Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Grundseminar SoSe 2017

Fabien Lapok

Reinforcement-Learning

Fakultechnik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Fabien Lapok

Reinforcement-Learning

Eingereicht am: 31. August 2017

Inhaltsverzeichnis

1 Motivation

In dem frei wählbarem Teil des Informatik-Studiums, der aus den Grund- und Hauptprojekt besteht habe ich beschlossen, mich mit einem Thema auseinanderzusetzen, welches für mich vollkommen neu ist. In der Recherche während des Grundseminars bin ich auf *Reinforcement-Learning* gestoßen, was Teilbereich des maschinellen Lernens ist. Reinforcement-Learning hat meine Aufmerksamkeit erregt, weil es sich im Kern mit einer interdisziplinären Fragestellung beschäftigt, die man beispielsweise auch in der Ökonomie, Neurowissenschaften und der Psychologie wiederfinden kann: die Wissenschaft der Entscheidungsfindung. Schon im frühen zwanzigsten Jahrhundert haben Iwan Petrowitsch Pawlow und später auch B. F. Skinner mit der klassischen und operanten Konditionierung das Lernverhalten von Tieren untersucht, indem sie mithilfe von Belohnung und Bestrafung bestimmtes Verhalten der Tiere gefördert bzw. gehemmt haben. Diese fundamentale Idee des Lernens durch Interaktion mit der Umwelt bedient sich auch das *Reinforcement-Learning*, in dem der Agent durch Belohnung und Bestrafung eine Lösung eines Problems zu *erlernen* versucht. Durch diese Vorgehensweise steht *Reinforcement-Learning* im Kontrast zu vielen anderen Lernmethoden des maschinellen Lernens wie *Supervised-Learning*, in dem das System durch Beispieldaten lernt.
Der nächste Abschnitt beschäftigt sich mit den Grundlagen von *Reinforcement-Learning*, den Problemtypen, die damit gelöst werden und den Herausforderungen vor denen *Reinforcement-Learning* steht. Darauf aufbauend werden im dritten Abschnitt einige Lösungsansätze aus der aktuellen Forschung für die Probleme von *Reinforcement-Learning* vorgestellt. Abschließend widmet sich der Abschnitt *Ausblick* meinem geplanten Vorgehen für das Grund- und Hauptprojekt.

2 Grundlagen

Anders als andere Lernverfahren (wie neuronale Netze oder Deep-Learning), die auch als überwachtes Lernen bezeichnet werden und bei denen das System mithilfe von Trainingsdaten angelernt wird, wird beim *Reinforcement-Learning* (RL) eine Strategie verfolgt, in dem das System durch Interaktion mit der Umwelt lernt. RL ist das erlernen von Aktionen zu gegebenen Situationen (oder Zuständen) - mit anderen Worten das Abbilden von Situationen auf Aktionen - mit dem Ziel, ein numerisches Belohnungssignal zu maximieren. Dem System wird dabei nicht vorgegeben, welche Aktion wann ausgeführt werden soll; vielmehr ist es Aufgabe des Systems durch Ausprobieren herauszufinden, welche Aktions- und Zustandsfolge die größtmögliche Belohnung einbringt. Dabei spielt oft nicht nur die unmittelbare Belohnung einer Aktion eine Rolle, sondern die Summe aller Belohnungen. (Sutton und Barto, 2012, S.4)

Es gibt viele Problemtypen, die mithilfe von RL gelöst werden können. Prominente Vertreter sind Brett- und Videospiele wie Go oder Atari Spiele, Roboter oder Drohnen, die bestimmte Manöver erlernen oder auch Algorithmen für das Börsengeschäft. Dabei wird grundlegend zwischen episodischen Entscheidungsproblemen, die ein oder mehrere Endzustände haben (beispielsweise Brettspiele) und nicht episodischen Entscheidungsproblemen, die fortlaufend sind (beispielsweise Algorithmen für das autonome Fahren im Fahrzeug). Im nächsten Abschnitt wird der Aufbau und die Funktionsweise grundlegender Algorithmen eines RL-Systems vorgestellt.

