Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Anwendungen 2

Jan Busch

Hierarchical Reinforcement Learning Using Simulation Blocks - Related Work

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einleitung

1.1 Forschungsprojekt FAUST

Das FAUST-Projekt [FAUST (2011)] aus dem Department für Informatik der Hochschule für Angewandte Wissenschaften Hamburg entwickelt Technologien für **F**ahrerassitenz- und **Au**tonome **Sys**t**e**me auf verschiedenen selbstentwickelten Fahrzeugplattformen. Die Schwerpunkte der verschiedenen Projekte liegen hierbei auf:

- Sensorik, Telemetrie und Bildverarbeitung
- Echtzeitsystemen und Bussystemen
- Software- und Hardwarearchitekturen
- Algorithmik und Steuerung

In vielen aktuellen Fahrzeugen finden sich bereits Assistenzsysteme in Form von Einparkhilfen oder Abstandsreglern. Die Anzahl solcher Helfer wird in Zukunft noch weiter zunehmen, um den Fahrer optimal zu unterstützen und dessen Sicherheit zu steigern.

Verschiedene Fahrzeuge des FAUST-Projektes nehmen jährlich am Carolo-Cup [Carolo-Cup (2011)] der technischen Universität Braunschweig teil. Bei diesem Hochschulwettbewerb müssen die Fahrzeuge selbstständig einen Parcours mit Hindernissen bewältigen, automatisch einparken und weitere Probleme lösen. Ziel ist es, die verschiedenen Disziplinen möglichst schnell und fehlerfrei zu absolvieren. Die Jury setzt sich aus Experten aus Wissenschaft und Industrie zusammen und bewertet die Ergebnisse sowie die Konzepte hinter den Ergebnissen. Die beiden Teams der HAW arbeiten stetig an der Verbesserung ihrer Fahrzeuge und deren Software. [Busch (2011)]

1.2 Simulationsumgebung TORCS

TORCS steht für „**T**he **O**pen **R**acing **C**ar **S**imulator“ und ist eine Simulationsumgebung für Rennwagen. Als Multiplatform-Software steht sie für Linux (x86, AMD64 and PPC), FreeBSD, MacOSX und Windows zur Verfügung. Der Quellcode ist unter der *GPL*[1] lizensiert und wird häufig als Plattform für Forschungszwecke verwendet. Darüber hinaus bietet sie viele Möglichkeiten:

- Programmierung der künstlichen Intelligenz der Fahrzeuge
- Design eigener Fahrzeuge, d.h. Aussehen und Fahrverhalten

[1] General Public License, siehe http://www.gnu.org/licenses/licenses.html

- Erstellung eigener Strecken
- Bereitstellung von Informationen über die Umgebung der Fahrzeuge und über das einzelne Fahrzeug selbst
  - z.B. Fahrbahnhaftung, Geschwindigkeit, Beschleunigung und Lenkwinkel

[Busch (2011)]

1.3 Eingrenzung des Themengebietes

Reinforcement Learning ist in der Forschung seit seinem Aufkommen ein beliebtes Gebiet. Es beschäftigt sich mit der Frage, wie ein Agent lernt, sich in einer stochastischen Umgebung optimal zu verhalten. Dies ist ein sehr generelles Problem, was viele unterschiedliche Aufgaben beschreibt. [Russell und Norvig (2010)]

In der Forschung gibt es viele Anwendungsfälle für *Reinforcement Learning*, allerdings hat *Reinforcement Learning* mit Aufgaben aus der realen Welt verschiedene Probleme. Die Zustands-Aktions-Räume sind meist kontinuierlich und sehr groß. Hochdimensionale Zustands-Aktions-Räume lassen den Speicherverbrauch exponentiell anwachsen und auch die zeitliche Ausführungsdauer des Lernens bis zur Konvergenz steigt so stark an, dass es im Prinzip nutzlos ist. [Jong und Stone (2007)]

Um mit diesem Problem umgehen zu können, wird eine geeignete Approximation benötigt. Sie soll ermöglichen, den teils großen oder gar unendlichen Zustands-(Aktions-)Raum endlich abzubilden. So könnten gegebenenfalls viele Probleme im Zusammenhang mit *Reinforcement Learning* behoben werden. Deshalb gibt es viele Konzepte zur Approximation, von denen im folgenden Kapitel einige vorgestellt werden.

