel ist es nun diese durch die vorherigen Eingaben festgelegten Eingaben zu eliminieren, um so die Eingabesequenz zu verkürzen.

Soll ein Netz die nächste Eingabe in der Eingabesequenz vorhersagen, so wird dieses Netz, falls es hinreichend viele Netzknoten besitzt, nach genügend langer Lernphase, genau die festgelegten Eingaben vorhersagen können. Hat man ein solches Netz, welches die determinierten Eingaben vorhersagt, so kann man die nicht vorhergesagten Eingaben an ein anderes Netz weiterleiten, das dann die nun reduzierte Eingabesequenz verarbeitet. Somit hat man die Eingabesequenz verkürzt, falls kausale Abhängigkeiten vorhanden waren, und dadurch kann der Fehlerrückfluß, der zur Lösung der Aufgabe nötig ist, verkleinert werden, da nun relevante Eingaben nicht mehr so weit auseinanderliegen.

3.1 Formale Betrachtung der Reduzierung

Nun wird die Reduzierung der Eingabesequenz formal betrachtet, wobei beliebige Näherungsfunktionen statt neuronaler Netze verwendet werden.

Sei ein p-Tupel $E = (e_1, \ldots, e_p) \in \mathrm{R}^{m \times p}$ mit $e_i \in \mathrm{R}^m$ gegeben, welches durch die differenzierbaren Funktionen $v : \mathrm{R}^{(m+l)} \to \mathrm{R}^l$ und $a : \mathrm{R}^{(m+l)} \to \mathrm{R}^m$ und deren Initialisierungswerte $v_0 \in \mathrm{R}^l$ und $a_0 \in \mathrm{R}^m$ eindeutig reduziert werden soll mit der Fehlerschranke ε. Aus dem entstehenden s-Tupel $RE(v, a, v_0, a_0, \varepsilon)((k_1, e_{k_1}), (k_2, e_{k_2}), \ldots, (k_s, e_{k_s})) \in (\mathrm{N}_p \otimes \mathrm{R}^m)^s, 1 \leq s \leq p$, soll E mit der Genauigkeit $\delta(\varepsilon)$ rekonstruierbar sein.

Es wird $RE(v, a, v_0, a_0, \varepsilon)$ induktiv erzeugt, wobei mit $e_0 \in \mathrm{R}^m$ beliebig

$$v_i = v(e_{i-1}, v_{i-1}), \quad a_i = a(e_{i-1}, v_{i-1}) \text{ für } 1 \leq i \leq p,$$

ist. Sei weiter

$$w(e_i, a_i) = \begin{cases} w(e_{i+1}, a_{i+1}) & \text{für } |e_i - a_i| < \varepsilon \text{ und } i < p \\ (i, e_i) & \text{für } |e_i - a_i| \geq \varepsilon \text{ und } i \leq p \\ \eta & \text{für } |e_i - a_i| < \varepsilon \text{ und } i = p \end{cases}$$

Es ist

$$RE(v, a, v_0, a_0, \varepsilon)(1) = w(e_1, a_1)$$

und ist

$$RE(v, a, v_0, a_0, \varepsilon)(t-1) = (k_{t-1}, e_{k_{t-1}}),$$

so gilt

$$RE(v, a, v_0, a_0, \varepsilon)(t) = w(e_{k_{t-1}}, v_{k_{t-1}}), \text{ für } w(e_{k_{t-1}}, v_{k_{t-1}}) \neq \eta \text{ und } 2 \leq t \leq s.$$

Sind die partiellen Ableitungen von v und a in einer Umgebung von $(e_i, x), x \in \mathrm{R}^l$, beschränkt, so kann man durch evtl. Verkleinerung von ε die Sequenz $\tilde{E} \in \mathrm{R}^{m \times p}$ aus $RE(v, a, v_0, a_0, \varepsilon)$ mit der Genauigkeit $\delta(\varepsilon)$ berechnen, so daß $|\tilde{E}(i) - E(i)| < \delta(\varepsilon),\ 1 \leq i \leq p$, ist.

Es ist

$$RE(v, a, v_0, a_0, \varepsilon) : \mathrm{N}_s \to \mathrm{N}_p \otimes \mathrm{R}^m$$

und $E : \mathrm{N}_p \to \mathrm{R}^m$. Sei die Länge einer Eingabesequenz gegeben durch $L(E) = p$ und $L(RE) = s$, genauer definiert man mit $G : \mathrm{N}_v \to \mathrm{K}$ die Länge von G durch $L(G) := v$.

Mit dieser Konstruktion kann man einen Reduktionsoperator definieren:

$$R(v, a, v_0, a_0, \varepsilon)(E) := RE(v, a, v_0, a_0, \varepsilon).$$

Sei weiter eine Menge $ME = \{E_i \mid 1 \leq i \leq r, E_i \in \mathrm{R}^{m \times p}, E_i = (e_{i1}, e_{i2}, \ldots, e_{ip})\}$ gegeben, so soll $R(v, a, v_0, a_0, \varepsilon)(ME) := \{R(v, a, v_0, a_0, \varepsilon)(E_i) \mid 1 \leq i \leq r, E_i \in ME\}$ sein.

Sei weiter

$$LE(R(v, a, v_0, a_0, \varepsilon)(ME)) := \sum_{i=1}^{r} L(R(v, a, v_0, a_0, \varepsilon)(E_i))$$

die Summe der Längen der Folgen aus ME. Ist ε und ME gegeben und wählt man v_0, a_0 beliebig, so kann man

$$K(v, a) := LE(R(v, a, v_0, a_0, \varepsilon)(ME))$$

und $K_i(v, a) := L(R(v, a, v_0, a_0, \varepsilon)(E_i))$ definieren, womit gilt $K(v, a) = \sum_{i=1}^{r} K_i(v, a)$.

Durch die Minimumsbestimmung von $K(v, a)$ erhält man eine Menge von Folgen minimaler Länge, welche durch Anwendung des Operators R auf die Menge von Folgen ME entsteht. Minimiert man nun schrittweise die Funktionen K_i, indem man ein K_i auswählt und die parametrisierten Funktionen v, a durch Gradientenabstieg zu $\tilde{v}, \tilde{a}$ verbessert, so daß $K_i(\tilde{v}, \tilde{a}) \leq K_i(v, a)$ gilt, dann ist zwar nicht sichergestellt, daß das globale Minimum von K gefunden wird, doch man minimiert K.

Es gilt

$$K_i(\tilde{v}, \tilde{a}) < K_i(v, a) \Leftrightarrow |\{a_j \mid |e_{ij} - a_j| < \varepsilon, 1 \leq j \leq p\}| < |\{\tilde{a}_j \mid |e_{ij} - \tilde{a}_j| < \varepsilon, 1 \leq j \leq p\}|,$$

weshalb man für die Verbesserung von v und a als Fehlerfunktion $F_i = \sum_{j=1}^{p} (e_{ij} - a_j)^2$ verwenden kann. Die Funktion a soll also das nächste Folgenglied bei einer Folge $E_i \in ME$ vorhersagen können, indem die vorangegangenen Folgenglieder verarbeitet werden.

3.2 Kausaldetektoren

Ein neuronales Netz N_1 mit l rekurrenten, versteckten Knoten und m Ausgabeknoten kann die Funktionen v und a repräsentieren. Die Fehlerfunktion ist oben gegeben, weshalb man das Netz N_1 durch BP trainieren kann. Dann kann $R(v, a, v_0, a_0, \varepsilon)$ kürzer geschrieben werden als $R(N_1, N_1^0, \varepsilon)$, wobei N_1^0 die Aktivierung der Knoten von N_1 zu Beginn einer Sequenz ist. Bei azyklischen Netzen ist N_1^0 die Aktivierung der Knoten in der 1. Lage.

Die Eingabe e_i und die eindeutige Zeitrepräsentation von i werden zu einem Netz N_2 weitergeleitet, falls e_i nicht durch N_1 mit einer Fehlerschranke ε vorhergesagt worden ist. Diese Idee stammt von Jürgen Schmidhuber und wird genauer in [Sch91a] beschrieben, eine kurze Erläuterung findet man auch in [Sch91c].

Das Netz N_2 besitzt dieselben Daten, welche in der ursprünglichen Eingabesequenz enthalten sind, falls das Netz N_2 das Netz N_1 simulieren kann. Eine vollständige Simulation von N_1 durch N_2 ist aber in den meisten Fällen zur Lösung der gestellten Aufgabe nicht notwendig, die Simulation dient lediglich zur Erzeugung der exakten ursprünglichen Eingabesequenz. Da in der verkürzten Eingabesequenz schon die gesamte Information enthalten ist, ist eine Simulation zur Aufgabenlösung meist nicht notwendig.

Es ist nun möglich, daß die Regelmäßigkeiten in der Eingabe verschieden leicht gelernt werden können. In diesem Fall kann man die Eingabe stufenweise verkürzen, d. h. das erste Netz N_1, welches die Eingaben vorhersagen soll, verkürzt die Eingabesequenz und leitet diese reduzierte Eingabesequenz weiter an ein zweites Netz N_2, welches die verkleinerte Eingabesequenz incl. Zeitrepräsentation weiter zusammenrafft. Dies geschieht, indem N_2 versucht, die nächste Eingabe in der verkürzten Eingabesequenz vorherzusagen. Allgemein werden nichtvorhergesagte Eingaben und zugehörige Zeitrepräsentation einer Stufe an die nächste Stufe weitergeleitet und dort verarbeitet, indem man auf der nächsten Stufe die Eingabesequenz weiter verkürzt oder versucht, die gestellte Aufgabe zu lösen, ohne nun so weit in die Vergangenheit propagieren zu müssen.

Hier nun ein Beispiel für die Notwendigkeit der Zeitrepräsentation. Sei

$$E_1 = (a, b, c, 0, e_{15}, \ldots, e_{1p}),$$

$$E_2 = (a, b, c, 0.5, e_{25}, \ldots, e_{2p}) \text{ und}$$

$$E_3 = (a, b, c, 1.0, e_{35}, \ldots, e_{3p}),$$

somit nimmt e_{i4} mit der Wahrscheinlichkeit $\frac{1}{3}$ die Werte 0, 0.5 und 1.0 an, ohne daß diese Eingabe vorhersagbar ist. Ein Netz wird jedoch lernen, den Erwartungswert $0.5 = \frac{1}{3} \cdot 0 + \frac{1}{3} \cdot 0.5 + \frac{1}{3} \cdot 1.0$ dieser Eingabe vorherzusagen, da so der durchschnittliche Fehler bei der Vorhersage dieser Eingabe minimal ist. In einem Drittel der Fälle würde das Netz die richtige Eingabe vorhersagen, ohne daß diese Eingabe vorhersagbar wäre. Die Eingabesequenz der nächsten Stufe würde aber nun weniger Information besitzen, als die ursprüngliche Eingabesequenz, da man in der verkürzten Sequenz nicht erkennt, wo eine Eingabe weggefallen ist und so wären bestimmte Aufgaben in der höheren Stufe nicht lösbar.

Dies Problem kann behoben werden, indem man statt der Zeitrepräsentation ein Vertrauensnetzwerk einführt, welches die Wahrscheinlichkeit für die korrekte Vorhersage der nächsten Eingabe angibt. Das Vertrauensnetzwerk wird auf eine Ausgabe von 1.0 trainiert, falls das Vorhersagenetzwerk die nächste Eingabe korrekt vorhergesagt hat, und auf 0 andernfalls. Das Vertrauensnetzwerk wird nun lernen die relative Häufigkeit der richtig vorhergesagten Eingaben

auszugeben und dies ist im Grenzfall (Gesetz der großen Zahlen) die Wahrscheinlichkeit für die richtig vorhergesagte Eingabe. Im vorher aufgeführten Beispiel würde das Vertrauensnetzwerk lernen, $\frac{1}{3}$ auszugeben, bei der 4. Eingabe in einer Sequenz.

Mit einem zusätzlichen Vertrauensnetzwerk wird nun die Eingabe nur zur nächsten Stufe gereicht, wenn die Wahrscheinlichkeit für die korrekte Eingabevorhersage unterhalb einer bestimmte Schwelle liegt. Somit werden zufällig richtig vorhergesagte Eingaben auch zur nächsten Stufe geleitet. Das Vertrauensnetzwerk verhindert also, daß zufällig richtig vorhergesagte Eingaben zu Störungen in der nächsten Stufe führen, da nicht erkennbar ist, ob in der Eingabesequenz einzelne Eingaben durch Zufall nicht zur höheren Stufe geleitet wurden.

3.2.1 Test mit 10 Schritten in die Vergangenheit

Nachfolgend wird ein Beispiel zum Testen von Netzen beschrieben, welche weit in der Vergangenheit zurückliegende Eingaben speichern müssen, um eine gewünschte Ausgaben zu liefern.

Sei $G_1^{p-1} = (a, b_1, b_2, \ldots, b_{p-1})$ und $G_2^{p-1} = (x, b_1, b_2, \ldots, b_{p-1})$, wobei $\{G_1^{p-1}, G_2^{p-1}\}$ eine Trainingssequenzmenge für ein neuronales Netz ist. Zu jedem Zeitpunkt liegt genau eine Eingabe von genau einer dieser Sequenzen an. Liegt zum Zeitpunkt t seit Lernbeginn b_{p-1} als Eingabe an, so muß das Netz nach Verarbeitung von b_{p-1} entscheiden, ob G_1^{p-1} oder G_2^{p-1} als Eingabesequenz verwendet wurde. Diese Entscheidung kann nur durch die Speicherung der Eingaben a bzw. x erfolgen.

Es wurden Versuche durchgeführt mit $p = 11$, wobei G_1^{10} oder G_2^{10} zufällig gewählt wurde und N_1 diese als Eingabesequenz erhielt. N_1 hatte 13 Eingabeknoten, je einen für die Eingabe a, x, b_i mit $1 \leq i \leq 10$ und einen Eins-Knoten, der eine konstante Aktivierung von 1.0 hatte. Ein Eingabeknoten (außer dem Eins-Knoten) ist genau dann aktiv (d. h. die Aktivierung ist 1.0), falls die entsprechende Eingabe anliegt, ansonsten hat der Eingabeknoten die Aktivierung 0. N_1 hatte keinen versteckten Knoten und 12 Ausgabeeinheiten, je einen entsprechend der Eingabe außer dem Eins-Knoten, um die nächte Eingabe vorherzusagen. Wie bei allen Versuchen wurde die Gewichtsmatrix mit Werten aus $[-0.2, 0.2]$ initialisiert. Es waren keine Episodengrenzen vorhanden, so daß N_1 am Ende einer Eingabesequenz die erste Eingabe der nächsten Sequenz vorhersagen mußte.

Bei einer Lernrate von $\alpha = 1.0$ war die maximale Fehlerquadratsumme von N_1 nach 100 angelegten Eingabesequenzen kleiner als 0.06, bis auf die Vorhersage der ersten Eingabe einer Sequenz. Dieses Ergebnis blieb bei einer Reihe von Versuchen konstant, nur daß öfters die maximale Fehlerquadratsumme nach 100 angelegten Sequenzen niedriger als 0.06 war.

Obwohl bei diesem Beispiel kein Vertrauensnetzwerk notwendig wäre, wurde ein solches doch getestet. Das Vertrauensnetzwerk V_1 hatte dieselben Eingabeknoten wie N_1, keine versteckten Knoten und eine Ausgabeeinheit. Mit $\varepsilon = 0.1$ als Fehlerschranke für ein nach 100 Seqenzen festgehaltenes N_1 und einer Lernrate von 1.0, war der Fehler von V_1 nach 100 angelegten Eingabesequenzen kleiner 0.1 für die Sollausgabe 1.0 und keiner 0.6 für die Sollausgabe 0.

Als Hemmschuh für die Verbesserung von V_1 wirkte die Verbindung von der Eins-Einheit zur Ausgabeeinheit, da bei der korrekten Vorhersage von $b_1, \ldots, b_{10}$ durch N_1 diese Verbindung auf einen großen positiven Wert gebracht wurde. Dies führte dazu, daß V_1 bei der ungenauen Vorhersage von a bzw. x auch einen hohen Wert lieferte, wegen dieser sich negativ auswirkenden Generalisierung wurde der Eins-Knoten entfernt, wonach sich bessere Resultate ergaben.

Die Eingabesequenzen wurden von N_1 wie folgt reduziert:

$$R(N_1, N_1^0, 0.1)(\{G_1^{10}, G_1^{10}\}) = \{((1, a)), ((1, x))\},$$

wobei die Zeitrepräsentation vernachlässigt wurde, so daß N_2 als Eingabesequenz a erhielt, falls an N_1 die Sequenz G_1^{10} anlag, und die Sequenz x, falls N_1 G_2^{10} verarbeitete. Nachdem b_{10} von den Netzen verarbeitet wurde, sollte N_2 bei vorherigem Anliegen von G_1^{10} den Wert 1.0 und bei vorherigem Anliegen von G_2^{10} den Wert 0 ausgeben. Bei einer Lernrate von 1.0 und einer Gewichtsinitialisierung aus $[-0.2, 0.2]$ war der Fehler von N_2 nach 100 Trainigssequenzen kleiner als 0.1. Die Netzstruktur von N_2 war genau dieselbe wie die von V_1 von oben. Dieses Ergebnis bestätigte sich bei mehreren Versuchen.

Ließ man die Netze parallel lernen, so war der maximale Fehler von N_2 nach 200 angelegten Trainingssequenzen jedesmal kleiner als 0.15.

Wählte man p größer z. B. $p = 101$, so erhielt man dasselbe Ergebnis mit Netzen, die den obigen entsprachen. Denn bei der Optimierung von N_1 spielt die Länge der Eingabesequenz keine entscheidende Rolle, es müssen nur mehr Gewichte justiert werden.

Ein rekurrentes Netz, welches nach der Methode von Williams und Zipser trainiert wurde für $p = 11$, mit 13 Eingabeknoten incl. Eins-Knoten, einem versteckten Knoten und einem Ausgabeknoten lieferte mit Lernrate $\alpha = 1.0$ nach 50 000 Trainingssequenzen einen Fehler kleiner als 0.1. Bei einer Lernrate $\alpha = 10.0$ erhielt man dieses Ergebnis schon nach 5 000 Trainingssequenzen.

