Studienarbeit

Neuronale Echtzeitregelung eines doppeltinversen Pendels

Teil B

Netze und Lernalgorithmen

Robert Laufer

geboren 12.11.72 in Werneck

laufer@informatik.uni-wuerzburg.de

Angefertigt am
Lehrstuhl für verteilte Systeme (Informatik III)
Bayerische Julius–Maximilians–Universität Würzburg

Betreuer: Prof. Dr.–Ing. P. Tran–Gia,
Dipl.–Inform. K. Tutschku
Dipl.–Inform. R. Müller

Inhaltsverzeichnis

1. Einführung

In dieser Studienarbeit „Neuronale Echtzeitregelung eines doppeltinversen Pendels“ wird die Anwendung von Neuronalen Netzen zur Regelung eines doppeltinversen Pendels geprüft. In Teil A, siehe Heuler (1996), der beiden zusammengehörigen Studienarbeiten wird das Pendelsystem beschrieben, ein Simulationsmodell erstellt und in ein Programm umgesetzt, sowie das Problem der Echtzeitregelung gelöst. Außerdem wurde ein Fuzzy–Regler zur Regelung des Systems konstruiert. In dem vorliegenden Teil B wird, basierend auf dem Simulationsmodell, ein Neuronales Netz zur Regelung erstellt. Um herkömmliche Regler konstruieren zu können, ist es fast immer nötig, die Differentialgleichungen des Systems zu kennen und deren Parameter zu bestimmen. Oft ist dies schwierig und nur unvollständig möglich. Auch für das Trainieren Neuronaler Netze zum Regeln eines inversen Pendels ist es meistens nötig, die Systemdynamik mit Formeln erfassen zu können, siehe Anderson (1989), Barto, Sutton, und Anderson (1983) und Suykens, De Moor, und Vandewalle (1994). Deswegen wurde in dieser Studienarbeit besonders darauf Wert gelegt, daß man möglichst wenig zusätzliches Wissen in das Trainieren eines Neuronalen Netzes zur Regelung stecken muß. Die dabei auftretenden Schwierigkeiten und deren Lösung werden ausführlich beschrieben und die Ergebnisse mit denen des Fuzzy-Reglers verglichen.

1.1. Problem des Stab–Balancierens

Beim Stab–Balancieren geht es darum, einen Stab wie in Abbildung 1.1 im senkrechten Gleichgewicht zu halten. In der Standardliteratur, siehe Geering (1989) S. 126 ff, ist ein Pendel beschrieben, dessen unteres Ende mit einem Gelenk auf einem Wagen befestigt ist, der entlang einer Schiene verschoben werden kann, siehe Abbildung 1.1. Die Bewegungen von Wagen und Stab sind auf die vertikale Ebene beschränkt. Der Zustand dieses Systems wird durch den Winkel und die Winkelgeschwindigkeit des Stabs sowie durch die Position und die Geschwindigkeit des Wagens beschrieben. Die einzige Kontrollaktion besteht darin, auf den Wagen eine Kraft auszuüben, die ihn nach rechts oder links stößt.
Einen Fehlschlag hat man, wenn der Stab über einen bestimmten Winkelbereich hinaus fällt, oder wenn der Wagen die Grenzen der Schiene überschreitet. Es gilt nun, die auf den Wagen wirkende Kraft so zu bestimmen, daß der Stab möglichst senkrecht gehalten wird und der Wagen innerhalb der ihm zur Verfügung stehenden Strecke bleibt. In dieser Arbeit wurde statt des in Abbildung 1.1 gezeigten einfachen inversen Pendels ein doppelt inverses Pendel benutzt.

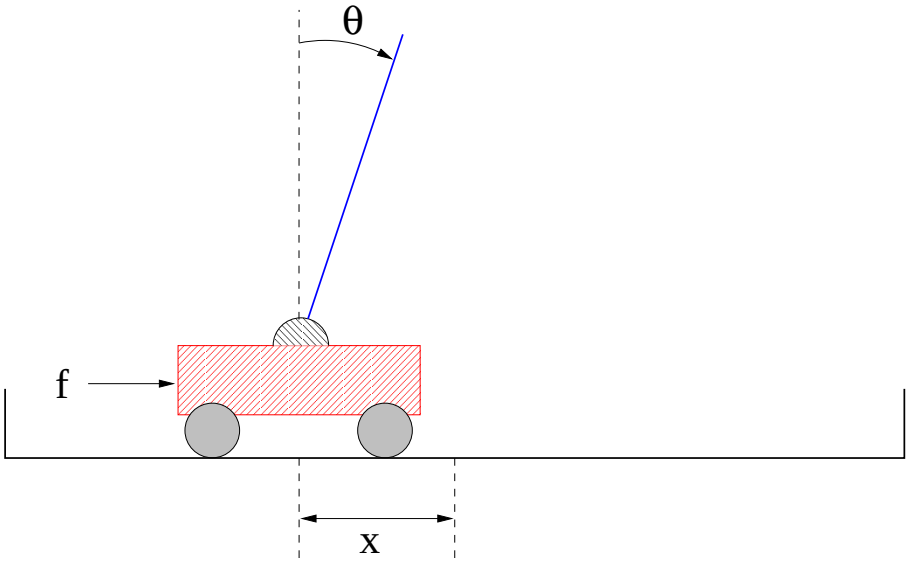
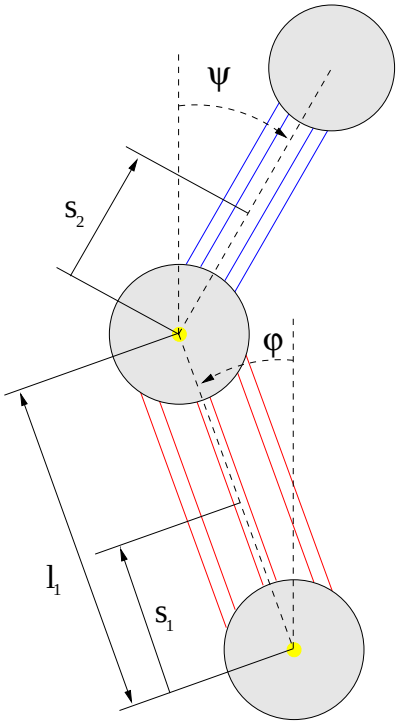


Abbildung 1.1: Wagen mit Stab

2. Pendelsystem

Das hier verwendete doppeltinverse Pendel ist in Abbildung 2.1 dargestellt. Auf der waagerechten Drehachse eines Motors ist ein Arm befestigt, der mittels der Steuerung des Motors bewegt werden kann. Am Ende des Motorarmes befindet sich ein Drehgelenk, an dem ein weiterer Arm befestigt ist, das eigentliche Pendel.
Die Winkel der beiden Arme kann man mittels Winkelencoder ablesen. Aus deren zeitlicher Änderung kann man ebenso die Winkelgeschwindigkeiten für Motor– und Pendelarm bestimmen.



Pendelwinkel	ψ
Schwerpunktabstand	s_2
Massenträgheitsmoment des Pendels	Θ_P

Antriebsarmwinkel	φ
Armlänge	l_1
Schwerpunktabstand	s_1
Massenträgheitsmoment des Arms	Θ_A

Abbildung 2.1: Aufbau und Beschreibung des Pendelsystems

Die Bewegungsgleichungen für dieses System und deren Herleitung sind bei Heuler (1996) zu finden. Sie wurden lediglich für die Simulation des Pendelsystems verwendet und flossen nicht in die Algorithmen zum Trainieren des Neuronalen Netzes ein.

3. Neuronale Netze zur Identifikation und Regelung

In diesem Kapitel werden zuerst knapp die Grundlagen Neuronaler Netze aufgezeigt. Anschließend wird erläutert, wie man prinzipiell Neuronale Netze zur Identifikation und zur Regelung von komplexen Systemen einsetzen kann. Weiter werden verschiedene spezielle, hier verwendete, Netzarten sowie der verwendete Lernalgorithmus vorgestellt. In den Kapiteln 4 und 5 wird die Verwendbarkeit dieser Netze für die Identifikation und Regelung des doppeltinversen Pendels untersucht.

3.1. Grundlagen

3.1.1. Neuronale Netze

Ein biologisches Neuronales Netz, siehe Abbildung 3.1, besteht aus Nervenzellen, den Neuronen, in einem Lebewesen. Die Neuronen haben jeweils Ein– und Ausgänge, mit denen sie untereinander verknüpft sind. Die Information, elektrische Impulse, die ein Neuron von seinen Eingängen bekommt, wird auf eine bestimmte Art verarbeitet. Daraus wird der Zustand des Neurons ermittelt. Der Zustand kann entweder angeregt oder nicht angeregt sein. Dieser Zustand wird über die Ausgänge an andere Neuronen weitergegeben, die daraus wiederum ihren Zustand ermitteln.

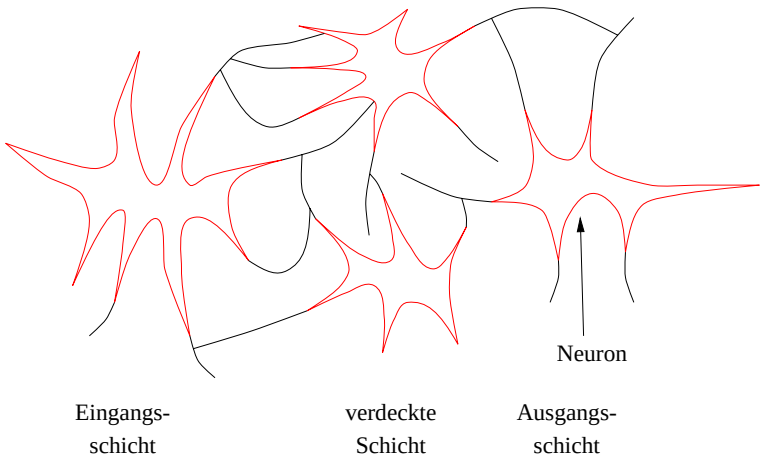


Abbildung 3.1: Biologisches Neuronales Netz

Wenn man ein biologisches Neuronales Netz auf dem Rechner simuliert, erhält man ein künstliches Neuronales Netz wie in Abbildung 3.2. Ebenso wie das biologische Neuronale Netz ist es aus Neuronen aufgebaut, die Verbindungen zueinander haben. Diese Neuronen sind in Schichten angeordnet. Hat jedes Neuron einer Schicht

Verbindungen zu allen Neuronen der folgenden Schicht, so nennt man das Netz vollvermascht. Aus den eingehenden Informationen ermittelt ein Neuron mittels einer Funktion seinen Zustand, meist durch Summenbildung der Eingänge und anschließende Skalierung mit einer Sigmoidfunktion. Dieser Zustand des Neurons kann auch Zwischenwerte annehmen[1]. Die Verbindungen zu anderen Neuronen können verschieden stark gewichtet werden. Legt man an der Eingangsschicht eine bestimmte Eingabe an, so liegt nach der Anpassung der Zustände der einzelnen Neuronen die Reaktion des Neuronalen Netzes an der Ausgangsschicht an. Hat das künstliche Neuronale Netz zum Beispiel eine mathematische Funktion gelernt, dann liefert es das Ergebnis dieser für die jeweiligen Eingabewerte in den Zuständen der Ausgabeneuronen.