2.1 Funktionsweise von Reinforcement-Learning

Vereinfacht dargestellt, besteht ein RL-System aus einem Agenten und seiner Umwelt. Der Agent kann in jedem Zustand aus einem Satz von Aktionen eine Aktion wählen. Führt der Agent eine Aktion aus, erhält er von der Umwelt ein Feedback - im Sinne von Belohnung bzw. Bestrafung - und auf Grundlage der gewählten Aktion ein Folgezustand. An diesem Beispiel kann man eine wesentliche Eigenschaft von RL erkennen: bei RL handelt es sich um ein episodisches bzw. sequenzielles Lernen, in dem der Agent, je nach Strategie, mit jeder Aktion besser werden kann. Die Abbildung 2.1 zeigt skizzenhaft die entsprechenden Komponenten eines RL-Systems.

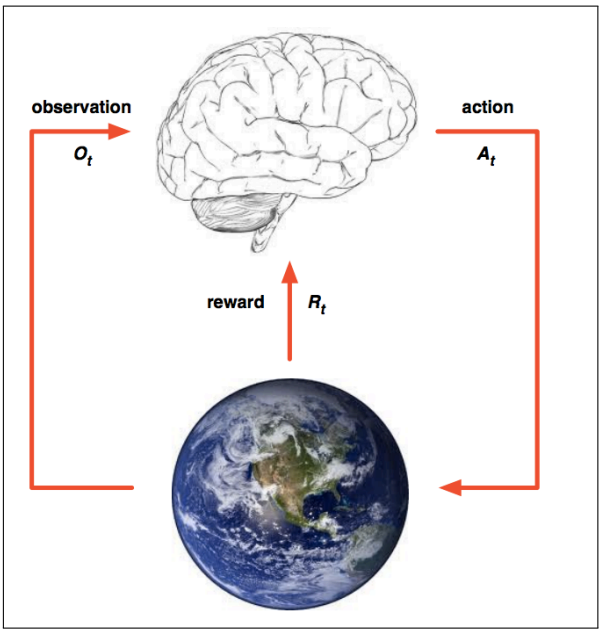

Abbildung 2.1: Agent und Umwelt (Silver, 2015)

Näher betrachtet, kann man neben der Umwelt und dem Agenten noch weitere Bestandteile eines RL-Systems identifizieren: eine (Verhaltens-)Strategie (*policy*), eine Belohnungsfunktion (*reward function*), eine Wertfunktion (*value function*) und optional ein Modell der Umwelt, auf die im Folgenden eingegangen wird.

Die policy beschreibt das Verhalten des Agenten zu einer bestimmten Zeit, also die Aktionen, die der Agent zu den jeweiligen Zuständen ausführt. Diese policy kann in einigen Fällen eine einfache Zuordnungsfunktion von Zustand auf Aktion sein.

Über die **reward function** wird indirekt das Ziel des RL-Problems definiert, indem sie dem Agenten für eine bestimmte Aktion in einem bestimmten Zustand eine Rückmeldung in Form einer Zahl zurückgibt. Ziel des Agenten ist im Laufe seiner Aktionsfolge die Summe dieses Feedbacks zu maximieren.

Die value function ist im Gegensatz zur reward function, die die unmittelbare Belohnung darstellt, eine Vorhersage über die zu erwartende Summe an Belohnung zu gegebenem Zustand. Bei der Vorhersage werden die (wahrscheinlichen) Folgezustände und die entsprechenden Belohnungen berücksichtigt.