3.2.2 Komplexeres Beispiel

Seien die Mengen von Teilsequenzen

$$TE_1 = \{(a_2, a_4), (a_1, a_4, a_5, a_6, a_7), (a_3), (a_4)\} \text{ und}$$

$$TE_2 = \{(a_2, a_3, a_1, a_4, a_5, a_6, a_7), (a_1, a_4, a_5, a_6, a_7), (a_3), (a_1), (a_3, a_4), (a_2, a_3, a_1)\}$$

gegeben. Reiht man zufällig gewählte Elemente aus TE_1 hintereinander, so sieht man in der entstehenden Eingabesequenz, daß (a_2) die Teilfolge (a_4) und (a_1) die Teilfolge (a_4, a_5, a_6, a_7) nach sich zieht. Bei einer Hintereinanderreihung von Elementen aus TE_2 kann man erkennen, daß sich an (a_2) die Teilfolge (a_3, a_1) und sich an (a_1, a_4) die Teilfolge (a_5, a_6, a_7) anschließt. Da man auf Episodengrenzen, d. h. auf Grenzen für die Eingabesequenzen, und auf eine Zeitrepräsentation verzichtete, ist die Länge der entstehenden Eingabesequenzen unerheblich. Es wurden stattdessen Vertrauensnetzwerke verwendet.

Die Aufgabe ist nun, bei einer durch Hintereinanderreihung von Elementen aus TE_1 bzw. TE_2 entstehenden Eingabesequenz nach der Verarbeitung von a_3 den Wert 1.0 auszugeben, falls seit dem letzten Auftreten von a_1 die Eingabe a_2 nicht vorhanden war, ansonsten ist nach Verarbeitung von a_3 der Wert 0 auszugeben. Nach der Verarbeitung von a_4 ist 1.0 auszugeben, falls seit dem letzten Auftreten von a_1 die Eingabe a_3 nicht zu beobachten war, ansonsten ist nach Abarbeitung von a_4 der Wert 0 auszugeben. Bei der Teilfolge (a_1, a_4) erfolgt nach der Verarbeitung von a_4 keine Ausgabe.

Hier hatte N_1 einen versteckten Knoten, einen Eins-Knoten und 7 Eingabe- und Ausgabeknoten. Zwischen 400 und 500 angelegte Teilsequenzen aus TE_1 und zwischen 800 und 1 000 angelegte Teilseqenzen aus TE_2 benötigte N_1, bei einer Lernrate von $\alpha = 1.0$, als Training, um den maximalen Fehler unter $\varepsilon = 0.15$ zu drücken.

Danach ist

$$R(N_1, N_1^0, 0.15)(TE_1) = \{(1, a_2), (1, a_1), (1, a_3), (1, a_4)\} \text{ und}$$

$$R(N_1, N_1^0, 0.15)(TE_2) =$$

$$\{((1, a_2), (4, a_4)), ((1, a_1), (4, a_4)), (1, a_3), (1, a_1), ((1, a_3), (2, a_4)), (1, a_2)\}.$$

Die durch N_1 reduzierte Eingabesequenz bekommt man, indem man die Teilfogen aus denen die Eingabesequenz besteht, reduziert und in derselben Reihenfolge hintereinanderreiht, wie die ursprüngliche Eingabesequenz aufgebaut war, man muß nur die entsprechende Zeitrepräsentation in der Eingabesequenz beachten. Die resultierenden Eingabesequenzen, aber ohne Zeitrepräsentation, bekam nun N_2 zu Gesicht, um die gestellte Aufgabe zu lösen. Das Netz N_2 besaß dieselbe Netzarchitektur wie N_1, aber mit nur einer Ausgabeeinheit und es wurde wieder eine Lernrate von $\alpha = 1.0$ verwendet.

Bei parallelem Lernen von N_1 und N_2 war der Fehler von N_2 ab durchschnittlich 8 000 angelegten Teilsequenzen aus TE_1 und ab durchschnittlich 10 000 angelegten Teilsequenzen aus TE_2 kleiner als 0.25.

Hier wurden auch Vertrauensnetzwerke getestet, wobei 2 verschiedene Konstruktionen dieser Netze betrachtet wurden. Im ersten Fall hatte V_1 dieselbe Architektur wie N_2 und benötigte nach dem Lernen von N_1 etwa dieselbe Zeit wie N_1, um Werte zu liefern, bei denen der Fehler klein genug war, um die Sollausgabe gut erkennen zu können.

Im zweiten Fall war V_1 ein azyklisches Netz, welches außer den 7 Eingabeknoten von oben noch zusätzlich die Aktivierung der versteckten Knoten von N_1 als Eingabe bekam. Hier hatte V_1 denselben maximalen Fehler, wie vorhin beschriebenes zyklisches V_1, nach durchschnittlich 240 aus TE_1 und durchschnittlich 600 aus TE_2 angelegten Teilsequenzen. Die Lernbeschleunigung rührt daher, daß V_1 den Zustand von N_1 nicht erst lernen muß, sondern ihn schon als Eingabe erhält.

Dieselbe Aufgabe zu lösen, wurde mit einem zyklischen Netz mit 7 Eingabeeinheiten, einer versteckten Einheit und einer Ausgabeeinheit versucht. Mit dem Algorithmus von Williams und Zipser, bei einer Lernrate von 1.0, konnte die Aufgabe nicht innerhalb von 100 000 angelegten Teilfolgen gelöst werden. Es wurden jedoch für TE_1 und TE_2 jeweils nur 2 Tests durchgeführt, aufgrund von Rechenzeitbeschränkungen. Deshalb ist es nicht auszuschließen, daß bei entsprechender Lernrate und günstiger Initialisierung der Algorithmus von Williams und Zipser doch zum Erfog führt. Da aber mit diesem Algorithmus viel weiter in die Vergangenheit propagiert werden muß, um die gestellte Aufgabe zu lösen, ist selbst im günstigsten Fall eine wesentlich langsamere Konvergenz anzunehmen, als bei der hier vorgestellten Methode.

3.2.3 Beispiel für mehrstufige Kausaldetektoren

Seien die kontextfreien Grammatiken

$$G_1 = (\{A, B, C, D\}, \{f, g, e_1, e_2, e_3\}, \{S \to ABBe_1, S \to BAAe_1, A \to CCDe_2, B \to DDCe_2,$$

$$C \to fgge_3, D \to gffe_3, \}, \{S\}) \text{ und}$$

$$G_2 = (\{A, B, C, D\}, \{f, g, e_1, e_2, e_3\},$$

$$\{S \to ABBe_1, S \to BAAe_1, S \to AAAe_1, S \to AABe_1, S \to BBBe_1,$$

$$S \to BBAe_1, A \to CCDe_2, B \to DDCe_2, C \to fgge_3, D \to gffe_3, \}, \{S\})$$

gegeben, so daß gilt $L_{G_1} \subset L_{G_2}$. Explizit ist

$$L_{G_1} =$$

$$\{fgge_3gffe_3gffe_3e_2gffe_3fgge_3fgge_3e_2gffe_3fgge_3fgge_3e_2e_1,$$
$$gffe_3fgge_3fgge_3e_2fgge_3gffe_3gffe_3e_2fgge_3gffe_3gffe_3e_2e_1\}$$

und

$$L_{G_2} =$$

$$\{fgge_3gffe_3gffe_3e_2gffe_3fgge_3fgge_3e_2gffe_3fgge_3fgge_3e_2e_1,$$
$$gffe_3fgge_3fgge_3e_2fgge_3gffe_3gffe_3e_2fgge_3gffe_3gffe_3e_2e_1,$$
$$fgge_3gffe_3gffe_3e_2fgge_3gffe_3gffe_3e_2fgge_3gffe_3gffe_3e_2e_1,$$
$$fgge_3gffe_3gffe_3e_2fgge_3gffe_3gffe_3e_2gffe_3fgge_3fgge_3e_2e_1,$$
$$gffe_3fgge_3fgge_3e_2gffe_3fgge_3fgge_3e_2gffe_3fgge_3fgge_3e_2e_1,$$
$$gffe_3fgge_3fgge_3e_2gffe_3fgge_3fgge_3e_2fgge_3gffe_3gffe_3e_2e_1\}.$$

Nachdem die Eingabe e_1 verarbeitet wurde, soll 1.0 ausgegeben werden, falls $w \in L_{G_1}$ als Eingabesequenz angelegt wurde, und 0, falls $w \in L_{G_2} \setminus L_{G_1}$ angelegt wurde.

N_1 ist ein zyklisches Netz mit 5 Eingabeeinheiten, 2 versteckten Knoten und 5 Ausgabeeinheiten und soll die nächste Eingabe in der Eingabesequenz vorhersagen. Jede Eingabe führt zu einer Aktivierung von 1.0 genau eines Eingabeknotens und ebenso soll genau der Ausgabeknoten mit 1.0 aktiv sein, welcher der nächsten Eingabe entspricht. Die restlichen Eingabe- und Ausgabeknoten haben eine Aktivierung von 0.

V_1 erhält die 5 Eingabeknoten von N_1 und zusätzlich die 2 versteckten Einheiten von N_1 als Eingabe, daher benötigt V_1 keine versteckten Knoten. Der einzige Ausgabeknoten von V_1 soll 1.0 sein, falls N_1 die nächste Eingabe mit einer Fehlertoleranz von $\varepsilon_1 = 0.18$ vorhersagen konnte, ansonsten soll die Ausgabe 0 sein.

Falls die Ausgabe von V_1 kleiner als 0.75 war, bekam N_2 die neue Eingabe von N_1 auch als Eingabe. Da die Eingabesequenzen durch die Endemarkierungen e_i 'getaktet' sind und ein Vertrauensnetzwerk verwendet wurde, konnte auf die Zeitrepräsentation verzichtet werden. Da N_2, V_2 dieselben Aufgaben wie N_1, V_1 nur für reduzierte Eingabesequenzen hatten, hatte N_2 dieselbe Architektur wie N_1 und V_2 dieselbe wie V_1. Auch hier war die Fehlertoleranz $\varepsilon_2 = 0.18$ und V_2 reichte die Eingabe von N_2 an das Netz N_3 weiter, falls die Ausgabe von V_2 kleiner als 0.75 war.

N_3 hatte denselben Aufbau wie N_2 bzw. N_1 mit der Ausnahme, daß N_3 nur eine Ausgabeeinheit hatte, mit der N_3 die gestellte Aufgabe mit reduzierten Eingabesequenzen lösen sollte.

Alle Netze wurden mit einer Lernrate von $\alpha = 1.0$ trainiert und die Gewichtsmatrizen wurden mit Werten aus $[-0.2, 0.2]$ initialisiert. N_1 und V_1 wurden eingefroren, falls die Ausgabe von V_1 zum Sollwert 1.0 größer als 0.85 war.

Es war dann

$$R(N_1, N_1^0, 0.18)(L_{G_2}) =$$

$$\{\ ((1,f),(5,g),(9,g),(13,e_2),(14,g),(18,f),$$
$$(22,f),(26,e_2),(27,g),(31,f),(35,f),(39,e_2),(40,e_1)),$$
$$((1,g),(5,f),(9,f),(13,e_2),(14,f),(18,g),$$
$$(22,g),(26,e_2),(27,f),(31,g),(35,g),(39,e_2),(40,e_1)),$$
$$((1,f),(5,g),(9,g),(13,e_2),(14,f),(18,g),$$
$$(22,g),(26,e_2),(27,f),(31,g),(35,g),(39,e_2),(40,e_1)),$$
$$((1,f),(5,g),(9,g),(13,e_2),(14,f),(18,g),$$
$$(22,g),(26,e_2),(27,g),(31,f),(35,f),(39,e_2),(40,e_1)),$$
$$((1,g),(5,f),(9,f),(13,e_2),(14,g),(18,f),$$
$$(22,f),(26,e_2),(27,g),(31,f),(35,f),(39,e_2),(40,e_1)),$$
$$((1,g),(5,f),(9,f),(13,e_2),(14,g),(18,f),$$
$$(22,f),(26,e_2),(27,f),(31,g),(35,g),(39,e_2),(40,e_1))\}.$$

N_2 und V_2 wurden eingefroren, falls die Ausgabe von V_2 zum Sollwert 1.0 größer als 0.85 war. Es war dann

$$R(N_2, N_2^0, 0.18)(R(N_1, N_1^0, 0.18)(L_{G_2})) =$$

$$\{((1,f),(14,g),(27,g),(40,e_1)),$$
$$((1,g),(14,f),(27,f),(40,e_1)),$$
$$((1,f),(14,f),(27,f),(40,e_1)),$$
$$((1,f),(14,f),(27,g),(40,e_1)),$$
$$((1,g),(14,g),(27,g),(40,e_1)),$$
$$((1,g),(14,g),(27,f),(40,e_1))\}.$$

Tabelle 3.1 zeigt die Versuchsergebnisse für sequentielles Lernen der einzelnen Stufen. Die längere Lerndauer der 2. Stufe gegenüber der 1. Stufe hat ihre Ursache in der um $\frac{2}{3}$ verkürzten Eingabesequenz, die die 2. Stufe erhält, somit bekommt die 1. Stufe pro Eingabesequenz 3mal mehr Beispiele von Teilfolgen, die reduziert werden können, als die zweite Stufe. Die erste Stufe kann pro Eingabesequenz 9 Teilfolgen reduzieren, die zweite Stufe hingegen nur 3 Teilfolgen. In der ersten Stufe wurden höchstens 250 Eingabesequenzen angelegt und bei erfolglosem Lernen, d. h. die oben beschriebene ideale Reduzierung wurde nicht gelernt, der Versuch abgebrochen, wobei in 2 von 10 Fällen der Versuch erfolglos war. Bei erfolglosen Versuchen war N_1 in einem lokalen Minimum, welches wie folgt aussah: Sei die Teilfolge $F = (e_3, f, g, g, e_3)$ gegeben, welche an der Stelle i in der Eingabesequenz E beginnt, so war

$$R(N_1, N_1^0, 0.18)(F) = ((i+1, f), (i+2, g)),$$

Versuchnr.	1.Stufe e < 0.15	2.Stufe e < 0.15	3.Stufe e < 0.2
1.	160	300	16 000
2.	150	225	28 000
3.	165	350	25 250
4.	157	450	19 500
5.	100	400	13 500
6.	100	300	10 000
7.	150	250	13 500
8.	100	375	25 000

Tabelle 3.1: *Die Fehlerschranke e in der 1. und 2. Stufe bezog sich auf die Fehlerquadratsumme und die Fehlerschranke der 3. Stufe bezog sich auf den Fehlerbetrag der einzigen Ausgabeeinheit. Die angegebenen Daten sind die Anzahl der angelegten Eingabesquenzen, bevor die Fehlerschranke unterschritten wurde.*

obwohl man diese Teilfolge zu $(i+1, f)$ reduzieren hätte können. Ein symmetrisches, auch aufgetretenes, lokales Minimum ergibt sich, wenn man f und g vertauscht.

In einem von 10 Fällen gelangte N_2 in ein analoges, vorher für N_1 beschriebenes lokales Minimum.

Nun zum parallelen Fall, wo alle Netze gleichzeitig lernen. Hier treten 'On-line'-Effekte auf, so ist z. B. N_1 in einem lokalem Minimum der 1. Stufe, doch N_2 lernt dieses zu korrigieren, aber dann findet N_1 das globale Minimum der 1. Stufe und N_2 kann sich nicht auf die neue Eingabesequenzen umstellen. Es kam auch vor, daß während der Lernphase für N_2, das Netz N_1 gelernt hatte, die Eingabesequenz auf der Stufe von N_2 weiter zu reduzieren.

Siehe hierzu Tabelle 3.2 , wo die Versuchsergebnisse zusammengestellt sind für den parallelen Fall, wobei bei 17 500 Eingabesequenzen abgebrochen wurde. In den Fällen, in denen N_1 und N_2 im globalen Minimum ihrer jeweiligen Stufe waren, hätte man nur länger warten müssen, bis N_3 gelernt hätte die Aufgabe zu lösen, wie der sequentielle Fall gezeigt hatte. Die lokalen Minima waren dieselben, wie im sequentiellen Fall, außer bei den Versuchen mit Nummer 14 und 16. Dort konnte N_1 schon Aufgaben der 2. Stufe lösen. Beispielsweise reduzierte N_1 bei Versuch 16 die Teilsequenz $F = (e_2, g, f, f, e_3, f, g, g, e_3)$, welche an der Stelle i in der Eingabesequenz E beginnt, zu

$$R(N_1, N_1^0, 0.18)(F) = ((i, e_2), (i+1, g))\,,$$

wobei beim globalen Minimum der 1. Stufe im sequentiellen Fall gelten würde

$$R(N_1, N_1^0, 0.18)(F) = ((i, e_2), (i+1, g), (i+5, f))\,.$$

Bei diesen Versuchen gab es Instabilitäten, d. h. die Gewichte wurden so groß, daß es einen Überlauf am Computer gab. Die Instabilitäten ergeben sich, da N_1 lernt Aufgaben von N_2, zu lösen, obwohl N_2 noch trainiert wird, deshalb erhält N_2 plötzlich kürzere Eingabesequenzen und bekommt einen großen Fehler. Will sich nun N_2 auf die neuen Sequenzen einstellen, so hat N_1 die Sequenzen evtl. schon wieder verkürzt. Da das Einstellen von N_2 auf die neuen Sequenzen längere Zeit in Anspruch nimmt, hat N_1 unter Umständen genügend Zeit weitere Strukturen in der Eingabesequenz zu erkennen.