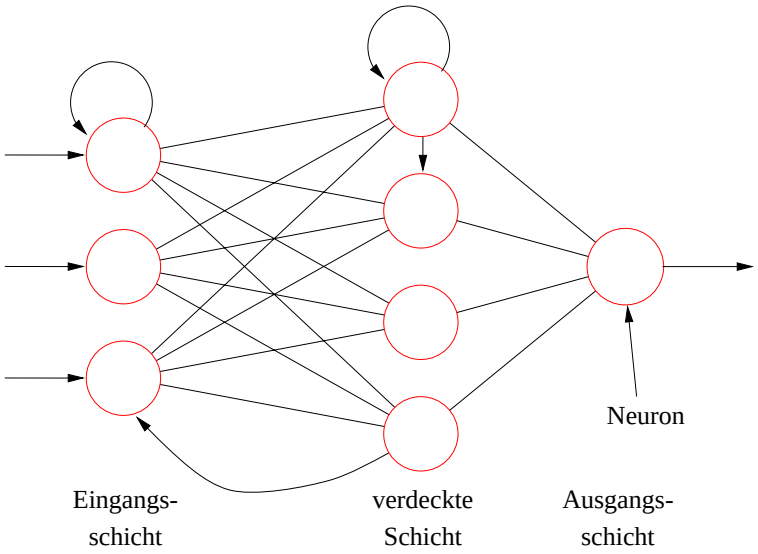


Abbildung 3.2: Rückgekoppeltes künstliches Neuronales Netz

Im Folgenden ist mit dem Begriff Neuronales Netz immer ein künstliches Neuronales Netz gemeint.

3.1.2. Identifikation

Identifikation eines technischen Systems, siehe Abbildung 3.3, bedeutet, dessen Verhalten nachzubilden zu versuchen. Meist handelt es sich bei diesen Systemen um nichtlineare Systeme, deren Dynamik nur mit einem großen Formelapparat erfaßbar ist. Will man sie nachbilden, stößt man auf das Problem, möglichst exakte Formeln zur Beschreibung der Systemdynamik zu bekommen. Mit Hilfe Neuronaler Netze geht man einen anderen Weg: Man schaltet parallel zu dem zu identifizierenden

[1] Statt zum Beispiel die diskreten Werte 0 und 1 zu verwenden, kann man rationale Zahlen wie 0.5 benutzen.

System ein Neuronales Netz und gibt an beide sowohl den Ausgangs–Zustand $x(t)$ des Systems als auch die auf das System angewandte Aktion $u(t)$. Den sich daraus ergebenden Zustand des Systems $x(t+1)$ vergleicht man mit der Ausgabe des Netzes und nimmt die Differenz als Fehlermaß für den Lernalgorithmus des Neuronalen Netzes.

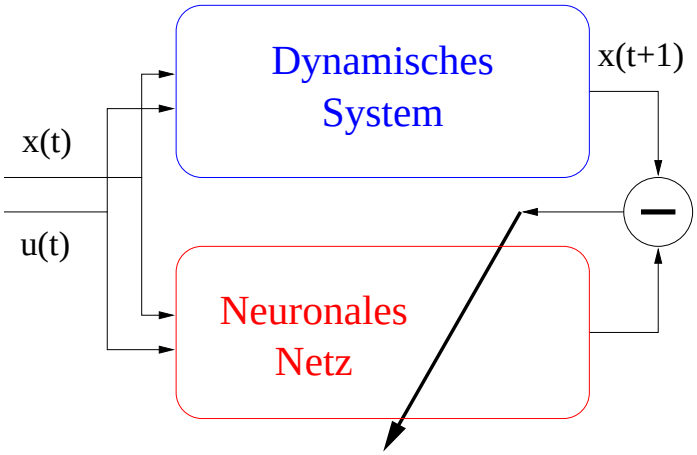


Abbildung 3.3: Identifikation eines Technischen Systems: Das Netz lernt den Folgezustand $x(t+1)$ des Systems auf die Aktion $u(t)$ im Zustand $x(t)$.

Hat man das System mit einem Neuronalen Netz identifiziert, so kann man dies zum Lernen eines Reglers für das System verwenden. Wünschenswert ist es, den Regler direkt am technischen System während des Betriebes desselben, also „Online“, lernen zu können. Oft ist dies aus technischen Gründen nicht möglich. Dies kann zum Beispiel aus Geschwindigkeitsgründen der Fall sein. Auch eine Verfälschung der Systemdynamik durch den Regler wäre denkbar. Nimmt man dahingegen das Neuronale Netz, welches das System identifiziert hat, so kann man mit ihm das System emulieren und so am Neuronalen Netz den Regler lernen.

3.1.3. Regelung

Schwieriger wird es bei der Regelung eines dynamischen Systems. Hierfür wird zuerst der Begriff der Regelung näher erläutert.

Definition der Regelung:

Bei der Regelung geht es darum, in einem dynamischen System, siehe Abbildung 3.4, eine bestimmte zeitveränderliche Größe, die Ausgangsgröße, auf einen fest vorgegebenen Wert, die Führungsgröße, zu bringen. Hierfür steht die beliebig veränderbare Stellgröße zur Verfügung, die die Ausgangsgröße stark beeinflußt. Oft kommt noch

eine unerwünschte Größe mit unerwarteten Änderungen hinzu, die die Ausgangsgröße vom erwünschten Verhalten abbringt, die sogenannte Störgröße.

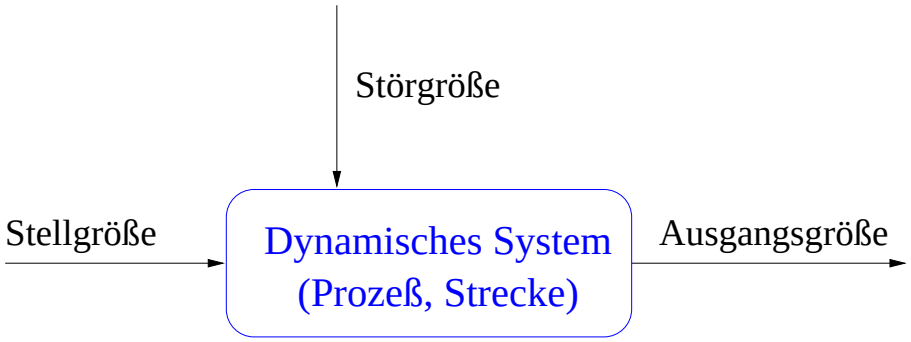


Abbildung 3.4: Blockbild eines dynamischen Systems

Um nun die Regelgröße an den gewünschten Verlauf anzupassen, ist die ständige Beobachtung der Strecke nötig. Die daraus gewonnenen Informationen werden dazu benutzt, die neue Stellgröße zu ermitteln, die die Störgröße ausgleichen soll. Eine Anordnung, die das leistet, siehe Abbildung 3.5, heißt Regelung.

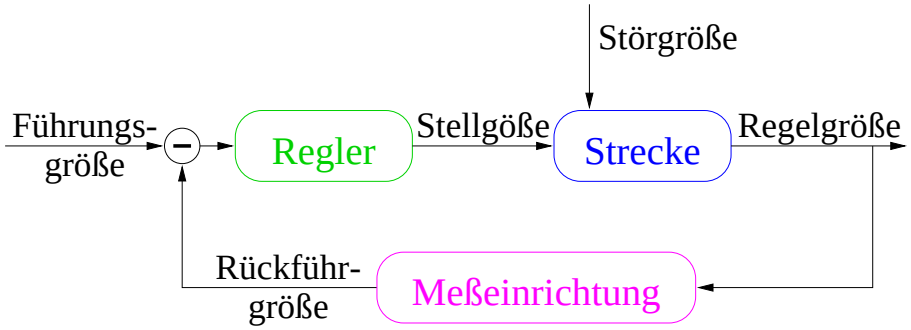


Abbildung 3.5: Blockbild einer Regelung

Um die Stellgröße ermitteln zu können, wird also ständig die Regelgröße beziehungsweise die Ausgangsgröße zurückgeführt. Im Gegensatz dazu wird bei einer Steuerung nur gemeldet, ob nach der Reaktion auf die Aktion der Steuerung bestimmte Bedingungen erfüllt sind oder nicht. An einen Regler gibt es Forderungen, die, wenn sie erfüllt sind, für seine Brauchbarkeit garantieren.

Forderungen an einen Regler:

- Stabilität:
  Dies ist das wichtigste Kriterium, das ein Regler erfüllen muß. Die Regelgröße

durchläuft normalerweise einen Einschwingvorgang, das heißt sie oszilliert um den Führungswert. Klingen die Oszillationen mit zunehmender Zeit ab, so ist die Regelung stabil. Bei Dauerschwingung oder zunehmender Schwingung bezeichnet man die Regelung als instabil und damit funktionsunfähig.

- Stationäre Genauigkeit:
  Eine ebenfalls unabdingbare Grundanforderung an einen Regler ist die Einhaltung einer hinreichenden stationären Genauigkeit. Dies bedeutet, daß die Differenz zwischen Rückführ– und Führungsgröße nach Beendigung des Einschwingvorgangs unter einer vorgegebenen Schranke liegen soll.

- Schnelligkeit und geringe Oszillationen:
  Diese beiden Forderungen stehen meist im Widerspruch, so daß man einen annehmbaren Kompromiß finden muß. Da der Führungswert in möglichst kurzer Zeit erreicht werden soll, muß die aufgeschaltete Stellgröße groß werden, was möglicherweise ein Überschwingen zur Folge hat.

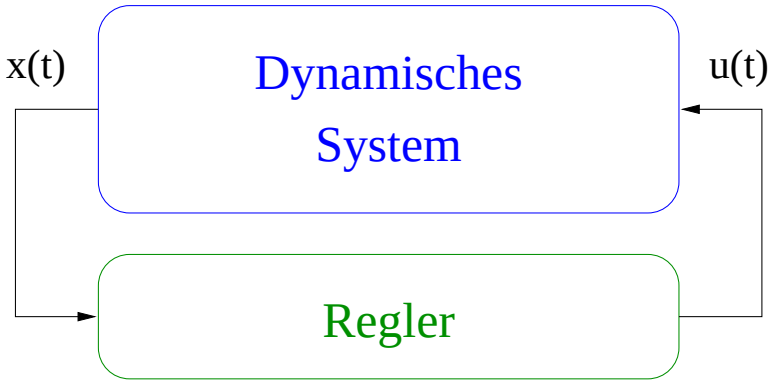


Abbildung 3.6: Prinzip der Regelung: Aus dem Zustand $x(t)$ des dynamischen Systems ermittelt der Regler die Stellgröße $u(t)$

Neuronale Netze sind für die Aufgabe der Regelung, siehe Abbildung 3.6, besonders geeignet, da sie die Fähigkeit besitzen, auch Abweichungen von der gelernten Systemdynamik gut ausgleichen zu können. Dies ist bei technischen Systemen oft der Fall, da hier leicht Abweichungen gegenüber den Bewegungsgleichungen auftreten. So kann man zum Beispiel die mechanischen Bewegungen einer lockeren Kette beim Anspannen nicht mit Formeln erfassen. Bekommt das Netz nun solche verfälschte Eingabeparameter, kann es diese gegebenenfalls ausgleichen und das System trotzdem regeln. Ein herkömmlicher Regler würde das nicht bewerkstelligen können, da er nach den Bewegungsgleichungen des Systems für sehr exakte Werte trainiert wurde und nur äußerst geringe Abweichungen davon kompensieren kann.