Das letzte Element in einem RL-System stellt das **Modell** der Umgebung dar, welches optional eingesetzt werden kann, um das Verhalten der Umgebung zu simulieren. Dieses Modell kann genutzt werden, um nach einer Aktion zu einem gegebenen Zustand den Folgezustand vorherzusagen, wodurch dem Agenten, die Möglichkeit gegeben wird in die Zukunft zu planen. (Sutton und Barto, 2012, S.7 ff)

Der nächste Abschnitt beschäftigt sich mit der Modellierung eines solchen RL-Systems, welches auch als Markov-Decision-Process bezeichnet wird.

2.2 Markov-Decision-Process

Der Markov-Decision-Process (MDP) ist ein Modell, um die Umwelt eines RL-Problems formell zu beschreiben. Um die Umwelt entsprechend als MDP beschreiben zu können, muss sie eine wichtige Eigenschaft besitzen, die als Markov-Eigenschaft bezeichnet wird. Die Markov-Eigenschaft besagt, dass der Folgezustand der Umwelt nur von dem aktuellen Zustand (und der aktuellen Aktion) abhängt und die Historie der vorherigen Zustände vernachlässigt werden kann. Für den Agenten bedeutet das, dass er seine Aktionen nur auf Basis des aktuellen Zustands treffen muss, ohne seine vergangenen Zustände und Aktionen in Betracht ziehen zu müssen. (Sutton und Barto, 2012, S.54f.)

Im Weiteren wird auf die formelle Definition von MDP verzichtet, die bei Bedarf im Standardwerk von Sutton und Barto (2012) nachgeschlagen werden kann. Stattdessen wird direkt auf zwei grundlegende Algorithmen hingewiesen, die für die Lösung des RL-Problems genutzt werden. Die folgenden Algorithmen werden auch als Bellman-Gleichungen bezeichnet. Die Formel 2.1 repräsentiert die im vorigen Abschnitt erwähnte *value function*. Die zweite Formel ist die sogenannte *action-value function*, die jeder möglichen Aktion a von jedem Zustand s eine Belohnung zuweist. (Bellman, 1957, S. 151)

$$V^*(s) \leftarrow \max_{a \in A} R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \cdot V^*(s') \tag{2.1}$$

$$Q^*(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \cdot \max_{a' \in A} Q^*(s',a') \tag{2.2}$$

Die Funktion $R(s,a)$ ist die *reward function*, die die Belohnung für das Ausführen einer Aktion a zum gegebenen Zustand s zurückgibt. γ ist der sogenannte *discound factor*, worüber die Berücksichtigung von in Zukunft liegenden Werten definiert wird. Für γ gilt $0 \leq \gamma \leq 1$ - je höher der Wert, desto gewichtiger sind die zukünftigen Zustände. T spiegelt die Transiti-

onswahrscheinlichkeit von dem Zustand s auf den Folgezustand s' für eine gegebene Aktion a wieder. Die Lösung dieser Bellman-Gleichungen erfolgt iterativ, wobei unterschiedliche Lösungsverfahren, wie *Value Iteration*, *Policy Iteration*, *Q-Learning* und *Sarsa*, genutzt werden (Silver, 2015).

Die iterative Lösung der Gleichungen kann bei großen Zustands- oder Aktionsräumen sehr aufwändig sein, was dazu führt, dass große Probleme mit der hier aufgeführten Vorgehensweise nicht gelöst werden können. Mit dieser Überlegung widmet sich der nächste Abschnitt einigen Herausforderungen von RL-Systemen.

2.3 Herausforderungen

Es gibt eine Reihe von Herausforderungen bei der Lösung von Problemen mit RL-Algorithmen. In diesem Abschnitt werden zwei für die weitere Arbeit wichtigsten Probleme aufgeführt, von denen einige Lösungsansätze im nächsten Abschnitt vorgestellt werden.

- *Zustandsraumexplosion* (*curse of dimensionality*) bezeichnet das im vorigen Abschnitt angedeutete Problem, in dem der Zustandsraum so groß ist, dass sich das Problem nicht mit den klassischen RL-Algorithmen lösen lässt. Spiele wie Schach oder Go haben so große Zustandsräume, dass sie mit den klassischen Mitteln nicht in einer akzeptabler Zeit gelöst werden können.