Nr.	lok. Min. N_1	lok. Min. N_2	glob. Min. N_1	glob. Min. N_2	max. Fe. N_3
1.	——	——	750	750	0.15 12 000
2.	750 - 1 500	1 500	1 500	750 - 1 500	0.15 5 250
3.	——	——	750	750	erfolglos
4.	——	——	750	750	erfolglos
5.	——	——	750	750	0.15 11 250
6.	——	——	750	750	erfolglos
7.	——	——	750	750	erfolglos
8.	ab 13 500 ü	12 750	750	12 750	erfolglos
9.	——	——	750	750	0.15 15 750
10.	——	ab 750	750	——	0.10 11 250
11.	——	——	750	750	0.25 17 250
12.	ab 750	1 500, 3 000	——	750, 2 250, 3 750	0.20 17 250
13.	——	——	750	750	erfolglos
14.	ab 3 750,10 500 ü	ab 750	750	——	erfolglos
15.	——	——	750	750	0.15 11 250
16.	ab 1 500,3 000 ü	2 250	750	1 500	erfolglos
17.	ab 750	ab 750	——	——	erfolglos
18.	ab 750	——	——	750	erfolglos
19.	ab 7 500	ab 750	750 - 7 500	——	erfolglos
20.	——	——	750	750	erfolglos

Tabelle 3.2: *Tabelle für paralleles Lernen von Kausaldetektoren in 3 Stufen mit der Trainigssequenz L_{G_2}. Bei den lokalen Minima ist angegeben, wieviel Trainigssequenzen ab Lernbeginn sich das Netz in einem lokalen Minimum befand. Bei den globalen Minima ist angegeben, nach wieviel angelegten Eingabesequenzen dies erreicht wurde. Mit 'ü' ist gemeint, daß die Gewichte extrem große Werte hatten und wuchsen bis es einen Überlauf für Float-Zahlen am Computer gab.*

Zum Vergleich wurden einige Versuche durchgeführt, aufbauend auf dem Algorithmus von Williams und Zipser, mit rekurrenten Netzen, welche jeweils zwei, fünf oder zehn versteckte, rekurrente Knoten besaßen und mit einer Anzahl von 75 000 bis 750 000 Eingabesequenzen trainiert wurden. Bei fünf Versuchen lernten die Netze nicht die gestellte Aufgabe zu lösen, wobei dreimal eine Lernrate von 1.0, einmal eine Lernrate von 0.1 und einmal eine Lernrate von 10.0 verwendet wurde. Aufgrund Rechenzeitbeschränkungen konnten keine weiteren Versuche durchgeführt werden. Die Aufgabe scheint jedoch sehr kompliziert zu sein für rekurrente Netze, da weder die letzten 27 Eingaben noch die ersten 13 Eingaben einer Eingabesequenz ausreichen, um die gestellte Aufgabe zu lösen.

3.3 Das Zeitüberbrücker-System

Die mehrstufigen Kausaldetektoren können jedoch zu einem System aus nur zwei Netzen kollabiert werden, wobei nach dem Training ein Netz N_1, auch **Automatisierer** genannt, alle Aufgaben der vorher beschriebenen Netze, außer den Vertrauensnetzwerken, übernimmt. Dieses von Jürgen Schmidhuber entwickelte Zeitüberbrücker-System wird in [Sch91c] genauer beschrieben.

Zu bestimmten Zeitpunkten $i_k,\ 2 \leq k \leq p$, während dem Anliegen einer Eingabesequenz, gibt es eine gewünschte Ausgabe und die gestellte Aufgabe für ein Netz ist es, diese Ausgabe zu erzeugen.

Füttert man diese gewünschte Ausgabe $d(i_k)$ im nächsten Schritt zusätzlich zu der Eingabe $e(i_k+1) \in \mathrm{R}^m$ in die Netze, so hat man eine erweiterte Eingabe $\tilde{e}(i_k+1)$, nur daß bestimmte Komponenten der neuen Eingabe lediglich zu festen Zeitpunkten ($d_l(i_k) \neq \zeta$) vorhergesagt werden müssen. Somit ist die Lösung der gestellten Aufgabe auf die Vorhersage der nächsten Eingabe zurückgeführt. Es ist

$$\tilde{E}(i) := \begin{pmatrix} E(i) \\ d(i) \end{pmatrix} \in \mathrm{R}^{m+n},\ 1 \leq i \leq p, \quad \text{und}$$

$$\tilde{E}(p+1) := \begin{pmatrix} E(1) \\ d(p+1) \end{pmatrix} \in \mathrm{R}^{m+n}$$

die neue Eingabe für das System. Ist $d_l(1) = \zeta, 1 \leq l \leq n$, so kann die Grenze für die Eingabesequenz überschritten werden, um als zusätzliche Eingabe zu $E_{j+1}(1)$, der aktuellen Sequenz E_{i+1}, die letzte gewünschte Ausgabe der letzten Sequenz E_i zu haben, es ist $\tilde{E}_{i+1}(1) = \binom{E_{i+1}(1)}{d^i(p+1)}$, wobei $d^i(p+1)$ die letzte gewünschte Ausgabe der letzten Sequenz ist und $\tilde{E}_i(p+1)$ fällt somit weg. Dann wird ein Schritt eingespart, aber es muß über die Grenzen der Eingabesequenzen propagiert werden. Versteckte Knoten müssen nie zurückgefüttert werden, da es für sie keine gewünschte Ausgabe gibt, womit man die neue Eingabe um einige Komponenten pro Eingabevektor verkleinern kann.

Das Netz N_2, auch **Zeitüberbrückungsnetzwerk** oder kürzer **Zeitüberbrücker** genannt, wird trainiert, die nächste Eingabe in der durch N_1 reduzierten Eingabesequenz vorherzusagen, wobei N_2 v_2 versteckte Knoten und $o_2 = m + l,\ 1 \leq l \leq n$, Ausgabeknoten besitzt. Außer den Ausgabeknoten für die nächste Vorhersage der Eingabe besitzt N_1 zusätzlich $v_2 + o_2$ Ausgabeknoten, um die Aktivierung der versteckten Knoten und der Ausgabeknoten von N_2 vorherzusagen. Also hat N_1 genau $o_1 = v_2 + o_2 + (m+l) = v_2 + 2(m+l)$ Ausgabeknoten. Die

Ausgabeknoten von N_1 und N_2 sind keine rekurrenten Knoten und sind untereinander nicht verbunden.

Gibt es Strukturen in der Eingabesequenz, welche größere Zeitsprünge in der Eingabesequenz zum Erkennen dieser Struktur notwendig machen, so wird diese Struktur zuerst von N_2 gelernt, da N_2 kürzere Eingabesequenzen erhält. Beim Auftreten einer Eingabe $\tilde{e}_a$, welche später zur Vorhersage einer in der Zukunft liegenden Eingabe $\tilde{e}_b, b > a$, benötigt wird, speichert N_2 diese anliegende Eingabe in seinen versteckten Knoten ab. Aus diesem Grund wird auch N_1 gezwungen, diese Eingabe $\tilde{e}_a$ entsprechend in seinen versteckten Knoten abzuspeichern, um die Aktivierung der Einheiten von N_2 im weiteren vorauszusagen. Lernt N_1 die Eingabe $\tilde{e}_a$ abzuspeichern, so besitzt N_1 die notwendige Information $\tilde{e}_b$ vorherzusagen und wird diese Vorhersage auch lernen. Ab jetzt kann die Eingabesequenz durch N_1 weiter reduziert werden, d. h. $\tilde{e}_b$ wird als Eingabe nicht mehr an N_2 anliegen. Dies führt dazu, daß sich N_2 neuen Aufgaben widmen kann, da N_2 keinen Fehler mehr bekommt bei der Vorhersage von $\tilde{e}_b$. Hat N_2 gelernt, eine Eingabe vorherzusagen, so wird das nötige Wissen zur Vorhersage dieser Eingabe an N_1 vermittelt. Sagt nun N_1 aufgrund dieses Wissens diese Eingabe korrekt vorher, so ist N_2 der Aufgabe entledigt, diese Eingabe vorherzusagen und kann nun versuchen, kompliziertere Strukturen in der Eingabesequenz zu erkennen, was bisher evtl. nicht möglich war, da viele Knoten zur Vorhersage dieser Eingabe benötigt wurden. So können schrittweise immer komplexere Strukturen in der Eingabesequenz ermittelt werden und durch nur ein einziges Netz N_1 erkannt werden.

Die Vorhersage der Aktivierung der Ausgabeeinheiten von N_2 durch N_1 ist notwendig, da in der reduzierten Eingabesequenz $\tilde{e}_a$ direkt vor $\tilde{e}_b$ stehen kann, d. h.

$$R(N_1, N_1^0, \varepsilon)(\tilde{E})(l) = (a, \tilde{e}_a) \text{ und}$$

$$R(N_1, N_1^0, \varepsilon)(\tilde{E})(l+1) = (b, \tilde{e}_b).$$

Ist dies der Fall, so können die versteckten Knoten von N_2 umgangen werden, indem die Ausgabeeinheiten von N_2 direkt durch Verbindungen von den Eingabeeinheiten beeinflußt werden. N_1 könnte darauf trainiert werden, statt die Aktivierung der Ausgabeknoten von N_2 die letzte Eingabe von N_2 auszugeben, doch in den Ausgabeknoten ist die relevante Information deutlicher enthalten.

Dieses System wurde mit G_1^{20} und G_2^{20} aus Abschnitt 3.2.1 getestet. N_1 hatte hier 24 Eingabeknoten (incl. Eins-Knoten und zurückgefütterte gewünschte Ausgabe), einen versteckten, rekurrenten Knoten und 47 nichtverbundene, nichtrekurrente Ausgabeknoten. Das Netz N_2 hatte ebenfalls dieselben 24 Eingabeknoten wie N_1, einen versteckten, rekurrenten Knoten (welcher nicht notwendig wäre) und 23 nichtverbundene, nichtrekurrente Ausgabeknoten. Diese Ausgabeknoten und der versteckte Knoten von N_2 mußten durch N_1 zusätzlich zur nächsten Eingabe vorhergesagt werden. Die beiden Nezte wurden parallel mit einer Lernrate von $\alpha = 1.0$ trainiert, wobei jeweils nur 3 Schritte zurückpropagiert wurde. Tabelle 3.3 zeigt die Ergebnisse, falls das System die Aufgabe in weniger als 5 000 angelegten Eingabesequenzen gelernt hatte und Tabelle 3.4 zeigt die Ergebnisse, falls das System zum Lösen der gestellten Aufgabe mehr als 5 000 angelegte Eingabesequenzen benötigte, was in 6 von 30 Tests der Fall war.

Abb. 3.1 zeigt die beiden Gewichtsmatrizen für N_1 nach erfolgreichem Lernen.

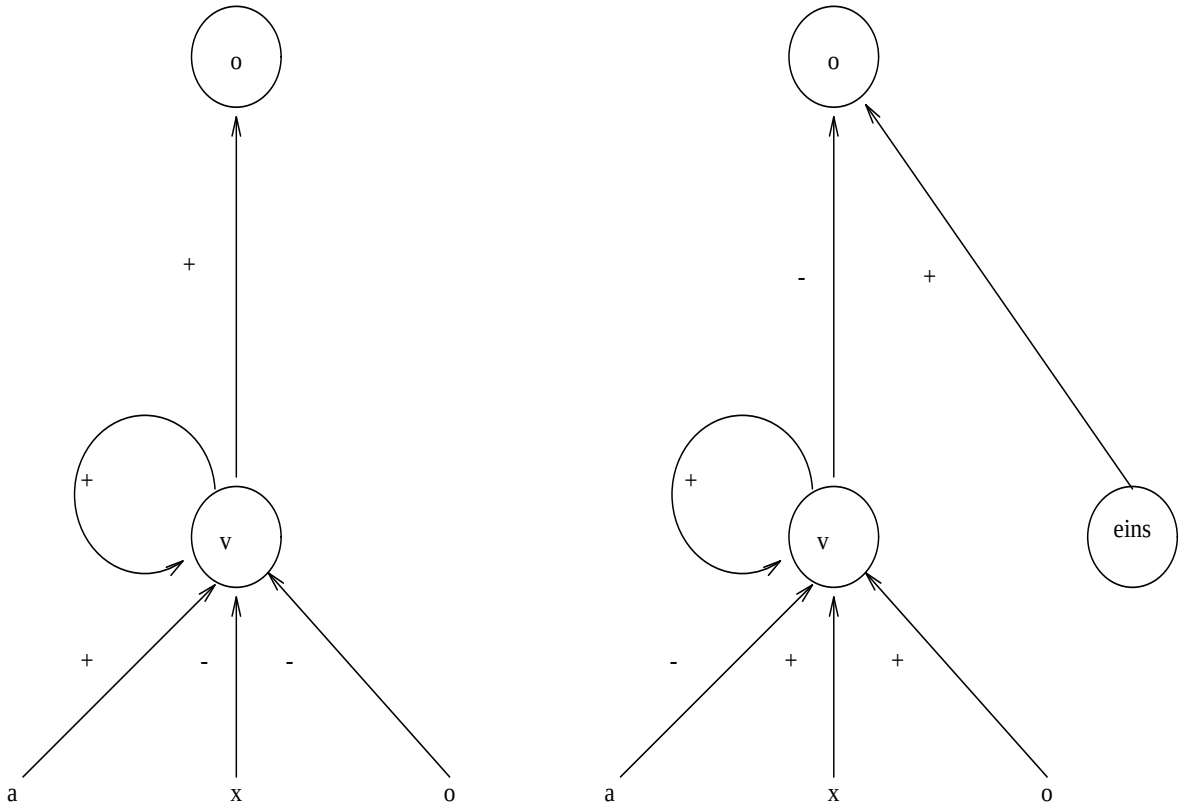
Um zu zeigen, wie schwer die Aufgabe für ein rekurrentes neuronales Netz zu lernen ist, wurden auch Versuche mit G_1^4, G_2^4 und G_1^6, G_2^6 durchgeführt. Es wurden die gewünschte Ausgaben

Versuchnr.	max. Fehler	angelegte Sequenzen
1.	0.11	3 000
2.	0.13	2 500
3.	0.02	3 500
4.	0.18	2 500
5.	0.09	3 500
6.	0.09	3 000
7.	0.10	3 000
8.	0.09	2 500
9.	0.12	4 000
10.	0.25	2 500
11.	0.14	2 500
12.	0.10	2 500
13.	0.07	3 000
14.	0.07	3 000
15.	0.09	2 500
16.	0.21	2 500
17.	0.09	3 000
18.	0.10	2 500
19.	0.11	3 000
20.	0.13	3 000
21.	0.10	2 500
22.	0.10	2 500

Tabelle 3.3: *Tabelle für das Lernen des Zeitüberbrücker-Systems für G_1^{20}, G_2^{20}, falls die Lernzeit weniger als 5 000 angelegte Eingabesequenzen benötigte. Der maximale Fehler ist der Fehlerbetrag von N_2 bei der Vorhersage der gewünschten Ausgabe.*

Versuchnr.	max. Fehler	angelegte Sequenzen
1.	0.05	30 000
2.	0.03	35 000
3.	0.06	25 000
4.	0.03	15 000
5.	0.05	30 000

Tabelle 3.4: *Tabelle für das Lernen des Zeitüberbrücker-Systems für G_1^{20}, G_2^{20}, falls die Lernzeit mehr als 5 000 angelegte Eingabesequenzen benötigte. Der maximale Fehler ist der Fehlerbetrag von N_2 bei der Vorhersage der gewünschten Ausgabe. Die längere Lernzeit ist auf die unerwünschte Einstellung von N_1 auf die aktuelle Aktivierung von N_2 zurückzuführen.*



Hier sind die Gewichtsmatrizen fuer den Zeitueberbruecker nach erfolgreichem Lernen gegeben.
Das Zeichen '+' bedeutet ein positives Gewicht und '-' bedeutet ein negatives Gewicht.
v ist die versteckte, rekurrente Einheit und o ist die Ausgabeeinheit fuer die gewuenschte
Sollausgabe und 'eins' ist der Einsknoten. Das 'o' unten ist die zurueckgefuetterte
gewuenschte Ausgabe.

Abbildung 3.1: *Dargestellt sind hier die beiden gleich häufig vorgekommenen Gewichtsmatrizen für den Zeitüberbrücker nach erfolgreichem Lernen. Große Gewichte, welche sich aufgrund der Effekte aus 3.3.1 aufbauten, wurden nicht berücksichtigt.*

Testnr.	α	max. angel. Seq.	erfolgr.	gesamt
1.	1.0	5 Millionen	14	16
2.	10.0	5 Millionen	2	9
3.	4.0	5 Millionen	5	9

Tabelle 3.5: *Tabelle der Versuche für G_1^4, G_2^4 mit rekurrenten Netzen. Angegeben ist die Lernrate, die maximale Anzahl der angelegten Eingabesequenzen vor Abbruch des Programms, die Anzahl der erfolgreich abgeschlossenen Versuche und die Gesamtzahl der Versuche.*

Testnr.	α	max. angel. Seq.	erfolgr.	gesamt
1.	1.0	5 Millionen	0	5
2.	10.0	5 Millionen	0	6
3.	0.1	5 Millionen	0	5

Tabelle 3.6: *Tabelle der Versuche für G_1^6, G_2^6 mit rekurrenten Netzen. Angegeben ist die Lernrate, die maximale Anzahl der angelegten Eingabesequenzen vor Abbruch des Programms, die Anzahl der erfolgreich abgeschlossenen Versuche und die Gesamtzahl der Versuche.*

zurückgefüttert, wie beim Zeitüberbrücker-System und außerdem 5 bzw. 7 Schritte zurückpropagiert. Das Netz hatte 8 bzw. 10 Eingabeknoten (incl. Eins-Knoten und gewünschter Ausgabe) und 7 bzw. 9 Ausgabeknoten, welche nicht rekurrent waren und untereinander nicht verbunden waren. Außerdem besaß das Netz einen versteckten, rekurrenten Knoten.

Tabelle 3.5 zeigt die Ergebnisse für G_1^4, G_2^4 und Tabelle 3.6 zeigt die Resultate für G_1^6, G_2^6. Tabelle 3.7 zeigt die Ergebnisse für G_1^4, G_2^4, für erfolgreiches Lernen. Bei G_1^4, G_2^4 wurden einige $\vartheta_v(t)$ für den versteckten Knoten v ausgegeben. Tabelle 3.8 zeigt die Werte, wobei o der Knoten für die gewünschte Ausgabe ist.

