ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Neuroinformatik
Fachgruppe Neurobotik

Actor-Critic Design mit Echo State Networks zur Steuerung eines mobilen Roboters.

Diplomarbeit an der Universität Ulm

Vorgelegt von:
Johannes Uhlemann
johannes.uhlemann@uni-ulm.de

Gutachter:
Prof. Dr. Günther Palm
Dr. Mohamed Oubbati

Betreuer:
Dr. Mohamed Oubbati

1. November 2011

Actor-Critic Design mit Echo State Networks zur Steuerung eines mobilen Roboters.

Johannes Uhlemann
johannes.uhlemann@uni-ulm.de

Diplomarbeit im Fach Informatik
Fakultät für Ingineurwissenschaften und Informatik
Institut für Neuroinformatik
Fachgruppe Neurobotik
Universtität Ulm

1. November 2011

Zusammenfassung

In dieser Diplomarbeit wird Actor-Critic Design, mit Echo State Networks (ESN) als Kritik, zur Steuerung eines mobilen Roboters in einer veränderlichen Umgebung untersucht. Das ESN wird durch Online Training auf die Kostenfunktion trainiert, um in einem kontinuierlichen Zustands- und Aktionsraum Belohnung zu sammeln und Bestrafung zu vermeiden. Durch Simulationen gewonnene Daten werden unter realen Bedingungen reproduziert.

Schlüsselwörter: Actor-Critic Design, Adaptive Critic Design, Echo State Networks, mobiler Roboter, Navigation, Reinforcement Learning, Adaptive Learning, kontinuierlicher Aktions- und Zustandsraum.

Actor-Critic Design using Echo State Networks for control of a mobile robot.

Johannes Uhlemann
johannes.uhlemann@uni-ulm.de

Thesis for the degree of Diplom-Informatik
Faculty of Engineering and Computer Science
Institute of Neural Information Processing
Research Group of Neurobotics
Ulm University

1st November 2011

Abstract

This thesis investigates, how well an actor-critic design based echo state network (ESN) can adapt a mobile robot behaviour in a changing environment. The focus is on online training of the critic ESN to esimate the cost function, in order to learn an adaptive control policy that acquieres rewards and avoids punishments in a continuous state/action space. Experimental results are provided to support simulation results with real world data.

Keywords: actor-critic design, echo state networks, mobile robot, navigation, reinforcement learning, adaptive learning, continuous state and action space.

Selbstständigkeitserklärung

Der Verfasser erklärt, dass er die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als die angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Ulm,

Danksagungen

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Anfertigung dieser Diplomarbeit unterstützt haben. Meinem Betreuer, Dr. Mohamed Oubbati, für die anregenden Diskussionen und die vielen Hinweise. Ebenso möchte ich mich bei Herrn Professor Günther Palm bedanken.

Schließlich möchte ich den Autoren der folgenden Werkzeuge danken, ohne die diese Arbeit in dieser Form nicht möglich gewesen wäre:

- LaTeX
- Python
- NumPy
- SciPy
- Matplotlib
- TikZ

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Autonome mobile Roboter, die sich frei in einer realen Umgebung bewegen können und aus gesammelten Erfahrungen lernen, sind seit Jahren Gegenstand von Forschung und den Träumen der Menschheit. Doch trotz der Herausforderung wurden bisher nur geringe Fortschritte erzielt, und so sind autonome mobile Roboter nach wie vor weit von der praktischen Realisierung entfernt. Diese Tatsache ist unter Anderem der Komplexität des Themas geschuldet.

Autonomie erfordert Anpassungsfähigkeit an neue Aufgaben und neue Bedingungen. Aus diesem Grund scheiden viele etablierte Lernverfahren, die eine vorgegebene Lösung erfordern, für die Steuerung eines Roboters aus. Reinforcement Learning ist ein Ansatz, der hingegen keine Vorgaben benötigt wie eine Aufgabe zu lösen ist, und erscheint daher besonders geeignet.

Auf der anderen Seite sind die klassischen Reinforcement Learning Verfahren nicht für einen kontinuierlichen Zustands- und Aktionsraum geeignet. Eine Lösung dieser Problematik stellen Neuronale Netze dar, da diese in der Lage sind, Probleme kontinuierlicher Natur zu verallgemeinern. Insbesondere sind Recurrent Neural Networks zudem in der Lage, auch dynamische Probleme nachzubilden.

Das Training von Recurrent Neural Networks hat sich bisher jedoch als sehr schwierig erwiesen, die existierenden Trainingsalgorithmen sind mit vielen Problemen behaftet. Eine vielversprechende Lösung dieser Problematik stellen Echo State Networks dar. Ausgangspunkt dabei ist ein Recurrent Neural Network mit einer fest definierten internen Neuronenschicht, lediglich die Ausgabeschicht muss trainiert werden. Der Hauptvorteil gegenüber anderen Recurrent Neural Networks besteht in dem deutlich einfacheren Training.

Schließlich besteht jedoch das Problem, dass die hier angesprochenen Formen von Neuronalen Netzen auf Supervised Learning basieren, welches für einen autonomen Roboter ungeeignet erscheint. Mit Actor-Critic Design lässt sich dieses Problem umgehen, da es die Kombination von mittels Supervised Learning trainierten Neuronalen Netzen und Reinforcement Learning erlaubt.

Betrachtet man diese in der kürzeren Vergangenheit erzielten Fortschritte, scheint die Lösung vieler Probleme der Realisierung autonomer Roboter in greifbare Nähe gerückt zu sein. Aufgrund dieser vielversprechenden Aussicht soll die Kombination von Actor-Critic Design mit Echo State Networks zur

Steuerung autonomer mobiler Roboter in dieser Arbeit genauer untersucht werden.

1.2 Zielstellung

In dieser Arbeit soll die Kombination von Reinforcement Learning und Recurrent Neural Networks zur Steuerung eines realen Roboters untersucht werden. Im Detail sollen dabei die folgenden Punkte betrachtet werden:

- Realisierung eines Actor-Critic Design basierten Agenten, mit Echo State Networks zur Realisierung der Kritik.
- Dieser Agent soll auf einem realen Roboter in einem realen Szenario getestet werden.
- Insbesondere soll die Anpassungsfähigkeit an Veränderungen in der Umgebung untersucht werden.
- Der Einfluss der verschiedenen Parameter des Echo State Networks auf das Verhalten des Agenten soll ermittelt werden.
- Schließlich soll eine weiterführende Analyse der internen Abläufe stattfinden.

1.3 Aufbau der Arbeit

Die Arbeit ist in 7 Kapitel gegliedert.

Kapitel 1: Es erfolgt eine Darstellung der grundlegenden Motivation hinter dieser Arbeit, sowie der Ziele, die erreicht werden sollen.

Kapitel 2: Die grundlegenden Begriffe „maschinelles Lernen“ und „Neuronale Netze“ werden beschrieben. Danach folgt eine grundlegende Beschreibung von Actor-Critic Design.

Kapitel 3: Nachdem die Grundlagen bereits eingeführt wurden, erfolgt eine Vertiefung in Actor-Critic Design und Echo State Networks. Nachfolgend werden bereits existierende Arbeiten zu dieser Kombination vorgestellt.

Kapitel 4: Die Umsetzung dieser Arbeit und die durchgeführten Versuche werden beschrieben. Neben den mathematischen Grundlagen des Agenten beinhaltet dies ebenso eine Beschreibung des verwendeten E-Puck Roboters und der erstellten Software.

Kapitel 5: Im ersten Teil erfolgt eine Darstellung der durch Simulation erzielten Ergebnisse, sowie eine erste Auswertung dieser. Im zweiten Teil schließen sich die unter realen Bedingungen erzielten Ergebnisse an.

Kapitel 6: Die im vorherigen Kapitel erzielten Ergebnisse werden eingehender analysiert.

Kapitel 7: Es erfolgt eine abschließende Betrachtung und Diskussion der Ergebnisse, einschließlich von Stärken und Schwächen des untersuchten Ansatzes. Zudem erfolgt ein Ausblick auf zukünftige Forschung. Schließlich wird der Inhalt der gesamten Arbeit zusammengefasst und eingeordnet.

Kapitel 2

Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe maschinelles Lernen und neuronale Netze eingehender betrachtet. Diese inhaltliche Einführung soll als Grundlage für die spätere Vertiefung dienen, an deren Ende Echo State Networks (ESNs) und Actor-Critic Design (ACD) stehen.

Dabei steht an dieser Stelle weniger die Exaktheit der definierten Begriffe im Mittelpunkt, als ein grober Überblick um den Inhalt dieser Arbeit besser einordnen zu können.

2.1 maschinelles Lernen

Lernen steht für das Erwerben neuer Kentnisse oder Fähigkeiten. Diese können mannigfaltiger Natur sein, gelernt werden können sowohl sehr einfache Dinge wie grundlegende Fertigkeiten, als auch vergleichsweise komplexe Dinge, wie logische Zusammenhänge, abstraktes Denken oder analytisches Vorgehen.

Die Frage nach der Komplexität ist in diesem Zusammenhang auch eine Frage des Lernenden, so erscheinen verschiedene Dinge aus verschiedenen Blickwinkeln verschieden komplex.

Das Ziel des maschinellen Lernens ist ein System, das selbstständig aus empirischen Daten neue Kentnisse und Fähigkeiten erwerben kann. Dies ist insbesondere vom Auswendiglernen zu unterscheiden, das System soll in der Lage sein die gemachten Erfahrungen zu verallgemeinern und daraus gelernte Dinge auf neue Situationen anwenden zu können.

Im Gegensatz zum normalen Entwickeln von Algorithmen geht es dabei nicht darum, für eine gegebene Situation eine optimale Lösung zu finden. Tatsächlich ist es so, dass die Notwendigkeit zu Lernen häufig zu weniger optimalen Lösungen führt, als eine speziell für eine ganz bestimmte Aufgabe erschaffene Lösung.

Der Vorteil besteht jedoch in der Anpassungsfähigkeit an neue Bedingungen und in der Selbstständigkeit einer lernenden Maschine. Es sind viele Situationen denkbar, in denen es nicht möglich ist, eine neue Lösung für veränderte Randbedingungen zu entwickeln. Und, obwohl es ausgehend vom derzeitigen Kentnisstand noch ein sehr weiter Weg ist, existieren auch Probleme, für die die Menschheit bisher noch keine Lösung gefunden hat.

Maschinelles Lernen kann auf 3 Arten erfolgen, diese werden nun vorgestellt. Eine besondere Bedeutung für diese Arbeit besitzt dabei Reinforcement Learning (RL).

2.1.1 Supervised Learning

Beim Supervised Learning, oder auch beaufsichtigtem Lernen, wird einem Agenten eine vorgefertigte Menge an Wissen präsentiert, d.h. eine Menge von Eingabewerten mit zugehörigen Ausgabewerten.

Der Agent soll nun dieses Wissen lernen. Der Lernfortschritt lässt sich dabei direkt über den Vergleich der vom Agenten gegebenen Antworten mit den Vorgaben ermitteln.

Der Erfolg des Agenten beim Lösen der eigentlichen Aufgabe, dem Verallgemeinern des präsentierten Wissen, hängt jedoch, neben dem Lernfortschritt ansich, auch maßgeblich von der Qualität der gelernten Daten ab.

Umfassen diese z.B. nur einen kleinen Teilbereich des eigentlichen Problems, wird der Agent nicht in der Lage, sein das gesamte Problem zu lösen. Ebenso können keine Daten verallgemeinert werden, die sich nicht verallgemeinern lassen, weil z.B. keine Gesetzmäßigkeiten vorhanden sind. Die Möglichkeiten des Agenten diese zu erkennen spielt hier ebenso eine Rolle, wie das Vorhandensein selbst.

Nach einer anfänglichen Lernphase wird der Agent mit neuen Eingabewerten konfrontiert, für die er basierend auf seinem Wissen gute Antworten liefern soll. Das Training muss an dieser Stelle nicht beendet sein, sondern kann bei Bekanntwerden von neuem Wissen auch fortgeführt werden.

Das elementare Wesen von Supervised Learning besteht darin, dass zu lernendes Wissen stets von außen als solches gekennzeichnet werden muss. Diese Auswahl ist in der Praxis leider alles andere als einfach, und muss auch bezogen auf die Eigenschaften des Agenten getroffen werden. So neigt ein Agent mit zuvielen Freiheitsgraden z.B. dazu, bei einer geringen Menge an präsentiertem Wissen einfach alle Ausgaben auswendig zu lernen. Dieses Problem ist während der Lernphase nur schwer zu erkennen, da es sich durch einen besonders guten Lernfortschritt auszeichnet, der Agent lernt das präsentierte Wissen also sehr exakt. Jedoch ist er nicht in der

Lage, dieses Wissen zu verallgemeinern, weil es für das Reproduzieren der Ausgaben während des Lernens nicht nötig ist.

Ein weiteres Problem besteht darin, dass das vom Agenten zu bearbeitende Problem von vornherein bekannt sein muss. Es ist nicht möglich, ohne Intervention von außen, z.B. dem Auswählen neuer, geeigneter Trainingsdaten für eine zusätzliche Trainingsphase, neue Probleme zu lösen.

Diese Probleme gestalten den Entwurf von Agenten für Supervised Learning recht schwierig, da viele problemspezifische Details berücksichtigt werden müssen. Zudem muss der Rahmen, in dem sich die Probleme abspielen, von vornherein bekannt sein. Aus dieser Betrachtungsweise heraus ist Supervised Learning wenig flexibel.

Als Vorteil kann jedoch der einfach nachzuvollziehende Ablauf angesehen werden. So lassen sich Lernphasen eindeutig identifizieren, und der Fortschritt des Agenten in diesen lässt sich recht einfach ermitteln.

2.1.2 Unsupervised Learning

Bei Supervised Learning wird ein System mit ausgewähltem Wissen trainiert. Im Gegensatz dazu wird bei Unsupervised Learning kein Wissen präsentiert.

Verglichen mit den anderen Lernmethoden hat diese Methode nur einen kleinen Anwendungsbereich, z.B. bei der Klassifizierung von großen Datenmengen. Hierbei sollen automatisch Gesetzmäßigkeiten in Daten erkannt werden, und diese dann anhand der gefundenen Zusammenhänge eingeordnet werden. Eine verbreitete Methode hierfür sind selbstorganisierende Karten.

Im weiteren Verlauf der Arbeit wird Unsupervised Learning keine Rolle spielen, es sei hier nur aus Gründen der Vollständigkeit erwähnt.

2.1.3 Reinforcement Learning

Ein Zwischenschritt dazwischen, einem System sorgfältig auserlesenes Wissen zu präsentieren welches dieses dann lernt, und einem System ohne

weitere Informationen Daten zu präsentieren, ist, die vom System produzierten Ausgaben zu bewerten. Das System kann sich auf der Basis dieser Bewertung dann selbst verbessern.

Tatsächlich ist diese Art des Lernens sehr weit verbreitet, und wird als Reinforcement Learning (RL) bezeichnet [Berridge and Kringelbach, 2008]. [Montague et al., 2004] zeigt zum Beispiel, dass Tiere durch gute Erfahrungen in der Vergangenheit, wie z.B. dem Auffinden von Futter, motiviert werden, ihr Verhalten zu reproduzieren um erneut gute Erfahrungen zu sammeln.

Für Tiere stellt dieses Verhalten eine wichtige Fähigkeit zum Überleben dar. Durch die Assoziation von Sinneseindrücken mit den gemachten Erfahrungen wird das Tier in die Lage versetzt, die Resultate von zukünftigen Aktionen vorherzusagen.

Neuere Modelle von tierischem Lernen gehen davon aus, dass das Lernen ansich über die Diskrepanz zwischen erwartetem Resultat und eingetretenem Resultat stattfindet [Salas et al., 2010]. Die entspricht dem formalen Modell von RL, welches aus 3 Punkten besteht:

- Vorhersage der Auswirkungen von Aktionen
- Darauf basierend Auswahl einer guten Aktion
- Aus Erfahrungen lernen um Vorhersagen zu verbessern.

Dieser Aufbau ist in Abbildung 2.1 verdeutlicht.

Durch RL kann ein System wie mit Supervised Learning darauf trainiert werden eine bestimmte Aufgabe zu lösen, der Lösungsweg muss jedoch, im Gegensatz zu Supervised Learning, nicht bekannt sein. Da kein Lehrersignal vorgegeben wird, besteht auch nicht die Gefahr, dass dieses lediglich auswendig gelernt wird.

Eine wichtige Fragestellung bei RL ist, in welchem Maß bereits gesammeltes Wissen benutzt wird um eine Aufgabe zu lösen, und in welchem Maß neues Wissen erworben wird.

Ist das Resultat von bestimmten Aktionen bereits bekannt, und stimmt das tatsächliche Resultat auch mit den Vorhersagen überein, kann kein neues Wissen gewonnen werden. Der Agent ist jedoch nicht in der Lage ohne weitere Daten zum Vergleich einzuschätzen, ob er sich, gemessen an seinen Möglichkeiten, erfolgreich verhält oder nicht.

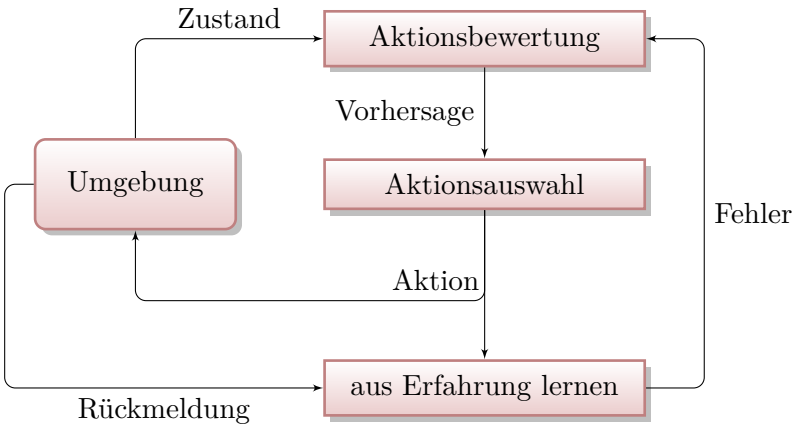


Abbildung 2.1: Reinforcement Learning. Die **Aktionsbewertung** sagt den Einfluss von Aktionen auf die zukünftige Bewertung voraus, die **Aktionsauswahl** wählt darauf basierend eine günstige Aktion aus und führt diese aus. Aus der, durch die Erfahrung gewonnenen, Bewertung der Aktion im Vergleich zur Vorhersage wird schließlich gelernt.

Um in einer solchen Sitation neues Wissen zu erwerben, muss ein Agent also neue Aktionen ausprobieren, um eine Grundlage zu erhalten auf deren Basis er vergleichen kann. Dafür müssen neue, unbekannte Aktionen ausgeführt werden, deren Resultat nicht vorhersehbar sind. Diese können, zumindest kurzfristig, auch zu schlechteren Resultaten führen.

Langfristig gesehen verbessern diese kurzfristig schlechten Resultate das Verhalten des Agenten, da ebenso auch neue, bessere Aktionen gefunden werden können. Eine besondere Bedeutung besitzt diese Problematik bei veränderlichen Problemen, da erworbenes Wissen mit der Zeit an Gültigkeit verlieren kann. Dieses muss also, durch die Erforschung neuer Möglichkeiten, stets aus dem aktuellen Stand gehalten werden.

Trotz dieser Problematik stellt RL eine sehr elegante Möglichkeit dar, die Probleme von Supervised Learning zu umgehen.

Es existiert eine Vielzahl von RL Algorithmen, die zum Teil sehr einfach aufgebaut sind, aber trotzdem erstaunliche Ergebnisse liefern. An dieser Stelle z.B. Q-Learning erwähnt [Sutton and Barto, 1998].

2.2 Neuronale Netze

In der ursprünglichen Bedeutung bezeichnen neuronale Netze miteinander verbundene Nerven in biologischen Lebensformen. Die offensichtliche Leistungsfähigkeit dieser Netze hat dazu geführt, ihre Architektur als Inspiration für künstliche Netze zu verwenden, in der Hoffnung eine ähnliche Leistungsfähigkeit und Vielseitigkeit zu erreichen.

Eine der bedeutendsten Vereinfachungen stellt das Perzeptron-Modell dar. Dieses Modell lässt sich besonders einfach berechnen und hat in vielen Netzarchitekturen seine Leistungsfähigkeit gezeigt. Ein Nachteil besteht jedoch darin, dass sich keine zeitlichen Abhängigkeiten modellieren lassen. Ein Perzeptron funktioniert quasi mit unendlicher Geschwindigkeit, ganz im Gegensatz zum biologischen Vorbild.

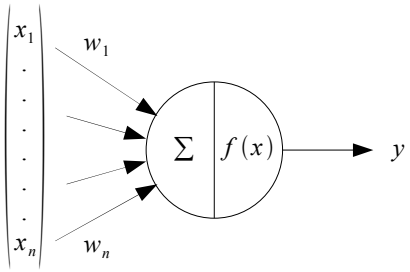


Abbildung 2.2: Perzeptron Modell

Abbildung 2.2 zeigt den Aufbau des Perzeptrons. Die Ausgabe y, also die Aktivierung des Neurons, ergibt sich über einen Eingabevektor x, einen Gewichtsvektor ω und einer Übertragungsfunktion $f(x)$ zu:

$$y = f\left(\sum_{i=1}^{n} x_i \cdot w_i\right) \tag{2.1}$$

Häufig wird noch die Neuronaktivierung $u \in \mathbb{R}$, $u = \langle x,w \rangle$ eingeführt. Es folgt dann $y = f(u)$.

Die Übertragungsfunktion f hat einen entscheidenden Einfluss auf die Charakteristik des Neurons. Sie bestimmt, wie empfindlich das Neuron auf

Eingabewerte reagiert. Eine lineare Funktion führt dazu, dass Veränderungen der Eingabewerte über den gesamten Wertebereich einen konstanten Einfluss auf die Ausgabe haben. Die häufiger benutzten beschränkten, nichtlinearen Funktionen, schränken diesen Einfluss auf einen gewissen Wertebereich ein, außerhalb dessen das Neuron in Sättigung geht. Das bedeutet, dass sich die Ausgabe trotz Änderungen der Eingabewerte nicht mehr nennenswert verändert.

Häufig kommt eine folgenden Funktionen zum Einsatz:

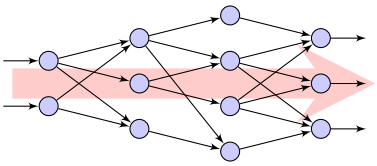
- lineare Funktion: $f(x) = x$
- Sprungfunktion: $f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$
- Sigmoidfunktion: $f(x) = \frac{1}{1+e^{-x}}$
- Tanh: $f(x) = \tanh(x)$

Die Linearität der letzten beiden genannten lässt sich über die Größe der Gewichte und damit dem Wertebereich von u beeinflussen Aussagen darüber, welche Funktionen in welchen Situationen besser funktionieren, lassen sich nur schwer treffen. Im Allgemeinen werden für sehr nichtlineare Aufgaben entsprechend auch sehr nichtlineare Übertragungsfunktionen verwendet.

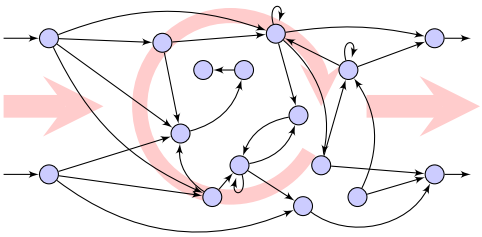
Feed Forward Networks (FFNs)

Einzelne Neuronen werden nun in einem Netz organisiert, d.h.miteinander verschaltet. Die Art und Weise wie dies stattfindet, also die Netzarchitektur, bestimmt maßgeblich die Eigenschaften. Eine der einfachsten Formen stellt das FFN dar. Die Neuronen werden in mehreren Schichten organisiert, wobei die Ausgabe einer Schicht jeweils als Eingabe der folgenden Schicht dient. Abbildung 2.3(a) verdeutlicht diesen Aufbau.

Für FFNs existieren sehr gute Lernalgorithmen, die meist auf einem Gradientenabstieg auf der Fehlerfläche der Netzausgabe in Abhängigkeit der Netzgewichte basieren, z.B. Error Backpropagation [Rumelhart et al., 1986]. Aus dieser Methodik ergeben sich diverse Vor- und Nachteile, so ist das Verhalten der Lernalgorithmen zwar gut vorhersehbar, lokale Optima in der Fehlerfläche stellen jedoch ein gravierendes Problem dar, für das es keine abschließende Lösung gibt.



(a)



(b)

Abbildung 2.3: (a) Schematischer Aufbau eines FFN. Der rote Pfeil steht für den Informationsfluss. Beim FFN wird aus der Eingabe direkt eine Ausgabe erzeugt, es handelt sich um eine statische Abbildung. Beim RNN (b) dagegen handelt es sich um eine dynamische Abbildung, die von Rückkopplungen beeinflusst wird.

Recurrent Neural Networks (RNNs)

Auf der anderen Seite stehen RNNs. Diese bestehen nur aus einer Schicht, abgesehen von Ein- und Ausgabe. In dieser können die Neuronen auf jede Weise miteinander verbunden sein (Abbildung 2.3(b)). Dies beinhaltet insbesondere auch Rückkopplungen.

Damit stellt das RNN quasi das Gegenstück zum FFN dar. Während beim FFN die interne Verschaltung einem festen Schema folgt, gibt es beim RNN dahingehend keine Einschränkungen. Es ließe sich auch ein FFN innerhalb eines RNN umsetzen.

Aufgrund der Komplexität der inneren Abläufe in einem RNN, gestaltet sich das Training deutlich schwieriger. Es existieren von FFNs inspirierte Algo-

rithmen, diese sind jedoch sehr rechenaufwendig und, im Gegensatz zu FFNs, ist die Konvergenz gegen eine Lösung nicht sichergestellt. Instabilität ist ein häufig auftretendes Problem [Jaeger, 2002].

Mit dem Perzeptron-Modell erzeugt das FFN eine statische Abbildung von Eingabewerten zu Ausgabewerten, dynamisches Verhalten lässt sich nicht modellieren. Im Gegensatz dazu hängt die Ausgabe eines RNN nicht nur von der aktuellen Eingabe ab, sondern auch von den vorherigen. Dynamisches Verhalten lässt sich so realisieren.

Training

In diesem Abschnitt soll nun näher auf das Training der Netzkonzepte FFN und RNN eingegangen werden. Die existierenden Lernalgorithmen für FFNs und RNNs fallen in die Kategorie Supervised Learning. Error Backpropagation ist das Grundprinzip vieler Lernalgorithmen, daher folgt nun eine Beschreibung des Error Backpropagation Lernalgorithmus für FFNs [Rumelhart et al., 1986].

Ausgangspunkt ist ein FFN mit dem Eingabevektor x, einer versteckten Schicht u, und der Ausgabeschicht y. Die Eingabeschicht ist über die Gewichtsmatrix $\mathbf{W_1}$ mit der versteckten Schicht verbunden. Die Ausgabe dieser Schicht ergibt sich zu:

$$u = f_1(\mathbf{W_1} \cdot x) \tag{2.2}$$

f_1 stellt die Ausgabefunktion der Neuronen der versteckten Schicht dar. Analog gilt für die Ausgabeschicht:

$$y = f_2(\mathbf{W_2} \cdot u) \tag{2.3}$$

Error Backpropagation lässt sich, obwohl hier nur für eine Schicht betrachtet, auf beliebig viele versteckte Schichten erweitern.

Ausgangspunkt beim Training ist stets ein Ausgabefehler E, der für die Abweichung des Netzausgabevektors y zur einem gewünschten Ausgabevektor t steht:

$$E = \sum_i (t_i - y_i)^2 = ||t - y||^2 \tag{2.4}$$

Über die Ableitung des Ausgabefehlers nach den Gewichten in $\mathbf{W_2}$ lässt sich nun ein Gradientenabstieg auf der Fehlerfläche durch Anpassen von

$\mathbf{W_2}$ durchführen:

$$\mathbf{W_2}' = \mathbf{W_2} - \delta \frac{\partial E}{\partial \mathbf{W_2}} \tag{2.5}$$

δ steht für eine positive Lernrate.

Im nächsten Schritt wird nun der Ausgabefehler durch die Gewichtsmatrix $\mathbf{W_2}$ „zurückpropagiert“, um die vorhergehende Gewichtsmatrix $\mathbf{W_1}$ anzupassen.

Dafür wird die Ableitung des Ausgabefehlers nach dem Ausgabevektors betrachtet. Über Verkettung kann nun der Einfluss der Gewichtsmatrix $\mathbf{W_1}$ auf den Ausgabefehler berechnet werden, dementsprechend kann $\mathbf{W_1}$ angepasst werden:

$$\mathbf{W_1}' = \mathbf{W_1} - \delta \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial \mathbf{W_1}} \tag{2.6}$$

Die Gewichtsmatrix $\mathbf{W_2}$ steckt hier in der Ableitung $\partial y / \partial u$. Die Richtung, in welche die Ausgabe der versteckten Schicht u trainiert wird, hängt also vom aktuellen Zustand der Gewichte $\mathbf{W_2}$ ab.

Über die Zeit wird so versucht, eine Konvergenz der Gewichte gegen ein Minimum auf der Fehlerfläche der Netzausgabe zu erreichen. Entscheidender ist an dieser Stelle, ob der Fehler wirklich gegen ein Minimum konvergiert, und mit welcher Geschwindigkeit er dies tut.

Es existieren viele Verbesserungen, die auf diesem grundlegenden Error Backpropagation Algorithmus basieren, wie etwa die Einführung eines Momentum [Rumelhart et al., 1986], Quickprop [Fahlman, 1988], SuperSAB [Tollenaere, 1990] oder RPROP [Riedmiller and Braun, 1993]. Alle diesen Algorithmen basieren jedoch auf dem grundlegenden Error Backpropagation Algorithmus.

Das Training von RNNs gestalltet sich im Vergleich deutlich schwieriger. Ein von Error Backpropagation inspirierter Ansatz ist Back Propagation Through Time (BPTT) [Werbos, 1990]. Ein RNN besteht lediglich aus einer versteckten Schicht u, deren Ausgabe sowohl von der Eingabe, als auch von der vorherigen Ausgabe dieser versteckten Schicht abhängt:

$$u(k+1) = f_1(\mathbf{W_u} \cdot u(k) + \mathbf{W_1} \cdot x(k)) \tag{2.7}$$

Die Netzausgabe ergibt sich zu

$$y(k) = f_2(\mathbf{W_2} \cdot u(k)) \tag{2.8}$$

Da nur eine interne Schicht existiert, lässt sich Error Backpropagation in der gerade vorgestellten Form nicht durchführen. Die Ausgabe des Reservoirs wird jedoch von dessen vorherigen Ausgaben beeinflusst, und auf dieser Grundlage kann Error Backpropagation durchgeführt werden.

