wirtschafts
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Fakultät für
Wirtschafts-
wissenschaften
Business School

Softwarelösungen für Reinforcement Learning

Stefan Selle

Professor für Wirtschaftsinformatik
Fakultät für Wirtschaftswissenschaften
Hochschule für Technik und Wirtschaft des Saarlandes

Saarbrücken, 06.03.2018

Kurzfassung

Reinforcement Learning (RL) oder auch bestärkendes Lernen fasst Lernprobleme zusammen, bei denen ein Agent mit seiner Umwelt interagiert, um ein bestimmtes Ziel zu erreichen. Der Agent macht Beobachtungen und kann selbstständig Entscheidungen treffen und Aktionen ausführen, die seinen Umweltzustand ändern. Er bekommt Rückmeldungen in Form von Belohnungen (engl. *Rewards*) und versucht eine Strategie bzw. Politik (engl. *Policy*) zu finden, um das vorgegebene Ziel zu erreichen. Einerseits kann das *Trial-and-Error*-Prinzip angewendet werden, um zunächst Erfahrungen zu sammeln (*Exploration*). Andererseits kann das bislang gelernte Wissen auch direkt angewendet werden, um kurzfristig Belohnungen zu bekommen (*Exploitation*). Mathematisch betrachtet können viele RL-Lernprobleme als Markov Entscheidungsprozess (engl. *Markov Decision Process (MDP)*) mit der Bellman-Gleichung als Ausgangspunkt für einen Lernalgorithmus besschrieben werden. Ziel ist es, eine optimale Strategie zu finden, um möglichst gute Zustände zu besetzen und am Ende die Summe der (diskontierten) Rewards zu maximieren. Hierzu werden spezielle Q-Wertefunktionen approximiert. Bekannte Algorithmen sind bspw. Q-Learning und SARSA. Mit dem Aufkommen von *Deep Learning (DL)* lassen sich auch tiefe Künstliche Neuronale Netzwerke (KNN) trainieren. KNN können u.a. zur Funktionsapproximation verwendet werden, somit lassen sich diese Verfahren auch in RL-Problemen einsetzen. Bekannte Algorithmen des *Deep Reinforcement Learning (DRL)* sind bspw. Deep Q-Network (DQN) oder DeepSARSA.

Diese Arbeit vermittelt die theoretischen Grundlagen des *Reinforcement Learning*. Ziel dieser Arbeit ist es, eine oder mehrere Softwarelösungen zum Thema *Reinforcement Learning* zu finden, um diese in Lehre und Forschung an der Hochschule für Technik und Wirtschaft des Saarlandes einzusetzen. Hierzu werden 37 *Open Source* Lösungen präsentiert, dazu gehören Umgebungen, Bibliotheken und Frameworks, Spezialsoftware sowie Tutorials. Die meisten dieser Lösungen sind auf der Plattform GitHub zu finden und in der Programmiersprache Python programmiert. Viele basieren auf dem DL-Framework TensorFlow. Mit Hilfe einer Nutzwertanalyse werden elf interessante Lösungen bewertet und miteinader verglichen. Nur fünf dieser Lösungen lassen sich auf ein Testsystem fehlerfrei installieren und ausführen. Drei interessante Lösungen werden näher analysiert und typische RL-Experimente mit verschiedenen Agenten bzw. Algorithmen durchgeführt. Durch den qualitativen Vergleich konnte schließlich eine geeignete RL-Software ermittelt werden, die zukünftig in Forschung und Lehre an der htw saar eingesetzt werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis **v**

Tabellenverzeichnis **vii**

Abkürzungsverzeichnis **ix**

1 Einleitung **1**

2 Theoretische Grundlagen **5**
2.1 Lernen . . . 5
2.2 Reinforcement Learning . . . 6
2.3 Problemstellungen . . . 7
2.4 Markovsche Entscheidungsprozesse . . . 10
2.5 Lösungsansätze . . . 12
2.6 Exploration vs. Exploitation . . . 16
2.7 Algorithmen . . . 17

3 RL-Software **21**
3.1 GitHub . . . 21
3.2 Programmiersprachen . . . 22
3.3 Klassifikation . . . 24

4 Umgebungen **27**
4.1 Arcade Learning Environment . . . 28
4.2 CARLA . . . 30
4.3 DeepMind Lab . . . 32
4.4 OpenAI Gym . . . 34
4.5 OpenAI Universe . . . 36
4.6 Project Malmo . . . 38
4.7 PyGame Learning Environment . . . 40
4.8 Retro Learning Environment . . . 42
4.9 Roboschool . . . 44
4.10 ViZDoom . . . 46

5 Bibliotheken und Frameworks **49**
5.1 ChainerRL . . . 51
5.2 Coach . . . 52
5.3 DeepRL . . . 54
5.4 Keras-RL . . . 55
5.5 OpenAI Baselines . . . 56
5.6 OpenAI Lab . . . 57
5.7 Ray RLlib . . . 58
5.8 RL4J . . . 59
5.9 RLLab . . . 60
5.10 TensorFlow Agents . . . 61

5.11 TensorForce . . . 62
5.12 TensorLayer . . . 63
5.13 Torch-TWRL . . . 64

6 Sonstige **65**
6.1 BURLAP . . . 66
6.2 MAgent . . . 68
6.3 PyBrain . . . 69
6.4 RLPark . . . 70
6.5 RLPy . . . 71
6.6 MMLF . . . 72
6.7 RL-Glue . . . 73
6.8 RL Toolkit . . . 74

7 Tutorials **75**
7.1 Basic-RL . . . 76
7.2 Practical-RL . . . 77
7.3 RL . . . 78
7.4 RL An Introduction . . . 79
7.5 RLcode . . . 80
7.6 RL with TensorFlow . . . 81

8 Analysen **83**
8.1 Testsystem . . . 83
8.2 Scoring . . . 84
8.3 Installationen und erste Tests . . . 91

9 Experimente **99**
9.1 Keras-RL . . . 99
9.2 OpenAI Lab . . . 107
9.3 Ray RLlib . . . 114
9.4 TensorLayer . . . 123
9.5 Fazit . . . 126

10 Zusammenfassung und Ausblick **129**

Quellenverzeichnis **135**

Abbildungsverzeichnis

2.1 Agenten-Umwelt-Interaktion des RL 6
2.2 Schematische Darstellung eines inversen Pendels auf einem Wagen 8
2.3 Schematische Darstellung des Mountain Car Problems 8
2.4 Gridworld 9
2.5 Cliff Walking Problem 9
2.6 Einfache Frozen Lake Umgebung als 4 x 4 Gitter 10
2.7 Politik-Iteration 12
2.8 Schematische Darstellung der Actor Critic Architektur 15
2.9 Schematischer Aufbau eines tiefen KNN zum DRL 18

3.1 Klassifikation von RL-Softwarelösungen 24

4.1 ALE: Die Atari-Spiele Pitfall und Space Invadors 28
4.2 ALE: 160x210-Pixel-Output des Emulators Stella zum Spiel Space Invaders 28
4.3 CARLA: Die gleiche Simulation in vier verschiedene Wettersituationen . . 30
4.4 CARLA: Output von 3 verschiedenen Sensoren zur gleichen Szene 31
4.5 CARLA: Grundrisse zweier verschiedener Welten 31
4.6 DeepMind Lab Level 1: Nav Maze 32
4.7 DeepMind Lab Level 2: Stairway to Melon 33
4.8 OpenAI Gym: Einfache, textbasierte Umgebungen 34
4.9 OpenAI Gym: Klassische Kontrollaufgaben 34
4.10 OpenAI Universe Architektur 36
4.11 OpenAI Universe: Flash-Spiel Dusk Drive 37
4.12 Project Malmo: Beispielumgebung mit Mensch und Agent als Spieler . . . 38
4.13 Project Malmo: Catch the Pig 39
4.14 PyGame LE: Beispielumgebungen 40
4.15 RLE: Atari 2600 Boxing und SNES Mortal Combat 42
4.16 RLE: Super Mario und F-Zero 42
4.17 Roboschool: Pong 44
4.18 Roboschool: Walker und Ant 44
4.19 Roboschool: Humanoid und Lauf mit drei Robotern 45
4.20 ViZDoom: Normaler Output vs Deep Buffer 46
4.21 ViZDoom: Normalmodus und Health Gathering Szenario 47

5.1 TensorFlow Programmierumgebung 50
5.2 Architektur des Coach-Frameworks 52
5.3 Algorithmen des Coach-Frameworks 52

6.1 Design-Prinzip von RL-Glue 73

8.1 Gym-Umgebung Go 92
8.2 Malmo: Minecraft 93
8.3 Malmo: Umgebung und Agent 93

9.1 Jupyter Notebook: Keras-rl Cart Pole SARSA: Python Quelltext 105

9.2 Jupyter Notebook: Keras-rl Cart Pole SARSA: Training 105
9.3 Jupyter Notebook: Keras-rl Cart Pole SARSA: Testergebnis 106
9.4 CartPole & SARSA 110
9.5 CartPole & DQN 111
9.6 CartPole & AC 112
9.7 Ray RLLib API 114
9.8 Ray RLLib Komponenten 115
9.9 Ray-Python-Programm im Jupyter-Notebook 119
9.10 Ray-TensorBoard mit Ergebnissen zu A3C 120
9.11 Ray-TensorBoard mit Ergebnissen zu ES 120
9.12 Ray-TensorBoard mit Ergebnissen zu DQN 121
9.13 Ray-TensorBoard mit Mouse-Over-Funktionalität 121

Tabellenverzeichnis

3.1 Übersicht von RL-Repositories auf der Plattform GitHub 21
3.2 Übersicht von Attributen zu einem Repository auf GitHub 22

4.1 Übersicht ausgewählter RL-Umgebungen auf der Plattform GitHub 27
4.2 Steckbrief zum ALE-Framework 29
4.3 Steckbrief zum Simulator CARLA 31
4.4 Steckbrief zur DeepMind-Lab-Umgebung 33
4.5 Steckbrief zur OpenAI-Gym-Umgebung 35
4.6 Steckbrief zur OpenAI-Universe-Plattform 37
4.7 Steckbrief zur Project Malmo Plattform 39
4.8 Steckbrief zur PyGame LE 41
4.9 Steckbrief zum Retro Learning Environment 43
4.10 Steckbrief zur Roboschool 45
4.11 Steckbrief zur ViZDoom Plattform 47

5.1 Übersicht ausgewählter RL-Softwarelösungen auf der Plattform GitHub . 49
5.2 Steckbrief zur ChainerRL-Bibliothek 51
5.3 Steckbrief zum Coach-Framework 53
5.4 Steckbrief zur DeepRL-Bibliothek 54
5.5 Steckbrief zur Keras-RL-Bibliothek 55
5.6 Steckbrief zur OpenAI-Baselines-Bibliothek 56
5.7 Steckbrief zum OpenAI Lab Framework 57
5.8 Steckbrief zum Framework Ray 58
5.9 Steckbrief zur RL4J-Bibliothek 59
5.10 Steckbrief zur Bibliothek RL Lab 60
5.11 Steckbrief zur Bibliotkek TensorFlow Agents 61
5.12 Steckbrief zur TensorForce-Bibliothek 62
5.13 Steckbrief zur TensorLayer-Bibliothek 63
5.14 Steckbrief zur Bibliothek Torch-TWRL 64

6.1 Übersicht spezieller RL-Lösungen auf der Plattform GitHub 65
6.2 Übersicht sonstiger RL-Lösungen 65
6.3 Steckbrief zur BURLAP-Bibliothek 67
6.4 Steckbrief zur MAgent-Plattform 68
6.5 Steckbrief zur PyBrain-Bibliothek 69
6.6 Steckbrief zur RLPark-Bibliothek 70
6.7 Steckbrief zum RLPy-Framework 71
6.8 Steckbrief zu MMLF 72
6.9 Steckbrief zu RL-Glue 73
6.10 Steckbrief zum RL-Toolkit 74

7.1 Übersicht ausgewählter RL-Tutorials auf der Plattform GitHub 75
7.2 Steckbrief zum Basic-RL-Tutorial 76
7.3 Steckbrief zum Practical-RL-Tutorial 77
7.4 Steckbrief zur den RL-Beispielen 78

7.5 Steckbrief zur den Beispielen von Sutton-&-Barto 79
7.6 Steckbrief zur den RLcode-Beispielen 80
7.7 Steckbrief zum RL-TF-Sammlung 81

8.1 Hardware-Komponenten 83
8.2 Kategorien, Merkmale und Gewichte 85
8.3 Punkteskala für die Bewertung der Alternativen 86
8.4 Scoring-Tabelle 1 zur Nutzwertanalyse 88
8.5 Scoring-Tabelle 2 zur Nutzwertanalyse 89
8.6 Scoring-Tabelle 3 zur Nutzwertanalyse 90
8.7 Ergebnis von Installation und Test der übrigen sieben Softwarelösungen . 97

9.1 Agenten der Softwarelösung Keras-rl 99
9.2 Ergebnisse der Simulationen mit den Agenten / Algorithmen für Cart Pole 106
9.3 Agenten der Softwarelösung OpenAI Lab 107
9.4 Argumente zum Kommandozeilenskript der Softwarelösung OpenAI Lab 108
9.5 Ergebnisse der Simulationen mit den Agenten / Algorithmen für Cart Pole 113
9.6 Agenten der Softwarelösung Ray RLlib 114
9.7 Argumente zum Kommandozeilenskript der Softwarelösung Ray RLlib . 115
9.8 Ergebnisse der Simulationen mit den Agenten / Algorithmen für Cart Pole 122

Abkürzungsverzeichnis

A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
AC	Actor Critic
ACER	Actor-Critic with Experience Replay
ACI	Artificial Collective Intelligence
ACKTR	Actor Critic using Kronecker-Factored Trust Region
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ALE	Arcade Learning Environment
API	Application Programming Interface
APG	Asynchronous Policy Gradient
ARTDP	Adaptive Real-Time Dynamic Programming
BC	Behavioral Cloning
BSD	Berkeley Software Distribution
BURLAP	Brown-UMBC Reinforcement Learning and Planning
BWL	Betriebswirtschaftslehre
C51	Categorical Deep Q-Network of 51-Atom Agent
CARLA	Car Learning to Act
CEM	Cross-Entropy Method
CDQN	Continuous Deep Q-Network
CLV	Customer Lifetime Value
CMAC	Cerebellar Model Articulation Controller
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSV	Comma Separated Values
D3PG	Distributed Deep Deterministic Policy Gradient
DART	Dynamic Animation and Robotics Toolkit

DC Distributed Computing

DDPG Deep Deterministic Policy Gradients

DDQN Double Deep Q-Network

DFKI Deutsches Forschungszentrum für Künstliche Intelligenz

DFP Direct Future Prediction

DL Deep Learning

DQFD Deep Q-Learning from Demonstration

DQN Deep Q-Network

DP Dynamische Programmierung

DPG Deterministic Policy Gradients

DPP Dynamic Policy Programming

DPS Direct Policy Search

DRL Deep Reinforcement Learning

DRQN Deep Recurrent Q-Network

EC2 Elastic Compute Cloud

ES Evolution Strategies

EWC Elastic Weight Consolidation

GAIL Generative Adversarial Imitation Learning

GLS Gleichungssystem

GPL GNU General Public License

GPS Guided Policy Search

GPU Graphics Processing Unit

GQ General Q-Learning

GTD Gradient Temporal Difference

HRA Hybrid Reward Architecture

HSE Higher School of Economics

HTTP Hypertext Transfer Protocol

INRIA Institut National de Recherche en Informatique et en Automatique

IDSIA Istituto Dalle Molle di studi sull'intelligenza artificiale

JDK Java Development Kit

JRE Java Runtime Environment

JSON JavaScript Object Notation

JVM Java Virtual Machine

KI Künstliche Intelligenz

KIT Karlsruher Institut für Technologie

KNN Künstliche Neuronale Netzwerke

LE Learning Environment

LTS Long Term Support

MC Monte Carlo

MBDPS Model-based Direct Policy Search

MDP Markov Decision Process

MIT Massachusetts Institute of Technology

MLP Multi-Layer Perceptron

MMC Mixed Monte Carlo

MMLF Maja Machine Learning Framework

MPI Message Passing Interface

MSE Mean Squared Error

MSI Micro-Star International

MuJoCo Multi-Joint dynamics with Contact

NAF Normalized Advantage Function

NEC Neural Episodic Control

NSQ N-step Q-Learning

ODE Open Development Environment

OpenGL Open Graphic Library

P3O Parallelized Proximal Policy Optimization

PAL Persistent Advantage Learning

PCL Path Consistency Learning

PER Prioritized Experience Replay

PG Policy Gradient

PGT Policy Gradient Theorem

PGN Policy Gradient Network

PLE PyGame Learning Environment

PNG Portable Network Graphics

POMDP Partially Observable Markov Decision Process

PPO Proximal Policy Optimization

PyPI Python Package Index

QMDP Q-Function Markov Decision Process

QR Quantile Regression

RBF Radial Basis Functions

REST Representational State Transfer

RL Reinforcement Learning

RLE Retro Learning Environment

RMSprop Root Mean Square Propagation

RNN Recurrent Neural Networks

SARSA State–Action–Reward–State–Action

SDK Software Development Kit

SDL Simple DirectMedia Layer

SGD Stochastic Gradient Descent

SNES Super Nintendo Entertainment System

SSH Secure Shell

SVG Stochastic Value Gradients

TD Temporal Difference

TDC Temporal Difference with Gradient Correction

TRPO Trust Region Policy Optimization

TU Technische Universität

UC University of California

UCB Upper Confidence Bounds

UCT UCB applied to Trees

UE4 Unreal Engine 4

UMBC University of Maryland, Baltimore County

VNC Virtual Network Computing

VPG Vanilla Policy Gradient

VWL Volkswirtschaftslehre

WD Western Digital

XML Extensible Markup Language

YSDA Yandex School of Data Analysis

1 Einleitung

Direkt nach der Geburt kann eine Antilope noch nicht einmal richtig stehen. Aber auch ohne die Hilfe ihre Mutter oder eines anderen Lehrers lernt sie in nur wenigen Stunden, sicher zu stehen, zu laufen und zu springen, wie ein YouTube-Video des *Wildlife* Foto- und Videographen Christof Schoeman sehr eindrucksvoll dokumentiert [Sch17]. Junge Antilopen oder Gazellen sind bereits kurz nach der Geburt in der Lage, sich mit über 30 km/h fortzubewegen. Das ist auch absolut notwendig, denn das einfache Ziel lautet: Überleben. Raubtiere, die auf Beutezug sind, haben ein Gespür dafür entwickelt, die schwächsten Tiere in einer Herde zu identifizieren. Die Jungtiere haben also nur eine Überlebenschance, wenn sie mit den anderen mithalten können. Wenn man sich das Video genauer ansieht, dann stellt man fest, dass die neugeborene Antilope nach dem Prinzip *Trial-and-Error* lernt. Sie macht einen Versuch aufzustehen und zu laufen. Dieser missling zwar, aber sie versucht es wieder und wieder und jedes Mal klappt es ein wenig besser. Denn auch ohne Lehrer bekommt sie eine Rückmeldung, ob der Versuch erfolgreich war oder nicht. Sie probiert zunächst einige Aktionen aus und folgt dann der Strategie, die erfolgsversprechend zu sein scheint. Diese Form von Lernen wird auch bestärkendes Lernen genannt, im Englischen spricht man von *Reinforcement Learning (RL)*.

Im Bereich der Künstliche Intelligenz (KI) beschäftigt sich das maschinelle Lernen damit, Lernvorgänge der Natur auf Maschinen zu übertragen. *Reinforcement Learning* kann bspw. dazu benutzt werden, dass Roboter laufen lernen. In den YouTube-Videos von Michel Tokic [Tok09] und Francis Wyffels [Wyf10] wird gezeigt, wie einfache, kleine Roboter mit Hilfe von RL das Krabbeln lernen. Hierzu benutzen die Roboter verschiedene Sensoren, um ihre Umwelt wahrzunehmen. Außerdem können die Roboter unterschiedliche Aktionen ausführen, je nach Konstruktion können verschiedene Servo-Motoren angesteuert werden.

Das Video von Martin Riedmiller der Universität Freiburg zeigt ein RL-Beispiel, in dem ein Roboterwagen, der sich in einer horizontalen Achse vor- und zurückbewegen kann, lernen muss, eine Stange, die in vertikaler Richtung um einen Drehpunkt am Wagen rotiert, aufrecht zu balancieren [Rie10]. Dieses Beispiel ist eine Erweiterung des bekannten *Cart Pole* Problems. In einem nicht ganz ernst gemeinten Experiment zeigen Petar Kormushev und Kollegen, wie ein Roboter lernt, Pfannkuchen in einer Pfanne zu wenden, indem diese so durch die Luft geschleudert werden, dass dabei ein Salto durchgeführt wird [KCC10]. Katharine Mülling und Kollegen von der Technischen Universität Darmstadt haben einem Roboter das Tischtennisspielen beigebracht [Mue+12]. Auch im Bereich des autonomen Fahrens wird *Reinforcement Learning* eingesetzt. Ein ebenfalls eher witziges Experiment zeigt ein Modellbau-Auto, welches eben nicht ferngesteuert wird, sondern sich autonom bewegt und Driften lernt [Lab15]. Diese Aufgabe trainiert es bis zu einer Perfektion, sodass das Driften bspw. auch auf sandigem Boden oder um ein anderes fahrendes Auto herum gelingt.

Reinforcement Learning muss nicht zwangsläufig nur auf Roboter und Maschinen angewendet werden, auch Computersysteme können damit trainiert werden. So hat das DeepMind-Team von Google wirkungsvoll demonstriert, wie ein Computersystem das Atarispiel

Breakout lernt [Pap15]. Anstelle der Sensoren bekommt das System die Bildschirmausgabe des Computerspiels als Pixelgrafik und den aktuellen Spielstand mitgeteilt. Einen Joystick braucht das System nicht zur Steuerung zu bedienen, denn die digitalen Signale, die der Joystick generieren würde, können auch direkt an das Spiel als Aktionen übertragen werden. Wesentlich bekannter geworden ist Google DeepMind durch das Spiel Go. Deren Computersystem AlphaGo, welches mit einer Kombination aus *Reinforcement Learning* und *Deep Learning* trainiert wurde, hat im März 2016 den südkoreanische Profispieler Lee Sedol mit 4:1 sensationell unter Turnierbedingungen geschlagen [Dee16a; Dee16c; Dee16e; Dee16g; Dee16i; Dee16b; Dee16d; Dee16f; Dee16h; Dee16j].

Das Konzept zu *Reinforcement Learning* lässt sich verallgemeinern und auf viele Probleme anwenden. Genaugenommen ist RL keine Methode, sondern es sind spezielle Arten von (Lern-)Problemen. Im Mittelpunkt dieser Probleme steht ein (Software-)Agent, der autonom mit seiner Umwelt interagiert, um ein bestimmtes, vorgegebenes Ziel zu erreichen. Der Agent macht Beobachtungen, trifft eigene Entscheidungen und führt Aktionen aus, die seinen Umweltzustand ändern. Er bekommt Rückmeldungen in Form von Belohnungen (engl. *Rewards*) und versucht eine Strategie bzw. Politik (engl. *Policy*) zu finden, um das vorgegebene Ziel zu erreichen. Er kann Erfahrungen sammeln (*Exploration* oder das bislang gelernte Wissen direkt anwenden, um kurzfristig Belohnungen zu erhalten (*Exploitation*). Um das langfristige Ziel zu erreichen und einen möglichst hohen *Return* zu bekommen, muss eine Kombination aus *Exploration* und *Exploitation* eingesetzt werden.

Mathematisch können viele RL-Lernprobleme durch ein Markov Entscheidungsprozess (engl. *Markov Decision Process (MDP)*) beschrieben werden, bei dem die Bellman-Gleichung der Ausgangspunkt bei der Entwicklung von Lernalgorithmen ist. Dabei werden Zustände über spezielle Q-Wertefunktionen bewertet und eine Strategie (*Policy*) gesucht, möglichst optimale Zustände zu besetzen. Bekannte Algorithmen sind bspw. Q-Learning und SARSA. In sehr großen Zustandsräumen werden die Wertefunktionen geschätzt bzw. durch Funktionen approximiert. Künstliche Neuronale Netzwerke (KNN) sind ebenfalls im Bereich Maschinenlernen der KI sehr populär und können u.a. zur Funktionsapproximation verwendet werden. Mittlerweile lassen sich auch sehr tiefe KNN erfolgreich trainieren und man spricht von *Deep Learning*. Dieser Durchbruch hat auch dem *Reinforcement Learning* einen weiteren Schub gegeben und es sind bspw. Algorithmen wie Deep Q-Network (DQN) oder DeepSARSA entwickelt worden.

Das Ziel dieser Arbeit ist es, einerseits die theoretischen Grundlagen zum *Reinforcement Learning* zu beschreiben und andererseits eine geeignete Softwarelösung zu finden, die in Forschung und Lehre an der Fakultät für Wirtschaftswissenschaften an der Hochschule für Technik und Wirtschaft des Saarlandes (htw saar) eingesetzt werden kann.

Die vorliegende Arbeit ist in zehn Kapiteln gegliedert. Nach der Einleitung werden in Kapitel 2 die theoretischen Grundlagen zum Thema *Reinforcement Learning* vorgestellt. In Kap. 3 wird beschrieben, nach welchen Kriterien die Softwarelösungen ausgewählt werden, die näher betrachtet werden sollen. Es sind insgesamt 37 Softwarelösungen, die in den nachfolgenden Kapiteln kurz präsentiert werden. Diese lassen sich in mehrere Kategorien einteilen: Umgebungen (Kap. 4), Bibliotheken und Frameworks (Kap. 5), Spezialsoftware (Kap. 6) und Tutorials (Kap. 7). Nicht alle diese Softwarelösungen können ausführlich getestet werden. Aus diesem Grund wird zunächst eine Nutzwertanalyse in Kap. 8 durchgeführt, um die Lösungen zu bewerten und miteinander zu vergleichen. Auf Basis dieser Analyse werden ausgewählte Lösungen auf ein Testsystem installiert

und getestet. In Kap. 9 werden dann mit diesen Lösungen Experimente zu typischen RL-Problemen durchgeführt. Anhand dieser Erfahrungen wird dann ein Fazit gezogen und mind. eine geeignete RL-Software ausgewählt. Im letzten Kapitel wird die Arbeit zusammengefasst und ein kurzer Ausblick gegeben.

2 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen zum Thema *Reinforcement Learning* gelegt. Zunächst wird allgemein auf das Thema Lernen und die unterschiedlichen Formen eingegangen. Anschließend werden klassische Problemstellungen des *Reinforcement Learning* beschrieben. Dann werden die wichtigsten Begriffe inklusive der mathematischen Notationen zum *Reinforcement Learning* vorgestellt. Viele RL-Probleme lassen sich mathematisch mit dem *Markov Decision Process (MDP)* beschreiben, wobei die Bellmann-Gleichung der Ausgangspunkt für viele Lernalgorithmen darstellt. Die Ideen hinter den wichtigsten Lösungsansätzen werden kurz erklärt und das damit verbundene Explorations-Exploitations-Dilemma erläutert. Abschließend werden die bekanntesten RL-Algorithmen präsentiert.

2.1 Lernen

Nach dem Duden versteht man unter Lernen u.a. das sich Aneignen von Wissen und Kenntnisse, das sich Einprägen dieser in seinem Gedächtnis sowie das Erwerben von Fertigkeiten [Dud17]. Sehr viel genauer definiert Gablers Wirtschaftslexikon [BM17] Lernen als

> *« absichtlicher (intentionales Lernen), beiläufiger (inzidentelles und implizites Lernen), individueller oder kollektiver Erwerb von geistigen, körperlichen und sozialen Kenntnissen und Fertigkeiten. Aus lernpsychologischer Sicht wird Lernen als ein Prozess der relativ stabilen Veränderung des Verhaltens, Denkens oder Fühlens (verarbeiteter Wahrnehmung der Umwelt oder Bewusstwerdung eigener Regungen) aufgefasst (Qualifikation). »*

Lernen ist die Grundvoraussetzung dafür, sich der Umwelt anpassen zu können und ggf. diese durch gezielte Aktionen zu verändern. Beim sogenannten **maschinellen Lernen** lernt nun ein künstliches System, d.h. eine Maschine bzw. ein Computerprogramm, indem es Wissen aus Erfahrungen generiert. Dabei unterscheidet man verschiedene Lernformen bzw. Lernprobleme, die im Folgenden kurz vorgestellt werden.

Überwachtes Lernen Überwachtes Lernen (engl. *Supervised Learning*) verwendet Expertenwissen, welches durch das System nachgebildet werden soll. D.h. zu jedem Zeitpunkt während des Lernprozesses ist vollständig bekannt, wie die korrekte Lösung aussieht. Diese wird dem System in jedem Lernschritt mitgeteilt, damit der Lernalgorithmus die Systemparameter entsprechend anpassen kann, um der korrekten Lösung näher zu kommen. Eine Herausforderung in dieser Art von Lernverfahren ist das Übertrainieren (engl. *Overfitting*). Hierbei werden die präsentierten Daten nur auswendig gelernt, das System verfügt aber nicht über generelles Wissen, welches dann in neuartigen Situation angewendet werden kann. In diesen Situationen versagt dann ein solches übertrainiertes System. Im Bereich *Data Mining* werden diese Lernformen für Aufgaben der Regression und Klassifikation verwendet.

Unüberwachtes Lernen Unüberwachtes Lernen (engl. *Unsupervised Learning*) bezeichnet Lernformen, in denen weder die Zielwerte bekannt sind (s.o.) noch eine Belohnung durch die Umwelt (s.u.) erfolgt. Der Lernalgorithmus versucht hier, verborgene Muster in den Daten zu finden. Im Bereich *Data Mining* werden diese Lernformen für Aufgaben der Segmentierung eingesetzt, z.B. um Clusteranalysen durchzuführen.

Bestärkendes Lernen Das bestärkende Lernen (engl. *Reinforcement Learning*) unterscheidet sich deutlich von den anderen beiden Lernformen. Es wird nicht durch bestimmte Lernmethoden definiert, sondern durch eine Klasse von Lernprobleme charakterisiert. Dabei interagiert ein Agent (Lerner bzw. Lernalgorithmus) immer mit seiner Umwelt, um ein bestimmtes Ziel zu erreichen. Der Agent nimmt die Umwelt war und kann Aktionen ausführen, um diese zu verändern. Er trifft selbstständig Entscheidungen, indem er in einem bestimmten Zustand eine bestimmte Aktion auswählt. Ihm wird aber nicht vorgegeben, welches die jeweilige beste Aktion in der jeweiligen Situation wäre (*Supervised Learning*). Stattdessen bekommt der Agent nach der Durchführung seiner Aktion eine Rückmeldung in Form einer Belohnung, die ggf. auch negativ sein kann, falls es eine schlechte Aktion war. Der Agent versucht nun mit Hilfe dieser sogenannten *Rewards* eine Strategie zu finden, um das vorgegebene Ziel zu erreichen. Dabei kann er nach dem *Trial-and-Error*-Prinzip vorgehen, um Erfahrungen zu sammeln (*Exploration*). Oder der Agent verwendet die bislang gemachten Erfahrungen und wendet dieses Wissen direkt an, um kurzfristig Belohnungen zu bekommen (*Exploitation*). Strategien, die nur auf Exploration oder nur auf Exploitation basieren, führen zu suboptimalen Lösungen (*Exploration-Exploitation-Dilemma*). Eine Herausforderung in RL-Problemen besteht also darin, die richtige Balance zwischen Exporation und Exploitation zu finden.