2 Vorstellung vergleichbarer Arbeiten

2.1 Least-Squares Reinforcement Learning

Methode der kleinsten Quadrate

Bei der Methode der kleinsten Quadrate[2] wird zu einer Menge von Datenpunkten eine Kurve gesucht, die möglichst nahe an den Datenpunkten liegt. Die Methode bestimmt die Parameter

[2] Engl.: Method of least squares

der Kurve so, dass die Summe der quadratischen Abweichung der Kurve von den Datenpunkten minimiert wird.

Im Kontext von *Reinforcement Learning* kann diese Methode genutzt werden, um mit großen Zustands- und Aktionsräumen umgehen zu können, d.h. den Zustands-Aktions-Raum zu approximieren.

Temporal Difference Learning

Temporal Difference Learning gehört zu der Gruppe der *Reinforcment Learning* Algorithmen, die gegenüber der dynamischen Programmierung kein Modell der Umgebung benötigen. Das Problem, dass die gemachte Erfahrung erst am Ende einer Episode genutzt werden kann, wie es bei den Monte Carlo Methoden der Fall ist, fällt bei TD-Learning ebenfalls weg. Im Folgenden sind die *Update*-Formeln des Monte Carlo Konzepts sowie des *Temporal Difference Learning*s zum Vergleich dargestellt:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \tag{1}$$

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} - \gamma V(s_{t+1}) - V(s_t)] \tag{2}$$

Hier zeigt sich der Unterschied des TD-Learnings deutlich. Bei TD-Learning (Formel 2) wird die Differenz zwischen der aktuell beobachteten Belohnung r_{t+1} und dem geschätzten Wert des erreichten Folgezustands $\gamma V(s_{t+1})$ gebildet. Beim Monte Carlo Konzept (Formel 1) hingegen wird die aufsummierte Belohnung R_t bis zum Ende der Episode ausgehend von Zeitpunkt t als Basis genommen.

Least-Squares Temporal Difference Learning

Bradtke und Barto nutzten die Methode der kleinsten Quadrate als erstes Forschungsteam im Kontext des *Temporal Difference Learnings*. Ihr Artikel *„Linear Least-Squares Algorithms for Temporal Difference Learning“* [Bradtke und Barto (1996)] zeigt, dass sich durch den Einsatz der *Least-Squares* Methode ein deutlich größerer Nutzen aus der Trainingserfahrung eines Agenten extrahieren lässt als es beim herkömmlichen *Temporal Difference Learning* der Fall ist. Dieser Vorteil ist jedoch mit einem höheren Rechenaufwand und einem höheren Speicherverbrauch pro Zeitschritt verbunden. Statistisch bleibt ihr Algorithmus allerdings effizienter, da aus jeder Beobachtung mehr Informationen gezogen werden können und der Algorithmus damit schneller konvergiert.

Policy Iteration

Die *Policy Iteration* oder auch Strategie-Verbesserung wird zum Finden der optimalen Strategie π^* eingesetzt. Der Algorithmus kann wie folgt beschrieben werden:

1. Beliebige initiale Strategie π_t mit $t = 0$.
2. Bewertung der Strategie π_t (*Policy Evaluation*)
3. Verbessern der Strategie π_t durch Wahl der besten Aktion für jeden Zustand auf Basis der in der Policy Evaluation berechneten Werte. Daraus ensteht die verbesserte Strategie π_{t+1}.
4. GOTO 2 mit Strategie π_{t+1} bis Strategie stationär, also π^*.

