

Abschlussbericht Karl-Steinbuch Stipendium Maschinelles Lernen für 2-Personen-Brettspiele

M. Comp. Sc. Stefan Faußer

Institut für Neuroinformatik, Universität Ulm, 89069 Ulm

18. Dezember 2008

Zusammenfassung

Im Rahmen dieses Projektes, finanziert durch ein Karl-Steinbuch Stipendium, wurden Methoden aus dem Bereich der Optimierung¹ und der Neuroinformatik am Brettspiel Dame angewandt, evaluiert und kombiniert. Dabei wurde ein Computeragent entworfen der keine festen Regeln einsetzt, sondern durch das Prinzip von Belohnung und Bestrafung lernt die Gesamtbelohnung zu maximieren. Der enorme Speicherbedarf der Methoden aus dem Bereich der Optimierung im Damespiel konnte durch die Approximation der Evaluierungsfunktionen mit einem Mehrschicht-Perzeptron (MLP) zufrieden stellend gelöst werden. Der hinsichtlich der Spielperformance beste Computeragent gewinnt gegen menschliche durchschnittlich gute bis sehr gute Dame-Spieler.

1 Einführung

Bereits seit Jahrzehnten implementieren Anwendungsentwickler Brettspiele auf Computer. Ob Backgammon, Vier Gewinnt, Schach, Mühle oder Dame: Computerprogramme versuchen stets gegen den menschlichen Gegenspieler zu gewinnen. Die Vorgehensweise, wie ein Computerprogramm ein Brettspiel spielt, ist jedoch häufig die selbe: Es wird in einer aktuellen Spielsituation ein Spielbaum mit einer bestimmten, beschränkten Tiefe berechnet und aufgrund des Spielbaumes die beste Zugmöglichkeit gewählt. Dies bedeutet, dass das Computerprogramm aufgrund der Tiefe des Suchbaumes nur eine bestimmte Anzahl von Spielrunden "voraus"denken kann, da diese Art der Auswahl des besten Spielzuges sehr viel Rechenzeit und Speicherkapazität benötigt.

Im Gegensatz dazu stehen Methoden aus dem Bereich des maschinellen Lernens, die eine Spielstrategie nicht vor jedem Zug berechnen, sondern diese bei jedem Zug iterativ durch das allgemeine Prinzip von Belohnung und Bestrafung verbessern und somit erlernen. Methoden aus dem Bereich der Optimierung (reinforcement learning methods)[1] wurden bereits in den Brettspielen Backgammon [5], Othello [6], Vier Gewinnt [7], Dame [8, 9] Schach [10] und sogar im komplexen Spiel Go [11, 12] angewandt. Die am meisten genutzte Methode ist das *Temporal Difference* (TD) lernen, gefolgt von den Monte Carlo Methoden. Bei den Monte Carlo Methoden wird pro Spielepisode ein Return-Wert (Belohnungswert) den in der Episode enthaltenen Aktionen bzw. Zuständen hinzugefügt und für die Berechnung der Aktionswerte bzw. Zustandswerte mit allen weiteren zugeordneten Werten gemittelt. In den

¹engl. reinforcement learning

TD Methoden hingegen kann der Lernvorgang durch den Einfluss von Folge-Aktionen bzw. Folge-Zuständen in einer Episode evtl. beschleunigt werden, wobei der Grad des Einflusses in der Methode $TD(\lambda)$ durch den Parameter λ gewählt werden kann. Weitere Ergänzungen an $TD(\lambda)$ wurden 1998 von Baxter et al. durchgeführt, wobei $TDLeaf(\lambda)$ und $TD - directed(\lambda)$ durch Kombination mit dem Suchalgorithmus *Minimax*[4] entstand. Grundlegender Unterschied zwischen den beiden ist das in $TDLeaf(\lambda)$ die Zustände den Knoten in einem Spielbaum entsprechen und das diese iterativ neu berechnet werden während in $TD - directed(\lambda)$ der Minimax Algorithmus für die direkte Berechnung der Belohnung genutzt wird. $TDLeaf(\lambda)$ wurde zuerst in Schach eingesetzt[10].