3.2. Neuronale Netzarten

In diesem Kapitel werden drei spezielle Arten von Neuronalen Netzen näher erläutert, die für das Regeln des doppeltinversen Pendels verwendet wurden.

3.2.1. Feedforward Netz

Bei einem „Feedforward Netz“, siehe Abbildung 3.7, gibt es nur vorwärtsgerichtete Verbindungen zwischen den einzelnen Neuronen. Nach Ablauf genau eines Zeitschritts liegt das Resultat des Netzes auf die Eingabe an den Ausgangsneuronen an. Es gibt keine Rückkopplung, das heißt das Netz kennt immer nur die aktuellen Eingabewerte und kann somit keine Zeitabhängigkeiten nachvollziehen. Das bedeutet für das doppelt inverse Pendel, daß das Feedforward Netz zwar auf eine bestimmte Position des Pendels hin reagieren kann, jedoch nicht weiß, in welcher Position sich das Pendel einen Zeittakt früher befunden hat. Dies wäre insbesondere dann von Nachteil, wenn man als Eingänge an das Netz nur die Winkelpositionen von Motor– und Pendelarm geben würde. In diesem Fall könnte das Feedforward Netz keine Aussage über die Winkelgeschwindigkeiten machen, was bei schnellen Bewegungen äußerst fatal wäre.

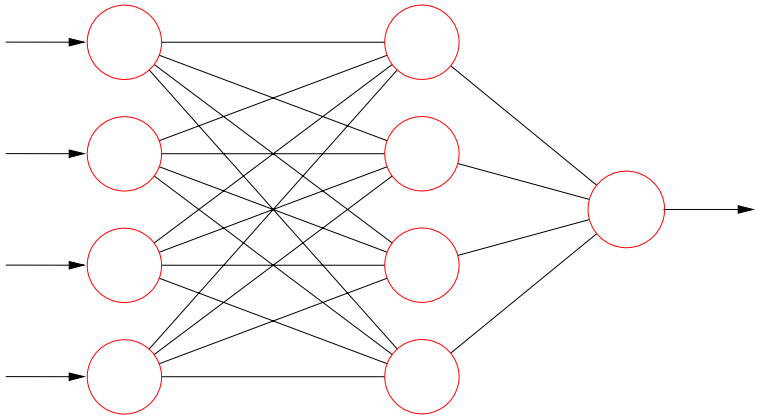


Abbildung 3.7: vollvermaschtes Feedforward Netz

3.2.2. Rekurrentes Multilayer Perceptron

Bei Rekurrenten Multilayer Perceptrons gibt es zusätzlich zu den Feedforwardverbindungen zeitlich verzögerte Verbindungen zwischen einzelnen Neuronen. Dabei können beliebige Neuronen miteinander verbunden werden, sogar ein Neuron mit sich selbst. Bei dem in Abbildung 3.8 dargestellten Netz werden zwei Neuronen der verdeckten Schicht zeitlich versetzt wieder in die Eingabeschicht zurückgeführt.

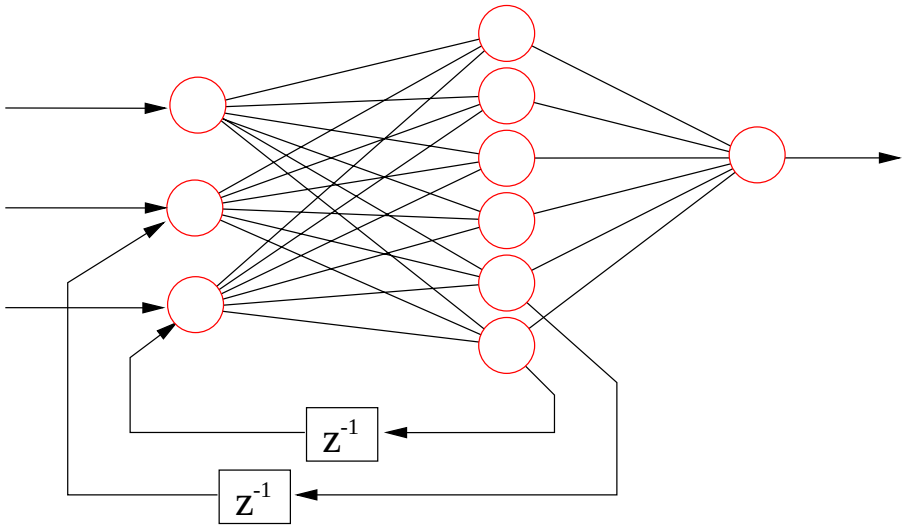


Abbildung 3.8: Rekurrentes Multilayer Perceptron

3.2.3. Elman Netz

Beim Elman Netz, siehe Elman (1989) und Elman (1990), einem speziellen rekurrenten Netz, wird die gesamte verdeckte Schicht einen Zeitschritt versetzt „gespeichert“. Dies wird dadurch erreicht, siehe Abbildung 3.9, daß man in der Eingangsschicht eine Kontextschicht anlegt, die dieselbe Größe wie die verdeckte Schicht hat. Die Verbindungen von der verdeckten Schicht zur Kontextschicht sind fest mit 1 gewichtet. Die Neuronen der Kontextschicht sind zu sich selbst mit einem vorgegebenen konstanten Gewicht rückgekoppelt. Es liegt an den Neuronen der Kontextschicht also immer der Wert einer „Zeitreihe“ über die Werte des zugehörigen Neurons der verdeckten Schicht an. Dadurch ist das Netz in der Lage, seinen inneren Zustand zu speichern. Mit dem Gewicht der Verbindungen der Neuronen in der Kontextschicht zu sich selbst kann man somit steuern, wie stark sich ein Elman Netz „erinnern“ kann. Ein Wert von 1 bedeutet, daß ein Neuron der Kontextschicht zum Bestimmen seines neuen Zustandes immer seinen alten Zustand mitberücksichtigt. Dieser alte Zustand entspricht einer Zeitreihe über alle vorherigen Zustände des zugehörigen Neurons der Hiddenschicht. Das Elman–Netz merkt sich also in der Kontextschicht eine Zeitreihe über alle Zustände der Hiddenschicht. Im Gegensatz dazu bedeudet ein Wert von 0, daß ein Neuron der Kontextschicht nur den letzten Zustand des zugehörigen Neurons der Hiddenschicht beachtet, und somit das Elman–Netz in der Kontextschicht nur den jeweils vorherigen Zustand der Hiddenschicht speichert. Der Hauptunterschied zu dem rekurrenten Netz in Abbildung 3.8 besteht darin, daß die Neuronen der verdeckten Schicht rückgekoppelt werden, und nicht speziell für diesen Zweck in die verdeckte Schicht und Ausgabeschicht eingefügte Neuronen. Gelernt wird das Elman Netz mit dem Backpropagation–Verfahren.

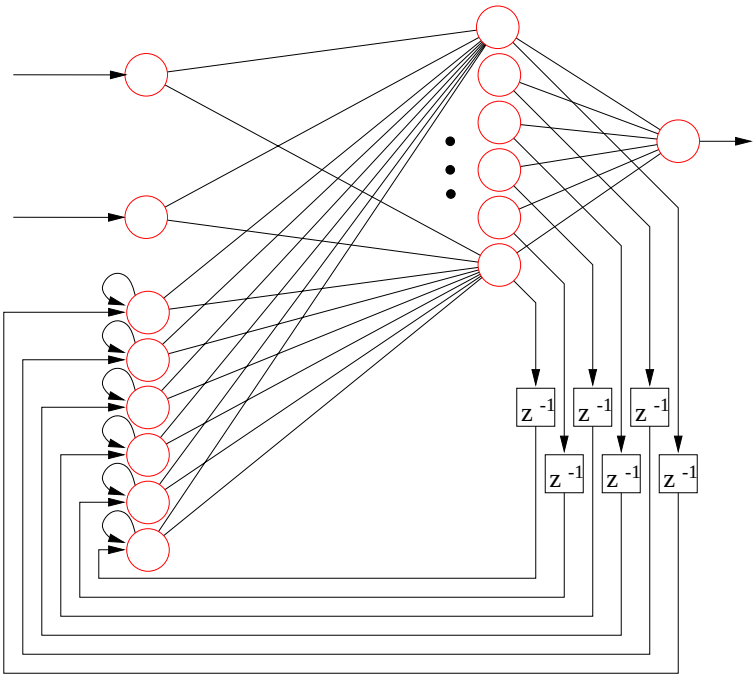


Abbildung 3.9: Prinzip des Elman Netzes: Parallel zur Eingabeschicht ist eine Kontextschicht, in der die verdeckte Schicht rückgekoppelt ist.

3.3. Der Backpropagation Algorithmus

Das Lernproblem des neuronalen Netzes besteht darin, die Gewichte unter den Neuronen so anzupassen, so daß ein definiertes Fehlermaß minimiert wird. Backpropagation, oder Fehlerrückführung, ist eine der am meisten verwendeten Lernmethoden zum Trainieren Neuronaler Netze. Die Backpropagation–Methode ist ein Gradientenabstiegsverfahren und wurde von Hinton, Rumelhart, und Williams (1986) ausführlich beschrieben.
Der Algorithmus findet ein lokales Minimum der Fehlerfunktion, indem die anfangs zufällig initialisierten Gewichte nach jedem Schritt durch Berechnung des Gradienten der Fehlerfunktion korrigiert werden.