Dafür wird die Ausgabe der versteckten Schicht zu verschiedenen Zeitpunkten als jeweils eigenständige Schicht interpretieren. Dabei muss allerdings ein Zeithorizont h definiert werden. Es ergibt sich dann folgender Schichtaufbau:

- 1. Schicht: $x(k)$
- 2. Schicht: $u(k-h)$
- 3. Schicht: $u(k-h+1)$
- $h+2$. Schicht: $u(k-1)$
- $h+3$. Schicht: $u(k)$
- $h+4$. Schicht: $y(k)$

Auf dieser Grundlage kann nun Error Backpropagation durchgeführt werden. Da jede Schicht einen anderen Zeitpunkt des selben Netzes darstellt, spricht man von Backpropagation Through Time.

Im Gegensatz zu FFNs ist jedoch keine Konvergenz garantiert [Jaeger, 2002]. Instabilität ist zudem ein häufiges Problem.

Ein weiterer Ansatz zum Training von RNNs ist Real Time Recurrent Learning (RTRL) [Williams and Zipser, 1989]. Im Gegensatz zu BPTT werden die Ausgabeneuronen als Teil des Reservoirs betrachtet. Dadurch ist es möglich, Rückkopplungen der Ausgabeneuronen auf das Reservoir zu modellieren.

Für jedes Neuron im Reservoir wird zu einem gegebenen Zeitpunkt t der quadratische Fehler, bezogen auf eine gewünschte Ausgabe, berechnet. Da sich im Reservoir auch Neuronen befinden, die nicht trainiert werden sollen, wird deren Fehler mit 0 festgelegt.

Seien u_k, $k \in R$ die Neuronen des Reservoirs und u_k, $k \in T$ diejenigen Neuronen des Reservoirs, die trainiert werden sollen, mit der zugehörigen gewünschten Ausgabe $t_k, k \in T$. Der Ausgabefehler der Reservoirneuronen ergibt sich dann zu:

$$e_k(t) = \begin{cases} u_k(t) - t_k(t) & \text{falls } k \in T \\ 0 & \text{sonst} \end{cases} \tag{2.9}$$

Der quadratische Ausgabefehler des Reservoirs zu:

$$E(t) = \frac{1}{2} \sum_{i \in R} e_i(t)^2 \tag{2.10}$$

Darauf basierend wird der summierte Ausgabefehler über einen Zeitraum $t_0 \leq t \leq t_1$ definiert als:

$$E_T(t_0, t_1) = \sum_{t=t_0}^{t1} E(t) \tag{2.11}$$

Dieser Fehler wird nun minimiert. Der Gradient des Fehlers über den gegebenen Zeitraum $\partial E_T(t_0, t_1)/\partial \mathbf{W}$ ergibt sich durch Aufsummieren der Gradienten der einzelnen Fehler:

$$\frac{\partial E_T(t_0, t_1)}{\partial \mathbf{W}} = \sum_{t=t_0}^{t1} \frac{\partial E(t)}{\partial \mathbf{W}} \tag{2.12}$$

Die Gewichte werden dann entsprechend

$$\mathbf{W}' = \mathbf{W} - \delta \frac{\partial E_T(t_0, t_1)}{\partial \mathbf{W}} \tag{2.13}$$

angepasst.

RTRL ist jedoch sehr rechenaufwendig und daher nicht zum Training großer Netze geeignet.

2.3 Actor Critic Design

Die im letzten Abschnitt vorgestellten Lernalgorithmen für Neuronale Netze setzen als Gemeinsamkeit stets ein Lehrersignal vorraus, auf Basis dessen dann die Gewichte im Netz angepasst werden.

Häufig ist es jedoch so, dass sich Situationen zwar in gut oder schlecht einordnen lassen, jedoch nicht bekannt ist, welche Folge von Aktionen von einer schlechten Situation zu einer gute Situation führt. Neuronale Netze sind in solchen Situationen nicht nutzbar, da ein Lehrersignal, welches letztlich auszuführende Aktionen vorgibt, und auf das das Netz trainiert werden könnte, nicht existiert.

Eine Lösung für dieses Problem stellt Actor-Critic Design (ACD) dar. Im Gegensatz zur Betrachtungsweise eines Agenten als Ganzes, wird dieser in 2 Komponenten unterteilt, dem Aktor und die Kritik. Der schematische Aufbau ist in Abbildung 2.4 dargestellt.

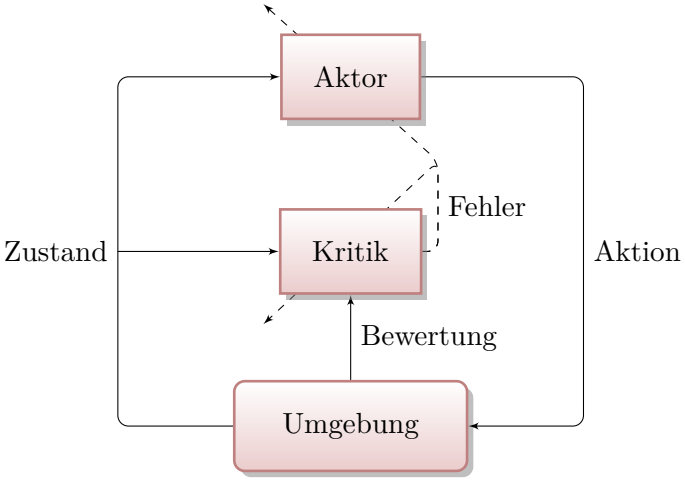


Abbildung 2.4: Grundlegendes Konzept von ACD. Basierend auf dem aktuellen Zustand führt der **Aktor** Aktionen aus. Dieser werden von der **Kritik** anhand der beobachteten Folgen bewertet, und sowohl Aktor als auch Kritik angepasst. Basierend auf [Sutton and Barto, 1998].

Dadurch entfällt die Voraussetzung eines Lehrersignals. Es wird jedoch vorausgesetzt, dass eine Bewertung für die Situation, in der sich der Agent

befindet, existiert.

Die Kritik wird darauf trainiert, diese Bewertung für alle Situationen vorherzusagen. Auf Basis dieses Wissens ist es auch möglich, den Aktor so zu trainieren, dass er möglichst gute Aktionen auswählt.

Die genaue Vorgehensweise an diesem Punkt ist nicht festgelegt, es sind mehrere Ansätze denkbar, den Aktor auf Basis des von der Kritik erworbenen Wissens zu trainieren. Ebenso ist es auch denkbar, die vom Aktor vorgeschlagenen Aktionen als zusätzliche Eingabe für die Kritik zu benutzen, wenn die Aktionen einen direkten Einfluss auf die Bewertung haben.

ACD stellt damit eine Umsetzung von RL dar, die Kritik lässt sich jedoch durch Supervised Learning trainieren, was die Verwendung von Neuronalen Netzen erlaubt.

In der Praxis hat sich gezeigt, dass ACD durchaus gute Ergebnisse liefert und auch in nichtlinearen, nicht statischen und verrauschten Umgebungen funktionieren kann [Prokhorov and Wunsch, 1997].

mathematisches Modell

Es folgt nun eine mathematische Beschreibung von ACD. Ausgangspunkt ist eine Beschreibung der Umgebung eines Agenten über:

$$S(k+1) = F[S(k),A(k),k] \tag{2.14}$$

$S(k)$ repräsentiert den Zustand des Agenten zum diskreten Zeitpunkt k, $A(k)$ die Aktion des Agenten. F beschreibt die Umgebung selbst.

Ausgehend vom aktuellen Zustand $S(k)$ und einer Aktion $A(k)$ des Agenten, ergibt sich im nächsten Zeitschritt ein Folgezustand $S(k+1)$. Weiterhin existiert eine Bewertung $U[S(k), A(k)]$, die sich aus Situation und Aktion zum Zeitpunkt k ergibt.

Auf Basis dieser Bewertung lässt sich eine Abschätzung der künftig zu erwartenden Bewertungen definieren:

$$J[S(k), A(k)] = \sum_{i=0}^{\infty} \gamma^i U[S(k+i),A(k+i)], \tag{2.15}$$

Das Ziel ist, diese Abschätzung zu maximieren. Der Gedanke an dieser Stelle ist, dass eine kurzfristige Maximierung der Bewertung U ebenso zu

einer langfristigen Maximierung führt. Der Zeitrahmen für den „kurzfristig“ an dieser Stelle steht, wird über den „discount-Faktor“ $0 \leq \gamma < 1$ definiert. Je größer γ, desto stärker der Einfluss von zukünftigen U, und desto weiter in die Zukunft reicht die Vorhersage.

Da J von zukünftigen Bewertungen abhängt, lässt es sich nur rückwirkend berechnen. Die Aufgabe der Kritik ist es, J für eine gegebene Situation zu approximieren. Diese Approximation wird im Folgenden $\hat{J}$ bezeichnet. Anschaulich kann dies in etwa wie eine Aussage „wenn ich jener Situation dies tue, wird das positive Folgen haben“ aufgefasst werden. Durch den unscharfen Charakter solcher Aussagen eignen sich Neuronale Netze besonders für diese Aufgabe.

Der Aktor kann sehr frei definiert werden, es ist auch möglich (und bei Neuronalen Netzen als Kritik üblich) die Kritik selbst in den Aktor einzubeziehen.

Kapitel 3

Actor Critic Design mit Echo State Networks

Im letzten Kapitel wurden die Grundlagen des maschinellen Lernens, insbesondere Reinforcement Learning (RL), und die Grundlagen Neuronaler Netze vorgestellt. Ebenso wurden einige der Stärken und Schwächen der vorgestellten Ansätze beleuchtet.

Auf diesen Grundlagen aufbauend folgt nun eine detaillierte Vorstellung der Echo State Network (ESN) Architektur, sowie der Kombination dieser mit ACD. Zudem werden einige Arbeiten vorgestellt, in denen diese Kombination bereits umgesetzt wurde.

3.1 Echo State Networks

Die beiden klassischen Ansätze zur Realisierung Neuronaler Netze, FFNs und RNNs, gehen mit einigen ungelößten Problemen heinher. So existieren für FFNs zwar gute Trainingsalgorithmen, und die Netze lassen sich verhältnismäßig einfach beschreiben und verstehen, jedoch ist es mit diesen nicht möglich dynamische Vorgänge abzubilden.

RNNs auf der anderen Seite sind zwar in der Lage jedwede Art von Dynamik theoretisch nachzubilden, in der Praxis tauchen jedoch ebenso einige gravierende Probleme auf. BPTT ist als Trainingsalgorithmus nicht sehr effektiv, und die damit trainierten Netze neigen zu Instabilitäten. Ebenso ist es schwierig, das Geschehen im Netz selbst nachzuvollziehen, da Dynamiken durch den Trainingsalgorithmus nach belieben entstehen und wieder verschwinden können.

Der Echo State Network (ESN) Ansatz wurde entwickelt, um diesen Problemen zu begegnen. ESNs stellen eine besondere Form von RNNs dar. Dabei werden die Gewichte der internen Neuronen auf sich selbst, im Gegensatz zum allgemeinen RNN, während des Lernens nicht verändert. Das Training geschieht somit einzig über die Ausgabegewichte. [Jaeger, 2002]

Diese Form von RNNs wird auch als Reservoir Computing (RC) bezeichnet. Das bedeutet, es existiert ein Reservoir das eine möglichst große Dynamik beinhaltet. Dieses Reservoir wird nach einer Initialisierung nicht weiter verändert, sondern es wird versucht, einzig über die Veränderung der Interpretation der Dynamiken im Reservoir, in Form der Auslesematrix, die gewünschte Ausgabe zu erzielen. [Jaeger, 2002] [Prokhorov, 2005]

Damit, im Falle von ESNs, das Reservoir ein stabiles Verhalten zeigt, müssen die Gewichte der internen Neuronen auf sich selbst so initialisiert werden, dass sie die Echo State Eigenschaft erfüllen. Diese bedeutet anschaulich, dass das Reservoir ohne Eingabewerte nach einer gewissen Zeit stets wieder in den selben Grundzustand zurückkehrt. Es existieren also keine sich selbst haltenden Zustände im Reservoir. Dadurch hängt der aktuelle Zustand des Reservoirs lediglich von den Eingaben der näheren Vergangenheit ab. [Jaeger, 2002]

Aufbau

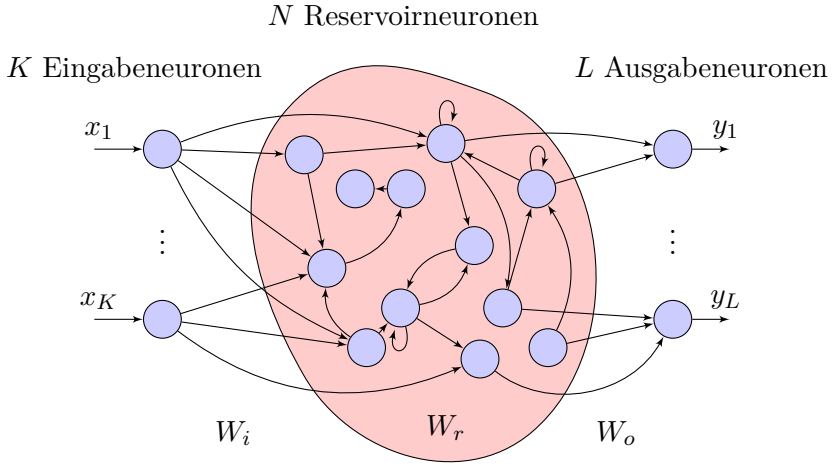


Abbildung 3.1: Schematische Darstellung eines ESN mit allen charakteristischen Parametern.

Ein ESN wird durch die folgenden Parameter charakterisiert:

- Den Ganzzahlen K, N und L, die die Anzahl der Eingabewerte und damit Eingabeneuronen, die Anzahl der Reservoirneuronen, und die Anzahl der Ausgabewerte und damit Ausgabeneuronen beschreiben.
- Eingabevektor $x(k) \in \mathbb{R}^K$, Reservoirausgabevektor $y_r(k) \in \mathbb{R}^N$ und Netzausgabevektor $y(k) \in \mathbb{R}^L$.
- Den Gewichtsmatrizen $\mathbf{W_i}(k) \in \mathbb{R}^{K \times N}$, $\mathbf{W_r}(k) \in \mathbb{R}^{N \times N}$ und $\mathbf{W_o}(k) \in \mathbb{R}^{N \times L}$, die die Gewichte der Eingabeschicht auf das Reservoir, des

Reservoirs auf sich selbst, und des Reservoirs auf die Ausgabeschicht beschreiben.

- Der Ausgabefunktion der Reservoirneuronen $f_r : \mathbb{R} \to \mathbb{R}$ und der Ausgabefunktion der Ausgabenneuronen $f_o : \mathbb{R} \to \mathbb{R}$.

Das Verhalten eines ESNs lässt sich über folgende Gleichungen beschreiben:

$$y_r(k) = f_r(\mathbf{W_i}(k)x(k) + \mathbf{W_r}(k)y_r(k-1)) \quad (3.1)$$

$$y(k) = f_o(\mathbf{W_o}(k)y_r) \quad (3.2)$$

Echo State Eigenschaft

In [Jaeger et al., 2007] wird die Echo State Eigenschaft wie folgt definiert:

Sei $C \subseteq \mathbb{R}^K$ die Menge der möglichen Eingabewerte. So ist die Echo State Eigenschaft genau dann erfüllt, wenn für jede Folge von Eingabewerten $x(k) \in C, k \in \mathbb{N}$ und für 2 beliebige initiale Reservoirausgabevektoren $y_r(0) \in \mathbb{R}^N$ und $y'_r(0) \in \mathbb{R}^N$ eine Folge $\delta(k)$ mit $\lim_{k\to\infty} \delta(k) = 0$ existiert, die

$$||y_r(k) - y'_r(k)||^2 < \delta(k) \;\; \forall k > 0 \quad (3.3)$$

erfüllt.

Anschaulich bedeutet diese Aussage, dass die Reservoirausgabe zweier, in bezug auf die Gewichtsmatrizen identischer, ESNs mit unterschiedlicher Vergangenheit bei gleichen Eingabewerten immer ähnlicher wird. Der Einfluss vergangener Eingaben auf die Reservoirausgabe wird immer geringer, je weiter die Eingaben in der Vergangenheit liegen.

Für Neuronen, die ihre Eingabewerte aufsummieren und eine sigmoide Ausgabefunktion besitzen, wurde beobachtet, dass die Echo State Eigenschaft stets erfüllt ist, wenn der Betrag des betragsmäßig größten Eigenwerts der Gewichtsmatrix $\mathbf{W_r}$ kleiner 1 ist [Jaeger, 2002]. Dieser wird auch als spektraler Radius α bezeichnet. Im Folgenden bezeichne $\rho(\mathbf{M})$ allgemein den spektralen Radius der Matrix $\mathbf{M}$.

Dabei handelt es sich jedoch ausschließlich um eine Beobachtung in der Praxis, für die bisher kein Gegenbeispiel gefunden wurde. Die Aussage selbst ist unbewiesen.

Erzeugung

In diesem Abschnitt soll nun auf praktische Aspekte beim Erstellen von ESNs eingegangen werden. Hierfür müssen die Gewichtsmatrizen $\mathbf{W_i}$ und $\mathbf{W_r}$ mit Werten besetzt werden.

Im Rahmen dieser Arbeit wurden die Einträge über Gleichverteilungen im Intervall $[-1;1]$ erzeugt. Der üblichen Praxis folgend wurden dabei nicht alle Einträge besetzt, sondern nur ein gewisser Anteil. Die restlichen Einträge wurden auf 0 gesetzt und stehen für nicht vorhandene Verbindungen. [Jaeger, 2002]

Der absolute Wertebereich der Gleichverteilung spielt für die Gewichtsmatrix $\mathbf{W_r}$ keine Rolle. Um einen bestimmten spektralen Radius α zu erreichen, und somit die Echo State Eigenschaft zu erfüllen, wird $\mathbf{W_r}$ erst mit den entsprechenden Zufallswerten besetzt, und dann so skaliert, dass der gewünschte spektrale Radius erreicht wird.

Es wird also zuerst eine Matrix $\mathbf{W_r}'$ erstellt, in der ein Anteil von c_{dr} Einträgen auf Zufallswerte einer Gleichverteilung in $[-1;1]$ gesetzt wird. $\mathbf{W_r}$ ergibt sich dann für ein festgelegtes α über:

$$\mathbf{W_r} = \frac{\alpha}{\rho(\mathbf{W_r}')} \cdot \mathbf{W_r}' \tag{3.4}$$

Die gesetzten Gewichte in $\mathbf{W_r}$ entsprechen somit einer Gleichverteilung in $[-\alpha/\rho(\mathbf{W_r}')\,;\alpha/\rho(\mathbf{W_r}')]$. $\mathbf{W_r}$ ist gegenüber einer Skalierung von $\mathbf{W_r}'$ invariant, da diese ebenso die Eigenwerte von $\mathbf{W_r}'$ und damit $\rho(\mathbf{W_r}')$ betreffen würde.

$\mathbf{W_i}$ wird ähnlich erstellt wie $\mathbf{W_r}'$. Der Parameter c_i steht für die Verbindungsdichte. Der Wertebereich der Gewichte spielt hier jedoch eine entscheidende Rolle. Eine Veränderung des Wertebereichs dieser Gewichte ist äquivalent zu einer Änderung des Wertebereichs aller Eingabewerte. Hohe Eingabewerte führen zu einer stärkeren Aktivierung und somit Sättigung der Neuronen im Reservoir, als niedrige. Deren Verhalten nimmt dadurch einen weniger linearen Charakter an.

Die Eingabewerte des ESNs wurden im Verlauf dieser Arbeit nicht weiter verändert, stattdessen wurde $\mathbf{W_i}$ mit dem Parameter r skaliert. Es wird also, analog zu $\mathbf{W_r}$, eine Rohmatrix $\mathbf{W_i}'$ erstellt, in der der Anteil c_i an

Einträgen auf Werte einer Gleichverteilung im Intervall $[-1; 1]$ gesetzt wird. $\mathbf{W_i}$ ergibt sich nun über

$$\mathbf{W_i} = r \cdot \mathbf{W_i}' \tag{3.5}$$

aus $\mathbf{W_i}'$.

So bleiben schließlich folgende Freiheitsgrade für die Erzeugung eines ESNs in einer definierten Umgebung, also bei vorgegebener Anzahl an Ein- und Ausgängen:

- Spektraler Radius α der Gewichtsmatrix $\mathbf{W_r}$
- Wertebereich r der Gewichte in der Eingabematrix $\mathbf{W_i}$
- Verbindungsdichte c_i der Eingabematrix
- Verbindungsdichte c_{dr} der Rückkopplungsmatrix
- Anzahl der Neuronen N im Reservoir
- Ausgabefunktion der Reservoir- und Ausgabeneuronen f_r und f_o

Die Ausgabewerte der Neuronen im Reservoir werden mit 0 initialisiert. Aufgrund der Echo State Eigenschaft hat dies nur einen geringfügigen Einfluss auf das weitere Netzverhalten. Durch Anlegen konstanter und realistischer Eingabewerte findet das Netz nach einiger Zeit in seinen eigenen Ruhezustand.

Training

Im Gegensatz zu allgemeinen RNNs werden bei ESNs nur die Ausgabegewichte trainiert. Dieser Ansatz vereinfacht das Training enorm, und vermeidet gleichzeitig die Schwachpunkte von Trainingsalgorithmen wie BPTT.

Die Trainingsalgorithmen für ESNs lassen sich in die Kategorie der Online- und Offlineverfahren einteilen. Beim Online-Training werden die Ausgabegewichte jeweils nur auf einen Ausgabewert trainiert, die Generalisierung von vielen Lerndaten ergibt sich aus der Summe der einzelnen Anpassungen. Im Gegensatz dazu wird beim Offline-Training ein kompletter Datensatz von Ausgabewerten auf einmal gelernt.

Im Gegensatz zu Lernverfahren für FFNs wird bei ESNs nicht die Assoziation von Ein- und Ausgabewert gelernt, sondern von Reservoirausgabe und Ausgabewert, da identische Eingabewerte je nach Situation zu unterschiedlichen Reservoirausgaben, und auch unterschiedlichen Netzausgaben, führen können und sollen.

Offline Learning

Neben den klassischen Offline Lernalgorithmen, bei denen die Anpassungen der Auslesematrix $\mathbf{W_o}$ für viele Lernschritte aufsummiert werden, lässt sich für ESNs ein sehr einfacher Offline-Lernalgorithmus herleiten, der einen Least Mean Squared Error Fit durchführt. [Jaeger, 2002]

Dafür wird zuerst eine Matrix von Reservoirausgaben M und zugehörigen Lerndaten T betrachtet:

$$\mathbf{M} = (y_r(1), \ldots, y_r(n)) \tag{3.6}$$
$$\mathbf{T} = (t(1), \ldots, t(n)) \tag{3.7}$$

Die mittlere quadratische Abweichung ist gegeben durch:

$$E_{MSE} = \sum_{i=1}^{n} (t(i) - y(i))^2 \tag{3.8}$$

Mittels dem Moore-Penrose Pseudoinversen der Matrix $\mathbf{M}$ lässt sich nun die Auslesematrix über

$$\mathbf{W_o} = \mathbf{M}^{-1} \cdot f_o^{-1}(\mathbf{T}) \tag{3.9}$$

so bestimmen, dass E_{MSE} den kleinstmöglichen Wert annimmt. Es müssen hierfür stets mindestens soviele Trainingsdaten vorhanden sein, wie Neuronen im Reservoir. [Jaeger, 2002]

Da $\mathbf{W_o}$ komplett neu besetzt wird, gehen vorherige Ergebnisse verloren. Ein progressiver Lernfortschritt kann einzig dadurch erreicht werden, indem schrittweise immer mehr Trainingsdaten gesammelt werden. Da hierfür das Pseudoinverse von $\mathbf{M}$ in jedem Schritt neu berechnet werden muss, ist diese Methode sehr rechenintensiv.

Ein weiteres Problem stellt das übertrieben genaue Nachbilden von Trainingsdaten dar. Dadurch ergeben sich streckenweise extrem große Gewichte in $\mathbf{W_o}$, die für nicht explizit trainierte Situationen zu unvorhersehbaren Netzausgaben führen können, und im Allgemeinen mit einer sehr schlechten Generalisierung einhergehen.

Es ist daher nötig, sehr viele Trainingsdaten zusammeln die möglichst alle Eingabewerte abdeckende.

Online Learning

Ein verhältnismäßig einfaches Lernverfahren stellt Online-Learning dar. Ausgangspunkt ist der quadratische Fehler des ESNs bezogen auf eine bestimmte, erwartete Ausgabe $t(k)$:

$$E(k) = ||t(k) - y(k)||^2 \tag{3.10}$$

Im nächsten Schritt wird nun der Einfluss von jedem Gewicht der Ausgabematrix $\mathbf{W_o}$ auf diesen Fehler berechnet. Mit dieser Information kann nun ein Gradientenabstieg auf der Fehlerfläche durchgeführt werden.

Dabei ergibt sich über die Kettenregel folgende Ableitung:

$$\frac{\partial E(k)}{\partial \mathbf{W_o}(k)} = \frac{\partial E(k)}{\partial y(k)} \cdot \frac{\partial y(k)}{\partial \mathbf{W_o}(k)} \tag{3.11}$$

Aus

$$\frac{\partial E(k)}{\partial y(k)} = -2 \cdot (t(k) - y(k)) \tag{3.12}$$

und

$$\frac{\partial y(k)}{\partial \mathbf{W_o}(k)} = \mathbf{W_o}(k) \cdot f_o(\mathbf{W_o}(k) \cdot y(k)) \tag{3.13}$$

folgt dann

$$\frac{\partial E(k)}{\partial \mathbf{W_o}(k)} = -2 \cdot (t(k) - y(k)) \cdot \mathbf{W_o}(k) \cdot f_o(\mathbf{W_o}(k) \cdot y(k)) \tag{3.14}$$

Anschaulicher ist mitunter die komponentenweise Betrachtung:

$$\frac{\partial E(k)}{\partial (\mathbf{W_o}(k))_{ij}} = \frac{\partial E(k)}{\partial (y(k))_j} \cdot \frac{\partial (y(k))_j}{\partial (\mathbf{W_o}(k))_{ij}} \tag{3.15}$$

$$\frac{\partial E(k)}{\partial (\mathbf{W_o}(k))_{ij}} = -2 \cdot [(t(k))_j - (y(k))_j] \cdot (y_r)_i \cdot \left(f_o'(\mathbf{W_o} y_r)\right)_j \tag{3.16}$$

Mittels der nun errechneten Ableitung $\partial E(k)/\partial \mathbf{W_o}(k)$ wird die Auslesematrix über folgende Anpassungsregel trainiert:

$$\mathbf{W_o}(k+1) = \mathbf{W_o}(k) - \delta \frac{\partial E(k)}{\partial \mathbf{W_o}(k)} \tag{3.17}$$

δ stellt eine positive Lernrate dar, die die Geschwindigkeit, mit der diese Anpassung stattfindet, vorgibt.

Die Lernrate hat einen großen Einfluss auf den Lernerfolg. Hohe Lernraten können dazu führen, dass Minima auf der Fehlerfläche übersprungen werden, und der Fehler letztlich um dieses Minimum oszilliert, ohne es zu erreichen. Ebenso kann eine große Lernrate jedoch auch zum Überspringen kleiner lokaler Minima führen wodurch ein globales Minimum gefunden wird, und das Lernergebnis somit signifikant verbessern.

Eine optimale Lernrate zu finden ist ein schwieriger Prozess, häufig ist es erforderlich die Ergebnisse individuell zu deuten.

3.2 Kombination mit Actor Critic Design

Bei der Kombination von ACD mit ESNs ergeben sich einige Unterschiede zu der allgemeinen Variante von ACD. Einige mögliche Umsetzungen werden in [Prokhorov and Wunsch, 1997] beschrieben, welche die Grundlage des in dieser Arbeit verwendeten Designs darstellen.

In [Prokhorov and Wunsch, 1997] wird zwischen den 3 Ansätzen Heuristic Dynamic Programming (HDP), Direct Heuristic Programming (DHP) und Globalized Dual Heuristic Programming (GDHP) unterschieden. HDP stellt die einfachste Form der Realisierung von ACD dar, weshalb es nun im Folgenden eingehender betrachtet wird.

Der schematische Aufbau ist in Abbildung 3.2 dargestellt. Bei HDP erfolgt die Vorhersage von $\hat{J}$ direkt durch das Critic Network. Dieses wird vom Aktor benutzt, um die Änderung der erwarteten Bewertung $\hat{J}$ in Abhängigkeit des Zustandes S, $\partial\hat{J}/\partial S$, zu berechnen, und auf Basis dessen eine Aktion auszuwählen. Basierend auf der Abweichung zwischen der beobachteten Bewertung U, und der Vorhersage $\hat{J}$, wird das Critic Netzwerk trainiert.

Das Kritik Netz approximiert die zukünftige zu erwartende Bewertung $J(k)$, die definiert ist durch:

$$J(k) = U(k) + \sum_{i=1}^{\infty} \gamma^i U(k+i) = \sum_{i=0}^{\infty} \gamma^i U(k+i) \tag{3.18}$$

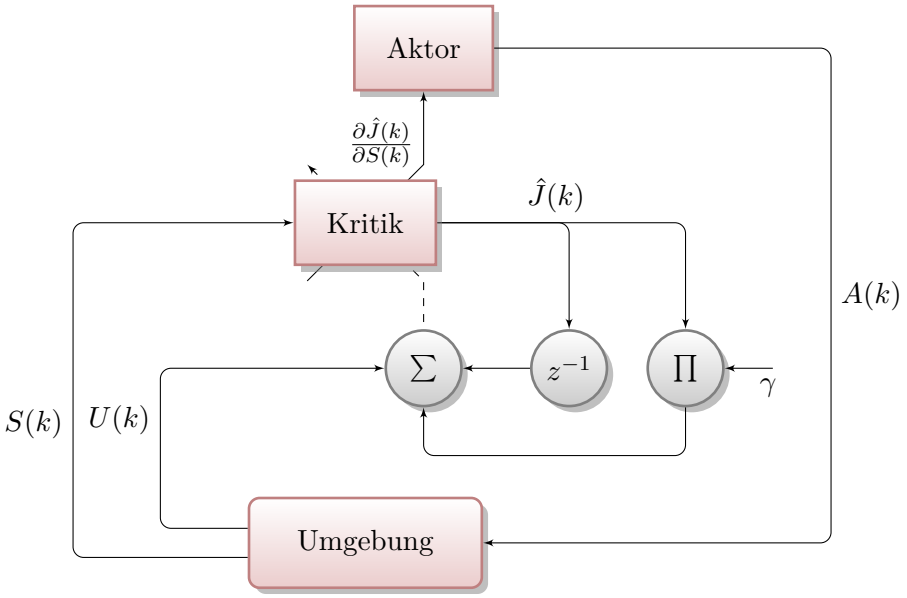


Abbildung 3.2: Schematische Darstellung des HDP Ansatzes nach [Prokhorov and Wunsch, 1997].