Da die Spielbaumkomplexität der oben genannten Brettspiele für den alleinigen Einsatz von MC und TD Methoden zu groß ist und bei Speicherung der Evaluierungsfunktionen, d.h. der Aktionswertefunktionen oder der Zustandswertefunktionen, mehr Speicherplatz benötigt werden würde wie aktuell zur Verfügung steht, wurden in allen obigen Literaturangaben die Methoden approximiert. Üblich ist eine neuronale Approximation, wobei ein Mehrschicht-Perzeptron (MLP) gefolgt von einem Radialen Basisfunktionen Netzwerk (RBF) am häufigsten eingesetzt wird.

In diesem Projekt wurden Methoden aus dem Bereich der Optimierung, speziell Monte Carlo Methoden und Temporal Difference Methoden, im Damespiel zusammen mit MLP und RBF Netzen angewandt, kombiniert und evaluiert. Die Entscheidung ist auf das Damespiel gefallen, da dieses eine durchschnittlich hohe Spielbaumkomplexität und einfache Regeln hat. Ein weiterer Grund war, das das Damespiel als teilweise gelöst[13] und unter Verwendung fester Regeln als unschlagbar gilt, diese Spielstärke jedoch noch nicht mit Methoden aus dem Bereich des maschinellen Lernens bei der Dame erreicht wurde.

2 Methoden

2.1 Übersicht über den Trainingsalgorithmus

2.1.1 Interaktionszyklus

Der Computeragent initialisiert den ersten Spielzustand s_0 oder erhält vom gegnerischen Spielagenten den Spielzustand s_0 , der eine Kodierung des Spielfeldes in Dame darstellt. Anschließend wird das *epsilon-greedy* Verfahren durchgeführt. Dazu zieht der Computeragent eine Zufallszahl im Intervall $[0, 1]$ und vergleicht diese mit dem Parameter ϵ . Ist die Zufallszahl $> \epsilon$, so wird der folgende Zustand exploriert, ansonsten exploitiert. Exploitiere bedeutet, dass der Computeragent gemäß seiner Strategie π eine Aktion ausführt, die in einen der Zustände s' aus der Menge aller möglichen Folgezustände $S_{successor}$ resultiert:

$$\pi(s) = \underset{s'}{\operatorname{argmax}}(V^\pi(s') | s' \in S_{successor}) \quad (1)$$

Der Folgezustand s' bzw. in diesem Beispiel s'_0 ist somit der Zustand mit dem maximalen Zustandswert $V(s')$. Der Computeragent führt die Aktion aus, die in Zustand s'_0 resultiert und gibt diesen Zustand an den gegnerischen Agenten weiter. Der gegnerische Agent kann als eine Blackbox mit der Eingabe des aktuellen Zustandes s_t und der Ausgabe des Folgezustandes s_{t+1} angesehen werden, wobei im Terminalzustand $t+1 = TC$, d.h. dem Zustand in dem einer

der beiden Spielteilnehmer entweder gewonnen, verloren oder unentschieden erreicht hat, zusätzlich ein Belohnungswert R_{TC} zurück gegeben wird:

$$R_{TC} = \begin{cases} 1, & \text{falls der Agent gewonnen hat} \\ 0, & \text{falls der Agent verloren hat} \end{cases} \quad (2)$$

Der Computeragent sammelt alle vom gegnerischen Agenten erhaltenen Zustände, d.h. $\{s_0, s_1, \dots, s_{TC}\}$, und nutzt diese zusammen mit dem Belohnungswert R_{TC} , um die Abschätzungen der Zustandswertefunktion $V(s)$ zu verbessern.

2.1.2 Monte Carlo lernen

Wurde eine komplette Episode gespielt, d.h. der Terminalzustand s_{TC} erreicht und ein Belohnungswert R_{TC} erhalten, so können die Zustandswerte $V(s_0), V(s_1), \dots, V(s_{TC})$ aktualisiert werden. Dazu gibt es für jeden Zustand s eine Menge $Return(s)$, die initial für jedes s leer ist. Bei dem Erhalt eines Belohnungswertes R_{TC} wird nun dieser jeder Menge $Return(s)$ hinzugefügt, dessen Zustand s in der vom Computeragenten gesammelten Episode vorkam. Die Neuberechnung der Zustandswertefunktionen $V(s)$ folgt nun nach den *every-visit* Monte Carlo Methoden und lautet:

$$V^\pi(s) = \text{average}[Return(s)], \text{ für jedes } s \text{ in Episode} \quad (3)$$