Die Bewertung der Strategie (Schritt 2) wird mit Formel 3 berechnet. Mithilfe dieser Werte wird in Schritt 3 die Strategie-Verbesserung mit Formel 4 durchgeführt. [Russell und Norvig (2010)]

$$V(s) \leftarrow \sum_{s'} P_{ss'}^{\pi(s)} [R_{ss'}^{\pi(s)} + \gamma V_k(s')] \tag{3}$$

$$\pi(s) \leftarrow argmax_a + \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma V(s')] \tag{4}$$

Least-Squares Policy Iteration

Der LSPI[3] Algorithmus von Lagoudakis und Parr [Lagoudakis und Parr (2003)] ist inspiriert durch den LSTDL Algorithmus von Bradtke und Barto. LSPI ist ein *off-policy* Algorithmus. Ein solcher Algorithmus beinhaltet, dass Trainingsdaten per Zufall gesammelt werden können und nicht das Ergebnis der Anwendung einer Strategie sein müssen. Die Grundidee des LSPI Algorithmus ist ähnlich dem *Q-Learning*. *Q-Learning* besitzt im eigentlichen Sinne keine Strategie, da sie nicht explizit gespeichert wird, sondern implizit aus den Werten der Zustands-Aktion-Werte-Funktion abgeleitet werden kann.

In Abbildung 1 ist der Algorithmus als Blockdiagramm dargestellt und wird im Folgenden erläutert. Bei LSPI wird die Zustands-Aktions-Werte-Funktion (Q-Funktion) mithilfe einer linearen Architektur approximiert, da es nur bei sehr kleinen Zustands-Aktions-Räumen möglich ist $Q \in \mathbf{R}^{|S|x|A|}$ darzustellen. Die Q-Werte werden über eine linear parametrisierte Kombination aus k-Basis-Funktionen approximiert:

[3] least-squares policy iteration

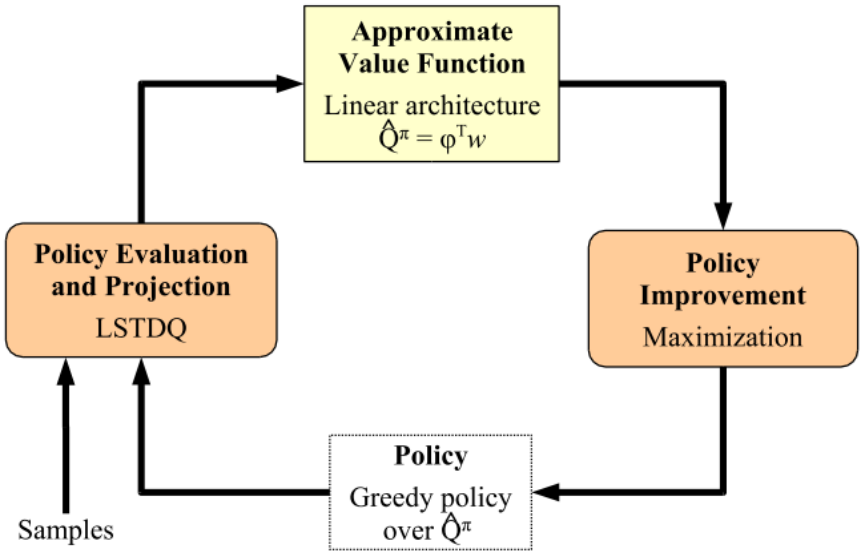

Abbildung 1: Least-squares policy iteration

$$\hat{Q}^\pi(s, a; w) = \sum_{j=1}^{k} \phi_j(s, a) w_j \tag{5}$$

Die Basis-Funktionen $\phi_j(s, a)$ können beliebig gewählt werden, aber sind über den Verlauf der Approximation fest. Sie sind meist nicht-lineare Funktionen $f(s, a)$. Die Werte für w_j sind die Parameter. Für gewöhnlich sind lineare Architekturen Polynome beliebigen Grades oder radiale Basisfunktionen.