Aus dem Zusammenhang

$$\begin{aligned} J(k) - U(k) - \gamma J(k+1) &= \sum_{i=0}^{\infty} \gamma^i U(k+i) - U(k) - \gamma \sum_{i=0}^{\infty} \gamma^i U(k+1+i) \\ &= \sum_{i=0}^{\infty} \gamma^i U(k+i) - U(k) - \sum_{i=1}^{\infty} \gamma^i U(k+i) \\ &= \sum_{i=0}^{\infty} \gamma^i U(k+i) - \sum_{i=0}^{\infty} \gamma^i U(k+i) \\ &= 0 \end{aligned}$$

lässt sich der Fehler der Approximation des Kritik Netzes herleiten. Ein Fehler in der Vorhersage von $J(k)$ führt in dieser Gleichung zu einem Ergebnis ungleich 0. Damit ergibt sich der quadratische Vorhersagefehler zu:

$$E(k) = \left[\hat{J}(k-1) - U(k-1) - \gamma \hat{J}(k)\right]^2 \tag{3.19}$$

Mit diesem Fehler kann nun das Kritik Netzwerk trainiert werden, etwa

nach einem Online Ansatz über:

$$\mathbf{W_o}(k+1) = \mathbf{W_o}(k) - \delta \frac{\partial E(k)}{\partial \mathbf{W_o}} \tag{3.20}$$

Aufgabe des Aktors ist es, Aktionen zu ermitteln, die den Zustand so verändern, dass sich die Bewertung verbessert. Dafür wird das von der Kritik erworbene Wissen benutzt. Sind genug Informationen bekannt, kann $\partial S/\partial A$ dire tberechnet werden, um dann mittels $\partial \hat{J}/\partial S$ über Verkettung einen Gradientenaufstieg auf $\hat{J}$ durchzuführen. Ist $\partial S/\partial A$ hingegen nicht bekannt, kann dies ebenso approximiert werden.

Als Erweiterung von HDP kann DHP angesehen werden. Dabei wird von der Kritik $\partial J/\partial S$ direkt vorhergesagt. Dies hat den Vorteil, dass es eine genauere Abbildung von $\partial J/\partial S$ erlaubt, als im HDP Ansatz möglich. Anschaulich kann dies darüber erklärt werden, dass die Kritik nicht versucht, J in seiner Gesamtheit vorherzusagen, sondern über die einzelnen Aspekte des Zustandes S, die J beeinflussen. Der Vorteil von DHP entsteht also beim Training der Kritik, welches sich dadurch jedoch auch komplexer gestaltet.

GDHP schließlich vereint die beiden Ansätze HDP und DHP. Die Kritk sagt sowohl J, als auch $\partial J/\partial S$ vorher. Beide Informationen werde schließlich benutzt, um eine Aktion auszuwählen. Die in [Prokhorov and Wunsch, 1997] durchgeführten Simulationen zeigen nur eine sehr geringfügige Verbesserung von GDHP im Vergleich zu DHP, der Aufwand zur Umsetzung von GDHP ist jedoch ungleich höher.

In [Prokhorov and Wunsch, 1997] werden zusätzlich aktionsabhängige Varianten der hier diskutierten Realisierungen HDP, DHP und GDHP vorgestellt.

Der Unterschied besteht darin, dass die Eingabe der Kritik nur aus dem aktuellen Zustand $S(k)$ besteht, sondern zusätzlich aus der vom Agenten gewählten Aktion $A(k)$. Diese zusätzliche Information ist nötig, wenn die Aktion des Agenten einen direkten Einfluss auf die Bewertung hat. Damit ist die Kritik in der Lage, diesen Einfluss ebenso vorherzusagen. In [Prokhorov and Wunsch, 1997] werden diese Varianten als Action Dependant HDP (ADHDP), Action Dependant DHP (ADDHP) und Action Dependant GDHP (ADGDHP) bezeichnet.

3.3 bisherige Arbeiten

Im Folgenden sollen nun 2 Arbeiten vorgestellt werden, in denen ACD, mit einem ESN nur Realisierung der Kritik, zur Steuerung eines mobilen Roboters bereits umgesetzt wurde. In beiden Arbeiten wurde die Kombination von ACD und ESNs mit HDP bzw. ADHDP realisiert.

In [Oubbati et al., 2011] wurde untersucht, in wie weit sich mit dem genannten Konzept eine Kollisionsvermeidung realisieren lässt.

Der Agent wurde periodisch auf ein Hindernis zu- und wieder wegbewegt (Abbildung 3.3). Nach einigen Episoden sollte der Agent lernen, dieses Hindernis vorherzusagen und durch Lenkbewegungen dem Hindernis und damit einer schlechten Bewertung auszuweichen.

Dem ESN wurden Zustand und Aktion des Agenten als Eingabe übergeben, dieses soll aus diesen Informationen eine Vorhersage über den zukünftig zu erwartenden Bewertung $\hat{J}$ erzeugen. Das Netz wird fortwährend durch einen Online Trainingsalgorithmus trainiert.

Eine einfache Action Policy wurde schließlich benutzt, um mit Hilfe der Kritik eine Aktion zu ermitteln, die zu einer Maximierung von $\hat{J}$ führen soll.

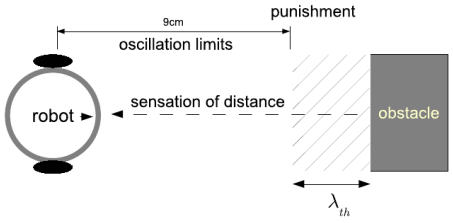


Abbildung 3.3: Der Agent wird periodisch auf das Hindernis zu- und wieder wegbewegt.

Der Zustand $S(k)$ des Agenten ist gegeben durch seine Position x und y, sowie den Ausgabewerten seiner 8 Nährungssensoren $d_1 \dots d_8$. Diese entsprechen jeweils der Entfernung zum nächsten Hindernis, das durch den jeweiligen Sensor erfasst wird.

$$S(k) = \{x(k), y(k), d_i(k) | i = 1,2,\dots,8\} \tag{3.21}$$

Die Bewertung eines Zustandes orientiert sich an der Ausgabe dieser Nährungssensoren. Unterschreitet ein Hindernis eine gewisse Entfernung λ_{th}, so resultiert eine schlechte Bewertung:

$$r_i(k) = \begin{cases} d_i(k) - \lambda_{th} & \text{wenn } d_i(k) < \lambda_{th} \\ 0 & \text{sonst} \end{cases} \tag{3.22}$$

$$U(k) = \sum_{i=1}^{8} r_i(k) \tag{3.23}$$

Der Roboter besitzt einen einfachen differenziellen Antrieb, dessen Wirkung sich über die Radwinkelgeschwindigkeiten, Raddurchmesser sowie der Entfernung vom Mittelpunkt ergibt.

Der Agent steuert den Roboter, indem er eine Lenkgeschwindigkeit ω vorgibt. $\omega > 0$ bedeutet, dass der Roboter nach links lenkt, $\omega < 0$ entsprechend nach rechts. Die Aktion entspricht damit $A(k) = \omega(k)$.

Eine Action Policy bestimmt nun mittels des ESNs diese Aktion:

$$A(k+1) = A(k) + \delta \frac{\partial \hat{J}(k)}{\partial A(k)} \tag{3.24}$$

δ entspricht einer Art Anpassungsrate.

Der Versuch fand in 2 Teilen statt. Im ersten Teil wurde der Roboter fremdgesteuert und oszillierend auf die Wand zu und wieder von ihr weg bewegt. Bei Annährung wurde eine negative Bewertung produziert, diese sollte das ESN vorhersagen.

Im zweiten Teil wurde der Roboter vom Agenten selbst gesteuert. In der Ausgangsposition war dieser gerade auf die Wand ausgerichtet, der Roboter fuhr nun los und wurde bei Erreichen der Wand wieder auf die Ausgangsposition zurückgesetzt. Nach einigen Episoden fing der Agent an der Wand auszuweichen, bevor eine negative Bewertung erzeugt wurde. Augenscheinlich hat der Agent die zu erwartende negative Bewertung durch die nahende Wand antizipiert und entsprechend reagiert.

Der selbe Agent wurde in [Koprinkova et al., 2010] untersucht. Nach einer anfänglichen Trainingsphase wurde dieser so in die Lage versetzt, den durch die Nährungssensoren entdeckten Hindernissen so auszuweichen, dass sich

der Roboter von diesen Entfernt. Dadurch vermeidet der Agent weitere Bestrafung.

In Abbildung 3.4 und Abbildung 3.5 sind die erzielten Resultate dargestellt.

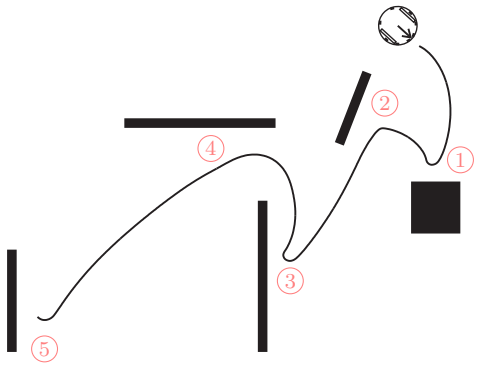


Abbildung 3.4: Gefahrene Strecke des Roboters in [Koprinkova et al., 2010]. Hindernissen wird, sobald von den Sensoren erfasst, erfolgreich ausgewichen.

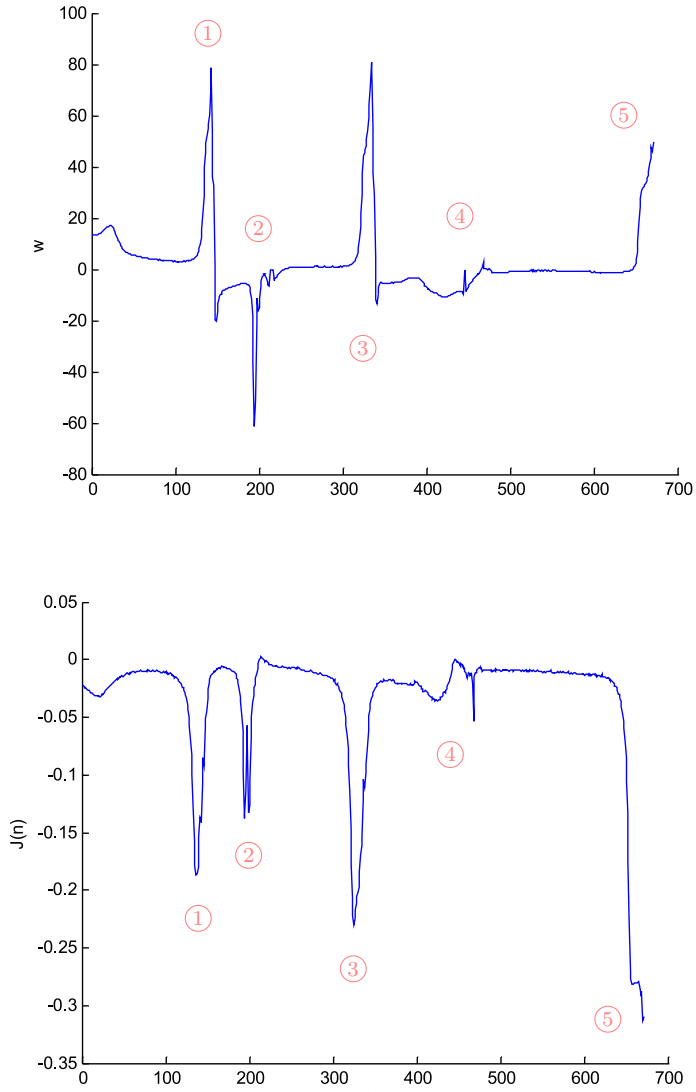


Abbildung 3.5: Lenkbewegungen und Bewertung des Agenten, die Situationen aus Abbildung 3.4 sind gekennzeichnet. Basierend auf [Koprinkova et al., 2010].

Kapitel 4

Umsetzung

In den letzten Kapiteln erfolgte eine allgemeine Einführung in die Themen maschinelles Lernen und neuronale Netze, sowie eine genauere Betrachtung der Kombination von ACD mit ESNs.

In diesem Kapitel folgt nun eine genaue Beschreibung der verwendeten Umsetzung dieser Konzepte, sowie der Rahmenbedingungen.

4.1 Überblick

Die im letzten Kapitel vorgestellten Ergebnisse zeigen, dass ACD mit einem ESN als Kritik prinzipiell auch in der Praxis funktioniert. Die in den vorgestellten Arbeiten umgesetzte Aufgabenstellung war jedoch verhältnismäßig einfach.

Daher soll im Rahmen dieser Arbeit das vorgestellte Konzept auf eine schwierigere Aufgabe übertragen werden. In den vorgestellten Ergebnissen bestand die Aufgabe des Agenten darin, den von einem Roboter erkannten Hindernissen auszuweichen um Kollisionen zu vermeiden.

In dieser Arbeit soll nun stattdessen eine Karte der Umgebung des Roboters erstellt werden. Hierauf aufbauend könnten später komplexere Dinge realisiert werden, z.B. könnten die in [Oubbati et al., 2011] vermiedenen Hindernisse kartiert werden, um diesen anschließend noch vor der eigentlichen Wahrnehmung ausweichen zu können.

Dabei ist diese Umgebung im Verlauf des Versuchs einer Veränderung unterworfen, der Agent soll in der Lage sein, mit diesen Veränderungen umzugehen. Zudem ist sowohl der Zustandsraum, als auch der Aktionsraum des Agenten kontinuierlich.

Folgend werden nun die genauen Rahmenbedingungen dieser Versuche beschrieben.

4.2 E-Puck Roboter

Im Verlauf der Arbeit wird der von der Ecole Polytechnique Fédérale de Lausanne entwickelte E-Puck Roboter benutzt.

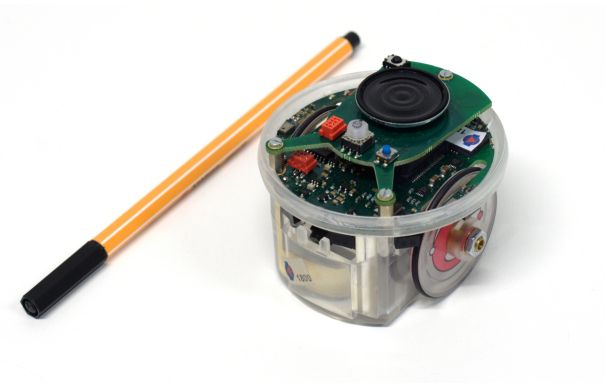


Abbildung 4.1: E-Puck Roboter

Dieser Roboter wird über 2 Schrittmotoren angetrieben, die auf gummierte Räder links und rechts des Roboters wirken. Den Motoren wird eine konstante Winkelgeschwindigkeit ω_l und ω_r vorgegeben, aus denen sich die Vorwärtsgeschwindigkeit und Rotation ergibt.

Die Odometrie beinhaltet 2 Konstanten, den jeweiligen Abstand der Räder vom Mittelpunkt des Roboters r sowie den Raddurchmesser d. Die Bewegung des Roboters, ausgedrückt in Vorwärtsgeschwindigkeit v und Winkelgeschwindigkeit ω ergibt sich dann folgendermaßen:

$$v = d\frac{\omega_r + \omega_l}{2} \tag{4.1}$$

$$\omega = d\frac{\omega_r - \omega_l}{2r} \tag{4.2}$$

Position und Ausrichtung entsprechend zu:

$$x(t) = \int_0^t \cos(\phi(\tau)) \cdot v(\tau)\, \mathrm{d}\tau \tag{4.3}$$

$$y(t) = \int_0^t \sin(\phi(\tau)) \cdot v(\tau)\, \mathrm{d}\tau \tag{4.4}$$

$$\phi(t) = \int_0^t \omega(\tau)\, \mathrm{d}\tau \tag{4.5}$$

Da die ersten Resultate bezüglich der Genauigkeit der Odometrie vergleichsweise schlecht ausfielen, wurde ein Modell möglicher Abweichungen entwickelt, und der Roboter entsprechend diesem Modell kalibriert. Dabei wurden

für die beiden Räder verschiedene Durchmesser d_l und d_r angenommen und experimentell ermittelt. Ebenso wurde der genaue Abstand der Räder zum Mittelpunkt r experimentell ermittelt.

Damit ergibt sich folgendes Modell der Bewegung des E-Puck:

$$v = \frac{\omega_r \cdot d_r + \omega_l \cdot d_l}{2} \tag{4.6}$$

$$\omega = \frac{\omega_r \cdot d_r - \omega_l \cdot d_l}{2r} \tag{4.7}$$

Zur Wahrnehmung seiner Umgebung verfügt der Roboter über 8 Infrarotsensoren. Diese Sensoren liefern bei einer Annährung an ein Objekt durch die Reflektion von ausgestrahltem Infrarotlicht ein Signal, funktionieren jedoch nur auf kurze Distanz und werden von vorhandenen weiteren Lichtquellen beeinflusst.

Sie kommen im Verlauf dieser Arbeit nicht zum Einsatz, bieten aber viele Ansatzpunkte für zukünfige Aufgabenstellungen.

Kommunikation

Die Kommunikation mit dem E-Puck Roboter findet über Bluetooth statt, dabei wird eine serielle Schnittstelle mit 115kbps emuliert. Die Steuerung erfolgt mittels einfacher ASCII Befehle, der E-Puck liefert entsprechend Antworten zurück. Eine Übersicht der für diese Arbeit relevanten Befehle findet sich in Tabelle 4.1.

Leider stellt die Firmware im E-Puck Roboter keinen Befehl zum Auslesen der Schrittposition der Räder und gleichzeitigem Neusetzen der Raddrehzahlen zur Verfügung, was zu einer gewissen, nicht vermeidbaren Ungenauigkeit in der Odometrie führt. Dies rührt daher, dass der Roboter beim Festlegen einer neuen Geschwindigkeit noch eine unbekannte Strecke mit der alten Geschwindigkeit weiterfährt.

Eine weitere Einschränkung besteht in dem beschränkten Wertebereich der Schrittposition. Es war daher nötig eine Überlauferkennung umzusetzen.

Dies rührt daher, dass nach einer Veränderung der Radgeschwindigkeiten nicht genau zu ermitteln ist, wieviele Schritte die Räder noch mit der alten Geschwindigkeit zurückgelegt haben.

Befehl	Bedeutung
D,[ω_l],[ω_r]	Legt die Geschwindigkeit für das linke und rechte Rads fest, Ganzzahlen im Bereich 0 bis 1000
L,[N],[Z]	Setzt LED N auf den Zustand Z (0=aus, 1=an, 2=invertieren)
P,[c_l],[c_r]	Setzt die Schrittzähler der Räder auf die entsprechenden Werte, Ganzzahl im Bereich -32768 bis 32767. 1000 steht für eine Umdrehung.
Q	Gibt die Werte der Schrittzähler für die Räder zurück.
N	Gibt die Werte der 8 Infrarot-Nährungssensoren zurück.

Tabelle 4.1: Befehle zur Steuerung des E-Puck Roboters. Diese werden in einer einfachen ASCII Darstellung übermittelt.

4.3 Umgebung

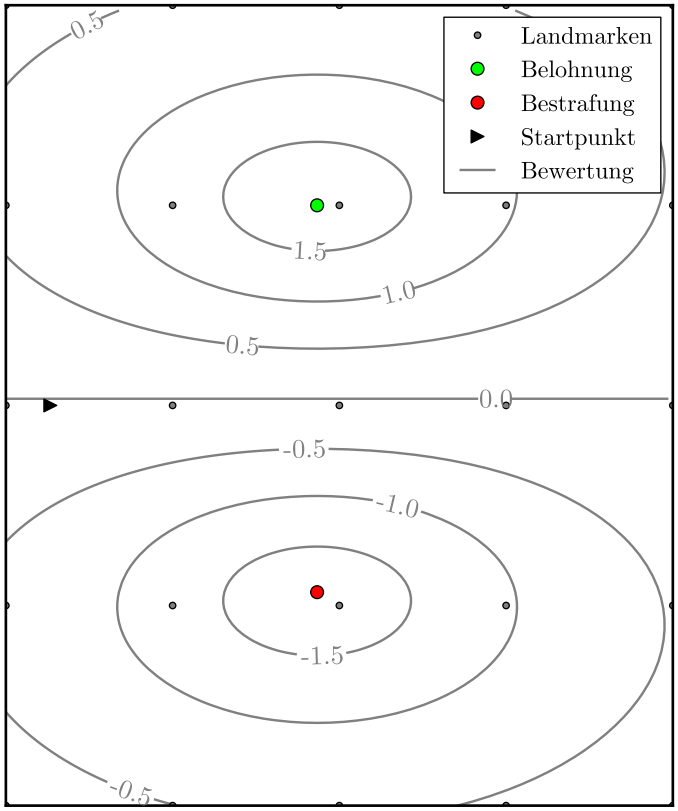


Abbildung 4.2: Die Versuchsumgebung. Dargestellt ist ein Belohnung generierender Punkt, ein Bestrafung generierender Punkt sowie die sich über die Entfernung zu diesen ergebende Bewertung für jede Position aus der Vogelperspektive.

Alle Versuche finden in der durch Abbildung 4.2 dargestellten Umgebung statt. Diese besteht aus einem ebenen Untergrund auf dem sich der Agent frei bewegen kann, und wird physikalisch nicht begrenzt. Der Agent kann die Umgebung also verlassen, in diesem Fall endet jedoch der Versuchsabschnitt. Innerhalb dieser Umgebung befinden sich diverse virtuelle Objekte wie Belohnung und Bestrafung, auf die der Agent reagieren soll. Sie besitzen keine physikalische Entsprechung, und können vom Roboter auch ohne Folgen überfahren werden.

Innerhalb dieser Umgebung befinden sich 25 virtuelle Landmarken, gleichmäßig verteilt. Der Agent nimmt die Entfernung zu jeder einzelnen wahr. Dies ist die einzige Möglichkeit des Agenten, seine Position innerhalb der Versuchsumgebung wahrzunehmen.

Die Motivation zu dieser Art der Positionsbestimmung bestand darin, dass ein ESN besser mit dieser Repräsentation der Position des Agenten zurechtkommt, als mit der abstrakten Beschreibung durch x und y Koordinaten. Große Nähe zu einer bestimmten Landmarke kann z.B. einfacher mit einem bestimmten Verhalten assoziiert werden, als ein besimmter Wertebereich für x und y.

Ebenso wäre das Netzverhalten sehr vom Wertebereich für x und y abhängig, niedrige Werte würden zu einem sehr linearen Verhalten führen, wohingegen große Werte zu einem sehr nichtlinearen Verhalten führen würden. Dies würde bedeuten, dass sich diese grundsätzliche Netzeigenschaft in Abhängigkeit der Entfernung des Agenten vom Koordinatenursprung verändern würde, was jedoch nicht gewünscht ist.

Die Landmarken sind charakterisiert durch ihre Position (x, y) und werden in der Menge

$$L = \{l_1, l_2, \ldots, l_n \,|\, l_i = (x_i, y_i)\} \tag{4.8}$$

zusammengefasst. Daraus wird eine Repräsentation der Entfernungen der einzelnen Landmarken errechnet:

$$e_i = \frac{1}{d \cdot ||l_i - p||^2} \tag{4.9}$$

Bei d handelt es sich um eine Konstante, die die „Sichtbarkeit“ der Landmarken bestimmt. Es gilt $0 < e_i \leq 1$, ein hoher Wert signalisiert Nähe zur entsprechenden Landmarke.

Diese Repräsentation der Entfernungen der Landmarken wird nun schließlich zum Vektor

$$e = \begin{pmatrix} e_1 \\ \vdots \\ e_n \end{pmatrix} \tag{4.10}$$

zusammengefasst.

Ermittelt wird der Vektor e über die Odometrie des Roboters. Es sind keine Sensoreingaben mit den Landmarken assoziiert, dies wäre jedoch eine denkbare Erweiterung.

Zusätzlich befindet sich in der Versuchsumgebung ein Belohnung generierender Punkt p_+ und ein Bestrafung generierender Punkt p_-. Jeder Position innerhalb der Versuchsumgebung wird eine Bewertung zugeordnet, die von der Entfernung zur Belohnung und Bestrafung abhängt. Diese ergibt sich, in ähnlicher Form wie bei den Landmarken, für die Position p zu:

$$U(p) = \frac{2}{1 + 6 \cdot ||p - p_+||^2} - \frac{2}{1 + 6 \cdot ||p - p_-||^2} \tag{4.11}$$

4.4 Agent

Der verwendete Agent basiert auf dem in Kapitel 3.2 vorgestellten HDP Ansatz aus [Prokhorov and Wunsch, 1997]. Ein ESN wird als Kritik verwendet.

Da die Aktion selbst keinen Einfluss auf die Bewertung hat, wurde auf ADHDP verzichtet. Die Eingabe des ESN besteht also lediglich aus dem Zustand des Agenten, $S(k) = e(k)$. Abbildung 4.3 zeigt den schematischen Aufbau.

Basierend auf dem aktuellen Zustand nimmt das ESN eine Vorhersage der zukünftigen, zu erwartenden Bewertungen vor. Da die Bewertung U in Abhängigkeit des Zustandes S stetig ist, und sich daher eine langfristige Veränderung in U durch eine kurzfristige Veränderung ankündigt, erschien die Verwendung des discount-Faktors γ überflüssung. Dieser wurde daher auf $\gamma = 0$ festgelegt. Das ESN sagt somit die direkt folgende Bewertung vorher. Als Ausgabefunktion der Reservoirneuronen wurde $f_r(x) = \tanh(x)$ gewählt, die Ausgabeschicht besteht aus linearen Neuronen ($f_o(x) = x$).

Auf Basis dieser Vorhersage wird mathematisch die Veränderung der Bewertung, in Abhängigkeit der Veränderung des Zustandes $\partial\hat{J}/\partial S$, bestimmt. Darauf basierend bestimmt eine Action Policy schließlich die auszuführende Aktion über:

$$\frac{\partial \hat{J}}{\partial \omega} = \frac{\partial \hat{J}}{\partial e} \cdot \frac{\partial e}{\partial (x, y)^T} \cdot \frac{\partial (x, y)^T}{\partial \omega} \tag{4.12}$$

$\partial\hat{J}/\partial e$ wird aus dem ESN gewonnen, die restlichen Ableitungen ergeben sich gemäß der Odometrie. Die Action Policy führt auf der erwarteten Bewertung

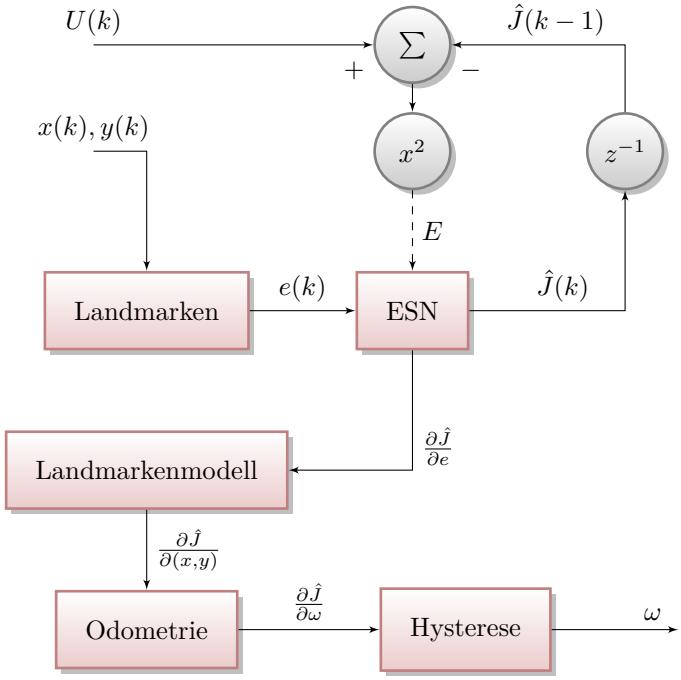


Abbildung 4.3: Schematischer Aufbau des Agenten.

$\hat{J}$, über die Auswahl entsprechender Aktionen ω, einen Gradientenaufstieg durch. Da lediglich die Ableitung von $\hat{J}$ benutzt wird, spielt der absolute Wertebereich der Ausgabe des ESNs für die Auswahl der Aktion keine Rolle. Lediglich Veränderungen der Netzausgabe durch Veränderungen von S sind entscheidend.

Das vorgegebene ω wird nun in eine Bewegung des Roboters umgesetzt:

$$w_l(k) = \begin{cases} 0.2 & \omega > 0 \\ 0.8 & \text{sonst} \end{cases}$$

$$w_r(k) = \begin{cases} 0.8 & \omega > 0 \\ 0.2 & \text{sonst} \end{cases}$$

Dies mag auf den ersten Blick wie eine diskrete Aktion erscheinen. Es lässt sich jedoch folgendermaßen die gewünschte Richtung des Agenten herleiten:

$$\theta'(k) = \arg\left(\frac{\partial \hat{J}}{\partial x} + i \cdot \frac{\partial \hat{J}}{\partial y}\right)$$

i steht in diesem Zusammenhang für die imaginäre Einheit. Die verwendete Action Policy sorgt nun dafür, dass sich der Agent unter Berücksichtigung der Bewegungsphysik des E-Puck (hier ist insbesondere der Kurvenradius gemeint) in diese Richtung dreht. $\theta'(k)$ ist kontinuierlich, somit kann der Agent den E-Puck in jede beliebige Richtung dirigieren.

4.5 Versuchsablauf

Um den im vorhergehenden Abschnitt vorgestellten Agenten untersuchen zu können, wurde eine Reihe von Versuchen durchgeführt. Um möglichst viele Ergebnisse für eine detaillierte Auswertung zu erhalten, wurde das Verhalten des Agenten so realitätsnah wie möglich simuliert. Um die Tauglichkeit unter realen Bedingungen zu zeigen, wurden einige ausgewählte Versuche auf dem realen E-Puck durchgeführt.

Ein Versuch besteht jeweils aus mehreren einzelnen, aufeinanderfolgenden Episoden. Zu Begin einer jeden Episode startet der Agent auf dem in Abbildung 4.2 dargestellten Startpunkt. Das ESN befindet sich vor Begin der ersten Episode in einem untrainierten Zustand.

In jeder Episode hat der Agent nun eine gewisse Zeit, seine Umgebung zu erkunden und sich in ihr zu bewegen. Währenddessen wird das ESN

fortlaufend trainiert. Die einzige Information, die der Agent neben seinem Zustand erhält, ist die Bewertung seiner aktuellen Position. Verlässt der Agent die Umgebung, wird die Episode vorzeitig beendet. Ebenso wird die Episode nach 300 Zeitschritten beendet.

Das vom ESN erworbene Wissen bleibt dem Agenten über das Ende einer Episode hinaus erhalten. In der nächsten Episode startet der Agent also erneut von der Startposition, jedoch mit einem bereits trainierten Netz. Die Erwartung ist nun, dass der Agent das in der vorherigen Episoden gesammelte Wissen benutzen kann, um ein besseres Resultat zu erzielen.