Der Algorithmus sieht folgendermaßen aus:
Sei net_{ij} der Eingang des Netzes an Knoten j in Schicht i. Berechnet wird net_{ij} nach folgender Formel:

$$net_{ij} = \sum_{k=1}^{N_{i-1}} w_{k,j}^{f,i} y_{i-1,k}(n) + \sum_{k=1}^{N_i} w_{k,j}^{r,i} y_{i,k}(n-1) \tag{3.1}$$

wobei $y_{i,j}$ die Aktivierung des Knotens j in Schicht i, $w_{k,j}^{f,i}$ ein vorwärtsgerichtetes Gewicht von Knoten k in Schicht $(i-1)$ zu Knoten j in Schicht i, $w_{k,j}^{r,i}$ ein rekurrentes Gewicht von Knoten k in Schicht i zu Knoten j in Schicht i und N_i die Anzahl der Knoten im Unternetz i ist. Der Zeitindex $(n-1)$ zeigt an, daß die Rückkopplung um einen Zeitschritt verzögert ist. Die Aktivierung $y_{i,j}(n)$ des Knotens j in Schicht i ist:

$$y_{i,j}(n) = \sigma(net_{i,j}(n)), \tag{3.2}$$

wobei der Operator σ die Aktivierungsfunktion des Knotens beschreibt. Hier wurde die logistische Aktivierungsfunktion, auch Sigmoidfunktion, verwendet:

$$\sigma(s) = \frac{1}{1-e^{-s}} \tag{3.3a}$$

$$\sigma'(s) = \sigma(s)(1-\sigma(s)). \tag{3.3b}$$

Um den Lernprozeß durchführen zu können, benötigt man ein Qualitätsmaß für das Verhalten des Netzes. Üblicherweise wird hierfür eine Kostenfunktion basierend auf einem Fehlermaß definiert. Der quadratische Fehler für ein Element der Testmenge zum Zeitschritt n lautet:

$$E(n) = \frac{1}{2}\sum_k \left[E_k(n)\right]^2, \tag{3.4}$$

mit

$$E_k(n) = \begin{cases} y_{d,k}(n) - y_k(n) & \text{falls } y_{d,k} \text{ definiert} \\ 0 & \text{sonst,} \end{cases} \tag{3.5}$$

wobei k ein Element der Indexmenge aller Ausgabeneuronen ist, $y_k(n)$ die aktuelle Ausgabe des Neurons k zum Zeitschritt n und $y_{d,k}(n)$ die erwünschte Ausgabe von k. Der Netzfehler für eine Testmenge der Länge N ist:

$$E = \sum_{n=0}^{N-1} E(n) \tag{3.6}$$

Die Fehlermaße der Gleichungen 3.5 und 3.6 nähern sich 0, wenn die Lernmethode eine Lösung findet. Die Gewichtsanpassungsfunktion lautet:

$$\Delta w_{ij}(n) = -\eta\frac{\delta E(n)}{\delta w_{ij}} = \eta\sum_k E_k(n)\frac{\delta y_k(n)}{\delta w_{ij}} \tag{3.7}$$

Mit Hilfe dieser Formeln läßt sich nun der Backpropagation–Algorithmus formulieren:

1. Initialisiere alle Gewichte mit zufälligen Werten.
2. Gebe zufällig ein Ein–Ausgabemuster der zu lernenden Funktion vor und berechne die Belegungen der verdeckten Schicht und Ausgabeschicht.
3. Für die so vorgegebenen Eingabewerte und Zielwerte korrigiere die Gewichte entsprechend der Formel (3.7).
4. Fahre fort bei 2, bis ein Abbruchkriterium erfüllt ist.

Mögliche Abbruchkriterien sind eine untere Grenze für den Fehler, eine feste Anzahl von Iterationsschritten oder eine untere Grenze für die Änderung der Netzausgabe.

4. Identifikation des Pendelsystems durch das Neuronale Netz

In diesem Kapitel wird erläutert, wie man ein Neuronales Netz zur Identifikation des doppeltinversen Pendels einsetzen kann. Weiterhin wird gezeigt, wofür man das so trainierte Neuronale Netz benutzen kann.
Wie bereits in Kapitel 3.1.2 erläutert, schaltet man parallel zu dem zu identifizierenden System, also dem doppeltinversen Pendel, das Neuronale Netz, siehe Abbildung 4.1. Die Eingabe an das Netz ist der aktuelle Zustand des Pendelsystems beschrieben durch Winkel, Geschwindigkeiten, etc., sowie die durch den Regler ermittelte Kraft. Das Pendelsystem bekommt als Eingabe nur die Kraft, die Eingabe des Zustandes ist nur fiktiv. Aus dem Unterschied der Ausgaben des Pendelsystems und des Neuronalen Netzes ist ein Fehlermaß ermittelbar und hiermit das Neuronale Netz trainierbar.

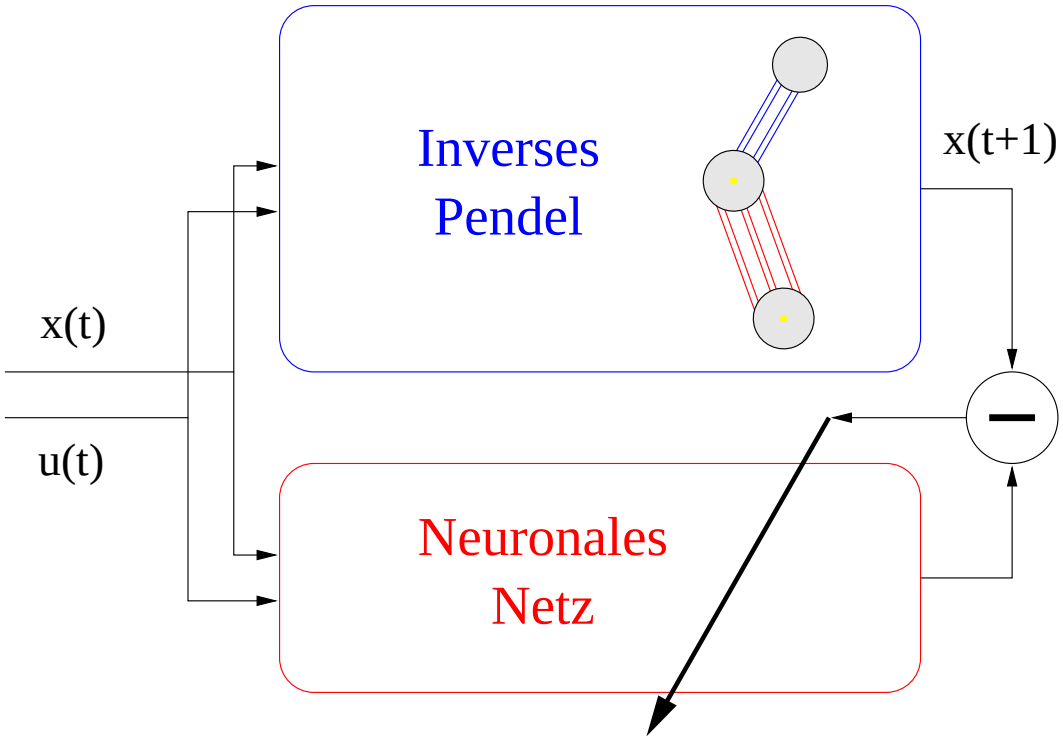


Abbildung 4.1: Identifikation des Pendelsystems.

Zur Identifikation muß man Online, also am laufenden doppeltinversen Pendel, lernen können. Außerdem benötigt man einen Regler, der das Pendel nicht umfallen läßt. Das Echtzeitlernen war am Hardwaremodell nicht möglich. Deswegen wurde ein mathematisches Modell erstellt, siehe Heuler (1996), mit dem das Pendelsystem

simuliert wurde. Durch das Verwenden eines Simulationsmodells ist es jedoch notwendig, zu überprüfen, ob das Modell mit dem Hardwaremodell annähernd übereinstimmt. Um das Pendelsystem identifizieren zu können, muß man dessen Reaktion auf einen Regler kennen. Deshalb benötigt man zuerst einen Regler, der das Pendelsystem so regelt, daß es nicht umfällt. Erst dann kann man es identifizieren. Hat man keinen Regler zur Verfügung, kann man dies mit dem Simulationsmodell umgehen. Ersetzt man das Pendel durch das Simulationsmodell, so kann man am Simulationsmodell das doppeltinverse Pendel identifizieren. Der Vorteil dieser Methode ist, daß kein gelernter Regler benötigt wird, der dafür sorgt, daß das Pendel beziehungsweise das Simulationsmodell stehen bleibt. Statt dessen kann man einen „Zufallsregler“ benutzen, der zufällig eine Stellgröße ermittelt. Für die Identifikation ist nur die Reaktion des doppeltinversen Pendels auf diese Stellgröße hin notwendig. Am Hardwaremodell ist das nicht möglich, da dieses bei einem Zufallsregler sehr schnell umfällt und jedesmal per Hand aufgerichtet werden muß.
Hat man erst einmal ein Neuronales Netz das System des doppeltinversen Pendels identifizieren lassen, so kann man dies zur Emulation des Pendelsystems, siehe Abbildung 4.2, benutzen. Dies ist vor allen Dingen dann notwendig, wenn man das Neuronale Netz zur Regelung des Systems nicht Online lernen kann. In dem Fall des doppeltinversen Pendels ist diese Notwendigkeit durch zwei Punkte gegeben. Zum einen ist für das Lernen des Neuronalen Reglers ein erheblicher Rechenaufwand, je nach Netzart und –größe, nötig, zum anderen müßte man das Pendel nach jedem Fehlschlag per Hand wieder aufrichten.

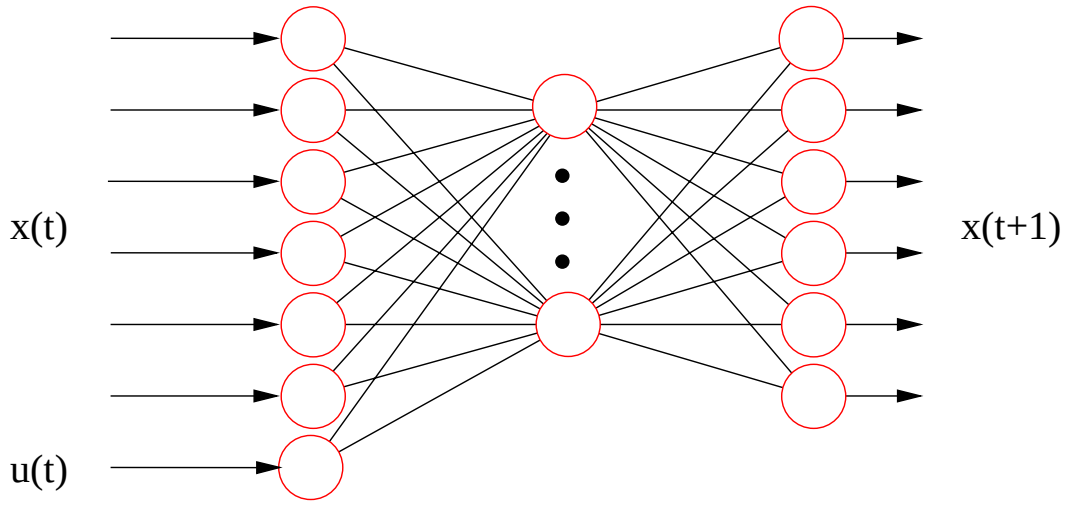


Abbildung 4.2: Emulator Netz: Das Netz bekommt als Eingabe den aktuellen Zustand $x(t)$ und die Kraft $u(t)$ und liefert als Ausgabe den nächsten Zustand $x(t+1)$. Es ersetzt das doppeltinverse Pendel.

Der Vorteil des Emulatornetzes konnte beim doppeltinversen Pendel nicht genutzt

werden. Das Emulatornetz konnte aus oben erwähnten Gründen nur am Simulationsmodell gelernt werden. Auch der Rechenaufwand des Simulationsmodells war geringer als der des Emulatornetzes, so daß nur das Simulationsmodell verwendet wurde.

5. Regelung des Pendelsystems

In diesem Kapitel werden die verschiedenen Verfahren vorgestellt, ein Neuronales Netz zum Regeln zu lernen. Hierbei wird prinzipiell erklärt, wie ein solches Verfahren funktioniert. Anschließend wird erläutert, wie es auf das hier verwendete doppeltinverse Pendel anwendbar ist.