Gibt es wie in unserem Falle nur einen Belohnungswert pro Episode, so kann die Notwendigkeit der Mengen $Return(s)$ aufgehoben werden:

$$V^\pi(s) = [n(s)V^\pi(s) + R_{TC}] \frac{1}{n(s) + 1}, \text{ für jedes } s \text{ in Episode} \quad (4)$$

Hierbei stellt $n(s)$ die Anzahl der für den Zustand s erhaltenen Belohnungswerten dar und $V^\pi(s)$ ist somit der Durchschnittswert von $n(s) + 1$ in Terminalzuständen erhaltenen Belohnungswerten, beginnend von s unter Strategie π . Da die Strategie π von den Zustandswerten $V^\pi(s)$ abhängt, bewirkt eine Verbesserung von $V^\pi(s)$ eine in Zukunft bessere Wahl eines Folgezustandes s' unter Verwendung von π .

2.1.3 Temporal Difference lernen

Im Vergleich zum Monte Carlo lernen werden hier die Zustandswerte $V(s)$ nach jeder Aktion des Computeragenten, resultierend in einen Zustand s , neu abgeschätzt:

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)) \quad (5)$$

Der (vorherige) Zustandswert $V(s_t)$ zur Zeit $t + 1$ ist somit eine durch Parameter γ abgeschwächte Abschätzung des Folgezustandswertes $V(s_{t+1})$ oder der Belohnungswert r_{t+1} , da zur Zeit $t + 1 = TC$ der Wert $r_{t+1} = R_{TC}$ und zu jeder anderen Zeit $r_{t+1} = 0$. Der Parameter $\alpha \in [0, 1]$ bestimmt die Lernrate.

2.2 Neuronale Approximation der Zustandswertefunktion $V(s)$

Um die Strategie π anzuwenden und zu verbessern, müssen die Zustandswerte $V(s)$ gespeichert werden. Würden dazu Tabellen genutzt werden, so würde dies beim Damespiel bei einem 8×8 Spielfeld $5^{32} * 4$ [Byte pro Zustand] \approx

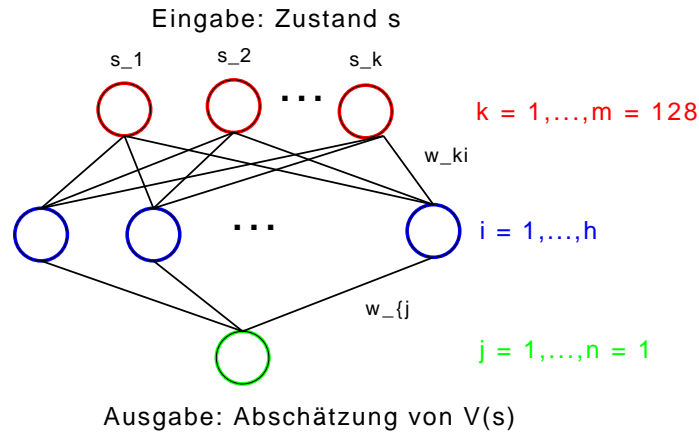


Abbildung 1: Mehrschicht-Perzeptron Schemata

9,31 * 10²² [TByte] Speicher beanspruchen (obere Grenze). Cybenko's Theorem [2] folgend, das besagt, dass ein Mehrschicht-Perzeptron (MLP) mit mindestens einer versteckten Schicht und einer sigmoiden Aktivierungsfunktion jede stetige Funktion mit einem beliebigen Grad an Genauigkeit approximieren kann, wird primär ein MLP genutzt um die Zustandswertefunktion $V(s)$ zu approximieren. Versuche mit einem Radialen-Basisfunktionen Netzwerk wurden ebenso durchgeführt, wie in einem weiteren Kapitel und in den Resultaten ersichtlich.