Hier ist dargestellt, wie die Strategiebewertung mit einem Algorithmus namens LSTDQ durchgeführt wird. Dieser Algorithmus ähnelt stark dem LSTDL Algorithmus. Bei Approximierung einer linearen Architektur lernt der Algorithmus sehr effizient die approximierte Zustands-Aktions-Wertefunktion $\hat{Q}^\pi$ für eine Strategie π. Wie beim *Supervised Learning*[4] werden hierfür Trainingdaten gesammelt und verarbeitet. Diese Trainigsdaten sind in Tupel der Form (s, a, r, s') organisiert. Hierbei steht s für den Zustand, a für die gewählte Aktion in diesem Zustand, r für die erhaltene Belohnung und s' für den Folgezustand.

[4] Hierbei werden Trainingsdaten verwendet um den Agenten anzulernen. Die Aufgabe des Agenten ist, jedem Eingabewert einen Ausgabewert zuzuordnen.

Ihren Ergebnissen zu Folge schneidet ihr Algorithmus LSPI im Vergleich zu einfachem Q-Learning, mit und ohne *Experience Replay*[5], deutlich besser ab, da schon mit einigen wenigen, zufälligen Versuchen ein gutes Ergebnis erzielt wird. Als Benchmark traten die Algorithmen in verschiedenen Disziplinen gegeneinander an: „Bicycle Riding“, bei der es gilt das Gleichgewicht auf einem Zweirad zu halten und „Inverted Pendulum“, bei der es darum geht, ein Pendel, welches seinen Schwerpunkt oberhalb der Achse hat und sich in seinem höchsten Punkt in einer instabilen Ruhelage befindet, aufzuschwingen und in dieser Position zu halten. Hierbei ist das Pendel auf dem höchsten Punkt eines Wagens montiert, der sich in der Horizontalen bewegen lässt.

2.2 Model-Based Function Approximation in Reinforcement Learning

Das Konzept, welches N. Jong und P. Stone in ihrem Artikel *Model-Based Function Approximation in Reinforcement Learning* beschreiben, lässt sich in drei Schritten grob zusammenfassen:

1. Anhand von Erfahrungsdaten wird ein nicht endliches Modell der Umwelt approximiert
2. Anschließend wird ein endliches Modell erstellt, abgeleitet von dem nicht endlichen Modell und einer endlichen Menge von Zuständen. Über das endliche Modell wird eine approximierte Werte-Funktion definiert.
3. Im letzten Schritt wird die optimale Werte-Funktion des endlichen Modells berechnet.

Jong und Stone (2007)

Ihr Konzept mündete in einem Algorithmus namens *Approximate Models Based on Instances (AMBI)*. Für jeden Zustand $s \in \mathcal{S}$ wählt der Algorithmus eine Aktion $a \in \mathcal{A}$, welches den Q-Wert für (s, a) der approximierten Werte-Funktion, also $\hat{Q}(s, a)$ maximiert. Der Q-Wert spiegelt den zu erwartenden Langzeitwert für ein Zustands-Aktions-Paar wieder. Die Basis dieser Schätzung ist die unmittelbare Belohnung und alle zu erwarteten Belohnungen.

Der erste Schritt des Konzepts ist im AMBI Algorithmus wie folgt gelöst: Auf Basis von Erfahrungsdaten D werden die zu erwartende Belohnung $\mathcal{R}_s^a$ und die Übergangswahrscheinlichkeit $\mathcal{P}_s^a$ für jede Transition approximiert. Jedes Datum $d \in \mathcal{D}$ repräsentiert eine Erfahrung der Form $d = \langle s_d, a_d, r_d, s'_d \rangle$ bestehend aus einem besuchten Zustand s_d, einer gewählten Aktion a_d, einer erhaltenen Belohnung r_d und einem beobachteten Zustandsübergang nach Zustand s'_d. Nun ist es möglich zu generalisieren. Das bedeutet man nimmt alle Zustände $\mathcal{S}^a = \{s_d | d \in \mathcal{D}\}$, die Menge der Zustände bei denen Aktion a gewählt wurde und nutzt