- Ein weiteres Problem stellt das Feedbacksignal der Umwelt dar. Es gibt Problemstellungen, in denen der Agent erst sehr spät oder sogar erst am Ende seiner Aktionsfolge ein Feedback erhält und dadurch nicht weiß, welche Aktionen in der Aktionsfolge gut oder schlecht waren. Um bei dem Beispiel Schach zu bleiben, erhält der Agent erst am Ende des Spiels ein Feedback darüber, ob seine *policy* zu einem Gewinn oder Niederlage geführt hat. Dabei ist schwer zu entscheiden, welche Zustände und Aktionen ausschlaggebend waren.

3 Aktuelle Forschung

Im vorigem Kapitel wurden zwei Herausforderungen von RL-Systemen vorgestellt, die einerseits auf die Komplexität des Problems und andererseits auf ein spätes Feedbacksignal zurückzuführen sind. Im folgenden werden zwei aktuelle Artikel vorgestellt, die sich mit diesen Problemen beschäftigen und Lösungsansätze dafür bieten.

3.1 Minimierung des Zustandsraums durch Approximationsfunktionen

Die Autoren des ersten Artikels *Mastering the Game of Go with Deep Neural Networks and Tree Search* (Silver u. a., 2016) zeigen einen Ansatz zum Lösen des klassischen Brettspiels Go, welches aufgrund seines riesigen Suchraums und der Schwierigkeit Positionen (bzw. Zustände) und Aktionen zu bewerten, lange Zeit als anspruchsvollstes klassisches Brettspiel galt. Dabei bauen sie auf einem Ansatz auf, das bereits vorher zum Erlernen von Atari Spielen von Mnih u. a. (2013) eingesetzt wurde und als *Deep-Reinforcement-Learning* (DRL) bezeichnet wird.

DRL kombiniert die beiden Lernverfahren *Reinforcement-Learning* mit *Deep-Learning* und wurde von Mnih u. a. (2013) verwendet um die *Strategie* des RL-Agenten direkt aus dem hochdimensionalem visuellen Input der Atari Spiele zu lernen, ohne spielspezifische Informationen oder handgestrickte visuelle Merkmale dem RL-Agenten vorzugeben. Der Agent lernte die Spiele ausschließlich aus dem Video-Input, der Belohnung und der Terminalausgaben. Das neuronale Netz diente in dem Szenario als nichtlineare Approximationsfunktion, die die im Abschnitt 2.2 erwähnte *action-value funktion* $Q(s, a)$ repräsentiert.

Auch Silver u. a. (2016) verwenden DRL mit einer Approximationsfunktion; hier zur Minimierung des Suchraums. Der Suchraum von dem Brettspiel Go erstreckt sich auf ungefähr 250^{150} mögliche Reihenfolgen von Zügen. Die Autoren sind zu dem Entschluss gekommen, dass eine umfassende Suche in der Größenordnung nicht durchführbar ist. Stattdessen stellen sie ein Ansatz vor, bei dem eine Minimierung des Suchraums durch die folgenden zwei Prinzipien möglich ist:

1. Erstens kann die Tiefe des Suchbaums durch eine Positionsbewertung reduziert werden. Das kann erreicht werden, indem der Suchbaum an einem Zustand s gekürzt wird und der darauf folgende Unterbaum über eine approximierte *value function* ($v(s) \approx v_*(s)$) ersetzt wird, welche das Ergebnis für den Zustand s schätzt. Dadurch muss der Suchbaum nicht mehr in der vollständigen Tiefe erkundet werden, sondern nur bis zu einer bestimmten Tiefe, woraufhin die Approximationsfunktion folgt.