5.1. Lernen mit Regler

Beim Lernen mit einem Regler nimmt man einen, wie in der Standardliteratur beschrieben konstruierten, Regler und identifiziert diesen mit dem Neuronalen Netz. Das bewirkt, daß das Neuronale Netz die Reaktionen des Reglers nachvollziehen kann und somit ebenso das doppeltinverse Pendel regeln kann. Für das Lernen stehen zwei Möglichkeiten zur Verfügung: Zum einen könnte man das Neuronale Netz parallel zum Regler Echtzeit lernen lassen, während der Regler das doppeltinverse Pendel regelt. Ein Echtzeitlernen ist hier nicht möglich gewesen.
Zum anderen ist eine einfachere Methode möglich, nämlich die des überwachten Lernens.
Beim überwachten Lernen sind bei jedem Trainingsmuster sowohl ein Eingangsmuster als auch das dazu gewünschte Ausgangsmuster vorzugeben. Der Trainingsprozeß verläuft als beaufsichtigtes „Lernen mit Lehrer“. Das zu lernende Eingangsmuster wird dem Netz präsentiert und ein Ausgangsmuster dazu berechnet. Aus der Differenz zwischen tatsächlicher und gewünschter Ausgabe wird der Fehler berechnet und damit das Netz gelernt. Dazu werden die Gewichtsfaktoren durch ein Lerngesetz, zum Beispiel Backpropagation, so adaptiert, bis das gewünschte Ausgangsmuster erreicht wird.
Das größte Problem bei diesem Verfahren ist, daß der Regler das Pendel schon so gut reguliert, daß man Ein- und Ausgabewerte nur noch für den Bereich erhält, der sehr nahe am Systemminimum, also an der Senkrechten ist. Durch diesen Mißstand bedingt lernt das Neuronale Netz auch nur das Verhalten in diesem Bereich und extrapoliert somit alle Werte, die außerhalb davon liegen. Das hat zur Folge, daß das Netz das Verhalten des Reglers nur in diesem kleinen Bereich lernt, sich außerhalb davon aber völlig anders verhalten kann.
Anhand eines Simulationsmodells läßt sich dieses Problem umgehen, indem man zum Beispiel das System in zufälligen Positionen startet. Am Hardwaremodell ist das sehr schwer möglich, da man an der resultierenden Mustermenge nicht exakt erkennen kann, ab welchem Moment sich das System frei bewegt hat und der Regler die Steuerung übernommen hat. Überdies muß dann zuerst einmal gewährleistet sein, daß der herkömmliche Regler das Pendel aus dem Startzustand überhaupt regeln kann. Herkömmliche Regler werden meist mit Hilfe von Kleinwinkelnäherung und Parameterbestimmung für kleine Bereiche ermittelt. Dadurch bedingt kann ein herkömmlicher Regler nur in einem kleinen Bereich das Pendel stabil regeln.

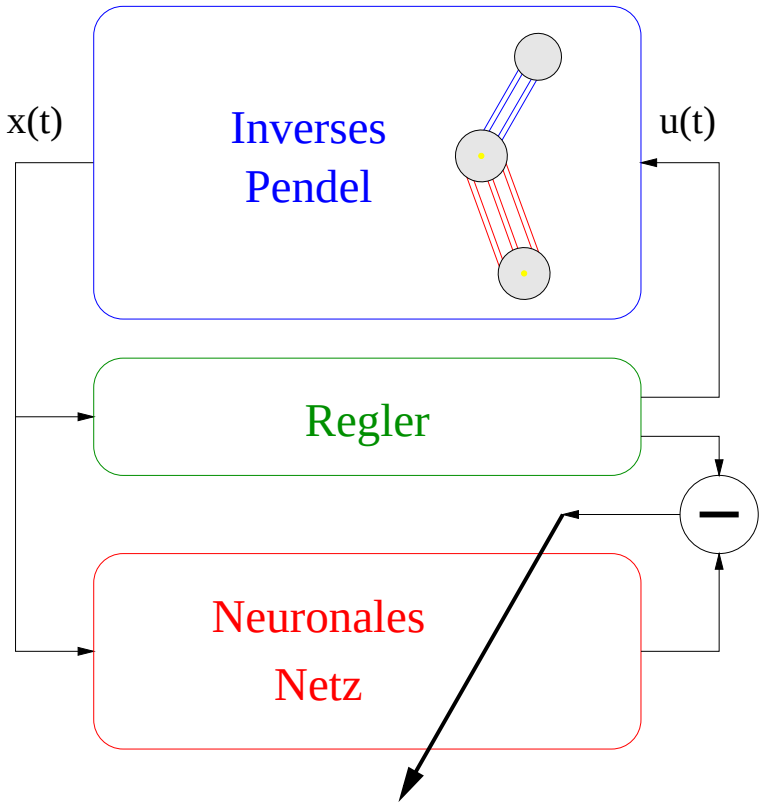


Abbildung 5.1: Lernen mit Regler: Das Netz lernt das Verhalten des Reglers

Der erste Versuch wurde mit dem Lernen eines rekurrenten Netzes mittels eines Fuzzy–Reglers gestartet. Das Hardware–Pendel wurde durch den Regler gesteuert und die Ein– und Ausgabewerte aufgezeichnet. Mit dieser Mustermenge wurde das Netz mit dem Backpropagation–Algorithmus trainiert.
Das Ergebnis war im Kleinwinkelbereich des Pendelarms sehr zufriedenstellend. Das Netz regelte das Pendel hier stabiler als der Regler, dessen Verhalten es gelernt hatte. Doch schon bei kleinen Abweichungen des Pendelarmwinkels[2] übersteuerte das Netz zu stark. Die Erklärung hierfür liegt auf der Hand:
Der Regler hat den Pendelarm ziemlich senkrecht reguliert, nur der Motorarm schwankte leicht hin und her. Die vom Regler gewonnene Mustermenge war somit nur auf diesen relativ kleinen Zustandsraum beschränkt. Das Netz konnte also auch nur das Verhalten des Reglers in diesem Bereich erlernen. Somit hat das Netz alle Werte, die außerhalb dieses Bereiches liegen, mehr oder weniger gut extrapoliert.

[2]Der Leser sei darauf hingewiesen, daß es sich hier um den Winkel des Pendelarms, nicht aber des Motorarms handelt. Gegenüber Abweichungen des Motorarmwinkels war das Netz stabil.

5.2. Lernen mit Gütemaß

Als zweiter Versuch wurde das Lernen mit einem Gütemaß, auch „Reinforcement–Learning“ genannt, angewendet. Reinforcement Learning kann als Spezialfall eines überwachten Lernverfahrens betrachtet werden. Die Gewichtsfaktoren werden für gute Aktionen verstärkt („belohnt“), daher auch der Begriff „reinforced“, und für schlecht durchgeführte Aktionen vermindert („bestraft“). Im Trainingsmustersatz wird allerdings nicht, wie beim überwachten Lernverfahren, das gewünschte Ausgangsmuster mitangegeben. Statt dessen wird nach jeder Aktion eine Bewertung derselben in Form eines skalaren Wertes errechnet, welcher ein Gütekriterium darstellt.
Die einfachste Variante davon ist den Fehlschlag, also das Umfallen des Pendels, als Fehlermaß zu nehmen. Das Problem hierbei ist, daß das Netz bis zum Fehlschlag mehrere Durchläufe hat, und man nicht mehr sagen kann, welcher Schritt wie stark ausschlaggebend für den Fehlschlag war. Außerdem nimmt mit längerem Lernen die Anzahl der Durchläufe bis zum Fehlschlag immer mehr zu (immer angenommen, das Netz wird durch Lernen verbessert). Das bedeutet, man kann das Netz anfänglich ziemlich schnell lernen, wohingegen es immer länger dauert, bis man das Netz noch einen Schritt weiter lernen kann. Ein weitere Variante besteht darin, siehe Abbildung 5.2, mittels einer Formel aus den Parametern des Systems ein Gütemaß zu bestimmen. Die Schwierigkeit hierbei ist, eine solche geeignete Bewertungsformel zu finden.
Als Algorithmus wurde das normale Backpropagation–Verfahren verwendet. Statt der Differenz zwischen gewünschter und tatsächlicher Ausgabe des Netzes wurde eine eigene Fehlerfunktion eingefügt (calculate_error() beziehungsweise calculate_error_without_move()). Diese Funktion simuliert einen Zeitschritt der Pendelbewegung und ermittelt hieraus mit Hilfe des Gütemaßes ein Fehlermaß.
Verschiedene Versuche ergaben, daß folgende Formel[3] ein sinnvolles Gütemaß darstellt:

$$50.0 * (pw - 0.5) + 10.0 * (mw - 0.5) \tag{5.1}$$

Bei den ersten Versuchen wurde nur die Differenz zwischen dem alten und dem neuen Gütemaß nach Ablauf eines Zeittaktes benutzt. Das hat zur Folge, daß das Netz lernt, diese Differenz zu minimieren, jedoch nicht das Gütemaß. Das heißt, das Pendel schwingt mit längerer Lerndauer des Netzes langsamer hin und her, jedoch hört es nicht damit auf, siehe Abbildung 5.3.
Um diesen Mißstand zu beheben, wurde zusätzlich eine Art Integral mit in das Fehlermaß aufgenommen. Das normale Gütemaß wurde immer aufsummiert und somit bewirkt, daß das Netz auch lernte, das Fehlermaß zu minimieren. Das so gelernte Netz hatte schon eine viel geringere Amplitude des Motorarms, und auch

[3] Die Werte für Pendelwinkel (pw) und Motorwinkel (mw) sind auf einen Bereich zwischen 0 und 1 umgerechnet.