[5] Wird zur beschleunigung des Lernens verwendet. Im Prinzip werden gegangene Pfade gespeichert und wiederholt durchlaufen. Dies beschleunigt die Ausbreitung der gewonnenen Informationen.

einen *Averager*, $\phi^{\mathcal{S}a} : \mathcal{S} \rightarrow \Delta(\mathcal{S}^a)$, der jeden Zustand $s \in \mathcal{S}$ auf eine Wahrscheinlichkeitsverteilung über $\mathcal{S}^a$ abbildet [Gordon (1995)]. Die zu erwartende Belohnung kann dann berechnet werden mit:

$$\hat{\mathcal{R}}_s^a = \sum_{d \in \mathcal{D}^a} \phi_s^{\mathcal{S}^a}(s_d) * r_d \tag{6}$$

Entsprechend ist die Approximierung der Übergangswahrscheinlichkeiten wie folgt definiert:

$$\hat{\mathcal{P}}_s^a(s') = \sum_{d \in \mathcal{D}^a | s + s'_d - s_d = s'} \phi_s^{\mathcal{S}^a}(s_d) \tag{7}$$

Basierend auf diesen Formeln, wird das beschriebene Konzept angewendet. Hierbei kann es zu Problemen kommen, wenn nicht genügend Trainingsdaten $\mathcal{D}^a$ in der Nähe von s vorhanden sind oder wenn $X = s'_d | d \in \mathcal{D}$ nicht genügend Zustände beinhaltet. Im ersten Fall führt das dazu, dass es nicht möglich ist die Auswirkungen von Aktion a zu bestimmen. Im zweiten Fall kommt es dazu, dass nicht alle Folgezustände für eine Aktion berechnet werden können.

Um diesem Problemen adäquat begegnen zu können, wurden bei AMBI zwei virutelle Zustände eingeführt. Der eine Zustand s^{opt} ist der optimale Zustand. Jede Aktion ausgehend von s^{opt} erhält eine Belohnung von V^{max} und geht in den Terminalzustand s^{term} über. Der Zustand s^{term} ist der zweite virutelle Zustand. Alle Aktionen ausgehend von s^{term} erhalten eine Belohnung von 0 und gehen wieder in s^{term} über. Der Zustand s^{term} repräsentiert alle Terminalzustände. Mit diesen Zuständen wird ein gewisser Optimismus auf ungewissem Terrain implementiert. Das bedeutet, dass über die Anzahl der Zustände bestimmt wird wie gut eine Approximierung ist. Wird ein gewisser Grenzwert nicht erreicht, wird s^{opt} als Ergebnis der Approximation eingesetzt. Das Modell muss für den Zustand s^{term} keine Approximation vornehmen, da alle Aktionen keine Auswirkungen im Terminalzustand haben.

3 Abgrenzung der eigenen Arbeitsziele

Um *Reinforcement Learning* nicht nur im Forschungkontext nutzen zu können, sondern auch im Kontext realer Probleme, wurden bereits durch viele Forschungsteams neue Ideen und Konzepte erstellt. All diese Konzepte zielen auf eine Verbesserung der bestehenden Algorithmen ab, d.h. auf Minimierung der Diskrepanz zwischen gelernten und optimalen Funktionen,

Beschleunigung des Lernvorgangs und Approximation von Zustands-Aktions-Räumen oder Werte-Funktionen.

Mein Konzept soll die vielen bestehenden Ansätze aufgreifen und auf deren Basis weitere Verbesserungen im Bereich Geschwindigkeit und Anwendbarkeit auf reale Probleme umfassen. Das genaue Konzept befindet sich noch im Aufbau, aber zeigt in erster Implementierung bereits eine positive Wirkung auf den Lernvorgang (hierzu mehr in Kapitel 5).