Dies bestätigt die theoretischen Ergebnisse aus Abschnitt 2.2 und das dort beschriebene Problem ist auch die Ursache für die lange Lernzeit.

Das System G_1^{20}, G_2^{20} wurde auch mit rekurrenten Netzen getestet, welche dieselbe Netzstruktur wie vorhin bei G_1^4, G_2^4 besaßen. Tabelle 3.9 zeigt die Ergebnisse für diesen Fall.

Mit rekurrenten, untereinander verbundenen Ausgabeknoten und zwei versteckten, rekurrenten Koten wurde obiges System auch getestet. Hier erhöht sich die Anzahl der Wege, entlang denen der Fehler zurückfließen kann. Bei günstiger Initialisierung kann sich deshalb ein Erfolg einstellen. Tabelle 3.10 zeigt die Ergebnisse, bei maximal 50 000 angelegten Eingabesequenzen.

Bei Versuchen, in denen 32 Schritte zurückpropgiert wurde, gab es Gewichtsexplosionen, d. h. es gab einen Überlauf der Float-Zahlen. Der Fehler schaukelte sich auf, falls die rekurrenten Gewichte zu hoch waren (siehe Abschnitt 2.1).

Testnr.	α	angelegte Seq.
1.	1.0	1 900 000
2.	1.0	3 500 000
3.	1.0	2 450 000
4.	1.0	900 000
5.	1.0	250 000
6.	1.0	750 000
7.	1.0	300 000
8.	1.0	600 000
9.	1.0	350 000
10.	1.0	2 250 000
11.	1.0	250 000
12.	1.0	350 000
13.	1.0	550 000
14.	1.0	200 000
15.	10.0	150 000
16.	10.0	350 000
17.	4.0	2 400 000
18.	4.0	500 000
19.	4.0	1 200 000
20.	4.0	100 000
21.	4.0	250 000

Tabelle 3.7: *Tabelle der Versuche für G_1^4, G_2^4 mit rekurrenten Netzen. Angegeben ist die Lernrate, die Anzahl der angelegten Eingabesequenzen, bevor der maximale Fehlerbetrag bei der gewünschten Ausgabe einen Wert von 0.1 unterschritten hat.*

Gewichte	$\vartheta_v(4)$	$\vartheta_v(3)$	$\vartheta_v(2)$	$\vartheta_v(1)$
$w_{ov} = 0.072$ $w_{vv} = 0.093$	$8.6 * 10^{-3}$	$1.8 * 10^{-4}$	$4.1 * 10^{-6}$	$9.3 * 10^{-8}$
$w_{ov} = 0.185$ $w_{vv} = 0.136$	$3.2 * 10^{-3}$	$1.0 * 10^{-4}$	$3.6 * 10^{-6}$	$1.3 * 10^{-7}$
$w_{ov} = 0.27$ $w_{vv} = -0.036$	$6.8 * 10^{-3}$	$-5.5 * 10^{-5}$	$4.9 * 10^{-7}$	$-4.4 * 10^{-9}$
$w_{ov} = 2.0$ $w_{vv} = -0.036$	$2.48 * 10^{-2}$	$-1.75 * 10^{-4}$	$1.3 * 10^{-6}$	$-1.0 * 10^{-8}$
$w_{ov} = 2.0$ $w_{vv} = 2.0$	$1.9 * 10^{-2}$	$8.8 * 10^{-3}$	$4.4 * 10^{-3}$	$2.2 * 10^{-3}$
$w_{ov} = 5.0$ $w_{vv} = 5.0$	$5.6 * 10^{-3}$	$2.3 * 10^{-4}$	$1.4 * 10^{-5}$	$9.0 * 10^{-7}$

Tabelle 3.8: *Für G_1^4, G_2^4 sind hier noch einige Werte ϑ für den Fehlerrückfluß aufgelistet, wobei v der versteckte Knoten und o der Ausgabeknoten sein soll. Bei den Werten in der letzten Zeile ergibt sich ein solch geringer Fehlerrückfluß, da aufgrund der hohen Gewichte die versteckte Einheit eine Aktivierung um 1.0 oder 0 hat, so daß die Ableitung extrem klein wird.*

Testnr.	α	max. angel. Seq.	erfolgr.	gesamt
1.	1.0	1 Million	0	3
2.	10.0	1 Million	0	2
3.	3.0	1 Million	0	2
4.	1.0	2 Millionen	0	1
5.	10.0	1.5 Millionen	0	1

Tabelle 3.9: *Tabelle der Versuche für G_1^{20}, G_2^{20} mit rekurrenten Netzen. Angegeben ist die Lernrate, die maximale Anzahl der angelegten Eingabesequenzen vor Abbruch des Programms, die Anzahl der erfolgreich abgeschlossenen Versuche und die Gesamtzahl der Versuche.*

Testnr.	α	gesamt	erfolgr.	bei	Modifikationen
1.	1.0	7	1	6 000	keine
2.	2.0	4	1	9 000	keine
3.	3.0	2	0		keine
4.	0.5	5	1	24 000	keine
5.	1.0	4	0		25 zurück
6.	2.0	4	0		Aus. nicht rek.
7.	1.0	12	0		Aus. nicht rek.

Tabelle 3.10: *Tabelle der Versuche für G_1^{20}, G_2^{20} mit rekurrenten Netzen, wobei nun auch die Ausgabeeinheiten rekurrent sind und 2 versteckte, rekurrente Knoten verwendet wurden. Es wurden maximal 50 000 Eingabesequenzen angelegt. Bei erfolgreichen Versuchen ist angegeben ('bei'), nach wievielen angelegten Eingabesequenzen der Fehler der Ausgebeeinheit für die gewünschte Ausgabe kleiner gleich 0.07 war. Bei einigen der Versuchen wurden 25 statt 21 Schritte zurückpropagiert und einige andere Versuche hatten nichtrekurrente, untereinander nicht verbundene Ausgabeeinheiten.*

3.3.1 Probleme beim Zeitüberbrücker-System

Automatisierer stellt sich auf aktuellen Zeitüberbrücker ein

Da Zeitüberbrücker (N_2) und Automatisierer (N_1) während einer Eingabesequenz für N_1 verschieden oft aktualisiert werden, gibt es 'On-line' Effekte beim Lernen der Vorhersage der Aktivierung der Zeitüberbrückerknoten durch N_1. Das Netz N_1 muß zwischen 2 Aktualisierungen von N_2 immer dieselbe Aktivierung der Zeitüberbrücker-Knoten vorhersagen, nun werden die Gewichte von N_1 so geändert, daß im nächsten Zeitschritt der Fehler kleiner bei der Vorhersage der Aktivierung der Knoten von N_2 ist. So kann zwischen 2 Aktualisierungen von N_2 der Fehler bei der Vorhersage der Zeitüberbrückerknoten durch N_1 sehr klein werden, da sich der Automatisierer auf den aktuellen Zeitüberbrückerzustand einpendelt. Während der im folgenden angelegten Eingabesequenzen muß diese konstante Ausgabe bei der Vorhersage der Zeitüberbrückereinheiten durch den Automatisierer verändert werden, falls sich die Aktivierung der Knoten von N_2 geändert hat. Hier ergibt sich evtl. wieder eine konstante Ausgabe durch N_1, wenn die Zeitspanne zwischen nächster und letzter Zeitüberbrücker-Aktualisierung hinreichend groß ist. So stellt sich N_1 nur auf den aktuellen Zustand von N_2 ein und lernt nicht, die dynamische Änderung der Aktivierung der Zeitüberbrückereinheiten zu simulieren.

Das eben beschriebene Problem ist die Ursache für die längere Lernphase (über 5 000) bei manchen Versuchen im vorherigen Kapitel mit G_1^{20}, G_2^{20}. Denn dort ändert der Zeitüberbrücker seine Aktivierungen nur alle 20 Schritte.

Mit G_1^{100} und G_2^{100} aus Abschnitt 3.2.1 wurde ein Zeitüberbrücker-System getestet, doch obiger 'On-line' Effekt verhinderte, daß N_1 lernte, die Ausgabeeinheit von N_2 für die gewünschte Ausgabe 1.0, falls die Sequenz $(a, b_1, b_2, \ldots, b_{100})$ zuletzt angelegen ist, und 0, falls die Sequenz $(x, b_1, b_2, \ldots, b_{100})$ zuletzt angelegen ist, vorherzusagen. Der Automatisierer konnte nur die Eingabe a bzw. x incl. der gewünschten Ausgabe für die vorherige Eingabesequenz nicht vorhersagen. Deswegen wurde N_2 nur bei der Eingabe a oder x, incl. der gewünschten, zurückgefütterten Ausgabe, aktualisiert, d. h. N_2 wurde nur alle 101 Schritte aktualisiert. Die Eingabe

Testnr.	max. Fehler	angelegte Sequenzen
1.	0.10	30 000
2.	0.05	25 000
3.	0.05	25 000
4.	0.16	28 000
5.	0.25	40 000

Tabelle 3.11: *Tabelle der Versuche für G_1^{100}, G_2^{100} beim Zeitüberbrücker-System. Der maximale Fehler ist der Fehlerbetrag der Ausgabeeinheit für die gewünschte Ausgabe.*

Knoten	Eins	a	x	v	g	b_{100}
v	-3.2	3.9	-1.8	7.3	-6.6	3.8
g	-2.0			5.1		
g_C	-6.2	4.5	-4.3	4.3	-5.1	2.2

Tabelle 3.12: *Gewichtstabelle für die Gewichte des Automatisierers des 5. Tests von Tabelle 3.11 nach erfolgreichem Lernen. In einer Zeile sind die Verbindungen zum Knoten in der 1. Spalte dieser Zeile aufgeführt, wobei in der 1. Spalte außer v nur Ausgabeknoten stehen. In der 1. Zeile stehen außer v nur Eingabeeinheiten, wobei g die zurückgefütterte gewünschte Ausgabe ist. v ist der rekurrente versteckte Knoten, g ist der Ausgabeknoten für die gewünschte Ausgabe, Eins ist der Einsknoten und g_C ist der Ausgabeknoten, der die Zeitüberbrückerausgabe für die gewünschte Ausgabe vorhersagen soll.*

a führte dazu, daß der Ausgabeknoten für die gewünschte Ausgabe nach der Aktualisierung von N_2 eine Aktivierung von 1.0 erhielt, wohingegen dieser Knoten bei der Eingabe x nach der Aktualisierung von N_2 eine Aktivierung von 0 besaß.

Das Netz N_1 wurde nun mit einer seiner Ausgabeeinheiten trainiert, welche zur Vorhersage des Ausgabeknotens von N_2 für die gewünschte Ausgabe zuständig war, während 100 Schritten einen Wert von 0 oder 1.0 auszugeben. Bei einer Lernrate von 1.0 wurde ein Fehler von 0.9 bei dieser Ausgabeeinheit von N_1 im 1. Schritt nach der Aktualisierung von N_2 auf einen Fehler von 0.1 im 30. Schritt zurückgeschraubt. Der Automatisierer konnte die dynamische Änderung des Zeitüberbrückers nicht korrekt vorhersagen, da sich der Automatisierer auf den aktuellen Zeitüberbrückerzustand so schnell einstellte, daß der Fehler, der zum Erlernen der dynamischen Änderung der Aktivierung der Zeitüberbrückerknoten notwendig ist, zu klein war.

In $\frac{1}{3}$ der Fälle lernte das System trotzdem, die gewünschte Ausgabe zu liefern, wobei durchschnittlich 30 000 angelegte Sequenzen notwendig waren. Tabelle 3.11 zeigt die Ergebnisse bei erfolgreichem Lernen.

Tabelle 3.12 zeigt die Gewichte, die bei erfolgreichem Lernen im 5. Test von Tabelle 3.11 entstanden. Es ist der störende Einfluß des 'On-line' Effekts deutlich erkennbar, wie man aus den Verbindungen von der Eins-Einheit sieht.

Dieser Effekt wird vermieden, wenn man die Lernrate des Automatisierers hinreichend klein wählt, so daß keine Einstellung des Automatisierers auf den aktuellen Zeitüberbrückerzustand möglich ist. Eine andere Möglichkeit ist der Verzicht auf die 'Real-Time' Version des Algorith-

mus, da so die Gewichte des Automatisierers während einer Eingabesequenz nicht verändert werden und deshalb der 'On-line' Effekt nicht möglich ist, es kommt dann aber zu sehr großen Gewichtsänderungen, da die einzelnen Änderungen aufsummiert werden.

Der 2. Grund für eine solch lange Lernzeit liegt darin, daß die versteckte, rekurrente Einheit von N_1 zu Lernbeginn auf 1.0 gesetzt wird. Dies geschieht durch starkes positives Gewicht von der Einseinheit und durch eine starke posititive Verbindung des Knotens auf sich. Die versteckte Einheit wird verwendet, die Ausgabeknoten für $a, x, b_1, \ldots, b_{100}$ auf 0 zu drücken, denn es soll ja nur der Knoten, welcher der neuen Eingabe zugeordnet ist, ungleich 0 sein. Der versteckte Knoten muß jedoch zum Speichern von a bzw. x verwendet werden, deshalb muß der Fehler bei der Vorhersage der gewünschten Ausgabe größer sein, als die zusätzlichen Fehler, die entstehen, falls die rekurrente Einheit keine Aktivierung um 1.0 hat. Der Fehler bei der Vorhersage der gewünschten Ausgabe muß um einen Faktor 100 * 100 = 10 000 größer sein, als der durchschnittliche Fehler, der bei einem Ausgabeknoten entsteht, falls dieser mit einer weniger aktiven rekurrenten Einheit auf null gedrückt wird. Dieser Faktor ergibt sich, da die gewünschte Ausgabe nur ein Mal pro 101 Schritte einen Fehler liefert, wohingegen der Fehler für das auf null Drücken der einzelnen Ausgabeknoten bei jedem Schritt bei 100 Knoten gegeben ist.

Strukturen verschiedener Komplexität

Gibt es in der Eingabesequenz Strukturen mit verschiedener Komplexität, so wird der Automatisierer lernen, die einfachste Struktur zu erkennen, wobei mit einfachster Struktur die Struktur mit geringster Komplexität gemeint ist, d. h. die Funktionen a, v aus 3.1 sind am wenigsten komplex. Mit der reduzierten Eingabesequenz wird N_2 dann versuchen, die zweiteinfachste Struktur vorherzusagen. Hat nun N_2 gelernt Teile von dieser Struktur vorherzusagen und kann N_1 die Aktivierung von N_2 gut vorhersagen, so wird N_1 evtl. lernen, auch Teile der zweiteinfachsten Struktur vorherzusagen. Dadurch werden jedoch die an N_2 anliegenden Eingabesequenzen weiter reduziert und der Zeitüberbrücker muß nun lernen, diese neue Sequenz richtig zu verarbeiten. Da nun N_2 zu vorher eine andere Dynamik in der Änderung der Aktivierung seiner Knoten besitzt, erhält N_1 einen großen Fehler bei der Vorhersage der Aktivierung der Knoten von N_2. Nun wird der Automatisierer versuchen zu lernen, diese neue Dynamik des Zeitüberbrückers vorherzusagen und deshalb evtl. verlernen, die gerade gelernten Teile der zweiteinfachsten Struktur richtig vorherzusagen. Aus diesem Grund bekommt N_2 wieder die ursprüngliche Eingabesequenzen und stellt sich wieder auf die alten Trainingssequenzen ein. Somit muß N_1 nun wieder die ursprüngliche Dynamik von N_2 vorhersagen und man ist wieder im Ausgangszustand.

Die Anzahl der von N_1 vorherzusagenden Aktivierungen von Knoten von N_2 ist immer größer als die Anzahl der Komponenten der Eingabe, welche durch N_1 vorherzusagen sind. Aus diesem Grund ist es sehr wahrscheinlich, daß bei einer Änderung der Trainingssequenzen für N_2 und der daraus resultierenden Änderung der Dynamik von N_2, der Fehler von N_1, bei der Vorhersage von N_2, wesentlich größer als der übrige Fehler von N_1 ist. Dies wird verstärkt, da der Fehler, bei der Vorhersage der einfachsten Struktur durch N_1, sehr gering ist.

Verkürzt ausgedrückt ist das Problem nun folgendes:
N_1 lernt eine weitere Eingabe vorherzusagen, deshalb bekommt N_2 andere Eingabesequenzen zu Gesicht. Daher ändert sich abrupt die Dynamik von N_2 und dies führt zu einem großen Fehler von N_1 bei der Vorhersage von N_2. Durch diesen großen Fehler verlernt N_1, die gerade gelernte

weitere Eingabe vorherzusehen. Nun bekommt N_2 wieder die ursprüngliche Eingabesequenz und der Systemzustand ist wie zu Beginn.