2.2 Reinforcement Learning

Ausgangspunkt ist ein System, in dem ein Agent mit seiner Umwelt interagiert. Schematisch ist diese Wechselwirkung in Abb. 2.1 dargestellt.

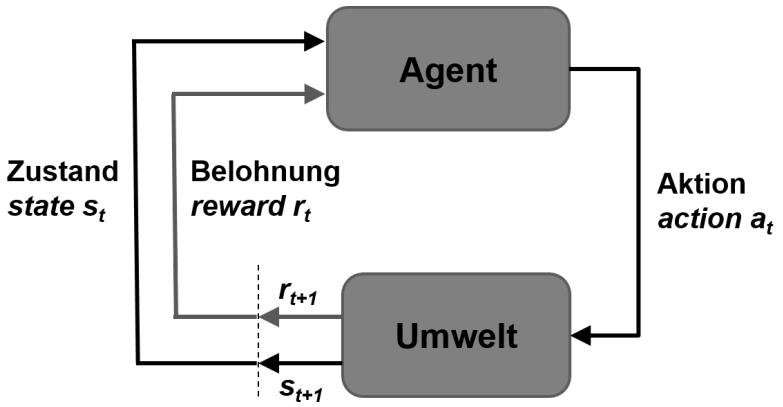


Abbildung 2.1: Agenten-Umwelt-Interaktion des RL – in Anlehnung an [SB98]

Zu jedem Zeitpunkt t mit $t = 0, 1, 2, ..., T$ nimmt der Agent den **Zustand** (engl. *State*) s_t aus einer endlichen Menge von möglichen Zuständen $\mathcal{S}$ wahr ($s_t \in S$). Anhand dieser Wahrnehmung wählt der Agent eine **Aktion** (engl. *Action*) a_t und führt diese aus. Die Menge der möglichen Aktionen $\mathcal{A}(s_t)$ ist ebenfalls endlich und es gilt: $a_t \in \mathcal{A}(s_t)$. Zum nächsten Zeitpunkt $t + 1$ erhält der Agent dann seine Belohnung (engl. *Reward*) als **Reward** $r_{t+1} \in \mathbb{R}$. Die Umwelt befindet sich dann im Folgezustand s_{t+1}. Der Reward gibt an,

wie zielführend die Aktion war. Ein positiver Wert entspricht einer Belohnung, ein negativer Wert einer Bestrafung. Lernen bedeutet in diesem Kontext mit Hilfe der Rewards gute Aktionen zu bestärken und schlechte zu vermeiden. Die Abbildung von Zuständen $\mathcal{S}$ auf Aktionen $\mathcal{A}$ wird auch als **Strategie** oder **Politik** (engl. *Policy*) bezeichnet: $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Die Werte von $\pi(s, a)$ sind Wahrscheinlichkeiten für das Auswählen von a im Zustand s, wobei gilt:

$$\sum_{a \in \mathcal{A}(s)} \pi(s, a) = 1. \tag{2.1}$$

Das Ziel ist es, die Werte

$$\pi_t(s, a) = Pr(a_t = a \,|\, s_t = s) \tag{2.2}$$

zu optimieren, sodass Aktionen, die zu einem guten Reward führen, durch eine hohe Wahrscheinlichkeit gekennzeichnet sind und deren Auswahl zukünftig bestärkt wird, während schlechte Aktionen mittels niedriger Wahrscheinlichkeit zukünftig vermieden werden sollen. Werden alle Rewards kummuliert, d.h. aufsummiert, so spricht man auch vom **Return**:

$$R_t = r_{t+1} + r_{t+2} + ... + r_T. \tag{2.3}$$

Dabei gibt T eine zeitliche Grenze, d.h. einen Horizont, für das sogenannte **episodische** Lernproblem an. In einem **kontinuierlichen** Lernproblem gibt es dagegen keine solche Grenze, d.h. $T \rightarrow \infty$. Der Return wird hier berechnet, indem zukünftige Rewards diskontiert werden, ähnlich also wie bei der Berechnung des Barwertes in der Finanzmathematik. Es gilt:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.4}$$

wobei der Parameter $\gamma \in [0, 1]$ den Diskontierungsfaktor angibt. Eine gemeinsame Notation für beide Formen von Lernproblemen wird durch die folgende Gleichung beschrieben

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}, \tag{2.5}$$

wobei $T \rightarrow \infty$ (kontinuierlich) und $\gamma = 1$ (episodisch) möglich sind, aber nicht gemeinsam auftreten dürfen.

2.3 Problemstellungen

Ein diesem Abschnitt werden einige Problemstellungen kurz vorgestellt, die zu den typischen *Reinforcement Learning* Aufgaben gehören bzw. die sich mit Methoden und Algorithmen des RL lösen lassen.

Cart Pole In der Problemstellung *Cart Pole* ist ein inverses Pendel als Stange (engl. *Pole*) auf einem Wagen (engl. *Cart*) montiert (vgl. Abb. 2.2). Die Drehachse des Pendels der Länge l und Masse m liegt auf der Oberseite des Wagens der Masse M. Der Winkel θ zwischen dem Pendel und der vertikalen Achse wird fortlaufend gemessen. Das Ziel ist es, das Pendel in eine stabile Lage, d.h. aufrechte Position bei $\theta = 0$, zu bringen. Hierzu

kann der Wagen in der horizontalen Achse mit der Kraft $\vec{F}$ bewegt werden. Mit Hilfe der Newtonschen Mechanik lässt sich dieses Problem mathematisch beschreiben und lösen. Es ist auch eine Standardaufgabe der Regelungstechnik. Eine prominente Anwendung ist in der Raumfahrt zu finden. Die wiederverwendbare Trägerrakete Grasshopper des US-amerikanischen Unternehmens SpaceX nutzt das Prinzip, um aufrecht starten und auch wieder landen zu können.

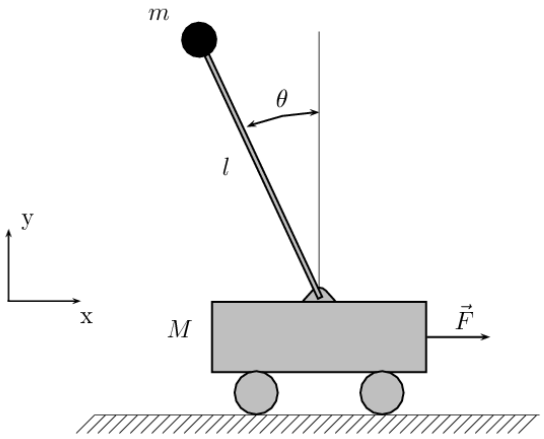


Abbildung 2.2: Schematische Darstellung eines inversen Pendels auf einem Wagen [Wik18i]

Mountain Car Der Fahrer eines schwach motorisierten Wagens (engl. *Car*) sitzt zwischen zwei Hügeln bzw. Bergen (engl. *Mountain*) in einer Mulde bzw. im Tal fest (vgl. Abb. 2.3). Das Ziel ist es, das Fahrzeug hinter die Zielflagge auf dem rechten Hügel zu bringen. Da der Motor aber zu schwach ist, um dieses Ziel auf direktem Weg zu erreichen, muss Schwung geholt werden. Hierzu nutzt der Wagen den anderen Hügel als Anlauf, in dem er also zunächst ein Stück rückwärts auf diesen fährt. Physikalisch findet dadurch ständig eine Umwandlung von potenzielle Energie in Bewegungsenergie und umgekehrt statt, die gegen die bremsende Schwerkraft wirkt. Zu diesem Problem existiert keine analytisch bekannte Lösung.

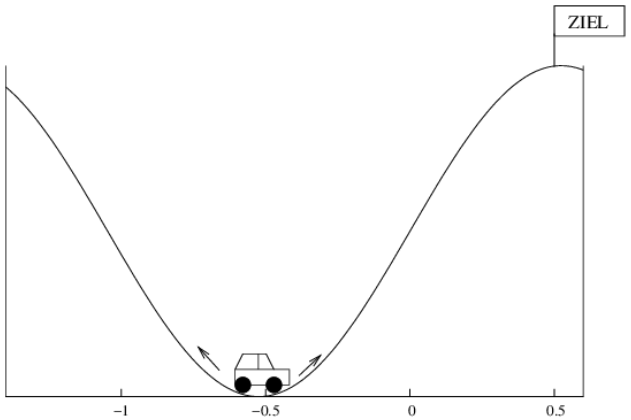


Abbildung 2.3: Schematische Darstellung des Mountain Car Problems [Rö09]

Es ist ein Standardproblem des *Reinforcement Learning*, u.a. auch weil es sich gut visualisieren lässt. Gemessen werden in jedem Zeitschritt die beiden kontinuierlichen Zustandsgrößen Position x_t und Geschwindigkeit v_t des Wagens. Drei diskrete Aktionen sind möglich:

Vorwärts beschleunigen ($a_t = +1$), rückwärts beschleunigen ($a_t = -1$) oder nichts tun ($a_t = 0$). Das System durchläuft nun eine vorher festgelegte Anzahl von Episoden. Eine Episode ist beendet, wenn der Wagen das Ziel erreicht hat (positiver Reward) oder wenn eine vorgegebene maximale Anzahl von Aktionen gemacht wurden (negativer Reward). Der Agent hat also keine Informationen bezüglich des Ziels auf dem rechten Hügel bis zu einem ersten Erfolg. Dieser verzögerte Reward macht das Lernen sehr schwierig. Das übergeordnete Lernziel des Agenten ist es, eine Strategie zu finden, mit der der Wagen in möglichst wenigen Schritten das Ziel erreicht.

Gridworld Diese Probleme basieren auf einer Welt (engl. *World*), die sich durch ein Gitter (engl. *Grid*) beschreiben lassen. In diesen Umgebungen gibt es einen Startfeld S, auf dem der Agent zu Beginn steht, und ein Zielfeld G (engl. *Goal*), welches er erreichen möchte. Durch einfache Spielzüge kann der Agent von einem Feld zu einem anderen gelangen, wobei vorher definiert werden muss, welche Züge erlaubt sind, bspw. nur in horizontaler und vertikaler aber nicht in diagonaler Richtung (vgl. Abb. 2.4).

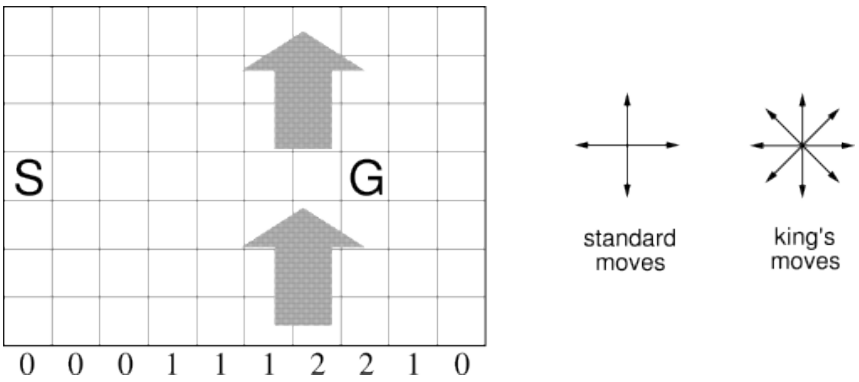


Abbildung 2.4: Gridworld [SB18]

Windy Gridworld ist bspw. eine Erweiterung. Hier kommt aus einer Richtung ein mehr oder weniger starker Wind, der dazu führt, dass der Agent gegen seinen Willen um mehrere Felder versetzt wird. In Abb. 2.4 geben die Zahlen unterhalb der Spalten jeweils die Windstärke an und entsprechen den Feldern, um die der Agent in jedem Spielzug versetzt wird, wenn er sich in dieser Spalte befindet. Dieses Problem kann auch derart modifiziert werden, dass der Wind zufällig weht. Eine andere Erweiterung ist *Cliff Walking*. In dieser Welt gibt es einen speziellen Bereich, *The Cliff*, in dem der Agent abstürzen würde. Der Reward ist hier -100 und der Agent wird dann wieder auf das Startfeld gestellt, wenn ein solches Feld betreten wird (vgl. Abb. 2.5).

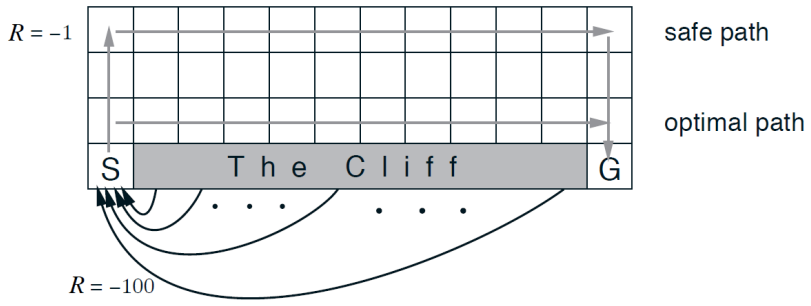


Abbildung 2.5: Cliff Walking Problem [SB18]

Ein weiteres Problem aus dem Bereich Gridworld ist *Frozen Lake*, zu deutsch: zugefrorener See. In dieser Welt gibt es spezielle Felder *F*, die gefroren (engl. *frozen*) sind und sicher vom Agenten betreten werden können. Es gibt aber auch gefährliche Löcher *H* (*engl. Hole*), in die er versinkt und ertrinkt (vgl. Abb. 2.6). Optional kann auch hier wieder ein Wind wehen.

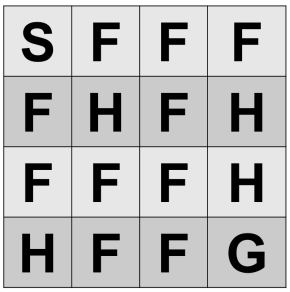


Abbildung 2.6: Einfache Frozen Lake Umgebung als 4 x 4 Gitter

Auch ein Labyrinth (engl. *Maze*) stellt ein Problem aus dem Bereich Gridworld dar. In dem Labyrinth gibt es zusätzliche Wände, die verhindern, dass sich der Agent frei in alle Richtungen bewegen kann. Das Ziel des Agenten ist es nun, möglichst schnell einen Ausweg aus diesem Labyrinth zu finden.

Computerspiele In jüngster Zeit werden auch Computerspiele als Lernumgebung für *Reinforcement Learning* Probleme verwendet. Ein Agent übernimmt dabei praktisch die Aufgabe eines menschlichen Spielers. Die Steuerung durch Tastatur, Maus, Joystick oder Controller lässt sich relativ leicht in Eingabesignale übersetzen. Der Reward in Form der aktuellen Punktezahl kann aus dem Spiel extrahiert werden, genauso wie die Informationen, falls ein Level erfolgreich abgeschlossen wurde oder das Spiel vorzeitig beendet ist (*Game over*). Anders sieht es allerdings mit den grafischen Informationen aus. Computerspiele haben je nach Erscheinungsjahr und Art eine mehr oder wenig aufwändige Grafik, ggf. sogar 3D-Grafik. Letztendlich lässt sich jede Grafik als Matrix aus Pixeln darstellen, wobei jeder Pixel einem Bildpunkt entspricht und man den zugehörigen Farbwert speichert. Bspw. liefert das Atari 2600 System aus den späten 1970er Jahren Bilder mit 160 x 210 Pixeln. Das sind also bereits 33.600 Informationen pro Bild. Der Agent muss also sehr viel mehr Informationen als in den anderen Aufgaben (s.o.) verarbeiten. Hierbei können neue Techniken wie bspw. *Deep Learning* unterstützen. Man spricht dann auch von *Deep Reinforcement Learning*.

2.4 Markovsche Entscheidungsprozesse

Falls ein Agent Informationen von der Umwelt erhält, in welchem Zustand er sich aktuell befindet, jedoch keinerlei Informationen, wie er in diesen Zustand gekommen ist, dann hat er kein Gedächtnis. Man bezeichnet dies auch als *Markov-Eigenschaft*. Die Auswahlwahrscheinlichkeit für den Folgezustand s' und den Reward r zum Zeitpunkt $t+1$ hängt dann nur vom Ausgangszustand s und der gewählten Aktion a zum Zeitpunkt t ab:

$$Pr\left\{s_{t+1} = s', r_{t+1} = r \,\middle|\, s_t, a_t\right\}. \tag{2.6}$$

Erfüllt das Lernproblem diese Markov-Eigenschaft, nennt man es Markov Entscheidungsprozess (engl. *Markov Decision Process (MDP)*). Sind der Zustands- und Aktionsraum endlich, dann gilt für beliebige Zustands-Aktions-Paare:

$$\mathcal{P}^a_{ss'} = Pr\left\{s_{t+1} = s' \,\middle|\, s_t = s, a_t = a\right\}, \tag{2.7}$$

d.h. die Wahrscheinlichkeit für den Folgezustand s' hängt nur vom Zustand s und der gewählten Aktion a ab. Der Erwartungswert des Rewards

$$\mathcal{R}^a_{ss'} = E\left\{r_{t+1} \,\middle|\, s_t = s, a_t = a, s_{t+1} = s'\right\} \tag{2.8}$$

ist dageben abhängig vom Zustand s, der Aktion a und dem Folgezustand s'. Im Reward spiegelt sich auch immer eine sehr kurzfristige Sichtweise wieder, denn es ist die Belohnung für eine durchgeführte Aktion des Agenten. Langfristig soll aber die Summe der diskontierten Rewards, also der Return R, maximiert werden. Deshalb wird noch der Begriff der Wertefunktion V (engl. *Value*) eingeführt. Diese besagt, wie wertvoll es für einen Agenten bei Befolgung der Strategie π ist, in einem bestimmten Zustand s zu sein:

$$V^\pi(s) = E_\pi\left\{R_t \,|\, s_t = s\right\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s\right\}, \tag{2.9}$$

wobei E_π angibt, dass es sich hierbei wieder um einen Erwartungswert handelt. Betrachtet man nun Zustands-Aktions-Paare, dann kann eine entsprechende Funktion

$$Q^\pi(s,a) = E_\pi\left\{R_t \,|\, s_t = s, a_t = a\right\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right\} \tag{2.10}$$

angegeben werden. Die Wertefunktionen V^π und Q^π werden im *Reinforcement Learning* durch Interaktion des Agenten mit seiner Umgebung geschätzt. Somit ist eine wichtige Komponente aller RL-Algorithmen diese Werte möglichst effizient zu schätzen. Die Wertefunktion hat eine besondere Eigenschaft: Sie ist rekursiv bezüglich der Zustände und es gilt:

$$V^\pi(s) = E_\pi\left\{R_t \,|\, s_t = s\right\} = \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]. \tag{2.11}$$

Gleichung 2.11 wird auch Bellman-Gleichung genannt und ist der Ausgangspunkt von vielen Lernalgorithmen. Ziel ist somit eine optimale Strategie π^* zu finden, welche die Wertefunktionen

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.12}$$

und

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) \tag{2.13}$$

maximiert. Es folgt daraus die Bellmansche Optimalitätsgleichung:

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right] \tag{2.14}$$

bzw.

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma \max_{a' \in \mathcal{A}(s)} Q^*(s',a')\right] \tag{2.15}$$

Falls also die optimale Wertefunktion gefunden ist, kann der Agent einer optimalen Strategie derart folgen, dass er in jedem Zustand immer die Aktion mit dem höchsten Q-Wert auswählt.

2.5 Lösungsansätze

Ausgangspunkt ist die Bellmansche Optimalitätsgleichung (2.14). Bei gegebenen Mengen $\mathcal{P}$ und $\mathcal{R}$ entspricht diese Gleichung einem linearen Gleichungssystem (GLS), das sich theoretisch mit algebraischen Methoden lösen lässt. Die Dimension N dieses Gleichungssystems entspricht der Anzahl der möglichen Zustände, also $|\mathcal{S}|$. Zur Lösung des GLS müssen Matrizen der Dimension N invertiert werden. Diese Operation ist etwa von der Ordnung N^3, d.h. für große Zustandsräume also sehr aufwendig.

Somit werden in der Praxis Näherungsverfahren verwendet, um die Gleichung zu lösen. Dabei kommen häufig zwei Operationen zum Einsatz: **Politik-Evaluation** (engl. *Policy Evaluation*) und **Politik-Verbesserung** (engl. *Policy Improvement*). Bei der Politik-Evaluation wird die Wertefunktion der aktuellen Politik V^π möglichst gut angenähert. Anschließend wird bei der Politik-Verbesserung auf Basis dieser Wertefunktion die Politik π verbessert. Danach wird dann wieder die Wertefunktion der neuen Politik geschätzt usw. Es handelt sich also um ein iteratives Verfahren, auch *Policy Iteration* genannt, welches in Abb. 2.7 skizziert ist.

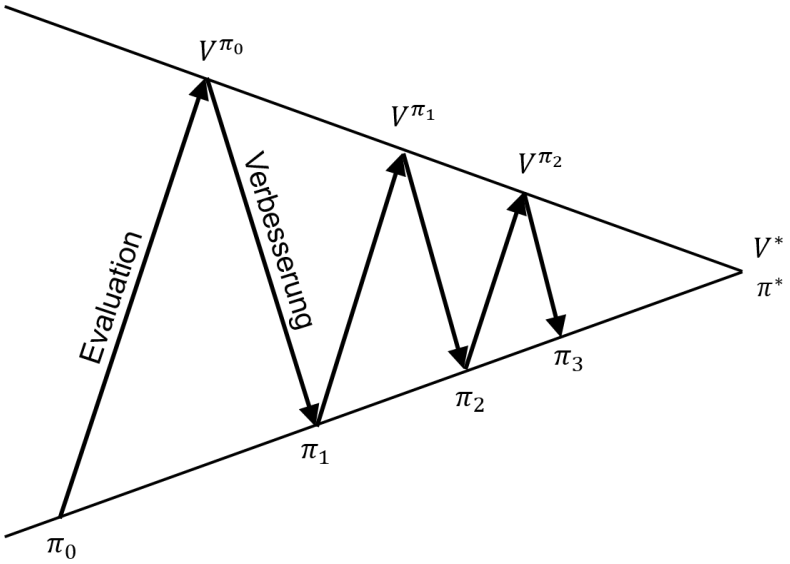


Abbildung 2.7: Politik-Iteration (engl. *Policy Iteration*) – in Anlehnung an [SB98]

Bei der *Policy Evaluation* handelt es sich auch um ein Problem der Vorhersage (engl. *Prediction*), denn die Wertefunktion V^π soll bei einer gegebenen Politik π möglicht gut abgeschätzt werden. *Policy Improvement* ist dagegen ein Optimierungs- bzw. Steuerungsproblem (engl. *Control Problem*), hier soll eine möglichst optimale Politik gefunden werden.

Man unterscheidet auch zwischen *On-Policy* und *Off-Policy* Verfahren. *On-Policy* Verfahren bewerten die Politik, die sie aktuell verfolgen. *Off-Policy* Verfahren verfolgen im Gegensatz dazu eine andere Politik als diejenige, die sie bewerten. Sie versuchen normalerweise die optimale Wertefunktion und die optimale Politik direkt zu approximieren.

Verschiedene Lösungsansätze für *Reinforcement Learning* Probleme unterscheiden sich größtenteils darin, je nachdem, wie viel über die Umwelt bekannt ist, wie die aktuelle Wertefunktion geschätzt wird und wie die Iterationsschritte Politik-Evaluation und Politik-Verbesserung miteinander kombiniert werden.

Dynamische Programmierung Wenn das vollständige Modell der Umgebung bekannt ist, dann sind die beiden Mengen $\mathcal{P}$ und $\mathcal{R}$ gegeben. Es lassen sich nun Verfahren aus dem Bereich der *Dynamischen Programmierung (DP)* anwenden, um die Bellmansche Optimalitätsgleichung zu lösen. Dabei muss der Agent nicht mit der Umwelt interagieren, sondern auf Grundlage der gegebenen Wahrscheinlichkeiten und Erwartungswerte der Rewards lassen sich die optimale Wertefunktion und Politik iterativ annähern. Im k-ten Iterationsschritt gilt:

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right] \tag{2.16}$$

Die neue Schätzung $V_{k+1}(s)$ für einen Zustand s wird also der aktuellen Schätzung $V_k(s')$ für den Nachfolgezustand s' berechnet. Allgemein nennt man ein Vorgehen, bei dem eine Schätzung durch Verwendung einer vorherigen Schätzung verbessert wird, auch *Bootstrapping*. Man bricht den Iterationsprozess ab, wenn die Änderungen sehr klein werden. Die aktuelle Wertefunktion ist dann der optimalen Lösung V^* angenähert. Daraus kann dann wiederum die optimale Politik π^* abgeleitet werden (*Policy Evaluation*).

Es kann aber auch ein Schritt zur Politik-Verbesserung (*Policy Improvement*) durchgeführt werden, um eine möglicherweise bessere Politik zu bekommen. Hierzu wird nicht mehr der bisherigen Politik π gefolgt, sondern gierig (engl. *greedy*) Aktion a ausgewählt, die den Reward maximiert:

$$\pi'(s) = \arg\max_a Q^\pi(s,a) = \arg_a \max \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right] \tag{2.17}$$

Daraus kann dann wiederum die zugehörige Wertefunktion bestimmt werden:

$$V^{\pi'}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi'}(s')\right] \tag{2.18}$$

Diese beiden Schritte können nun interaktiv im Wechsel als Politik-Iteration (*Policy Iteration*) durchgeführt werden (vgl. Abb. 2.7). Ein Nachteil ist dabei allerdings, dass in jedem Iterationsschritt eine komplette Politik-Evaluation durchgeführt werden muss, die ihrerseits ein mehrmaliges Durchlaufen über alle Zustände des Zustandsraum erfordert.

Ein wichtiger Spezialfall ist die Wert-Iteration (*Value Iteration*), bei dem die Politik-Evaluation bereits nach einem Durchlauf abgebrochen wird.

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right] \tag{2.19}$$

Monte-Carlo-Methoden Liegt kein vollständiges Modell der Umgebung vor, d.h. die Übergangswahrscheinlichkeiten und Erwartungswerte der Rewards sind unbekannt, dann müssen sie aus den Erfahrungen geschätzt werden. Hierzu können *Monte Carlo* (MC)-Methoden benutzt werden, allerdings werden Erfahrungen in Form von Beispielen (Folge von Zuständen, Aktionen und Rewards) benötigt. Somit muss der Agent mit seiner Umgebung interagieren, um diese Erfahrungen zu sammeln. Für die vorhandenen Beispiele werden dann die Mittelwerte des Returns berechnet. Das ist aber immer erst nach einer Lernepisode möglich. Bei genügend vielen Beispielen wird der Mittelwert dann gegen den Erwartungswert konvergieren. Somit können diese Verfahren nur bei diskreten (episodischen) nicht bei kontinuierlichen Lernproblemen eingesetzt werden (vgl. Abschnitt 2.2). Des Weiteren sind MC-Methoden keine Bootstrapping-Methoden. Auch bei den MC-Verfahren lassen sich *Policy Evaluation* und *Policy Improvement* iterativ miteinander verbinden.

Temporal Difference Learning Ein Nachteil der MC-Methoden ist, dass erst nach dem Ende einer Episode der nächste Lernschritt vollzogen werden kann. *Temporal Difference (TD)* Methoden kombinieren nun den Bootstrapping-Ansatz der Dynamischen Programmierung mit dem modellfreien Vorgehen der Monte-Carlo-Methoden. Einerseits lassen sich TD-Methoden somit in Umgebungen einsetzen, in denen die Übergangswahrscheinlichkeiten und Erwartungswerte der Belohnungen unbekannt sind, andererseits können gesammelte Erfahrungen bzw. gemachte Beobachtungen innerhalb einer Episode sofort im weiteren Vorgehen benutzt werden.

Die einfachste TD-Methode, TD(0) genannt, wird beschrieben durch:

$$V(s_t) \leftarrow V(s_t) + \alpha \left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right) = V(s_t) + \alpha \cdot \delta_t \tag{2.20}$$

Die Wertefunktion $V(s_t)$ wird hier direkt aktualisiert, wenn der nächste Zustand s_{t+1} bekannt ist, der nächste Reward r_{t+1} beobachtet und die Wertefunktion $V(s_{t+1})$ geschätzt wurden. Der Wert eines Zustands wird somit aus der Summe aus dem nächsten beobachteten Reward und der diskontierten geschätzten Wertefunktion angenähert, mit γ als Diskontierungsfaktor. δ_t nennt man auch TD-Fehler. Die Lernrate $\alpha \in (0, 1]$ bestimmt, wie stark die Beobachtung in die neue Schätzung einfließt.

Die TD(0)-Methode benutzt nur den nächsten Reward, sie kann somit auch als 1-Schritt-Methode bezeichnet werden. Wenn nun nicht nur der nächste, sondern auch die n folgenden Rewards in der neuen Schätzung berücksichtigt werden sollen, kommt man zur n-Schritt-TD-Vorhersage (engl. *n-Step TD Prediction*):

$$\Delta V_t(s_t) = \alpha \left[R_t^{(n)} - V_t(s_t)\right] \tag{2.21}$$

mit

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{n-1} r_{t+n} + \gamma^{n-1} V_t(s_{t+n}) \tag{2.22}$$

$\Delta V_t(s_t)$ ist der Wert, mit dem die Wertefunktion $V_t(s_t)$ addiert wird, um eine Aktualisierung (engl. *Update*) vorzunehmen. $R_t^{(n)}$ ist der n-Step-Return zur Zeit t.

Um nun eine Brücke zu den Monte Carlo Methoden zu bauen, kann das Konzept der sogenannten *Eligibility Traces*, zu Deutsch: geeignete Spuren, benutzt werden. Dabei werden Rewards, die weiter in der Zukunft liegen mit dem Parameter $\lambda \in [0, 1]$ gemäß eines exponentiellen Zerfalls gewichtet. Für das als TD(λ) bekannte Verfahren gilt:

$$\Delta V_t(s_t) = \alpha \left[R_t^{\lambda} - V_t(s_t)\right] \tag{2.23}$$

mit

$$R_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} \tag{2.24}$$

Der erste Grenzfall $\lambda = 0$ führt wieder zu TD(0), also dem klassischen 1-Schritt-Verfahren. Der andere Grenzfall mit $\lambda = 1$, also TD(1), entspricht wiederum dem Vorgehen von Monte-Carlo-Methoden (s.o.).

Actor Critic Methoden Diese Methoden gehören zu den TD-Methoden. Der Agent enthält hier aber zwei neue interne Komponenten: *Actor* und *Critic* (vgl. Abb. 2.8). Die Politik π wird hier unabhängig von der Wertefunktion im *Actor* gespeichert, während der *Critic* die Wertefunktion V^π lernt, auf Basis der vom *Actor* momentan verfolgten Politik. Die Komponente *Critic* bewertet die vom *Actor* ausgeführten Aktionen a, während der *Actor* wiederum dieses Feedback benutzt, um seine Politik zu verbessern.