Der Unterschied zu bestehenden Konzepten ist der Angriffspunkt um Verbesserungen herbei zu führen. Statt mit einer immer komplexeren Methode zu approximieren, soll ein Modell mithilfe einfacherer, aber mehrerer Methoden approximiert werden. Die verschiedenen Approximierungen bringen einen Vorteil bei unterschiedlichen Problemen, denn die Wahrscheinlichkeit bei mehr als einer Approximation die geeignete für das aktuelle Problem zu finden, ist deutlich höher als bei nur einer.

Um die Leistung des Algorithmus zu zeigen, haben Jong und Stone ihren AMBI Algorithmus verschiedenen Benchmarks unterzogen. Beim *Mountain Car Task* zeigte ihr Algorithmus eine bessere Performance im Vergleich zu anderen Algorithmen, wie z.B. *Least Squares Policy Iteration*. Beim *Mountain Car Task* muss ein untermotorisiertes Auto aus einem Tal entkommen. Um das Tal verlassen zu können, reicht die Leistung des Motors nicht aus. Als Folge dessen muss das Fahrzeug vor und zurück fahren und die Wände des Tals nutzen. Auch in einem zweiten Benchmark konnte der Algorithmus überzeugen.

4 Zusammenfassung

Die in Kapitel 2 vorgestellten Arbeiten *Least-Squares Reinforcement Learning* und *Model-Based Function Approximation in Reinforcement Learning* zeigen den Einfallsreichtum, wenn es um die Verbesserung bestehender Konzepte und Algorithmen im Bereich des *Reinforcement Learning* geht.

Der LSPI[6] Algorithmus von Lagoudakis und Parr zeigt im Vergleich zu gängigen Algorithmen aus dem Bereich des *Reinforcement Learning*, d.h. standard *Reinforcement Learning* mit und ohne *Experience Replay* eine deutliche Verbesserung. Dies wurde erreicht, indem sie mithilfe von Trainingsdaten über die Methode der kleinsten Quadrate die Zustands-Aktions-Werte-Funktion approximierten.

Der AMBI Algorithmus aus der Arbeit von Jong und Stone approximiert sowohl ein Modell zur Vorhersage von Folgezuständen und Belohnungen für das Ausführen einer Aktion in einem Zustand als auch die Werte-Funktion mithilfe von *Averagern*. Dies zeigte bei verschiedenen

[6] least-squares policy iteration

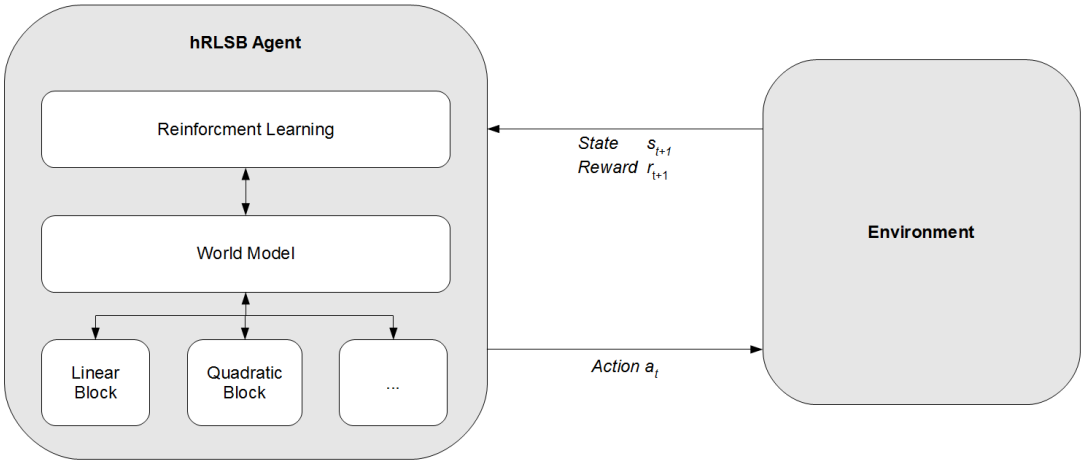

Abbildung 2: Hierarchical Reinforcement Learning using Simulation Blocks

Benchmarks, u.A. beim *Mountain Car Task*, eine bessere Performance im Vergleich zu anderen Algorithmen, wie z.B. *Least Squares Policy Iteration*.