Mit dem Beispiel aus Abschnitt 3.2.3, mit den Grammatiken G_1 und G_2, wurden Versuche gestartet mit dem Zeitüberbrücker-System. N_1 lernt die 1. Komplexitätsstufe sehr gut vorherzusagen, d. h. die Teilsequenzen, an der Stelle i beginnend, werden wie folgt reduziert:

$$R(N_1, N_1^0, \varepsilon)((g, f, f, e_3)) = ((i, g)) \text{ und}$$
$$R(N_1, N_1^0, \varepsilon)((f, g, g, e_3)) = ((i, f)).$$

Der Automatisierer reduziert keine anderen als obige Teilsequenzen und der Zeitüberbrücker ist dabei, die 2. Komplexitätsstufe zu lernen. Wäre N_2 in der 2. Komplexitätsstufe schon perfekt, so würde es zu folgender Reduktion, von bei i beginnender Teilsequenzen der 2. Komplexitätsstufe, kommen:

$$R(N_2, N_2^0, \varepsilon)((g, f, f, e_2)) = ((i, g)) \text{ und}$$

$$R(N_2, N_2^0, \varepsilon)((f, g, g, e_2)) = ((i, f)).$$

Doch während der Lernphase von N_2 lernt auch N_1 größere Teilsequenzen reduzieren wie z. B. die Teilsequenz $(e_2, f, g, g, e_3, g, f, f, e_3, g)$ beginnend an der Stelle i:

$$R(N_1, N_1^0, \varepsilon)((e_2, f, g, g, e_3, g, f, f, e_3, g)) = ((i, e_2), (i+1, f), (i+9, g)).$$

Dadurch erhält N_2 weiter reduzierte Eingabesequenzen und N_1 kann die Aktivierung der Knoten von N_2 nicht vorhersagen. Deswegen gibt es betragsmäßig große Gewichtsänderungen von N_1, da N_1 nun einen großen Fehler erhält und obige Verkürzung der Teilsequenz verlernt wird und es gibt folgende Reduzierung:

$$R(N_1, N_1^0, \varepsilon)((e_2, f, g, g, e_3, g, f, f, e_3, g)) = ((i, e_2), (i+1, f), (i+5, g), (i+9, g)).$$

Bei größerer Lernrate ($\alpha > 1.0$) kann es auch zu Gewichtsexplosionen bei den Gewichten von N_1 kommen, d. h. N_1 hat um Zehnerpotenzen zu große Gewichte, die wieder abgebaut werden müssen, was aber sehr lange Zeit in Anspruch nimmt, da die Aktivierung der Knoten von N_1 nahe an 1.0 oder 0 liegt und deshalb die Ableitung der Aktivierungsfunktion sehr niedrig ist. Die Gewichtsexplosion entsteht durch den plötzlich auftretenden großen Fehler bei der Vorhersage von N_2 durch N_1.

Dieses Problem kann behoben werden, wenn man die 'Off-line' Version, die im Abschnitt 3.3.2 beschrieben ist, des Algorithmus verwendet, da dann die Eingabesequenzen für N_2 konstant bleiben.

Bei hinreichend kleiner Lernrate wird dieser nachteilige Effekt nicht dazu führen, daß das System nicht mehr aus dem lokalen Minimum findet. Da durch die auftretenden plötzlichen Änderungen das Netz N_1 nicht alles bisherige Gelernte wieder verlernt, d. h. die gelernte Reduktion der Eingabesequenz durch N_1 bleibt erhalten.

3.3.2 Modifikationen des Algorithmus

Diese von Schmidhuber vorgeschlagenen Modifikationen dienen dazu, die vorher beschriebenen Probleme zu umgehen. Eine genau Beschreibung der Abänderungen und weitere Modifikationen des Algorithmus findet man in [Sch91c].

Die 'Off-line' Version

Das Problem des vorherigen Abschnitts kann vermieden werden, falls man erst N_1 trainiert und dann N_2 und nun wieder N_1 und so weiter. Das Netz N_2 erhält bei festgehaltenen N_1 immer dieselben Eingabesequenzen im Gegensatz zu der Version, in der N_1 und N_2 parallel lernen.

Die Frage, wann das Trainieren von N_1 bzw. N_2 abgebrochen werden kann, ist jedoch schwierig, da man nie weiß, ob die Netze noch hinzulernen. Oder man stoppt das Lernen von N_1, obwohl bestimmte Teilsequenzen nur in manchen Fällen reduziert werden und in anderen Fällen nicht. So könnte N_2 verschieden reduzierte Eingabesequenzen erhalten, d. h. eine reduzierbare Teilsequenz ist in einer Eingabesequenz reduziert und in einer anderen Eingabesequenz nicht, was zu längerer Lernphase von N_2 führt, da nicht generalisiert werden kann.

Räumliche Selektion

Die Eingabe und Vorhersage bei N_2 kann sich auf die Komponenten des Eingabevektors beschränken, welche durch N_1 nicht vorhergesagt werden können. Hier ist gemeint, daß N_2 nur von den Ausgabeknoten Fehler bekommt, die durch N_1 nicht richtig vorhergesagt worden sind. Bei der Eingabe für N_2 sollten die durch N_1 korrekt vorhergesagten Eingaben einem festen konstanten Wert entsprechen (z. B. 0). Es wäre auch sinnvoll in einer Umgebung dieses konstanten Wertes keine Eingabewerte zuzulassen, um für N_2 deutlich zu machen, daß diese Eingabe keine Information enthält.

Wie man sich leicht überlegt, benötigt man für N_2 nur die von N_1 nicht vorhergesagten Eingaben, denn die übrigen Eingaben sind von den vorherigen Eingaben kausal abhängig und können mittels einer entsprecheden Funktion berechnet werden.

Zur Erleichterung der Lösung der Aufgabe für N_2 könnte N_2 jedesmal den gesamten Eingabevektor erhalten.

Andere Zeitrepräsentation

Das Netz N_2 kann auch die aktuelle Aktivierung von N_1 als Eingabe bekommen, statt der Zeitrepräsentation, da N_1 den aktuellen Zustand der Umgebung in seinen Knoten gespeichert hat. Hier ist nicht die absolute Zeit entscheidend, sondern die Situation, in der sich N_1 gerade befindet. Dies ist dann evtl. mehr problemorientiert, als die Eingabe der absoluten Zeit, doch ist nicht gesichert, daß der Zustand von N_1 eindeutig ist, d. h. daß bei einem bestimmten Zustand von N_1 die vorherige Teilsequenz eindeutig ist.

Modifikation der Vorhersage der Zeitüberbrückerknoten

Um zu verhindern, daß N_1 beim Lernen von N_2 zu viele Knoten vorhersagen muß, ist es evtl. besser bei der Architektur von N_2 einen 'Flaschenhals' zu erzeugen, d. h. es existiert eine minimale Anzahl von Knoten, über die alle Information zu den Ausgabeeinheiten weitergeleitet wird. Die Ausgabeknoten werden nur durch diese Knoten aktiviert, also müssen diese Knoten die gesamte Information tragen. Nun reicht es, wenn N_1 nur die Aktivierung dieser Knoten vorhersagt, womit die Information kompakter gelernt wird.

Der Zeitüberbrücker als Experte

Bis jetzt war der Zeitüberbrücker Experte für große Zeitabstände, doch man kann N_2 auch verwenden, um kompliziertere Eingabevorhersagen schneller zu lernen. Hierzu bekommt N_2 vor jeder vorherzusagenden Eingabe e_i, die also von N_1 nicht vorhergesagt wurde, die direkt in der ursprünglichen Eingabesequenz vorangehende Eingabe e_{i-1}. Man kann auch mehrere vorherige Eingaben $e_{i-l}, e_{i-l+1}, \ldots, e_{i-1}$ als Eingabe für N_2 liefern, doch N_2 erhält nur einen Fehler bei der Vorhersage von e_i. Somit konzentriert sich N_2 auf die schwierig vorherzusagenden Eingaben und lernt diese evtl. vorhersagen. Dann wird das Wissen, wie man diese Eingaben berechnet und vorhersagt auf N_1 übertragen und N_1 wird lernen, auch die komplizierteren Eingaben vorherzusagen. Jetzt kann sich N_2 auf noch komplizierter zu berechnende Eingaben konzentrieren und versuchen, diese vorherzusehen.

Kapitel 4

Konstanter Fehlerrückfluß

Eine andere Möglichkeit, um zu verhindern, daß der Fehler beim Zurückpropagieren durch die Zeit zu klein wird, ist den Fehlerrückfluß konstant zu halten. Entsteht der Fehler $e_i(t)$ zum Zeitpunkt t an dem Ausgabenknoten i und wird dieser zu dem Eingabeknoten k zum Zeitpunkt $t - l$ zurückgereicht, so soll der Fehler, der dort ankommt, genau so groß sein, wie der Fehler, der an dem Eingabeknoten k zum Zeitpunkt $t - l - 1$ ankommt. Somit werden für das Zustandekommen eines Fehlers an den Ausgabeknoten alle vergangenen Eingaben gleich bewertet und nicht, wie im herkömmlichen Backpropagation, die letzten Eingaben viel stärker gewichtet. Dies ist bei Aufgaben, bei denen vergangene Eingaben Auswirkungen auf die gewünschte Ausgabe haben, sehr hilfreich und führt zu starker Konvergenzbeschleunigung. Im Unterschied zum Zeitüberbrücker-System muß hier die Eingabesequenz nicht reduzierbar sein, d. h. es braucht keine Struktur in der Eingabesequenz vorhanden sein. Dieses Verfahren ist sinnvoll, falls abstraktere Eingaben vorliegen, die keine offensichtliche Struktur besitzen, wie z. B. Namen von Unterprogrammen, die ausgeführt wurden, wichtige Situationen, die in der Vergangenheit vorhanden waren oder Zusammenfassung von vergangenen Zuständen der Umgebung.

Wie schon in Abschnitt 2.2 gezeigt wurde, kann es zur Auslöschung eines Fehlers kommen, wenn der Fehler auf unterschiedlichen Wegen in die Vergangenheit fließt, da Fehlerrückfluß über negative Gewichte zu einer Vorzeichenänderung des zurückfließenden Fehlers führt. Deshalb wird der Rückfluß des Fehlers nur über eine einzige Einheit durchgeführt. Diese Einheit baut eine Brücke in die Vergangenheit durch ein starkes positives, rekurrentes Gewicht auf sich, das während des Lernens fixiert ist. Bei verschieden BP-Lernaufgaben mit rekurrenten Einheiten und einer Abhängigkeit der Sollausgabe von vergangenen Zuständen kann beobachtet werden, daß während des Lernvorgangs des Netzes erst eine Brücke in die Vergangenheit gebaut wird, indem positive rekurrente Verbindungen der Knoten auf sich geschafft werden. Dadurch kann jetzt ein betragsmäßig höherer Fehler in die Vergangenheit fließen, somit werden vergangene Zustände in größerem Maße berücksichtigt.

Eine solche Einheit, auch **konstanten Fehlerrückflußknoten (KFR-Knoten)** genannt, mit einem starken positiven rekurrenten Gewicht auf sich kann als Systeminvariante betrachtet werden, welche an- und ausgeschaltet werden kann und so Information über die Zeit hinweg rettet. Durch die Verbindungen von den Eingabeknoten zu dem KFR-Knoten wird der Einluß der Eingabe auf den KFR-Knoten gesteuert.

Mozer [Moz90] beschreibt ein verwandtes System, wobei der Fehler bei der rekurrenten Einheit i aber mit einem Faktor $\tau + (1 - \tau)y_i(1 - y_i)w_{ii}$ zurückfließt, was bedeutet, daß der KFR-Knoten mit diesem Faktor ab- bzw. zunimmt. Dieser Faktor ist kleiner 1 für $w_{ii} < 4.0$

und für $w_{ii} > 4.0$ kann sich ein Fehlerrückfluß größer oder kleiner 1 ergeben, je nach der Aktivierung y_i. Das Aufbauen eines Gewichts von 4.0 beansprucht aber einige Zeit. Bei Mozer ist die Abhängigkeit einer solchen Einheit i von der Eingabeeinheit j

$$\frac{\partial y_i}{\partial z_j} = (1-\tau)y_i(1-y_i)w_{ij}.$$

D. h. das An- bzw. Ausschalten eines solchen Knotens zieht nur eine kleine Aktivierungsänderung nach sich. Nach einiger Zeit ist der KFR-Knoten 'verschmiert', d. h. die Aktivierungen des an- und des ausgeschalteten KFR-Knotens differieren sehr wenig, da noch eine Änderung der Aktivierung durch spätere Eingaben hinzukommt. Die beiden extremsten neuen Aktivierungen von $y_i(t)$ zu einem Zeitpunkt t sind $a_1 = \tau y_i(t-1)$ und $a_2 = \tau y_i(t-1) + (1-\tau)$. Betrachtet man diese Variable zu einem späteren Zeitpunkt $t+u$, wobei b_1 die Aktivierung von $y_i(t+u)$ für $y_i(t) = a_1$ und b_2 die Aktivierung von $y_i(t+u)$ für $y_i(t) = a_2$ ist, so ist bei gleicher Eingabe nun $b_2 > b_1$, d. h. b_1 kann auf b_2 gesetzt werden, aber nicht umgekehrt. Wie dies zeigt, kann man mit diesem Verfahren den KFR-Knoten nicht beliebig aus- und einschalten. Mozer benützt sein Verfahren aber nur, um vergangene Eingaben in einem 'Context-Layer' zwischenzuspeichern und will wahrscheinlich nicht alle vergangenen Eingaben gleich bewerten. Er möchte vielmehr den Fehler verschieden stark in die Vergangenheit schicken.

Wie in Abschnitt 1.3 hergeleitet wurde, ist $\vartheta_j(t) = f_j'(s_j(t)) \sum_l \vartheta_l(t+1)w_{lj}$. Betrachtet man nur den k-ten Knoten, so ergibt sich

$$\vartheta_j(t) = f_j'(s_j(t))w_{kj}\vartheta_k(t+1).$$

Für einen konstanten Fehlerrückfluß vom k-ten zum j-ten Knoten muß also gelten

$$f_j'(z)w_{kj} = 1, \text{ d. h.}$$

$$f_j(z) = \frac{z}{w_{kj}} + c.$$

Nur eine lineare Einheit läßt einen konstanten Fehlerrücklauf auf sich selbst, bei festem Gewicht, zu. Deshalb kann man eine lineare Einheit mit einem Gewicht von 1.0 auf sich verwenden. Es sind hier aber Episodengrenzen notwendig, da sonst die Aktivierung des linearen Knotens zu stark anwächst und man keine geeigneten Werte für das An- bzw. Ausschalten des KFR-Knotens durch vergangene Eingaben findet. Es muß dann auch nur bis zu den Episodengrenzen zurückpropagiert werden.

Hat man ein Netz mit einem KFR-Knoten, so sollte erst das Netz trainiert werden, wobei die Gewichte zu dem KFR-Knoten nicht geändert werden, da ansonsten dieser Knoten auf einen konstanten Wert, z. B. 1.0, gesetzt wird oder Aufgaben übernimmt, die auch andere versteckte Knoten lösen können. In diesem Falle ist es schwer, die Verbindungen so abzuändern, daß dieser Knoten wieder frei zur Verfügung steht. Das Netz wird trainiert, ohne die Verbindungen zum KFR-Knoten zu ändern, wobei nur u Schritte in die Vergangenheit propagiert wird, bis keine Leistungsverbesserung mehr festzustellen ist. Nun wird mit dem KFR-Knoten die Vergangenheit abgetastet, indem man den Fehler, der an diesem Knoten ankommt, in die Vergangenheit weiterreicht, hier werden nur die Verbindungen, die zu diesem Knoten führen oder von ihm ausgehen, verändert. Außerdem wird nur ein Fehler in die Vergangenheit geschickt, wenn der maximale Fehler an einem der Ausgabeknoten eine feste Schranke ε überschreitet, denn in diesen Fällen könnte die gewünschte Ausgabe von vergangenen Zuständen abhängen. Findet nun

auch hier keine Leistungsverbesserung mehr statt, so wird wieder wie oben das Netz trainiert, ohne die Verbindungen zum KFR-Knoten zu ändern. Im Gegensatz zu oben kann hier dem Netz Information aus der Vergangenheit zur Verfügung stehen, welche im KFR-Knoten und dadurch evtl. auch in den anderen versteckten Knoten gespeichert ist.

Da der KFR-Knoten Verbindungen zu den versteckten Knoten hat und diese wiederum Verbindungen zum KFR-Knoten haben, kann der KFR-Knoten auch von mehreren Eingaben in der Vergangenheit beinflußt sein. Diese Eingaben müssen einen zeitlichen Abstand haben, der kleiner als u ist, da bei den normalen versteckten Knoten nur u Schritte in die Vergangenheit propagiert wird. In diesem Fall muß nun aber öfters zwischen Lernen des KFR-Knotens und Lernen des übrigen Netzes hin- und hergeschaltet werden, da die versteckten Knoten das Abspeichern von Information für den KFR-Knoten lernen müssen.

Ist zwischen den relevanten Eingaben für den KFR-Knoten ein zeitlicher Abstand größer als u oder gibt es mehrere Systeminvarianten, so wird der gerade trainierte KFR-Knoten abgesichert, indem man verbietet, daß sich die Verbindungen zum KFR-Knoten ändern. Wird nun der aktuelle KFR-Knoten wie ein versteckter Knoten betrachtet, so kann ein neuer KFR-Knoten hinzugefügt werden. Ergibt das Hinzunehmen einer neuen Systeminvarianten keine Leistungsverbesserung, so ist keine neue Systeminvariante vorhanden oder es war zu schwer diese zu finden.

Auch hier kann Schmidhubers Idee der räumlichen Trennung, siehe hierzu [Sch91c], greifen, indem man evtl. nur die Fehler e_i in die Vergangenheit schickt, die eine feste obere Grenze überschritten haben. Somit wird der KFR-Knoten speziell für diese Fehler trainiert, dadurch könnten aber andere Fehler vergrößert werden. Diese werden dann beim Lernen des gesamten Netzes wieder erniedrigt.