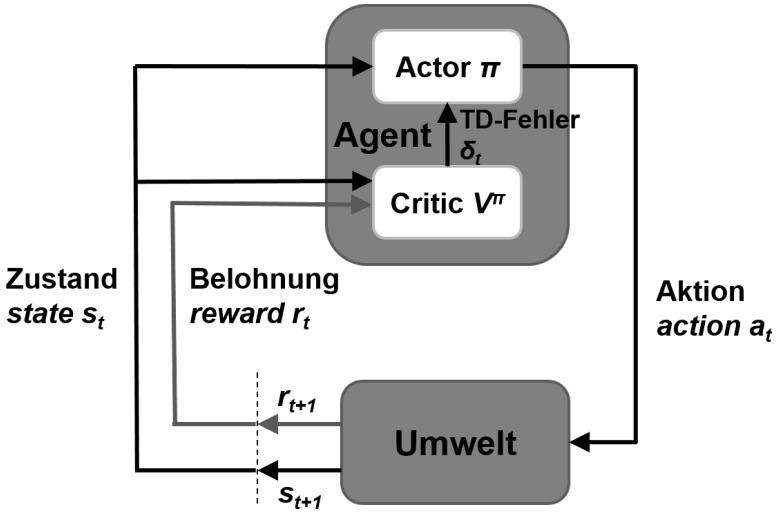


Abbildung 2.8: Schematische Darstellung der Actor Critic Architektur

Die Politik kann auf unterschiedliche Art vom *Actor* gespeichert werden, u.a. auch durch Künstliche Neuronale Netzwerke (KNN). Der *Critic* bewertet die vom *Actor* ausgewählte Aktion in Form des TD-Fehlers:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.25}$$

Funktionsapproximation In überschaubaren RL-Problemen können die Funktionen V und Q als Tabellen gespeichert werden. Im Allgemeinen sind die Zustandsräume und die Zahl der möglichen Aktionen jedoch sehr groß. Deshalb besteht enormer Speicherbedarf. Andererseits liegen meist gar nicht genügend Beispieldaten vor. Somit werden die Funktionen V und Q oft approximiert. Hierzu sind verschiedene Verfahren bekannt: Regression, Polynome, Splines, Künstliche Neuronale Netzwerke (KNN) usw. Funktionsapproximation ist auch eine wichtige Anwendung aus dem Bereich des überwachten Lernen (vgl. Kap. 2.1). Häufig werden zur Optimierung der Parameter Gradientenabstiegsverfahren (engl. *Gradient Descent Methods*) eingesetzt. Dabei berechnet man in einem Zustand die Gradienten, d.h. die ersten partiellen Ableitungen, der Fehler- bzw. Kostenfunktion, die es zu minimieren gilt, und bewegt sich dann in Richtung der stärksten negativen Steigung.

Betrachtet man als Input die Zustände s mit einer Wahrscheinlichkeitsverteilung $P(s)$ und als Output die Wertefunktion V^π, dann gilt für die zu approximierende Funktion V_t:

$$\text{MSE}(\vec{\theta_t}) = \sum_{s \in \mathcal{S}} P(s) \left[V^\pi(s) - V_t(s)\right]^2 \quad \rightarrow \quad \min \tag{2.26}$$

Dabei stellt MSE die mittlere quadratische Abweichung (engl. *Mean Squared Error*) als Fehlerfunktion dar, während $\vec{\theta_t}$ den Parameter-Vektor beschreibt, nachdem folglich diese Funktion partiell differenziert wird.

2.6 Exploration vs. Exploitation

Am Ende von Kap. 2.1 wurde bereits das *Exploration-Exploitation*-Dilemma angesprochen. In diesem Abschnitt werden nun konkrete Strategien vorgestellt, wie man dieses Thema behandeln kann.

Greedy-Strategien *Greedy* ist das englische Wort für gierig. In einem gierigen Algorithmus wird in jedem Schritt derjenige Folgezustand ausgewählt, der bis dahin den größten Gewinn bzw. das beste Ergebnis verspricht. Im Sinne eines *Reinforcement Learning* Problems würde demnach ein Agent im Zustand s zufällig eine Aktion a aus einer Menge von bislang optimal eingeschätzten Aktionen $\mathcal{A}^*(s)$ auswählen:

$$a(s) \in \mathcal{A}^*(s) = \arg_a \max Q(s,a) \tag{2.27}$$

Aktionen dieser Menge besitzen den höchsten, geschätzten Q-Wert in Zustand s. Es ist zu erwarten, dass durch diese Wahl von a der kumulierte Reward maximiert wird. Es gilt:

$$\pi(s,a) = \begin{cases} \frac{1}{|\mathcal{A}^*(s)|} & \text{falls } a \in \mathcal{A}^*(s) \\ 0 & \text{ansonsten .} \end{cases} \tag{2.28}$$

Somit nutzt die Greedy-Strategie das bislang gelernte Umgebungswissen gierig aus. Die Werte von Q sind jedoch nur geschätzt. Es könnte also sein, dass es Aktionen gibt, die unterschätzt werden, also zukünftig zu höheren kumulierten Rewards führen können, jedoch nicht in der Menge $\mathcal{A}^*(s)$ enthalten sind.

Um diesen Umstand entgegen zu wirken, wird die Greedy-Strategie etwas modifiziert und der Parameter ϵ eingeführt:

$$\pi(s,a) = \begin{cases} \frac{1-\epsilon}{|\mathcal{A}^*(s)|} + \frac{\epsilon}{|\mathcal{A}(s)|} & \text{falls } a \in \mathcal{A}^*(s) \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{ansonsten ,} \end{cases} \tag{2.29}$$

wobei $\epsilon \in [0,1]$ gilt. Man nennt ϵ auch Explorationsrate. Für den Extremfall $\epsilon = 0$ bekommt man wieder die normale Greedy-Strategie (Gl. 2.28), also Exploitation. Im anderen Extremfall $\epsilon = 1$ wird immer zufällig aus allen Aktionen gewählt, denn jede Aktion ist hier gleich wahrscheinlich. Dies ist der Explorationsfall. Für die Wahl des Parameter ϵ benötigt man also das richtige Fingerspitzengefühl.

Aus diesem Grund wurde die *Decreasing-ϵ*-Strategie eingeführt. Es wird mit einem hohen Wert des Parameters ϵ begonnen, also mit hoher Explorationsrate. Im Laufe des Lenprozesses wird dann der Wert von ϵ immer weiter verringert. Denn mit zunehmender Anzahl an Interaktionen wird das Wissen über die langfristigen Auswirkungen von Aktionen des Agenten immer sicherer, was ausgenützt werden sollte. Hierzu wird der Abschwächungsfaktor λ eingeführt, mit $\lambda \in (0,1)$ und es gilt:

$$\epsilon_{t+1} = \lambda \cdot \epsilon_t \tag{2.30}$$

Diese Strategie ist aber nur dann in RL-Problemen geeignet, wenn deterministische Rewards oder stationäre Reward-Verteilungen vorliegen. Ändert sich dagegen die Umgebung, dann muss ein Neuerkunden des Agenten ermöglicht werden, was bei $\epsilon \to 0$ aber nicht mehr funktionieren würde.

Eine weitere Strategie, die *ϵ-First*-Strategie, unterteilt den Lernprozess in zwei Phasen. In der Explorationsphase werden nur zufällige Aktionen gewählt ($\epsilon = 1$). In der darauffolgenden Exploitationsphase beginnt die Greedy-Strategie ($\epsilon = 0$) und es werden nur Aktionen mit dem höchsten Q-Wert gewählt. Der Zeitpunkt zum Umschalten der Lernphasen muss vom Experimentator festgelegt werden.

Softmax-Strategie Ein Nachteil aller Greedy-Strategien ist, dass eine Gleichverteilung der Aktionen auf Basis von geschätzten Q-Werten angenommen wird, die zur Exploration ausgewählt werden. Bei der Softmax-Strategie werden nun die Auswahlwahrscheinlichkeiten mit Hilfe der Softmax-Funktion, eine normalisierte Exponentialfunktion, berechnet. Die Softmax-Funktion kommt ursprünglich aus der Themodynamik und ist dort als Gibbs-Boltzmann-Verteilung bekannt. In einem geschlossenen System, das sich im thermodynamischen Gleichgewicht bei einer bestimmten (makroskopischen) Temperatur befindet, gibt diese Verteilung die Wahrscheinlichkeit an, dass sich ein Teilchen dieses Systems in einem bestimmten (mikroskopischen) Energiezustand befindet, wobei die Boltzmann-Konstante genau den Zusammenhang zwischen Temperatur und Energie angibt. Übertragen auf das Exploration-Exploitation-Dilemma ergibt sich:

$$\pi(s,a) = \frac{\exp(\frac{Q(s,a)}{\tau})}{\sum_b \exp(\frac{Q(s,b)}{\tau})} \tag{2.31}$$

Der Temperatur-Parameter $\tau > 0$ dient zur Steuerung zwischen Exploration und Exploitation. Aktionen mit einem relativ hohen Q-Wert bekommen nun auch eine relativ hohe Auswahlwahrscheinlichkeit. Der erste Grenzfall $\tau \to 0$ führt zu einem Verhalten wie in der Greedy-Strategie (Exploitation). Der zweite Grenzfall $\tau \to \infty$ führt dazu, dass alle Aktionen mit der gleichen Wahrscheinlichkeit ausgewählt werden (Exploration). Die Auswahl eines geeigneten Werts für den Parameter τ ist nicht einfach. Denn abhängig von dem gewählten Wert für τ können die Exponentialfunktionen in Gl. 2.31 je nach der Größe der Q-Werte zu extrem großen oder kleinen Zahlen führen, die sich selbst im 64-Bit-Modus numerisch ggf. nicht mehr ordentlich mit dem verwendeten Computerprogramm darstellen lassen.

2.7 Algorithmen

In diesem Abschnitt werden die bekanntesten Algorithmen kurz vorgestellt, mit deren Hilfe sich Problemstellungen des *Reinforcement Learning* lösen lassen.

Q-Learning Dieser Algorithmus wurde 1989 von Christopher J. C. H. Watkins entwickelt [Wat89] und gehört zu den *Off-Policy* TD *Control* Verfahren. Die aktuellen Beobachtungen des Agenten in Form des erhaltenen Rewards r_{t+1} und des Folgezustands s' wird benutzt, um die Aktions-Wertefunktion $Q(s,a)$ zu schätzen. Nach Ausführung der Aktion a_t in Zustand s_t beschreibt die Die Lernregel

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right] \tag{2.32}$$

wie $Q(s_t,a_t)$ aktualisiert wird. Deshalb hat der Algorithmus auch diesen Namen. Der Klammerausdruck hinter der Lernrate α kann wieder als TD-Fehler interpretiert werden (vgl. Gl. 2.20 aus Kap. 2.5). In Pseudocode kann das Verfahren wie folgt beschrieben werden.

Algorithmus 1 Q-Learning – in Anlehnung an [SB98]

Initialisiere $Q(s,a)$ beliebig
for all Episoden **do**
    Initialisiere Startzustand s
    repeat
        Wähle Aktion a gemäß Politik $\pi(s,a)$
        Führe Aktion a aus, beobachte Reward r und Folgezustand s'
        $b^* \leftarrow \arg\max_{b \in \mathcal{A}(s')} Q(s',b)$
        $\Delta \leftarrow r + \gamma Q(s',b^*) - Q(s,a)$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\Delta$
        $s \leftarrow s'$
    until s ist Endzustand

Zu Q-Learning gibt es mittlerweile zahlreiche Erweiterungen, bspw. Correlated-Q [GH03], Coco-Q [KK10; Sod+13], Friend-and-Foe-Q [Lit01], GQ(λ) [MS10], Greedy-GQ [Mae+10] oder Double-Q [van10]. Nach dem Durchbruch beim Training tiefer Künstlicher Neuronaler Netzwerke (KNN) kann man *Deep Learning* Techniken auch zur Funktionsapproximation in Kombination mit Q-Learning in *Reinforcement Learning* Problemen einsetzen. Hierzu wurden Deep Q-Network (DQN) [Mni+15], Double Deep Q-Network (DDQN) [vGS15] und Dueling DQN [WFL15] entwickelt. Mit diesen neuartigen *Deep Reinforcement Learning* Algorithmen lassen sich Probleme lösen, die sehr viele Eingabeinformationen verarbeiten müssen, wie bspw. die grafischen Ausgaben von Computerspiele (vgl. Kap. 2.3). Abb. 2.9 zeigt den schematischen Aufbau eines tiefen KNNs mit vielen Schichten, um die Eingabeinformationen in Steuerungsaktionen des Joysticks umzusetzen.

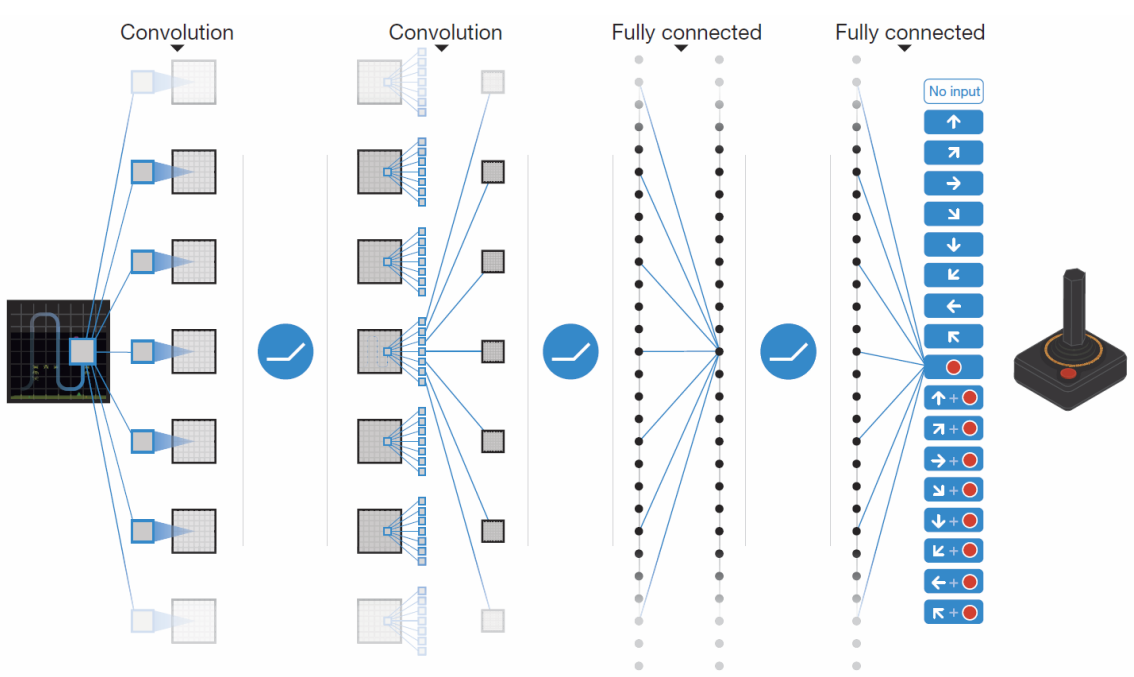


Abbildung 2.9: Schematischer Aufbau eines tiefen KNN zum DRL [Mni+15]

Die Erklärung der Funktionsweise von KNN und DL ist nicht Bestandteil dieser Arbeit. Interessierten Lesern sei das Standardwerk von Ian Goodfellow, Yosuah Bengio und Aaron Courville empfohlen [GBC16].

Beim DDQN werden DQN und das Double-Q-Learning miteinander kombiniert. Die Double-Q Methode soll die Überanpassung (*Overfitting*) verringern. Das Dueling DQN ist eine spezielle KNN-Architektur, in der zwei Datenströme durch das Netzwerk geschickt werden: Einer zur Approximation der $V(s)$-Wertefunktion in Abhängigkeit der Zustände s und der andere zum Lernen von Aktionsvorteilen $A(s,a)$ bezüglich s und der Aktionen a. Letztendlich wird dann aber wieder die Q-Funktion approximiert: $Q(s,a) = V(s) + A(s,a)$. *Dueling* bedeutet eigentlich Duellieren, kontextbezogen ist damit gemeint, dass beide Größen V und A wesentlichen Einfluss auf das Ergebnis nehmen können.

SARSA Dieser Algorithmus wurde 1994 von Gavin A. Rummery und Mahesan Niranjan [RN94] vorgeschlagen, der Name *State–Action–Reward–State–Action (SARSA)* stammt aber von Richard S. Sutton [Sut96]. SARSA ist ein *On-Policy* TD *Control* Verfahren. Ziel ist auch hier das Erlernen der Aktions-Wertefunktion Q statt der Wertefunktion V durch Schätzung der Werte $Q(s,a)$ für die verwendete Policy $\pi(s,a)$. Die Lernregel

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right] \tag{2.33}$$

benötigt nicht den maximalen Q-Wert des Folgezustands wie der Q-Learning-Algorithmus (s.o.), sondern es wird der Q-Wert der tatsächlich gewählten Aktion im Folgezustand $Q(s_{t+1},a_{t+1})$ benutzt. Der SARSA-Algorithmus kann in Pseudocode wie folgt beschrieben werden.

Algorithmus 2 SARSA – in Anlehnung an [SB98]

Initialisiere $Q(s,a)$ beliebig
for all Episoden **do**
    Initialisiere Startzustand s
    Wähle Aktion a von Zustand s gemäß Politik $\pi(s,a)$ abgeleitet von Q
    repeat
        Führe Aktion a aus, beobachte Reward r und Folgezustand s'
        Wähle Aktion a' und Folgezustand s' gemäß Politik $\pi(s',a')$ abgeleitet von Q
        $\Delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
        $Q(s,a) \leftarrow Q(s,a) + \alpha\Delta$
        $s \leftarrow s'; a \leftarrow a'$
    until s ist Endzustand

Auch zu SARSA gibt es wieder zahlreiche Erweiterungen, bspw. Expected SARSA [van+09], Async One-Step SARSA [Mni+16] oder in Anlehnung an das Double-Q-Learning (s.o.) und in Kombination mit *Deep Learning* die Algorithmen Double SARSA, Double Expected SARSA, Deep Double SARSA und Deep Expected Double SARSA [GDH16]. Viele klassischen RL-Algorithmen hat Csaba Szepesvári in seinem Buch mathematisch detailliert beschrieben [Sze10].

3 RL-Software

Dieses Kapitel ist als Vorbereitung zu den nächsten vier Kapiteln gedacht. In diesem Kapitel wird beschrieben, nach welchen Kriterien die Softwarelösungen zu Problemen des *Reinforcement Learning* ausgewählt wurden, die in den nachfolgenden Kapiteln präsentiert werden. Dabei werden einige Grundbegriffe kurz erklärt und auf die wesentlichen Unterschiede von populären Programmiersprachen in dieser Domäne eingegangen. Abschließend wird eine für diese Arbeit sinnvolle Klassifikation der Softwarelösungen durchgeführt und vorgestellt.

3.1 GitHub

GitHub ist eine Internetplattform, auf der *Open Source* Softwareprojekte ihre Quellcodes bereitstellen können [Wik18g]. Der Name setzt sich zusammen aus den Worten *Git* und *Hub*. *Git* ist ein verteiltes Versionsverwaltungssystem [Wik18f]. Ein *Hub* ist der zentrale Knoten in einem Netzwerk. Am 02.02.2018 wurde eine Suche auf GitHub zu dem Begriff *Reinforcement Learning* durchgeführt [Git18]. 5.307 Repositories wurden gefunden. Repositories sind Projektverzeichnisse, die mit Git verwaltet und auf der Plattform bereitgestellt werden. Von diesen 5.307 Repositories lassen sich 4.374 einer Programmiersprache zuordnen. Manche Repositories enthalten gar keine Quellcodes, sondern sind bspw. als Sammlung nützlicher Links angelegt. In Tabelle 3.1 sind die Suchergebnisse nach Programmiersprachen aufgeschlüsselt dargestellt.

Nr	Sprache	Anzahl	Anteil
1	Python	2.650	60,6 %
2	Jupyter Notebook	712	16,3 %
3	Java	258	5,9 %
4	C++	223	5,1 %
5	HTML	155	3,5 %
6	Matlab	144	3,3 %
7	JavaScript	83	1,9 %
8	C#	60	1,4 %
9	TeX	50	1,1 %
10	Lua	39	0,9 %

Tabelle 3.1: Übersicht von RL-Repositories auf der Plattform GitHub [Git18] (Stand: 02.02.2018)

Es fällt auf, dass die mit Abstand meisten Repositories zu *Reinforcement Learning* in der Programmiersprache Python geschrieben sind. Jupyter Notebook ist eigentlich keine eigene Programmiersprache, sondern eine *Open Source* Webapplikation, mit der Dokumente erstellt und ausgetauscht werden können. Sehr häufig wird diese Anwendung benutzt, um Python-Quelltexte zu schreiben und auszuführen. Somit wird sicherlich auch ein großer

Teil der Jupyter Notebook Repositories Python-Skripte beinhalten. In den nächsten Kapiteln werden ausgewählte Repositories und die dahinterliegenden Projekte vorgestellt. Es ist aufgrund der Vielzahl der Repositories nicht möglich, alle vorzustellen. Hierzu muss also eine Auswahl getroffen werden. Die Ergebnisse auf GitHub lassen sich nach der besten Übereinstimmung zum Suchbegriff (*Best Match*), den meisten positiven Bewertungen (*Most Stars*), den meisten privaten Repository-Kopien (*Most Forks*) oder der neusten Aktualisierung (*Recently updated*) sortieren. Außerdem kann man einige statistische Daten zu jedem Repository abrufen, deren Attribute in Tab. 3.2 kurz erklärt sind.

Nr	Attribut	Erklärung
1	Watch	Projekt beobachten
2	Star	Positive Bewertung
3	Fork	Eigene, private Variante des Repositories
4	Commits	Einreichen von Quellcode (neu, geändert)
5	Contributors	Am Projekt beteiligte Mitglieder
6	Issues Open	Frage stellen, Fehler melden
7	Issues Closed	Issue beantwortet
8	Pull Requests Open	Anfrage an den Admin zwecks Commit
9	Pull Requests Closed	Anfrage abgeschlossen
10	Release (Date)	Nr. der Release-Version und Datum
11	Last Commit	Zeitstempel der letzten Änderung

Tabelle 3.2: Übersicht von Attributen zu einem Repository auf GitHub

3.2 Programmiersprachen

In Tabelle 3.1 (vgl. Kap. 3.1) wurden die Repositories zum Thema *Reinforcement Learning* der Plattform GitHub nach Programmiersprachen unterteilt dargestellt. Vier dieser Sprachen, die für die nachfolgenden Kapitel relevant sind, werden in diesem Kapitel kurz vorgestellt. Generell ist eine Programmiersprache eine formale Sprache, in der Anweisungen schriftlich als Quelltext (engl. *Source Code*) formuliert sind, die der Computer dann ausführt. Die Syntax bestimmt dabei die Regeln, nach denen Schlüsselwörter, Ausdrücke, Operatoren usw. zu einem ausführbaren Programm angeordnet werden müssen. Syntaxfehler lassen sich von einem Compiler finden, einem Programm, das den Quelltext übersetzt. Programme, die noch Syntaxfehler enthalten, können nicht ausgeführt werden. Der Begriff Semantik steht für den Sinn und die Bedeutung des Programms hinter dem Quelltext. Semantische Fehler sind schwer zu finden. Diese treten normalerweise erst zur Laufzeit auf, d.h. während das Programm ausgeführt wird, und sich das Programm dann nicht so verhält, wie es eigentlich von den Programmierern beabsichtigt war. Heutzutage werden meistens objektorientierte Programmiersprachen benutzt. Sie zählen zu den höheren Programmiersprachen der 5. Generation. Durch sie soll die Komplexität von Software reduziert und die Wiederverwendung von einzelnen Teilen unterstützt werden. Des Weiteren kommen häufig Skriptsprachen zum Einsatz. Dies sind einfache Sprachen, die für kleine Programme gedacht sind und nicht per Compiler übersetzt werden, sondern direkt über einen Interpreter ausgeführt werden.

Python Die Programmiersprache Python wurde ursprünglich von Guido van Rossum 1991 entwickelt, wird aber mittlerweile von der Python Software Foundation weiterentwickelt und veröffentlicht [Wik18l]. Sie zählt sowohl zu den objektorientierten Programmiersprachen als auch zu den Skriptsprachen. Python zeichnet sich durch seine Einfachheit, Übersichtlichkeit und Erweiterbarkeit aus. Die Sprache kommt mit relativ wenigen Schlüsselwörtern (vereinfacht: Vokabeln) aus. Eine Besonderheit ist, dass Anweisungsblöcke durch Einrückungen strukturiert werden. Mit dem Paketmanager PIP lassen sich sehr leicht Programmbibliotheken als zusätzliche Pakete vom zentralen Repository *Python Package Index (PyPI)* installieren. Im Bereich *Data Science* gibt es eine große Auswahl an wissenschaftlichen Bibliotheken wie z.B. NumPy, Pandas, Matplotlib usw. Die Python-Distribution Anaconda enthält bereits viele dieser Pakete und außerdem das Jupyter Notebook. Python ist für die gängigen Betriebssysteme Linux, MacOSX und Windows frei erhältlich und wird aktuell in zwei Versionslinien entwickelt. Python 2 gibt es seit 2000 und aktuell ist die Version 2.7.14. Python 3 ist seit 2008 verfügbar und 3.6.4 ist die aktuellste Version dieser Linie. Beide Versionslinien sind jedoch nicht kompatibel zueinander, d.h. man muss sich für eine entscheiden. Falls man freie Wahl hat, sollte man die neuere, also Version 3, benutzen.

Java Die objektorientierte Programmiersprache Java wurde von James Gosling und Kollegen bei Sun Microsystems entwickelt und 1995 veröffentlicht [Wik18j]. Inzwischen gehört Sun Microsystems zum Unternehmen Oracle. Die Java-Technologie besteht aus dem *Java Development Kit (JDK)*, einem Entwicklungswerkzeug inklusive mitgelieferter Standard-Bibliotheken sowie der *Java Runtime Environment (JRE)*, einer Laufzeitumgebung, die wiederum die *Java Virtual Machine (JVM)*, also eine virtuelle Maschine, enthält. Das Ziel hinter diesem Technologie-Konzept ist die Plattformunabhängigkeit. Java-Programme lassen sich leicht auf ein anderes System portieren. Hierzu muss eine passende JRE installiert sein. Für die gängigen Betriebssysteme (Linux, MacOSX und Windows) existieren solche Java-Laufzeitumgebungen. Der Java-Quelltext wird zunächst vom Java-Compiler in sogenannten Java-Bytecode übersetzt. Dieser wird dann vom Interpreter der JVM unter Verwendung des *Class Loader* ausgeführt. Aktuell ist Java in der Version 9 verfügbar.

C++ Die Programmiersprache C++ ist bereits 1979 von Bjarne Stroustrup bei AT&T entwickelt worden und gilt als objektorientierte Erweiterung zur imperativen und prozeduralen Programmiersprache C [Wik18b]. Der Inkrement-Operator ++ erhöht den Wert einer Variablen um Eins. In Anlehnung daran hat die Programmiersprache C++ somit ihren Namen bekommen. C++ ermöglich eine besonders effiziente und maschinennahe Programmierung, weshalb bspw. vor allem Betriebssysteme, eingebettete Systeme, virtuelle Maschinen, Treiber usw. in dieser Sprache programmiert sind. Der Compiler übersetzt im ersten Schritt den Quelltext in Objektcode und im zweiten Schritt baut der Linker diesen Objektcode zu einem ausführbaren Programm zusammen, wobei optional noch weitere Bibliotheken eingebunden werden. Der Quelltext lässt sich nicht so leicht portieren wie im Fall von Java oder Python. Für jedes Zielsystem, auf dem das Programm ausgeführt werden soll, müssen ggf. Anpassungen im Quelltext vorgenommen werden und dann muss die Software für jedes der Zielsysteme separat gebaut werden. Dieser hohe Entwicklungsaufwand wird aber mit einer sehr guten Performance belohnt. Die aktuelle Version ist C++17.

Lua Die Programmiersprache Lua, portugiesisch für Mond, ist von Roberto Ierusalimschy, Waldemar Celes und Luiz Henrique de Figueiredo entwickelt und 1993 veröffent-

licht worden [Wik18k]. Lua zählt zu den imperativen, objektorientierten Skriptsprachen. Vor der Ausführung wird der Quelltext in Bytecode übersetzt, wodurch eine ähnliche Plattformunabhängigkeit wie bei Java (s.o.) erreicht wird. Hauptsächlich wird Lua als eingebettete Skriptsprache verwendet. Ein typisches Einsatzgebiet sind Computerspiele. Hier werden computergesteuerte Spielcharaktere bspw. mittels KI-Algorithmen von Lua gesteuert, während die Spiel-Engine in C/C++ programmiert ist. Das Computerspiel World of Warcraft bietet bspw. eine Programmierschnittstelle (engl. *Application Programming Interface (API)*) in Lua an, damit Erweiterungen (*Add-ons*) von der Community entwickelt werden können. Aktuell ist Lua in der Version 5.3.4 verfügbar.

3.3 Klassifikation

Softwarelösungen zu *Reinforcement Learning* lassen sich zwar hinsichtlich der verwendeten Programmiersprache (vgl. Kap. 3.1 und 3.2) unterscheiden, jedoch ist eine andere Form der Klassifikation sinnvoller. In Kap. 2 haben wir bereits festgestellt, dass ein Agent nicht losgelöst von seiner Umwelt betrachtet werden kann. Der Agent interagiert mit seiner Umwelt und er löst dabei ein Problem, indem ein geeigneter Algorithmus zum Einsatz kommt. Diese Teile können auch als Komponenten eines Softwaresystems betrachtet werden. Wenn man sich die Softwarelösungen zum Thema *Reinforcement Learning* auf GitHub genauer ansieht, stellt man fest, dass sich diese auch in einer der Kategorien einsortieren lassen, die in Abb. 3.1 dargestellt sind.

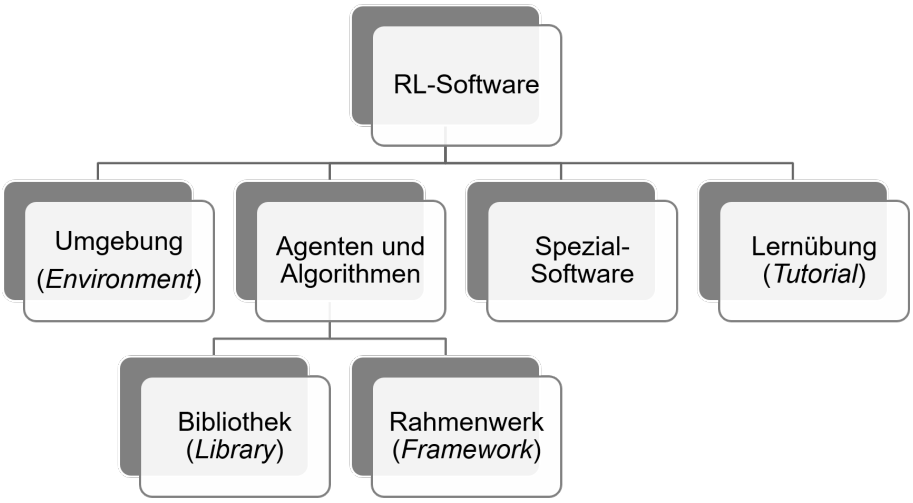


Abbildung 3.1: Klassifikation von RL-Softwarelösungen

Einige Softwarelösungen beschäftigen sich bspw. nur mit dem Thema Umgebung (engl. *Environment*) bzw. den Problemstellungen, die die Agenten lösen sollen. Ausgewählte Softwarelösungen zu dieser Kategorie werden in Kap. 4 vorgestellt.