Der Erfolg des Agenten kann auf verschiedene Arten eingeschätzt werden:

- Zu jedem Zeitpunkt ist der Position, auf der sich der Agent befindet, eine Bewertung $U(k)$ zugeordnet. Diese Bewertung erlaubt eine Einschätzung des Agenten zu einem beliebigen Zeitpunkt innerhalb des Versuchs.

- Um eine Episode als Ganzes einschätzen zu können, kann die erhaltene Bewertung über die gesamte Episode hinweg gemittelt werden. Diese Möglichkeit wird im Verlauf der Arbeit häufiger genutzt. Sei $U_1(k)$, $1 \leq k \leq t$ die Bewertung über eine bestimmte Episode hinweg, so ergibt sich die durchschnittliche Bewertung dieser Episode zu:

$$\overline{U_1} = \frac{1}{t} \sum_{i=1}^{k} U_1(i) \tag{4.13}$$

- Schließlich kann die, im vorherigen Punkt angesprochene, Bewertung einer Episode über mehrere Episoden hinweg gemittelt werden. Sei $\overline{U_i}$ die Bewertung der i-ten Episode, so ergibt sich die durchschnittliche Bewertung während der Episoden 2 bis 4 z.B. zu:

$$\overline{U}_{2,3,4} = (\overline{U}_2 + \overline{U}_3 + \overline{U}_4)/3 \tag{4.14}$$

  Die Episoden gehen also mit dem gleichen Gewicht ein, unabhängig von ihrer Länge.

4.6 Beschreibung der Software

Für die Realisierung der Versuche wurde eine Softwareumgebung erstellt. Eine genauere Beschreibung dieser folgt nun.

Die Aufgaben dieser Softwareumgebung lassen sich wie folgt zusammenfassen:

- Kommunikation mit dem E-Puck Roboter, Odometrie und Abstraktion der einzelnen Parameter wie Geschwindigkeit und Position.
- Simulation des E-Puck Roboters, seines Verhaltens und der Interaktion mit einer simulierten Umgebung.
- Erzeugung und Berechnung von Neuronalen Netzen, insbesondere ESNs, sowie das Training dieser.
- Umsetzung des Agenten, des Versuchsablaufs und der Bewertung.
- Aufzeichnung relevanter Daten während des Versuchs.
- Bereitstellung eines abstrakten Interfaces zu sämlichen Versuchsparametern, um automatisiert das Verhalten für viele verschiedene Parameter simulieren und auswerten zu können.

Wahl der Programmiersprache

Zu Begin der Arbeit lag die Präferenz bei C++, aus Gründen der Performance. Es hat sich jedoch schnell gezeigt, dass die Berechnung des ESNs den mit Abstand rechenaufwendigsten Schritt darstellt.

Dies betrifft vor allem die Berechnung des Reservoirs, da hier eine Matrixmultiplikation vorkommt, deren Komplexität in $O(n^3)$ liegt (bei n Neuronen im Reservoir).

Für Matrixmultiplikationen existieren bereits sehr gute Bibliotheken wie z.B. BLAS (Basic Linear Algebra Subprograms). BLAS existiert seit 1979 und wird seitdem kontinuierlich weiterentwickelt, das Spektrum reicht von automatischer Optimierung durch Kompilieren, anschließendem Benchmarken verschiedener Code-Varianten und Auswahl der besten beim Übersetzen der Bibliothek, über sorgfältig handoptimierten Assemblercode bis hin zu

MPI-fähigen Varianten, die es erlauben über ein Netzwerk mehrere Rechner für den Programmierer völlig transparent an einzelnen Operationen mitrechnen zu lassen.

Aus diesem Grund dürfte BLAS jeder naiven Implementierung eine Matrixmultiplikation überlegen sein. LINPACK (Linear Algebra Package) basiert ebenso auf BLAS und stellt komplexere Funktionen zur Verfügung. Der bekannte LINPACK Benchmark, der z.B. zur Ermittlung der Top-500 Liste der schnellsten Computer der Welt dient, basiert direkt auf LINPACK und damit auf BLAS.

Zur Realisierung wurde schließlich die Programmiersprache Python verwendet.

Über die NumPy Bibliothek (Numerical Python) ist ein direkter Zugriff von Python heraus auf die BLAS Routinen möglich. Darüber hinaus liefert NumPy auch viele Funktionen, die von Matlab her bekannt sind, z.B. das für Offline-Learning wichtige Pseudoinverse einer Matrix.

Da aus Sicht der Performance nur geringe Einschnitte gegenüber C++ (ebenfalls mit BLAS) zu erwarten waren, und die Software sehr viel Flexibilität bei der Umsetzung erfordert, fiel die Wahl der Programmiersprache auf Python.

Die grundlegenden Funktionsblöcke der Software und ihre Aufteilung auf die Source-Dateien ist in Abbildung 4.4 dargestellt.

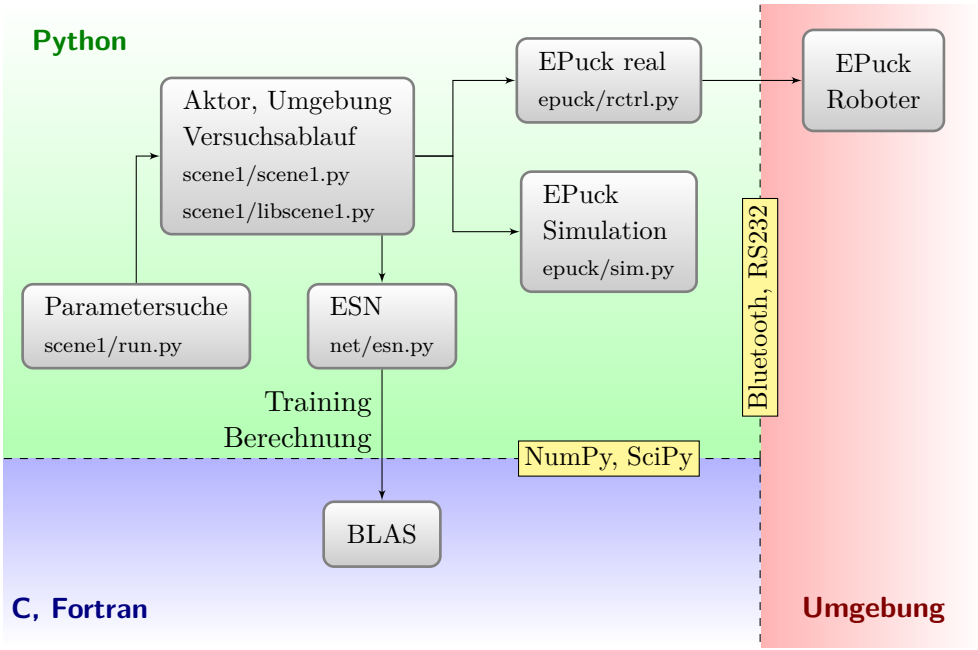


Abbildung 4.4: Organisation der Software mit den zugehörigen Source Dateien. Entsprechend den genannten Anforderungen an die Softwareumgebung wurde diese in Funktionsblöcke unterteilt.

Kapitel 5

Ergebnisse

In diesem Kapitel werden die in den Versuchen erzielten Ergebnisse dargestellt, sowie eine erste Einschätzung vorgenommen.

Im ersten Teil werden die durch Simulationen gewonnenen Daten präsentiert. Diese dienen einer ersten Einschätzung der Auswirkungen der verschiedenen Parameter des ESNs auf den Erfolg des Agenten. Zusätzlich soll ein erster Eindruck über die Anpassungsfähigkeit des Agenten an Veränderungen in der Umgebung gewonnen werden. Schließlich werden erste Methoden eingeführt, die große Menge der gewonnenen Daten darzustellen, auszuwerten und vergleichen. Es folgt eine Bewertung dieser Ergebnisse.

Im zweiten Teil werden die Versuche schließlich auf reale Bedingungen übertragen, um zu überprüfen, in wie weit sich die in den Simulationen gewonnenen Ergebnisse unter realen Bedingungen reproduzieren lassen.

Abschließend folgt eine Einschätzung der gewonnenen Ergebnisse.

5.1 Simulation

5.1.1 ESN Parameter

Um einen Überblick über den Einfluss der Parameter des ESNs auf das Verhalten des Agenten zu erlangen, wurden zu Begin Simulationen mit einer festen Versuchsumgebung durchgeführt.

Für jede Parameterkombination wurden 6 Versuche durchgeführt, mit jeweils anderen Startwerten für den Zufallsgenerator. Auf diese Weise wird der Einfluss statistischer Effekte verringert. Jeder einzelne Versuch besteht aus 20 simulierten Episoden, welche jeweils nach 300 Zeitschritten abgebrochen wurden.

Abbildung 5.1 zeigt das Verhalten des Agenten in einer einzelnen Episode eines Versuchs. Die Parameter wurden folgendermaßen gewählt: $N = 600$, $r = 4$, $c_i = 0.2$, $c_{dr} = 0.4$, $\alpha = 0.8$. Zu Anfang befindet sich der Agent auf der Startposition und bewegt sich dann nach links, wobei die Bewertung seiner aktuellen Position langsam zunimmt. Nach einer Weile fängt der Agent an Kreise zu fahren, dieses Verhalten wird später noch eingehender betrachtet.

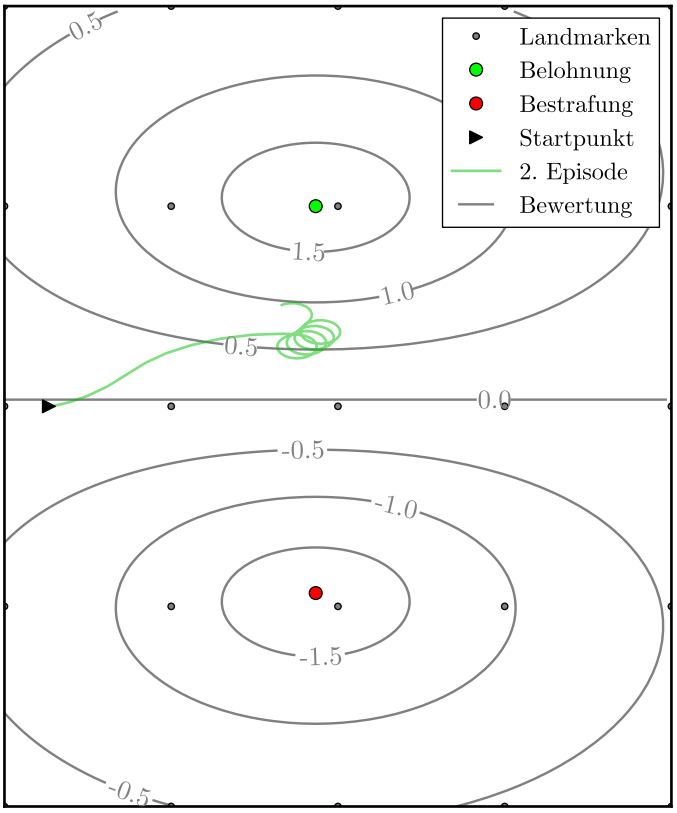


Abbildung 5.1: Bewegung des Agenten über eine Episode. Im ersten Teil findet eine Annährung an die Belohnung statt, danach fängt der Roboter an im Kreis zu fahren.

Die Positionsbewertung, und die Vorhersage dieser durch das ESN, sind in Abbildung 5.2 dargestellt. Über die gesamte Episode hinweg gemittelt ergibt sich eine Bewertung von etwa 0.529.

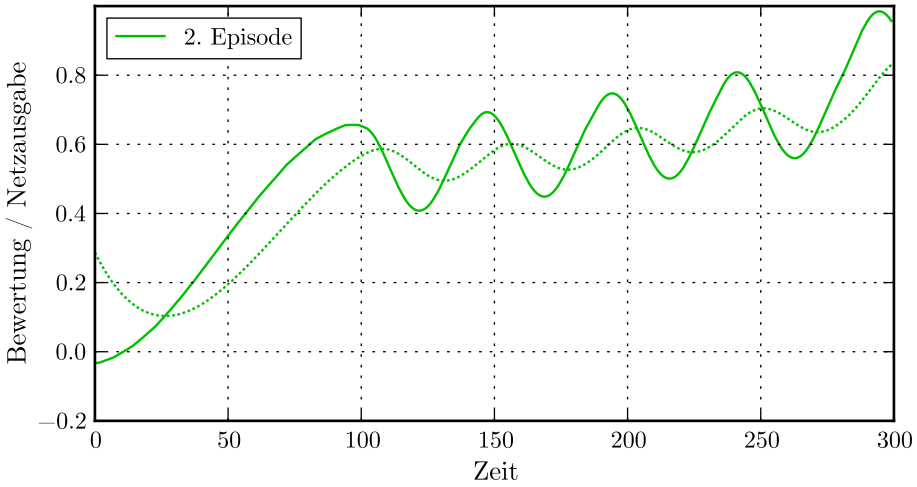


Abbildung 5.2: Die zu Abbildung 5.1 gehörende Positionsbewertung (durchgezogene Linie) und Netzausgabe (unterbrochene Linie) des Agenten. Deutlich zu erkennen ist die Rotation des Agenten ab etwa $t = 100$. Ebenso ist zu erkennen, dass sich der Agent trotzdem weiter auf die Belohnung zubewegt.

Abbildung 5.3 zeigt die ersten 10 Episoden des Versuchs. Besonders in den ersten Episoden verändert sich das Verhalten des Agenten, die Belohnung wird in der 4. Episode erreicht. In den folgenden Episoden entfernt sich der Agent langsam von der Belohnung. Die vom Agenten durchschnittlich erzielte Bewertung über die einzelnen Episoden ist in Abbildung 5.4 dargestellt.

Diese Darstellung erlaubt eine schnelle Bewertung des Agenten, ohne alle Episoden einzeln betrachten zu müssen. Der Versuchsablauf lässt sich in 2 Phasen gliedern:

In einer Lernphase lernt der Agent seine Umgebung kennen, häuft dabei aber neues Wissen an. Das Verhalten verändert sich in den einzelnen Episoden. Dies führt zu einer steilen Lernkurve, die Bewertung der einzelnen Episoden steigt schnell an. In der zweiten Phase ist die Bewertung des Agenten verhältnismäßig konstant. Vorhandenes Wissen wird ausgenutzt, scheinbar wird jedoch kein neues Wissen mehr gesammelt.

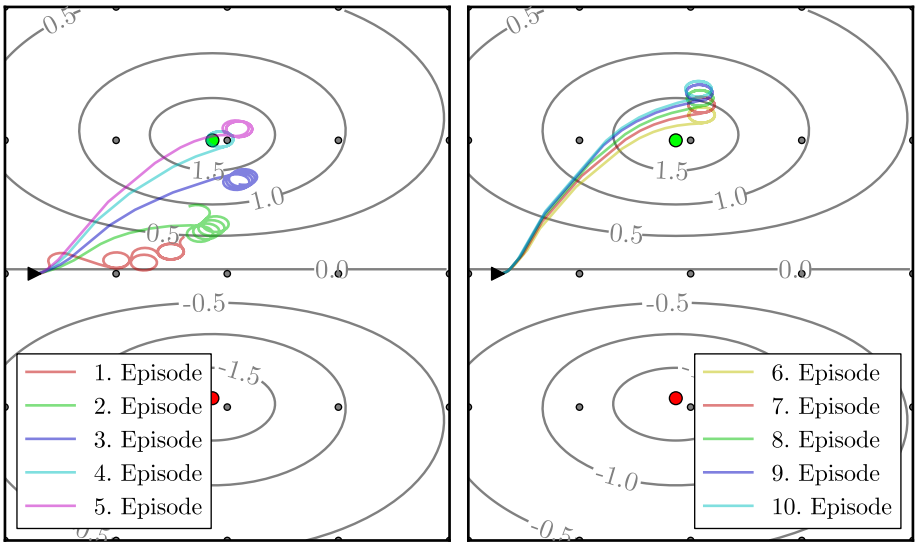


Abbildung 5.3: Darstellung der ersten 10 Episoden. Es ist deutlich zu sehen wie der Agent bis zur 4. Episode immer erfolgreicher wird und sich der Belohnung immer weiter annährt, sich danach jedoch wieder etwas von der Belohnung entfernt.

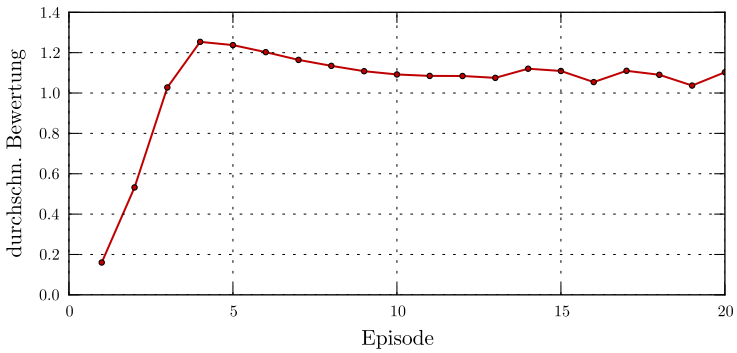


Abbildung 5.4: Die vom Agenten in der jeweiligen Episode erzielte durchschnittliche Bewertung.Deutlich zu sehen ist die Verbesserung bis inkl. zur 4. Episode, danach jedoch eine leichte Verschlechterung. Nach den ersten 10 Episoden hat sich das Verhalten des Agenten weitestgehend stabilisiert.

Abbildung 5.5 zeigt einen fehlgeschlagenen Versuch. Der Agent beginnt zwar damit, die Umgebung zu erkunden, fährt sich aber nach 4 Episoden fest. Die durchschnittliche Bewertung der Episoden ist in Abbildung 5.6 dargestellt und beträgt praktisch 0.

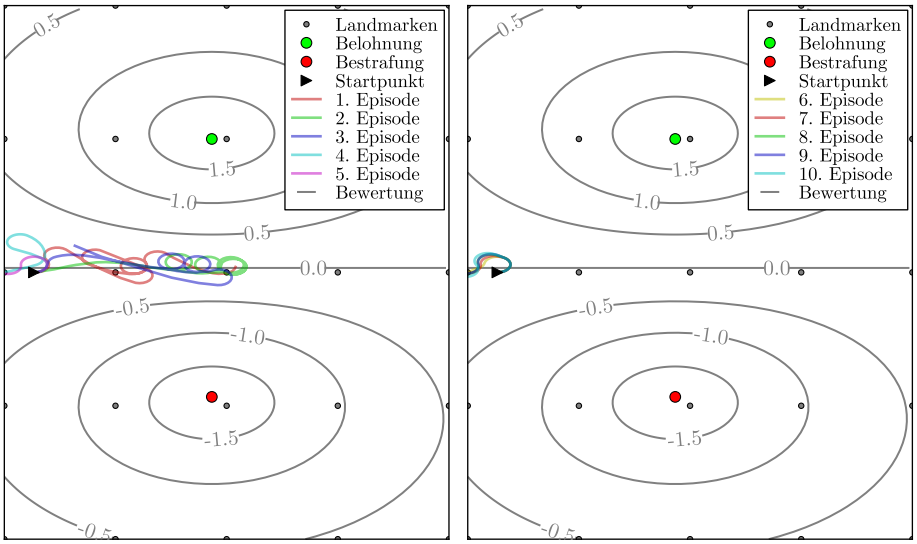


Abbildung 5.5: In diesem Versuch war der Agent nicht in der Lage, die Belohnung zu finden. Die Parameter wurden mit $N = 200$, $r = 1$, $c_i = 0.05$, $c_{dr} = 0.10$ und $\alpha = 0.80$ gewählt.

Im nächsten Schritt wurden nun das Verhalten des Agenten für verschiedene Parameter des ESN simuliert, um einen Eindruck über die Auswirkungen der Parameter auf den Erfolg des Agenten zu erhalten. Die Kombinationen der folgenden Parameter wurden untersucht:

- $N \in \{100, 200, 400, 600\}$
- $\alpha \in \{0.6, 0.8, 0.95\}$
- $(c_i, c_{dr}) \in \{(0.05, 0.10), (0.10, 0.20), (0.20, 0.30), (0.30, 0.40)\}$
- $r_i \in \{1, 4, 16\}$

Mit jeder Parameterkombination wurden 6 Versuche durchgeführt. Die Bewertung des Agenten wurde über diese 6 Versuche gemittelt, um den

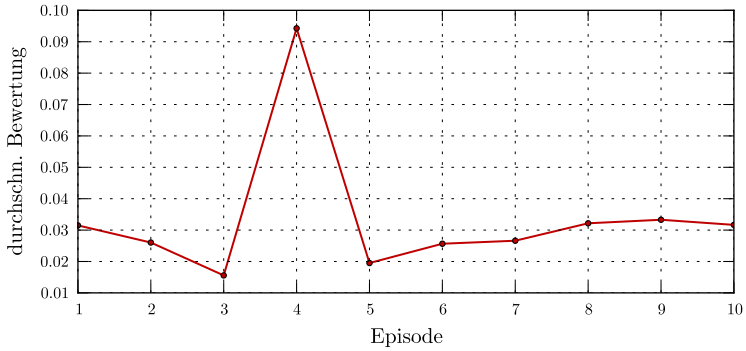


Abbildung 5.6: Die durchschnittliche Bewertung des in Abbildung 5.5 dargestellten Versuchs.

Einfluss statistischer Effekte zu verringern. Die vom Agent in den jeweiligen Episoden erzielte Bewertung ist in den Abbildungen 5.7, 5.8 und 5.9 dargestellt.

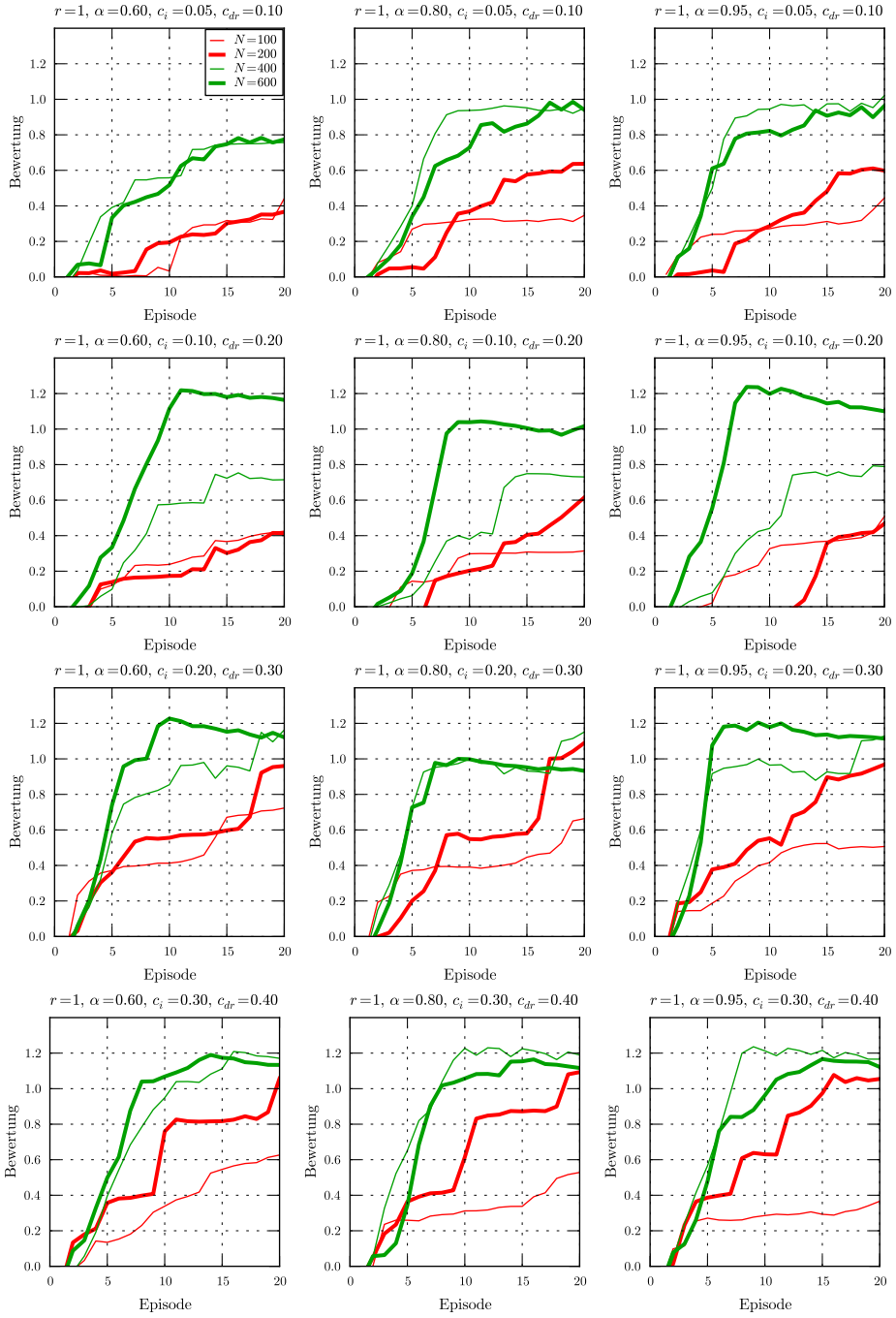


Abbildung 5.7: Ergebnisse für $r = 1$.

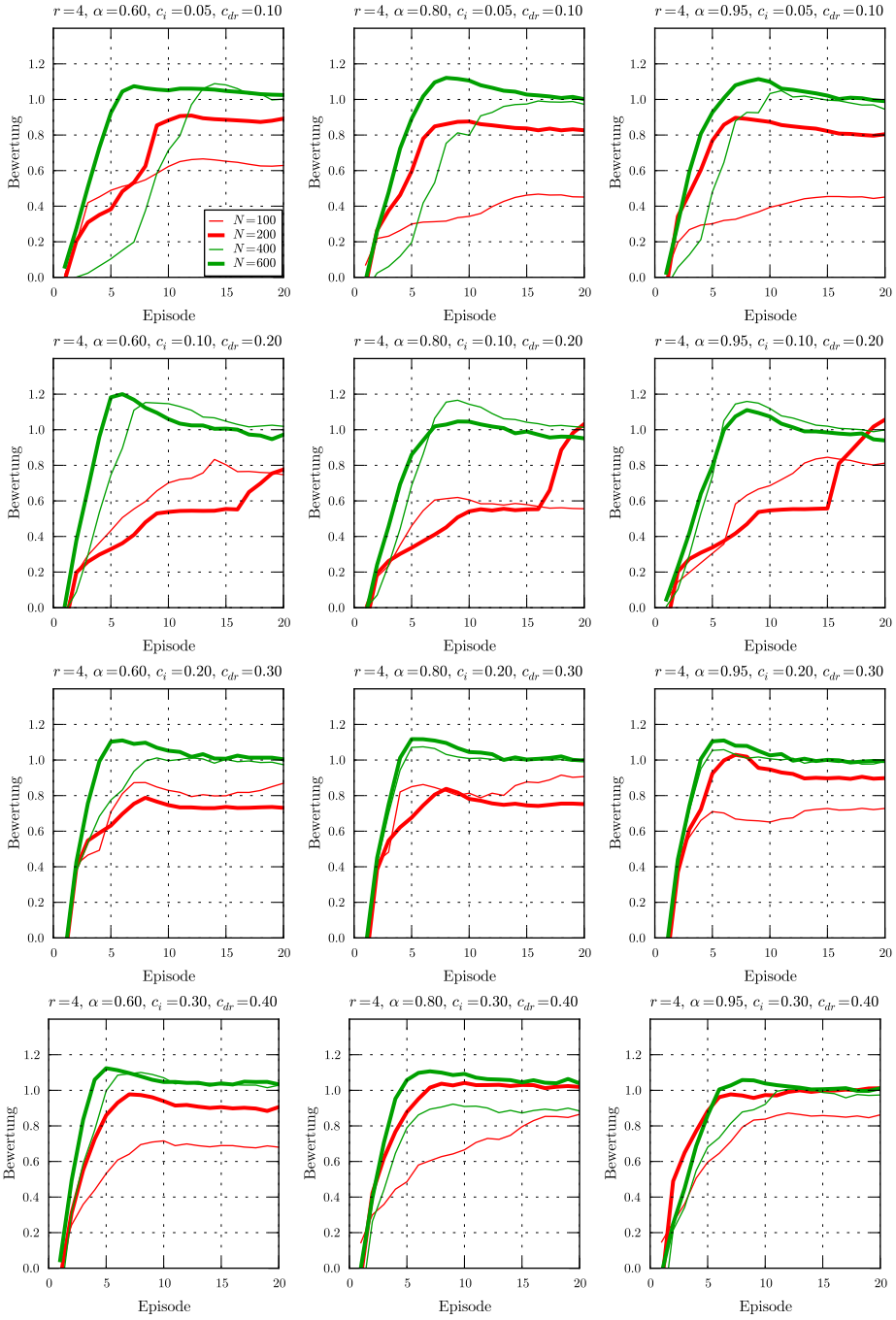


Abbildung 5.8: Ergebnisse für $r = 4$.

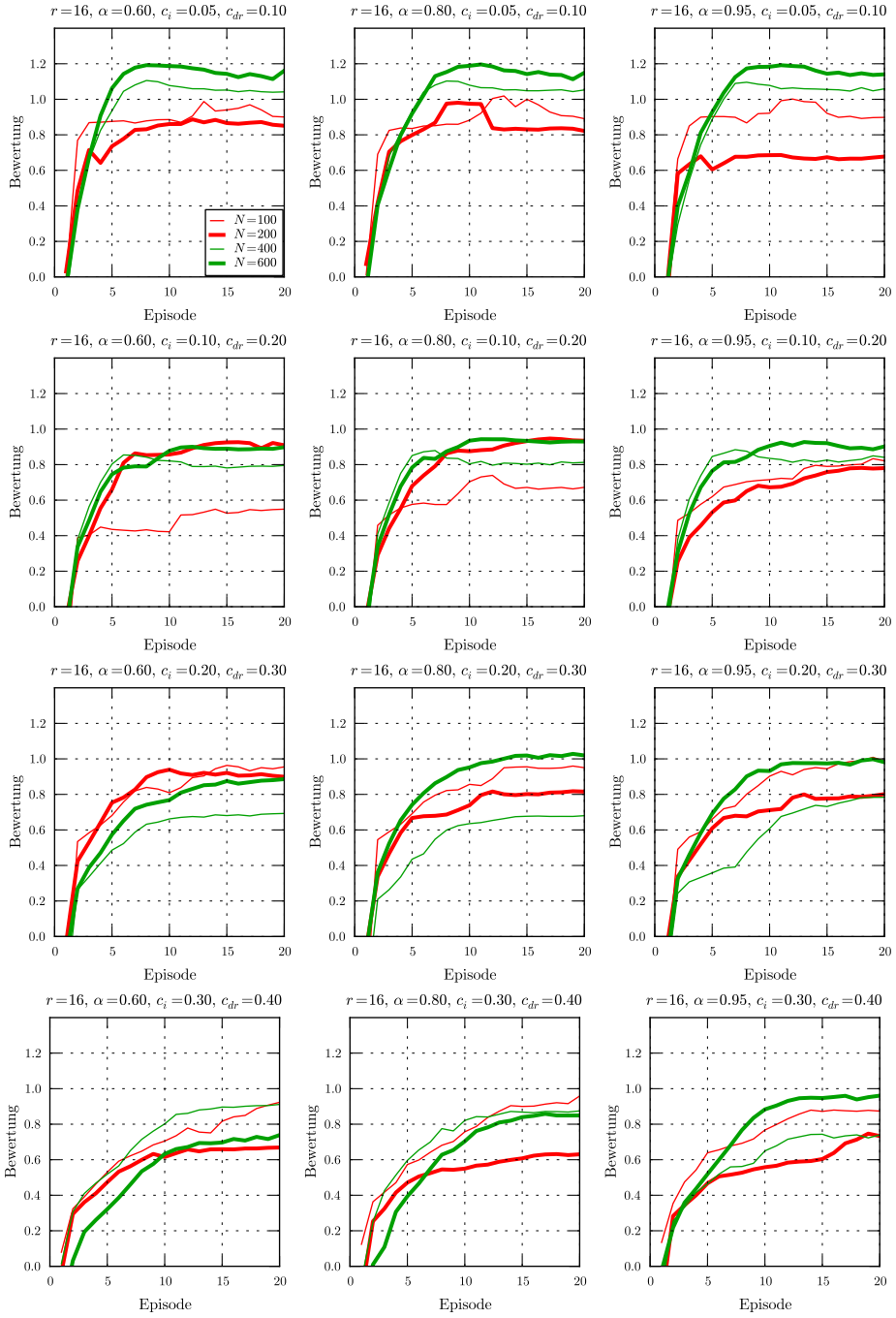


Abbildung 5.9: Ergebnisse für $r = 16$.