Im Idealfall sollte der KFR-Knoten einen hohen und einen niedrigen Wert haben, welcher von den vergangenen Eingaben abhängig ist und relativ konstant bleibt. Außerdem sollte das rapide Anwachsen einer linearen Einheit evtl. vermieden werden, um den KFR-Knoten besser steuern zu können. Dies kann durch eine logistische Einheit erreicht werden. Im folgenden wird nur das Gewicht w einer logischtischen Einheit auf sich und die negative Verbindung b zum Eins-Knoten betrachtet. Außerdem sollte die Aktivierungsfunktion für diese Einheit 2 Attraktoren $0 < x_1 < 0.5$ und $1.0 > x_2 > 0.5$ haben und einen konstanten Fehler zurückschicken, falls die Aktivierung dieser Einheit gleich x_1 oder gleich x_2 ist. Der Wert x ist ein **Attraktor** der Funktion f, falls eine Umgebung $U(x)$ existiert, so daß

$$z \in U(x) \Rightarrow f^n(z) \rightarrow x, \text{ für } n \rightarrow \infty$$

gilt. Es wird die Funktion $\tilde{f}(z) = \frac{1}{1+\exp(-wz+b)}$ betrachtet, da nur die rekurrente Verbindung w und die Verbindung b vom Eins-Knoten relevant sein sollen, denn alle anderen Netzeingaben sollten klein sein, wenn der KFR-Knoten nicht aus- oder angeschaltet werden soll. Es muß also gelten:

$$1 = w(1 - x_1)x_1 = w(1 - x_2)x_2$$

hieraus folgt

$$x_2 = 1 - x_1 \text{ weiter sei } x := x_1.$$

Für die Fixpunkte a der Funktion $\tilde{f}(z)$ ergibt sich

$$a = \frac{1}{1 + \exp(-wa + b)} \Rightarrow \frac{1}{a} - 1 = \exp(-wa + b) \Rightarrow \ln(1 - a) - \ln(a) = -wa + b$$

$$\Rightarrow w = \frac{\ln(a) - \ln(1-a) + b}{a};\ a \in \{x, (1-x)\}.$$

Also gilt

$$\frac{\ln(x) - \ln(1-x) + b}{x} = \frac{\ln(1-x) - \ln(x) + b}{1-x}$$

$$\Rightarrow 0 = b\left(\frac{1}{1-x} - \frac{1}{x}\right) + \frac{\ln(1-x) - \ln(x)}{1-x} - \frac{\ln(x) - \ln(1-x)}{x} =$$

$$b\frac{x-1+x}{(1-x)x} + \frac{x\ln(1-x) - x\ln(x) - (1-x)\ln(x) + (1-x)\ln(1-x)}{(1-x)x}$$

$$\Rightarrow b = \frac{\ln(x) - \ln(1-x)}{2x-1}.$$

Für w ergibt sich deshalb

$$w = \frac{\ln(x) - \ln(1-x) + \frac{\ln(x)-\ln(1-x)}{2x-1}}{x} =$$

$$= \frac{2x\ln(x) - 2x\ln(1-) - \ln(x) + \ln(1-x) + \ln(x) - \ln(1-x)}{(2x-1)x} =$$

$$= \frac{2\left(\ln(x) - \ln(1-x)\right)}{2x-1},\ \text{also } w = 2b,$$

was man auch für $a = 1-x$ erhält. Die Konvergenz und der Einzugsbereich der Fixpunkte wird nun im folgenden berechnet. Sei $\tilde{f}(z) = \frac{1}{1+\exp(-wz+b)}$, so gilt $\tilde{f}(x) = x$ und $\tilde{f}(1-x) = 1-x$ für obiges w und b. Es ist nun $\tilde{f}'(x) = w\tilde{f}(x)(1-\tilde{f}(x)) = wx(1-x)$ und $\tilde{f}'(1-x) = w\tilde{f}(1-x)(1-\tilde{f}(1-x)) = wx(1-x)$, für die Konvergenz muß nun gelten $wx(1-x) < 1$.

$$x(1-x)\frac{2\left(\ln(x) - \ln(1-x)\right)}{2x-1} < 1$$

$$\Leftrightarrow \ln\left(\frac{x}{1-x}\right) > \frac{2x-1}{2x(1-x)} = \frac{1}{2}\left(\frac{1}{1-x} - \frac{1}{x}\right) =$$

$$= \frac{1}{2}\left(\frac{1}{1-x} - \frac{1-x}{1-x} - \frac{1}{x} + \frac{x}{x}\right) = \frac{1}{2}\left(\frac{x}{1-x} - \frac{1-x}{x}\right).$$

Sei weiter $t := \ln(\frac{x}{1-x})$, so ergibt sich folgende Gleichung:

$$t > \frac{1}{2}\left(\exp(t) - \exp(-t)\right) = \sinh(t),$$

sinh hat eine Tangente in 0 mit Steigung 1 und für $t > 0$ ist die Steigung von sinh größer 1 und für $t < 0$ ist die Steigung von sinh auch größer 1, also muß gelten:

$$t < 0 \Leftrightarrow \frac{x}{1-x} < 1 \Leftrightarrow x < \frac{1}{2}$$

Für $x < \frac{1}{2}$ sind also x und $1-x$ anziehende Fixpunkte, d. h. $\tilde{f}$ konvergiert gegen x in einer Umgebung von x und $\tilde{f}$ konvergiert gegen $1-x$ in einer Umgebung von $1-x$. Nun zum Einzugsbereich der Fixpunkte

$$\tilde{f}(x_1) < \frac{1}{2} \Leftrightarrow \frac{1}{1+\exp(-2bx_1+b)} < \frac{1}{2} \Leftrightarrow$$

$$\exp(-2bx_1 - b) > 1 \quad (\ln \text{ ist monoton wachsend}) \Leftrightarrow -2bx_1 + b > 0$$
$$\Leftrightarrow x_1 < \frac{1}{2}, \text{ ebenso folgt } \tilde{f}(x_1) > \frac{1}{2} \Leftrightarrow x_1 > \frac{1}{2}.$$

Als Ergebnis hat man nun:

$$x_1 < \frac{1}{2} : \tilde{f}^n(x_1) \to x \text{ und}$$

$$x_1 > \frac{1}{2} : \tilde{f}^n(x_1) \to 1 - x \text{ für}$$

$$\tilde{f}(z) = \frac{1}{1 + \exp(-b(2z - 1))} \text{ mit}$$

$$b = \frac{\ln(x) - \ln(1 - x)}{2x - 1}.$$

In Abbildung 4.1 sind für $b = 2.5$ und $b = 3.0$ die Kurven dargestellt, wobei auch die Kurve $y(x) = x$ eingezeichnet ist.
Für konstanten Fehlerrückfluß sollte $wx(1 - x) = 1$ gelten, dies geht wegen obiger Konvergenzbedingung $wx(1 - x) < 1$ nur für $x = \frac{1}{2}$, was $w = 4$ und $b = 2$ liefert.
(Mit l' Hospital erhält man dies auch wie folgt:

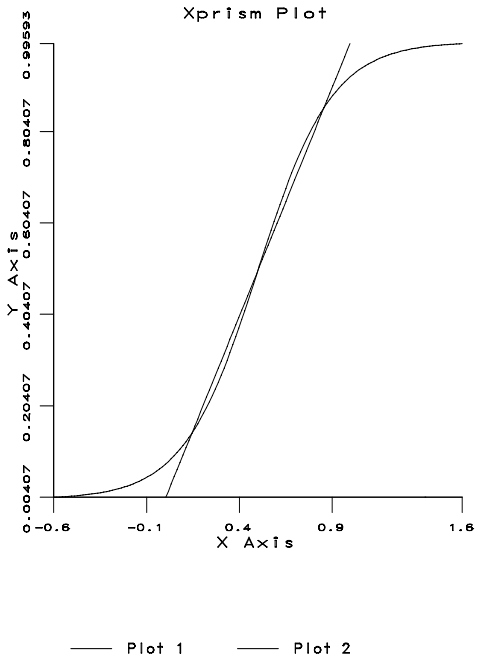
$$w = \lim_{x \to \frac{1}{2}} 2\frac{\ln(\frac{x}{1-x})}{2x - 1} =$$

$$\lim_{x \to \frac{1}{2}} 2\frac{\frac{1}{\frac{x}{1-x}}\frac{1}{(1-x)^2}}{2} = \lim_{x \to \frac{1}{2}} \frac{1}{x(1 - x)} = 4 \Rightarrow b = 2.$$

)

Für $w = -2b$ und $b = \frac{\ln(x) - \ln(1-x)}{2x-1}$ ergibt sich ein Oszillatorknoten, d. h. $\tilde{f}(x) = 1 - x$ und $\tilde{f}(1 - x) = x$. Zur Speicherung kann auch ein solcher Oszillatorknoten verwendet werden, denn wird seine Aktivierung zum Zeitpunkt t auf x oder $1-x$ gesetzt und nachfolgend nicht erheblich gestört, so ist später zu jedem Zeitpunkt erkennbar, ob zum Zeitpunkt t die Aktivierung auf x oder $1 - x$ gesetzt wurde.

Man sieht deutlich, daß das Vorhandensein von Attraktoren und ein konstanter Fehlerrückfluß sich widersprechende Ziele sind. Wählt man feste Attraktoren $x, (1 - x)$, so kann es zu Vergrößerung des Fehlers kommen, falls die Aktivierung der Einheit zwischen x und $(1 - x)$ liegt. Liegt andererseits die Aktivierung der Einheit zwischen 0 und x oder zwischen $(1-x)$ und 1, so vermindert sich der zurückgegebene Fehler. Trotzdem kann man bei Problemen, bei denen man zur Lösung der Aufgabe nicht zu weit in die Vergangenheit schauen muß, Attraktoren fest wählen. Man kann diese Methode evtl. noch verbessern, falls man alle Fehler mit dem Faktor $\gamma = \frac{\delta_i(t)}{\max_{1 \le l \le p-1} |\delta_i(t-l)|}$ multipliziert. Dadurch erhält man eine Vergrößerung bzw. Verkleinerung aller Fehler, was einer gesteuerten Lernrate gleichkommt.



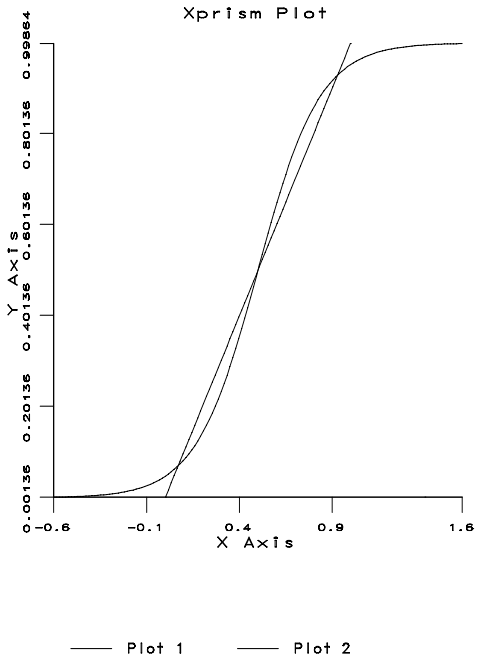


Abbildung 4.1: *Hier sind die Kurven* $\tilde{f}(x) = \frac{1}{1+\exp(-b(2x-1))}$ *und* $y(x) = x$ *dargestellt. Für die Kurve oben wurde* $b = 2.5$ *und für die Kurve unten wurde* $b = 3.0$ *gewählt. Die Schnittpunkte der Kurven mit der Winkelhalbierenden sind die Attraktoren der Kurven, außer der Schnittpunkt bei* $x = 0.5$*, welcher ein abstoßender Fixpunkt ist.*

4.1 Variable Gewichte

Einen konstanten Fehlerrückfluß erhält man, falls man $w_{kk}(t) = \frac{1}{f_k'(s_k(t))}$ setzt. Hier erhält man nun folgende Ableitung dieses Gewichts nach den anderen Gewichten:

$$\frac{\partial w_{kk}(t)}{\partial w_{ij}} = -\frac{f_k''(s_k(t))}{(f_k'(s_k(t)))^2}\left(\sum_{l=m+1,l\neq k}^{m+n}\frac{\partial y_{l-m}(t)}{\partial w_{ij}}w_{kl} + \delta_{ik}z_j(t) + \frac{\partial w_{kk}(t)}{\partial w_{ij}}y_{k-m}(t)\right)$$

$$\Rightarrow \frac{\partial w_{kk}(t)}{\partial w_{ij}}\left(1 + \frac{f_k''(s_k(t))}{(f_k'(s_k(t)))^2}y_{k-m}(t)\right) = -\frac{f_k''(s_k(t))}{(f_k'(s_k(t)))^2}\left(\sum_{l=m+1,l\neq k}^{m+n}\frac{\partial y_{l-m}(t)}{\partial w_{ij}}w_{kl} + \delta_{ik}z_j(t)\right)$$

$$\Rightarrow \frac{\partial w_{kk}(t)}{\partial w_{ij}} = -\frac{f_k''(s_k(t))}{\left(f_k'(s_k(t))\right)^2 + f_k''(s_k(t))y_{k-m}(t)}\left(\sum_{l=m+1,l\neq k}^{m+n}\frac{\partial y_{l-m}(t)}{\partial w_{ij}}w_{kl} + \delta_{ik}z_j(t)\right).$$

Wegen

$$\frac{\partial y_{k-m}(t+1)}{\partial w_{ij}} = f_k'(s_k(t))\left(\sum_{l=m+1,l\neq k}^{m+n}\frac{\partial y_{l-m}(t)}{\partial w_{ij}}w_{kl} + \delta_{ik}z_j(t) + \frac{\partial w_{kk}(t)}{\partial w_{ij}}y_{k-m}(t)\right) \text{ gilt}$$

$$\frac{\partial y_{k-m}(t+1)}{\partial w_{ij}} = f_k'(s_k(t))\left[1 - \frac{f_k''(s_k(t))y_{k-m}(t)}{\left(f_k'(s_k(t))\right)^2 + f_k''(s_k(t))y_{k-m}(t)}\right] \cdot \left(\sum_{l=m+1,l\neq k}^{m+n}\frac{\partial y_{l-m}(t)}{\partial w_{ij}}w_{kl} + \delta_{ik}z_j(t)\right).$$

Berücksichtigt man den Term $\frac{f_k''(s_k(t))y_{k-m}(t)}{\left(f_k'(s_k(t))\right)^2+f_k''(s_k(t))y_{k-m}(t)}$ nicht, so ergibt sich ein relativer Verfahrensfehler dieser Größe. Für lineare Funktionen ist dieser Term gleich 0, will man aber Attraktoren haben, so kann keine lineare Funktion gewählt werden. Bei nichtlinearen Funktionen sollte man welche mit großer Steigung und kleiner Krümmung wählen, um den Verfahrensfehler klein zu halten. Man kann evtl. um 0 eine lineare Funktion und ab einer bestimmten Grenze eine Funktion mit größerer Steigung wählen.

Für Attraktoren hat man die Gleichung $x = f\left(\frac{x}{f'(x)}\right)$. Sei nun $f(x) = kx^n$, deshalb gilt $f'(x) = knx^{n-1}$, somit erhält man

$$f\left(\frac{x}{f'(x)}\right) = k\left(\frac{1}{knx^{n-2}}\right)^n = \frac{1}{k^{n-1}n^n x^{n(n-2)}}.$$

Also gilt für den Attraktor x_a:

$$k^{n-1}n^n x_a^{n^2-2n+1} = 1 \Leftrightarrow nk^{n-1}n^{n-1}x_a^{(n-1)^2} = 1 \Leftrightarrow n\left(knx_a^{n-1}\right)^{n-1} = 1$$

$$\Leftrightarrow knx_a^{n-1} = \frac{1}{\sqrt[n-1]{n}} \Leftrightarrow x_a^{n-1} = \frac{1}{kn\sqrt[n-1]{n}} \Leftrightarrow x_a = \frac{1}{\sqrt[n-1]{kn\sqrt[n-1]{n}}}.$$

Man wählt im Intervall $[-x_a, x_a]$ eine lineare Funktion und außerhalb dieses Intervalls die Funktion $f(x) = kx^n$, für welche man folgenden Verfahrensfehler hat:

$$\frac{f''(x)y}{(f'(x))^2 + f''(x)y} = \frac{1}{\frac{(f'(x))^2}{f''(x)y} + 1} =$$

$$\frac{1}{\frac{k^2n^2x^{2n-2}}{kn(n-1)x^{n-2}y} + 1} = \frac{1}{\frac{kn}{y(n-1)}x^n + 1} < \varepsilon,$$

falls der relative Verfahrensfehler kleiner ε sein soll. Betrachtet man nur den Fall für positives x und y (der negative Fall ist symmetrisch), so ist $x = x_a$ das minimale x, für das diese Gleichung gilt. Nimmt man eine nicht zu starke Änderung in der Aktivierung an, so kann man $y = x = x_a$ setzen, womit man nun folgende Gleichungen erhält:

$$\frac{1}{\frac{kn}{(n-1)}x_a^{n-1} + 1} < \varepsilon \Leftrightarrow \frac{1}{\frac{kn}{n-1}\frac{1}{kn\sqrt[n-1]{n}} + 1} < \varepsilon,$$

wegen $(n-1)\sqrt[n-1]{n} > -1$, für $n < 1$, ist dies äquivalent zu

$$1 < \varepsilon + \varepsilon\frac{1}{(n-1)\sqrt[n-1]{n}} \Leftrightarrow \frac{1-\varepsilon}{\varepsilon} < \frac{1}{(n-1)\sqrt[n-1]{n}}$$

$$\Leftrightarrow (n-1)\sqrt[n-1]{n} < \frac{\varepsilon}{1-\varepsilon}.$$

Es muß also $n > 1$ gewählt werden und je näher n an 1 liegt, desto kleiner ist der Verfahrensfehler.

Wählt man logistische Aktivierungsfunktionen, so hat man $w = \frac{1}{y(1-y)}$, und man erhält einen konstanten Fehlerrückfluß, doch man muß auf Attraktoren verzichten. Denn setzt man $b = -\frac{1}{2}w$, so verschieben sich die Attraktoren laufend in Richtung 1 bzw. 0 und w wächst schnell und stark an.

Setzt man hingegen $b = \ln(\frac{1-c}{c}) + wc$, falls der KFR-Knoten eine Aktivierung kleiner als $\frac{1}{2}$ hat und $b = \ln(\frac{c}{1-c}) + w(1-c)$ bei einer Aktivierung größer als $\frac{1}{2}$, so erhält man 2 'Halbattraktoren' x und $1-x$, d. h. $f^n(z)$ konvergiert gegen x nur für $x \leq z < \frac{1}{2}$ und $f^n(z)$ konvergiert gegen $1-x$ nur für $\frac{1}{2} < z \leq 1-x$. Doch außerhalb dieses Bereichs ergibt sich Divergenz. Man sollte in diesem Fall versuchen, bei Episodengrenzen den KFR-Knoten auf $\frac{1}{2}$ zu setzen und bei kleinen Gewichten wird, je nach Eingabe, eine der beiden 'Halbattraktoren' angestrebt.