Andere Lösungen konzentrieren sich auf Agenten und Algorithmen, die zur Lösung von RL-Problemen eingesetzt werden können. Diese Softwarelösungen lassen sich weiter unterteilen in Bibliothek (engl. *Library*) oder Rahmenwerk (engl. *Framework*). Vereinfacht gesagt ist eine Bibliothek eine Sammlung von Komponenten (hier Agenten bzw. Algorithmen), die so programmiert sind, dass sie sich sehr leicht wiederverwenden lassen. Im Unterschied zu einer Software-Bibliothek bietet ein Framework aber nicht nur wiederverwendbare Komponenten an, sondern diese Komponenten arbeiten bereits innerhalb der

vorgegebenen Rahmenstruktur zusammen. Bezogen auf das Thema RL bieten diese Frameworks also neben den Komponenten Agenten und Algorithmen auch Komponenten zu Experimenten (Parameterstudien) und Analysen an, die über definierte Schnittstellen bereits miteinander und mit einer Umgebung verbunden sind. Man versucht diese Komponenten möglichst voneinander zu entkoppeln, um Flexibilität und Erweiterbarkeit sicherzustellen, denn so lassen sich einzelne Komponenten leicht austauschen. Kapitel 5 stellt Bibliotheken und Frameworks zu RL-Lösungen vor.

In Kap. 6 werden spezielle Softwarelösungen zum *Reinforcement Learning* präsentiert. Dies sind Lösungen mit einem anderen Schwerpunkt wie bspw. Multiagentensysteme oder Frameworks, die bereits eigene Umgebungen beinhalten und deren Komponenten nicht so stark voneinander entkoppelt sind, dass sie sich leicht austauschen lassen. Außerdem werden in diesem Kapitel auch *Open Source* Softwarelösungen vorgestellt, deren Quelltext sich nicht als Repository auf der Plattform GitHub befindet.

Auf GitHub findet man auch Repositories mit Lerneinheiten, die auch auf Bücher oder Kurse beziehen. Diese *Tutorials* enthalten ebenfalls Quelltext zu RL-Problemen mit Lösungsvorschlägen, die meistens sehr ausführlich dokumentiert sind. Die Algorithmen lassen sich zwar nicht wie eine Bibliothek (wieder)verwenden, trotzdem kann der Quelltext als Grundlage für eigene Programmierungen verwendet werden. Solche Tutorials werden in Kap. 7 vorgestellt.

4 Umgebungen

Softwareagenten befinden sich nicht im luftleeren Raum, sondern sie interagieren immer mit ihrer Umgebung. Ansonsten könnten sie auch nicht dazulernen. Einige Teams von Programmierern und Wissenschaftlern haben sich darauf spezialisiert, Softwarelösungen zu entwickeln, die Umgebungen für Probleme des *Reinforcement Learning* bereitzustellen. Diese Softwarelösungen müssen definierte Schnittstellen haben, um Umweltzustände abzufragen und Aktionen durchzuführen. Es können sehr einfache Umgebungen zu klassischen RL-Problemen (Pendulum, Cart Pole, Mountain Car) und Spielen (Tic-Tac-Toe, Go) sein, aber auch 2D-Umgebungen zu Computerspiele-Klassikern von Atari (Missile Command, Pac-Man, Pitfall, Space Invaders) oder Nintendo (Mortal Combat, Super Mario). Noch komplexer sind 3D-Umgebungen aus der Ich-Perspektive zu jüngeren, populären Computerspielen (Doom, Minecraft, Wolfenstein). Mittlerweile können sogar Echtzeit-Simulationen zum Thema autonomes Fahren, in denen bspw. auch verschiedene Wettersituationen berücksichtigt werden, zum Trainieren von Agenten eingesetzt werden.

Tabelle 4.1 zeigt eine Übersicht von ausgewählten Lernumgebungen für RL-Probleme in alphabetischer Reihenfolge. Die Anzahl der Sterne (engl. *Stars*) sind positive Bewertungen von Mitgliedern der Plattform GitHub für die jeweilige Softwarelösung (Stand: 02.02.2018).

Nr	Name	Stars	Organisation
1	Arcade Learning Environment	736	University of Alberta
2	CARLA	858	Intel Labs
3	DeepMind Lab	4.460	Google DeepMind
4	OpenAI Gym	9.937	OpenAI
5	OpenAI Universe	6.548	OpenAI
6	Project Malmo	2.743	Microsoft
7	PyGame Learning Environment	427	University of Western Ontario
8	Retro Learning Environment	137	Technion (Israel)
9	Roboschool	1.022	OpenAI
10	ViZDoom	564	Poznań University

Tabelle 4.1: Übersicht ausgewählter RL-Umgebungen auf der Plattform GitHub [Git18]

In den nachfolgenden Kapiteln werden diese ausgewählten Lernumgebungen nacheinander kurz vorgestellt. Feststellen lässt sich bereits, dass die Softwarelösung *OpenAI Gym* innerhalb der Plattform GitHub am beliebtesten und bekanntesten ist. Des Weiteren werden zwei andere Lösungen vorgestellt, die ebenfalls von der Non-Profit-Organisation OpenAI entwickelt wurden. Ansonsten sind mit Lernumgebungen von Google, Intel und Microsoft drei weitere namenhafte Unternehmen in der Tabelle vertreten.

4.1 Arcade Learning Environment

Das Framework Arcade Learning Environment (ALE) ist von Marc G. Bellemare, Yavar Naddaf, Joel Veness und Michael Bowling an der Universität von Alberta (Kanada) in der objektorientierten Programmiersprache C++ entwickelt worden [Bel+13]. ALE benutzt den *Open Source* Emulator Stella, um Umgebungen zu mehr als 50 Videospielen der Atari 2600 Plattform für KI-Forscher bereit zu stellen [Mot+18].

Die Spielekonsole Atari 2600 wurde seit 1997 über 30 Millionen Mal verkauft [Wik18a]. Sie zählt damit zu den erfolgreichsten Spielekonsolen ihrer Zeit. Mehr als 1200 Spiele wurden für Atari 2600 entwickelt und veröffentlicht. Bekannte Spiele sind bspw. Pac-Man, Pitfall, Missile Command und Space Invaders (siehe auch Abb. 4.1).

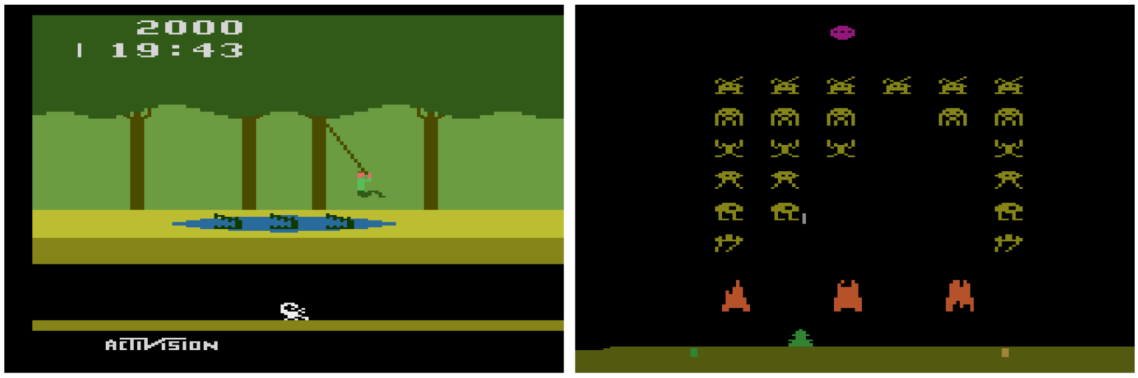


Abbildung 4.1: ALE: Die Atari-Spiele Pitfall (links) und Space Invadors (rechts) [Bel+13]

Der Emulator kann die Eingabesignale des Joysticks oder der Tastatur entgegen nehmen und liefert als Ergebnis ein Bild mit einer Auflösung von 160 x 210 Pixel mit 7-Bit-Farbtiefe in Wiederholungsraten von 60 Frames pro Sekunde, also in 60 Hz (siehe Abb. 4.2).

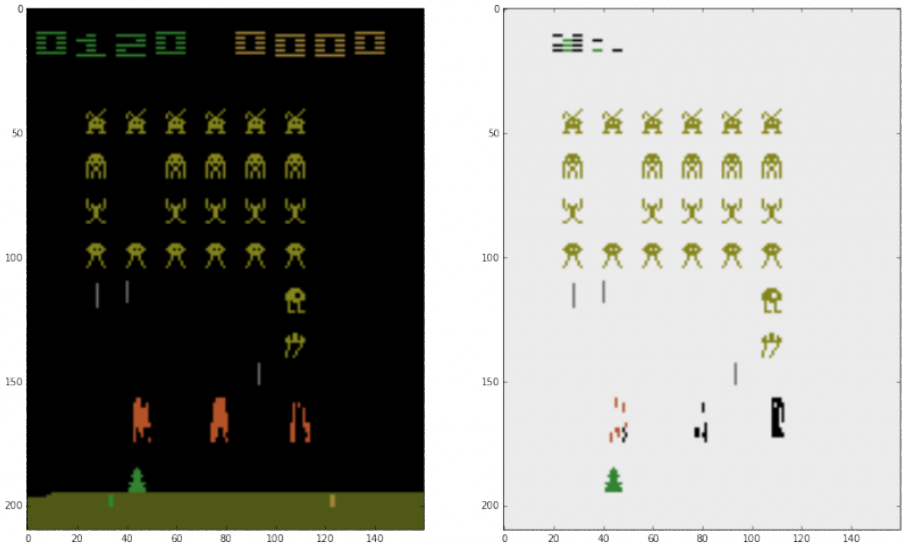


Abbildung 4.2: ALE: 160x210-Pixel-Output des Emulators Stella zum Spiel Space Invaders [Lab18]

Auch Stella ist in C++ programmiert. Die Kommunikation zwischen den Agenten und dem Kern des Emulators wird mit Hilfe von *Pipes* realisiert. Dies ist in C++ direkt möglich, zu Java gibt es ein Beispiel und darüber hinaus wurde eine spezielle Erweiterung als

Schnittstelle zu Python entwickelt. Dadurch kann bspw. auch der aktuelle Spielstand abgefragt werden. Außerdem wird das Signal extrahiert und weitergeleitet, wenn das Spiel beendet ist. Das Framework lässt sich auf allen gängigen Plattformen installieren und nutzen. Einige Werkzeuge zur Visualisierung werden ebenfalls mitgeliefert.

Tabelle 4.3 fasst die wichtigsten Merkmale von ALE zusammen [BMH+18].

Name	Arcade Learning Environment
Organisation	University of Alberta
Webseite	https://github.com/mgbellemare/Arcade-Learning-Environment
Lizenz	GPL Version 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	C++
APIs	C++, (Java), Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	53 ǀ 736 ǀ 191
Commits ǀ Contributors	718 ǀ 25
Issues: Open ǀ Closed	24 ǀ 82
Requests: Open ǀ Closed	7 ǀ 127
Release (Date)	0.6.0 (01.12.2017)
Last Commit	29.12.2017

Tabelle 4.2: Steckbrief zum ALE-Framework [BMH+18]

4.2 CARLA

Car Learning to Act (CARLA) wurde von Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez und Vladlen Koltun in Zusammenarbeit der Intel Labs mit dem Toyota Research Institute und dem Computer Vision Center (Barcelona) unter Mitwirkung von Néstor Subirón und Francisco Perez entwickelt [Dos+17]. Es handelt sich dabei nicht um eine Umgebung für klassische *Reinforcement Learning* Probleme, sondern wie der Name schon sagt, um einen *Open Source* Simulator für die Forschung im Bereich des autonomen Fahrens. CARLA ist als Client/Server-Architektur konzipiert.

Der Server benutzt das *Open Source* Framework Unreal Engine 4 (UE4) [Wik18n]. Diese Spiele- und Grafik-Engine generiert das 3D-Modell zu der Simulation. Hierzu werden statische Objekte wie Gebäude, Bäume, Zäune, Verkehrszeichen und die Infrastruktur (Fahrbahn, Fußgängerweg, Markierungen usw.) sowie dynamische Objekte wie Fahrzeuge (16 Modelle) und Fußgänger (50 Modelle) berücksichtigt. 14 verschiedene Wettersituationen lassen sich simulieren (siehe Abb. 4.3). Dieser Teil wurde in der objektorientierten Programmiersprache C++ erstellt.

Abbildung 4.3: CARLA: Die gleiche Simulation in vier verschiedene Wettersituationen [Dos+17]

Der Client verwendet zur Kommunikation mit dem Server und als API die Skriptsprache Python (in Version 2 oder 3). Es lassen sich bspw. die folgenden Aktionen durchführen:

Nr	Aktor	Wertebereich			Erklärung
1	Gaspedal	0	bis	1	0 = kein Gas, 1 = Vollgas
2	Bremse	0	bis	1	0 = keine Bremse, 1 = Vollbremsung
3	Lenkrad	-1	bis	+1	-1 = voll links, +1 = voll rechts
4	Handbremse	0	oder	1	1 = Handbremse angezogen
5	Schaltung	0	oder	1	0 = vorwärts, 1 = rückwärts

Außerdem lassen sich Sensor- und Kamera-Informationen abfragen (siehe Abb. 4.4) und

die Kamera kann ebenfalls dabei gesteuert werden: Bewegen (oben, unten, links, rechts) und Zoomen (hinein, hinaus).

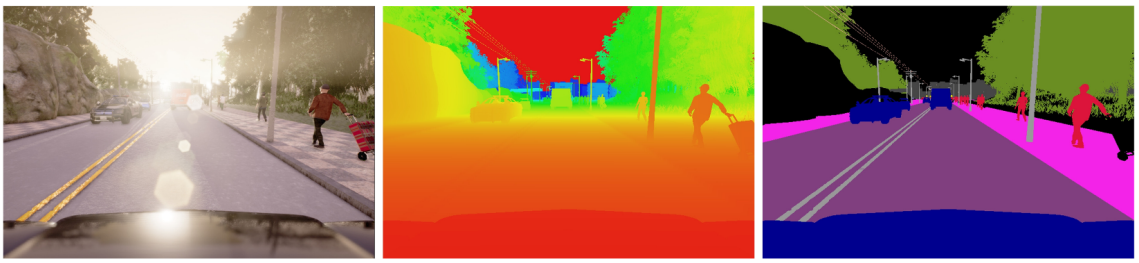


Abbildung 4.4: CARLA: Output von 3 verschiedenen Sensoren zur gleichen Szene [Dos+17]

Momentan existieren zwei verschiedene Modelle (Welten) als Basis für die Simulation (siehe Abb. 4.5).

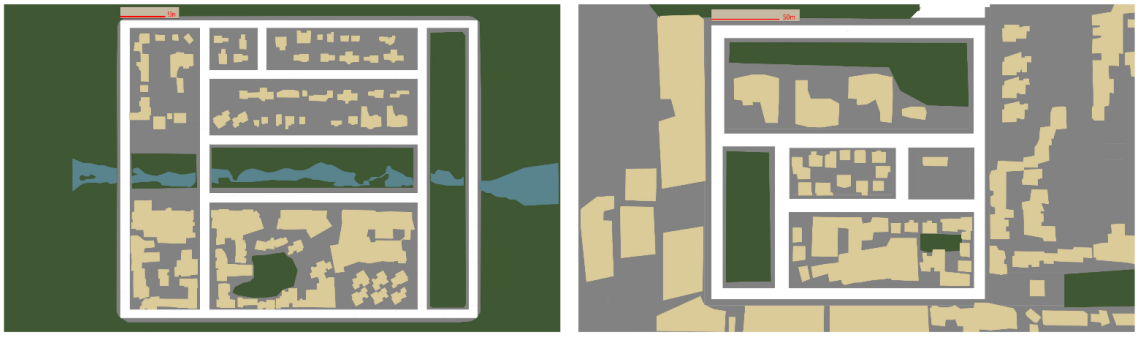


Abbildung 4.5: CARLA: Grundrisse zweier verschiedener Welten [Dos+17]

Tabelle 4.3 fasst die wichtigsten Merkmale von CARLA zusammen [Sub+18].

Name	CARLA
Organisation	Intel Labs
Webseite	https://github.com/carla-simulator/carla
Lizenz	MIT License
Plattformen	Linux, Windows
Geschrieben in	C++
APIs	C++, Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	84 ǀ 858 ǀ 167
Commits ǀ Contributors	730 ǀ 6
Issues: Open ǀ Closed	69 ǀ 120
Requests: Open ǀ Closed	2 ǀ 22
Release (Date)	0.7.1 (25.01.2018)
Last Commit	02.02.2018

Tabelle 4.3: Steckbrief zum Simulator CARLA [Sub+18]

4.3 DeepMind Lab

DeepMind Lab ist eine Plattform für agentenbasierte KI-Forschung des Unternehmens DeepMind [Bea+16]. 2010 haben Demis Hassabis, Shane Legg und Mustafa Suleyman das Start-up DeepMind in London gegründet. 2014 wurde DeepMind dann von Google Inc. übernommen, d.h. mittlerweile ist es Teil der Alphabet-Gruppe.

Die Lernumgebung von DeepMind Lab basiert auf Quake III Arena, einem Computerspiel des sogenannten Ego-Shooter-Genres, welches von dem Unternehmen id Software 1999 auf dem Markt gebracht wurde. Der Quelltext zur Spiel-Engine wurde 2005 als *Open Source* veröffentlicht. Daraufhin ist *ioquake3* entstanden, eine freie Spiel-Engine unter Linux, MacOSX und Windows, die von einer Community weiter entwickelt und gepflegt wird. DeepMind Lab benutzt außerdem die freien Werkzeuge *q3map2* und *bspc*, um die Spielwelten und Level zu generieren.

Die Aufgaben, die das DeepMind Lab für die lernenden Agenten zur Verfügung stellt, haben mit dem ursprünglichen Ego-Shooter Quake jedoch nichts mehr gemein. Beispielsweise muss man per 3D-Navigation den Ausgang eines Labyrinths finden (vgl. Abb. 4.6).

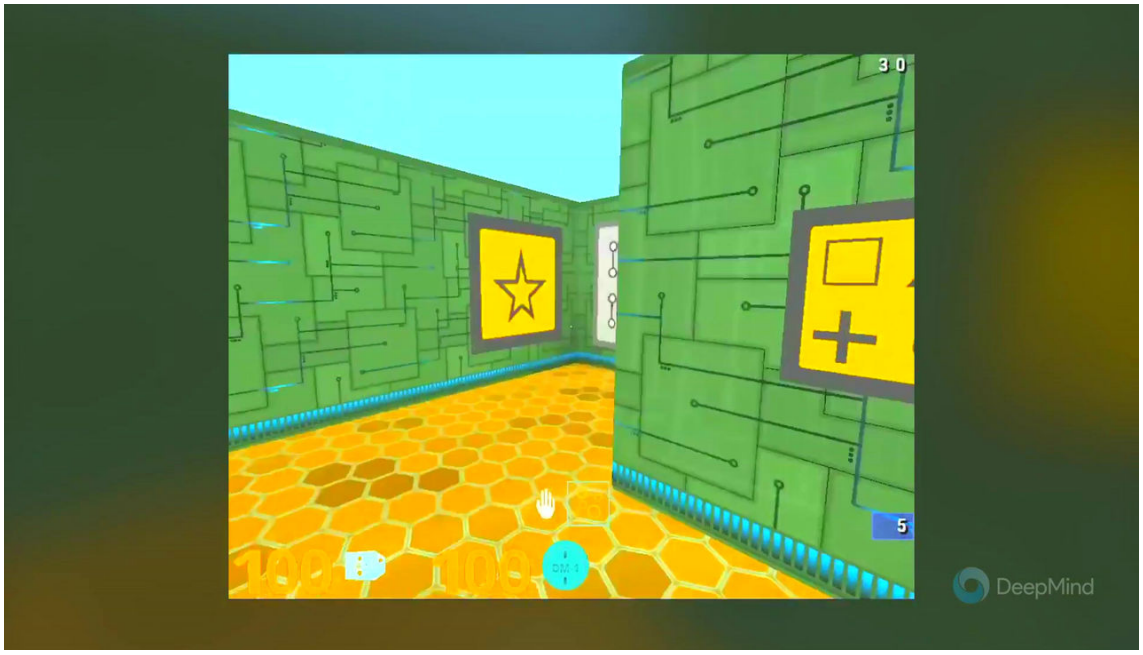


Abbildung 4.6: DeepMind Lab Level 1: Nav Maze [Gig16]

In einem nächsten Spiellevel namens *Stairway to Melon* gilt es, möglichst viele Punkte zu erreichen (vgl. Abb. 4.7). Hierzu sind Äpfel und Zitronen im Labyrinth verteilt, die man automatisch einsammelt, wenn man an ihnen vorbeigeht. Äpfel bringen dabei positive Punkte (kurzfristige Belohnung), während Zitronen negative Punkte (kurzfristige Bestrafung) verursachen. Den Hauptgewinn, die Melone, findet man aber nur, wenn man zuvor einen Weg gewählt hat, auf dem auch Zitronen liegen. Dies ist bereits ein sehr schönes und einfaches Lernbeispiel für das *Reinforcement Learning*.

Abbildung 4.7: DeepMind Lab Level 2: Stairway to Melon [Gig16]

Sehr viel komplexere Umgebungen sind bzw. die *Laser Tag* Level: *Chasm*, *Hallway Slope*, *Horseshoe Color* und *Space Bounce Hard*. DeepMind Lab ist nur für das Betriebssystem Linux verfügbar. Unabhängig von der verwendeten Linux-Distribution müssen einige zusätzlichen Softwarepakete installiert sein: SDL 2.0, OpenGL, gettext, Lua, Python und Bazel. Eine Besonderheit ist dabei, dass derzeit leider nur die Version 2 von Python unterstützt wird.

Tabelle 4.4 fasst die wichtigsten Merkmale des DeepMind Labs zusammen [BK18].

Name	DeepMind Lab
Organisation	Google DeepMind
Webseite	https://github.com/deepmind/lab
Lizenz	GPL 2.0
Plattformen	Linux
Geschrieben in	C/C++
APIs	Lua, Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	460 ǀ 4.460 ǀ 924
Commits ǀ Contributors	147 ǀ 4
Issues: Open ǀ Closed	7 ǀ 69
Requests: Open ǀ Closed	3 ǀ 5
Release (Date)	January_2018 (26.01.2018)
Last Commit	31.01.2018

Tabelle 4.4: Steckbrief zur DeepMind-Lab-Umgebung [BK18]

4.4 OpenAI Gym

OpenAI Gym ist ein *Toolkit* für *Reinforcement Learning* [Bro+16]. Die Non-Profit-Organisation OpenAI wurde am 11.12.2015 gegründet und beschäftigt sich seitdem mit der Erforschung Künstlicher Intelligenz (engl. *Artificial Intelligence (AI)*) auf Basis von *Open Source* Software. Ein wichtiger Geldgeber ist Elon Musk, der bereits im Jahr 2000 den Bezahldienst PayPal mitgegründet hat und der sich mittlerweile in zwei anderen Unternehmen engagiert, dem privaten Raumfahrtunternehmen SpaceX und dem Elektroautohersteller Tesla.

Gym ist das englische Wort für Trainingshalle. OpenAI Gym enthält eine große Sammlung von Umgebungen in Form von *Partially Observable Markov Decision Process (POMDP)*, in der sich die Softwareagenten beweisen können. Die einfachsten Umgebungen sind textbasiert, bspw. können die Probleme *Copy* und *Frozen Lake* mit einer reinen Textausgabe dargestellt werden (siehe Abb. 4.8).

Abbildung 4.8: OpenAI Gym: Einfache, textbasierte Umgebungen

Außerdem lassen sich klassische Kontrollaufgaben aus der Literatur wie bspw. *Mountain Car* und *Cart Pole* verwenden, wobei hierzu grafische Ausgaben erfolgen (siehe Abb. 4.9).

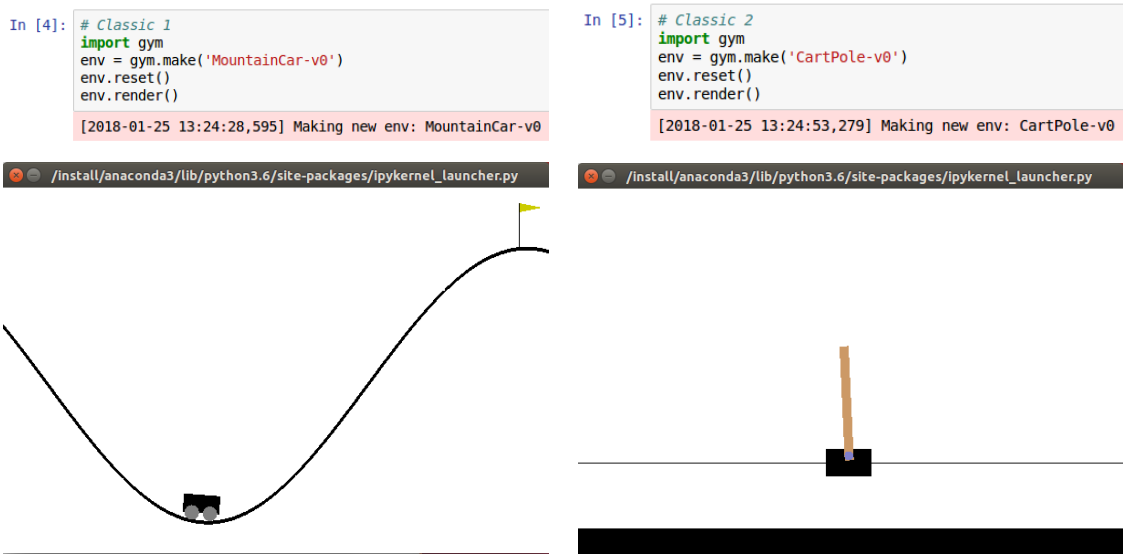


Abbildung 4.9: OpenAI Gym: Klassische Kontrollaufgaben

Komplexere Probleme bzw. Umgebungen sind bspw. Roboter-Simulationen in 2D (Swimmer, Hopper, Walker usw.) oder 3D (Ant, Humanoid usw.) sowie bekannte Atari-Spiele (Astroids, Breakout, Centipede, Pitfall, Space Invaders usw.) oder andere Spiele (Minecraft, Doom, Soccer usw.).

OpenAI Gym lässt sich sowohl unter Linux als auch MacOSX installieren und mit Python in den Versionen 2 oder 3 verwenden. Es lassen sich zwei Installationsarten unterscheiden, eine minimale und eine komplette Installation. Einige spezielle Umgebungen basieren auf MuJoCo [MuJ18]. Dies ist eine Physik-Engine, mit der realistische Simulationen unter Berücksichtigung physikalischer Gesetzmäßigkeiten durchgeführt werden können. MuJoCo ist keine *Open Source* sondern kommerzielle Software. Eine 30-Tage gültige Lizenz zum Testen kann man jedoch kostenlos bekommen.

Tabelle 4.5 fasst die wichtigsten Merkmale von OpenAI Gym zusammen [Bro+18a].

Name	OpenAI Gym
Organisation	OpenAI
Webseite	https://github.com/openai/gym
Lizenz	The MIT License
Plattformen	Linux, MacOSX
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	701 ǀ 9.937 ǀ 2.258
Commits ǀ Contributors	766 ǀ 91
Issues: Open ǀ Closed	200 ǀ 283
Requests: Open ǀ Closed	72 ǀ 301
Release (Date)	0.9.6 (01.02.2018)
Last Commit	29.01.2018

Tabelle 4.5: Steckbrief zur OpenAI-Gym-Umgebung [Bro+18a]

4.5 OpenAI Universe

OpenAI Universe ist eine Software-Plattform, um künstliche Intelligenzen zu trainieren und deren Erfolge zu messen. Es bietet eine Schnittstelle zum OpenAI Gym (vgl. Kap. 4.4). Das Besondere ist der universelle Ansatz: OpenAI Universe kann aus einem beliebigen, existierenden Computerprogramm eine OpenAI Gym Umgebung machen, ohne dass man direkten Zugriff auf den Quelltext haben muss.

Universe benutzt hierzu *Virtual Network Computing (VNC)* als Schnittstelle und eine Client-/Server-Architektur. VNC ist eine Software, die den Bildschirminhalt eines entfernten Rechners (Server) auf einem lokalen Rechner (Client) anzeigt und im Gegenzug Tastatur- und Mausbewegungen des lokalen Rechners an den entfernten Rechner sendet [Wik18o]. Die Umgebung ist somit der VNC Server und der Agent benutzt ein VNC Remote Desktop Programm als Client (vgl. Abb. 4.10).

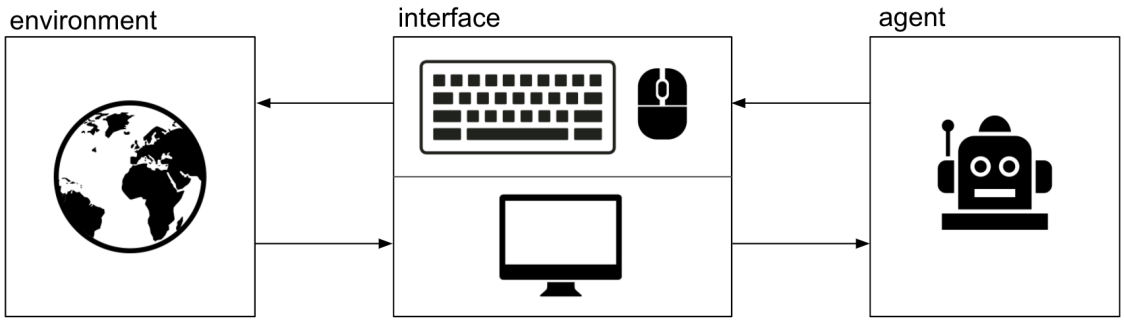


Abbildung 4.10: OpenAI Universe Architektur [Pet+16]

Der VNC-Client ist in Go programmiert und kann Pixel-Grafik in einer Auflösung von 1024 x 768 mit einer Frequenz von 60 Frames pro Sekunde liefern. Dieses Programm lässt sich als Bibliothek in Python einbinden und verwenden. Außerdem benutzt Universe einen Docker Container. Dies ist eine *Open Source* Software, um Anwendungen zu isolieren und zu virtualisieren [Wik18c]. In diesem Container läuft dann die eigentliche Anwendung ab, im Fall von Universe ist das dann die Umgebung. Des Weiteren benutzt Universe noch einen WebSocket Server, um Belohnungssignale für *Reinforcement Learning* Aufgaben an den Client zu senden. Das ist im Prinzip ein Webserver, der nach dem Verbindungsaufbau auch Nachrichten an den Client ohne vorheriger Anfrage (engl. *Request*) senden kann [Wik18p]. Somit enthält der Docker Container also drei Komponenten: Das Computerprogramm, welches als Umgebung dient und zwei Server zum Datenaustausch (VNC und WebSocket).

Mit Hilfe dieser Technik wurden aus vielen Computerprogrammen Lernumgebungen generiert. Auch das *Arcade Learning Environment* (vgl. Kap. 4.1) wird in einem Docker Container gepackt, um so Atari 2600 Spiele einzubinden. Im Internet findet man über 30.000 kostenlose Flash-Spiele, die mit ActionScript von Adobe programmiert sind und sich per Webbrowser spielen lassen, vorausgesetzt die Erweiterung (Plug-in) Flash ist installiert. Diese Spiele sind bezogen auf die Grafik bereits wesentlich anspruchsvoller als die Atari-Klassiker, aber trotzdem noch sehr einfach in Bezug auch den Spielablauf. Abb. 4.11 zeigt das Flash-Spiel Dusk Driver als OpenAI Universe Umgebung, gespielt von einem Softwareagenten.