2.2.1 Aufbau und Implementation des Mehrschicht Perzeptrons (MLP)

Das MLP hat die Aufgabe, den Zustandswert $V(s)$ eines eingegebenen Spielzustandes s abzuschätzen. Unter Verwendung der offiziellen Spielregeln für English draughts hat das Damespiel $8 \times 8 = 64$ Spielfelder, wovon nur die Hälfte, d.h. 32, genutzt werden und es pro Spielfeld 5 Belegungsmöglichkeiten (weiß, schwarz, leer, weiße Dame, schwarze Dame) gibt. Um eine ausgeglichene Verteilung zu erreichen, wurde die binäre Bit-Sequenz 0001 für schwarz, 0010 für schwarze Dame, 0100 für weiß, 1000 für weiße Dame und 0000 für ein leeres Spielfeld gewählt. Somit ergibt sich eine Anzahl von 128 Eingabeneuronen und ein Ausgabeneuron für die Bewertung im Intervall $[0, 1]$. Die Anzahl der Neuronen in der versteckten Schicht wurde zwischen 40 und 250 gewählt (siehe Resultate). In Abbildung 1 ist die allgemeine Schemata zu sehen.

Als Transferfunktion in der versteckten Schicht und der Ausgangsschicht wurde eine nicht-lineare und sigmoide logistische Funktion gewählt, da intuitiv gesehen der Ausgabewert des neuronalen Netzes eine Gewinnwahrscheinlichkeit für den angegebenen Zustand darstellt:

$$f(x) = \frac{1}{1 + \exp(-x)} \quad (6)$$

$$f'(x) = f(x)(1 - f(x)) \quad (7)$$

Wie üblich ist die Transferfunktion für die Eingabeschicht die Identität. Abhängig vom gewählten Lernverfahren ist das Trainingssignal T_s für jeden Zustand s anders. Bei *Monte Carlo* lernen lautet die Berechnung von T_s^{MC} für jeden besuchten Zustand s folgendermaßen:

$$T_s^{MC} = R_{TC} \gamma^{TC-t(s)} \quad (8)$$

In dieser Gleichung ist T_s gleich dem diskontierten Belohnungswert R_{TC} , wobei der Diskontierungsfaktor sich aus TC , der die Anzahl der besuchten Zustände in dieser Episode darstellt und $t(s)$ der die Index-Nummer des Zustandes s zwischen 1 und TC zurück gibt, zusammensetzt. Dadurch gibt es keine Diskontierung im letzten Zustand s_{TC} und ansteigende Diskontierung in Richtung des ersten Zustandes s_0 in der Episode. Die Stärke der Diskontierung kann durch dem Parameter $0 < \gamma \leq 1$ bestimmt werden. Der Fehler zwischen den Trainingssignalen und den realen Werten einer ganzen Episode lautet:

$$E^{MC} = \sum_s ||T_s^{MC} - V(s)||^2 \quad (9)$$

Hingegen bei *Temporal Difference* lernen lautet die Berechnung von T_s^{TD} zur Zeit $t + 1$ für Zustand s_t folgendermaßen:

$$T_s^{TD} = \begin{cases} R_{TC}, & \text{falls } t + 1 = TC \\ \gamma V(s_{t+1}), & \text{sonst} \end{cases} \quad (10)$$

Der Fehler zwischen einem Trainingssignal und einem realen Wert in einer Episode lautet:

$$E_s^{TD} = ||T_s^{TD} - V(s)||^2 \quad (11)$$

Das MLP lernt nun durch Änderungen der Gewichtswerte, den angegebenen Fehlerterm und demnach die Abschätzungen der Zustandswertefunktionen zu verbessern. Bei MC lernen finden diese Gewichtsänderungen Offline, d.h. einmal pro Episode, und bei TD lernen Online, d.h. pro besuchten Zustand in einer Episode, statt. Für weitergehende Literatur wie die Gewichtswerte durch Ableitung des Fehlerterms verändert werden sollen, wird [3] empfohlen. Die Lernrate α wie sie noch in der Gleichung (5) vorkam, ist nicht mehr nötig, da das MLP eine eigene Lernrate η äquivalent dazu nutzt. Weiterhin kann in [7] nachgelesen werden, weshalb Fehlerterm (9) eine Approximierung der Monte Carlo Evaluierungsfunktion bewirkt.