4.2 Linearer KFR-Knoten

Nun zur linearen Aktivierungsfunktion $f(x) = kx$, wobei der KFR-Knoten ein Gewicht von $\frac{1}{k}$ auf sich hat. Sei $x(t) = x^1 + S(t)$ die entscheidende Information zum Zeitpunkt t, welche im KFR-Knoten gespeichert werden soll, um später eine gewünschte Ausgabe zu erzeugen. Wobei x^1 die Information enthält und $S(t)$ eine Störung bedeutet, d. h. $S(t)$ beinhaltet Information, welche nicht in dem KFR-Knoten gespeichert werden muß. Weiter sei $x(t+l) = S(t+l)$ die

Netzeingabe zu späteren Zeitpunkten, wobei S die Störung des KFR-Knotens durch die Eingabe ist, somit ergibt sich wegen $y(u+1) = k\left(\frac{1}{k}y(u) + S(u)\right) = y(l) + kS(l)$

$$y(t+v) = kx^1 + y(t-1) + k\sum_{l=0}^{v-1} S(t+l).$$

Also ist das Verhältnis von Störung des KFR-Knotens durch die Netzeingabe und der im KFR-Knoten abgespeicherten Eingabe unabhängig von der Steigung der Aktivierungsfunktion des KFR-Knotens. Die Steigung skaliert nur die Aktivierung, d. h. die lineare Einheit hat bei kleiner Steigung eine nicht so starke Aktivierung.

Um die Störungen $S(t+l)$ des KFR-Knotens durch die für die Speicherung der Information in dem KFR-Knoten unwichtigen Eingaben zu vermindern, ist es besser, wenn der Fehler nicht konstant zu den früheren Eingaben fließt, sondern gewichtet wird. Hierzu kann man einige Knoten verwenden, welche die Zeit zwischen den Grenzen der Eingabesequenz repräsentieren, oder man verwendet die Eingabeknoten und die versteckten Knoten des Netzes. Von diesen Knoten, welche die Zeit oder den Zustand des Netzes festlegen, führen Verbindungen zu einer Gewichtungseinheit. Die Aktivierung dieser Einheit wird zu jedem Zeitpunkt mit der Netzeingabe des KFR-Knotens multipliziert, wodurch man die neue Netzeingabe des KFR-Knotens erhält, somit hat man eine Gewichtung der Netzeingabe des KFR-Knotens durch die Zeit bzw. durch den Zustand des Netzes. Die resultierende Einheit, d. h. der KFR-Knoten und die Gewichtungseinheit, ist ähnlich einer Sigma-Pi Einheit aus [RHW86] auf Seite 353, wobei hier der Teil der Netzeingabe, welcher durch die Einheit selbst erzeugt wird, nicht mit der Gewichtungseinheit multipliziert wird, da man einen konstanten Fehlerrückfluß haben will.

Ist der Gewichtungsknoten y_g ein logistischer Knoten, so kann man als Gewichtung $2y_g$ verwenden. Die richtige Gewichtung wird gelernt, indem man über y_g zu den Einheiten propagiert, welche die Zeit oder den Zustand des Netzes repräsentieren, und so die Gewichte von diesen Knoten zu der Gewichtungseinheit y_g entsprechend ändert.

Diese Gewichtung ist auch vorteilhaft, wenn über den KFR-Knoten verschiedene Fehler zurückfließen, dann kann sich der größte Fehler durchsetzen und die Einheit wird verwendet, um diesen Fehler zu minimieren.

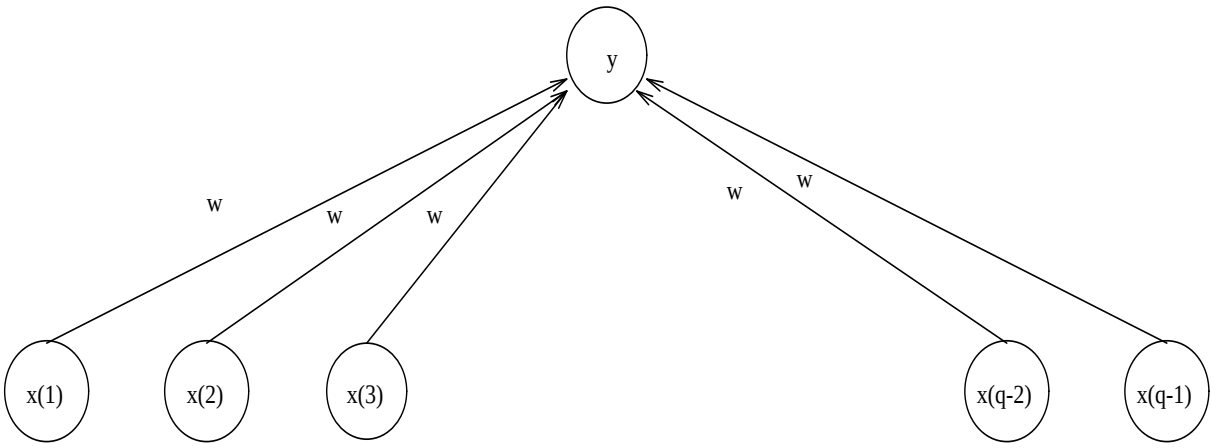
Ist die Steigung $k = 1$ und setzt man die KFR-Einheit zu Beginn jeder Eingabesequenz explizit auf 0, so hat der KFR-Knoten zum Zeitpunkt q die Aktivierung

$$y(q) = \sum_{l=1}^{q-1} e_l = \sum_{l=1}^{q-1} x(l).$$

Man kann also die Ersatznetzstruktur für den KFR-Knoten von Abb. 4.2 verwenden. In dieser Ersatznetzstruktur liegen an der KFR-Einheit zum Zeitpunkt q die letzten $q-1$ Netzeingaben an, wobei alle Eingaben dieselben Gewichte zum KFR-Knoten besitzen.

4.3 Experimente zum konstanten Fehlerrückfluß

Es wurden Versuche durchgeführt mit G_1^{100} und G_2^{100} aus Abschnitt 3.2.1, wobei das Netz die nächste Eingabe und nach Anliegen von b_{100} zusätzlich die gewünschte Ausgabe vorhersagen mußte. Tabelle 4.1 zeigt die Ergebnisse ohne Zeitrepräsentation und Tabelle 4.2 zeigt die Ergebnisse mit Zeitrepräsentation. Für die Zeitrepräsentation wurden 11 Knoten verwendet,



Ersatzschaltbild fuer einen KFR-Knoten y, wo die letzten (q-1) Eingaben anliegen. Die Knoten unten symbolisieren

einen Eingabevektor und w ist der Gewichtsvektor von dem Eingabevektor zum KFR-Knoten.

Abbildung 4.2: *Für einen linearen KFR-Knoten y mit Steigung 1.0 kann das hier gezeigte Ersatzschaltbild verwendet werden. Es liegen an dem KFR-Knoten die letzten* $(q-1)$ *Eingaben zum Zeitpunkt q an, falls die versteckten Knoten nicht berücksichtigt werden. Die Gewichte von den einzelnen Eingaben zum KFR-Knoten sind immer dieselben. Propagiert man in diesem Ersatzschaltbild, so muß man für die Gewichtsänderung den Durchschnitt über die letzten* $(q-1)$ *Eingaben bilden.*

Versuchnr.	angel. Eingabeseq.	max. Fehler
1.	5 000	0.1
2.	5 600	0.2
3.	5 400	0.2
4.	4 000	0.3
5.	5 200	0.2

Tabelle 4.1: *Versuch mit konstantem Fehlerrückfluß ohne Zeitrepräsentation. Der maximale Fehler ist der Absolutbetrag des Fehlers für die Ausgabeeinheit für die gewünschte Ausgabe. Es ist die Anzahl der angelegten Eingabesequenzen angegeben.*

womit sich 10 Intervalle darstellen lassen. Ist $0 \leq q \leq 100$ im i-ten Intervall, d. h. es ist $(i-1)*10 \leq q \leq i*10$, so hat der Zeitknoten i eine Aktivierung von $\frac{q-(i-1)*10}{10}$ und der Zeitknoten $i+1$ hat eine Aktivierung von $\frac{i*10-q}{10}$, alle anderen Zeitknoten haben die Aktivierung null. Diese Repräsentation der Zeit wurde analog aus [Chr90] übernommen, wo Koordinaten in dieser Weise kodiert waren, um die Summe der Aktivierungen der Eingabeknoten auf dem Wert 1.0 zu halten.

Das verwendete Netz hatte keine versteckte Einheit, eine lineare KFR-Einheit mit Steigung 1.0, die 103 Eingabeeinheiten (incl. der Eins-Einheit) und 103 Ausgabeeinheiten (incl. der gewünschten Ausgabe). Es wurde die Lernrate $\alpha = 1.0$ verwendet und nach 400 angelegten Eingabesequenzen wurde nur die KFR-Einheit trainiert mit $\alpha = 0.01$, d. h. es wurden nur Verbindungen, die zum KFR-Knoten führen und Verbindungen, die von ihm wegführen, verändert. Nach 4 000 Zyklen wurde wieder, wie zu Beginn, das gesamte Netz trainiert, ohne die Verbindungen zum KFR-Knoten zu ändern, mit einer Lernrate von $\alpha = 1.0$. Die KFR-Einheit wurde zu Episodenanfang, d. h. zu Anfang jeder Eingabesequenz, auf 0 gesetzt, um vorher gespeicherte Information zu löschen; würde man die gewünschte Ausgabe bei der nächsten Eingabe wieder zurückfüttern, wie in Abschnitt 3.3, so wäre explizites Zurücksetzen nicht notwendig.

Da von der Struktur in der Eingabesequenz kein Gebrauch gemacht wurde, wurden für zusätzliche Tests Eingabesequenzen $E = (e_1, e_1, e_2, \ldots, e_{101})$ verwendet mit $e_1 \in \{x, a\}$ und $e_i \in \{b_1, b_2, \ldots, b_{100}\}$ für $2 \leq i \leq 101$. War bei einer Eingabesequenz $e_1 = a$, so sollte nach Abarbeitung von e_{101} der Wert 1.0 ausgegeben werden und für $e_1 = x$ sollte nach Anliegen von e_{101} der Wert 0 ausgegeben werden. Tabelle 4.3 zeigt die Ergebnisse für diesen Fall, wobei obige Zeitrepräsentation verwendet wurde.

4.4 Neuronaler Langzeitspeicher

Bis jetzt besteht die KFR-Einheit aus einem linearen Knoten $(i, f_i(x)) = (i, kx)$, mit dem Gewicht $w_{ii} = \frac{1}{k}$ auf sich, und einem Gewichtungsknoten y_g, welcher die Netzeingabe für den Knoten i in Abhängigkeit des Netzzustandes oder der Zeitrepräsentation gewichtet.

Man könnte nun statt eines linearen Knotens mehrere lineare Knoten verwenden, bei denen die Netzeingabe allesamt durch y_g gewichtet wird. Damit wird erreicht, daß komplexere Information gespeichert werden kann und man erhält eine Art 'Speicherzelle'.

Während der KFR-Knoten lernt, müssen alle von diesem Knoten ausgehenden Gewichte exakt berechnet werden, z. B. muß oft ein Wert von 0 erreicht werden, da der KFR-Knoten erst

Versuchnr.	angel. Eingabesequ.	max. Fehler
1.	4 200	0.12
2.	4 200	0.28
3.	4 200	0.39
4.	4 600	0.37
5.	4 200	0.16
6.	4 400	0.26
7.	4 800	0.30
8.	4 600	0.21
9.	4 800	0.28
10.	4 200	0.18
11.	4 200	0.12
12.	4 200	0.05
13.	4 800	n.g.
14.	4 200	0.09
15.	4 800	0.34
16.	4 200	0.08
17.	4 200	0.22
18.	4 800	0.21
19.	4 600	0.25
20.	4 200	0.12
21.	4 200	0.24
22.	4 200	0.16
23.	4 200	0.08
24.	4 400	0.23
25.	4 200	0.23
26.	4 200	0.27
27.	4 800	n.g.
28.	4 200	0.19
29.	4 400	0.27
30.	4 600	0.21

Tabelle 4.2: *Versuch mit konstantem Fehlerrückfluß mit Zeitrepräsentation. Der maximale Fehler ist der Absolutbetrag des Fehlers für die Ausgabeeinheit für die gewünschte Ausgabe. Es sind die angelegten Eingabesequenzen angegeben. Die Abkürzung 'n.g.' bedeutet, daß bei 4 800 angelegten Eingabesequenzen der maximale Fehler über 0.40 lag.*

Versuchnr.	angel. Eingabesequ.	max. Fehler
1.	4 200	0.07
2.	4 200	0.21
3.	4 200	0.27
4.	4 800	n.g.
5.	4 600	0.32
6.	4 200	0.06
7.	4 600	0.27
8.	4 200	0.06
9.	4 200	0.22
10.	4 400	0.07
11.	4 200	0.15
12.	4 200	0.13
13.	4 600	0.28
14.	4 600	0.26
15.	4 200	0.18
16.	4 200	0.18
17.	4 200	0.16
18.	4 600	0.15
19.	4 200	0.18
20.	4 200	0.12
21.	4 200	0.06
22.	4 200	0.10
23.	4 200	0.12
24.	4 200	0.25
25.	4 200	0.22
26.	4 200	0.11
27.	4 200	0.11
28.	4 200	0.06
29.	4 200	0.13
30.	4 400	0.26

Tabelle 4.3: *Versuch mit konstantem Fehlerrückfluß mit Zeitrepräsentation, aber ohne Struktur in der Eingabesequenz. Der maximale Fehler ist der Absolutbetrag des Fehlers für die Ausgabeeinheit für die gewünschte Ausgabe. Es sind die angelegten Eingabesequenzen angegeben. Die Abkürzung 'n.g.' bedeutet, daß bei 4 800 angelegten Eingabesequenzen der maximale Fehler über 0.40 lag.*

eine hohe und dann wieder eine geringe Aktivierung besitzt. Diese Aktivierungsdifferenz ist evtl. nur für eine einzige in der Zukunft liegende gewünschte Ausgabe o von Interesse, d. h. alle zeitlich vor dieser gewünschten Ausgabe o liegenden Ausgaben werden von dem KFR-Knoten nur gestört. Diese Störung könnte verkleinert werden, wenn man zu jedem Zeitpunkt eine zu y_g analoge Einheit y_a hat, die mit dem linearen Knoten multipliziert wird, um so die Ausgabeaktivierung zu erhalten. Der Knoten y_a hat Verbindungen von den Zeitrepräsentationsknoten oder von den Eingabeknoten und versteckten Knoten, somit wird die Netzausgabe der linearen Knoten gewichtet. Die linearen Knoten werden dann vor allem verwendet, den größten Fehler zu minimieren, da dieser Fehler die Gewichte von y_a am stärksten ändert und somit laufend stärker gewichtet wird.

Nun hat man ein 'Speichermodul' $(g, a, (i_1, \ldots, i_k))$ mit den linearen Knoten $i_1, \ldots, i_k$ und Netzeingabegewichtungsknoten g für diese linearen Knoten und Ausgabegewichtungsknoten a für diese linearen Knoten. Der Knoten g ist dafür verantwortlich, daß zur richtigen Zeit Information abgespeichert wird und a holt die gespeicherte Information zu einem bestimmten Zeitpunkt aus dem Speicher.

Falls die Gewichtungseinheit a bzw. g eine logistische Einheit ist, sollte für die Gewichtungsfunktion G gelten: $G(0) = 0$, $G(0.5) = 1$ und $G(1) = k$. Verwendet man für G eine Potenzfunktion, so gilt $G(x) = kx^z$, falls $z = \frac{\ln k}{\ln 2}$ ist. Somit erhält man eine Gewichtung von $G(y_{g/a}(t))$ zum Zeitpunkt t.

Kapitel 5

Problem der Ressourcenbeschaffung

Sowohl beim Zeitüberbrücker-System als auch beim konstanten Fehlerrückfluß werden die rekurrenten Knoten verwendet, die einfachste Aufgabe zu lösen. Dies ist bei G_1^{p-1} und G_2^{p-1} aus Abschnitt 3.2.1 der Fall, dort werden die rekurrenten Knoten auf einen Wert nahe an 1.0 bei logistischen Knoten und eine hohen negativen oder positiven Wert bei linearen Knoten gesetzt. Dies geschieht durch betragsmäßig hohe Gewichte vom Eins-Knoten und durch starkes positives Gewicht des rekurrenten Knotens auf sich. Der rekurrente Knoten fungiert dann wie der Eins-Knoten und drückt die Aktivierung für die Vorhersageeinheiten von b_i auf 0. Ist der rekurrente Knoten ein logistischer Knoten, so ergibt eine Aktivierung nahe bei 1.0 eine kleine Ableitung, d. h. die Verbindungen zu diesem Knoten ändern sich nur sehr langsam.

Tabelle 5.1 zeigt die Verbindung von der Eins-Einheit zu einem solchen rekurrenten Knoten beim Zeitüberbrücker-System, wobei die Lernrate $\alpha = 1.0$ verwendet wurde. Die einzelnen Gewichte wurden in Abständen von 1 000 angelegten Eingabesequenzen von G_1^{100} und G_2^{100} ausgegeben, wobei die erste Ausgabe nach wenigen tausend angelegten Eingabesequenzen erfolgte. Ansonsten war die Architektur des Zeitüberbrücker-Systems wie in Abschnitt 3.3.

Bei einer Änderung der Gewichte zu den rekurrenten Knoten hat dies Einfluß auf die Aktivierung von $p + 1$ Ausgabeeinheiten bei jedem Schritt. Der durch die Änderung der Gewichte zu einem rekurrenten Knoten erzeugte zusätzliche Fehler kann größer sein als der evtl. selten auftretende Fehler, für den die rekurrente Einheit benötigt wird.