2. Zweitens kann die Breite des Suchbaums durch stichprobenartiges Ausprobieren von Aktionen aus einer *policy function* $p(a|s)$ minimiert werden, welche eine Wahrscheinlichkeitsverteilung über mögliche Aktionen a in einem gegebenen Zustand s ist. Diese Funktion gibt aus einer Menge von allen Aktionsmöglichkeiten die Aktionen zurück, die für die Lösung des Problems am vielversprechendsten sind.

Den Lernprozess haben die Autoren in drei Stufen unterteilt:

Die erste Stufe bestand daraus, Vorhersagen von Expertenzügen zu erlernen, was das Fundament für die *policy Funktion* $p(a|s)$ darstellt. Dafür wurde ein neuronales Netz mit 30 Millionen Positionen echter Beispieldaten aus Go Spielen trainiert.

Die zweiten Stufe des Lernprozesses diente der Verbesserung des zuvor trainierten *policy Netzwerks* mithilfe von *Reinforcement-Learning*, indem die Autoren Spiele zwischen dem aktuellen *policy Netzwerks* und einer zufällig ausgewählten vorherigen Iteration des *policy Netzwerks* durchführten.

Die letzten Stufe konzentrierte sich auf die Approximationsfunktion $v(s)$, welche für die Vorhersage des Ergebnisses für ein Zustand (oder Position) s in einem Spiel unter der Verwendung einer *policy* p bestimmt. Dafür wurde ein neuronales Netzwerk trainiert, welches das Ergebnis für die zuvor am besten trainierte *policy Netzwerks* vorhersagt.

Das Ergebis der Autoren ist ein System, welches als erstes, den besten europäischen und später auch den Weltbesten Go Spieler geschlagen hat.

Der Einsatz von Approximationsfunktionen haben sich bei Problemen mit großem Zustandsraum auch in weiteren Beispielen z.B. (V. Mnih u. a., 2015) als vielversprechend erwiesen. Sie haben aber weiterhin Schwierigkeiten mit Problemen, die ein spätes und seltenes Belohnungssignal besitzen. Im nächsten Abschnitt wird eine Methode vorgestellt, die sich mit dieser Problemklasse beschäftigt.

3.2 Definition von Zwischenzielen zum Lösen von Problemen mit spätem Feedback

Die Autoren des zweiten Artikels *Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation* verwenden zusätzlich zum DRL eine hierarchische Zerlegung des Problems, was auch als *Hirarchical-Reinforcement-Learning* (HRL) bezeichnet wird. HRL wird genutzt um komplexe Probleme in kleinere Probleme aufzuteilen, die einzeln gelöst und anschließend zusammengeführt werden um das ursprüngliche Problem zu bewältigen. HRL zielt auf eine Minimierung des Zustandraumes, die Wiederverwendbarkeit von Teillösungen und eine breitere Erkundung des RL-Agenten ab. (Hengst, 2012)

Die Arbeit von Kulkarni u. a. (2016) baut auf zwei Konzepten, die typisch für HRL-Systeme sind auf: *Semi-Markov-Decision-Problems* (semi-MDPs) und *value-function decomposition*:

1. Semi-MDPs sind erweiterte MDPs, die zusätzlich abstrakte Aktionen beinhalten. Abstrakte Aktionen umfassen eine unbestimmte Anzahl an Aktionen über den ursprünglichen Aktionsraum A und definieren darüber eine mehrstufige Zustandstransition, woraus sich die Belohnung der abstrakten Aktion ergibt. (Hengst, 2012, S. 297)

2. Die value-funktion decomposition oder hierarchische Value-Funktionen bezeichnet eine modifizierte *value function*, die die Belohnung für den Zustand s abhängig von einem Zwischenziel bewertet. Der ursprünglichen Funktion $V(s)$ wird ein weiterer Parameter g hinzugefügt, der das Zwischenziel repräsentiert. Die neue Funktion $V(s, g)$ bewertet den Zustand s für das Erreichen des Ziels $g \in G$ (R. S. Sutton und Precup, 2011; T. Schaul und Silver, 2015).