2.2.2 Aufbau und Implementation des Radialen Basis-Funktionen Netzwerkes (RBF)

Der Aufbau des RBF Netzes ist dem Aufbau des MLP Netzes sehr ähnlich. Als Transferfunktion für die versteckte Schicht wurde die Gauss-Funktion genutzt und die Transferfunktion der Ausgabeschicht ist wie üblich die gewichtete Summe der Ausgangswerte der versteckten Schicht. Die Zentren der Gauss-Glocken in der versteckten Schicht wurden äquidistant verteilt und die Breite der Gauss-Glocken wurde sinngemäß, d.h. so dass sich die Gauss-Glocken leicht überlappen, gewählt. Um die Komplexität des RBF-Netzwerkes zu verringern, wurde ein Parameter σ für die Breite aller Gauss-Glocken gewählt und nicht ein Parameter σ_i pro Eingabeneuron i . Sowohl die Gewichte in der Zwischenschicht wie auch die Gewichte der Ausgabeschicht wurden adaptiert. Eine vorherige Clustering der Eingabedaten zum Festlegen der Gewichte in der Zwischenschicht, wie bei RBF-Netzwerken üblich, war hier nicht möglich bzw. sinnvoll, da es beim Damespiel zuviele mögliche Eingabewerte gibt. Die Eingabecodierung und die Trainingssignale wurden wie bei einem MLP-Netzwerk festgelegt. Für die Adaptierung der Gewichtswerte wird auf [3] verwiesen.

2.3 Ensemble lernen

Während der Evaluierungsphase wurden mehrere Computeragenten, d.h. mehrere verschiedene neuronale Netze, produziert, die demnach unterschiedliche Wissensbasen hatten. In dieser Phase kam auch die Idee, die Wissensbasen miteinander zu einer neuen Entscheidung zu fusionieren. Für diesen intuitiven Ansatz wurden Entscheidungen mehrerer Computeragenten per *Voting* fusioniert. Dies bedeutet, dass der Folgezustand der von allen beteiligten Computeragenten am meisten ausgewählt wurde, genommen wurde. Bei einer unentschiedenen Wahl wurde der Folgezustand zufällig gewählt. Dieses Verfahren ist als *Bagging Ensemble* lernen oder auch *Bootstrapping* bekannt.

2.4 Episoden sammeln in Datenbanken

Der Computeragent wurde mit der Fähigkeit ausgestattet, bereits absolvierte Episoden in einer Datenbank zu sammeln. Diese Datenbank konnte in einem späteren Verlauf verwendet werden, um gänzlich neue Computeragenten Offline (derzeitig nur mit MC lernen, wobei auch ein TD lernen Offline wäre) zu trainieren.

3 Resultate

Um den intelligenten Computeragenten zu trainieren, wurde ein gegnerischer Agent mit selber Wissensbasis, d.h. dem selben neuronalen Netz, genutzt. Weiterhin hat der gegnerische Agent mit der selben Exploitationsrate ϵ agiert, d.h. mit der Rate $1 - \epsilon$ zufällige Aktionen ausgeführt, um sowohl wiederholende Episoden zu vermeiden wie auch eine ausgeglichene Anzahl von positiven und negativen Belohnungswerten zu ermöglichen. Nach dem Training wurde die Spielperformance des Computeragenten gegen einen weiteren gegnerischen Agenten, der 100% zufällige jedoch regelkonforme Aktionen ausführt, getestet. Die Anzahl der Gewinne des intelligenten Computeragenten wurde in einer Gewinn-Quote nach 200.000 Test-Episoden ermittelt:

$$\text{Gewinn-Quote} = \frac{\text{Anzahl der gewonnenen Spiele}}{\text{Anzahl der verlorenen Spiele}} \quad (12)$$

Die Parameter wurden folgendermaßen gesetzt: Die initialen Gewichtswerte für das MLP wurden für $h < 250$ zwischen -0.77 und $+77$ gesetzt, bei $h = 250$ zwischen -0.4 und $+0.4$. Der Diskontierungsfaktor γ ist 0.99 für alle Testergebnisse. Die Anzahl der Neuronen in der versteckten Schicht h wie auch die Lernrate η kann in der Tabelle 1 mit den Testergebnissen nachgelesen werden. Die ersten drei Agenten (MLP Nr. 1 bis 3) hatten eine konstante Exploitationsrate von $\epsilon = 0.7$ wobei bei dem letzten Agenten diese schrittweise erhöht, d.h. die Explorationsrate verringert wurde. Die Ergebnisse in der Tabelle beziehen sich auf ein MLP Netzwerk, dass mit MC lernen trainiert wurde. Die Ergebnisse für TD lernen mit einem MLP Netzwerk unterscheiden sich kaum, so dass für diese keine separate Tabelle erstellt werden musste. Der Agent mit einem RBF Netzwerk hingegen konnte zu keinen derartig guten Ergebnissen gebracht werden, bei $h = 120$ konnte er eine maximale Gewinn-Quote von rund 200 erreichen. Parameter-Optimierungen wie eine absteigende Lernrate oder ein ansteigendes ϵ die sich bei einem MLP Netzwerk bei den Experimenten bewährt haben, konnte das RBF-Netzwerk auch nicht verbessern.