Da sich in dieser Form nur schwer ein Überblick über die Ergebnisse gewinnen lässt, wurden diese in Tabellenform zusammengefasst. Die Bewertungen der einzelnen Episoden wurden dafür gemittelt. Die Episoden wurden so zusammengefasst, dass die einzelnen Phasen berücksichtigt werden. Um die Übersichtlichkeit zu verbessern, wurden die Ergebnisse entsprechend ihrem Zahlenwert eingefärbt.

			N				N				N			
r	c_i	c_{dr}	100	200	400	600	100	200	400	600	100	200	400	600
1	0.05	0.10	0.02	0.03	0.21	0.11	0.18	0.02	0.26	0.20	0.26	0.02	0.29	0.29
	0.10	0.20	0.14	0.48	0.10	0.35	0.11	0.00	0.15	0.15	0.21	0.00	0.12	0.38
	0.20	0.30	0.40	0.05	0.38	0.43	0.42	0.28	0.45	0.44	0.27	0.44	0.56	0.54
	0.30	0.40	0.17	0.51	0.32	0.43	0.25	0.45	0.56	0.26	0.19	0.47	0.51	0.40
4	0.05	0.10	0.49	0.52	0.18	0.54	0.36	0.63	0.26	0.57	0.40	0.62	0.38	0.62
	0.10	0.20	0.47	0.48	0.50	0.69	0.36	0.47	0.44	0.56	0.33	0.49	0.45	0.58
	0.20	0.30	0.43	0.52	0.64	0.76	0.50	0.51	0.67	0.75	0.58	0.57	0.68	0.76
	0.30	0.40	0.39	0.56	0.58	0.70	0.37	0.61	0.47	0.71	0.44	0.66	0.48	0.66
16	0.05	0.10	0.72	0.60	0.65	0.73	0.70	0.64	0.63	0.72	0.71	0.57	0.58	0.70
	0.10	0.20	0.49	0.51	0.58	0.53	0.54	0.49	0.62	0.53	0.53	0.51	0.57	0.59
	0.20	0.30	0.44	0.53	0.36	0.37	0.47	0.47	0.27	0.58	0.47	0.45	0.28	0.53
	0.30	0.40	0.45	0.37	0.50	-0.09	0.41	0.36	0.29	0.17	0.46	0.37	0.46	0.44
			$\alpha = 0.60$				$\alpha = 0.80$				$\alpha = 0.95$			

Tabelle 5.1: Zusammenfassung der Simulationsergebnisse der ersten 5 Episoden. Es sind sehr deutliche Unterschiede zwischen den Parametern sichtbar.

In Tabelle 5.1 ist die über die Episoden 1 bis 5 gemittelte Bewertung dargestellt, dies ist die Lernphase des Agenten. Die Parameter haben einen recht deutlichen Einfluss auf das Verhalten des Agenten.

- Kleine Netze mit wenigen Neuronen scheinen eher schlecht zu funktionieren, große Netze dagegen besonders gut.
- Der Einfluss des spektralen Radius α scheint allgemein sehr gering zu sein.
- r und die Verbindungsdichten c_i und c_{dr} scheinen sich gegenseitig zu beeinflussen. Für $r = 1$ und $r = 4$ funktionieren hohe Verbindungsdichten eher besser, bei $r = 16$ scheint das Gegenteil der Fall zu sein.

Tabelle 5.2 zeigt die Bewertung über die Episoden 5 bis 10. In dieser Phase hat sich das Verhalten des Agenten bereits weitestgehend stabilisiert. Die Ergebnisse sind somit von Einschwingeffekten bereinigt und zeigt in

			N				N				N			
r	c_i	c_{dr}	100	200	400	600	100	200	400	600	100	200	400	600
1	0.05	0.10	0.26	0.37	0.78	0.81	0.56	0.32	0.88	0.94	0.40	0.41	0.91	1.04
	0.10	0.20	0.46	0.46	0.66	1.01	0.46	0.33	0.89	0.98	0.48	0.07	0.75	1.15
	0.20	0.30	0.52	0.61	0.95	1.16	0.74	0.64	1.14	1.13	0.59	0.96	1.15	1.14
	0.30	0.40	0.51	0.81	0.99	1.04	0.39	0.85	1.12	1.00	0.58	0.90	1.07	1.08
4	0.05	0.10	0.78	0.75	0.71	1.06	0.64	0.96	1.01	1.07	0.71	1.01	1.04	1.07
	0.10	0.20	0.96	0.88	1.12	1.12	0.70	0.86	1.10	1.01	0.72	0.88	1.10	1.05
	0.20	0.30	0.81	0.93	1.00	1.08	0.81	0.97	1.04	1.09	0.81	0.99	1.03	1.09
	0.30	0.40	0.78	0.95	1.07	1.07	0.71	0.98	1.04	1.10	0.83	0.95	1.09	1.07
16	0.05	0.10	0.88	0.89	1.08	1.17	0.87	0.92	1.08	1.14	0.92	0.89	1.07	1.13
	0.10	0.20	0.80	0.85	0.97	0.88	0.70	0.83	0.98	0.89	0.80	0.82	0.97	0.91
	0.20	0.30	0.85	0.90	0.86	0.70	0.87	0.81	0.83	0.95	0.84	0.81	0.66	0.90
	0.30	0.40	0.72	0.68	0.86	0.61	0.68	0.76	0.78	0.65	0.75	0.75	0.94	0.90
			$\alpha = 0.60$				$\alpha = 0.80$				$\alpha = 0.95$			

Tabelle 5.2: Zusammenfassung der Simulationsergebnisse der Episoden 6 bis 10. In praktisch allen Fällen hat sich der Agent deutlich verbessert. Unterschiede zwischen den einzelnen Parametern sind nach wie vor deutlich sichtbar.

etwa das Maximum, das der Agent erreicht. Zu sehen ist eine deutliche Verbesserung gegenüber Tabelle 5.1.

Die Charakteristika entsprechen weitestgehend den über die ersten 5 Episoden beobachteten. Ein größerer Wert für α scheint die Ergebnisse nun etwas zu verbessern. Zusammenfassend lässt sich sagen, dass ein großer Bereich im Parameterraum zu existieren scheint, in dem die Agenten zufriedenstellend funktionieren.

Um einen besseres Verständnis der Verbindungsdichteparameter c_i und c_{dr} zu gewinnen, wurden weitere Simulationen mit einer feineren Abstufung dieser Parameter durchgeführt. Die Ergebnisse sind in Tabelle 5.3 dargestellt.

Es zeigt sich ein starker Einfluss von c_i. Die Verbindungsdichte im Reservoir selbst, c_{dr}, scheint dagegen nur einen sehr geringen Einfluss auf das Verhalten des Agenten zu haben. Dieses Ergebnis ist etwas ungewöhnlich, der Einfluss von c_{dr} wurde vor Durchführung der Simulationen als deutlich größer eingeschätzt.

Im nächsten Schritt wurde untersucht, welches Wissen der Agent über die Umgebung gesammelt hat. Das Wissen, welches dem Agenten präsentiert wird, wird durch Online-Learning in die Gewichtsmatrix des ESN übertragen. Dabei beeinflusst das Lernen der Bewertung einer spezifischen Position nicht nur die Ausgabe für diese selbst, sondern ebenso die Ausgabe für

r	c_{dr}	c_i 0.05	c_i 0.10	c_i 0.20	c_i 0.05	c_i 0.10	c_i 0.20	c_i 0.05	c_i 0.10	c_i 0.20	c_i 0.05	c_i 0.10	c_i 0.20
1	0.10	0.56	0.45	0.61	0.32	0.24	0.93	0.88	0.85	0.94	0.94	1.09	0.95
	0.20	0.22	0.46	0.70	0.49	0.33	0.65	0.98	0.89	0.93	0.79	0.98	1.06
	0.40	0.50	0.40	0.37	0.57	0.74	0.54	0.76	0.88	1.12	0.89	0.98	1.04
	0.60	0.50	0.43	0.85	0.44	0.46	0.89	0.71	0.77	0.91	1.06	0.99	1.05
4	0.10	0.64	0.88	0.85	0.96	0.97	0.92	1.01	1.13	1.06	1.07	1.08	1.08
	0.20	0.54	0.70	0.84	0.84	0.86	0.70	1.06	1.10	1.10	1.05	1.01	1.10
	0.40	0.75	0.64	0.62	0.85	0.93	1.01	0.86	1.09	1.12	1.14	1.13	0.98
	0.60	0.54	0.90	0.64	0.70	1.03	0.96	0.93	1.02	1.16	1.13	1.14	1.10
16	0.10	0.87	0.97	0.80	0.92	0.99	0.90	1.08	0.90	0.77	1.14	0.83	0.84
	0.20	0.67	0.70	0.86	0.95	0.83	0.76	1.10	0.98	0.81	1.04	0.89	0.79
	0.40	0.87	0.84	0.85	0.93	0.90	0.73	1.06	0.79	0.90	1.11	0.90	0.68
	0.60	0.62	0.86	0.68	1.04	0.95	0.97	1.01	0.90	0.95	1.17	0.96	0.89
		$N = 100$			$N = 200$			$N = 400$			$N = 600$		

Tabelle 5.3: Genauere Abstufung der Verbindungsdichte. α wurde mit 0.8 gewählt, dargestellt ist der Durchschnitt der Episoden 6 bis 10. Es zeigt sich, dass vor allem c_i einen großen Einfluss auf die Ergebnisse hat.

alle anderen Positionen. Dadurch erzeugt das Netz auch für Positionen Ausgaben, die der Agent noch nicht besucht hat. Ebenso verändert sich die Ausgabe für bereits besuchte Positionen mit jedem Lernschritt. Dies soll nun untersucht werden.

Zu diesem Zweck wurde zuerst die Dynamik der Netzausgabe bestimmt. Hierfür wurde die Umgebung, in 2 verschiedenen Richtungen, auf vorgegebenen Wegen durchfahren und die Bewertung des ESNs für identische Positionen, aber aus unterschiedlichen Richtungen kommend, verglichen.

Es hat sich dabei gezeigt, dass die Netzausgabe nur sehr wenig Dynamik aufweist, und daher ignoriert werden kann. Eine genauere Analyse der Dynamik findet in Kapitel 6.1.2 statt.

Um nun das Wissen des Agenten zu einem bestimmten Zeitpunkt zu ermitteln, wurde eine Kopie des Agenten zu einem bestimmten Zeitpunkt während des Versuchs erstellt. Nun wurden sämtliche Positionen in der Umgebung durchfahren, und die Bewertung des ESNs aufgenommen und visualisiert. Der Agent hatte keine Möglichkeit Aktionen auszuführen und wurde auch nicht trainiert.

Als Ausgangspunkt diente der am Anfang des Kapitels in Abbildung 5.1 vorgestellte Versuch. Abbildung 5.10 zeigt das Wissen des Agenten zu

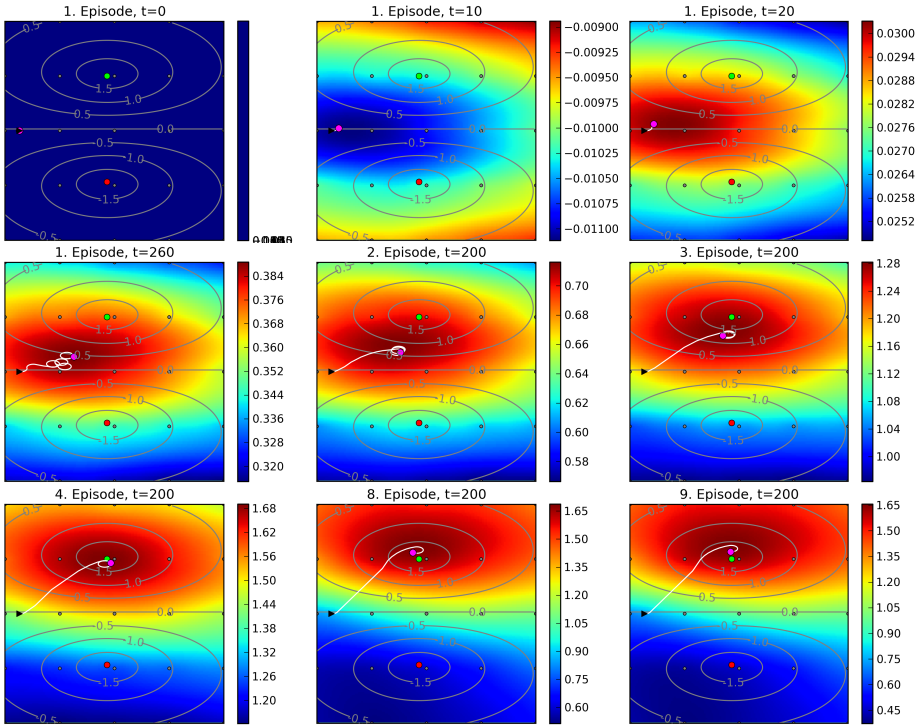


Abbildung 5.10: Das vom Agenten über seine Umgebung gesammelte Wissen für einige charakteristische Zeitpunkte des in Abbildung 5.3 dargestellten Versuchs.

ausgewählten Zeitpunkten, die Unterteilung der Umgebung in gut und schlecht ist sehr deutlich zu sehen. Auffällig ist, dass der Wertebereich der Netzausgabe sehr starken Schwankungen unterworfen ist. Diese Beobachtung wird in Kapitel 6.2.2 genauer untersucht. Darüber hinaus scheint das ESN die Bewertung der Positionen jedoch recht gut nachzubilden. Im Detail sind jedoch einige kleine Abweichungen sichbar, so befindet sich am Ende des Versuchs der vom ESN vorhergesagte Bereich der maximalen Belohnung leicht oberhalb der realen Belohnung. Dies erklärt auch das Verhalten des Agenten, sich in den letzten Episoden leicht oberhalb dieser aufzuhalten.

5.1.2 Adaptivität

Im nächsten Schritt wurde die Anpassungsfähigkeit des Agenten an eine für ihn unvorhersehbare Veränderung in der Umgebung untersucht. Dafür wurden Belohnung und Bestrafung nach der 5. Episode vertauscht. Da der Agent über die einzelnen Versuche hinweg kein Gedächtnis besitzt, erscheint ihm diese Veränderung zufällig.

Für einen Beobachter mag diese Veränderung sehr einfach erscheinen, da effektiv die Bewertungen für alle Positionen mit -1 multipliziert werden. Dies ließe sich theoretisch einfach durch ein Invertieren aller Gewichte der Ausgabeneuronen des ESNs erreichen.

Für den Agenten selbst stellt dies jedoch wegen der eingeschränkten Sichtbarkeit der Umgebung eine völlig neue Situation dar. Das Gesamtbild, das sich lediglich das Vorzeichen der Bewertung geändert hat, würde für den Agenten erst sichtbar werden, nachdem er die gesamte Umgebung besucht hat.

Abbildung 5.11 zeigt einen erfolgreichen Versuch. Nachdem die Umgebung verändert wurde, passt sich der Agent in den Episoden 6 bis 9 an diese Veränderung an. In der 10. Episode schließlich wird die Belohnung gefunden. In Abbildung 5.12 ist zum Vergleich ein Versuch dargestellt, in dem sich der Agent, nachdem die Umgebung verändert wurde, festfährt. Dies stellt ein häufiger auftretendes Problem dar.

Die Betrachtung der ESN Parameter ist in den Tabellen 5.4 bis 5.6 dargestellt. Es wurden wieder 6 Versuche für jede Parameterkombination simuliert, und die Ergebnisse gemittelt. Die in Tabelle 5.4 dargestellten

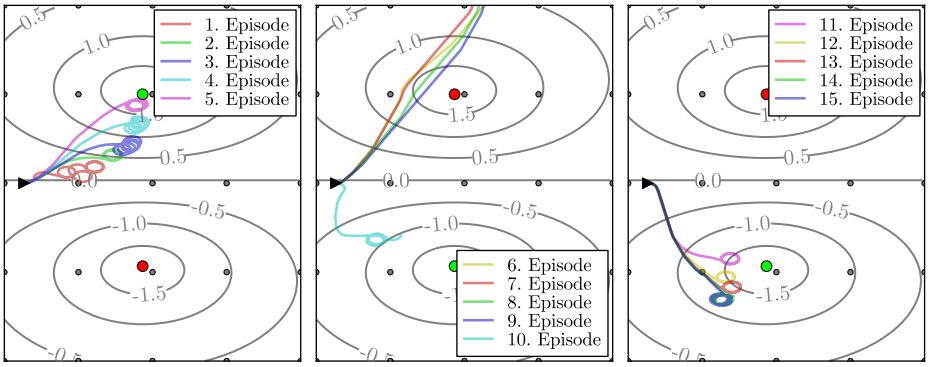


Abbildung 5.11: In diesem Versuch passt sich der Agent, nach 4 erfolglosen Episoden, erfolgreich an die veränderte Umgebung an. Parameter: $r = 4$, $c_i = 0.30$, $c_{dr} = 0.40$, $\alpha = 0.80$, $N = 600$

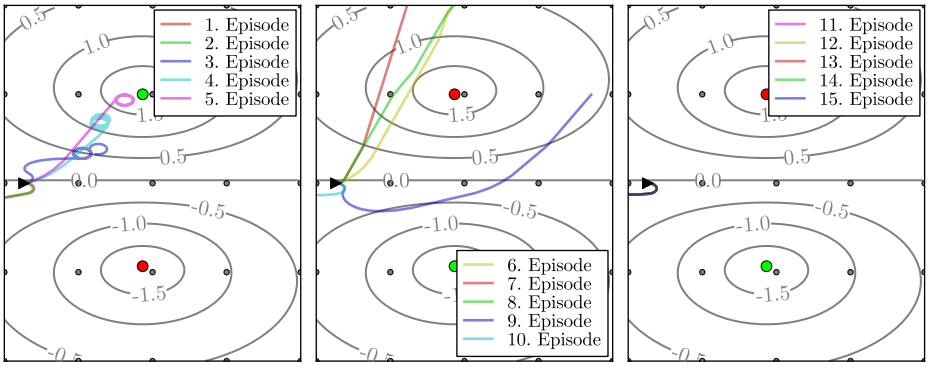


Abbildung 5.12: Der Agent fährt sich nach Veränderung der Umgebung fest. Über die letzten 6 Episoden ändert sich das Verhalten nicht. Parameter: $r = 4$, $c_i = 0.10$, $c_{dr} = 0.20$, $\alpha = 0.95$, $N = 400$

Ergebnisse entsprechen denen in Tabelle 5.1, da die Umgebung erst am Ende der 5. Episode verändert wurde.

			N				N				N			
r	c_i	c_{dr}	100	200	400	600	100	200	400	600	100	200	400	600
1	0.05	0.10	0.02	0.03	0.21	0.11	0.18	0.02	0.26	0.20	0.26	0.02	0.29	0.29
	0.10	0.20	0.14	0.48	0.10	0.35	0.11	0.00	0.15	0.15	0.21	0.00	0.12	0.38
	0.20	0.30	0.40	0.05	0.38	0.43	0.42	0.28	0.45	0.44	0.27	0.44	0.56	0.54
	0.30	0.40	0.17	0.51	0.32	0.43	0.25	0.45	0.56	0.26	0.19	0.47	0.51	0.40
4	0.05	0.10	0.49	0.52	0.18	0.54	0.36	0.63	0.26	0.57	0.40	0.62	0.38	0.62
	0.10	0.20	0.47	0.48	0.50	0.69	0.36	0.47	0.44	0.56	0.33	0.49	0.45	0.58
	0.20	0.30	0.43	0.52	0.64	0.76	0.50	0.51	0.67	0.75	0.58	0.57	0.68	0.76
	0.30	0.40	0.39	0.56	0.58	0.70	0.37	0.61	0.47	0.71	0.44	0.66	0.48	0.66
16	0.05	0.10	0.72	0.60	0.65	0.73	0.70	0.64	0.63	0.72	0.71	0.57	0.58	0.70
	0.10	0.20	0.49	0.51	0.58	0.53	0.54	0.49	0.62	0.53	0.53	0.51	0.57	0.59
	0.20	0.30	0.44	0.53	0.36	0.37	0.47	0.47	0.27	0.58	0.47	0.45	0.28	0.53
	0.30	0.40	0.45	0.37	0.50	-0.09	0.41	0.36	0.29	0.17	0.46	0.37	0.46	0.44
			$\alpha = 0.60$				$\alpha = 0.80$				$\alpha = 0.95$			

Tabelle 5.4: Durchschnittliche Bewertung der Episoden 1 bis 5. Dies entspricht den in Tabelle 5.1 dargestellten Ergebnissen.

			N				N				N			
r	c_i	c_{dr}	100	200	400	600	100	200	400	600	100	200	400	600
1	0.05	0.10	-0.18	-0.18	-0.30	0.00	-0.32	-0.10	-0.42	-0.28	-0.30	-0.21	-0.25	-0.25
	0.10	0.20	-0.14	-0.27	0.08	-0.11	-0.24	-0.03	-0.15	0.16	0.03	0.07	0.05	-0.35
	0.20	0.30	-0.45	0.04	-0.28	-0.36	-0.40	0.01	-0.28	-0.37	-0.29	-0.31	-0.43	-0.63
	0.30	0.40	-0.15	-0.24	-0.05	-0.47	-0.19	-0.38	-0.28	-0.31	-0.13	-0.34	-0.29	-0.38
4	0.05	0.10	-0.17	-0.17	0.21	-0.70	0.02	-0.36	-0.03	-0.65	-0.01	-0.03	-0.03	-0.46
	0.10	0.20	-0.01	0.05	-0.45	-0.80	0.28	0.05	-0.39	-0.61	0.29	-0.05	-0.46	-0.68
	0.20	0.30	-0.06	-0.37	-0.57	-0.83	-0.08	-0.15	-0.66	-0.84	0.01	-0.17	-0.72	-0.84
	0.30	0.40	0.14	-0.17	-0.53	-0.79	-0.04	-0.32	-0.26	-0.77	0.19	-0.47	-0.45	-0.79
16	0.05	0.10	-0.40	-0.44	-0.79	-0.83	-0.32	-0.54	-0.75	-0.61	-0.31	-0.54	-0.74	-0.61
	0.10	0.20	-0.17	-0.14	-0.73	-0.71	-0.17	-0.14	-0.70	-0.69	-0.20	-0.25	-0.72	-0.69
	0.20	0.30	-0.27	-0.26	-0.50	-0.38	-0.30	-0.15	-0.44	-0.43	-0.27	-0.26	-0.55	-0.66
	0.30	0.40	0.16	-0.02	-0.44	0.00	0.21	0.11	-0.38	-0.11	0.15	0.10	-0.34	-0.42
			$\alpha = 0.60$				$\alpha = 0.80$				$\alpha = 0.95$			

Tabelle 5.5: Ergebnisse der Episoden 6 bis 9, direkt nach Vertauschen von Belohnung und Bestrafung. Deutlich zu sehen ist der Einbruch bei allen Parameterkombinationen.

Zwischen der 6. und 9. Episode, dargestellt in Tabelle 5.5, bricht die Bewertung des Agenten massiv ein. In der 6. Episode sucht dieser in den meisten Fällen noch die alte Position der Belohnung auf, da sich an dieser Stelle nun aber die Bestrafung befindet, wird eine entsprechend schlechte Bewertung erzeugt. Parameterkombinationen, die in den ersten 5 Episoden zu besonders guten Ergebnissen geführt haben, führen nun zu besonders schlechten Ergebnissen. Dies lässt sich dadurch erklären, dass

			N				N				N			
r	c_i	c_{dr}	100	200	400	600	100	200	400	600	100	200	400	600
1	0.05	0.10	0.02	0.06	0.38	0.79	0.11	0.12	0.69	0.88	0.18	0.03	0.56	0.45
	0.10	0.20	0.06	0.06	0.45	0.55	0.10	0.09	0.13	0.90	0.30	0.31	0.53	0.66
	0.20	0.30	0.23	0.45	0.78	0.56	0.59	0.66	0.95	0.83	0.36	0.49	0.87	0.62
	0.30	0.40	0.63	0.29	0.63	1.00	0.28	0.50	0.75	0.68	0.44	0.40	0.50	0.83
4	0.05	0.10	0.54	0.52	0.90	-0.03	0.74	0.68	0.59	-0.06	0.64	0.86	0.86	0.18
	0.10	0.20	0.73	0.81	0.87	0.23	0.87	0.78	1.01	0.54	0.80	0.74	0.70	0.40
	0.20	0.30	0.87	0.96	0.75	0.52	0.85	1.00	0.86	0.63	0.88	0.84	0.79	0.54
	0.30	0.40	0.89	0.93	0.87	0.66	0.85	0.95	0.93	0.87	0.74	0.98	0.88	0.86
16	0.05	0.10	0.28	0.39	0.18	0.44	0.37	0.36	0.30	0.36	0.42	0.53	-0.17	0.45
	0.10	0.20	0.46	0.80	0.41	0.55	0.84	0.82	0.55	0.17	0.74	0.72	0.69	0.50
	0.20	0.30	0.55	0.75	0.06	0.77	0.24	0.67	0.74	0.87	0.33	0.49	0.04	0.68
	0.30	0.40	0.64	0.80	0.78	0.61	0.82	1.00	0.65	0.74	0.57	0.80	0.87	0.79
			$\alpha = 0.60$				$\alpha = 0.80$				$\alpha = 0.95$			

Tabelle 5.6: Episoden 10 bis 14. In vielen Fällen hat sich der Agent an die neuen Bedingungen angepasst und liefert wieder gute Ergebnisse.

sich diese Agenten in den ersten 5 Episoden der Belohnung besonders weit angenähert haben, und sich nun entsprechend der Bestrafung besonders weit annähren.

Tabelle 5.6 zeigt die Resultate in den Episoden 10 bis 14. Die meisten Agenten waren in der Lage, die neue Position der Belohnung zu finden und liefert dementsprechend wieder gute Ergebnisse. Leider ist zu beobachten, dass sich einige der Agenten in bestimmten Situationen festgefahren haben, was sich durch entsprechend schlechte Bewertungen äußert.

5.2 Zwischenfazit

Zusammenfassend lässt sich sagen, dass der untersuchte ACD Ansatz sowohl im Fall einer festen Umgebung, als auch im Fall einer sich verändernden Umgebung funktioniert hat. Besonders im ersten Fall ist der Einfluss der verschiedenen Parameter des ESNs zu sehen. Dies wird in Kapitel 6 nocheinmal genauer betrachtet.

Bei einer sich verändernden Umgebung sind die Resultate stark von Instabilitäten im Verhalten des Agenten geprägt. Bleibt dieses aus, schafft es der Agent jedoch sich an die Veränderungen in der Umgebung anzupassen und findet erfolgreich die neue Position der Belohnung.

5.3 Durchführung auf dem realen Roboter

Da sich der Agent in den Simulationen recht erfolgreich verhalten hat, wurde nun das Verhalten unter realen Bedingungen untersucht. Es wird an dieser Stelle nur ein Versuch betrachtet, da sich die erfolgreichen Versuche sehr ähneln. Zudem wäre es zu aufwendig, auch nur annährend die selbe Menge an Versuchen real durchzuführen, wie simuliert wurden.

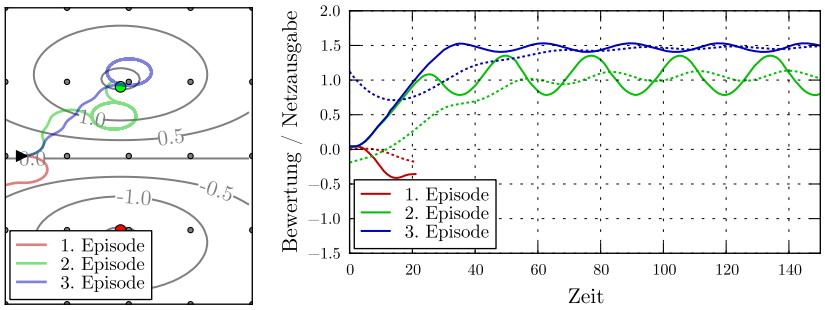
Ein Überblick über den gesamten Versuch ist in Abbildung 5.13 und Abbildung 5.14 dargestellt. Nach der 5. Episode wurden Belohnung und Bestrafung, wie in den Simulationen, vertauscht. Da der E-Puck mit den, in den Simulationen gewählten, Raddrehzahlen einige Probleme zeigte, mussten Kurvenradius und Geschwindigkeit etwas größer gewählt werden.

Der Versuch lässt sich wieder in 3 Phasen unterteilen. In der ersten Phase lernt der Agent, wie im ersten Versuchsteil, seine Umgebung kennen. Die wichtigsten Abschnitte der ersten Phase sind in Abbildung 5.15 dargestellt, fast der gesamte Lernprozess findet in den ersten beiden Episoden statt.