Kulkarni u. a. (2016) stellen einen RL-Agenten vor, welcher durch *intrinsische Motivation* [1] das Problem des rarem Feedbacks lösen soll. Dafür wird ein Framework vorgestellt, welches die Autoren als *hirarchical-DQN* (h-DQN) bezeichnen. Das Framework integriert die zuvor erwähnte hierarchische *value function* in den Agenten. Ihr Modell trifft über zwei hierarchischen Ebenen Entscheidungen:

1. Ein sogenannter *Meta-Controller* definiert über den aktuellen Zustand ein Zwischenziel g.

2. Im zweiten Schritt entscheidet ein darunterliegendes Modul, der *Controller*, anhand des Zustandes und des gesetzten Zwischenziels g, welche Aktion ausgeführt werden, um das Zwischenziel zu erreichen oder bei Misserfolg die Lernfolge zu beenden.

[1] Mit intrinsischer Motivation meinen die Autoren ein von dem Agenten selbst gesetztes Zwischenziel.

Die beiden Schritte werden wiederholt bis eine optimale Strategie gefunden wurde um das übergeordnete Ziel zu erreichen. Die Autoren verwenden als Beispielproblem unteranderem das Atari-Spiel *Montezuma's Revenge*, welches ein sehr spätes Feedback-Signal hat und mit vorigen RL-Agenten unzureichend gelöst wurde. Das Zusammenspiel von *Meta-Controller* und *Controller* lässt sich anhand des folgenden Auszugs der Bilderfolge des Spiels verdeutlichen. In

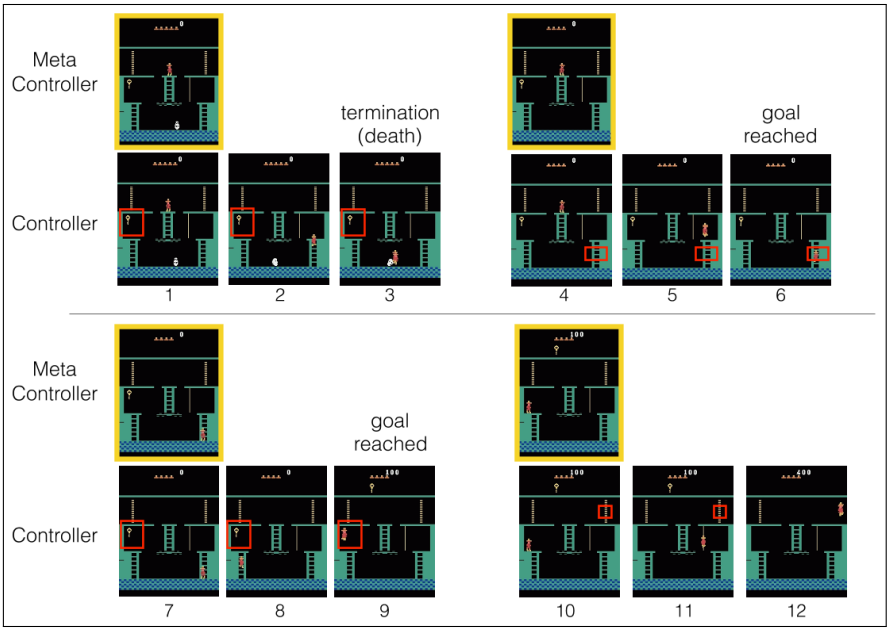

Abbildung 3.1: Zusammenspiel von *Meta-Controller* und *Controller* (Kulkarni u. a.)

der Bilderfolge oben Links der Abbildung 3.1 sieht man den Input des *Meta-Controllers*, der dem *Controller* ein Ziel (den Schlüssel) vorgibt. Nach dem iterativen Durchlaufen des zweiten Schritts bricht der *Controller* aufgrund der Termination durch den Tod ab. Anschließend bewertet der *Meta-Controller* (Bilderfolge oben Rechts) die Situation neu und gibt dem *Controller* die Leiter als neues Ziel vor. Nach dem Erreichen des Ziels wiederholt sich der Vorgang, bis die Tür geöffnet wird.