5 Status quo

In Projekt 1 habe ich mich mit Möglichkeiten zur Approximation des Zustands- und Aktionsraums beschäftigt. Hierbei wurde ein Konzept mit dem Namen *Hierarchical Reinforcement Learning Using Simulation Blocks (hRLSB)* erstellt, welches in Abbildung 2 dargestellt ist.

In diesem Konzept gibt es gemäß der Architektur von *Reinforcement Learning* eine Umwelt, die dem Agenten den Zustand s_t mitteilt. Der Agent wirkt auf die Umwelt mit Aktionen a_t. Daraufhin geht die Umwelt in den Zustand s_{t+1} über und teilt dem Agenten den neuen Folgezustand, sowie eine Belohnung r_{t+1} mit.

Der hRLSB-Agent besteht aus mehreren Komponenten. Der *Reinforcement Learning* Block steht für den Lernalgorithmus, wobei im aktuellen Konzept *Q-Learning* zum Einsatz kommt. *Q-Learning* konvergiert bewiesenermaßen bei diskreten Zustandsräumen mit einer Wahrscheinlichkeit von 1. Dies gilt zwar nicht bei kontinuierlichen Zustandsräumen, jedoch ist bei geeigneter Approximation auch hier die Wahrscheinlichkeit der Konvergenz nahe 1, sodass eine beinahe optimale Q-Funktion erreicht wird. [Russell und Norvig (2010)]

Um den Lernvorgang zu beschleunigen, sieht das Konzept ein *World Model* vor. Dieses *World Model* kann mit einer Reihe von Blöcken ausgestattet werden, die zu Beginn mit Trainingsdaten

initialisiert werden müssen. Konkret bedeutet das, dass eine Aktionen versucht wird und zusammen mit dem Folgezustand ein Datenpunkt im Modell gespeichert wird. Durch Ausführen mehrere Aktionen wird so ein Trainigsdatensatz erstellt, auf dessen Basis die verschiedenen Simulations-Blöcke Vorhersagen über Zustands-Aktions-Paare geben können, die nicht in den Trainigsdaten enthalten sind.

Das *World Model* hilft nach dem Training das Lernen zu beschleunigen indem zu jedem realen Interaktionsschritt mit der Umwelt k Schritte simuliert werden. Während der Simulation wählt der Agent eine Aktion und führt diese jedoch nicht auf der Umwelt aus, sondern auf dem World Model. Das World Model führt für jeden Block einen Fehlerwert. Der Fehlerwert gibt an, in wie weit das gelernte Verhalten mit den Trainingsdaten übereinstimmt. Anhand des Fehlerwerts wählt das World Model den besten Block aus. Dieser Block gibt zu der gewählten Aktion eine Vorhersage über den Folgezustand ab.

Innerhalb des Projekts ist eine erste Implementierung des Konzepts entstanden. Hierbei wurden für die Blöcke verschiedene *Least-Squares* Methoden verwendet. Nach kurzer Trainingsphase zum Sammeln einer angemessenen Menge an Trainingsdaten ist schon in der ersten Implementierung eine Beschleunigung des Lernverfahrens zu beobachten. Diese Erkenntnisse machen Mut für die Weiterführung des Projekts.