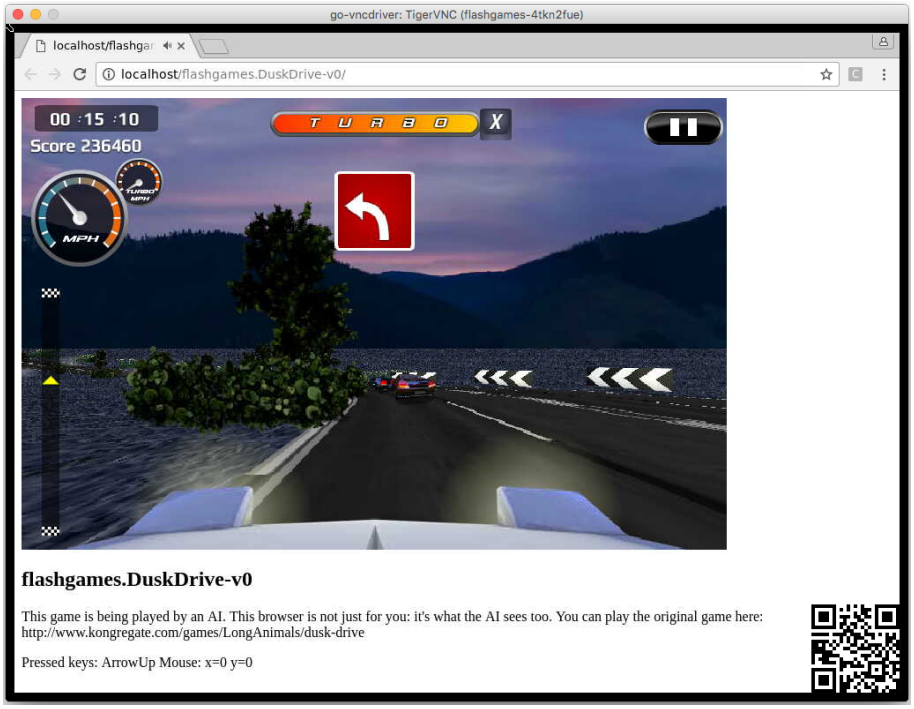


Abbildung 4.11: OpenAI Universe: Flash-Spiel Dusk Drive [Pet+16]

Mittlerweile enthält OpenAI Universe mehr als 1000 Umgebungen. Darunter sind auch sogenannte *Browser Tasks*. Dies sind Prozessabläufe, die normalerweise von einem Menschen mit einem Webbrowser durchgeführt werden, z.B. die Buchung einer Flugreise. Solche Aufgaben lassen sich aber auch von einem Softwareagenten erlernen. Zukünftig soll es noch weitere Schnittstellen zu anderen KI-Plattformen geben, bspw. zum *Project Malmo* von Microsoft (vgl. Kap. 4.6).

Tabelle 4.6 fasst die wichtigsten Merkmale von OpenAI Universe zusammen [Bro+18b].

Name	OpenAI Universe
Organisation	OpenAI
Webseite	https://github.com/openai/universe
Lizenz	The MIT License
Plattformen	Linux, MacOSX
Geschrieben in	Go, Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	401 ǀ 6.548 ǀ 718
Commits ǀ Contributors	209 ǀ 23
Issues: Open ǀ Closed	75 ǀ 113
Requests: Open ǀ Closed	8 ǀ 34
Release (Date)	0.21.3 (05.05.2017)
Last Commit	18.05.2017

Tabelle 4.6: Steckbrief zur OpenAI-Universe-Plattform [Bro+18b]

4.6 Project Malmo

Project Malmo ist eine Plattform für das Experimentieren und Forschen mit Künstlicher Intelligenz [M.+16]. Sie wird von Microsoft bereitgestellt und basiert auf dem populären Computerspiel Minecraft, welches zu den sogenannten Open-World-Spielen gehört. Dies sind Computerspiele, in denen der Spieler frei entscheiden kann, was er wann und wie in der Spielwelt erkunden möchte.

Minecraft wurde ursprünglich 2009 vom schwedischen Programmierer Markus Persson in der Programmiersprache Java entwickelt. 2010 gründete Persson die Firma Mojang, die dann 2014 an Microsoft verkauft wurde. Seitdem entwickelt Microsoft Minecraft weiter.

Malmo ist an das Design-Konzept von RL-Glue angelehnt (vgl. Kap. 6.7), in dem die drei Komponenten Agenten, Umwelt und Experiment über eine Schnittstelle miteinander verbunden sind. Malmo erweitert dieses Konzept um Echtzeit-Interaktionen und Mulit-Agenten-Aufgaben. Die *MissionSpec* spezifiziert eine Mission, also eine Aufgabe, die der Agent lösen muss. Hierzu wird eine XML-Datei als Konfiguration benutzt, in der die Karte, Belohnungssignale, Beobachtungsmöglichkeiten, Aktionsmöglichkeiten usw. definiert und beschrieben werden. Der *AgentHost* erzeugt auf dieser Grundlage dann eine Mission in der Minecraft-Umgebung und bindet Agenten daran. Dies ist also die zentrale Schnittstelle zwischen Umgebung und Agent. Auf technischer Ebene werden Sockets zur Kommunikation benutzt und JSON als Austauschformat verwendet. Ein *MissionRecord* zeichnet das Spielgeschehen in Form von Logdateien, Bildern und Videos auf. Außerdem gibt es auch eine *HumanActionComponent*, sodass auch Menschen als Spieler in das Geschehen eingreifen können (vgl. Abb. 4.12).

Abbildung 4.12: Project Malmo: Beispielumgebung mit Mensch und Agent als Spieler [Kun+15]

Im Jahr 2017 wurde von Microsoft ein Wettbewerb durchgeführt, die *Malmo Collaborative AI Challenge* [KSG17]. An diesem haben mehr als 80 Teams von Studierenden aus 26 Ländern teilgenommen. Ziel war es, Agenten zu erzeugen und zu trainieren, die in dem Spiel *Catch the Pig* ein Schwein fangen (vgl. Abb. 4.13). Dieses Ziel kann aber nur dann erreicht werden, wenn Agenten zusammenarbeiten, d.h. es handelt sich um ein Kooperationsspiel – angelehnt an das Beispiel Hirschjagd aus der Spieltheorie.

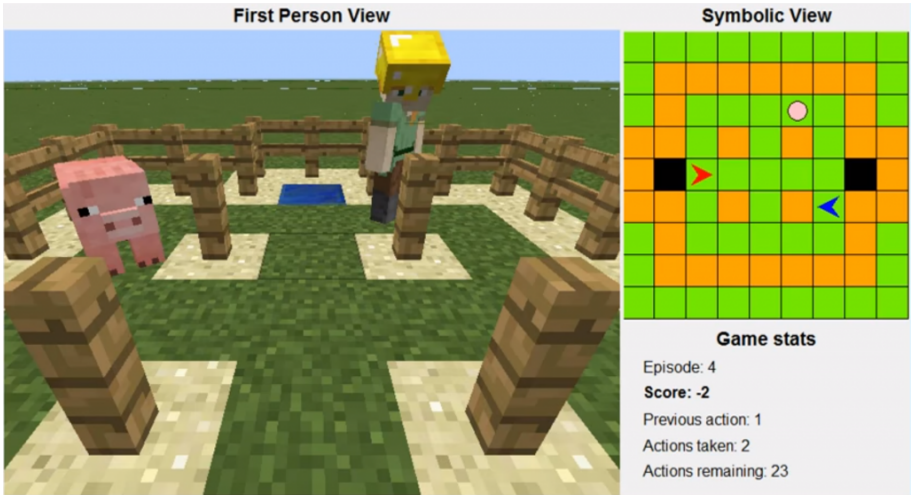


Abbildung 4.13: Project Malmo: Catch the Pig [KSG17]

Project Malmo lässt sich auf allen gängigen Betriebssystemen (Linux, MacOSX und Windows) installieren und bietet eine API zu vielen Programmiersprachen (C++, C#, Java, Lua und Python).

Tabelle 4.7 fasst die wichtigsten Merkmale von Project Malmo zusammen [BHH+18].

Name	Project Malmo
Organisation	Microsoft
Webseite	https://github.com/Microsoft/malmo
Lizenz	The MIT License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Java
APIs	C++, C#, Java, Lua, Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	281 ǀ 2.743 ǀ 424
Commits ǀ Contributors	1.195 ǀ 17
Issues: Open ǀ Closed	107 ǀ 398
Requests: Open ǀ Closed	5 ǀ 149
Release (Date)	0.31.0 (26.09.2017)
Last Commit	17.11.2017

Tabelle 4.7: Steckbrief zur Project Malmo Plattform [BHH+18]

4.7 PyGame Learning Environment

PyGame Learning Environment (PLE) ist eine Lernumgebung, die von Norman Tasfi, einem Master-Studenten der Universität von Western Ontario (Kanada), in der Programmiersprache Python entwickelt wurde. Die Software orientiert sich sehr stark an die Schnittstelle von Arcade Learning Environment (ALE) (vgl. Kap. 4.1). Somit lassen sich Projekte, die bereits ALE verwenden, mit wenig Aufwand an PLE anpassen und umgekehrt [TU18].

Die folgenden Spiele sind als Umgebung bereits verfügbar: Catcher, Monster Kong, FlappyBird, Pixelcopter, Pong, PuckWorld, RaycastMaze, Snake und WaterWorld (vgl. Abb. 4.14). Das bekannteste ist sicherlich FlappyBird vom vietnamesischen Entwickler Dong Nguyen aus dem Jahr 2013 [Wik18e]. Bei diesem Spiel muss der Spieler einen Vogel durch eine von rechts nach links scrollende Spielwelt steuern, um so zwischen Röhren hindurchzufliegen und Punkte zu sammeln. Die App wurde auch deshalb so bekannt, weil sie im Februar 2014 bereits wieder aus den App Stores entfernt wurde. Die offizielle Begründung des Entwicklers war, dass das Spiel süchtig macht.

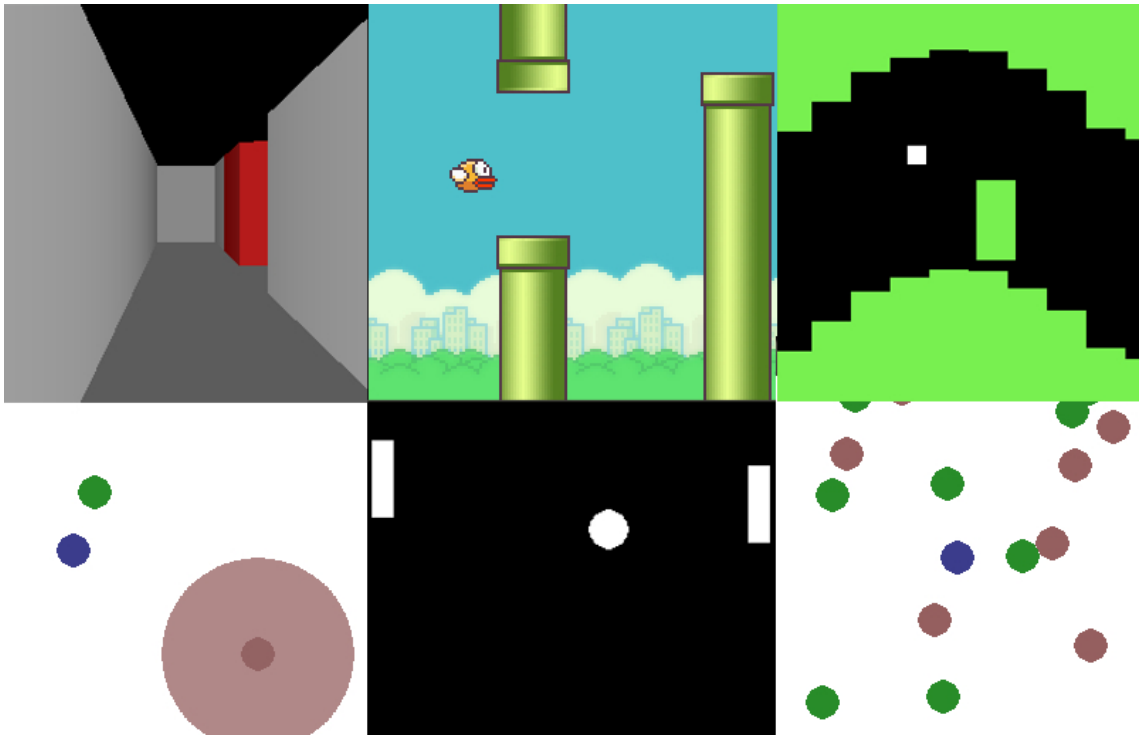


Abbildung 4.14: PyGame LE: Beispielumgebungen [TU18]

PLE wurde bislang nur unter der Version 2 von Python erfolgreich getestet, d.h. die Version 3 wird möglicherweise nicht korrekt unterstützt. Es müssen die Pakete numpy, pillow und pygame zusätzlich installiert werden. In der Python-Distribution Anaconda sind die ersten beiden bereits enthalten.

PyGame ist von Pete Shinners entwickelt worden und enthält Module für die Ein- und Ausgabe von Computerspielen [Shi18]. Damit lassen sich Eingabegeräte wie Tastatur, Maus oder Joystick abfragen und Grafik sowie Sound als Ausgabe zurückliefern. PyGame benutzt für die Ausgaben wiederum die Bibliothek Simple DirectMedia Layer (SDL).

Tabelle 4.8 fasst die wichtigsten Merkmale von PyGame LE zusammen [TU18].

Name	PyGame LE
Organisation	University of Western Ontario
Webseite	https://github.com/ntasfi/PyGame-Learning-Environment
Lizenz	MIT License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	22 ǀ 427 ǀ 93
Commits ǀ Contributors	120 ǀ 14
Issues: Open ǀ Closed	4 ǀ 16
Requests: Open ǀ Closed	1 ǀ 26
Release (Date)	-
Last Commit	28.09.2017

Tabelle 4.8: Steckbrief zur PyGame LE [TU18]

4.8 Retro Learning Environment

Retro Learning Environment (RLE) ist eine Lernumgebung, die von Nadav Bhonker, Shai Rozenberg und Itay Hubara am Technion, dem Israel Institute of Technology in Haifa, in der Programmiersprache C++ entwickelt wurde und Schnittstellen zu C++, Lua und Python anbietet [BRH16]. RLE basiert auf Arcade Learning Environment (vgl. Kap. 4.1) und LibRetro, eine API mit der Spiele und Emulatoren erstellt werden können [Lib18]. Für Atari 2600 Spiele wird der Emulator Stella eingesetzt. Für Spiele des Super Nintendo Entertainment System (SNES) wird der Emulator SNES9X verwendet, welcher bereits in LibRetro enthalten ist.

Das Super Nintendo Entertainment System, auch Super Nintendo genannt, ist eine 16-Bit-Spielekonsole, die 1990 veröffentlicht wurde [Wik18m]. Für das SNES wurden ca. 1350 Spiele entwickelt, aber nicht alle diese international vermarktet. Das meistverkaufte Spiel war Super Mario World. Andere populäre Spiele waren bspw. The Legend of Zelda: A Link to the Past, Donkey Kong Country, Street Fighter II, Super Metroid, Megaman und Secret of Mana. Diese Nintendo-Spiele sind viel komplexer als die Spiele des Atari 2600 Systems (vgl. Abb. 4.15).

Abbildung 4.15: RLE: Atari 2600 Boxing (links) und SNES Mortal Combat (rechts) [BRH16]

Beispielsweise konnten mit dem Atari-Joystick 18 unterschiedliche Aktionen gesteuert werden. Beim Nintendo-Controller sind es bereits 720 Kombinationsmöglichkeiten. Die Grafikauflösung ist ebenfalls deutlich höher: 256 x 224 bzw. 512 x 448 Pixel bei 16-Bit Farbtiefe. RLE unterstützt bereits die folgenden Nintendo-Spiele: Mortal Kombat, Super Mario All Stars, F-Zero, Castle Wolfenstein und Gradius III (vgl. Abb. 4.16).

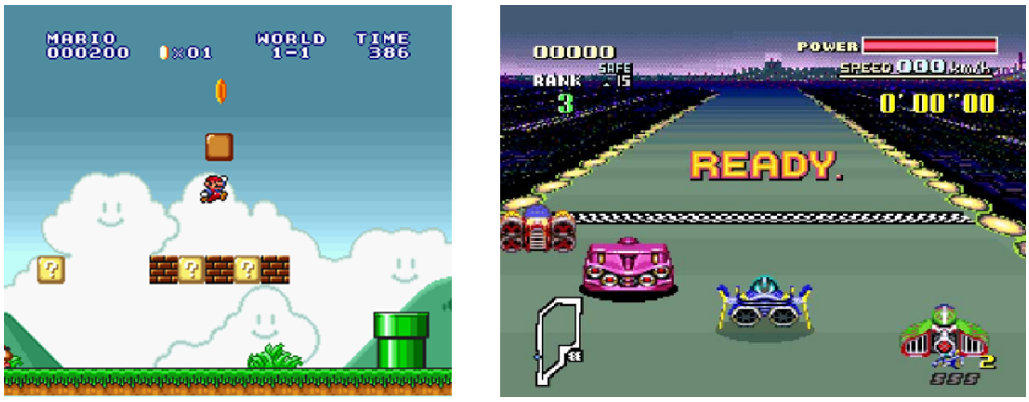


Abbildung 4.16: RLE: Super Mario (links) und F-Zero (rechts) [BRH16]

Insbesondere das Spiel Wolfenstein 3D ist noch erwähnenswert. Dieses ist 1994 das erste 3D-Spiel für Nintendo gewesen. Aus der Egoperspektive heraus muss sich der Spieler durch ein Labyrinth bewegen und dabei Feinde eliminieren. Allerdings steht das Spiel auch in der Kritik, weil die Story im Zweiten Weltkrieg angesiedelt ist und in der Spielwelt Kennzeichen verfassungswidriger Organisationen dargestellt werden. In Deutschland steht das Spiel auf dem Index für jugendgefährdende Medien.

Tabelle 4.10 fasst die wichtigsten Merkmale von RLE zusammen [BRC+18].

Name	Retro Learning Environment
Organisation	Technion (Israel)
Webseite	https://github.com/nadavbh12/Retro-Learning-Environment
Lizenz	GPL Version 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	C++
APIs	C++, Lua, Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	17 ǀ 137 ǀ 30
Commits ǀ Contributors	233 ǀ 9
Issues: Open ǀ Closed	0 ǀ 21
Requests: Open ǀ Closed	2 ǀ 12
Release (Date)	1.1.5 (22.03.2017)
Last Commit	31.01.2018

Tabelle 4.9: Steckbrief zum Retro Learning Environment [BRC+18]

4.9 Roboschool

Roboschool ist eine Erweiterung zu OpenAI Gym (vgl. Kap. 4.4), die von Oleg Klimov und Kollegen der Non-Profit Organisation OpenAI entwickelt wurde [Kli+18]. Durch diese Erweiterung werden nun auch Umgebungen im Bereich der Robotersteuerung zur Verfügung gestellt.

Ein Problem von OpenAI Gym ist, dass einige Umgebungen auf Basis der kommerziellen Software MuJoCo entwickelt wurden. Roboschool benutzt stattdessen zwei *Open Source* Physik-Simulatoren: Bullet Physics SDK [CB+18] und Dynamic Animation and Robotics Toolkit (DART) [Lee+18]. Dadurch konnten acht Umgebungen von OpenAI Gym, die auf MuJoCo basieren, durch neue von Roboschool ersetzt werden. Außerdem wird das Trainieren mehrerer Agenten in der gleichen Umgebung ermöglicht. Hierzu kann bspw. die Umgebung Multiplayer-Pong benutzt werden (vgl. Abb. 4.17).

Abbildung 4.17: Roboschool: Pong [KS17]

Aktuell sind die folgenden Umgebungen in Roboschool enthalten: InvertedPendulum, InvertedPendulumSwingup, InvertedDoublePendulum, Reacher, Hopper, Walker, HalfCheetah, Ant, Humanoid, HumanoidFlagrun, HumanoidFlagrunHarder und Pong (vgl. Abb. 4.18 und Abb. 4.19). Insbesondere der Humanoid Walker wurde gegenüber der namensgleichen Umgebung im OpenAI Gym deutlich verbessert.

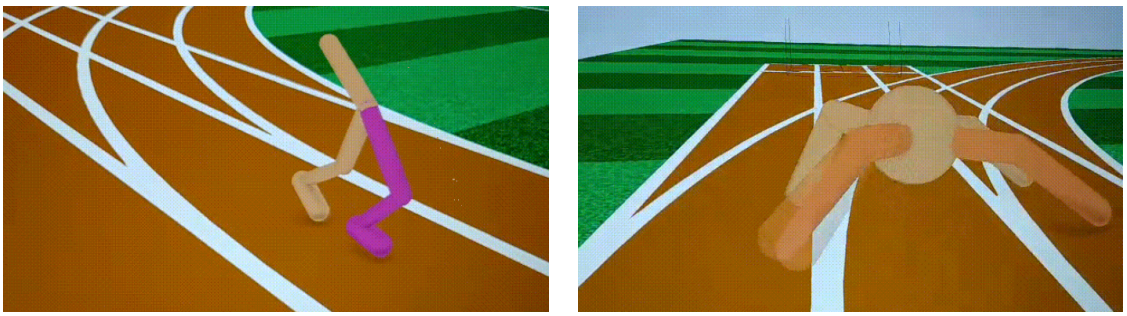


Abbildung 4.18: Roboschool: Walker (links) und Ant (rechts) [KS17]

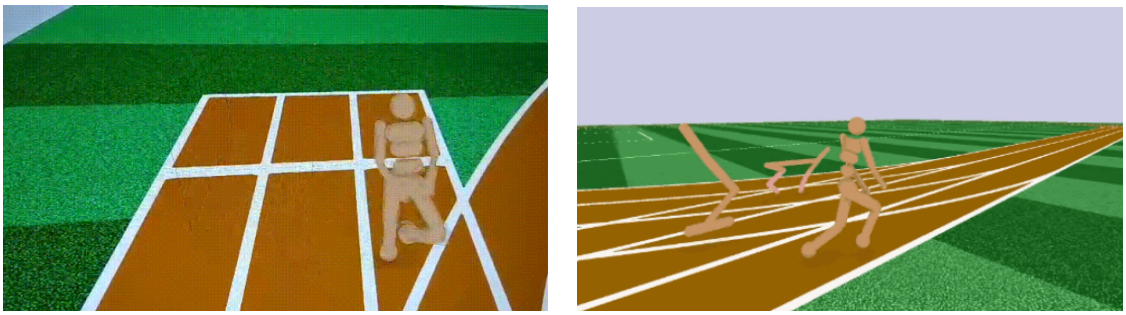


Abbildung 4.19: Roboschool: Humanoid (links) und Lauf mit drei Robotern (rechts) [KS17]

Tabelle 4.10 fasst die wichtigsten Merkmale von Roboschool zusammen [Kli+18].

Name	Roboschool
Organisation	OpenAI
Webseite	https://github.com/openai/roboschool
Lizenz	MIT License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ⁞ Star ⁞ Fork	123 ⁞ 1.022 ⁞ 194
Commits ⁞ Contributors	36 ⁞ 4
Issues: Open ⁞ Closed	32 ⁞ 74
Requests: Open ⁞ Closed	5 ⁞ 5
Release (Date)	-
Last Commit	04.12.2017

Tabelle 4.10: Steckbrief zur Roboschool [Kli+18]

4.10 ViZDoom

Die Plattform ViZDoom wurde von Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek und Wojciech Jaśkowski and der Poznań University of Technology (Polen) in C++ entwickelt [Kem+16]. ViZ steht dabei für die Anwendung von *Reinforcement Learning* aufgrund von visuellen Informationen. Diese bekommt der Agent als Bildschirmausgabe des Spiels Doom.

Doom gehört zu den Ego-Shooter-Computerspielen. Der erste Teil der Serie wurde 1993 veröffentlicht. Die 3D-Grafik-Engine konnte schrägwinklige Grundrisse umsetzen und unterschiedlich hohe, auch runde, Räume erzeugen. Damit wurden zur damaligen Zeit neue Maßstäbe im Bereich der 3D-Grafik gesetzt [Wik18d].

ViZDoom unterstützt Linux und Windows und bietet Schnittstellen für C++, Java, Lua und Python an. Die Grafikauflösung lässt sich einstellen und im synchronen Modus sind Wiederholungsraten von bis zu 7.000 Frames pro Sekunde möglich. Im asynchronen Modus kann eine Zeitraffer-Funktion benutzt werden. Episoden lassen sich unabhängig vom Modus aufzeichnen. Eine Besonderheit ist auch die Zugriffsmöglichkeit auf den *Deep Buffer* der Grafik-Engine. Dadurch kann dem Agenten weitere visuelle Informationen als Input zum Trainieren gegeben werden (vgl. Abb. 4.20).

Abbildung 4.20: ViZDoom: Normaler Output (links), Deep Buffer (rechts) [Kem+16]

Normalerweise ist der Spieler, respektive Agent, in Doom in einer Art Kampfmodus. In einem anderen Szenario namens *Health Gathering* befindet sich der Agent in einem Labyrinth, in dem der Boden durch eine Säure kontaminiert ist, die ihm Lebensenergie entzieht. Der Agent muss nun versuchen, Medizin-Kits einzusammeln, um dadurch wieder an Le-

bensenergie zu gewinnen, und er muss versuchen, blauen Phiolen, die Gift enthalten, auszuweichen (vgl. Abb. 4.21).

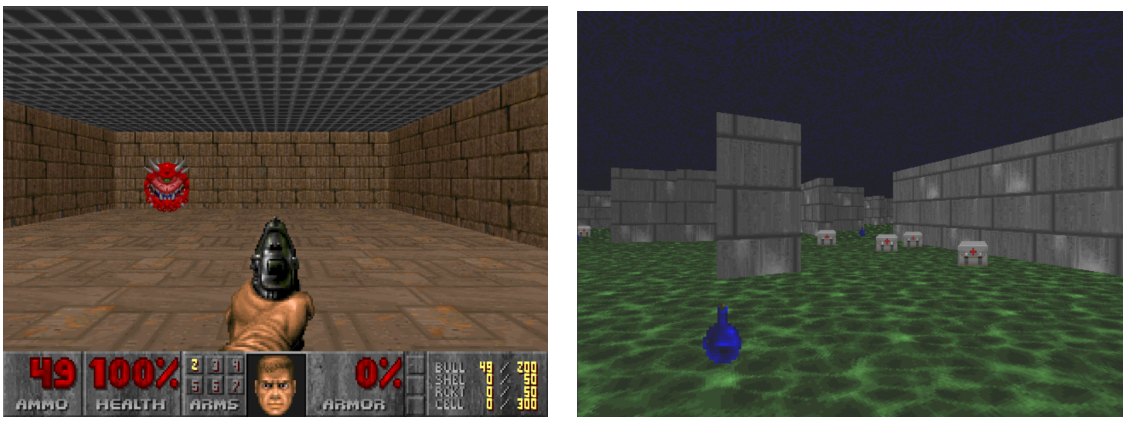


Abbildung 4.21: ViZDoom: Normalmodus (links), Health Gathering Szenario (rechts) [Kem+16]

Tabelle 4.11 fasst die wichtigsten Merkmale von ViZDoom zusammen [KW+18].

Name	ViZDoom
Organisation	Poznań University of Technology
Webseite	https://github.com/mwydmuch/ViZDoom
Lizenz	The MIT License
Plattformen	Linux, Windows
Geschrieben in	C++
APIs	C++, Java, Lua, Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	45 ǀ 564 ǀ 161
Commits ǀ Contributors	1.295 ǀ 22
Issues: Open ǀ Closed	50 ǀ 189
Requests: Open ǀ Closed	0 ǀ 37
Release (Date)	1.1.5pre (22.10.2017)
Last Commit	28.01.2018

Tabelle 4.11: Steckbrief zur ViZDoom Plattform [KW+18]

5 Bibliotheken und Frameworks

In Kap. 4 wurden zehn Softwarelösungen vorgestellt, die Umgebungen bereitstellen, in denen sich Softwareagenten beweisen können. In diesem Kapitel werden nun 13 Bibliotheken und Frameworks zum Thema *Reinforcement Learning* vorgestellt, die selbst zwar keine Umgebungen enthalten, dafür aber diejenigen aus Kap. 4 verwenden können. Der Fokus dieser Softwarelösungen liegt auf Agenten und Algorithmen. Alle Softwarelösungen benutzen als Basis eine andere Bibliothek oder ein anderes Framework und sind auf der Plattform GitHub vorhanden.

Tabelle 5.1 zeigt eine Übersicht dieser Bibliotheken und Frameworks in alphabetischer Reihenfolge. Die *Stars* geben wieder die insgesamt erhaltenen positiven Rückmeldungen bis zum 02.02.2018 von der GitHub-Community an.

Nr	Name	Stars	Organisation	Basis
1	Chainer RL	395	Preferred Networks	Chainer
2	Coach	601	Intel AI Labs	TensorFlow, Neon
3	Deep RL	353	University of Alberta	PyTorch
4	Keras-rl	2.127	KIT	Keras
5	OpenAI Baselines	3.245	OpenAI	TensorFlow, Theano
6	OpenAI Lab	229	Open Community	TensorFlow, Keras
7	Ray RLlib	1.847	UC Berkeley	TensorFlow, PyTorch
8	RL4J	226	Skymind	DL4J (Java)
9	RL Lab	1.455	UC Berkeley, OpenAI	TensorFlow, Theano
10	TF Agents	612	Google Brain	TensorFlow
11	TensorForce	1.229	University of Cambridge	TensorFlow
12	TensorLayer	3.129	Imperial College London	TensorFlow
13	Torch-twrl	259	Twitter	Torch (Lua)

Tabelle 5.1: Übersicht ausgewählter RL-Softwarelösungen auf der Plattform GitHub [Git18]

Die meisten Softwarelösungen basieren auf TensorFlow, eine *Open Source* Bibliothek für numerisches Rechnen, die von Mitarbeitern des Google Brain Teams entwickelt wurde [Ten18]. Tensoren sind mehrdimensionale Arrays, in einer Dimension spricht man von Vektoren, in zwei Dimensionen von Matrizen. Eine Matrix-Multiplikation ist eine mathematische Operation der linearen Algebra, die häufig in Optimierungsproblemen vorkommt. In TensorFlow werden alle mathematischen Operationen graphenbasiert formuliert. Jeder Knoten repräsentiert dabei eine mathematische Operation und jede Kante ein Tensor. Der Fluss (engl. *Flow*) durch den Graphen spiegelt dann die Berechnung wieder. Diese lassen sich auf Prozessoren (CPUs) oder Grafikprozessoren (GPUs) ausführen. TensorFlow unterstützt insbesondere *Deep Learning (DL)*. Das sind Lernverfahren, bei denen tiefe Künstliche Neuronale Netzwerke (KNN) trainiert werden. TensorFlow bietet

in erster Linie eine *Application Programming Interface (API)* für die Programmiersprache Python an, aber auch C++, Java und Go werden bereits rudimentär unterstützt (vgl. Abb. 5.1). Außerdem werden *Estimators* zur Verfügung gestellt. Dies sind bereits komplette *Machine Learning* Modelle, die sehr einfach konfiguriert und trainiert werden können. Mit wenig Aufwand lassen sich so bereits Ergebnisse erzielen: Gütekriterien zur Beurteilung der Modellqualität und Prognosen zur Anwendung des Modells.