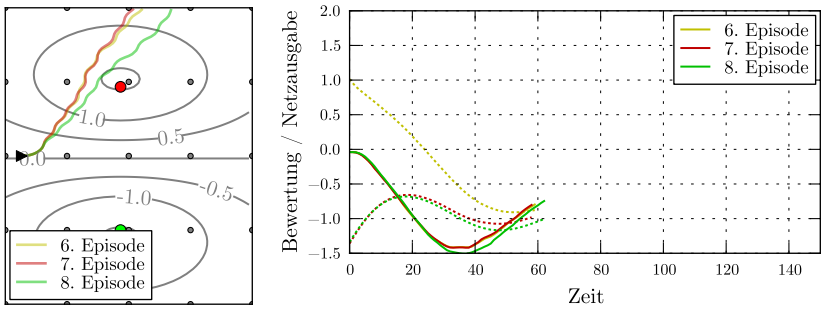
In der ersten Episode lenkt der Agent nach rechts und nähert sich der Bestrafung an, dadurch erfährt er eine negative Bewertung. Dadurch bewertet der Agent den Bereich rund um seine Startposition negativ, dies ist für $t = 10$ und $t = 21$ zu sehen. Da die negative Bewertung vom Agenten aus gesehen rechts der Startposition erfolgte, liegt der Mittelpunkt des vom Agenten schlecht erachteten Bereichs ebenso rechts der Startposition.

Dadurch lenkt der Agent zu Begin der 2. Episode nach links. In der Folge nähert er sich der Belohnung und erfährt eine gute Bewertung. Zum Ende der 2. Episode stimmt die Vorhersage der Kritik bereits weitestgehend mit der realen Bewertung U überein.

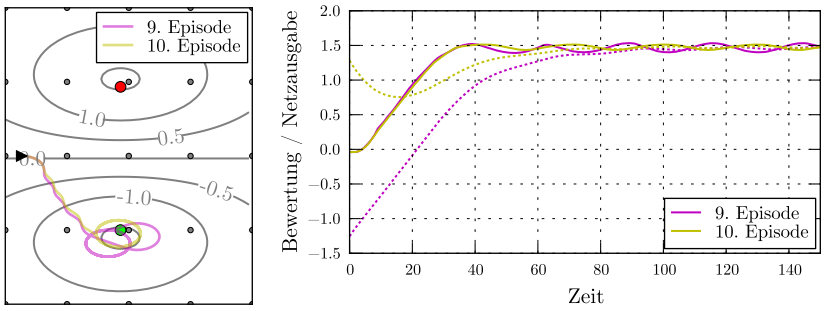
In den Episoden 3 bis 5 verändert sich das Wissen des Agenten kaum mehr, entsprechend gleichen sich auch die Aktionen weitestgehend. Das führt wiederrum dazu, dass kein neues Wissen gesammelt wird.



(a) Erste Phase: Der Agent lernt seine Umgebung kennen. Dies entspricht dem vorherigen Versuch mit statischer Umgebung. Die 4. und 5. Episode entsprechen der 3. Episode.



(b) Zweite Phase: Die Umgebung wurde verändert. Das Wissen des Agenten über seine Umgebung passt nicht mehr zu dieser. Nach erreichen der ehemaligen Belohnung merkt der Agent dies offensichtlich und fährt geradeaus weiter.



(c) Dritte Phase: Der Agent hat sich an die neue Umgebung angepasst und liefert nun wieder gute Ergebnisse.

Abbildung 5.13: Verhalten des Agenten bei sich verändernder Umgebung. Der komplette Versuch lässt sich in 3 Phasen unterteilen, diese sind von (a) bis (c) dargestellt. Links ist jeweils das Verhalten des Agenten dargestellt, rechts die Bewertung (durchgehende Linie) und die vom Netz vorhergesagte Bewertung (unterbrochene Linie)

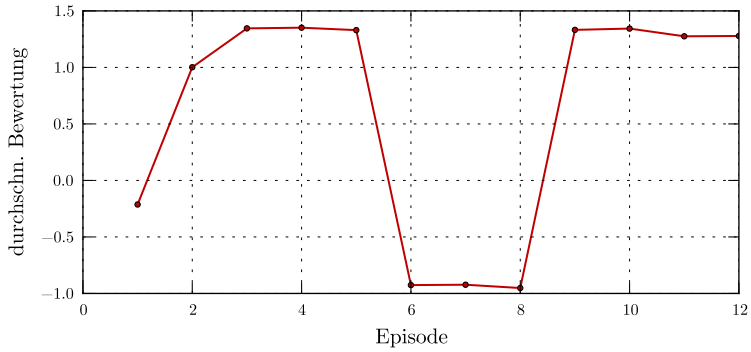


Abbildung 5.14: Die vom Agenten in den Episoden erzielte durchschnittliche Bewertung. Die Phase der Anpassung ist deutlich sichtbar.

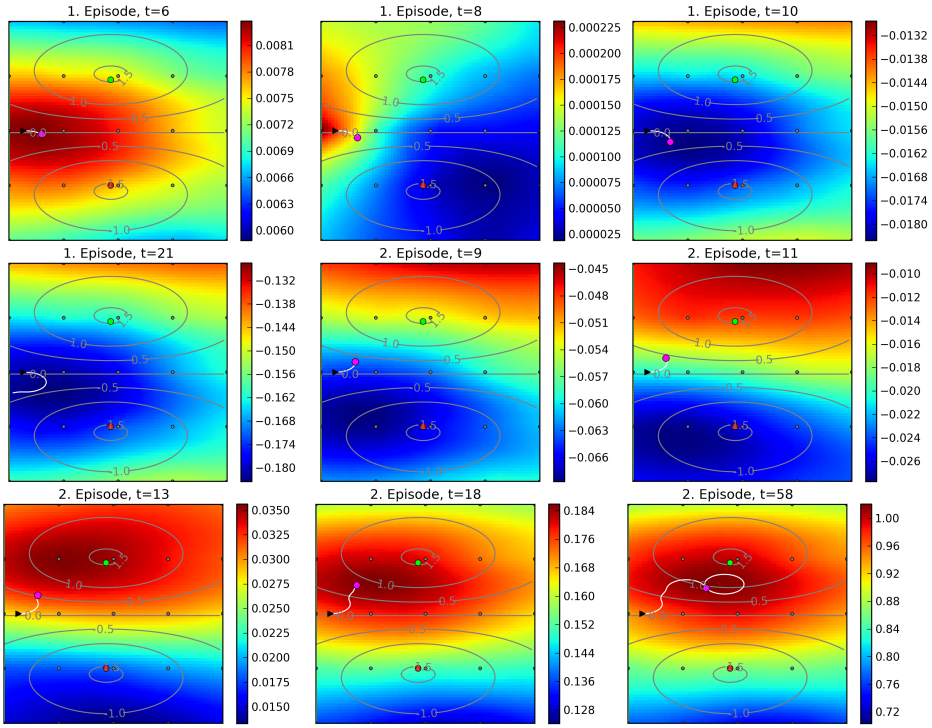


Abbildung 5.15: Erster Teil der ersten Phase des Versuchs. Nach einigen Anfangsfluktuationen stabilisiert sich das Wissen

Abbildung 5.17 zeigt die zweite Phase des Versuchs. Belohnung und Bestrafung wurden nun vertauscht. Vor allem in der ersten Episode dieser Phase, der 6. Episode im Versuch, verändert sich das Wissen des Agenten sehr stark. Der bisher als gut erachtete Bereich schiebt sich nach oben aus der Karte heraus, und der als schlecht erachtete Bereich wieder zurück in Richtung Startposition. So gesehen kehrt sich die Entwicklung der ersten Phase um.

In den folgenden 2 Episoden fällt diese Veränderung geringer aus, setzt sich aber trotzdem fort. Wichtig für das Verhalten des Agenten ist der Zeitpunkt in Episode 8, zu dem sich der als schlecht erachtete Bereich bis leicht oberhalb der Startposition verschoben hat ($t = 47$). Aufgrund dieser Veränderung lenkt der Agent nun in der folgenden 9. Episode, die den Anfang der 3. Phase markiert (Abbildung 5.18), nicht mehr nach links, sondern nach rechts in Richtung Belohnung.

In der 3. Phase wiederholt sich das Verhalten der 1. Phase, der Agent fährt erstmalig zur neuen Position der Belohnung, das Wissen wird dementsprechend angepasst.

Das Verhalten des Agenten zu Begin der Episoden wird hauptsächlich davon beeinflusst, in welcher Richtung der als schlecht erachtete Bereich liegt. Diesen versucht der Agent zu vermeiden und lenkt entsprechend in die entgegengesetzte Richtung. Von besonderer Bedeutung sind daher die Zeitpunkte, zu denen der als schlecht vorhergesagte Bereich die Seite des Agenten wechselt. Dies führt unmittelbar zu einem anderen Verhalten des Agenten, und beeinflusst in der Folge den gesamten Ablauf der Episode.

In Abbildung 5.19 werden die während des Versuchs gesammelten Daten mit Bildern der dargestellten Zeitpunkte gegenübergestellt. Die Odometrie hat, auch Dank der durchgeführten Kalibrierung des E-Pucks, zufriedenstellend funktioniert.

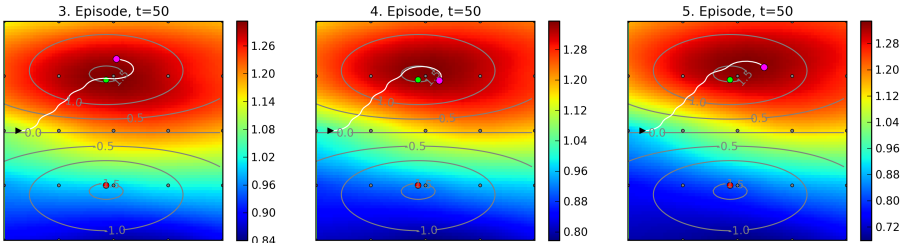


Abbildung 5.16: Zweiter Teil der ersten Phase. Das Wissen des Agenten verändert sich kaum.

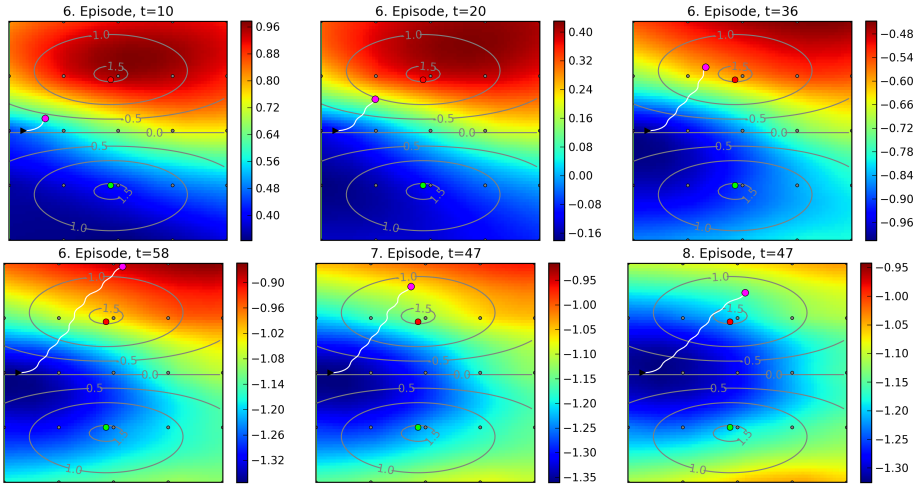


Abbildung 5.17: Zweite Phase des Versuchs. Die Umgebung wurde nun verändert. Der Agent erkennt dies und das Wissen wird entsprechend angepasst.

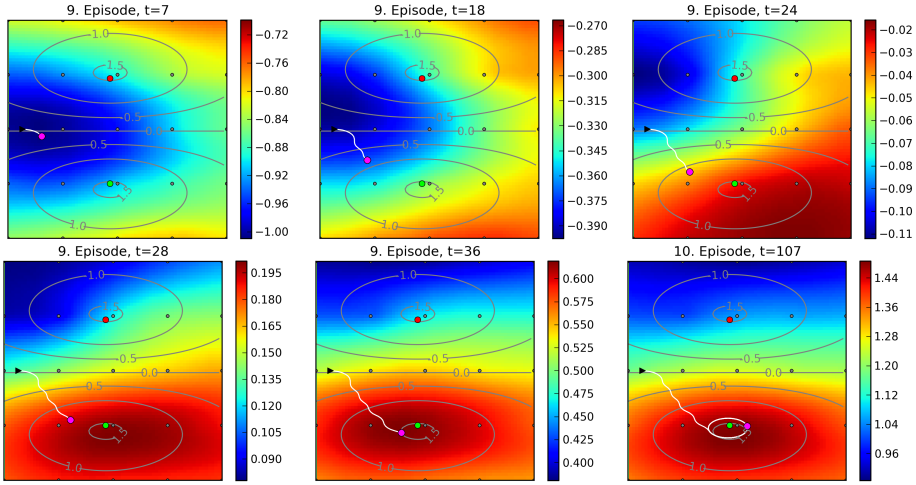


Abbildung 5.18: Dritte Phase. Da der Bereich aus Sicht des Agenten links der Startposition nun schlecht bewertet wird, lenkt der Agent nach rechts und findet die neue Position der Belohnung.

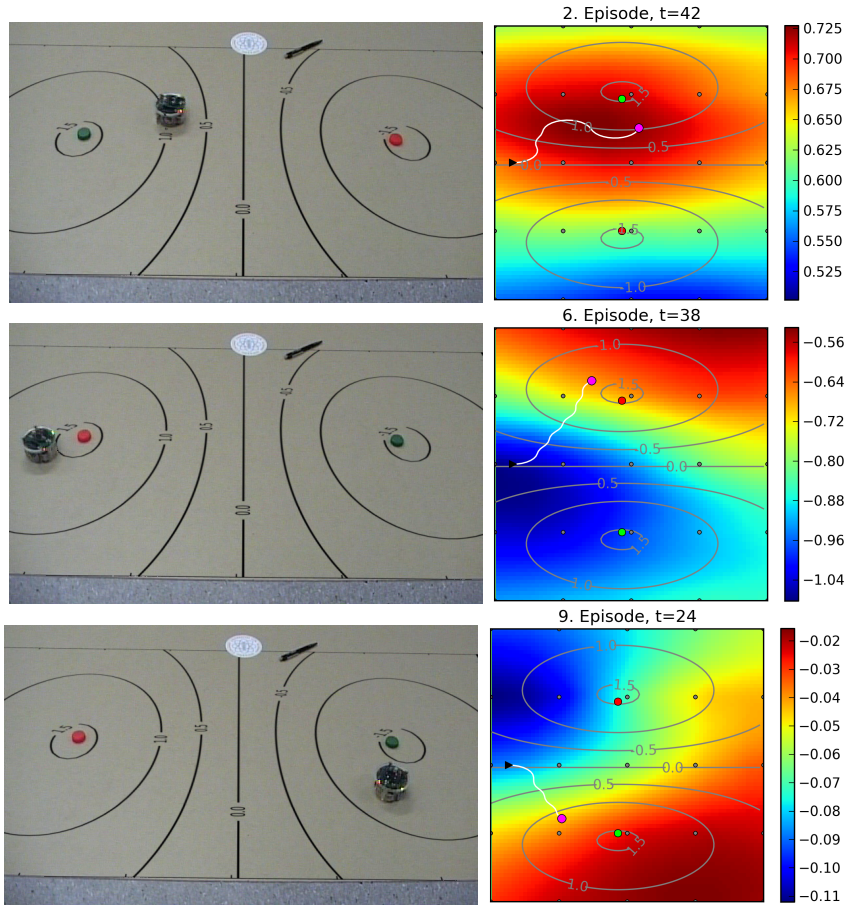


Abbildung 5.19: Gegenüberstellung der gesammelten Daten mit Bildern des Versuchs.

5.4 Bewertung

Die durchgeführten Versuche zeigen, dass das getestete ACD Konzept in der Lage ist, sich unter realen Bedingungen in einer Umgebung zurechtzufinden, und sich an Veränderungen dieser anzupassen. Die Umgebung wird beobachtet, mit dem eigenen Wissen verglichen, und dieses Wissen wird, wenn nötig, angepasst. Darauf basierend werden schließlich Aktionen ausgewählt.

Leider muss jedoch auch gesagt werden, dass instabiles Verhalten des Agenten ein Problem darstellt. Dies tritt besonders bei den Versuchen mit sich verändernder Umgebung zutage. Wann immer der Agent sich an eine für ihn neue Situation anpasst, besteht das Rikiso, dass er sich in einem bestimmten Verhalten festfährt.

Kapitel 6

Analyse der Ergebnisse

Im letzten Kapitel wurden die Resultate der durchgeführten Versuche dargestellt und eingehender betrachtet. Dabei wurden lediglich die ESN Parameter und ihr Einfluss auf das Ergebnis betrachtet.

In diesem Kapitel soll nun eine eingehendere Betrachtung erfolgen, welche internen Abläufe im Agenten zu den beobachteten Ergebnissen führen, und wie die im letzten Kapitel betrachteten Parameter diese internen Abläufe beeinflussen.

Das Ziel besteht darin, durch Verständnis der internen Abläufe Möglichkeiten zu finden, das Abschneiden des Agenten gezielt zu verbessern.

6.1 Echo State Network

In diesem Abschnitt wird versucht, die Abläufe im ESN genauer zu verstehen.

6.1.1 Linearität der Neuronen

Die Frage, in wie weit ein Problem linear oder nichtlinear veranlagt ist, stellt sich bei nahezu jedem Einsatz von Neuronalen Netzen. In den vorgestellten Versuchen wurden sehr viele verschiedene Parameter zur Erzeugung des ESNs untersucht, einige lieferten deutlich bessere Ergebnisse als andere. Um ein Verständnis der Wirkungsweise dieser Parameter zu erlangen, wurde die Linerität der Neuronen untersucht.

Hierfür wurde vom Agenten die komplette Umgebung auf einem vorgegebenem Weg abgefahren, und die Aktivität der Neuronen im Reservoir aufgezeichnet. Da bei einem ESN nur die Ausgabematrix trainiert wird, hat der Lernzustand keinen Einfluss auf das Verhalten der Neuronen im Reservoir. Es kann also auch ein völlig untrainiertes Netz benutzt werden.

Aus den Aufzeichnungen der Aktivität eines jeden Neurons wurde nun ein Maß für die Linearität bestimmt. Für einen kleinen Wertebereich, um eine Referenzaktivierung u_{ref}, lässt sich die Neuronausgabe mit folgender linearen Gleichung approxmieren:

$$y(u) \approx y(u_{ref}) + \frac{\partial y(u_{ref})}{\partial u_{ref}} \cdot (u - u_{ref}) \tag{6.1}$$

Die Ableitung $\partial y(u_{ref})/\partial u_{ref}$ entspricht dem Anstieg dieser linearen Funktion. Veränderungen von diesem Anstieg über den Wertebereich führen zu nichtlinearem Verhalten.

Für ein Neuron mit linearer Ausgabefunktion entspricht diese Approximation der tatsächlichen Neuronausgabe über den gesamten Wertebereich. Je stärker sich der Anstieg der Neuronausgabe in Bezug zur Aktivierung ändert, desto kleiner wird der Bereich, in dem diese Approximation hinreichend genau funktioniert. Dies rührt daher, dass der Anstieg keine Konstante ist.

Um nun ein Maß für die Linearität eines Neurons zu erhalten, wurde der Anstieg der Neuronausgabe für alle aufgetretenen Aktivierungen u_i berechnet, und zu einer Menge A zusammengefasst:

$$A = \left\{ \frac{\partial f_r(u_i)}{\partial u_i} \;\middle|\; \forall i \right\} \tag{6.2}$$

f_r steht für die Ausgabefunktion des Reservoir-Neurons. Der kleinste aufgetretene Anstieg der Neuronausgabe wurde nun in Verhältnis zum größten gesetzt:

$$l = \frac{\min A}{\max A} \tag{6.3}$$

Dies stellt das verwendete Maß für die Linearität eines einzelnen Neurons dar. Es gilt $0 \leq l \leq 1$. Ein niedriger Wert steht für eine starke Veränderung des Anstiegs über den tatsächlich aufgetretenen Wertebereich, und damit für ein sehr nichtlineares Neuron. Umgekehrt steht ein hoher Wert für ein sehr lineares Neuron.

Im nächsten Schritt wurde nun die Häufigkeitsverteilung von l ermittelt. Um den Einfluss statistischer Effekte zu minimieren, wurde der Durchschnitt aus 5 Netzen gleicher Parameter gebildet.

Auf diese Weise lässt sich ein Überblick über die quantitative Verteilung gewinnen. Ein solches Ergebnis ist in Abbildung 6.1 dargestellt. Die Netzparameter führen scheinbar zu recht vielen nichtlinearen Neuronen mit $0 \leq l \leq 0.1$. Im Bereich $0.1 < l \leq 0.7$ existieren weniger Neuronen, die Anzahl steigt für $0.7 < l \leq 0.9$ wieder an.

In Abbildung 6.2 sind Aktivierung und Ausgabe von 2 verschiedenen Neuronen dargestellt, während der Agent über die gesamte Versuchsumgebung

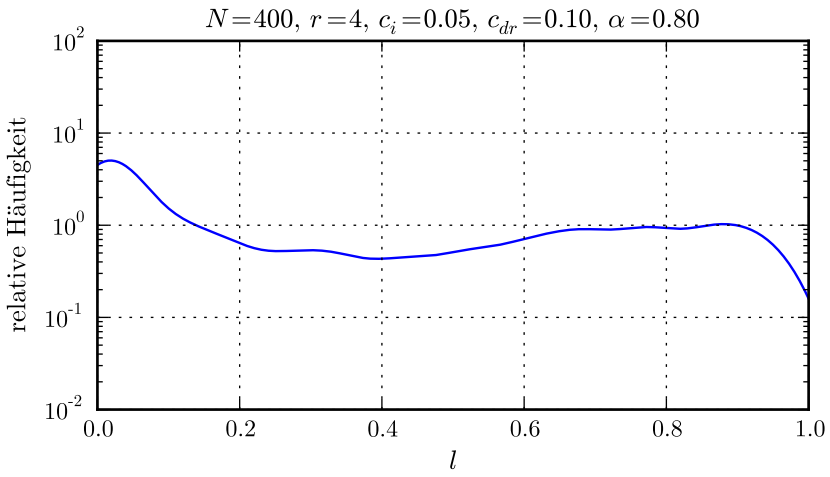


Abbildung 6.1: Linearität der Neuronen im Reservoir an einem Beispiel.

bewegt wurde. Neuron 368 zeigt ein sehr lineares Verhalten, die zugeordnete Linearität beträgt $l = 0.8$. Die Aktivierung wird weitestgehend verzerrungsfrei wieder ausgegeben.

Neuron 123 zeigt mit $l = 0.03$ dagegen ein sehr nichtlineares Verhalten. Die Ausgabe erreicht sehr oft Werte nahe -1. Im Bereich $t > 10000$ ist deutlich zu sehen, dass die negativen Spitzen der Aktivierung in der Ausgabe abgeschnitten werden, während die positiven Spitzen ausgegeben werden. Eine Betrachtung der Hüllkurve der Aktivierung im Bereich $1000 < t < 4000$ zeigt ebenso, dass der untere Teil der Aktivierung in der Ausgabe nur stark gedämpft erscheint.

Als nächstes wurde nun der Einfluss der verschiedenen Parameter untersucht. Der Parameter r hat in diesem Zusammenhang vermutlich den größten Einfluss, da er direkt den Wertebereich der Aktivierung der mit der Eingabeschicht verbundenen Neuronen beeinflusst. Niedrige Werte für r sollten die Linearität begünstigen, hohe Werte dagegen Nichtlinearität.

Abbildung 6.3 zeigt den ermittelten Einfluss von r auf ein Netz mit den dargestellten Parametern. Überwiegen für $r = 1$ noch lineare Neuronen, besteht das Netz für $r = 16$ fast vollständig aus sehr nichtlinearen Neuronen. Die theoretische Betrachtung hat sich somit bestätigt.

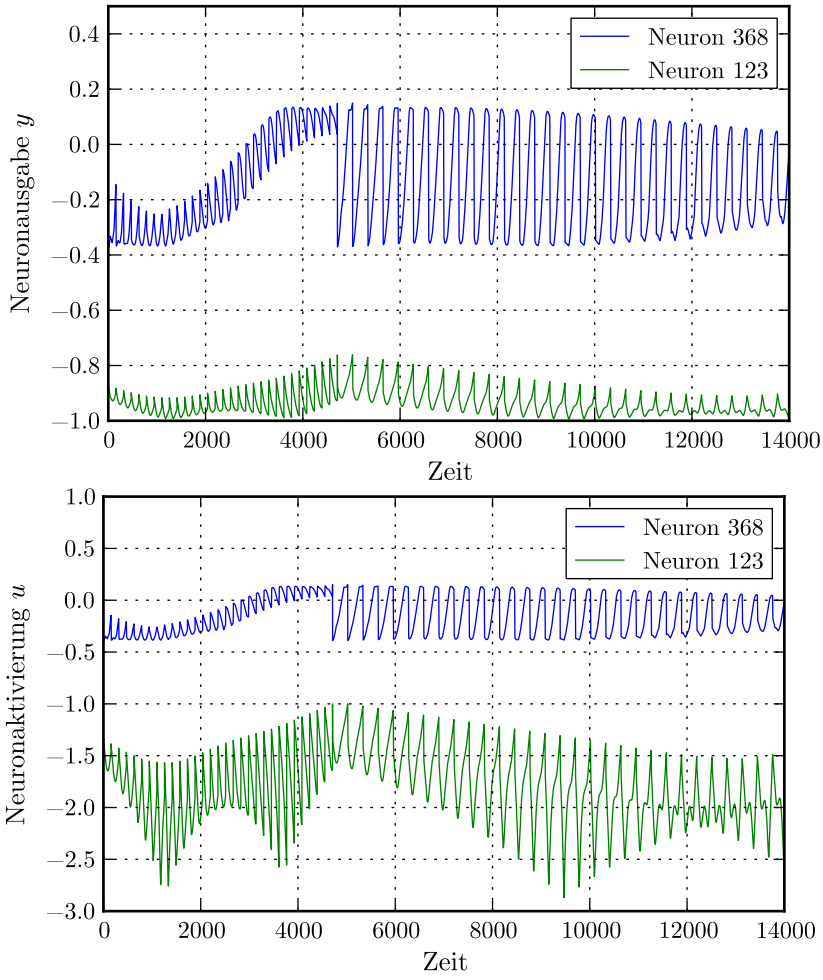


Abbildung 6.2: Ein als sehr linear klassifiziertes und ein als sehr nichtlinear klassifiziertes Neuron, oben die Ausgabe, unten die Neuronaktivierung. Während das weitestgehend lineare Neuron 368 die Aktivierung ziemlich direkt wiedergibt, wird die Aktivierung bei Neuron 123 stark verzerrt, vor allem für $t > 10000$.

Als nächstes wurde der Einfluss von c_{dr} untersucht, das Ergebnis ist in Abbildung 6.4 dargestellt.

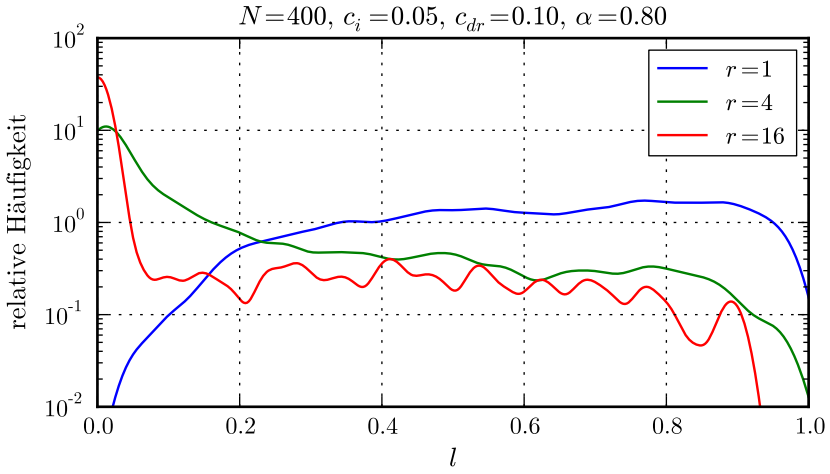


Abbildung 6.3: Linearität der Neuronen für verschiedene Werte für r. Der Einfluss auf die Linearität ist sehr deutlich.

Erstaunlicherweise scheint dieser Parameter keinen messbaren Einfluss zu besitzen. Eine stärkere Rückkopplung im Reservoir sollte in der Theorie auch zu einer größeren Neuronaktivierung führen, dies ist jedoch offenbar nicht der Fall. Eine mögliche Erklärung bestünde darin, dass die Rückkopplungen im Reservoir allgemein nur sehr schwach ausgeprägt sind.

Es folgt der Parameter c_i. Die Ergebnisse in Abbildung 6.5 zeigen, ebenso wie r, einen großen Einfluss. Die Größe der Aktivierung eines Neurons steigt mit jedem Eingabewert, mit dem es verbunden ist. Daher ist dieses Ergebnis nicht verwunderlich.

Damit ließe sich auch der variierende Einfluss von r und c_i auf den Erfolg des Agenten in Kapitel 5.1.1 erklären. Beide Parameter beeinflussen die Linearität, es scheint dafür jedoch einen optimalen Bereich zu geben. Werden die Neuronen durch große Werte für r und c_i dagegen zu nichtlinear, sinkt der Erfolg des Agenten. Ebenso verhält es sich für zu kleine Werte, dann werden die Neuronen zu linear.

Als nächstes wurde α betrachtet. Obwohl der Einfluss hier nur sehr schwach ausgeprägt ist, lässt er sich bei genauerer Betrachtung im Bereich $l < 0.1$

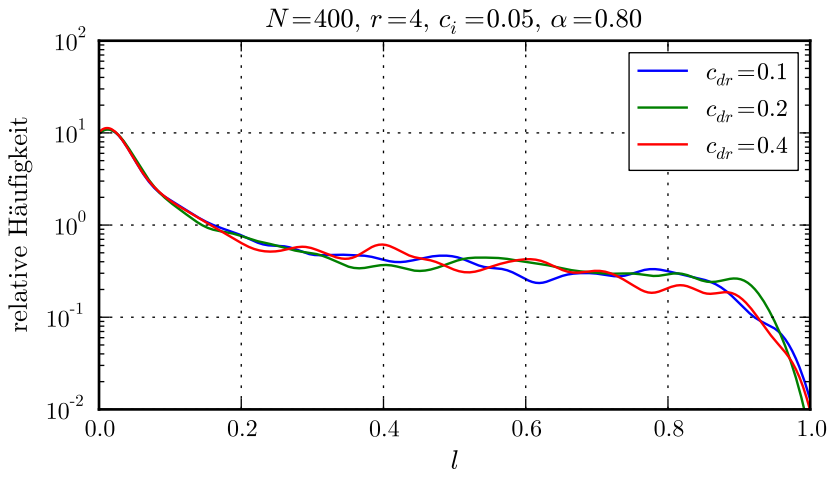


Abbildung 6.4: Linearität der Neuronen für verschiedene Werte für c_{dr}. Entgegen der Erwartung ist kein Einfluss sichtbar.