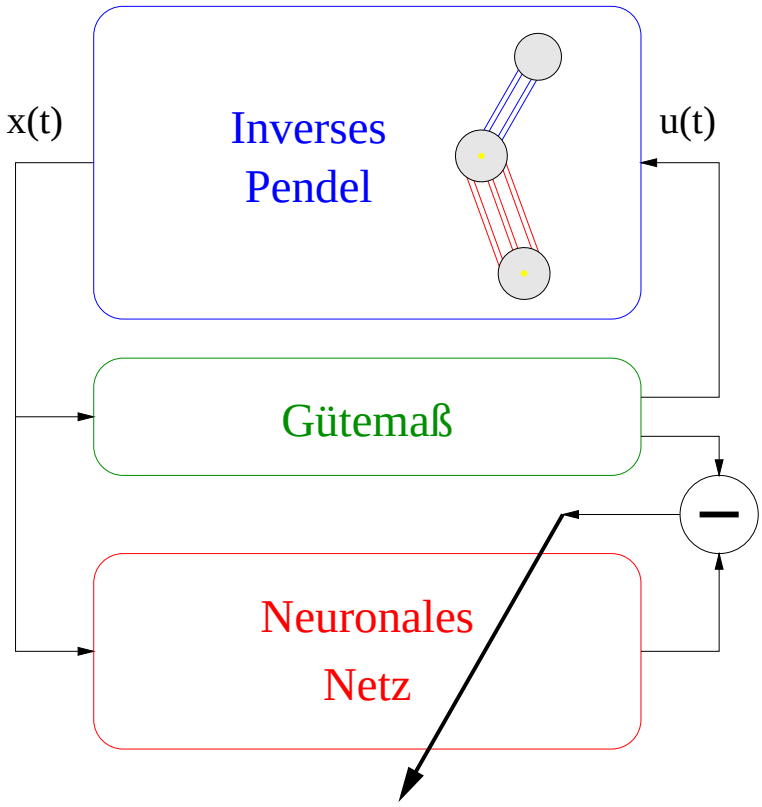


Abbildung 5.2: Lernen mit Gütemaß

die Schwingungsfrequenz war kleiner. Jedoch war das Netz noch nicht in der Lage, Motor– und Pendelarm gleichzeitig in der Senkrechten zu halten, siehe Abbildung 5.4.
Die Auslenkungen des Motorarms lagen teilweise bei 90^o, so daß das Netz nicht zur Anwendung auf das Hardware–Problem geeignet war. Dieses nämlich erlaubt technisch nur Auslenkungen von etwa 45^o.
Das verwendete Fehlermaß sah nach einer Skalierung auf einen Bereich von -1 bis 1 etwa folgendermaßen aus:

$$\frac{\arctan(0.05 * ((err - olderr) * 30.0 + 0.08 * integral))}{\pi} \tag{5.2}$$

Hierbei wurden *err* und *olderr* jeweils nach Formel (5.1) berechnet.
Die Schwierigkeit beim Finden dieser Formeln besteht immer darin, daß das Netz in Fällen mit großem Fehler nicht gegen die maximale Ausgabe konvergiert, da es sonst zu einem Stillstand des Lernprozesses in dieser Position kommt. Das Netz liefert dann nur diese Ausgabe, egal welche Systemparameter es als Eingabe hat.
Zu erklären ist das folgendermaßen: Das Netz weiß nicht, ob das Pendel wegen seiner Aktion umgefallen ist, oder ob es sowieso nicht mehr zu halten war. Es versucht

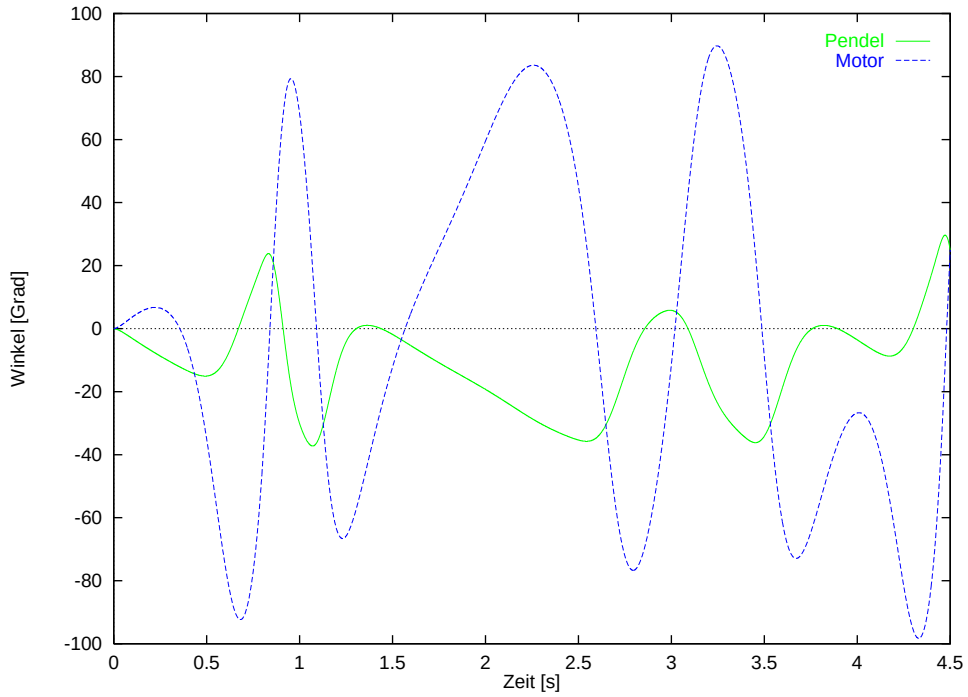


Abbildung 5.3: Schwingungsablauf ohne Integral

also, den Fehler zu minimieren und wendet seine größtmögliche Ausgabe an. Durch den Backpropagation–Algorithmus bedingt verfängt sich das Netz jedoch bei diesem Maximalwert, wenn es einmal dort angelangt ist. Man muß also beim Finden einer brauchbaren Formel für das Gütemaß sehr vorsichtig sein, damit der Fehler am Rand des Gültigkeitsbereichs nicht zu groß wird und eben dieses Problem auftritt. Wählt man dagegen das Fehlermaß so, daß der Fehler immer ziemlich klein bleibt, dann ist das gelernte Netz zu träge. Es kann dann das Pendel nicht schnell genug korrigieren und reagiert erst, wenn es schon zu spät ist.
Es zeichnete sich ab, daß das Netz dem Zeittakt nicht schnell genug folgen konnte. Deshalb wurde dieser verfeinert. Der normale Zeittakt liegt – durch die Hardware bedingt – bei 1 kHz. Ein mit 10 kHz an der Simulation gelerntes Netz war bei seiner Anwendung bei 10 kHz dann auch etwas besser als die mit nur 1 kHz gelernten Netze, siehe Abbildung 5.5. Die maximale Auslenkung war geringer geworden, und auch die Frequenz der Oszillation war geringer. Jedoch war das Ergebnis noch nicht auf das Hardware–Modell anzuwenden, da die Auslenkungen noch bis 60^o gingen. Überdies wäre das Netz sowieso nicht anzuwenden gewesen, da die Steuerkarte, die für das Hardware–Modell verwendet wurde, maximal 2kHz an Steuerfrequenz zuließ.
An Abbildung 5.5 sieht man gut, daß der Backpropagation–Algorithmus nicht immer das globale Minimum, sondern oft auch lokale Minima der Fehlerfunktion findet. So regelt das Netz das Pendel in einem Bereich von 0^o bis 6^o, und der Motorarm

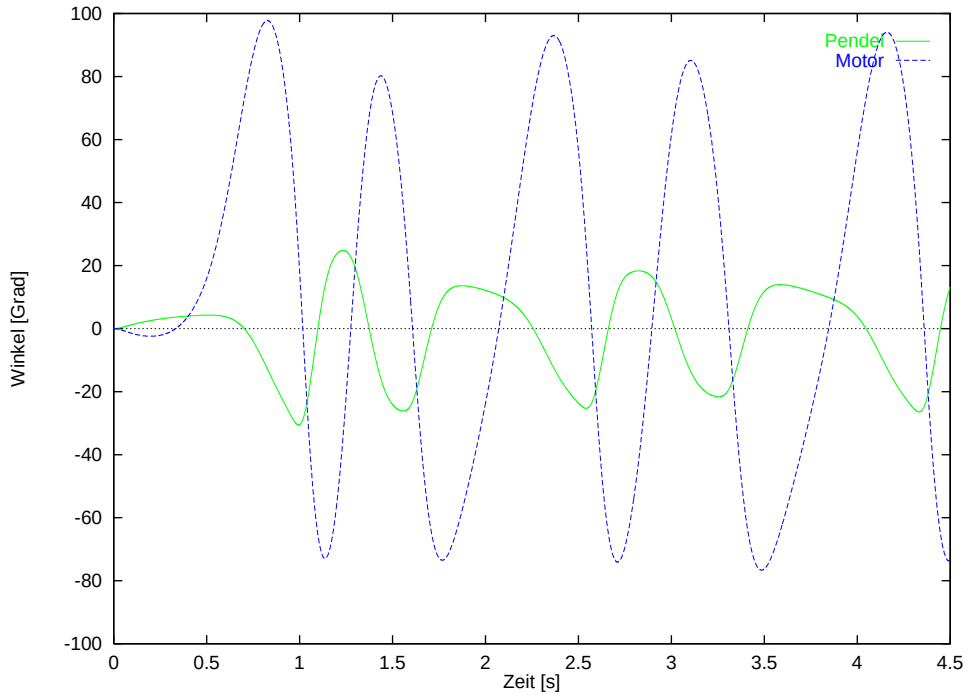


Abbildung 5.4: Pendel– und Motorwinkel mit Integral

pendelt sich bei etwa 30^o ein.
Dies führte zu der Idee, das mit 10 kHz gelernte Netz bei einer Frequenz von 1 kHz, also der eigentlichen Zielfrequenz, zu untersuchen. Das Ergebnis war überraschend gut. Das Netz regelte das Pendel zwar nicht mehr so stabil wie vorher (die Auslenkungen des Pendels waren bei etwa 5^o), jedoch schwankte der Motorarm auch nur um einen etwa gleich großen Winkelbereich.
Zu erklären ist das folgendermaßen: Bei einer Lernfrequenz von 1 kHz war das Neuronale Netz nicht in der Lage, dem Bewegungsablauf des Pendels schnell genug zu folgen und darauf zu reagieren. Bei einer Lernfrequenz von 10 kHz konnte das Neuronale Netz die Systemdynamik schnell genug adaptieren. Interessant ist, daß das so gelernte Netz beim Regeln mit 1 kHz, siehe Abbildungen 5.6, Start beim Nullpunkt, und 5.7, Start mit Auslenkung, besser war als mit 10 kHz, siehe Abbildung 5.5, Start beim Nullpunkt.
Dieser Erfolg wurde anschließend noch verbessert, indem wieder der ursprüngliche Lerntakt von 1 kHz verwendet wurde. Nun wurde aber statt einem Lernschritt je Zeittakt eine größere Anzahl Lernschritte durchgeführt. Dazu ging man folgendermaßen vor:
Zuerst wurde die zufällig gewählte Startposition des Pendelsystems einige Male durch das Netz geregelt, ohne einen Lernalgorithmus auf das Netz anzuwenden, um im Netz die Rückkopplungen richtig vorzubelegen. Jetzt wurde der Zustand des Pendelsystems sowie die Werte in den Kontextneuronen des Elman–Netzes gespei-