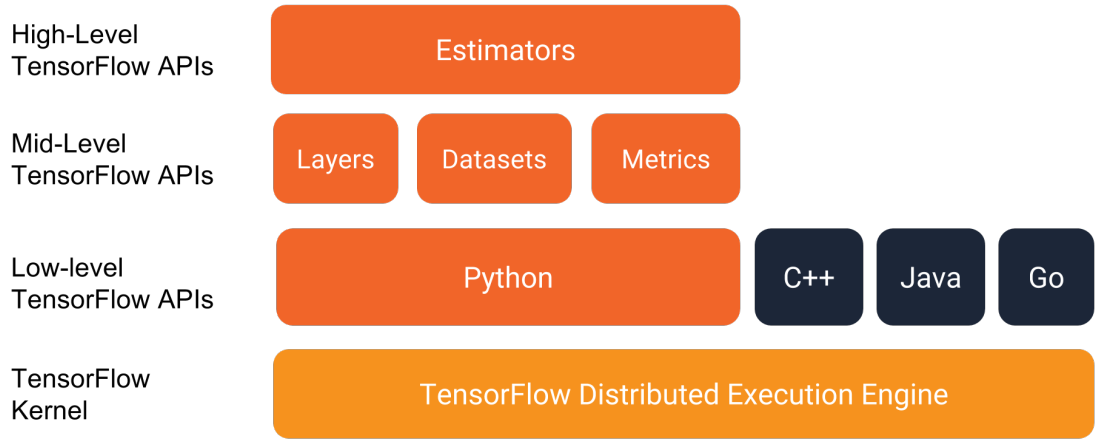


Abbildung 5.1: TensorFlow Programmierumgebung [Ten18]

Ähnliche Softwarelösungen wie TensorFlow sind Chainer [Net18], DL4J [GNP+18], Keras [Cho+18], Neon [Par+18], Theano [Ben+17] und Torch [Col+18] bzw. PyTorch [PyT18]. Mit Ausnahme von DL4J und Torch unterstützen diese Lösungen ebenfalls Python als API. DL4J benutzt Java und Torch setzt auf Lua als Programmiersprache.

In den nachfolgenden Kapiteln werden die ausgewählten Softwarelösungen aus Tab. 5.1 nacheinander kurz vorgestellt. Der Schwerpunkt dabei liegt auf der Aufzählung der implementierten RL-Algorithmen. Die meisten dieser Lösungen sind entweder an einer Universität oder von Forschungsgruppen namhafter Organisationen wie Intel, Google, OpenAI und Twitter entwickelt worden. Die innerhalb der GitHub-Community beliebtesten Lösungen sind OpenAI Baselines und TensorLayer.

5.1 ChainerRL

ChainerRL ist eine *Deep Reinforcement Learning (DRL)* Bibliothek, die von Yasuhiro Fujita, Toshiki Kataoka, Yuta Kikuchi u.a. bei Preferred Networks entwickelt wurde [FK+18]. Sie basiert auf dem flexiblen DL-Framework Chainer [Tok+15][Net18]. Die folgenden Algorithmen sind in Python implementiert:

1 Asynchronous Advantage Actor Critic (A3C)
2 Actor-Critic with Experience Replay (ACER)
3 Asynchronous N-step Q-Learning (NSQ)
4 Deep Q-Network (DQN), inkl.
  Double Deep Q-Network (DDQN), Persistent Advantage Learning (PAL), Double PAL, Dynamic Policy Programming (DPP)
5 Deep Deterministic Policy Gradients (DDPG), inkl.
  Stochastic Value Gradients (SVG)(0)
6 Policy Gradient Theorem (PGT)
7 Path Consistency Learning (PCL)
8 Proximal Policy Optimization (PPO)

Mit Ausnahme von DDPG können alle Algorithmen auf diskrete Probleme angewendet werden. Mit Hilfe der Normalized Advantage Function (NAF) können Algorithmen, die auf Q-Funktionen basieren, auch zur Lösung von kontinuierlichen Problemen verwendet werden: bspw. DQN und NSQ. Es werden Umgebungen von ALE (vgl. Kap. 4.1) und OpenAI Gym (vgl. Kap. 4.4) unterstützt. ChainerRL funktioniert sowohl mit Version 2 als auch mit Version 3 von Python. Zusätzlich müssen hierzu die folgenden Pakete installiert werden: cached-property, future, numpy, pillow und scipy. Diese sind aber bereits in der Python-Distribution Anaconda enthalten.

Tabelle 5.2 fasst die wichtigsten Merkmale der ChainerRL-Bibliothek zusammen [FK+18].

Name	ChainerRL
Organisation	Preferred Networks
Webseite	https://github.com/chainer/chainerrl
Lizenz	MIT License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	65 ǀ 395 ǀ 86
Commits ǀ Contributors	1.482 ǀ 9
Issues: Open ǀ Closed	40 ǀ 44
Requests: Open ǀ Closed	21 ǀ 120
Release (Date)	0.3.0 (08.12.2017)
Last Commit	29.01.2018

Tabelle 5.2: Steckbrief zur ChainerRL-Bibliothek [FK+18]

5.2 Coach

Coach ist ein *DRL*-Framework, das ursprünglich von der Firma Nervana Systems entwickelt wurde. 2016 wurde Nervana Systems vom US-amerikanischen Chipherstellers Intel übernommen. Das Framework ist so konstruiert, das seine Komponenten (Umgebungen, Algorithmen, Exploration Policies, Netzwerk-Architekturen usw.) sorgfältig entkoppelt sind (vgl. Abb. 5.2). Die Architektur ist also flexibel und erweiterungsfähig, d.h. Komponenten können leicht ausgetauscht und wiederverwendet werden.

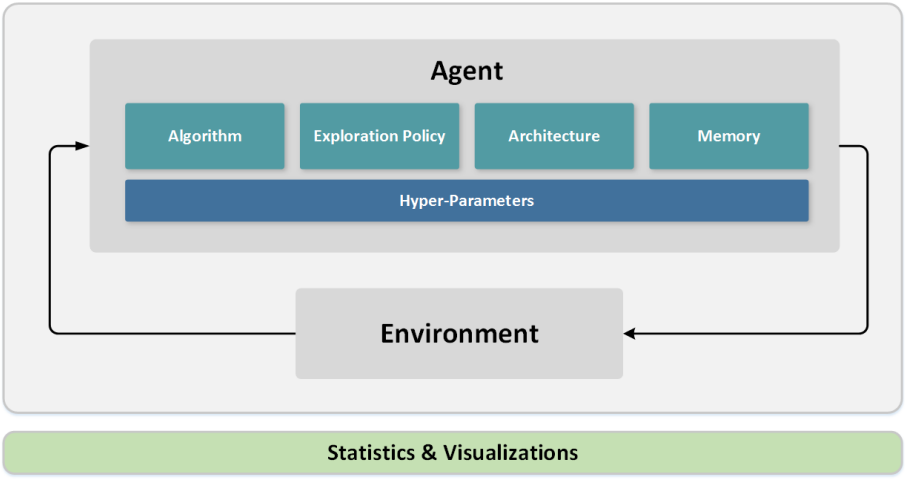


Abbildung 5.2: Architektur des Coach-Frameworks [CLN+18a]

Viele *State-of-the-Art* Algorithmen sind in Python programmiert (vgl. Abb. 5.3).

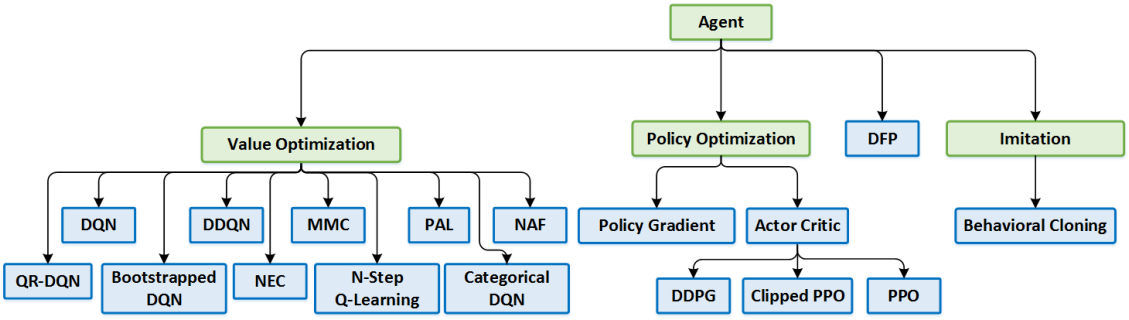


Abbildung 5.3: Algorithmen des Coach-Frameworks [CLN+18a]

Einige dieser Algorithmen lassen sich parallelisieren und als Form des Verteilten Rechnens (engl. *Distributed Computing (DC)*) ausführen:

1. Deep Q-Network (DQN)
2. Double Deep Q-Network (DDQN)
3. Dueling DQN
4. Mixed Monte Carlo (MMC)
5. Persistent Advantage Learning (PAL)
6. Categorical Deep Q-Network of 51-Atom Agent (C51)
7. Quantile Regression (QR)-DQN
8. Bootstrapped DQN

9	N-step Q-Learning (NSQ)	DC
10	Neural Episodic Control (NEC)	
11	Normalized Advantage Function (NAF)	DC
12	Policy Gradient Theorem (PGT)	DC
13	Asynchronous Advantage Actor Critic (A3C)	DC
14	Deep Deterministic Policy Gradients (DDPG)	DC
15	Proximal Policy Optimization (PPO)	
16	Clipped PPO	DC
17	Direct Future Prediction (DFP)	DC
18	Behavioral Cloning (BC)	

Die Umgebungen von OpenAI Gym (vgl. Kap. 4.4), Roboschool (vgl. Kap. 4.9) und ViZDoom (vgl. Kap. 4.10) werden unterstützt sowie Gym-Extensions [Hen+18], eine Erweiterung zu OpenAI Gym. PyBullet kann als Schnittstelle zur Simulationssoftware Bullet Physics SDK eingesetzt werden [CB+18]. Mit dem Simulator CARLA kann sogar Forschung im Bereich des autonomen Fahrens betrieben werden (vgl. Kap. 4.2).

Coach verwendet TensorFlow [Ten18] und Neon [Par+18]. Letzteres ist eine Bibliothek für *Deep Learning* aus den Intel AI Labs, mit der sich tiefe Künstliche Neuronale Netzwerke sowohl auf CPUs als auch auf GPUs trainieren lassen.

Tabelle 5.3 fasst die wichtigsten Merkmale der Coach-Frameworks zusammen [CLN+18b].

Name	Coach
Organisation	Nervana Systems (Intel)
Webseite	https://github.com/NervanaSystems/coach
Lizenz	Apache License 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	68 ǀ 601 ǀ 91
Commits ǀ Contributors	69 ǀ 10
Issues: Open ǀ Closed	6 ǀ 25
Requests: Open ǀ Closed	1 ǀ 19
Release (Date)	0.9.0 (19.12.2017)
Last Commit	27.01.2018

Tabelle 5.3: Steckbrief zum Coach-Framework [CLN+18b]

5.3 DeepRL

DeepRL ist eine *Deep Reinforcement Learning (DRL)* Bibliothek, die von Shangtong Zhang, einem Master-Studenten und Assistenten von Richard S. Sutton an der Universität von Alberta erstellt wurde [ZC18]. Die Bibliothek basiert auf PyTorch, einem Python-Paket für Tensor-Berechnungen auf tiefen KNN mit GPU-Unterstützung und [PyT18]. Die folgenden Algorithmen sind in DeepRL implementiert [ZC18]:

1. Asynchronous Advantage Actor Critic (A3C)
2. Asynchronous One-Step Q-Learning
3. Async One-Step SARSA
4. Asynchronous N-step Q-Learning (NSQ)
5. Continuous A3C
6. Deep Q-Network (DQN)
7. Double Deep Q-Network (DDQN)
8. Dueling DQN
9. Distributed Deep Deterministic Policy Gradient (D3PG)
10. Hybrid Reward Architecture (HRA)
11. Parallelized Proximal Policy Optimization (P3O)
12. Action Conditional Video Prediction

Umgebungen von OpenAI Gym (vgl. Kap. 4.4) und Roboschool [Kli+18] werden unterstützt. DeepRL ist kompatibel mit beiden Python-Varianten: Version 2 und 3.

Tabelle 5.4 fasst die wichtigsten Merkmale der DeepRL-Bibliothek zusammen [ZC18].

Name	DeepRL
Organisation	Shangtong Zhang
Webseite	https://github.com/ShangtongZhang/DeepRL
Lizenz	Apache License V 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	23 ǀ 353 ǀ 66
Commits ǀ Contributors	124 ǀ 2
Issues: Open ǀ Closed	1 ǀ 14
Requests: Open ǀ Closed	2 ǀ 6
Release (Date)	-
Last Commit	19.01.2018

Tabelle 5.4: Steckbrief zur DeepRL-Bibliothek [ZC18]

5.4 Keras-RL

Keras-RL ist eine *Deep Reinforcement Learning (DRL)* Bibliothek, die Matthias Plappert am Karlsruher Institut für Technologie (KIT) auf Basis von Keras [Cho+18] erstellt hat. Die folgenden Algorithmen sind in Python implementiert [Pla18a]:

1 Deep Q-Network (DQN)
2 Double Deep Q-Network (DDQN)
3 Dueling DQN
4 Continuous Deep Q-Network (CDQN)
bzw. Normalized Advantage Function (NAF)
5 Deep Deterministic Policy Gradients (DDPG)
6 Cross-Entropy Method (CEM)
7 Deep SARSA
8 Asynchronous Advantage Actor Critic (A3C)

Keras selbst benötigt wiederum als Basis Theano [Ben+17] oder TensorFlow [Ten18]. Somit lassen sich alle Algorithmen auf CPUs oder GPUs trainieren. Des Weiteren wird OpenAI Gym (vgl. Kap. 4.4) unterstützt. Keras-RL funktioniert sowohl mit Version 2 als auch mit Version 3 von Python.

Tabelle 5.5 fasst die wichtigsten Merkmale der Keras-RL-Bibliothek zusammen [Pla18a].

Name	Keras-RL
Organisation	Karlsruher Institut für Technologie (KIT)
Webseite	https://github.com/matthiasplappert/keras-rl
Lizenz	MIT License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch | Star | Fork	151 | 2.127 | 489
Commits | Contributors	219 | 22
Issues: Open | Closed	41 | 47
Requests: Open | Closed	11 | 66
Release (Date)	0.4.0 (04.12.2017)
Last Commit	04.12.2017

Tabelle 5.5: Steckbrief zur Keras-RL-Bibliothek [Pla18a]

5.5 OpenAI Baselines

OpenAI Baselines ist eine Bibliothek bzw. Sammlung von *Deep Reinforcement Learning* Algorithmen, die in Python Version 3 implementiert wurden und von Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, Yuan-Hong Liao und weiteren Mitgliedern der Non-Profit-Organisation OpenAI bereitgestellt werden [Dha+18]:

1 Advantage Actor Critic (A2C)
2 Actor-Critic with Experience Replay (ACER)
3 Actor Critic using Kronecker-Factored Trust Region (ACKTR)
4 Deep Deterministic Policy Gradients (DDPG)
5 Deep Q-Network (DQN)
6 Generative Adversarial Imitation Learning (GAIL)
7 Proximal Policy Optimization (PPO) v1 — Multi-CPU per MPI
8 Proximal Policy Optimization (PPO) v2 — GPU
9 Trust Region Policy Optimization (TRPO)

Umgebungen von OpenAI Gym (vgl. Kap. 4.4) und Roboschool [Kli+18] werden in den Beispielen verwendet. Außerdem wird die *Deep Learning* Bibliothek TensorFlow [Ten18] eingesetzt.

Tabelle 5.6 fasst die wichtigsten Merkmale der OpenAI-Baselines-Bibliothek zusammen [Dha+18].

Name	OpenAI Baselines
Organisation	OpenAI
Webseite	https://github.com/openai/baselines
Lizenz	MIT License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch | Star | Fork	286 | 3.245 | 791
Commits | Contributors	98 | 32
Issues: Open | Closed	88 | 99
Requests: Open | Closed	38 | 49
Release (Date)	-
Last Commit	29.01.2018

Tabelle 5.6: Steckbrief zur OpenAI-Baselines-Bibliothek [Dha+18]

5.6 OpenAI Lab

OpenAI Lab ist ein Framework für *Deep Reinforcement Learning*, das von Wah Loon Keng und Laura Graesser entwickelt wurde [KG18a]. Als Basis werden TensorFlow [Ten18] und Keras [Cho+18] verwendet. Der Einsatz von PyTorch [PyT18] ist geplant. Die folgenden Algorithmen sind bereits oder werden demnächst in Python implementiert:

1 Deep Q-Network (DQN)
2 Double Deep Q-Network (DDQN)
3 Deep SARSA
4 Off-Policy SARSA
5 Prioritized Experience Replay (PER)
6 Deterministic Policy Gradients (DPG) Off-Policy Actor-Critic
7 Deep Deterministic Policy Gradients (DDPG) Actor-Critic with Target Networks
8 Cross-Entropy Method (CEM) demnächst
9 Dyna demnächst

Geplant sind Implementierungen der folgenden Algorithmen: Dueling DQN, Asynchronous Advantage Actor Critic (A3C), Trust Region Policy Optimization (TRPO), Q*(λ), Retrace(λ), Neural Episodic Control (NEC) und Elastic Weight Consolidation (EWC). Eine Schnittstelle zu OpenAI Gym (vgl. Kap. 4.4) ist vorhanden. Das Framework unterstützt beim Einrichten von Experimenten und Durchführen von Parameterstudien und Optimierungen mit Hilfe von Protokollierungen (Komponente *Logging*), Visualisierungen (Komponente *Plots*) und Analysen (Komponente *Analytics*). Der jeweilige Versuchsaufbau wird in einem standardisierten JSON-Format gespeichert: Umgebung, Agent, Algorithmus, Parametereinstellungen usw. Die *Fitness Matrix* enthält die besten Resultate der durchgeführten Experimente für die Umgebungen und Algorithmen.

Tabelle 5.7 fasst die wichtigsten Merkmale von OpenAI Lab zusammen [KG18a].

Name	OpenAI Lab
Organisation	-
Webseite	https://github.com/kengz/openai_lab
Lizenz	MIT License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	21 ǀ 229 ǀ 41
Commits ǀ Contributors	1.555 ǀ 2
Issues: Open ǀ Closed	0 ǀ 3
Requests: Open ǀ Closed	2 ǀ 132
Release (Date)	1.0.5 (27.04.2017)
Last Commit	07.10.2017

Tabelle 5.7: Steckbrief zum OpenAI Lab Framework [KG18a]

5.7 Ray RLlib

Ray ist ein flexibles und hochperformantes Framework für verteiltes Rechnen, das von Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan und Ion Stoicaan an der Kalifornischen Universität Berkeley entwickelt wurde [Mor+17]. Die Systemschicht von Ray ist in C++ (ca. 72 % des Quelltextes) und die Applikationsschicht in Python (ca. 28 % des Quelltextes) programmiert. Mit Ray RLlib steht eine skalierbare *Deep Reinforcement Learning* Bibliothek zur Verfügung, in der die folgenden implementierten Algorithmen enthalten sind [Lia+18a]:

1 Proximal Policy Optimization (PPO)
   spez. Variante von Trust Region Policy Optimization (TRPO)
2 Asynchronous Advantage Actor Critic (A3C)
3 Deep Q-Network (DQN)
4 Evolution Strategies (ES)

Diese Algorithmen lassen sich in einer Umgebung von OpenAI Gym (vgl. Kap. 4.4) ausführen. Des Weiteren lässt sich Ray Tune integrieren. Dies ist ein Werkzeug, um Parameterstudien effizient durchzuführen (*Hyperparameter Optimization*). Außerdem werden andere Frameworks und Bibliotheken wie bspw. TensorFlow [Ten18] und PyTorch [PyT18] unterstützt.

Tabelle 5.8 fasst die wichtigsten Merkmale der Ray-Bibliothek zusammen [Nis+18].

Name	Ray
Organisation	UC Berkeley
Webseite	https://github.com/ray-project/ray
Lizenz	Apache License 2.0
Plattformen	Linux, MacOSX
Geschrieben in	C++, Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	131 ǀ 1.847 ǀ 285
Commits ǀ Contributors	1.350 ǀ 39
Issues: Open ǀ Closed	172 ǀ 392
Requests: Open ǀ Closed	37 ǀ 901
Release (Date)	0.3.0 (28.11.2017)
Last Commit	02.02.2018

Tabelle 5.8: Steckbrief zum Framework Ray [Nis+18]

5.8 RL4J

RL4J ist eine *Deep Reinforcement Learning* Bibliothek, die von Ruben Fiszel während seines zwei monatigen Praktikums bei der Firma Skymind 2016 entwickelt wurde [FAG+18]. Sie gehört zu DL4J [GNP+18], einer Bibliothek zum Thema *Deep Learning*. Der Buchstabe J in Namen gibt bereits an, dass die Bibliotheken für Java entwickelt wurden. Einige der zugrunde liegenden Berechnungen sind aufgrund von Geschwindigkeitsvorteilen allerdings auch in C/C++ implementiert. Mit Hilfe von Keras [Cho+18] verfügt DL4J über eine API zur Programmiersprache Python. Zwei RL-Algorithmen sind bisher implementiert:

1. Deep Q-Network (DQN)
2. Asynchronous Advantage Actor Critic (A3C)

Zur Installation von RL4J kann Apache Maven, ein Werkzeug zu Bauen von Software, benutzt werden [Fou18]. Umgebungen von Project Malmo (vgl. Kap. 4.6) lassen sich einsetzen, da eine API für Java vorhanden ist. OpenAI Gym (vgl. Kap. 4.4) lässt sich ebenfalls verwenden, hierzu muss allerdings `gym-http-api` installiert und konfiguriert werden: Dies ist ein HTTP-Server, der eine REST-API zur Verfügung stellt, um mit OpenAI Gym in anderen Sprachen als Python zu kommunizieren [Ols+18].

Tabelle 5.9 fasst die wichtigsten Merkmale der RL4J-Bibliothek zusammen [FAG+18].

Name	RL4J
Organisation	Skymind
Webseite	https://github.com/deeplearning4j/rl4j
Lizenz	Apache License, Version 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Java
APIs	Java
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	48 ǀ 226 ǀ 84
Commits ǀ Contributors	182 ǀ 17
Issues: Open ǀ Closed	10 ǀ 37
Requests: Open ǀ Closed	1 ǀ 38
Release (Date)	0.9.2 (08.12.2017)
Last Commit	01.02.2018

Tabelle 5.9: Steckbrief zur RL4J-Bibliothek [FAG+18]

5.9 RLLab

Die Bibliothek RLLab wurde von Rocky Duan, Peter Chen, Rein Houthooft, John Schulman und Pieter Abbeel an der Kalifornischen Universität Berkeley in Zusammenarbeit mit OpenAI entwickelt [Dua+16]. Die folgenden Algorithmen sind in Python (Version 2 und 3) implementiert:

1. REINFORCE
2. Truncated Natural Policy Gradient
3. Reward-Weighted Regression
4. Relative Entropy Policy Search
5. Trust Region Policy Optimization (TRPO)
6. Cross-Entropy Method (CEM)
7. Covariance Matrix Adaption Evolution Strategy
8. Deep Deterministic Policy Gradients (DDPG)

Die Hauptmodule benutzen Theano [Ben+17] als zugrundeliegendes Framework. Außerdem wird TensorFlow [Ten18] unterstützt. Die Umgebungen von OpenAI Gym (vgl. Kap. 4.4) lassen sich verwenden. Berechnungen auf Amazons EC2 Cluster werden ebenfalls unterstützt. Außerdem sind Werkzeuge zur Visualisierung der Ergebnisse vorhanden.

Tabelle 5.10 fasst die wichtigsten Merkmale der RL Lab Bibliothek zusammen [DP+18].

Name	RLLab
Organisation	UC Berkeley, OpenAI
Webseite	https://github.com/rll/rllab
Lizenz	MIT License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	149 ǀ 1.455 ǀ 457
Commits ǀ Contributors	142 ǀ 30
Issues: Open ǀ Closed	76 ǀ 74
Requests: Open ǀ Closed	6 ǀ 57
Release (Date)	-
Last Commit	20.11.2017

Tabelle 5.10: Steckbrief zur Bibliothek RL Lab [DP+18]

5.10 TensorFlow Agents

TensorFlow Agents ist eine *Reinforcement Learning* Bibliothek auf Basis von TensorFlow [Ten18], die von Danijar Hafner, James Davidson und Vincent Vanhoucke des Forschungsprojekts Google Brain entwickelt wurde [HDV17]. Es ist aber kein offizielles Produkt von Google. Es wurde bisher nur ein Algorithmus in Python implementiert [Haf+18]:

1 Batch PPO (Proximal Policy Optimization)

Dieser Algorithmus ist eine spezielle, optimierte Version von PPO. Implementierungen von weiteren Algorithmen sollen folgen. Die Quellcodes von TensorFlow Agents sind sowohl für Version 2 als auch für Version 3 von Python ausführbar. Das Paket ruamel_yaml muss ggf. zusätzlich installiert werden, es ist zumindest schon in der Python-Distribution Anaconda enthalten. Die Schnittstelle von OpenAI Gym (vgl. Kap. 4.4) wurde erweitert, sodass auch mehrere parallele Umgebungen benutzt werden können.

Tabelle 5.11 fasst die wichtigsten Merkmale der TensorFlow-Agents-Bibliothek zusammen [Haf+18].

Name	TensorFlow Agents
Organisation	Google Brain
Webseite	https://github.com/tensorflow/agents
Lizenz	Apache License 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	63 ǀ 612 ǀ 88
Commits ǀ Contributors	37 ǀ 9
Issues: Open ǀ Closed	3 ǀ 9
Requests: Open ǀ Closed	0 ǀ 15
Release (Date)	1.3.0 (18.01.2018)
Last Commit	29.01.2018

Tabelle 5.11: Steckbrief zur Bibliotkek TensorFlow Agents [Haf+18]

5.11 TensorForce

TensorForce ist eine *Deep Reinforcement Learning* Bibliothek, die auf Basis von TensorFlow [Ten18] von den Doktoranten Michael Schaarschmidt und Alexander Kuhnle (jeweils Universität von Cambridge) sowie Kai Fricke (Helmut-Schmidt-Universität) entwickelt wurde [SKF+18]. Die folgenden Algorithmen und Agenten sind in Python implementiert:

1	Asynchronous Advantage Actor Critic (A3C)	verschiedene Agenten
2	Trust Region Policy Optimization (TRPO)	`trpo_agent`
3	Normalized Advantage Function (NAF)	`naf_agent`
4	Deep Q-Network (DQN)	`dqn_agent`
5	Double Deep Q-Network (DDQN)	`ddqn_agent`
6	N-step DQN	`dqn_nstep_agent`
7	Vanilla Policy Gradient (VPG), REINFORCE	`vpg_agent`
8	Deep Q-Learning from Demonstration (DQFD)	
9	Proximal Policy Optimization (PPO)	`ppo_agent`
10	Random Agent	`random_agent`
11	Constant Agent	`constant_agent`

TensorForce ist kompatibel mit Python in den Versionen 2 und 3. Der Hauptunterschied zu anderen DRL-Bibliotheken ist die strenge Trennung zwischen Umgebung, Agent und Update-Logik. Deshalb werden auch die Umgebungen von DeepMind Lab (vgl. Kap. 4.3), OpenAI Gym (vgl. Kap. 4.4), OpenAI Universe (vgl. Kap. 4.5), Arcade Learning Environment (vgl. Kap. 4.1) und Maze Explorer unterstützt.

Tabelle 5.12 fasst die wichtigsten Merkmale von TensorForce zusammen [SKF+18].

Name	TensorForce
Organisation	University of Cambridge
Webseite	https://github.com/reinforceio/tensorforce
Lizenz	Apache License V 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch | Star | Fork	107 | 1.229 | 199
Commits | Contributors	1.398 | 26
Issues: Open | Closed	29 | 174
Requests: Open | Closed	1 | 117
Release (Date)	0.3.5.1 (13.01.2018)
Last Commit	29.01.2018

Tabelle 5.12: Steckbrief zur TensorForce-Bibliothek [SKF+18]

5.12 TensorLayer

TensorLayer ist eine *Deep Reinforcement Learning* Bibliothek, die auf Basis von TensorFlow [Ten18] vom Doktoranten Hao Dong am Imperial College London mit Hilfe von Kollegen und einer Open Community entwickelt wurde [Don+17]. Der Schwerpunkt dieser Bibliothek liegt eher im Bereich DL und weniger im Bereich RL. Die folgenden Algorithmen sind in Python jeweils für entsprechende Umgebungen implementiert [DM+18]:

1	Policy Gradient Network (PGN)	Atari Pong
2	Deep Q-Network (DQN)	Frozen Lake
3	Q-Learning	Frozen Lake
4	Asynchronous Policy Gradient (APG)	Atari Pong
5	Actor Critic (AC)	Cartpole
6	Asynchronous Advantage Actor Critic (A3C)	Bipedal Walker

Die Umgebungen stammen aus dem OpenAI Gym (vgl. 4.4). Neben TensorFlow müssen die Pakete numpy und matplotlib installiert werden. Diese sind aber bereits in der Python-Distribution Anaconda enthalten.

Tabelle 5.13 fasst die wichtigsten Merkmale der TensorLayer-Bibliothek zusammen [DM+18].

Name	TensorLayer
Organisation	Imperial College London
Webseite	https://github.com/tensorlayer/tensorlayer
Lizenz	Apache License 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch | Star | Fork	397 | 3.129 | 776
Commits | Contributors	1.539 | 51
Issues: Open | Closed	40 | 132
Requests: Open | Closed	2 | 121
Release (Date)	1.7.4 (01.02.2018)
Last Commit	02.02.2018

Tabelle 5.13: Steckbrief zur TensorLayer-Bibliothek [DM+18]

5.13 Torch-TWRL

Torch-TWRL ist eine *Reinforcement Learning* Bibliothek, die auf Basis des Frameworks Torch [Col+18] von Kory Mathewson u.a. bei Twitter entwickelt wurde [Mat+18]. Torch stellt viele Algorithmen aus dem Bereich des Maschinenlernens bereit, unterstützt dabei das Rechnen auf GPUs und bietet mit der imperativen Skriptsprache Lua eine API, die ansonsten oft bei der Entwicklung von Computerspielen zum Einsatz kommt. Denn computergesteuerte Spielcharaktere können bspw. mittels KI-Algorithmen gesteuert werden. Die folgenden Agenten bzw. Algorithmen sind in Torch-TWRL vorhanden [Mat+18]:

1	Random Agent	
1	Temporal Difference (TD)	Q-Learning, SARSA
2	Policy Gradient Network (PGN)	Multi-Layer Perceptron (MLP), Softmax

Zur Installation von Torch-TWRL wird der Paketmanager LuaRocks benötigt [Lua18]. Die Umgebungen von OpenAI Gym (vgl. Kap. 4.4) lassen sich verwenden, wobei hierzu `gym-http-api` installiert und konfiguriert werden muss (vgl. Kap. 5.8).

Tabelle 5.14 fasst die wichtigsten Merkmale von Torch-TWRL zusammen [Mat+18].

Name	Torch-TWRL
Organisation	Twitter
Webseite	https://github.com/twitter/torch-twrl
Lizenz	MIT License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Lua
APIs	Lua
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	42 ǀ 259 ǀ 47
Commits ǀ Contributors	38 ǀ 3
Issues: Open ǀ Closed	8 ǀ 12
Requests: Open ǀ Closed	1 ǀ 16
Release (Date)	0.1 (16.09.2016)
Last Commit	30.01.2017

Tabelle 5.14: Steckbrief zur Bibliothek Torch-TWRL [Mat+18]

6 Sonstige

In diesem Kapitel werden zunächst fünf Softwarelösungen zum Thema *Reinforcement Learning* vorgestellt, die sich nicht in die Kategorie Umgebung (vgl. Kap. 4) oder Bibliothek bzw. Framework von Agenten / Algorithmen (vgl. Kap. 5) einsortieren lassen. Diese Softwarelösungen beinhalten entweder beide Komponenten, also sowohl eigene Umgebungen als auch Agenten und Algorithmen, oder sie sind Speziallösungen zu besonderen RL-Problemen.