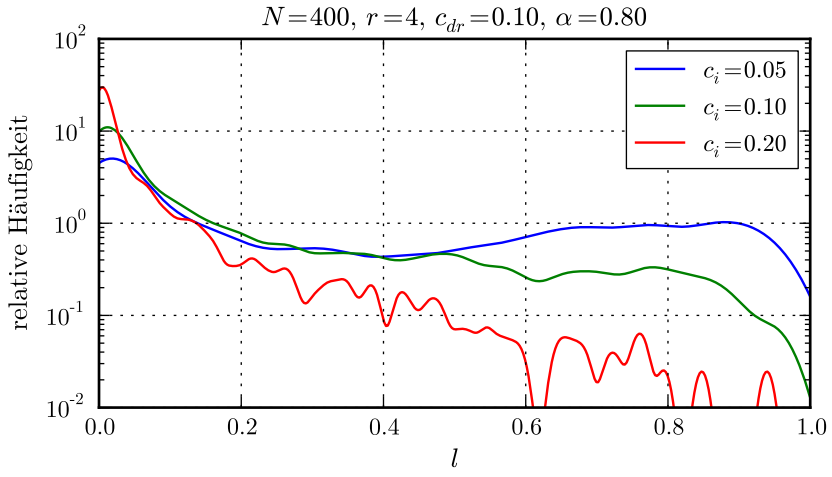


Abbildung 6.5: Linearität der Neuronen für verschiedene Werte für c_i.

und $l > 0.6$ erkennen. Größere Werte führen zu einer stärkeren Rückkopplung im Netz und damit zu einer größeren Nichtlinearität der Neuronen. Der geringe Einfluss zeigt jedoch auch hier, dass die Rückkopplungen nur einen kleinen Teil zur Aktivierung der Reservoirneuronen beitragen können.

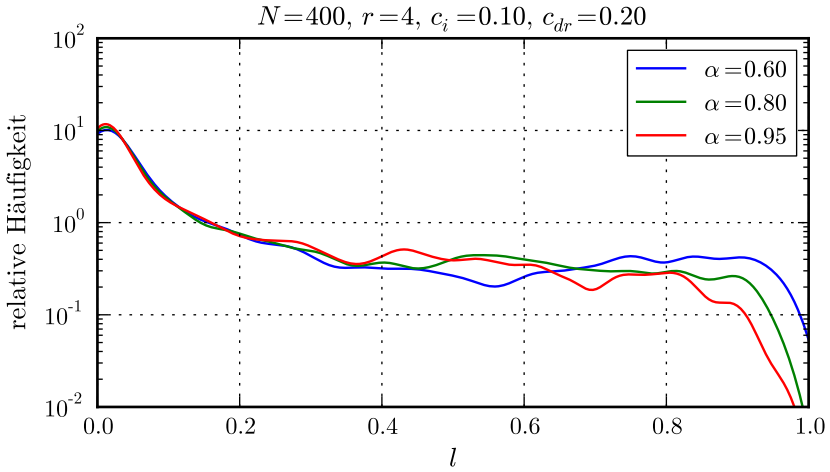


Abbildung 6.6: Linearität der Neuronen für verschiedene Werte für α.

Schließlich wurde noch die Neuronenanzahl betrachtet. Die Ergebnisse in Abbildung 6.7 zeigen keinen Einfluss. Einerseits führen mehr Neuronen auch zu mehr Verbindungen. Bei fester Verbindungsdichte steigt die Anzahl der Verbindungen der Eingabewerte zum Reservor linear mit N, die Anzahl der Verbindungen im Reservoir selbst steigt dagegen mit dem Quadrat von N. Auf der anderen Seite verteilt sich die Aktivierung der Neuronen auch stärker im Reservoir, und es müssen mehr Neuronen aktiviert werden. Diese Effekt scheinen sich gegenseitig aufzuheben, oder sind nur sehr schwach ausgeprägt, was gerade in Anbetracht des Einflusses von c_{dr} nicht verwunderlich wäre.

Es scheinen nur 2 Parameter einen messbaren Einfluss auf die Linearität der Reservoirneuronen zu haben, r und c_i. Diese beiden Parameter üben auch einen großen Einfluss auf den Erfolg des Agenten in Kapitel 5.1.1 aus. Es scheint einen recht großer Bereich der Linearität zu geben, in dem der Agent zufriedenstellend funktioniert. In besonders extremen Situationen, also besonders großer Linearität oder Nichtlinearität, verschlechtert sich das Verhalten jedoch. Dies ist der Fall, wenn sowohl r als auch c_i sehr große

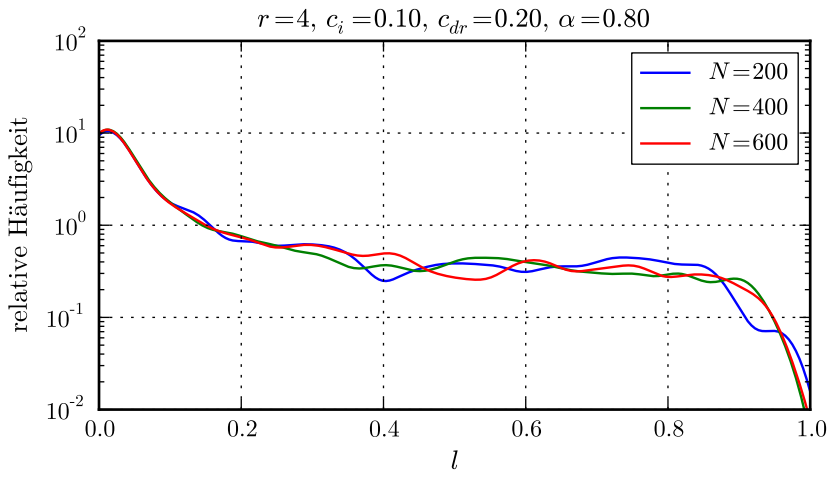


Abbildung 6.7: Linearität der Neuronen für verschiedene Werte für N. Es ist kein Einfluss messbar.

oder sehr kleine Werte annehmen.

6.1.2 Netzdynamik

Ein wichtiger Aspekt von ESNs sind die Rückkopplungen innerhalb des Reservoirs. Diese beeinflussen den Ausgabewert des Netzes, und beeinflussen dadurch ebenso das Lernverhalten des Agenten.

Der Einfluss der Netzparameter auf diese Rückkopplungen wird nun genauer betrachtet. Die erste Herausforderung besteht darin, ein Maß hierfür herzuleiten. Zu diesem Zweck wurden mit dem Roboter Punkte in der Umgebung aus verschiedenen Richtungen angefahren. Wurde der Punkt erreicht, wurde die Netzaktivierung aufgezeichnet. Der Vergleich der Netzaktivierungen im selben Punkt, jedoch jeweils mit verschiedenen Vergangenheiten, erlaubt nun Rückschlüsse auf den Einfluss der Vergangenheit auf die Netzaktivierung, und damit eine Einschätzung der Dynamik.

Das Ergebnis dieser Untersuchung für einen einzelnen Punkt ist eine Menge von Netzaktivierungsvektoren, y_1 bis y_m. Um die Abweichung zwischen diesen in einem einzelnen Wert zusammenzufassen, wurde nun der Durchschnitt der paarweisen quadratischen Abweichung berechnet:

$$E_D = \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} \frac{||y_i - y_j||^2}{N} \tag{6.4}$$

Der Therm $||y_i - y_j||^2/N$ entspricht dem Effektivwert von $y_i - y_j$. Damit steht E_D anschaulich für die durchschnittliche quadratische Abweichung der Ausgabe eines einzelnen Neurons. Im letzten Schritt wurde E_D für viele verschiedene Positionen auf der Karte berechnet und arithmetisch gemittelt. Die erzielten Ergebnisse sind in Tabelle 6.1 dargestellt, es wurden 100 Positionen gemittelt, die jeweils aus 4 verschiedenen Richtungen angefahren wurden. Da die sich ergebende Werte sehr klein sind, wurde jeder Wert mit 10^3 skaliert, um eine bessere Darstellung zu erlauben.

Es ergeben sich folgende Beobachtungen:

- Der Einfluss von α ist deutlich zu sehen. Dies war zu erwarten, da über α allgemein die Stärke der Rückkopplungen im Reservoir beeinflusst wird.
- Große Werte für r scheinen die Netzdynamik zu verringern. Dieser Effekt lässt sich über die Sättigung der Reservoirneuronen durch die

hohen Aktivierungen erklären. Dadurch werden die Rückkopplungen abgeschwächt.

- Der Einfluss von c_i und c_{dr} scheint komplexerer Natur zu sein. So führt ein Ansteigen dieser beiden Parameter für $\alpha = 0.6$ und $r \leq 4$ zu einer Zunahme der Netzdynamik, für $\alpha = 0.95$, $N = 600$ und $r = 1$ dagegen zu einer deutlichen Abnahme. Es ist zu vermuten, dass c_i, ähnlich wie r, die Dynamik abschwächt, c_{dr} sie dagegen verstärkt.

- Ein großes Reservoir scheint die Rückkopplungen in einigen Fällen zu verstärken.

			N				N				N			
r	c_i	c_{dr}	100	200	400	600	100	200	400	600	100	200	400	600
1	0.05	0.10	1.38	1.37	1.44	1.49	2.82	2.73	3.01	3.23	4.83	4.88	5.55	6.26
	0.10	0.20	1.59	1.67	1.69	1.72	2.92	3.20	3.22	3.27	4.85	5.45	5.40	5.47
	0.20	0.30	1.92	1.91	1.87	1.91	3.45	3.29	3.25	3.35	5.62	5.01	4.98	5.20
	0.30	0.40	2.04	2.03	2.04	2.03	3.43	3.52	3.43	3.44	5.05	5.40	5.07	5.13
4	0.05	0.10	1.32	1.51	1.49	1.58	2.06	2.45	2.43	2.60	2.82	3.47	3.49	3.77
	0.10	0.20	1.67	1.63	1.66	1.69	2.52	2.48	2.56	2.55	3.34	3.34	3.49	3.43
	0.20	0.30	1.82	1.82	1.82	1.84	2.67	2.64	2.67	2.72	3.46	3.41	3.47	3.55
	0.30	0.40	1.93	2.02	1.91	1.92	2.75	2.93	2.76	2.79	3.50	3.78	3.56	3.60
16	0.05	0.10	1.41	1.79	1.71	1.85	1.95	2.55	2.39	2.59	2.44	3.21	2.99	3.22
	0.10	0.20	1.67	1.62	1.67	1.77	2.23	2.20	2.27	2.41	2.69	2.70	2.78	2.94
	0.20	0.30	1.66	1.76	1.72	1.68	2.28	2.39	2.37	2.29	2.77	2.87	2.89	2.80
	0.30	0.40	1.77	1.84	1.80	1.77	2.40	2.53	2.46	2.42	2.91	3.08	2.99	2.94
			$\alpha = 0.60$				$\alpha = 0.80$				$\alpha = 0.95$			

Tabelle 6.1: Darstellung der ermittelten Netzdynamik für verschiedene Parameterkombinationen. Für jeden Parametersatz wurden 6 Simulationen durchgeführt und der Mittelwert gebildet. Die Werte wurden zur besseren Darstellung mit 10^3 multipliziert.

Um den Einfluss der Rückkopplungen im Vergleich zu dem Einfluss der Eingabewerten selbst abschätzen zu können, wurde nun letzterer ermittelt. Das Vorgehen hierfür entsprach weitestgehend dem zum Ermitteln der Dynamik, jedoch wurde nicht jeweils eine Position aus verschiedenen Richtungen angefahren, sondern verschiedene Positionen aus der selben Richtung.

Das Ergebnis dieser Untersuchung ist in Tabelle 6.2 dargestellt. Im Gegensatz zu Tabelle 6.1 wurden die Werte nicht skaliert. Der Einfluss durch Veränderungen in den Eingabewerten scheint die Netzrückkopplungen dementsprechend um Größenordnungen zu überwiegen.

			N				N				N			
r	c_i	c_{dr}	100	200	400	600	100	200	400	600	100	200	400	600
1	0.05	0.10	0.10	0.10	0.10	0.11	0.12	0.11	0.12	0.12	0.13	0.13	0.14	0.14
	0.10	0.20	0.13	0.14	0.14	0.14	0.14	0.15	0.16	0.16	0.15	0.16	0.17	0.18
	0.20	0.30	0.20	0.19	0.19	0.19	0.21	0.20	0.20	0.20	0.23	0.21	0.21	0.21
	0.30	0.40	0.21	0.21	0.21	0.21	0.22	0.22	0.22	0.22	0.23	0.24	0.23	0.23
4	0.05	0.10	0.18	0.20	0.19	0.20	0.19	0.21	0.21	0.21	0.20	0.22	0.22	0.23
	0.10	0.20	0.27	0.26	0.27	0.27	0.28	0.27	0.28	0.28	0.28	0.28	0.29	0.29
	0.20	0.30	0.39	0.37	0.36	0.36	0.40	0.37	0.37	0.36	0.40	0.37	0.37	0.37
	0.30	0.40	0.39	0.40	0.39	0.39	0.40	0.41	0.40	0.40	0.41	0.42	0.40	0.40
16	0.05	0.10	0.24	0.26	0.26	0.27	0.25	0.27	0.26	0.28	0.25	0.28	0.27	0.28
	0.10	0.20	0.39	0.38	0.38	0.38	0.39	0.38	0.38	0.38	0.40	0.39	0.39	0.39
	0.20	0.30	0.54	0.52	0.51	0.50	0.54	0.52	0.51	0.50	0.54	0.52	0.52	0.51
	0.30	0.40	0.53	0.55	0.54	0.53	0.53	0.55	0.54	0.53	0.53	0.55	0.54	0.53
			$\alpha = 0.60$				$\alpha = 0.80$				$\alpha = 0.95$			

Tabelle 6.2: Im Vergleich zur Dynamik der Einfluss durch statische Änderungen, d.h. Änderungen in den Eingabewerten. Die Werte wurden nicht skaliert.

Der Einfluss der einzelnen Parameter auf den Einfluss der Eingabewerte auf die Neuronaktivierung deckt sich weitestgehend mit der Analyse der Linearität im letzten Abschnitt. Dies ist nicht verwunderlich, da eine durch die Eingabewerte hervorgerufene, hohe Aktivierung der Neuronen im Reservoir ebenso zu einer großen Nichtlinearität der Neuronen, wie zu einer großen Abhängigkeit der Ausgabe von den Eingabewerten führt.

6.1.3 Experimente

Bei der Untersuchung der Linearität der Reservoirneuronen hat sich gezeigt, dass die Rückkopplungen im Reservoir scheinbar nur einen recht geringen Einfluss auf die Ausgabewerte der Neuronen haben. Diese Vermutung wird durch die Analyse der Netzdynamik unterstützt, und liefert auch eine Erklärung für den geringen Einfluss von α auf das Verhalten des Agenten.

Es wurde nun versucht, den Einfluss dieser Rückkopplungen zu verstärken. Analog zum Einfluss von c_i und r auf die Wirkung der Eingabewerte auf das Reservoir, werden die Rückkopplungen von c_{dr} und α beeinflusst.

Leider wird α im klassischen Echo State Network auf den Wertebereich $0 < \alpha < 1$ eingeschränkt. Dies hängt mit der gewünschten Echo State Eigenschaft zusammen [Jaeger, 2002], die aussagt, dass die Reservoiraktivierung ausschließlich von den vergangenen Eingabewerten des Netzes abhängt. Größere Werte für α können zu instabilem Netzverhalten führen, in dem Sinne, dass sich Reservoiraktivierungen immer weiter aufschaukeln. Ebenso können sich selbst haltende Zustände entstehen, wenn z.B. ein Neuron mit einem sehr hohen positiven Gewicht auf sich selbst zurückkoppelt.

Während die mögliche Instabilität im Allgemeinen durchaus ein Problem darstellt, ist dies für die in dieser Arbeit verwendeten Netze nicht der Fall. Die Reservoir-Neuronen besitzen eine beschränkte Ausgangsfunktion mit $-1 < y < 1$, demzufolge ist ein Aufschaukeln der Netzaktivierungen nicht möglich. Um zu sehen, wie sich der Agent bei einer starken dynamischen Komponente des Netzes verhält, wurde daher eine Reihe von Simulationen mit $\alpha > 1$ durchgeführt. Einer dieser Versuche ist in Abbildung 6.8 dargestellt.

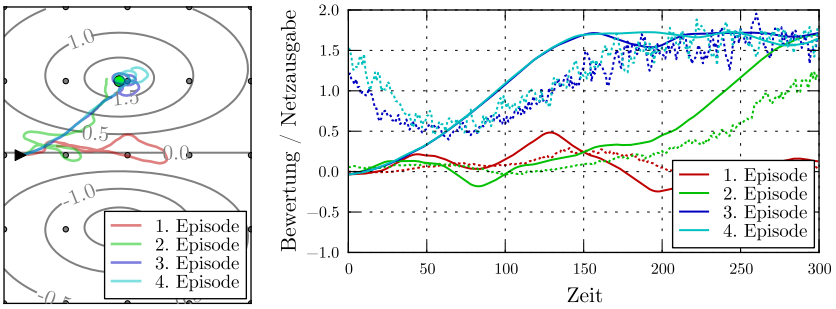


Abbildung 6.8: Ergebnisse für $r = 4$, $n = 400$, $c_i = 0.1$, $c_{dr} = 0.5$, $\alpha = 5$. Das starke Rauschen der Netzausgabe weist auf Oszillationen im Reservoir hin, trotzdem schneidet der Agent ab der 3. Episode sehr gut ab.

Im Vergleich zu den vorherigen Versuchen fallen eine Reihe von Unterschieden auf. So variiert der Weg, den der Agent fährt, in den ersten Episoden deutlich stärker als in den vorherigen Versuchen.

Es scheint, dass der Agent in der ersten Episode willkürlich durch die Umgebung fährt. Zum Ende der 2. Episode hat er die Belohnung gefunden. In den folgenden Episoden ist das Verhalten nun sehr zielgerichtet,

der Agent fährt direkt zur Belohnung. Im Gegensatz zu den vorherigen Versuchen fängt der Agent bei der Belohnung zwar auch an Kreise zu fahren, diese wiederholen sich jedoch nicht. Es ist daher davon auszugehen, dass dieses Verhalten auf dem Wissen des Agenten um den optimalen Aufenthaltsort basiert ist, und nicht auf einer möglichen Instabilität des Verhaltens.

Die starken Schwankungen der Netzausgabe legen nahe, dass Teile des Netzes oszillieren. Interessant ist dieser Aspekt in Hinblick darauf, dass genau diese Form von Netzinstabilität üblicherweise vermieden wird. Im hier betrachteten Anwendungsfall scheint sie aber, ganz im Gegensatz zu den Erwartungen, den Erfolg des Agenten zu steigern.

Eine denkbare Begründung ist, dass die Oszillationen im Netz die Stabilität des Agenten erhöhen. In den vorherigen Versuchen wurde stets beobachtet, dass der Agent nach einer gewissen Zeit in jeder Episode anfängt, sich im Kreis zu bewegen. Dieses Verhalten ist nur dann von Dauer, wenn der Agent nach einer Gewissen Zeit exakt zu einem früheren Zustand zurückkehrt.

Die Oszillationen im Netz können an dieser Stelle als Zufallselement aufgefasst werden, welches genau das verhindert. Interessant ist in diesem Zusammenhang, dass das absichtliche Hinzufügen von Rauschen schon von [Jaeger, 2002] in Zusammenhang mit dem Offline-Training von ESNs vorgeschlagen wird, um die Stabilität des Trainings zu verbessern.

Bezogen auf den durchgeführten Versuch ließe sich das bessere Abschneiden des Agenten auch so interpretieren, dass er durch das Rauschen mehr unterschiedliche Situationen durchläuft und so mehr Wissen ansammelt.

6.2 Training

In diesem Abschnitt wird das Trainging des ESNs genauer untersucht. Für ein erfolgreiches Verhalten des Agenten ist es wichtig, dass dieser in der Lage ist, aus Erfahrungen zu lernen und diese zu generalisieren, so das künftige Entscheidungen von diesem Wissen profitieren können.

6.2.1 Veränderung der Auslesematrix

Um einen Eindruck zu bekommen, wann der Agent bzw. das ESN lernt, also sich verändert, und wann nicht, wurde die Veränderung der ESN Auslesematrix über die Zeit aufgetragen. Der auf dem realen Roboter durchgeführte Versuch aus Abschnitt 5.3 wurde als Grundlage benutzt.

Die Gewichte werden beim Online-Training entsprechend folgender Gleichung angepasst:

$$\mathbf{W_o}(k+1) = \mathbf{W_o}(k) - \delta \frac{\partial E(k)}{\partial \mathbf{W_o}(k)} \tag{6.5}$$

Diese Anpassung korreliert mit dem Fehler der Netzausgabe, da dieser linear in die Anpassung der einzelnen Gewichte eingeht. Ebenso beeinflusst die Neuronaktivierungen des Reservoirs die Anpassung der Ausgabematrix über $\partial E(k)/\partial \mathbf{W_o}(k)$

Die Veränderung der Auslesematrix zu einem gegebenen Zeitpunkt k wurde folgendermaßen definiert:

$$\Delta \mathbf{W_o}(k) = ||\mathbf{W_o}(k) - \mathbf{W_o}(k-1)||^2 \tag{6.6}$$

Abbildung 6.9 zeigt die Veränderung in einer einzelnen Episode. Der komplette Versuchsablauf ist in Abbildung 5.13 dargestellt. In Abbildung 6.10 ist die, über jeweils eine vollständige Episode gemittelte, Veränderung dargestellt.

Es sind deutlich verschiedene Phasen des Lernens erkennbar. Diese scheinen stark damit in Verbindung zu stehen, in wie weit das Wissen des Agenten über seine Umgebung mit seinen Erfahrungen übereinstimmen. Große Abweichungen führen entsprechend zu großen Veränderungen der Auslesematrix.

Auffällig ist, dass das Netz auch in Episoden, in denen sich das Verhalten des Agenten nicht weiter verändert, fortwährend angepasst wird. Eine mögliche Erklärung hierfür erfolgt im nächsten Abschnitt.

Davon abgesehen entsprechen die Beobachtungen weitestgehend den Erwartungen. Am Anfang des Versuchs finden etwas größere Veränderungen statt, der Agent lernt seine Umgebung kennen. Hat sich das Verhalten des Agenten stabilisiert, fallen auch die Veränderungen niedrig aus.

In der 6. Episode trifft der Agent auf die veränderte Umgebung, es finden sehr umfangreiche Anpassungen statt. Diese setzen sich fort, bis der Agent die Belohnung in der 9. Episode wiedergefunden hat.

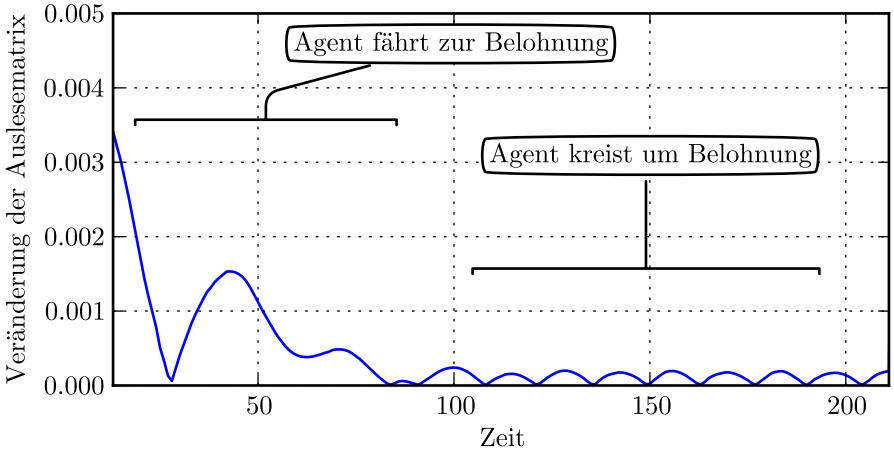


Abbildung 6.9: Veränderung der Auslesematrix innerhalb der 3. Episode.

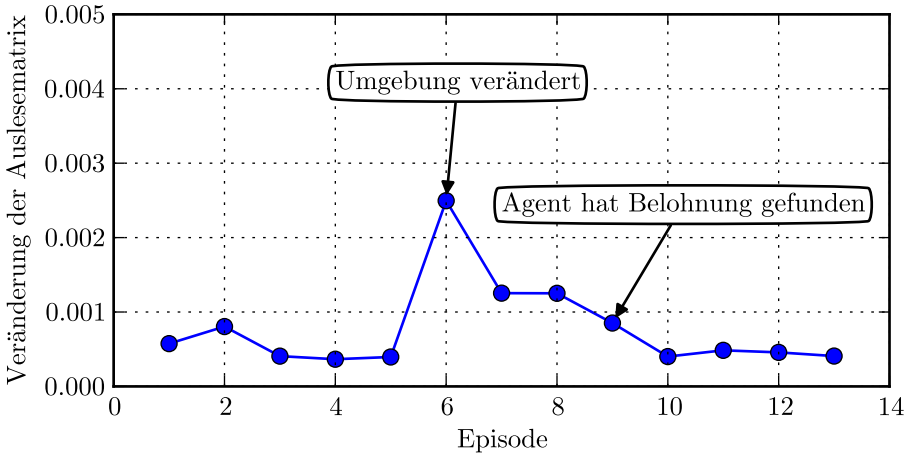


Abbildung 6.10: Veränderung der Auslesematrix über die jeweilige Episode gemittelt. Nachdem die Umgebung verändert wurde, hat sich das Wissen des Agenten besonders stark verändert. Ebenso sind stärkere Veränderungen in den Episoden sichtbar, in denen der Agent die Belohnung zum ersten mal gefunden bzw. wiedergefunden hat.

6.2.2 Gleichanteil der Netzausgabe

Die Darstellung des Wissens des Agenten über seine Umgebung in Kapitel 5 zeigt eine sehr starke Verschiebung des absoluten Wertebereichs der Netzausgabe, obwohl sich der, für das Verhalten des Agenten wichtige, Unterschied zwischen den einzelnen Positionen nicht verändert.

Um dies zu verdeutlichen, ist in Abbildung 6.11 eine Gegenüberstellung der Netzausgabe zu zwei, zeitlich nahe beieinanderliegenden, Zeitpunkten des in 5.3 durchgeführten Versuchs dargestellt.

Zwischen den dargestellten Zeitpunkten liegen lediglich 30 Zeitschritte, diese stammen zudem aus einer Episode, in der sich das Verhalten des Agenten bereits stabilisiert hat. Trotzdem hat sich der Wertebereich der Netzausgabe um etwa 0.25 verschoben, wie an den Skalen rechts neben den Darstellungen zu sehen ist. Der Umfang des Wertebereichs hat sich ebenfalls verändert.

Erklären lässt sich dieses Phänomen, wenn angenommen wird, dass im Netz einige Neuronen mit einer weitestgehend konstanten Ausgabe existieren.

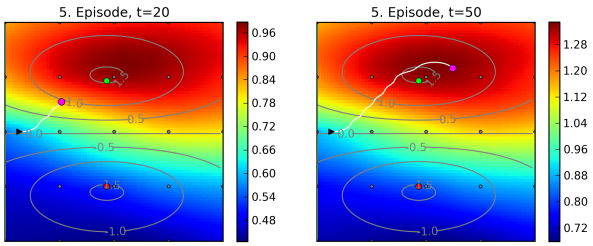


Abbildung 6.11: Obwohl die Darstellungen 2 nah beieinander liegende Zeitpunkte zeigen, in denen sich das Wissen des Agenten kaum verändert hat, gibt es eine große Verschiebung im absoluten Wertebereich der Ausgabe.

Dies wird nun an einem Beispiel betrachtet.

Sei die Reservoirausgabe z.B. definiert durch:

$$y_r(k) = \begin{pmatrix} f(k) \\ 1 \end{pmatrix}$$

Die Ausgabefunktion der Reservoirneuronen soll $f_r(x) = x$ sein. Die Ausgabematrix sei nun definiert durch $\mathbf{W_o}(k) = (w_1(k), w_2(k))$. Damit ergibt sich die Netzausgabe zu:

$$y(k) = \mathbf{W_o}(k) y_r(k) = f(k) \cdot w_1 + w_2$$

Das Netz soll darauf trainiert werden, die Funktion $f(k)$ zu approximieren. Die Ausgabegewichte werden hierfür in jedem Lernschritt entsprechend dem Online-Trainingsalgorithmus angepasst. Zur besseren Übersichtlichkeit sei $\Delta y(k) = f(k) - y(k)$ die Abweichung der Netzausgabe.

$$\begin{aligned} w_i(k+1) &= w_i(k) - \delta \frac{\partial E}{\partial w_i} \\ \partial E(k)/\partial w_1 &= -2 \cdot f(k) \cdot \Delta y(k) \\ \partial E(k)/\partial w_2 &= -2 \cdot \Delta y(k) \\ w_1(k+1) &= w_1(k) + 2\delta \cdot f(k) \cdot \Delta y(k) \\ w_2(k+1) &= w_2(k) + 2\delta \cdot \Delta y(k) \end{aligned}$$

Ist die Netzausgabe zu klein, wird w_2 also entsprechend erhöht, ist es zu groß, wird w_2 entsprechend verringert. Dies passiert, obwohl in y_{r2} keinerlei Information enthalten ist.

Erreicht $\Delta y(k)$ nun einen Wert nahe 0 hört das Training auf. w_2 begünstigt dies, da der Fehler in der Netzausgabe durch die Anpassung von w_2 schneller gegen 0 tendiert, als es ohne der Fall wäre.

Verändert sich $f(k)$ sehr langsam, wie es bei der Positionsbewertung U der Fall ist, wird die Netzausgabe zu einem gewissen Anteil von w_2 bestimmt. Dadurch hört das Training auf bevor $w_1 = 1$ erreicht wird, womit das Netz fertig trainiert wäre. Ändert sich $f(k)$ dagegen sehr schnell, heben sich die Veränderungen von w_2 im Mittel auf.

In den durchgeführten Versuchen durchläuft die Positionsbewertung, und damit auch die Ausgabe auf welche das ESN trainiert wird, den Wertebereich sehr langsam. Eine Episode besteht aus maximal 300 Zeitschritten. In einer erfolgreichen Episode hat sich der Agent der Belohnung nach 150 Zeitschritten angenährt, in dieser Zeit steigt die Bewertung langsam von 0 auf ≈ 1.75. Über die restliche Zeit bleibt sie weitestgehend konstant.

Ist die Belohnung erreicht, werden die Ausgabegewichte der konstanten Neuronen im Reservoir nun solange angepasst, bis $E(k) \approx 0$ eintritt. Ab diesem Punkt findet praktisch keine weitere Anpassung der Ausgabematrix statt. Erst wenn sich die Bewertung wieder verändert, werden auch wieder die Gewichte der Neuronen angepasst.

Diese Anpassung der Gewichte der konstanten Neuronen verhindert, dass Gewichte von Neuronen angepasst werden, die zu einer stimmigen Ausgabe über den gesamten Zeitraum beitragen würden. Würde die Bewertung, und damit das Lehrersignal im Online-Training, in jedem kurzen Zeitraum einen großen Wertebereich abdecken, würden sich die Anpassungen der Gewichte konstanter Neuronen gegenseitig kompensieren. In den durchgeführten Versuchen ist dies jedoch nicht der Fall.