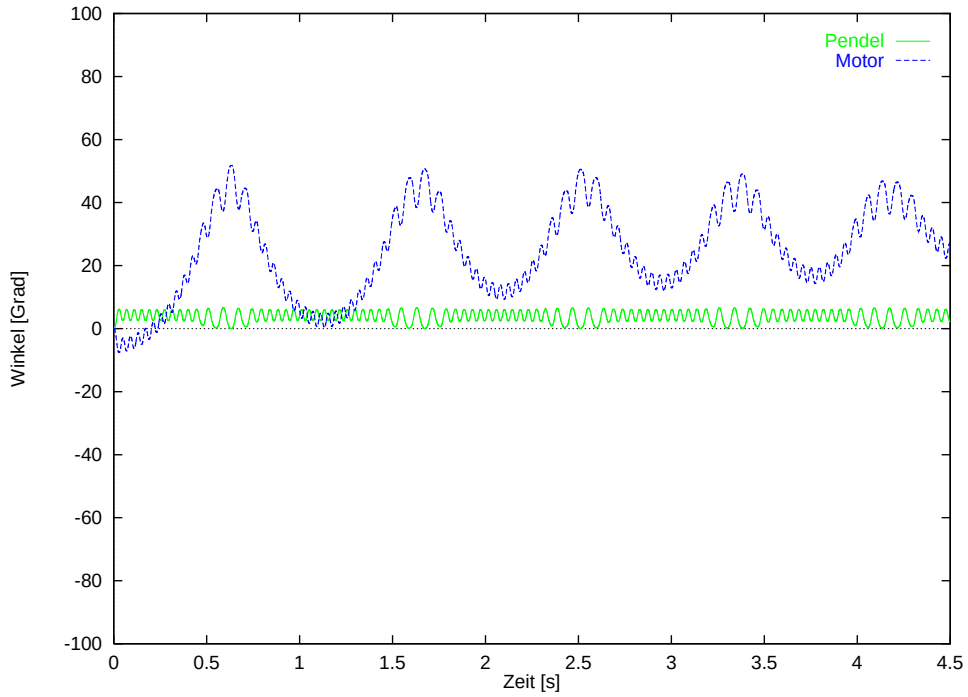


Abbildung 5.5: Training bei 10 kHz, Regelung bei 10 kHz

chert. Jeweils von der selben Startkonfiguration aus wurde dann ein Zustand des Pendels durch das Netz propagiert und mit dem daraus resultierenden Fehlermaß das Netz gelernt. Nach einer gewissen Zahl von Wiederholungen dieses Vorganges wurde das Pendelsystem wieder einen Schritt durch das Netz gesteuert und dieser neue Zustand von Pendelsystem und Netz gespeichert. Ein Problem bei dieser Methode war das Integral, das ebenfalls bei jeder dieser Schleifen wieder auf den ursprünglichen Wert zurückgesetzt werden mußte.
Die so gelernten Netze lieferten die besten Ergebnisse, siehe Abbildungen 5.8, Start beim Nullpunkt, und 5.9, Start mit Abweichung.
Bemerkenswert ist, daß ein Netz, das mit einem Gewicht am Ende des Pendels gelernt wurde, das Pendel mit drei Gewichten und Mutter noch stabiler regeln konnte. So wurde das Netz in den Abbildungen 5.8, Start beim Nullpunkt, und 5.9, Start mit Abweichung, zwar mit einem Gewicht am Ende des Pendels gelernt, beim Regeln aber auf das Pendel mit drei Gewichten und einer Schraubenmutter angewandt. Das selbe Netz, auf ein Pendel mit nur einem Gewicht angesetzt, liefert die etwas schlechteren Schwingungskurven 5.10, Start beim Nullpunkt, und 5.11, Start mit Abweichung.