Tabelle 6.1 zeigt eine Übersicht dieser Softwarelösungen in alphabetischer Reihenfolge. Die *Stars* entsprechen positiven Bewertungen der GitHub-Community (Stand: 02.02.2018).

Nr	Name	Stars	Organisation	API
1	BURLAP	205	Brown University & UMBC	Java
2	MAgent	553	Shanghai Jiao Tong University	Python
3	PyBrain	2.454	TU München, IDSIA	Python
4	RLPark	21	INRIA	Java
5	RLPy	179	MIT & Carnegie Mellon Univ.	Python

Tabelle 6.1: Übersicht spezieller RL-Lösungen auf der Plattform GitHub [Git18]

Außerdem werden drei weitere Softwarelösungen in Tabelle 6.2 präsentiert, die sich jedoch nicht auf der Plattform GitHub befinden, trotzdem aber zum Thema *Reinforcement Learning* entwickelt worden sind.

Nr	Name	Stars	Organisation	API
6	MMLF	-	DFKI & Uni Bremen	Python
7	RL-Glue	-	University of Alberta	Diverse
8	RLToolkit	-	University of Alberta	Python

Tabelle 6.2: Übersicht sonstiger RL-Lösungen

Alle Lösungen haben gemein, dass Sie an Universitäten oder staatlich geförderten Forschungseinrichtungen entwickelt worden sind, viele davon in einer Kooperation. Die meisten dieser Lösungen verwenden wieder einmal die Programmiersprache Python als API.

6.1 BURLAP

Brown-UMBC Reinforcement Learning and Planning (BURLAP) ist eine in Java geschriebene Software-Bibliothek. Ursprünglich wurde diese von James MacGlashan während seiner Dissertation an der University of Maryland, Baltimore County (UMBC) entwickelt [Mac13]. Anschließend wechselte er zur Brown University, wo der Code seitdem auch von ihm und seinen Mitarbeitern und Studierenden weitergepflegt wird. Die Bibliothek zeichnet sich dadurch aus, dass sehr viele klassische Lernverfahren und Algorithmen implementiert sind [Mac+18a]:

1 Classic goal-directed deterministic forward-search planning
- Breadth-first Search 1
- Depth-first Search 2
- A* 3
- IDA* 4
- Statically Weighted A* 5
- Dynamically Weighted A* 6

2 Stochastic Planning
- Value Iteration 7
- Policy Iteration 8
- Prioritized Sweeping 9
- Real-time Dynamic Programming 10
- UCB applied to Trees (UCT) (Upper Confidence Bounds (UCB)) 11
- Sparse Sampling 12
- Bounded Real-time Dynamic Programming 13

3 Learning
- Q-learning 14
- SARSA 15
- Actor Critic (AC) 16
- Potential Shaped RMax 17
- Adaptive Real-Time Dynamic Programming (ARTDP) 18

4 Value Function Approximation
- Gradient Descent SARSA 19
- Least-Squares Policy Iteration 20
- Fitted Value Iteration 21
- Framework for implementing linear and non-linear VFA 22
- Cerebellar Model Articulation Controller (CMAC) / Tile Coding 23
- Radial Basis Functions (RBF) 24
- Fourier Basis Functions 25

5 Inverse Reinforcement Learning
- Maximum Margin Apprenticeship Learning 26
- Multiple Intentions Maximum-likelihood Inverse RL 27
- Receding Horizon Inverse Reinforcement Learning 28

6 Multi-agent Q-learning and Value Iteration
Q-learning with an n-step action history memory 29
Friend-Q 30
Foe-Q 31
Correlated-Q 32
Coco-Q 33
7 Single-agent partially observable planning algorithms
Finite horizon optimal tree search 34
Q-Function Markov Decision Process (QMDP) 35
Belief MDP conversion (standard MDP algorithms) 36
8 Sonstige
The Options framework 37
Reward Shaping 38

Die Bibliotkek enthält selbst einige Umgebungen, wie bspw. Grid Worlds, Blocks World, Lunar Lander, Mountain Car, Cart Pole, Frostbite, Blockdude, Grid Games, Bimatrix Games. Es sind Erweiterungen für ROS-powered Robots und Minecraft sowie BURLAP Rosbridge, BURLAP Craft und BURLAP Weka vorhanden. Außerdem existiert eine Schnittstelle zu RLGlue (vgl. Kap. 6.7).

BURLAP verwendet das Werkzeug Apache *Maven* zum Bauen und Testen der Bibliothek [Fou18]. Auf dem zentralen *Repository* von Maven ist die Version 3.0.1 vom 04.08.2016 abgelegt. Wenn ein Java-Projekt mit Hilfe von Maven erstellt wird, dann muss man nur die Datei `pom.xml` anpassen und diese Version von BURLAP wird automatisch heruntergeladen und eingebunden.

Tabelle 6.3 fasst die wichtigsten Merkmale der BURLAP-Bibliothek zusammen [Mac+18b].

Name	BURLAP
Organisation	Brown University & UMBC
Webseite	https://github.com/jmacglashan/burlap
Lizenz	Apache License 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Java
APIs	Java
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	29 ǀ 205 ǀ 138
Commits ǀ Contributors	1.013 ǀ 13
Issues: Open ǀ Closed	11 ǀ 19
Requests: Open ǀ Closed	1 ǀ 119
Release (Date)	-
Last Commit	27.11.2017

Tabelle 6.3: Steckbrief zur BURLAP-Bibliothek [Mac+18b]

6.2 MAgent

MAgent ist eine Forschungsplattform für Multi-Agenten *Reinforcement Learning*, die von Lianmin Zheng, Jiacheng Yang, Han Cai, Weinan Zhang, Jun Wang und Yong Yu an der Shanghai Jiao Tong University entwickelt wurde [Zhe+17]. Skalierbarkeit ist ein wesentliches Merkmal der Plattform: Es ist möglich, bis zu einer Millionen Agenten auf einen einzigen GPU-Server zu verwalten. Zukünftig werden solche Methoden des *Artificial Collective Intelligence (ACI)* auf viele Bereiche angewendet werden. Eine großangelegte Spielwelt dient als Umgebung. In dieser Umgebung werden die Agenten als Rechtecke dargestellt. Sie besitzen detaillierte Informationen der lokalen Umgebung und ggf. auch globale Informationen. Sie können sich in der Umgebung bewegen, drehen und angreifen. Diese Programmteile sind in C++ geschrieben, damit schnelle Simulationen möglich sind.

Außerdem wurde eine Schnittstelle in Python (Version 2 oder 3) programmiert, um die Agenten zu steuern (Belohnungsfunktionen, Ereignisse, usw.). Die folgenden Algorithmen sind implementiert:

1. Deep Q-Network (DQN)
2. Deep Recurrent Q-Network (DRQN)
3. Advantage Actor Critic (A2C)

Drei Beispiele wurden erstellt, um den Wettbewerb (engl. *Competition*) und die Kooperation (engl. *Cooperation*) der Agenten zu untersuchen. Im Beispiel *Pursuit* kooperieren Jäger-Agenten, um Beute zu fangen. Im Beispiel *Gathering* ist Nahrung eine knappe Ressource und die Agenten müssen sich in ihren Aktionen entscheiden, ob sie Nahrung essen oder andere Agenten töten. Das dritte Beispiel *Battle* beschreibt, wie der Name bereits andeutet, den Kampf von zwei Armeen von Agenten.

Tabelle 6.4 fasst die wichtigsten Merkmale von MAgent zusammen [Zhe+18].

Name	MAgent
Organisation	Shanghai Jiao Tong University
Webseite	https://github.com/geek-ai/MAgent
Lizenz	MIT License
Plattformen	Linux, MacOSX
Geschrieben in	C++, Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	38 ǀ 553 ǀ 78
Commits ǀ Contributors	122 ǀ 7
Issues: Open ǀ Closed	1 ǀ 7
Requests: Open ǀ Closed	0ǀ 6
Release (Date)	-
Last Commit	09.01.2018

Tabelle 6.4: Steckbrief zur MAgent-Plattform [Zhe+18]

6.3 PyBrain

PyBrain ist eine modulare Bibliothek für Maschinenlernen in der Programmiersprache Python (Version 2). PyBrain steht für **Py**thon-**B**ased **R**einforcement Learning, **A**rtificial **I**ntelligence and **N**eural Network Library [Sch+10]. Die Software wurde von wissenschaftlichen Mitarbeitern und Master-Studierenden an der Technischen Universität München in Deutschland und am Dalle Molle Institut für Künstliche Intelligenz in der Schweiz unter der Leitung von Prof. Jürgen Schmidhuber entwickelt. Die folgenden Algorithmen zum Thema RL sind implementiert:

1	Q-Learning	mit/ohne *Eligibility Traces*
2	SARSA	
3	Neural Fitted Q-iteration	
4	REINFORCE	
5	Natural Actor-Critic	
6	Epsilon-Greedy Exploration	diskret
7	Boltzmann Exploration	diskret
8	Gaussian Exploration	kontinuierlich
9	State-Dependent Exploration	kontinuierlich

Umgebungen in PyBrain sind Szenarien oder Testfälle, in denen Agenten trainiert oder Algorithmen getestet werden können. Folgende Umgebungen sind im PyBrain-Paket bereits enthalten: ShipSteering, Mazes (MDP/POMDP), FlexCube sowie Roboter Johnnie und PencilBalancer aus dem Open Development Environment (ODE). Eine Schnittstelle zu RL-Glue ist vorhanden. Die aktuelle Version 0.31 ist vom 17.12.2017, wobei auf der Plattform GitHub andere Release-Nummern angegeben sind.

Tabelle 6.5 fasst die wichtigsten Merkmale der PyBrain-Bibliothek zusammen [SBR+18].

Name	PyBrain
Organisation	TU München, IDSIA
Webseite	https://github.com/pybrain/pybrain
Lizenz	BSD Software License
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	238 ǀ 2.454 ǀ 749
Commits ǀ Contributors	992 ǀ 32
Issues: Open ǀ Closed	116 ǀ 43
Requests: Open ǀ Closed	23 ǀ 50
Release (Date)	0.3.3 (09.01.2015)
Last Commit	17.12.2017

Tabelle 6.5: Steckbrief zur PyBrain-Bibliothek [SBR+18]

6.4 RLPark

RLPark ist eine *Reinforcement Learning* Bibliothek, die von Thomas Degris und Kollegen am Institut National de Recherche en Informatique et en Automatique (INRIA) entwickelt wurde [Deg+18b]. Die folgenden Algorithmen sind in Java implementiert:

1	On-Policy Control	1	SARSA
		2	Expected SARSA
		3	Actor Critic (AC) – Gauss (kontinuierlich)
		4	Actor Critic (AC) – Boltzmann (diskret)
		5	Average reward AC
2	Off-Policy Control	6	Off-Policy AC
		7	Q-Learning
		8	Q(λ)
		9	Greedy-GQ (General Q-Learning)
		10	Softmax-GQ
3	On-Policy Prediction	11	Temporal Difference (TD)
		12	TD(λ)
4	Off-Policy Prediction	13	GTD(λ)
		14	GQ(λ)
		15	Temporal Difference with Gradient Correction (TDC)

Die Umgebungen Mountain Car, Swing-up Pendulum, Random Walk und Continuous World sind als RL-Probleme enthalten. Zum Visualisieren der Ergebnisse wird Zephyr verwendet [Deg18].

Tabelle 6.6 fasst die wichtigsten Merkmale der RLPark-Bibliothek zusammen [Deg+18a].

Name	RLPark
Organisation	INRIA
Webseite	https://github.com/rlpark/rlpark
Lizenz	Eclipse Public License 1.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Java
APIs	Java
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	3 ǀ 21 ǀ 14
Commits ǀ Contributors	380 ǀ 4
Issues: Open ǀ Closed	0 ǀ 0
Requests: Open ǀ Closed	1 ǀ 0
Release (Date)	-
Last Commit	08.01.2014

Tabelle 6.6: Steckbrief zur RLPark-Bibliothek [Deg+18a]

6.5 RLPy

RLPy ist ein *Reinforcement Learning* Framework, welches von Alborz Geramifard, Robert H. Klein, Christoph Dann, William Dabney und Jonathan P. How in Zusammenarbeit von Massachusetts Institute of Technology (MIT) und Carnegie Mellon University entwickelt wurde. Der Schwerpunkt des Projekts liegt auf *Value-Function-based RL* mit linearer Funktionsapproximation und diskrete Aktionen [Ger+15]. Die folgenden Algorithmen sind in Python implementiert:

1. Temporal Difference (TD)
2. Q-Learning
3. SARSA
4. Natural AC
5. Least-Squares Policy Iteration
6. Greedy-GQ(λ)

RLPy enthält außerdem eine große Sammlung von Problemen und Umgebungen, u.a. die Klassiker Mountain Car und Cart Pole, wobei es zu letzterem sieben verschiedene Varianten gibt. Außerdem sind die folgenden Probleme bzw. Umgebungen enthalten [Ger+13]: Acrobot with Euler Integration, Acrobot with Runge-Kutta Integration, Bicycle Balancing, BlocksWorld, Linear Chain MDP, Fifty-State Chain MDP, FlipBoard, GridWorld, HIV Treatment, Helicopter Hovering, Intruder Monitoring, Persistent Search and Track Mission, Pacman, Pinball, PuddleWorld, RCCar, System Administrator und Swimmer.

Tabelle 6.7 fasst die wichtigsten Merkmale des RLPy-Frameworks zusammen [Dan+18].

Name	RLPy
Organisation	MIT u.a.
Webseite	https://github.com/rlpy/rlpy
Lizenz	BSD 3-Clause
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python
Statistik vom	02.02.2018
Watch | Star | Fork	17 | 179 | 59
Commits | Contributors	2.233 | 14
Issues: Open | Closed	13 | 1
Requests: Open | Closed	3 | 10
Release (Date)	1.3.6 (13.07.2015)
Last Commit	22.04.2016

Tabelle 6.7: Steckbrief zum RLPy-Framework [Dan+18]

6.6 MMLF

Das Maja Machine Learning Framework (MMLF) ist für Probleme aus dem Bereich *Reinforcement Learning* vom Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI) in Zusammenarbeit mit der Arbeitsgruppe Robotik der Universität Bremen entwickelt worden [Med18]. Die folgenden Algorithmen und Agenten sind in Python implementiert [ME11]:

1	Actor Critic (AC)	ActorCriticAgent
2	Direct Policy Search (DPS)	DPS_Agent
3	Dyna TD	DynaTDAgent
4	Fitted R-Max	FittedRMaxAgent
5	Model-based Direct Policy Search (MBDPS)	MBDPS_Agent
6	Monte Carlo (MC)	MonteCarloAgent
7	Policy Replay	PolicyReplayAgent
8	Random	RandomAgent
9	RoundRobin	ExampleAgent
10	Temporal Difference (TD)	TDLambdaAgent

Das MMLF enthält bereits einige Umgebungen, wie bspw. Labyrinthe (Maze 2D, Maze Cliff), Varianten des Pole Balancing Problems (Single Pole Balancing, Fully-observable Double Pole Balancing, Partially-observable Double Pole Balancing), das Mountain Car Testproblem sowie Spiele (Pinball, 17-und-4).

Zur Zeit wird dieses Framework nicht mehr weiterentwickelt. Die aktuellste Version der Software ist von 2013. Der offizielle Status des Projektes lautet: inaktiv.

Tabelle 6.8 fasst die wichtigsten Merkmale von MMLF zusammen [ME11].

Name	MMLF
Organisation	DFKI & Uni Bremen
Webseiten	http://mmlf.sourceforge.net
Akt. Version	(16.04.2013)
Lizenz	GPL 2.0
Plattformen	Windows, MacOSX, Linux
Geschrieben in	Python
APIs	Python

Tabelle 6.8: Steckbrief zu MMLF [ME11]

6.7 RL-Glue

RL-Glue ist eine Software-Plattform im Bereich *Reinforcement Learning*, die von Brian Tanner und Adam White an der University of Alberta entwickelt wurde. *Glue* bedeutet Klebstoff und dieser Begriff drückt sehr gut die zentrale Idee hinter dieser Plattform aus: RL-Glue stellt eine Schnittstelle bereit, um Agenten, Umgebungen und Experimente miteinander zu verbinden (vgl. Abb. 6.1), selbst wenn diese in unterschiedlichen Programmiersprachen erstellt sind [TW09].

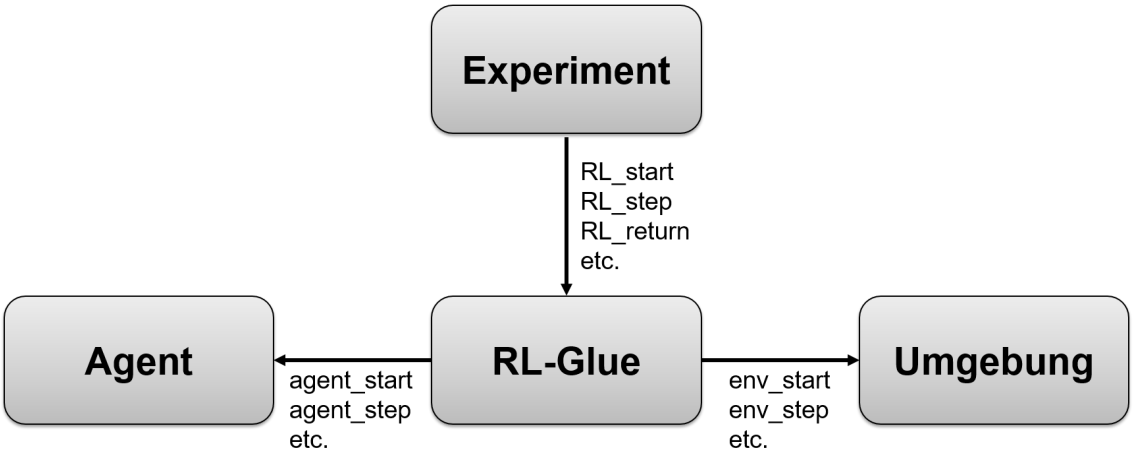


Abbildung 6.1: Design-Prinzip von RL-Glue

Somit können diese Module leichter wiederverwendet und ausgetauscht werden. Glue selbst ist in C++ programmiert, bietet aber bspw. auch Application Programming Interface (API) zu Java und Python an. Algorithmen selbst sind nicht Bestandteil dieser Softwarelösung.

Tabelle 6.9 fasst die wichtigesten Merkmale von RL-Glue zusammen [TW10].

Name	RL-Glue
Organisation	University of Alberta
Webseite	http://glue.rl-community.org
Akt. Version	3.0.4. (06.06.2009)
Lizenz	Apache License 2.0
Plattformen	Linux, MacOSX, Windows
Geschrieben in	C/C++
APIs	C/C++, Go, Java, Lisp, Matlab, Python

Tabelle 6.9: Steckbrief zu RL-Glue [TW10]

6.8 RL Toolkit

Das RL Toolkit ist eine Sammlung von nützlichen Programmen und Demos zum Thema *Reinforcement Learning*, das von der RLAI-Gruppe unter der Leitung von Richard S. Sutton an der University of Alberta entwickelt wurde [Sut16]. Er ist auch einer der Autoren des Standardwerks zum Thema RL [SB98]. Die Software ist in Python programmiert, ursprünglich in Version 2 und seit Oktober 2016 auch für Version 3 verfügbar.

Die folgenden Demos zu RL-Problemen werden bereitgestellt: Mountain Car, Maintenance Example, Gridworld, Function Approximation und Tiles Demo.

Tabelle 6.10 fasst die wichtigsten Merkmale des RL-Toolkits zusammen [Sut16].

Name	RL-Toolkit
Organisation	University of Alberta
Webseiten	http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/RLtoolkit/RLtoolkit1.0.html
Akt. Version	1.0 für Python Version 3 (Okt. 2016)
Lizenz	Public Domain
Plattformen	Linux, MacOSX, Windows
Geschrieben in	Python
APIs	Python

Tabelle 6.10: Steckbrief zum RL-Toolkit [Sut16]

7 Tutorials

In diesem Kapitel werden sechs Tutorials zum Thema *Reinforcement Learning* vorgestellt. Sie haben gemeinsam, dass Sie sich auf der Plattform GitHub befinden, in Englisch erstellt sind, Quelltexte zu Beispielen in der Programmiersprache Python enthalten und sie größtenteils sehr populär sind, d.h. viele *Stars* aus der GitHub-Community erhalten haben (Stand: 02.02.2018). In den Python-Skripten werden Umgebungen und Algorithmen verwendet, weshalb somit auch diese Tutorials zu Softwarelösungen im weiteren Sinne gehören.

Tabelle 7.1 zeigt eine Übersicht dieser Tutorials in alphabetischer Reihenfolge.

Nr	Name	Stars	Ersteller	Land
1	Basic RL	341	Víctor Mayoral	Spanien
2	Practical RL	1.565	Fedor Ratnikov	Russland
3	RL	5.617	Denny Britz	England
4	RL An Introduction	2.808	Shangtong Zhang	Kanada
5	RL Code	1.693	Lee Woongwon	Südkorea
6	RL with TensorFlow	1.040	Morvan Zhou	China

Tabelle 7.1: Übersicht ausgewählter RL-Tutorials auf der Plattform GitHub [Git18]

Der Beispielcode aus diesen Tutorials kann prinzipiell auf allen gängigen Plattformen (Linux, MacOSX, Windows) ausgeführt werden, ggf. werden jedoch zusätzlich Pakete (z.B. numpy) oder Bibliotheken (z.B. TensorFlow) benötigt und müssen somit vorher installiert werden. Einige der Tutorials verwenden auch Jupyter Notebooks.

7.1 Basic-RL

Basic-RL ist ein Tutorial zum Thema *Reinforcement Learning* von Víctor Mayoral Vilches. Der Spanier ist Mitgründer der Firma Erle Robotics und hat Universitätsabschlüsse in Elektrotechnik und Computerwissenschaft. Sein Online-Tutorial besteht aus 15 Lerneinheiten inklusive vieler Python-Skripte, wobei an einigen Themen noch gearbeitet wird [May18]:

1	Q-learning	
2	SARSA	
3	Exploring OpenAI Gym	
4	Q-learning in OpenAI Gym	
5	Deep Q-learning (DQN)	
6	Deep Convolutional Q-learning	
7	Reinforcement Learning with ROS and Gazebo	
8	Reinforcement Learning in DOOM	unvollendet
9	Deep Deterministic Policy Gradients (DDPG)	
10	Guided Policy Search (GPS)	unvollendet
11	A review of different AI techniques for RL	in Arbeit
12	Reviewing Policy Gradient methods	
13	Continuous-state spaces with DQN	fehlt
14	Benchmarking RL techniques	
15	Reviewing Vanilla Policy Gradient (VPG)	fehlgeschlagen

Das ZIP-Archiv muss nur von GitHub heruntergeladen und entpackt werden. Zu jedem Thema ist ein eigener Ordner mit einer Beschreibung als `README.md`-Datei und den zugehörigen Python-Skripten enthalten. Zu einigen Aufgaben wird das OpenAI Gym (vgl. Kap. 4.4) verwendet.

Tabelle 7.2 fasst die wichtigsten Merkmale des Basic-RL-Tutorials zusammen [May18]. 341 Sterne, aktuell

Name	Basic RL
Ersteller	Víctor Mayoral
Webseite	https://github.com/vmayoral/basic_reinforcement_learning
Lizenz	GPL v 3.0
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	40 ǀ 341 ǀ 116
Commits ǀ Contributors	204 ǀ 3
Issues: Open ǀ Closed	1 ǀ 2
Requests: Open ǀ Closed	0 ǀ 1
Last Commit	28.01.2018

Tabelle 7.2: Steckbrief zum Basic-RL-Tutorial [May18]

7.2 Practical-RL

Practical-RL ist Tutorial und eine Online-Materialsammlung von Kursen zu *Reinforcement Learning*, die an der *Higher School of Economics (HSE)* und an der *Yandex School of Data Analysis (YSDA)* in Englisch bzw. Russisch von Fedor Ratnikov, Oleg Vasilev, Pavel Shvechikov und Alexander Fritsler gehalten wurden bzw. aktuell gehalten werden [Rat+18]. Das Curriculum besteht aus 11 Lerneinheiten, die in 9 Wochen durchgenommen werden. Zwei Lerneinheiten zu speziellen KNN-Themen sind dabei als Repetitorien gedacht. Somit wird Wissen zu Künstlichen Neuronalen Netzwerken und insbesondere zu *Deep Learning (DL)* vorausgesetzt.

1	RL as blackbox optimization	
2	Value-based methods	
3	Model-free reinforcement learning	
4	Deep Learning (DL) Approximate RL: DQN	Repetitorium
5	Exploration and exploitation	
6	Policy gradient methods	
7	Recurrent Neural Networks (RNN) POMDP Learning	Repetitorium
8	RL outside games: Sequence learning	
9	Trust Region Policy Optimization	

Zu den Unterlagen gibt es Python-Skripte und Jupyter Notebooks. Es wird das OpenAI Gym als Umgebung (vgl. Kap. 4.4) benutzt und die Bibliotheken TensorFlow, Lasagne, PyTorch und Theano eingesetzt.

Tabelle 7.3 fasst die wichtigsten Merkmale des Practical-RL-Tutorials zusammen [Rat+18].

Name	Practical RL
Ersteller	Fedor Ratnikov u.a.
Webseite	https://github.com/yandexdataschool/Practical_RL
Lizenz	MIT License
Statistik vom	02.02.2018
Watch ∣ Star ∣ Fork	147 ∣ 1.565 ∣ 355
Commits ∣ Contributors	563 ∣ 23
Issues: Open ∣ Closed	4 ∣ 14
Requests: Open ∣ Closed	1 ∣ 24
Last Commit	29.01.2018

Tabelle 7.3: Steckbrief zum Practical-RL-Tutorial [Rat+18]

7.3 RL

Denny Britz hat eine Sammlung von Implementierungen von RL-Algorithmen in Python 3 zusammengetragen [Bri16]. Die Aufgaben und Lösungen stammen aus dem Buch von Sutton & Barto [SB18] oder aus dem Kurs *Reinforcement Learning* von David Silver am *University College London* [Sil15]. Die Skripte nutzen Umgebungen von OpenAI Gym (vgl. Kap. 4.4). Einige Algorithmen sind mit Hilfe der TensorFlow-Bibliothek programmiert. Die meisten Lösungen sind als Jupyter Notebook angegeben. Die folgenden Algorithmen sind implementiert:

1 Dynamic Programming Policy Evaluation
2 Dynamic Programming Policy Iteration
3 Dynamic Programming Value Iteration
4 Monte Carlo Prediction
5 Monte Carlo Control with Epsilon-Greedy Policies
6 Monte Carlo Off-Policy Control with Importance Sampling
7 SARSA (On Policy TD Learning)
8 Q-Learning (Off Policy TD Learning)
9 Q-Learning with Linear Function Approximation
10 Deep Q-Learning for Atari Games
11 Double Deep-Q Learning for Atari Games
12 Deep Q-Learning with Prioritized Experience Replay i.A.
13 Policy Gradient: REINFORCE with Baseline
14 Policy Gradient: Actor Critic with Baseline
15 Policy Gradient: Actor Critic with Baseline for Continuous Action Spaces
16 Deterministic Policy Gradients for Continuous Action Spaces i.A.
17 Deep Deterministic Policy Gradients (DDPG) i.A.
18 Asynchronous Advantage Actor Critic (A3C)

Drei Algorithmen sind noch in Arbeit (i.A.). Tabelle 7.4 fasst die wichtigsten Merkmale der RL-Beispiele zusammen [BB+18].

Name	WildML
Ersteller	Denny Britz
Webseite	https://github.com/dennybritz/reinforcement-learning
Lizenz	MIT License
Statistik vom	02.02.2018
Watch | Star | Fork	470 | 5.617 | 1.837
Commits | Contributors	209 | 28
Issues: Open | Closed	48 | 42
Requests: Open | Closed	4 | 44
Last Commit	28.01.2018

Tabelle 7.4: Steckbrief zur den RL-Beispielen [BB+18]

7.4 RL An Introduction

Diese Sammlung von Beispielen trägt den Namen *Reinforcement Learning: An Introduction* in Anlehnung an das Standardwerk von Richard S. Sutton und Andrew G. Barto [SB98]. Shangtong Zhang, ein Master-Student und Assistent von Richard S. Sutton der *University of Alberta* hat die meisten Beispiele der geplanten 2. Auflage dieses Buches [SB18] in Python programmiert und zusammengetragen. Der Aufbau des GitHub-Repository orientiert sich dabei an der Gliederung des Buches [Zha+18]:

1	Introduction	Tic-Tac-Toe
2	Multi-armed Bandits	Ten Armed Testbed
3	Finite Markov Decision Processes	Grid World
4	Dynamic Programming	Grid World, Car Rental, Gamblers Problem
5	Monte Carlo Methods	Blackjack
6	Temporal Difference Learning	Random Walk, Windy Grid World, Cliff Walking
7	n-step Bootstrapping	Random Walk
8	Planning and Learning with Tabular Methods	Various Maze
9	On-policy Prediction with Approximation	Random Walk, Square Wave
10	On-policy Control with Approximation	Mountain Car, Access Control
11	Off-policy Methods with Approximation	Baird
12	Eligibility Traces	RandomWalk, Mountain Car

Einige Beispiele fehlen allerdings noch: Pole-Balancing, Draw Poker, Soap Bubble, Rod Maneuvering, Backgammon, The Acrobot und Go. Die neue Auflage des Standardwerks hat auch 17 Kapitel. Die vorhandenen Python-Skripte sind kompatibel mit den Versionen 2 und 3. Als weitere Pakete werden nur Numpy, Matplotlib, Six und Seaborn benötigt, welche alle bereits in der Python-Distribution Anacoda vorhanden sind.

Tabelle 7.5 fasst die wichtigsten Merkmale der Sutton-&-Barto-Beispiele zusammen [Zha+18].

Name	RL An Introduction
Ersteller	Shangtong Zhang
Webseite	https://github.com/ShangtongZhang/reinforcement-learning-an-introduction
Lizenz	Apache License 2.0
Statistik vom	02.02.2018
Watch ǀ Star ǀ Fork	291 ǀ 2.808 ǀ 1.115
Commits ǀ Contributors	161 ǀ 12
Issues: Open ǀ Closed	1 ǀ 32
Requests: Open ǀ Closed	0 ǀ 28
Last Commit	06.01.2018

Tabelle 7.5: Steckbrief zur den Beispielen von Sutton-&-Barto [Zha+18]

7.5 RLcode

RLcode ist ein Projekt, um Beispiele zum *Reinforcement Learning* zu präsentieren, die sehr elegant in Python programmiert sind [WYL+18]: Die Quelltexte sind minimal gehalten, aufgeräumt, gut lesbar und verständlich. Zum Entwicklungsteam gehören u.a. Lee Woongwon, Hyeokryeol Yang und Brian Lee aus Südkorea. Es werden 4 verschiedene Probleme betrachtet, die mit Hilfe ausgewählter Algorithmen, gelöst werden:

1	Grid World	1	Policy Iteration
		2	Value Iteration
		3	Monte Carlo
		4	SARSA
		5	Q-Learning
		6	Deep SARSA
		7	REINFORCE
2	CartPole	8	Deep Q-Network (DQN)
		9	Double Deep Q-Network (DDQN)
		10	Policy Gradient
		11	Advantage Actor Critic (A2C)
		12	Asynchronous Advantage Actor Critic (A3C)
3	Atari Breakout	13	DQN
		14	DDQN
		15	Dueling DQN
		16	A3C
	Atari Pong	17	Policy Gradient
4	Mountain Car	18	DQN

Neben Python 3 werden die Bibliotheken Tensorflow und Keras sowie die Pakete numpy, pandas, matplot, pillow, Skimage und h5py benötigt. Die Python-Distribution Anaconda enthält bereits fast alle diese Pakete, nur Skimage muss zusätzlich installiert werden.