Dadurch besteht die Netzausgabe zu einem gewissen Anteil aus einer Konstante, die durch das Training lediglich darauf angepasst wird, die aktuell trainierte Ausgabe nachzubilden. Wird diese erreicht, hört das Training auf, und die Gewichte der Neuronen, die zu einer sinnvollen Aufgabe betragen würden, werden ebenso nicht weiter trainiert.

Bei diesem Effekt handelt es sich um eine grundlegende Einschränkung des Online-Trainingsalgorithmus. Die Auswirkungen sind in den Netzausgaben in Abbildung 5.13(a) sichtbar. So ist diese zu Anfang von Episode 2 geringer

als die Bewertung, da Episode 1 mit einer negativen Bewertung geendet hat. Umgekehrt ist sie zu Anfang der 3. Episode größer, da die 2. Episode mit einer sehr großen Bewertung endet.

6.2.3 Offline Learning

Aufgrund der angesprochenen Probleme des Online-Trainings wird in diesem Abschnitt das Verhalten des Agenten unter Verwendung des eingeführten Offline-Trainings untersucht, um einen Überblick zu erlangen, in wie weit das Online-Training den Erfolg des Agenten limitiert.

Dabei werden alle Lerndaten, die beim Online Lernen sequenziell trainiert werden, gespeichert, und in jedem Lernschritt in ihrer Gesamtheit präsentiert. Aufgrund dessen eignet sich Offline Training nicht in einer veränderlichen Umgebung. Es wurde daher lediglich eine konstante Umgebung untersucht. Die Ergebnisse sind in Abbildung 6.12 und Abbildung 6.13 dargestellt. Die Parameter wurden folgendermaßen gewählt: $N = 400$, $r = 4$, $c_i = 0.10$, $c_{dr} = 0.20$, $\alpha = 0.80$. Das Ergebniss zeigt, dass der Erfolg des Agenten durch den Online Lernalgorithmus stark begrenzt wird.

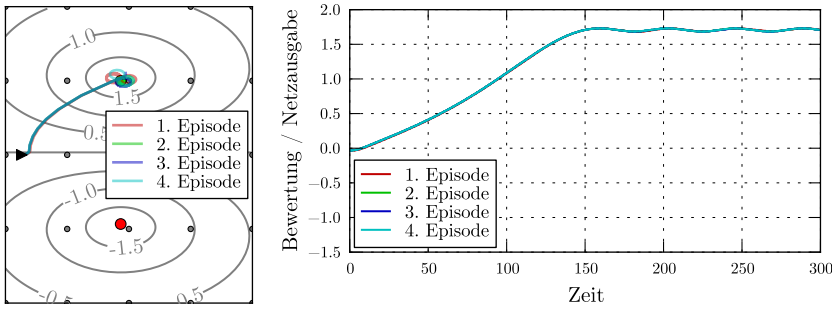


Abbildung 6.12: Der Agenten mit Offline-Lernalgorithmus. Zu sehen ist ein nahezu perfektes Verhalten, schon in der ersten Episode wird die Belohnung gefunden. Interessantes Detail: Der Agent fährt nahezu senkrecht zu den Linien gleicher Bewertung. Dies entspricht exakt dem Gradientenaufstieg auf der Bewertungsfläche, der im Aktor umgesetzt wird.

Eine der Herausforderungen bei der Wahl der ESN Parameter bestand also darin, diese so zu bestimmen, dass das Netz im Besonderen mit Online-Training funktioniert. Es wäre demnach nicht verwunderlich, wenn sich

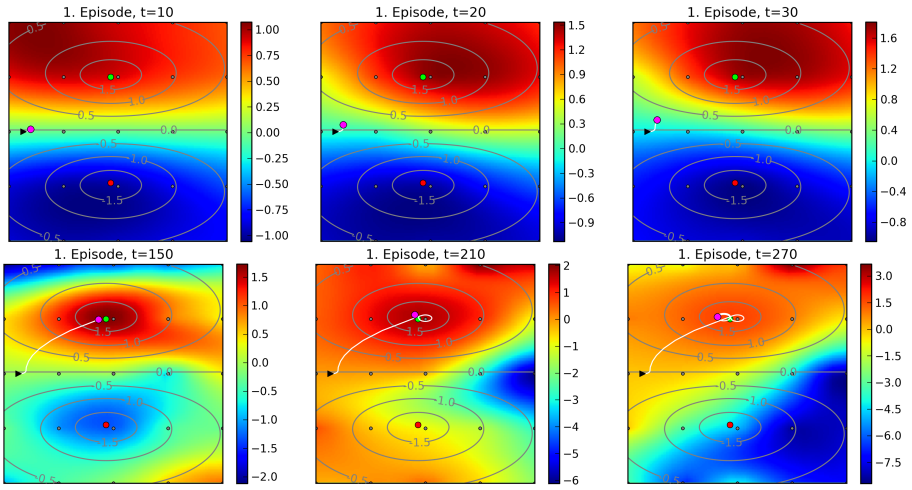


Abbildung 6.13: Darstellung des Wissens des Agenten über die Umgebung. Im Gegensatz zu Online Learning produziert das Netz für die Bereiche, die der Agent noch nicht besucht hat, mitunter sehr große Ausgaben.

für einen anderen Lernalgorithmus andere Parameter als besser erweisen würden.

6.3 Adaptivität

In diesem Abschnitt soll die Anpassungsfähigkeit des Agenten an neue Situationen gesondert untersucht werden. Bei den bisherigen Versuchen mit Veränderungen der Umgebung fand diese stets zu Begin der 5. Episode statt. Zu beobachten war darauf folgend stets eine Anpassungsphase, in der sich das Wissen des Agenten an die neue Situation angepasst hat. Nach dieser Anpassung war der Agent in der Lage, die neue Belohnung aufzusuchen.

Die Länge dieser Anpassungsphase wird scheinbar vom Wissen des Agenten bestimmt. Die Resultate in Kapitel 5.3 zeigen, dass der Agent erst dann die neue Position der Belohnung findet, wenn das veraltete Wissen abgebaut wurde.

Daher wurde nun die Umgebung zu jeweils anderen Zeitpunkten verändert.

Die Frage die an dieser Stelle geklärt werden soll, ist, in wie weit sich frühere Erfahrungen des Agenten auf die Anpassungsfähigkeit an neue Situationen auswirken. Die Resultate dieser Simulationen sind in Abbildung 6.14 dargestellt.

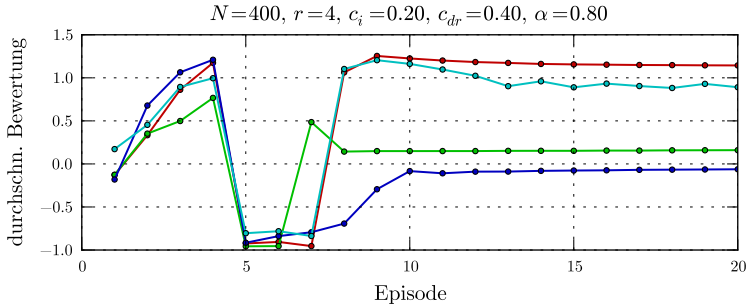
Die steigende Bewertung in den ersten Episoden zeigt die initiale Lernphase, in der sich der Agent langsam an die Belohnung herantastet. Wird nun die Umgebung verändert, agiert der Agent basierend auf seinem nun veralteten Wissen, und es folgt entsprechend eine schlechte Bewertung.

Die Länge dieser Phase scheint direkt davon abzuhängen, wie lange die Umgebung unverändert geblieben ist. Dies legt die Vermutung nahe, dass auch dann noch Wissen im ESN gespeichert wird, wenn sich das Verhalten des Agenten bereits stabilisiert hat, und auch die Darstellung des Wissens des Agenten über die Umgebung keine weitere Veränderung aufzeigt.

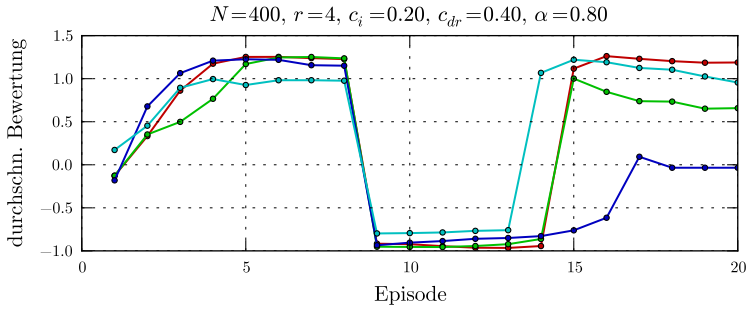
Nach dieser Lernphase zeigt sich an einigen Stellen instabiles Verhalten. So fährt sich der Agent z.B. in 2 Versuchen in Abbildung Abbildung 6.14(a) fest, entspreched wird keine gute Bewertung erzielt.

Als nächstes wurde untersucht, ob die Länge der Umgewöhnungsphase des Agenten ein Maximum besitzt. Dafür wurde der Zeitbereich etwas ausgedehnt (Abbildung 6.15). Die maximal erreichbare Länge der Umgewöhnungsphase beträgt etwa 7 Episoden, diese wird nach 14 Episoden ohne Veränderungen in der Umgebung erreicht. Nach diesen Episoden scheint sich das Wissen des Agenten nicht bzw. nur noch minimal zu verändern.

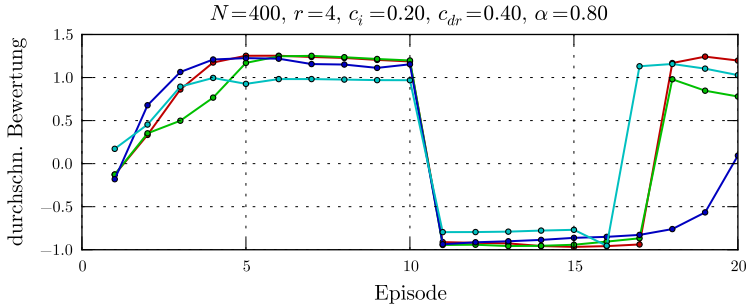
Schließlich wurde noch der Einfluss der Netzdynamik untersucht. Dafür wurden Simulationen mit verschiedenen Werten für α durchgeführt (Abbildung 6.16). Es ist kein Einfluss feststellbar. Dieses Ergebnis war, in Anbetracht der geringen Ausprägung der Netzdynamik, zu erwarten. Interessant wäre jedoch das Verhalten des Agenten mit einer deutlich stärker ausgeprägten Netzdynamik. Dies lässt sich, ohne tiefgreifende Veränderungen, jedoch nur durch Verletzung der Echo State Eigenschaft durch $\alpha > 1$ erzielen, und wurde nicht betrachtet. Die entstehenden Netzoszillationen würden einen gravierenden Einfluss darstellen.



(a) Die Umgebung wurde zu Begin der 5. Episode verändert.



(b) Die Umgebung wurde zu Begin der 9. Episode verändert.



(c) Die Umgebung wurde zu Begin der 11. Episode verändert.

Abbildung 6.14: Anpassung des Agenten an neue Umgebungen. Belohnung und Bestrafung wurden zu unterschiedlichen Zeitpunkten vertauscht. Je später dies passiert, desto länger braucht der Agent, um sich anzupassen.

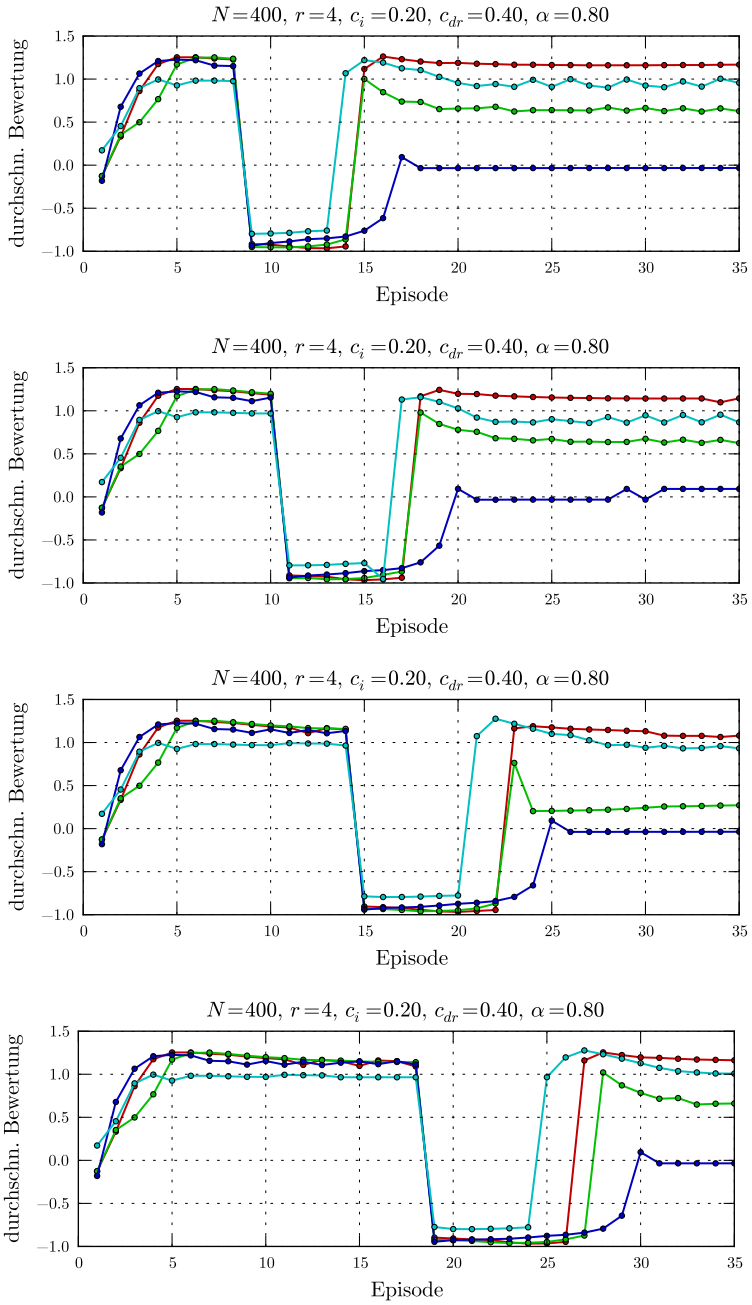


Abbildung 6.15: Die Umgebung wurde immer später verändert. Ab einer Veränderung zu Begin der 15. Episode hat dies praktisch keinen Einfluss auf die Anpassungsdauer des Agenten.

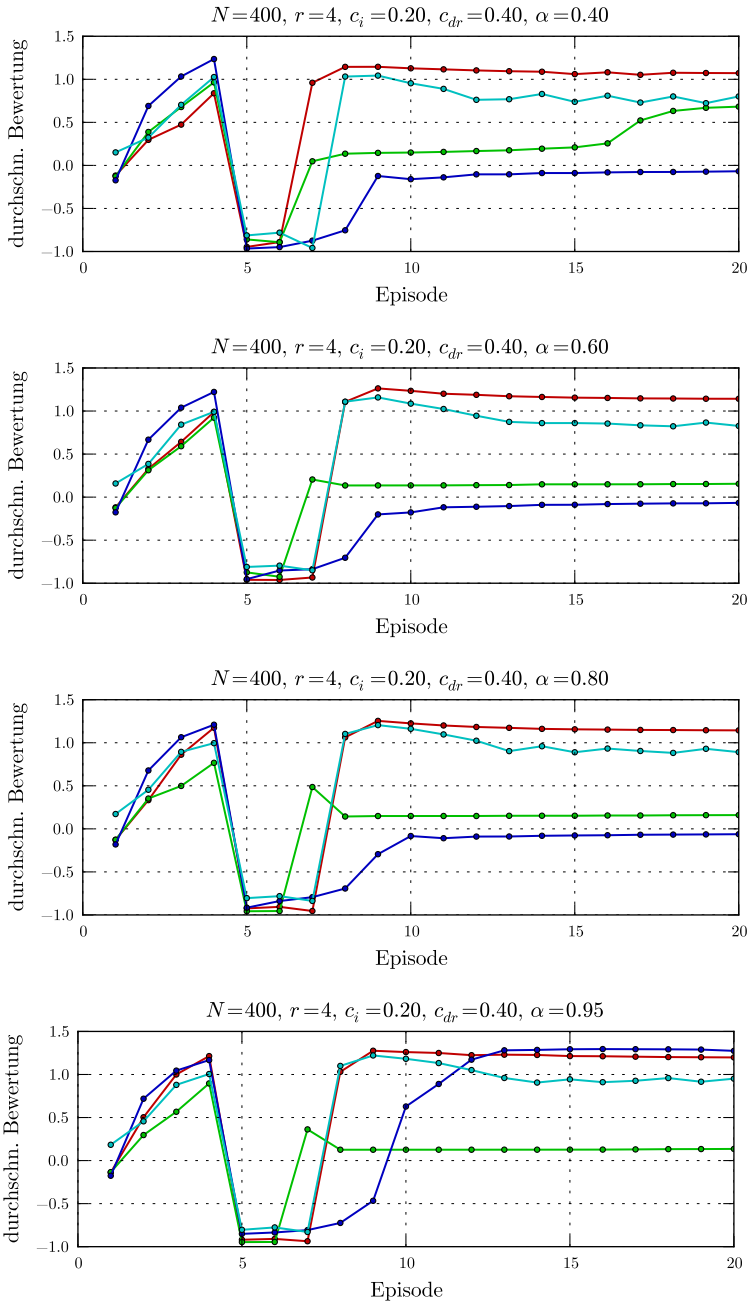


Abbildung 6.16: Die Anpassungsdauer wird vom spektralen Radius des ESNs nicht beeinflusst.

6.4 Stabilität

Unter Instabilität in Zusammenhang mit einem Agenten wird verstanden, dass sich das Verhalten des Agenten auf eine ungewollte Weise wiederholt, wodurch dieser seiner eigentlichen Funktion nicht mehr gerecht werden kann. Stabilität hingegen ist das Ausbleiben von Instabilität.

In den durchgeführten Versuchen fiel der Agent häufiger durch instabiles Verhalten auf. Dies soll nun eingehender betrachtet werden.

In besonders extremen Situationen sind die Instabilitäten gut als solche erkennbar. Es ist jedoch sehr schwierig einzuschätzen, in wie weit sich diese auf Versuche auswirken, in denen der Agent verhältnismäßig erfolgreich ist. So ist oft zu beobachten, dass der Agent in der Nähe der Belohnung anfängt Kreise zu fahren. Obwohl es das optimale Verhalten ist um die Belohnung zu kreisen, lässt sich nicht einschätzen, in wie weit die Entfernung dieser Kreise von der Belohnung durch Instabilitäten beeinflusst wird.

Die größte Quelle für instabiles Verhalten stellt jedoch die Versuchsumgebung selbst dar, genauer gesagt die Begrenzung hinter dem Agenten. Fährt sich dieser einmal in dem Verhalten fest, die Umgebung auf kürzestem Wege zu verlassen, bleibt dieses meist über viele Episoden hinweg bestehen. In den nun sehr kurzen Episoden wird kaum neues Wissen gesammelt, dadurch ist es dem Agenten auch nur kaum möglich, aus seinem Verhalten zu lernen.

Obwohl es an sich ein leichtes wäre, die Umgebung etwas zu vergrößern, ist dies nicht notwendigerweise eine Lösung des Problems. So haben Versuche gezeigt, dass sich der Agent auch in solchen Fällen in dem Verhalten festfahren kann, die Umgebung auf kürzestem Wege zu verlassen.

Das Problem ist an dieser Stelle eher das Fehlen einer negativen Bewertung für dieses Verhalten. Oder anders ausgedrückt, fehlt dem Agenten die Motivation, seine Umgebung zu erforschen. Das angesprochene Verhalten äußert sich lediglich dadurch negativ, dass dem Agenten das Wissen fehlt, besser zu handeln. Dies ist dem Agenten selbst jedoch nicht bewusst.

Kapitel 7

Zusammenfassung

Im Rahmen dieser Arbeit wurde die Umsetzung von Actor-Critic Design mit einer auf Echo State Networks basierenden Kritik zur Steuerung eines autonomen mobilen Roboters untersucht.

Die Aufgabe des Agenten bestand darin, eine Belohnung in seiner Umgebung zu finden, Bestrafung zu vermeiden und sich an eine Veränderung in dieser Umgebung anzupassen.

Die Betrachtung der Parameter des Echo State Networks in Kapitel 5.1.1 lieferte wichtige Ansatzpunkte für die erfolgreiche Realisierung des Agenten. Leider scheint das Ergebnis dieser Betrachtung jedoch stark vom verwendeten Online-Lernalgorithmus geprägt zu sein. Die „Extraktion“ des vom Agenten gesammelten Wissens belegt jedoch, dass das Echo State Network in der vorgesehenen Art und Weise funktioniert.

In Kapitel 5.3 wurde schließlich die Funktionsfähigkeit unter realen Bedingungen untersucht und bestätigt.

Im Anschluss an die Durchführung der Versuche wurde in Kapitel 6 versucht, die internen Abläufe im Agenten genauer zu analysieren.

Die Betrachtung der Linearität der Neuronen des Echo State Networks in Kapitel 6.1.1 liefert eine neue Sichtweise auf den Einfluss der verschiedenen Parameter des Echo State Networks. Auf Grund der Ergebnisse, die eine sehr geringe Ausprägung der Dynamik im Reservoir nahelegen, folgte schließlich eine Analyse der Netzdynamik in Kapitel 6.1.2, die zu interessanten Resultaten geführt hat. Basierend auf diesen Ergebnissen wurden in Kapitel 6.1.3 schließlich einige Experimente durchgeführt, die durch einige unvorhergesehene Effekte zu einer deutlichen Verbesserung des Agenten geführt haben.

Als nächstes wurde der Lernalgorithmus eingehender betrachtet. Ausgehend von den Veränderungen der Ausgabegewichte in Kapitel 6.2.1 wurde ein inhärentes Problem des Online-Trainings in Kapitel 6.2.2 identifiziert. Aufgrund dieser Ergebnisse wurde schließlich ein Vergleich mit einem Offline-Lernalgorithmus in Kapitel 6.2.3 durchgeführt, durch den die Einschränkungen durch Online Training offensichtlich wurden.

Ebenso wurde die Anpassungsfähigkeit des Agenten in Kapitel 6.3 genauer untersucht.

Ein weiteres Problem bestand in gelegentlich auftretender Instabilität des Agenten. Gemeint ist damit, dass sich der Agent in einem bestimmten

Verhalten festfährt. In diesen Situationen wird vom Agenten kein neues Wissen mehr gesammelt, wodurch sich folglich auch das Verhalten des Agenten nicht mehr verändert. Dieser Effekt hat leider einen nicht zu unterschätzenden Einfluss auf die Ergebnisse, wie in Kapitel 6.4 dargestellt.

Um die Versuche und insbesondere die Analyse durchführen zu können wurde eine Softwareumgebung zur Simulation des E-Puck Roboters geschaffen, die es erlaubt durch sehr umfangreiche Simulationen einzelne Aspekte des Agenten genauer zu untersuchen. Die erstellten Simulationen ließen sich in einem nächsten Schritt durch minimale Änderungen auf den realen E-Puck übertragen, und führten so zu den praktischen Ergebnissen.

In der Zusammenfassung hat der Agent die gestellte Aufgabe, trotz der angesprochenen Probleme, erfolgreich gelößt. Er war in der Lage, in einer ihm unbekannten Umgebung die Belohnung zu finden, und sich an eine Veränderung in dieser Umgebung anzupassen. Basierend auf diesen Ergebnissen wurde ein Paper bei der International Conference on Agents and Artifical Intelligence 2012 (ICAART) eingereicht [Oubbati et al., 2012]. Dieses wurde akzeptiert.

7.1 Ausblick

Basierend auf den Ergebnissen dieser Arbeit sind viele Ansatzpunkte für zukünftige Forschung denkbar.

Der Online-Lernalgorithmus wurde als einer der begrenzenden Faktoren identifiziert. Die Ergebnisse zu Offline-Training zeigen, dass mit den verwendeten Echo State Networks bessere Resultate erreichbar sind. Daher erscheint es vielversprechend, mögliche Verbesserungen zu erforschen. Dies könnte sich auch in Hinblick auf die Stabilität als interessant erweisen. Die durchgeführten Experimente mit für Echo State Networks eigentlich unzulässigen Werten für α zeigen, dass auch in dieser Hinsicht Verbesserungen denkbar sind. So existieren bereits einige Ansätze, das Reservoir vorzukonditionieren. Denkbar wäre z.B. die Umsetzung von Intrinsic Plasticity [Schrauwen et al., 2008].

Der in [Xue et al., 2007] vorgeschlagene Ansatz der Decoupled Echo State Networks with Lateral Inhibition scheint ebenso geeignet, die Leistungsfähigkeit des Agenten zu verbessern. So ließe sich z.B. die Linearität der Neuronen gezielter beeinflussen, ebenso wäre es möglich ohne zusätzlichen Rechenaufwand deutlich größere Netze zu simulieren.

Eine weitere Möglichkeit den Rechenaufwand zu reduzieren, könnten die in [Deng and Zhang, 2007] vorgestellten Scale-free highly-clustered Echo State Networks darstellen. Anstatt mit einem festgelegten Reservoir zu starten wird dieses iterativ entwickelt. Dadurch entwickeln sich lokale Cluster, die individuell spezialisieren. Dies führt letztlich zu einem effizienteren Reservoir.

Die Versuche ließen sich realistischer gestalten, indem auf die Unterteilung in einzelne Episoden verzichtet wird. Die Probleme bezüglich der Stabilität verhinderten ein solches Vorgehen leider. Diese ließen sich möglicherweise umgehen, indem die Erforschung der Umgebung als ein festes Ziel des Agenten festgelegt wird. Darüber hinaus könnten auch verschiedene Ansätze des Actor-Critic Design umgesetzt werden.

Die Versuchsumgebung wurde in dieser Arbeit bewusst einfach gewählt. Es wäre in zukünftigen Arbeiten jedoch denkbar, diese deutlich aufwendiger zu gestalten. So könnten mehrere Belohnungen existieren oder sich die Umgebung langsam aber kontinuierlich verändern. Ebenso wäre es denkbar, statt den virtuellen Landmarken von Sensoren erfasste, echte Objekte als Bezugspunkte zu benutzen. Auf diese Weise ließe sich möglicherweise die fehleranfällige Odometrie durch Schrittzählen umgehen.

Bei einer Betrachtung der in dieser Arbeit erzielten Ergebnisse, und den möglichen Verbesserungen, wird deutlich, dass die Kombination von Actor-Critic Design mit Echo State Networks eine vielversprechende Grundlage zur Realisierung autonomer mobiler Roboter bietet.

Literaturverzeichnis

[Berridge and Kringelbach, 2008] Berridge, K. C. and Kringelbach, M. L. (2008). Affective neuroscience of pleasure: reward in humans and animals. *Psychopharmacology*, 199(3):457–480.

[Deng and Zhang, 2007] Deng, Z. and Zhang, Y. (2007). Collective behavior of a small-world recurrent neural system with scale-free distribution. *IEEE Transactions on Neural Networks*, 18(5):1364–1375.

[Fahlman, 1988] Fahlman, S. E. (1988). An Empirical Study of Learning Speed in Back-Propagation Networks. Technical Report CMU-CS-88-162, Carnegie-Mellon Univ.

[Jaeger, 2002] Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach. Technical Report 159, AIS Fraunhofer, St. Augustin, Germany.

[Jaeger et al., 2007] Jaeger, H., Lukoševičius, M., Popovici, D., and Siewert, U. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352.

[Koprinkova et al., 2010] Koprinkova, H. P., Oubbati, M., and Palm, G. (2010). Adaptive critic design with echo state network. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1010–1015.

[Montague et al., 2004] Montague, R. P., Hyman, S. E., and Cohen, J. D. (2004). Computational roles for dopamine in behavioural control. *Nature*, 431:760–767.

[Oubbati et al., 2011] Oubbati, M., Kächele, M., Koprinkova-Hristova, P., and Palm, G. (2011). Anticipating rewards in continuous time and space with echo state networks and actor-critic design.

[Oubbati et al., 2012] Oubbati, M., Uhlemann, J., and Palm, G. (2012). Learning and adaptation of a mobile robot in a changing environment based on adaptive critic ESN. In *4th International Conference on Agents and Artificial Intelligence (accepted)*.

[Prokhorov, 2005] Prokhorov, D. (2005). Echo state networks: Appeal and challenges. In *Proc. of International Joint Conference on Neural Networks*, page 1463–1466.

[Prokhorov and Wunsch, 1997] Prokhorov, D. and Wunsch, D. (1997). Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8:997–1007.

[Riedmiller and Braun, 1993] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the rprop algorithm. *IEEE International Conference on Neural Networks*, 1993(3):586–591.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. *Mit Press Computational Models Of Cognition And Perception Series*, pages 318–362.

[Salas et al., 2010] Salas, R., Baldwin, P., De Biasi, M., and Montague, R. P. (2010). Bold responses to negative reward prediction errors in human habenula. *Frontiers in Human Neuroscience*, 4.

[Schrauwen et al., 2008] Schrauwen, B., Wardermann, M., Verstraeten, D., Steil, J. J., and Stroobandt, D. (2008). Improving reservoirs using intrinsic plasticity. *Neurocomput.*, 71:1159–1171.

[Sutton and Barto, 1998] Sutton, R. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. The MIT Press, Cambridge, MA, London, England.

[Tollenaere, 1990] Tollenaere, T. (1990). Supersab: Fast adaptive back propagation with good scaling properties. *Neural Networks*, pages 561–573.

[Werbos, 1990] Werbos, P. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[Williams and Zipser, 1989] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

[Xue et al., 2007] Xue, Y., Yang, L., and Haykin, S. (2007). Decoupled echo state networks with lateral inhibition. *Neural Networks*, 20(3):365–376.

Abkürzungsverzeichnis

ESN Echo State Network
RNN Recurrent Neural Network
ACD Actor-Critic Design
FFN Feed Forward Network
RL Reinforcement Learning
RC Reservoir Computing
u_n Neuronaktivierung
y_n Neuronausgabe oder Netzausgabevektor
u_r Aktivierung der Reservoir-Neuronen eines ESN
y_r Ausgabe der Reservoir-Neuronen eines ESN
y Ausgabe eines ESN
$\mathbf{W_i}$ Gewichtsmatrix der Eingabewerte auf das Reservoir eines RNN
$\mathbf{W_r}$ Gewichtsmatrix der Reservoirneuronen auf sich selbst eines RNN
$\mathbf{W_o}$ Gewichtsmatrix der Reservoirneuronen auf die Ausgabewerte eines RNN
BPTT Back Propagation Through Time
RTRL Real Time Recurrent Learning
EKF Extended Kalman Filtering
HDP Heuristic Dynamic Programming
DHP Direct Heuristic Programming
GDHP Globalized Dual Heuristic Programming
ADHDP Action Dependant HDP
ADDHP Action Dependant DHP
ADGDHP Action Dependant GDHP