Tabelle 1: Gewinn-Quote mehrerer intelligenter Computeragenten gegen einen Agenten der alle seine Aktionen zufällig und regel-konform ausführt

MLP Nr.	Gewinn-Quota	h	η	Episoden
1	87	40	0.05	500.000
	94			+500.000
	149			+500.000
	172			+3.500.000
2	124	120	0.05	500.000
	167			+500.000
	367			+4.000.000
3	367	120	0.1	2.000.000
	1024		0.05	+3.000.000
	1525		0.025	+4.000.000
4	587	250	0.5	2.000.000
	2082		0.25	+3.000.000
	11110		0.125	+4.000.000

Eine Fusionierung der Entscheidungen hatte bei mehreren voneinander separat trainierten Agenten mit nur wenigen Neuronen in der versteckten Schicht ($h = 40$) eine Verbesserung der Gesamtentscheidung zur Folge. Die Gewinn-Quote konnte von rund 170 auf 250 angehoben werden. Hingegen konnten Agenten die bereits bessere Gewinn-Quoten hatten ($h = 120$) mit dieser Methode nicht verbessert werden.

Neue Agenten, die mit den besuchten Spielepisoden eines bereits fortgeschrittenen Agenten aus der Datenbank trainiert wurden, konnten in der Lernzeit anfänglich verbessert werden. So war es bei $h = 40$ möglich, eine Gewinn-Quote von 180 nach bereits 3.000.000 Episoden statt 5.000.000 Episoden zu erreichen. Die Gewinn-Quote selbst konnte dadurch nicht wesentlich gesteigert werden.

3.1 Diskussion der Resultate

Konnte mit einem RBF-Netzwerk nur eine maximale Gewinn-Quote von rund 200 erreicht werden, so war die Gewinn-Quote bei einem MLP-Netzwerk deutlich über diesem Wert, ist jedoch wie in Tabelle 1 ersichtlich stark Parameter abhängig. Generell gesehen ist ein MLP mit mehr Neuronen in der versteckten Schicht h und mehr Trainingsepisoden spielstärker, wobei beide Werte limitiert werden sollten (Stichwort: Mehr lokale Minima). Die Spielperformance konnte weiter durch eine schrittweise fallende Lernrate η und durch eine schrittweise fallende Explorierungsrate $1 - \epsilon$ gesteigert werden. Der 4. Agent mit einer Gewinn-Quote von rund 10.000 ist in der Lage, einen durchschnittlich guten menschlichen Gegenspieler zu besiegen. Dies wurde in Experimenten mit drei Testpersonen und dem grafischen Interface des Computeragenten getestet. Es war zu erwarten, dass durch das *Bagging Ensemble* lernen nur die Spielperformance von mittelstarken Agenten angehoben werden konnte, nicht zu erwarten war die schlechte Spielperformance von Agenten mit einem RBF-Netzwerk.