An dem Beispiel wird deutlich, wie das übergeordnete Ziel (das Öffnen der Tür) in die kleinen Ziele (erreichen der Leiter, erreichen des Schlüssels, der Rückweg über die Leiter und abschließend die Tür) aufgeteilt wird. Hierbei wird auch das späte Feedbacksignal deutlich, welches erst auftritt, wenn der Schlüssel erreicht wird und bereits zwei Zwischenziele des *Meta-Controllers* verfolgt wurden. Der von den Autoren vorgestellte Ansatz war in der Lage, zwei Probleme mit spätem Feedback zu lösen, welche bis dahin mit DRL noch nicht gelöst werden könnten.

4 Ausblick

Das folgende Grundprojekt soll dafür genutzt werden, sich ein Überblick über die vorhandenen Werkzeuge und ihrer Funktionsweise zu beschaffen. Dabei soll ein übersichtliches episodisches Problem, wie beispielsweise das Würfelspiel *Kniffel* als Übungsbeispiel fungieren. Nach dem Beherrschen der grundlegenden Theorien und Strategien von RL soll sich im darauf folgenden Hauptprojekt an die vorgestellten Problemtypen und ihrer Lösungsansätze mit einem geeigneten Beispiel genähert werden. Hier sind Probleme mit einem komplexen Zustandsraum oder ein nichtepisodisches (bzw. fortlaufendes) Problem denkbar. Aufbauend auf den im Grund- und Hauptprojekt erlangten Erkenntnissen, soll in der Master-Thesis ein Konzept entwickelt werden, welches sich auf das HRL bezieht und im Idealfall eine Strategie vorstellt, welche Zwischenziele noch effektiver erkennt und definiert.

Literaturverzeichnis

[Bellman 1957] Bellman, Richard: *Dynamic Programming*. 1957

[Hengst 2012] Hengst, Bernhard: *Chapter 9 Hierarchical Approaches. In Reinforcement Learning State-of-the-Art, von Marco Wiering und Martijn van Otterlo (Eds.), S. 293-323*. 2012

[Kulkarni u. a.] Kulkarni, Tejas D. ; Narasimhan, Karthik R. u. a.. – https://www.semanticscholar.org/paper/Hierarchical-Deep-Reinforcement-Learning [Online; zugriff am 29. August 2017]

[Kulkarni u. a. 2016] Kulkarni, Tejas D. ; Narasimhan, Karthik R. u. a.: Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. (2016)

[Mnih u. a. 2013] Mnih, Volodymyr u. a.: Playing Atari with Deep Reinforcement Learning. (2013)

[R. S. Sutton und Precup 2011] R. S. Sutton, M. Delp T. Degris P. M. Pilarski A. W. ; Precup, D.: *Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, Seiten 761–768. International Foundation for Autonomous Agents and Multiagent Systems*. 2011

[Silver 2015] Silver, David: *Introduction to Reinforcement Learning*. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/intro_RL.pdf. 2015. – [Online; zugriff am 22. August 2017]

[Silver u. a. 2016] Silver, David ; Huang, Aja u. a.: Mastering the Game of Go with Deep Neural Networks and Tree Search. (2016)

[Sutton und Barto 2012] Sutton, Richard S. ; Barto, Andrew G.: *Reinforcement Learning: An Introduction, 2nd edition*. 2012

[T. Schaul und Silver 2015] T. Schaul, K. G. ; Silver, D.: *Universal value function approximators. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), Seiten 1312–1320.* 2015

[V. Mnih u. a. 2015] V. Mnih, D. Silver A. A. Rusu J. Veness M. G. Bellemare A. Graves M. Riedmiller A. K. Fidjeland G. O. u. a.: Human-level control through deep reinforcement learning. Nature, 518(7540):529–533. (2015)