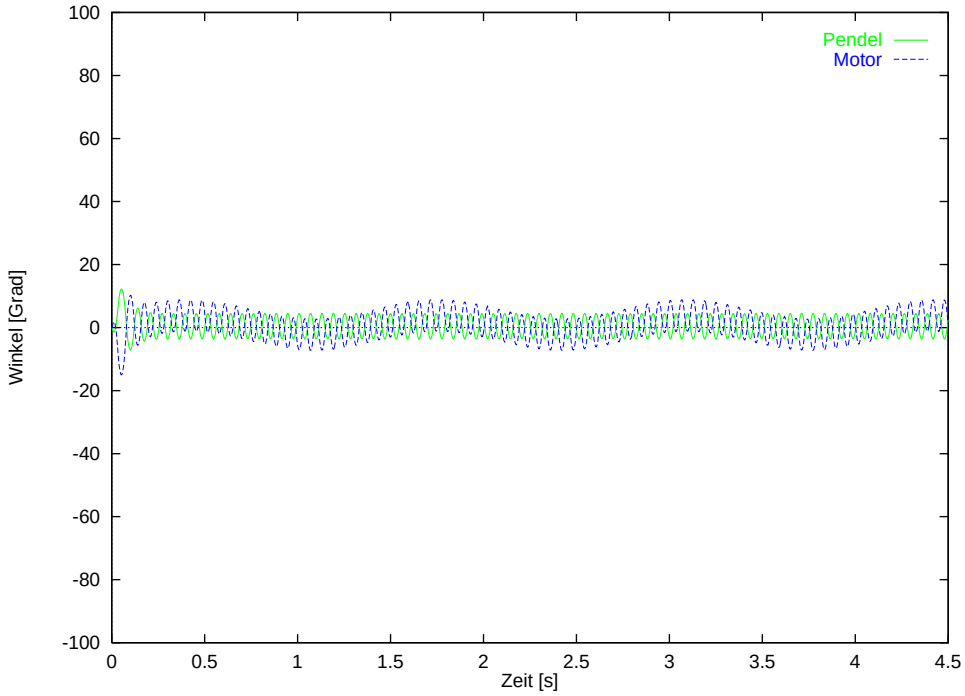


Abbildung 5.6: Training bei 10 kHz, Regelung bei 1 kHz

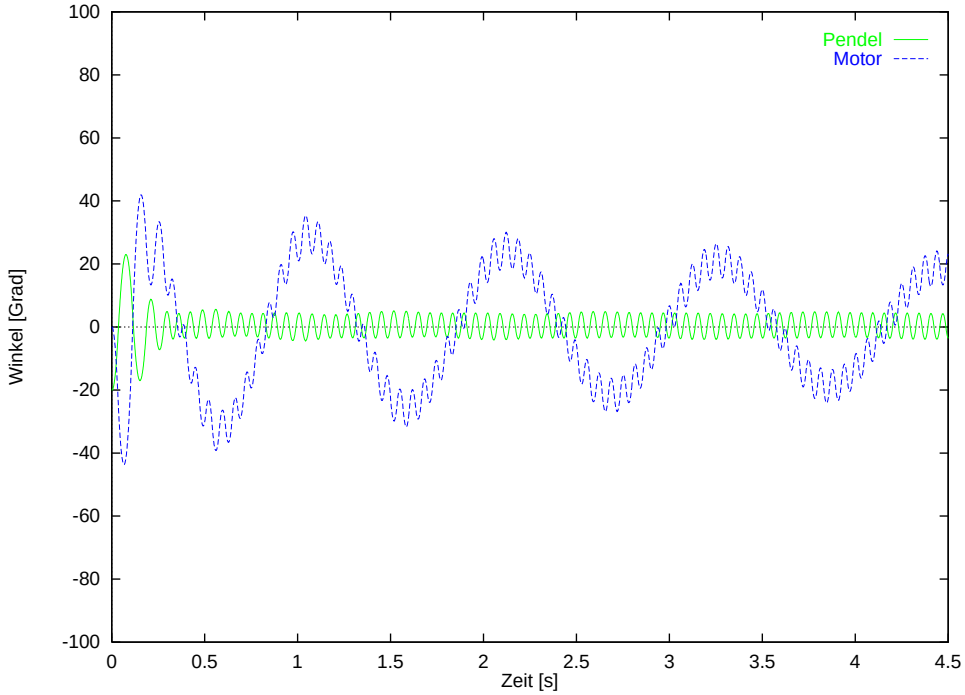


Abbildung 5.7: Einschwingen bei einer Auslenkung des Pendels von -20^o

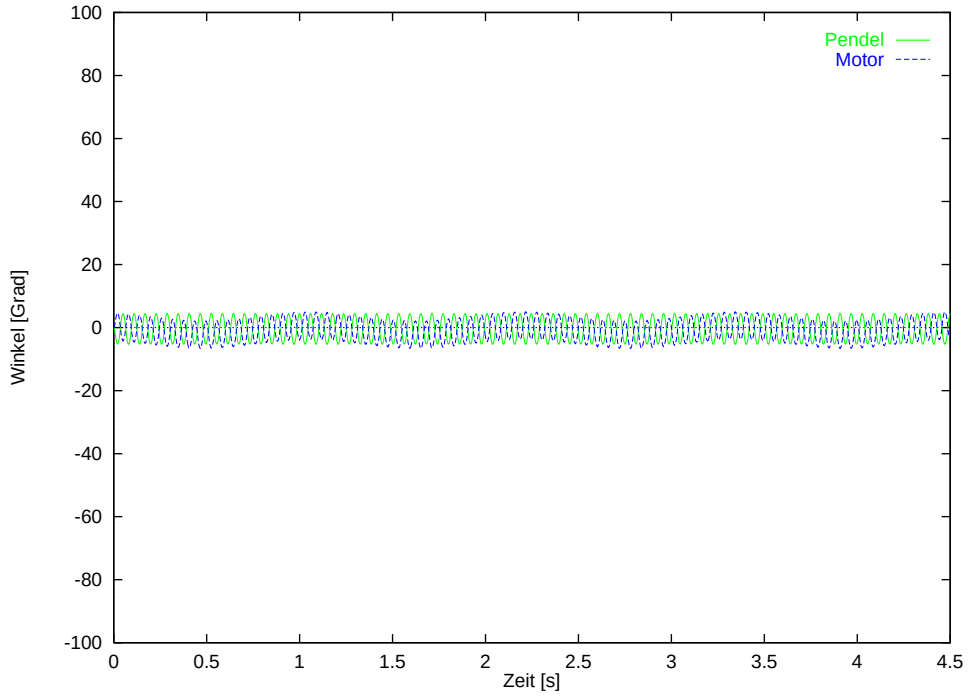


Abbildung 5.8: Training mit 10-facher Wiederholung, 3 Gewichte mit Schraube

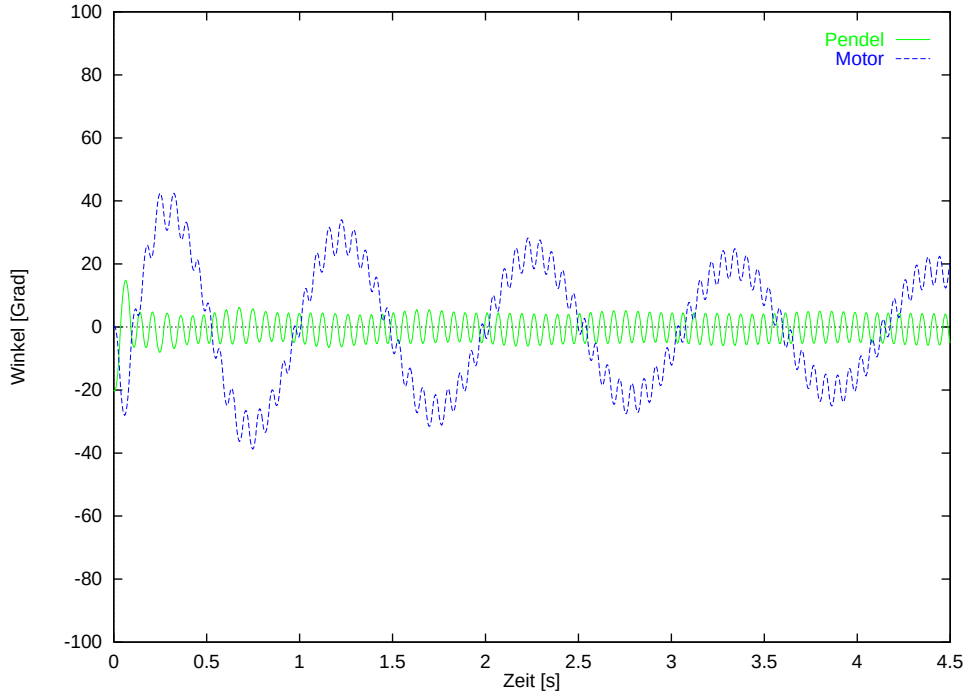


Abbildung 5.9: Einschwingen bei einer Auslenkung des Pendels von -20^o

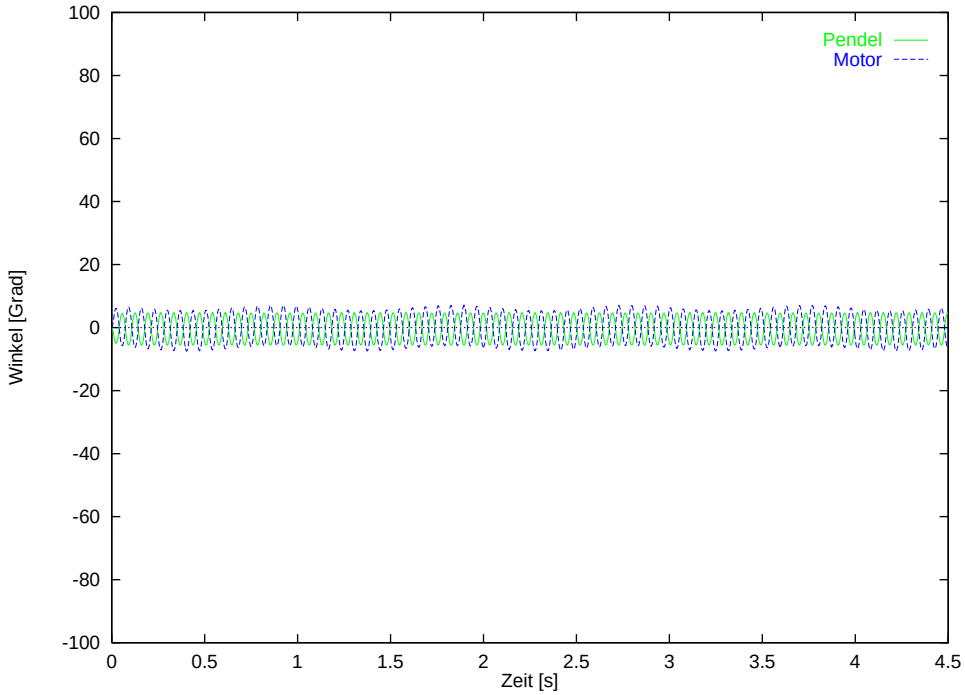


Abbildung 5.10: Pendel– und Motorwinkel, 1 Gewicht

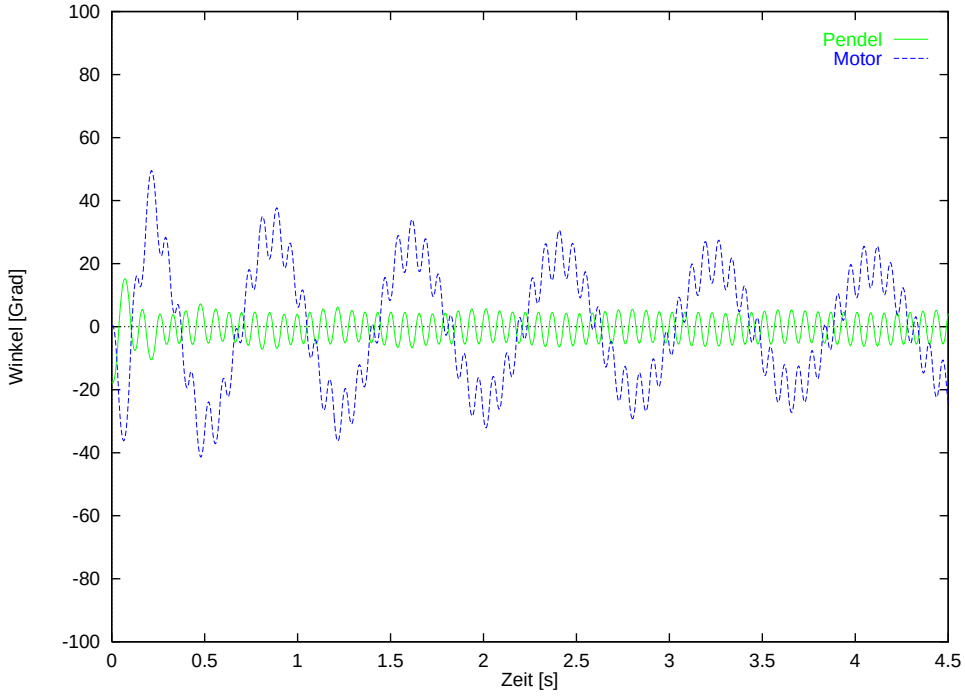


Abbildung 5.11: Einschwingen bei einer Auslenkung des Pendels von -17.5^o; 1 Gewicht

6. Zusammenfassung

Zuerst wurde das doppeltinverse Pendel mit einem Fuzzy–Regler geregelt. Hierbei wurde die Aktion des Reglers, die Stellgröße, siehe Abbildung 3.5, in Abhängigkeit von dem Systemzustand des doppeltinversen Pendels, der Regelgröße, mit aufgezeichnet. Anhand dieser Ein– Ausgabetupel für den Regler wurde ein Rekurrentes Multilayer Perceptron trainiert, siehe Abbildung 5.1. Das so gelernte Netz konnte das Pendel innerhalb eines kleinen Winkelbereichs des Pendelarms stabiler regeln als der Fuzzy–Regler. Da die Wertepaare beim Regeln mit dem Fuzzy–Regler aufgezeichnet waren, lagen sie in einem sehr kleinen Winkelbereich für den Pendelarm. Das Neuronale Netz konnte daher nicht lernen, wie es sich außerhalb dieses Bereichs verhalten soll. Dies hatte zur Folge, daß die Regelung mit dem Neuronalen Netz außerhalb dieses Bereichs instabil war.
Im Gegensatz zum Lernen mit Regler wurde dann ein Neuronales Netz ohne Regler gelernt. Hierzu war ein Simulationsmodell nötig, siehe Heuler (1996), da die vorhandene Hardware nicht schnell genug war, um das Neuronale Netz in Echtzeit lernen zu lassen. Anhand eines Gütemaßes wurde der Fehler für den Backpropagation–Algorithmus bestimmt. Es wurden die drei verschiedenen Netzarten Feedforward, Rekurrentes Multilayer Perceptron und Elman Netz verwendet. Das Elman Netz zeichnete sich vor allem durch Schnelligkeit aus. Aber auch die Ergebnisse der Regelung waren mit einem Elman Netz besser als mit einem Rekurrenten Multilayer Percetron. Die Regelung war jedoch noch nicht auf Hardware anwendbar, da der Motorarm noch zu große Winkel ansteuerte. Deswegen wurde versucht, das Neuronale Netz mit einer größeren Zeitauflösung lernen zu lassen. Das mit 10-fachem Zeittakt gelernte und regelnde Neuronale Netz lieferte schon bessere Ergebnisse als die bisherigen. Dies wurde noch verbessert, indem das Neuronale Netz nicht mit derselben Zeitauflösung regelte, mit der es trainiert worden war. Bei 10-facher Zeitauflösung beim Lernen und normaler Zeitauflösung beim Regeln waren die Ergebnisse zufriedenstellend. Verbessert wurden die Ergebnisse noch dadurch, daß das Neuronale Netz nicht mit 10-facher Zeitauflösung trainiert wurde, sondern statt dessen bei jedem Simulationsschritt mit Backpropagation diesen Vorgang 10 Mal wiederholt hat. Das so trainierte Neuronale Netz war gegen Störungen von außen beziehungsweise veränderte Systemparameter, wie zum Beispiel andere Gewichte am Pendel sehr robust.
Es konnte im Rahmen dieser Studienarbeit gezeigt werden, daß Neuronale Netze zum Lernen komplexer algebraischer Gleichungen, hier speziell das Regeln eines doppeltinversen Pendels, durchaus geeignet sind.

A. Funktionen

Die hier aufgeführten Funktionen sind speziell für das Lernen des Neuronalen Netzes. Alle weiteren Funktionen sind in Heuler (1996) dokumentiert. Dies ist nur eine Ergänzung dazu.

A.1. Datei: neterror.c

Funktionen:

Wählt einen zufälligen Startpunkt:

```
void random_init(void);
```

Testet, ob das Pendel innerhalb gewisser Grenzen ist:

```
int fault(void);
```

Simuliert die Bewegung auf die neue Kraft hin und berechnet daraus das Fehlermaß. Das Pendel hat sich nachher weiterbewegt:

```
double calculate_error(double newforce);
```

Simuliert die Bewegung auf die neue Kraft hin und berechnet daraus das Fehlermaß. Das Pendel steht nachher noch an der selben Position:

```
double calculate_error_without_move(double newforce);
```

Hilfsfunktion zum Bestimmen des Fehlermaßes; wird von calculate_error() aufgerufen; Formel für das Gütemaß:

```
double calc_err(void);
```

Speichert den alten Fehler:

```
void SaveNeterror(void);
```

Setzt den Fehler wieder zurück:

```
void RestoreNeterror(void);
```

A.2. Datei: simulation.c

Funktionen:

Simuliert die Bewegung des Pendels während eines Zeittaktes:

```
void Simulate(void);
```

Speichert die Simulation:

```
void SaveSimulation(void);
```

Stellt den gespeicherten Zustand wieder her:

```
void RestoreSimulation(void);
```

A.3. Datei: elman_jordan.c

Funktionen:

Speichert die Aktivierung der Kontextschicht:

```
void SaveElmanNetActivation(NET *);
```

Stellt die gespeicherte Aktivierung der Kontextschicht wieder her:

```
void RestoreElmanNetActivation(NET *);
```

B. Literaturverzeichnis

Anderson, C. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine 9 (13)*, 31–37.

Barto, A., R. Sutton, und C. Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, Cybernetics 13 (5)*, 834–846.

Elman, J. L. (1989). Structured representations and connectionist models. In *Proceedings of the 11th Annual Conference*, pp. 17–25. Cognitive Science Society: Erlbaum.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science 14*, 179–211.

Geering, H. P. (1989). *Meß- und Regelungstechnik, Mathematische Grundlagen, Entwurfsprinzipien, Beispiele*. Springer Verlag. Berichtigter Nachdruck.

Heuler, M. (1996). Neuronale Echtzeitregelung eines doppeltinversen Pendels – Simulation, Steuerung und Programm. Studienarbeit Neuronale Identifikation und Regelung am Lehrstuhl für Informatik III, Universität Würzburg, Institut für Informatik. Teil A von Neuronale Echtzeitregelung eines doppeltinversen Pendels.

Hinton, G., D. Rumelhart, und R. Williams (1986). Learning representations by backpropagation errors. *Nature 323*, 533–536.

Suykens, J. A., B. L. R. De Moor, und J. Vandewalle (1994). Static and dynamic stabilizing neural controllers, applicable to transition between equilibrium points. In *Neural Networks*, Volume 7 (5), pp. 819–831.