4 Konklusion und Aussicht

Im Laufe des durch das Karl-Steinbuch Stipendium geförderten Projektes konnte ein Computeragent geschaffen werden, der in etwa die Spielstärke eines menschlichen durchschnittlich guten Dame-Spielers aufweist. Die schlechte Spielperformance des RBF-Netzwerkes ist noch gänzlich ungeklärt, wobei ein Ansatzpunkt für die Fehlersuche die Codierung der Spielzustände ist und eine Neuronen-abhängige Breite der Gauss-Glocke σ_i statt einem allgemeinen σ eine weitere Besserung bringen könnte. Es war zu erwarten, dass MC lernen und TD lernen in etwa gleich gute Resultate erbringen und dass die hauptsächlichen Performance-Steigerungen durch die Anpassung eines neuronalen Netzwerkes erbracht werden können. In folgenden wissenschaftlichen Arbeiten werden wir uns mit der Untersuchung der Eigenschaften der Evaluierungsfunktionen hinsichtlich ihrer Stetigkeit und ihrer stetigen Differenzierbarkeit widmen und dazu sowohl die Darstellung der Zustandsräume wie auch die Evaluierungsfunktionen selbst untersuchen. Wir erhoffen uns für ein bestimmtes Brettspiel wie z.B. Dame diese Eigenschaften der Evaluierungsfunktionen z.B. durch Änderungen an der Zustandscodierung verbessern zu können um damit besser von einem MLP oder einem RBF Netzwerk approximiert werden zu können. Dadurch sollte sich die Spielperformance eines Brettspieles, dass Markovsche Entscheidungsprozesse (MDPs) beinhaltet, deutlich steigern.

A Danksagung

Ich möchte mich aufrichtig bedanken bei

- der MFG mbH für die Darreichung eines Karl-Steinbuch Stipendiums ohne dieses ich niemals die Zeit gefunden hätte an diesem Projekt zu arbeiten
- Herrn Friedhelm Schwenker, dem akademischen Oberrat am Institut für Neuroinformatik an der Universität in Ulm, der mit Rat und Tat für mich da war, wann immer ich ihn brauchte
- Herrn Günther Palm der durch dessen Zustimmung und Gutachten meine Bewerbung für ein Karl-Steinbuch Stipendium möglich gemacht hat
- meiner Freundin Anna für ihr Verständnis für die Zeit, die ich für diese Arbeit genutzt habe

Literatur

- [1] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, *MIT Press* (Cambridge, MA, 1998).
- [2] G.V. Cybenko, Approximation by Superpositions of a Sigmoidal function, *Mathematics of Control, Signals and Systems, vol. 2 pp. 303-314. electronic version* (1989).
- [3] Christopher M. Bishop, Neural Networks for Pattern Recognition, *Oxford University Press* (1995)
- [4] Stuart J. Russel and Peter Norvig, Artificial Intelligence: A Modern Approach, 2nd Edition, *Prentice Hall* (2002).

- [5] Gerald Tesauro, Temporal Difference Learning and TD-Gammon, *Communications of the ACM*, Vol. 38, No. 3 (1995).
- [6] Nees Jan van Eck, Michiel van Wezel, Reinforcement Learning and its Application to Othello, *Department of Computer Science, Faculty of Economics, Erasmus University, The Netherlands* (2004).
- [7] Stefan Faußer and Friedhelm Schwenker, Neural Approximation of Monte Carlo Policy Evaluation Deployed in Connect Four, *Artificial Neural Networks in Pattern Recognition, Proceedings of the Third IAPR Workshop, ANNPR 2008, vol. 5064, pages 90 to 100* (2008).
- [8] Jan Peter Patist and Marco Wiering, Learning to Play Draughts using Temporal Difference Learning with Neural Networks and Databases, *Proceedings of the Thirteenth Belgian-Dutch Conference on Machine Learning, edited by Ann Nowe, pp.* (2004).
- [9] Ing C. L. Dubel, L. Lefakis Bsc, Ing J. Brandsema, S. Szóstkiewicz Bsc, Reinforcement learning project: AI Checkers Player, *Utrecht University* (2006).
- [10] Jonathan Baxter, Andrew Tridgell, Lex Weaver, Knightcap: A chess program that learns by combining TD(λ) with game-tree search, *Proceedings of the 15th International Conference on Machine Learning* (1998).
- [11] R Ekker, ECD van der Werf, LRB Schomaker, Dedicated TD-learning for Stronger Gameplay: applications to Go, *Proceedings of Benelearn 2004 Annual Machine Learning Conference of Belgium and The Netherlands (Brussels, BE, January 8-9, eds. A. Nowé, T. Lennaerts, K. Steenhaut), pp. 46-52* (2004)
- [12] David Silver, Richard Sutton, and Martin Müller, Reinforcement Learning of Local Shape in the Game of Go, *IJCAI* (2007)
- [13] J. Schaeffer, Y. Bjornsson, N. Burch, A. Kishimoto, M. Muller, R. Lake, P. Lu and S. Sutphen, Solving Checkers, *Department of Computing Science, University of Alberta* (2005)