Literatur

[Abbeel u. a. 2006] ABBEEL, Pieter ; QUIGLEY, Morgan ; NG, Andrew Y.: Using inaccurate models in reinforcement learning. In: *Proceedings of the 23rd international conference on Machine learning*. New York, NY, USA : ACM, 2006 (ICML '06), S. 1–8. – URL http://doi.acm.org/10.1145/1143844.1143845. – ISBN 1-59593-383-2

[Amato und Shani 2010] AMATO, Christopher ; SHANI, Guy: High-level reinforcement learning in strategy games. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*. Richland, SC : International Foundation for Autonomous Agents and Multiagent Systems, 2010 (AAMAS '10), S. 75–82. – URL http://portal.acm.org/citation.cfm?id=1838206.1838217. – ISBN 978-0-9826571-1-9

[Bradtke und Barto 1996] BRADTKE, Steven J. ; BARTO, Andrew G.: Linear Least-Squares algorithms for temporal difference learning. In: *Machine Learning* 22 (1996), S. 33–57. – URL http://dx.doi.org/10.1007/BF00114723. – 10.1007/BF00114723. – ISSN 0885-6125

[Busch 2011] BUSCH, Jan: Modellbasiertes Reinforcement Learning im Kontext des FAUST-Projektes. In: *Anwendungen 1* (2011)

[Carolo-Cup 2011] CAROLO-CUP: *Homepage Carolo-Cup*. 2011. – URL http://www.carolocup.de/

[FAUST 2011] FAUST: *FAUST Homepage*. 2011. – URL http://www.informatik.haw-hamburg.de/faust.html

[Gordon 1995] GORDON, Geo Rey J.: Stable Function Approximation in Dynamic Programming. In: *in Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann, 1995

[Jong und Stone 2007] JONG, Nicholas K. ; STONE, Peter: Model-based function approximation in reinforcement learning. In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA : ACM, 2007 (AAMAS '07), S. 95:1–95:8. – URL http://doi.acm.org/10.1145/1329125.1329242. – ISBN 978-81-904262-7-5

[Jong und Stone 2008] JONG, Nicholas K. ; STONE, Peter: Hierarchical model-based reinforcement learning: R-max + MAXQ. In: *Proceedings of the 25th international conference on Machine learning*. New York, NY, USA : ACM, 2008 (ICML '08), S. 432–439. – URL http://doi.acm.org/10.1145/1390156.1390211. – ISBN 978-1-60558-205-4

[Kaelbling u. a. 1996] KAELBLING, Leslie P. ; LITTMAN, Michael L. ; MOORE, Andrew W.: Reinforcement Learning: A Survey. In: *CoRR* cs.AI/9605103 (1996). – URL http://arxiv.org/abs/cs.AI/9605103

[Lagoudakis und Parr 2003] LAGOUDAKIS, Michail G. ; PARR, Ronald: Least-squares policy iteration. In: *J. Mach. Learn. Res.* 4 (2003), December, S. 1107–1149. – URL http://portal.acm.org/citation.cfm?id=945365.964290. – ISSN 1532-4435

[Moore und Atkeson 1993] MOORE, Andrew W. ; ATKESON, Christopher G.: Prioritized sweeping: Reinforcement learning with less data and less time. In: *Machine Learning* 13 (1993), S. 103–130. – URL http://dx.doi.org/10.1007/BF00993104. – 10.1007/BF00993104. – ISSN 0885-6125

[Russell und Norvig 2010] RUSSELL, Stuart ; NORVIG, Peter ; EDITION, Third (Hrsg.): *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010

[Sutton 1988] SUTTON, Richard S.: Learning to Predict by the Methods of Temporal Differences. In: *Mach. Learn.* 3 (1988), August, S. 9–44. – URL http://portal.acm.org/citation.cfm?id=637912.637937. – ISSN 0885-6125

[Sutton und Barto 1998] SUTTON, Richard S. ; BARTO, Andrew G.: *Reinforcement Learning: An Introduction*. The Mit Press, 1998. – URL http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html

[TORCS 2011] TORCS: *Homepage von TORCS, The Open Racing Car Simulator*. 2011. – URL http://torcs.sourceforge.net/

Abbildungsverzeichnis