Die rekurrente Einheit wird für Aufgaben ver(sch)wendet, die auch ohne diese Einheit gelöst werden können. Sollen beispielweise l Ausgabeknoten eine konstante Aktivierung von 1.0 haben und soll mit einer zusätzlichen Ausgabeeinheit eine andere Aufgabe gelöst werden, so werden die rekurrenten Knoten auf 1.0 gesetzt und dazu verwendet, die Aktivierung der l Ausgabeeinheiten auf 1.0 zu setzen.

Bei Aufgaben mit für die jetzige gewünschte Ausgabe wichtigen in der Vergangenheit liegenden Eingaben ist dieses Problem besonders gravierend, da erst Aufgaben gelöst werden, bei denen man nicht weit in die Vergangenheit sehen muß und hierzu alle verfügbaren Knoten verwendet werden. Dies erschwert zusätzlich das Lernen mit für die gewünschte Ausgabe relevanten zurückliegenden Eingaben.

Elman [Elm91] begegnet diesem Problem, indem er schrittweise versteckte Knoten zu seinem Netz hinzuaddiert. Auch Ash [Ash89] und Müller und Reinhardt [MR90] addieren eine versteckte Einheit hinzu, falls die Fehlerabnahme unter eine feste Grenze fällt. Yamashita, Hirose und Hijiya [YHH91] schlagen auch diese Methode vor, nur daß die letzte hinzugefügte Einheit entfernt wird, falls keine Leistungsverbesserung eingetreten ist. Fahlman und Lebiere [FL90]

angel. Seq. in 1 000	Knoten 1	Knoten 2	Knoten 3
1	2.32	2.33	3.68
2	2.27	2.22	3.72
3	2.20	1.99	3.74
4	2.14	1.23	3.73
5	2.55	1.22	3.72
6	1.79	1.24	3.70
7	1.08	1.54	3.68
8	1.04	1.57	3.66
9	1.02	1.56	3.63
10	1.00	1.54	3.60
11	—-	—-	3.56
12	—-	—-	3.51
13	—-	—-	3.45

Tabelle 5.1: *Aufgelistet ist die Verbindung von der Eins-Einheit zu einem rekurrenten Knoten beim Zeitüberbrücker-System, wobei das Gewicht erstmals nach wenigen tausend angelegten Eingabesequenzen aufgeführt wurde.*

fügen auch eine versteckte Einheit hinzu, welche einer ganzen Lage entspricht, und trainieren nur diese Einheit, während das übrige Netz eingefroren wird. Alle Lagen haben Verbindungen zu den darüberliegenden Lagen. Es werden also bei den bisher beschriebenen Methoden zu festen Zeitpunkten dem System zusätzliche Mittel zur Verfügung gestellt, um eine Aufgabe zu lösen.

Besser wäre es jedoch, wenn das Netz selbst feststellen könnte, wann neue Mittel nötig sind, d. h. wann keine Leistungsverbesserung mehr zu erkennen ist. Eine Möglichkeit ist eine Gewichtselimination wie Chauvin [Cha90] vorschlägt. Hierbei geht in die Fehlerfunktion die Anzahl und der Betrag der Gewichte ein, wobei der zusätzliche Fehler mit der Anzahl und dem Betrag der Gewichte wächst. Es wird also eine Lösung des Problems mit wenig und kleinen Gewichten gesucht, d. h. es werden Ressourcen zurückgehalten. Doch bei größerer Anzahl von Knoten ist nicht sichergestellt, daß nicht viele betragskleine Gewichte zu einer starken Aktivierung des Knotens führen. Außerdem ist es nicht von Nachteil, falls bei großen Gewichten die Netzeingabe trotzdem um 0 liegt, da sich die großen Gewichte gegenseitig auslöschen können und so der Knoten für weitere Aufgaben zur Verfügung steht. Auch bei der von Hanson und Pratt [HP90] angeregten Methode wird die Fehlerfunktion modifiziert, hier aber so, daß die versteckten Einheiten auf eine Netzausgabe von Null trainiert werden. Hertz [JHP91] beschreibt einen Algorithmus, bei dem die Gewichte, welche nicht relevant für das Netzverhalten sind, auf Null gedrückt werden. Weigend [ASWH91] läßt die Komplexität des Neztes in die Fehlerfunktion eingehen, wobei ein komplexeres Netz einen größeren Fehler erhält. Bei Weigend ist die Komplexität aber lediglich durch die Anzahl der Verbindungen bestimmt. Ein Weg zur Lösung des Problems wird von Honavar und Uhr [HU88] vorgeschlagen, hier kann das Netz seine eigene Topologie verändern, indem Verbindungen geschaffen und neue Knoten hinzugefügt werden.

Nachdem das Netz trainiert wurde, kann die Relevanz der Gewichte für das Verhalten des

Netzes berechnet werden. So werden bei der von Mozer und Smolensky [MS89] bevorzugten Methode die partiellen Ableitungen berechnet, um die Wichtigkeit der Verbindungen für die Netzausgabe zu erhalten. Karnin [Kar90] berechnet diese partiellen Ableitungen schon während der Lernphase. Und Le Cun [YLS90] verwendet zur Berechnung der Relevanz der Gewichte die 2. Ableitung der Fehlerfunktion.

Braucht das Netz zusätzliche versteckte Knoten, welche noch keine Aufgabe übernommen haben, so sollte diese Ressourcenbeschaffung vom Netz selbst durchgeführt werden und dies sollte erkannt werden, aufgrund mangelnder Leistungsverbesserung des Netzes.

Die Güte des Netzes zum Zeitpunkt t sei gegeben durch $G(t) := \frac{1}{s}\sum_{i=1}^{s} E(t-s)$, dem Mittelwert der letzten s Fehler, wobei $s > rp$ sein sollte, d. h. s sollte mindestens r Eingabesequenzen beinhalten. Sei

$$AG_u(t) = \left(G\left(\left[\frac{t}{s}\right]\right), G\left(\left[\frac{t}{s}\right]-1\right), \ldots, G\left(\left[\frac{t}{s}\right]-u\right)\right),$$

wobei hier [.] die Gauß-Klammer ist.

Sei weiter

$$V(t) = \frac{\max_{0\le i,j\le u} \mid AG_u(t)(i) - AG_u(t)(j) \mid}{\max_{0\le i\le u} \mid AG_u(t)(i) \mid},$$

d. h. V ist ein Maß für die Verbesserung des Netzes während der letzten us Schritte.

Man definiert mit $V_E(t) = \sum_{k,d_k(t)=\zeta} \lambda_k(c_k - y_k(t))^2$ den Fehler für die Nichtausgabeeinheiten zum Zeitpunkt t, wobei der Fehler des Knotens k mit λ_k gewichtet ist. Für einen logistischen Knoten k ist $c_k = 0.5$ und für einen linearen Knoten ist $c_k = 0$.

Hieraus ergibt sich ein neuer Gesamtfehler $E_{\text{ges}}(t)$ zum Zeitpunkt t mit $E_{\text{ges}}(t) = E(t) + F(V(t))V_E(t)$. F sollte eine positive, monoton steigende Funktion sein, welche unterhalb einer Schwelle die 0-Funktion ist, d. h. $x < \gamma \Rightarrow F(x) = 0$. Letzte Bedingung ist deshalb notwendig, da über die versteckten Knoten Fehler zurückfließen, die beliebig klein sein können, aber dennoch wichtig sind. Das Gesamtziel ist es $E(t)$ zu minimieren, aber die letzte Gleichung könnte minimiert werden, indem man $F(V(t))$ minimiert, also einen konstanten Fehler $E(t)$ erzeugt, deshalb wird $\frac{\partial F(V(t))}{\partial w_{ij}} = 0$ gesetzt.

Abschließende Betrachtungen

Die lange Lernzeit bzw. Instabilität während des Lernens bei rekurrenten Netzen, welche Information abspeichern sollen, um später eine gewünschte Ausgabe zu erzeugen, resultiert daraus, daß der Einfluß einzelner Gewichte auf die Netzausgabe von anderen Gewichten abhängig ist. Diese Abhängigkeit wächst exponentiell mit der Zeitspanne der Abspeicherung der Information, wie in Kapitel 2 gezeigt wurde.

Durch eine Reduktion der Eingabesequenz bei einer für 10 Zeitschritte gespeicherten Information wurde eine Leistungsvervesserung um den Faktor 25 erreicht, gegenüber herkömmlichen BP-Algorithmen aus Kapitel 1 (Siehe Abschnitt 3.2.1). Wollte man jedoch die gesamte Eingabe vorhersagen, so wurde mit den sonst üblichen Algorithmen überhaupt kein Erfolg erzielt, im Gegensatz zu der neuen Methode mit Kausaldetektoren. Bei einem komplexeren Beispiel (3.2.2) und bei einer Eingabesequenz mit verschiedenen Komplexitätsstufen (3.2.3) konnte mit den BP-Lernalgorithmen aus Kapitel 1 ebenfalls kein Lernerfolg innerhalb der verfügbaren Rechenzeit verbucht werden. Durch eine Reduktion der Eingabesequenz mit Kausaldetektoren konnte die Aufgabe gelöst werden.

Das in 3.3 vorgestellte Zeitüberbrücker-System kann die mehrstufigen Kausaldetetoren in ein System von nur zwei Netzen zusammenstürzen lassen. In diesem System werden alle Komplexitätsstufen erfaßt, d. h. es sind keine Überlegungen über die Komplexität der Eingabe notwendig. Bei praktischen Verfahren kann es aber zu gegenseitig negativen Einflüssen der Netze aufeinander kommen, so daß kein 'Real-Time' Algorithmus und eine 'Off-line' Version verwendet werden muß. Auch hier wurde gezeigt, daß es Aufgaben gibt, die durch das neuartige Zeitüberbrücker-System gelöst werden konnen, nicht jedoch mit herkömmlichen Methoden. Das Zeitüberbrücker-System ist für Aufgaben der Systemidentifikation geeignet, da hier ein einziges Netz die Dynamik in der Eingabesequenz erkennen muß.

Sind in den Eingabesequenzen keine lokalen Strukturen vorhanden, so können aber durch eine Konzeption des Netzes, wie in Kapitel 4 vorgeschlagen wird, auch Aufgaben gelöst werden, bei welchen Information über größere Zeiträume gespeichert werden muß. Auch hier konnten Aufgaben gelöst werden, welche mit den Algorithmen aus Kapitel 1 nicht lösbar waren. Die Idee des konstanten Fehlerrückflusses kann zu einem neuronalen Langzeitspeicher ausgebaut werden, welcher noch getestet werden muß. Auch die in Kapitel 5 vorgeschlagene Methode zur Ressourcenbeschaffung muß noch getestet werden.

Haben in einer Eingabesequenz für ein neuronales Netz zeitlich weit auseinanderliegende Eingaben Einfluß auf eine gewünschte Ausgabe, so können mit den Methoden von Kapitel 3 und Kapitel 4 manche Aufgaben innerhalb akzeptabler Rechenzeit gelöst werden, im Gegensatz zu den herkömmlichen BP-Lernalgorithmen. Liegen die relevanten Eingaben extrem weit auseinander, so sind die neuartigen Methoden aus Kapitel 3 und 4 die einzig bekannten Lernverfahren, um die Aufgabe in realistischer Zeit zu lösen, da die anderen Lernverfahren zurückliegende Ein-

gaben exponentiell zur vergangenen Zeitspanne in den Gradienten der Fehlerfunktion eingehen lassen.

Literaturverzeichnis

[And86] C. W. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems*. PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1986.

[Ash89] T. Ash. Dynamic node creation in backpropagation networks. In *Connection Science*, Band 1, Seiten 365–375, 1989.

[ASWH91] D. E. Rumelhart A. S. Weigend und B. A. Huberman. Generalization by weight-elimination with application to forecasting. In D. Touretzky und D. S. Lippman, Editoren, *Advances in Neural Information Processing Systems 3, in press*. San Mateo, CA: Morgan Kaufmann, 1991.

[BSA83] A. G. Barto, R. S. Sutton, und 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.

[Cha90] Y. Chauvin. Generalization performance of overtrained networks. In L. B. Almeida und C. J. Wellekens, Editoren, *Proc. of the EURASIP'90 Workshop, Portugal*, Seite 46. Springer, 1990.

[Chr90] Ronald L. Chrisley. Cognitive map construction and use: A parallel distributed processing approach. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, und G. E. Hinton, Editoren, *Proc. of the 1990 Connectionist Models Summer School*, Seiten 287–300. San Mateo, CA: Morgan Kaufmann, 1990.

[Elm91] J. L. Elman. Incremental learning, or the importance of starting small. Technical Report CRL Technical Report 9101, Center for Research in Language, University of California, San Diego, 1991.

[FL90] S. E. Fahlman und C. Lebiere. The cascade-correlation architecture. In D. S. Touretzky, Editor, *Advances in Neural Information Processing Systems 2*, Seiten 524–532. San Mateo, CA: Morgan Kaufmann, 1990.

[Ghe89] M. Gherrity. A learning algorithm for analog fully recurrent neural networks. In *IEEE/INNS International Joint Conference on Neural Networks, San Diego*, Band 1, Seiten 643–644, 1989.

[Hoc90] Josef Hochreiter. Implementierung und Anwendung eines 'neuronalen' Echtzeit-Lernalgorithmus für reaktive Umgebungen , 1990. Fortgeschrittenenpraktikum, Institut für Informatik, Technische Universität München.

[HP90] S. J. Hanson und L. Y. Pratt. Comparing biases for minimal network construction with backpropagation. In D. S. Touretzky, Editor, *Advances in Neural Information Processing Systems 1*, Seiten 177–185. San Mateo, CA: Morgan Kaufmann, 1990.

[HU88] V. Honavar und L. Uhr. A network of neuron-like units that learns to perceive by generation as well as reweighting of its links. In D. S. Touretzky, T. J. Sejnowski, und G. E. Hinton, Editoren, *Proc. of the 1988 Connectionist Models Summer School*, Seiten 52–61. San Mateo, CA: Morgan Kaufmann, 1988.

[JHP91] A. Krogh J. Hertz und R. G. Palmer. *Introduction to the theory of neural computation.* Addison Wesley, 1991.

[Jor86] M. I. Jordan. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, San Diego, 1986.

[Jor88] M. I. Jordan. Supervised learning and systems with excess degrees of freedom. Technical Report COINS TR 88-27, Massachusetts Institute of Technology, 1988.

[Kar90] E. D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on neural networks*, 1:239–242, 1990.

[Moz89] 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.

[Moz90] M. C. Mozer. Connectionist music composition based on melodic, stylistic, and psychophysical constraints. Technical Report CU-CS-495-90, University of Colorado at Boulder, 1990.

[MP69] M. Minsky und S. Papert. *Perceptrons.* Cambridge, MA: MIT Press, 1969.

[MR90] B. Müller und J. Reinhardt. *Neural networks: An introduction.* Springer Verlag, 1990.

[MS89] M. C. Mozer und P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. *Connection Science*, 1:3–26, 1989.

[Mun87] 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*, Seiten 165–176, 1987.

[NW89] Nguyen und 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.*, Band 1, Seiten 357–364, 1989.

[Pea88] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. Technical report, Dept. of Comp. Sci., Carnegie-Mellon Univ., Pittsburgh, 1988.

[Pea89] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269, 1989.

[RF87] A. J. Robinson und F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

[RF89] T. Robinson und 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*, Seiten 836–843, 1989.

[RHW86] D. E. Rumelhart, G. E. Hinton, und R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart und J. L. McClelland, Editoren, *Parallel Distributed Processing*, Band 1, Seiten 318–362. MIT Press, 1986.

[Rob89] A. J. Robinson. *Dynamic Error Propagation Networks.* PhD thesis, Trinity Hall and Cambridge University Engineering Department, 1989.

[Roh89] R. Rohwer. The 'moving targets' training method. In J. Kindermann und A. Linden, Editoren, *Proceedings of 'Distributed Adaptive Neural Information Processing', St.Augustin, 24.-25.5,*. Oldenbourg, 1989.

[Sam59] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:210–229, 1959.

[Sch90a] J. H. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1(4):403–412, 1990.

[Sch90b] J. H. Schmidhuber. Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report FKI-126-90 (revised), Institut für Informatik, Technische Universität München, November 1990. (Revised and extended version of an earlier report from February.).

[Sch90c] J. H. Schmidhuber. Recurrent networks adjusted by adaptive critics. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, Washington, D. C.*, Band 1, Seiten 719–722, 1990.

[Sch91a] J. H. Schmidhuber. Adaptive decomposition of time. In O. Simula, Editor, *Proceedings of the International Conference on Artificial Neural Networks ICANN 91, to appear.* Elsevier Science Publishers B.V., 1991.

[Sch91b] J. H. Schmidhuber. Learning to control fast-weight memories: An alternative to recurrent nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München, March 1991.

[Sch91c] J. H. Schmidhuber. Neural sequence chunkers. Technical Report FKI-148-91, Institut für Informatik, Technische Universität München, April 1991.

[SH91] J. H. Schmidhuber und R. Huber. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems, to appear*, 1991.

[Wer87] 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.

[Wer88] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1, 1988.

[Wil88] 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.

[WZ88] R. J. Williams und D. Zipser. A learning algorithm for continually running fully recurrent networks. Technical Report ICS Report 8805, Univ. of California, San Diego, La Jolla, 1988.

[YHH91] K. Yamashita Y. Hirose und S. Hijiya. Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, 4:61–66, 1991.

[YLS90] J. S. Denker Y. LeCun und S. A. Solla. Optimal brain damage. In D. S. Touretzky, Editor, *Advances in Neural Information Processing Systems 2*, Seiten 598–605. San Mateo, CA: Morgan Kaufmann, 1990.

Ich möchte allen, die am Zustandekommen dieser Arbeit beteiligt waren, meinen Dank aussprechen. Besonders bedanken möchte ich mich bei Herrn Prof. Dr. W. Brauer, für die Vergabe des interessanten Themas, sowie bei meinem Betreuer Dr. J. Schmidhuber für die zahlreichen Hilfestellungen und Anregungen und für seine Diskussionsfreudigkeit bei der Suche nach Lösungen für die aufgetretenen Probleme. Außerdem möchte ich den System-Administratoren danken, welche die benötigten Rechner und Software warteten.