Tabelle 7.6 fasst die wichtigsten Merkmale der RLcode-Beispiele zusammen [WYL+18].

Name	RLcode
Ersteller	Lee Woongwon, Hyeokryeol Yang, Brian Lee
Webseite	https://github.com/rlcode/reinforcement-learning
Lizenz	MIT License
Statistik vom	02.02.2018
Watch | Star | Fork	95 | 1.693 | 338
Commits | Contributors	264 | 10
Issues: Open | Closed	8 | 23
Requests: Open | Closed	2 | 41
Last Commit	28.11.2017

Tabelle 7.6: Steckbrief zur den RLcode-Beispielen [WYL+18]

7.6 RL with TensorFlow

Reinforcement Learning with TensorFlow ist eine Sammlung von Methoden und Tutorials des Chinesen Morvan Zhou [Zho18]. Die Python-Skripte der 15 Lerneinheiten verwenden die TensorFlow-Bibliothek und teilweise Umgebungen aus dem OpenAI Gym (vgl. 4.4), wie z.B. Pendulum, CartPole und MountainCar:

1	Simple entry example	1D-Treasure
2	Q-learning	Maze
3	SARSA	Maze
4	SARSA(λ)	Maze
5	Deep Q-Network (DQN)	Maze
6	Using OpenAI Gym	CartPole, MountainCar
7	Double Deep Q-Network (DDQN)	Pendulum
8	DQN with Prioitized Experience Replay	MountainCar
9	Dueling DQN	Pendulum
10	Policy Gradients with Softmax	CartPole, MountainCar
11	Actor Critic	CartPole, Pendulum
12	Deep Deterministic Policy Gradients (DDPG)	Pendulum
13	Asynchronous Advantage Actor Critic (A3C)	Pendulum
14	Dyna-Q	Maze
15	Proximal Policy Optimization (PPO)	Pendulum

Zusätzlich werden noch 4 Experimente betrachtet: 2D Car, Robot arm, BipedalWalker und LunarLander.

Tabelle 7.7 fasst die wichtigsten Merkmale der RL-TF-Sammlung zusammen [Zho18].

Name	RL with TensorFlow
Ersteller	Morvan Zhou
Webseite	https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow
Lizenz	Apache License 2.0
Statistik vom	02.02.2018
Watch | Star | Fork	91 | 1.040 | 670
Commits | Contributors	76 | 3
Issues: Open | Closed	0 | 28
Requests: Open | Closed	2 | 2
Last Commit	28.01.2018

Tabelle 7.7: Steckbrief zum RL-TF-Sammlung [Zho18]

8 Analysen

In den letzten Kapiteln wurden 37 Softwarelösungen zum Thema *Reinforcement Learning* vorgestellt. In diesem Kapitel werden nun einige dieser Lösungen analysiert, ausgewählt, installiert und getestet. Die Installationen und Tests werden auf einem speziellen Testsystem durchgeführt, welches zunächst in Kap. 8.1 beschrieben wird. Die Analyse und Auswahl erfolgt u.a. mit einem Scoring-Modell und wird in Kap. 8.2 erläutert. Das Vorgehen bei den Installationen und Tests wird schließlich in Kap. 8.3 dargestellt.

8.1 Testsystem

Die Hardware des Testsystems besteht aus mehreren Komponenten (vgl. Tab. 8.1), wobei die Grafikkarte im Mittelpunkt steht und alle anderen Komponenten passend zu dieser ausgewählt wurden. Denn mittlerweile werden rechenintensive Anwendungen häufig auf Grafikprozessoren (engl. *Graphics Processing Unit (GPU)*) statt auf Hauptprozessoren (engl. *Central Processing Unit (CPU)*) ausgeführt. Es kommt ein High-End-Modell mit der GeForce GTX 1080 Ti GPU von Nvidia zum Einsatz. Sie verfügt über 3584 Streamprozessoren, taktet mit 1594 MHz und kann auf 11 GB eigenen Speicher zugreifen.

Nr.	Komponente	Hersteller	Bezeichnung (und Merkmale)
1	Gehäuse	Corsair	Carbide 270R Window Tower
2	Netzteil	Corsair	RM1000X (1000 Watt)
3	Mainboard	MSI	Z270 Gaming Pro Carbon
4	Prozessor	Intel	Core i7-7700K (4 x 4,2GHz)
5	CPU-Kühler	Alpenföhn	Brocken Eco
6	Speicher	Corsair	DIMM 64GB DDR4-2400 Kit
7	SSD	Samsung	960 EVO (SSD 250GB, M.2)
8	Festplatte	WD	WD40EFRX (4TB, SATA 600)
9	Grafikkarte	Gigabyte	AORUS GeForce GTX 1080 Ti

Tabelle 8.1: Hardware-Komponenten

Die verwendete Software ist *Open Source* oder *Freeware*. Es fallen also keine Lizenzkosten für deren Nutzung an. Als Betriebssystem wurde somit Linux ausgewählt und Ubuntu 16.04 LTS auf dem Rechner installiert. LTS steht für *Long Term Support*. Die Version 16.04 wird bis April 2021 von Ubuntu unterstützt, d.h. bis dahin gibt es regelmäßig Aktualisierungen, die einen stabilen und sicheren Betrieb gewährleisten. Neben dem Grafikkarten-Treiber sind auch zwei weitere Softwarelösungen von Nvidia installiert. Mit dem Werkzeug CUDA 9.1 lassen sich parallele Rechnungen auf der GPU ausführen. CuDNN 7.0.5 ist eine Software-Bibliothek, die auf CUDA basiert und mit der sich tiefe Künstliche Neuronale Netzwerke trainieren lassen. CuDNN steht für *CUDA Deep Neural Network Library*. Diese Software ist in C/C++ programmiert. Mittlerweile gibt es aber auch viele *Deep Learning*-Softwarelösungen, die CuDNN unterstützen und weitere *high level* Schnittstellen

zu anderen Programmiersprachen anbieten. Die DL-Bibliothek *TensorFlow* von Google bietet bspw. eine solche API für Python an. Python ist deshalb so interessant, weil diese Programmiersprache im Bereich *Data Science* sehr verbreitet ist. Aus diesem Grund wurde zunächst Anaconda 5.0.1 für Python 3.6 installiert. Anaconda ist eine Python-Distribution, in der u.a. auch Pakete mitgeliefert werden, die zur Verarbeitung und Analyse von großen Datenmengen eingesetzt werden können. Außerdem ist das Jupyter Notebook enthalten, ein Webbasiertes Frontend, mit dem sich Python-Skripte einfach per Browser erstellen und ausführen lassen. Das Jupyter Notebook wurde so eingerichtet, dass es sich per SSH-Tunnel von einem anderen Rechner aus bedienen lässt. Somit unterstützt das Testsystem die Client/Server-Architektur. Von einem beliebigen Client aus kann nun bspw. mit dem Programm PuTTY der SSH-Tunnel zu dem Linux-Server aufgebaut und anschließend im Webbrowser mit dem Jupyter Notebook Python-Skripte auf dem Server editiert und ausgeführt werden. Schließlich wurde auf dem Linux-Server noch die bereits erwähnte Bibliothek TensorFlow 1.5.0 installiert.

8.2 Scoring

Insgesamt wurden in den vorherigen Kapiteln 37 Softwarelösungen zum Thema *Reinforcement Learning* präsentiert. Nicht alle diese Lösungen können im Rahmen dieser Arbeit getestet werden. Also muss eine Auswahl getroffen werden: Es werden daher nur Umgebungen (vgl. Kap. 4) und Bibliotheken bzw. Frameworks (vgl. Kap. 5) berücksichtigt. Diese Lösungen unterstützen ein modulares Architekturkonzept, d.h. die Komponenten (Umgebung, Agent bzw. Algorithmus, Experiment usw.) lassen sich leicht austauschen und wiederverwenden. Spezielle und sonstige Lösungen (vgl. Kap. 6) sowie Tutorials (vgl. Kap. 7) werden nicht weiter betrachtet, da sich diese Softwarelösungen schlecht wiederverwenden und erweitern lassen. Generell sollte sich die Software auf dem Testsystem (vgl. Kap. 8.1) installieren lassen, d.h. Linux als Betriebssystem und Python in der Version 3 als Programmiersprache sind als K.O.-Kriterien gesetzt.

Aus den Beschreibungen der Bibliotheken und Frameworks geht bereits hervor, dass alle diese 13 Softwarelösungen eine Schnittstelle zu den Umgebungen von *OpenAI Gym* besitzen. Somit kann *Gym* als Quasi-Standard einer RL-Plattform für Lernumgebungen bezeichnet werden. Außerdem gibt es mit *Universe* und *Roboschool* bereits zwei Erweiterungen der Non-Profit-Organisation *OpenAI*. Die meisten positiven Bewertungen (*Stars*) wurden ebenfalls für *OpenAI Gym* von der GitHub-Community vergeben. Diese Plattform lässt sich unter Linux verwenden und bietet mit Python in Version 3 auch die passende API an. Einen Nachteil hat *Gym* allerdings: Einige spezielle Umgebungen basieren auf MuJoCo, einer kommerziellen Physik-Engine. Mit Hilfe von *Roboschool* lassen sich aber bereits einige dieser Umgebungen ersetzen, weil *Roboschool* für seine Umgebungen zwei *Open Source* Physik-Engines verwendet. Ansonsten sind die Umgebungen der Unternehmen Intel, Google und Microsoft interessant. *DeepMind Lab* von Google funktioniert derzeit allerdings nur mit Python in der Version 2 und ist somit nicht kompatibel. Der Simulator *CARLA* aus den Intel Labs benötigt ggf. eine noch stärkere Hardware als die des Testsystems. Zumindest aber das *Project Malmo* von Microsoft sollte ebenfalls getestet werden.

Von den 13 Bibliotheken und Frameworks bieten 11 Python als Programmierschnittstelle an, die meisten setzen außerdem die *Deep Learning*-Bibliothek TensorFlow als Basis ein. Zu diesen 11 Softwarelösungen wird nun eine Nutzwertanalyse durchgeführt, um herauszufinden, welches die geeignetsten Lösungen in dem Kontext dieser Arbeit sind.

Nutzwertanalyse Nach Zangemeister ist eine Nutzwertanalyse eine [Zan70]:

> *« Analyse einer Menge komplexer Handlungsalternativen mit dem Zweck, die Elemente dieser Menge entsprechend den Präferenzen des Entscheidungsträgers bezüglich eines multidimensionalen Zielsystems zu ordnen. Die Abbildung der Ordnung erfolgt durch die Angabe der Nutzwerte (Gesamtwerte) der Alternativen. »*

Mit einer Nutzwertanalyse lässt sich also herausfinden, welche Alternative den höchsten Nutzen bietet. Sie wird auch Scoring-Verfahren oder Scoring-Modell genannt. Mathematisch betrachtet, gilt es die Zielfunktion:

$$f(x_1, ..., x_n) = w_1 f_1(x_1) + ... + w_n f_n(x_n) \tag{8.1}$$

zu maximieren. Dabei sind x_i das zu bewertende Merkmal i, w_i das Gewicht zum Merkmal i und $f_i(x_i)$ die Bewertungsfunktion des i-ten Merkmals, wobei n die Anzahl der Merkmale angibt. Es gilt: $0 \leq w_i \leq 1$ und die Nebenbedingung:

$$\sum_{i=1}^{n} w_i = 1 \tag{8.2}$$

muss erfüllt sein.

Merkmale, Gewichte und Skalen Um eine Nutzwertanalyse durchzuführen müssen gemäß Gl. 8.1 die Merkmale, die zugehörigen Gewichte und die Bewertungsfunktionen bzw. Skalen definiert werden. Für die Analyse in dieser Arbeit werden fünf Kategorien betrachtet und deren Gewichte auf jeweils 20 Prozent gesetzt (vgl. Tab. 8.2).

Nr	Kategorie	Merkmal	Gewichtung	
1	Funktionalität	Algorithmen	10 %	20 %
		Experimente	5 %	
		Schnittstellen	5 %	
2	Popularität	Watch	5 %	20 %
		Star	10 %	
		Fork	5 %	
3	Aktualität	Last Commit	10 %	20 %
		Release Date	5 %	
		Issues & Requests	5 %	
4	Reputation	Commits	5 %	20 %
		Contributors	5 %	
		Organisation	10 %	
5	Dokumentation	GitHub-Readme	5 %	20 %
		Webseite	10 %	
		API	5 %	

Tabelle 8.2: Kategorien, Merkmale und Gewichte

Es gibt keinerlei Präferenzen, sondern jede Kategorie wird als gleich wichtig erachtet. Das ist zwar einfach, aber meistens auch sehr effektiv. Jede Kategorie wird durch drei

Merkmale beschrieben, insgesamt sind es also 15 Merkmale. Hier findet eine einfache Abstufung bezüglich der Gewichte statt. In jeder Kategorie gibt es ein Hauptmerkmal, das mit 10 Prozent gewichtet wird, alle übrigen mit 5 Prozent.

Bevor nun die einzelnen Kategorien und Merkmale näher beschrieben werden, wird die Skala definiert. Zur Bewertung der 15 Merkmale der 11 Alternativen wird das Schulnotensystem verwendet, wobei jede Note einer Punktzahl zwischen null und fünf zugeordnet ist (vgl. Tab. 8.3).

Note	Bedeutung	Punkte
1	sehr gut	5
2	gut	4
3	befriedigend	3
4	ausreichend	2
5	ungenügend	1
6	mangelhaft	0

Tabelle 8.3: Punkteskala für die Bewertung der Alternativen

Die **Funktionalität** der Software lässt sich anhand der bereits implementierten Komponenten Algorithmen bzw. Agenten und Experimente sowie Schnittstellen zu externen Systemen bzw. Umgebungen bewerten, wobei der Schwerpunkt auf den RL-Algorithmen liegt. Der Einfachheit halber werden hier quantitative Kriterien zur Bewertung benutzt, d.h. die Anzahl der Algorithmen (bzw. Agenten), Experimente (bzw. Beispiele) und Schnittstellen werden gezählt und miteinander verglichen. Hierzu werden die jeweiligen Quelltexte aus dem GitHub-Repository analysiert. Die Maximalwerte zu jedem Merkmal werden zusätzlich notiert. Mit Hilfe einer linearen Interpolation

$$f_i(x_i) = \frac{5}{x_i^{\max}} \cdot x_i \tag{8.3}$$

kann dann die individuelle Punktzahl berechnet werden. In Gl. 8.3 ist x_i der Wert für die Anzahl der Algorithmen ($i = 1$), Experimente ($i = 2$) und Schnittstellen ($i = 3$), während $x_i^{\max}$ die Maximalwerte zu diesen drei Merkmalen sind. Wenn also eine Softwarelösung diesen Maximalwert erreicht, bekommt sie auch 5 Punkte. Der theoretisch mögliche Minimalwert von Null führt auch zu null Punkten.

Die Kategorie **Popularität** ist indirekt bereits ein gutes Bewertungskriterium für die Softwarelösungen. Mitglieder der GitHub-Community haben bereits für sich entschieden, ob sie die Lösung näher beobachten (*Watch*), positiv bewerten (*Star*) oder selbst Änderungen an der Software vornehmen (*Fork*). Nicht jedes Mitglied der GitHub-Plattform muss aber unbedingt ein Programmierer sein. Insofern bekommt das Merkmal *Stars* in dieser Kategorie das höchste Gewicht. Alle diese Größen wurden bereits quantitativ am 02.02.2018 bestimmt und sind in den Tabellen im Kap. 5 bereits dargestellt. Die Berechnung der jeweiligen Punktzahl erfolgt wieder gemäß mittels Gl. 8.3. Es gilt: $x_4^{\max} = 397$ für *Watch*, $x_5^{\max} = 3.245$ für *Star* und $x_6^{\max} = 791$ für *Fork*.

Aktualität einer Softwarelösung ist wichtig, denn wenn Software nicht regelmäßig aktualisiert wird, dann veraltet sie. Veraltete Software wird irgendwann nicht mehr eingesetzt

und verschwindet dann wieder vom Markt. Die Weiterentwicklung von Software wird in Pflege und Wartung unterteilt. Pflege von Software bedeutet, dass Fehler (engl. *Bugs*) korrigiert werden. Unter Wartung versteht man die Anpassung der Software an neue Benutzerwünsche (engl. *Change Requests*), geänderte Systemumgebungen, neue gesetzliche Bestimmungen usw. Das Ergebnis ist eine neue Version der Software (engl. *Update*). Versionsnummern geben Aufschluss über die Historie der Software. Ein Software-*Release* ist eine freigegebene Version. Eine Nummer kleiner Eins weißt normalerweise auf eine Beta-Version hin, die also noch nicht ganz ausgereift ist und sich in einem experimentellen Stadium befindet. Das wichtigste Merkmal in dieser Kategorie ist das Datum des *Last Commit*. Dieses Datum gibt an, wann die letzte Änderung am Quelltext vorgenommen wurde. Das *Release Date* ist das Datum der letzten *Release*-Version. *Issues* sind von Mitgliedern gemachte Meldungen und entsprechen den oben genannten *Bugs* und *Change Requests*. Die sogenannten *Pull Requests* sind von Mitgliedern vorgenommene Änderungen am Quelltext, die über eine Kontrollinstanz (Administrator) ggf. in das Repository einfließen werden. Zu allen diesen Größen liegen wieder quantitative Werte vor. Zum Datum gibt es allerdings keinen natürlichen Nullpunkt, d.h. dieses Merkmal wird mit einer Intervallskala beschrieben. Der Stichtag der Messung (02.02.2018) ist aber der Referenzwert, zu dem ein Abstand in Tagen bestimmt wird. Ein Abstand von null Tagen ist gleichbedeutend mit optimaler Aktualität und wird mit 5 Punkten bewertet. Weiterhin wird festgelegt, dass ein Abstand von 100 Tagen oder mehr keine Punkte mehr bekommt. Zwischenwerte werden wieder linear interpoliert:

$$f_i(x_i) = \begin{cases} 5 - \frac{5}{100} \cdot x_i & \text{falls } 0 \leq x_i \leq 100 \\ 0 & \text{falls } x_i > 100 \end{cases} \tag{8.4}$$

mit x_i als Wert für den Abstand in Tagen zum Referenz-Datum 02.02.2018. Die Punktzahl für *Issues & Requests* kann wieder mit Hilfe der Gl. 8.3 berechnet werden. Allerdings muss der Wert für x_i zunächst aus vier Einzelwerten bestimmt werden:

$$x_9 = \#(\text{Issues Closed}) - \#(\text{Issues Open}) + \#(\text{Requests Closed}) - \#(\text{Requests Open}) \tag{8.5}$$

Gelöste Probleme und Anfragen werden also positiv bewertet, während noch offene Probleme und Anfragen negativ gezählt werden.

Die Kategorie **Reputation** enthält zwei Merkmale, die sich wieder quantitativ feststellen lassen: *Commits* und *Contributors*. Anhand dieser Zahlen kann bewertet werden, wie aktiv die Programmierer an dem Projekt arbeiten. Auch hier kann wieder Gl. 8.3 benutzt werden, um die Punkte zu bestimmen. Ein noch höheres Gewicht wird der Organisation beigemessen, die hinter dem Projekt und der Softwarelösung steht. Die Bewertung dieser Organisation erfolgt allerdings qualitativ. Namhafte, internationale Unternehmen mit hoher Liquidität werden am stärksten eingeschätzt (5 Punkte). Gleich dahinter rangieren renommierte Forschungsinstitute bzw. Universitäten (4 Punkte). Am schwächsten wird eine Open Community bewertet (1 Punkt). Sonstige Organisationen liegen dazwischen.

Für Anfänger ist eine gute **Dokumentation** der Software sehr hilfreich. Den ersten Eindruck bekommt man auf der Projektstartseite auf der Plattform GitHub. Dort wird nämlich auch immer der Inhalt der Readme-Datei direkt angezeigt. Diese ist also wie eine Visitenkarte. Einige Softwarelösungen haben weitere, meistens externe Webseiten, auf denen die Software mehr oder weniger umfangreich beschrieben ist. Dies ist meistens eine gute Informationsquelle für Anfänger, die erste Erfahrungen mit der Software sammeln

möchten. Für fortgeschrittene Anwender und Programmierer steht dagegen die Offenlegung der Programmierschnittstelle (API) im Vordergrund. Diese findet man allerdings nur selten bei den 11 Alternativen. Diese drei Merkmale lassen sich nur qualitativ mit den Schulnoten bewerten und dann entsprechend Tab. 8.3 in Punkte umrechnen.

Nutzwerte (Scores) In den Tabellen 8.4 bis 8.6 sind die Ergebnisse der Nutzwertanalyse dargestellt. Aus Darstellungsgründen wurden die Daten auf drei Tabellen aufgeteilt. Tabelle 8.4 zeigt zunächst die einzelnen Punktzahlen für die Softwarelösungen Chainer RL (vgl. Kap. 5.1), Coach (vgl. Kap. 5.2), Deep RL (vgl. Kap. 5.3) und Keras-rl (vgl. Kap. 5.4).

Alternativen:		**Chainer RL**		**Coach**		**Deep RL**		**Keras-rl**	
Merkmal	Gew.	Pkt.	NW	Pkt.	NW	Pkt.	NW	Pkt.	NW
Funktionalität									
Algorithmen	10 %	4,444	0,444	5,000	0,500	3,056	0,306	2,222	0,222
Experimente	5 %	2,609	0,130	5,000	0,250	0,870	0,044	1,739	0,087
Schnittstellen	5 %	1,667	0,083	5,000	0,250	1,667	0,083	0,833	0,042
Popularität									
Watch	5 %	0,819	0,041	0,856	0,043	0,290	0,015	1,902	0,095
Star	10 %	0,609	0,061	0,926	0,093	0,544	0,054	3,277	0,327
Fork	5 %	0,537	0,027	0,575	0,029	0,417	0,021	3,091	0,155
Aktualität									
Last Commit	10 %	4,800	0,480	4,700	0,470	4,300	0,430	2,000	0,200
Release Date	5 %	2,200	0,110	2,750	0,138	0,000	0,000	2,000	0,100
Issues & Requests	5 %	0,475	0,024	0,171	0,009	0,078	0,004	0,281	0,014
Reputation									
Commits	5 %	4,765	0,238	0,222	0,011	0,399	0,020	0,704	0,035
Contributors	5 %	0,882	0,044	0,980	0,049	0,196	0,010	2,157	0,108
Organisation	10 %	3,000	0,300	5,000	0,500	1,000	0,100	4,000	0,400
Dokumentation									
GitHub-Readme	5 %	3,500	0,175	5,000	0,250	4,000	0,200	5,000	0,250
Webseite	10 %	3,000	0,300	5,000	0,500	0,000	0,000	3,000	0,300
API	5 %	4,000	0,200	0,000	0,000	0,000	0,000	0,000	0,000
Nutzwerte (NW):			**2,658**		**3,090**		**1,286**		**2,335**

Tabelle 8.4: Scoring-Tabelle 1 zur Nutzwertanalyse

Die Softwarelösung Coach erreicht mit 3,090 Gesamtpunkten den höchsten Nutzwert in dieser ersten Gruppe. Das liegt vor allem am Funktionsumfang von Coach. Diese Lösung bietet mit Abstand die meisten implementierten Algorithmen, Experimente und Schnittstellen im Vergleich zu allen anderen an. Außerdem ist eine sehr gute Dokumentation zum Einstieg vorhanden. Mit dem US-amerikanischen Chiphersteller Intel steht auch ein namhaftes und finanzstarkes Unternehmen hinter dem Projekt. Coach ist verglichen mit anderen Alternativen jedoch noch nicht sehr populär. Das liegt sicherlich auch daran, dass die ersten Quellcodes zu diesem Projekt erst am 01.10.2017 auf der Plattform GitHub veröffentlicht wurden. Es ist eben noch ein relativ junges Projekt und Produkt.

Die nächste Tabelle 8.5 enthält die berechneten Punktwerte für die Lösungen OpenAI Baselines (vgl. Kap. 5.5), OpenAI Lab (vgl. Kap. 5.6), Ray RLlib (vgl. Kap. 5.7) und RL Lab (vgl. Kap. 5.9).

Alternativen:		**Baselines**		**OpenAI Lab**		**Ray RLlib**		**RL Lab**	
Merkmal	Gew.	Pkt.	NW	Pkt.	NW	Pkt.	NW	Pkt.	NW
Funktionalität									
Algorithmen	10 %	2,500	0,250	2,500	0,250	1,389	0,139	2,222	0,222
Experimente	5 %	1,957	0,098	2,609	0,130	3,261	0,163	3,261	0,163
Schnittstellen	5 %	1,667	0,083	0,833	0,042	1,667	0,083	1,667	0,083
Popularität									
Watch	5 %	3,602	0,180	0,265	0,013	1,650	0,083	1,877	0,094
Star	10 %	5,000	0,500	0,353	0,035	2,846	0,285	2,242	0,224
Fork	5 %	5,000	0,250	0,259	0,013	1,802	0,090	2,889	0,144
Aktualität									
Last Commit	10 %	4,800	0,480	0,000	0,000	5,000	0,500	1,300	0,130
Release Date	5 %	0,000	0,000	0,000	0,000	1,700	0,085	0,000	0,000
Issues & Requests	5 %	0,102	0,005	0,614	0,031	5,000	0,250	0,226	0,011
Reputation									
Commits	5 %	0,315	0,016	5,000	0,250	4,341	0,217	0,457	0,023
Contributors	5 %	3,137	0,157	0,196	0,010	3,824	0,191	2,941	0,147
Organisation	10 %	5,000	0,500	5,000	0,500	4,000	0,400	5,000	0,500
Dokumentation									
GitHub-Readme	5 %	2,000	0,100	3,500	0,175	2,000	0,100	3,000	0,150
Webseite	10 %	0,000	0,000	5,000	0,500	5,000	0,500	5,000	0,500
API	5 %	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
Nutzwerte (NW):			**2,619**		**1,949**		**3,086**		**2,392**

Tabelle 8.5: Scoring-Tabelle 2 zur Nutzwertanalyse

In dieser Gruppe schneidet Ray RLlib der University of California Berkeley mit einem Nutzwert von 3,086 am besten ab. Die Softwarelösung ist sehr aktuell, viele Wissenschaftler und Programmierer leisten einen wertvollen Beitrag und die Dokumentation auf der Webseite ist sehr gut. Hervorzuheben ist auch das *Hyperparameter Search Tool*, mit dem sich sehr einfach Parameterstudien durchführen lassen. Außerdem wird das Verteilen und Trainieren von Multi-Agenten unterstützt.

Bemerkenswert ist, dass die beiden Softwarelösungen Ray RLlib und Coach in dieser Nutzwertanalyse fast identische Gesamtpunktzahlen erreicht haben, auch wenn sie sich ansonsten stark voneinander unterscheiden.

Die dritte Tabelle 8.6 fasst schließlich noch die Zahlen zu den Softwarelösungen TF Agents (vgl. Kap. 5.10), TensorForce (vgl. Kap. 5.11) und TensorLayer (vgl. Kap. 5.12) zusammen.

Alternativen:		**TF Agents**		**TensorForce**		**TensorLayer**	
Merkmal	Gew.	Pkt.	NW	Pkt.	NW	Pkt.	NW
Funktionalität							
Algorithmen	10 %	0,278	0,028	3,056	0,306	1,667	0,167
Experimente	5 %	0,217	0,011	1,957	0,098	1,304	0,065
Schnittstellen	5 %	0,833	0,042	3,333	0,167	0,833	0,042
Popularität							
Watch	5 %	0,794	0,040	1,348	0,067	5,000	0,250
Star	10 %	0,943	0,094	1,894	0,189	4,821	0,482
Fork	5 %	0,556	0,028	1,258	0,063	4,905	0,245
Aktualität							
Last Commit	10 %	4,800	0,480	4,800	0,480	5,000	0,500
Release Date	5 %	4,250	0,213	4,000	0,200	4,950	0,248
Issues & Requests	5 %	0,097	0,005	1,204	0,060	0,973	0,049
Reputation							
Commits	5 %	0,119	0,006	4,495	0,225	4,949	0,247
Contributors	5 %	0,882	0,044	2,549	0,128	5,000	0,250
Organisation	10 %	5,000	0,500	4,000	0,400	4,000	0,400
Dokumentation							
GitHub-Readme	5 %	3,000	0,150	5,000	0,250	5,000	0,250
Webseite	10 %	0,000	0,000	5,000	0,500	5,000	0,500
API	5 %	0,000	0,000	5,000	0,250	5,000	0,250
Nutzwerte (NW):			**1,640**		**3,382**		**3,945**

Tabelle 8.6: Scoring-Tabelle 3 zur Nutzwertanalyse

TensorLayer erhält einem Nutzwert von 3,945 Punkten. Das ist das beste Ergebnis aller 11 Softwarelösungen. Obwohl kein namhaftes Unternehmen hinter diesem Projekt steckt, sondern die Software vom Doktoranten Hao Dong und Kollegen am Imperial College London entwickelt wurde, schneidet sie in fast allen Kategorien sehr gut ab. Es ist eine populäre, aktuelle, sehr gut dokumentierte Softwarelösung, die viele Unterstützer hat. Lediglich die Funktionalität ist ein Schwachpunkt, den man kritisieren kann. Hier ist noch Potenzial nach oben, d.h. es könnten noch weitere Algorithmen, Experimente und Schnittstellen programmiert werden.

TensorForce ist gemäß der Nutzwertanalyse mit einem Gesamtscore von 3,382 die zweitbeste aller Alternativen. Auch diese Softwarelösung wird nicht durch ein namhaftes Unternehmen unterstützt, sondern deren Entwicklung wird maßgeblich vom Doktorant Michael Schaarschmidt und Kollegen der renommierten Cambridge University vorangetrieben. Die Lösung ist hinsichtlich der fünf Kategorien sehr ausgeglichen, einziger Schwachpunkt ist die Popularität.

Neben Coach und Ray RLlib sollten also auch unbedingt TensorLayer und TensorForce installiert und getestet werden. Die Plattform OpenAI Gym dient allen diesen vier Lösungen als Lieferant für Umgebungen zum Testen.

8.3 Installationen und erste Tests

In diesem Kapitel wird dargestellt, wie die ausgewählten Softwarelösungen OpenAI Gym, Project Malmo, Coach, Ray RLlib, TensorForce und TensorLayer auf dem Testsysten (Ubuntu 16.04 LTS, Python Version 3.6, TensorFlow 1.5.0) installiert und getestet werden.

OpenAI Gym Diese Softwarelösung lässt sich als Minimal-Version oder als Komplett-Version installieren. Mit der Minimal-Installation lassen sich bspw. bereits die klassischen Kontrollaufgaben aus der Literatur ausführen. Möchte man dagegen auch die Simulationen und Spiele als Umgebungen nutzen, dann wird die vollständige Installation benötigt. Die folgenden Listings zeigen die Installationsanweisungen in einer Bash-Shell unter Linux.

```
# Ins Installationsverzeichnis wechseln
cd /install
# Aktuelle Version von GitHub holen
git clone https://github.com/openai/gym.git
# Ins Verzeichnis gym wechseln
cd gym
# Installation per PIP
pip install -e .
```

Listing 8.1: Minimal-Installation von OpenAI Gym – in Anlehnung an [Bro+18a]

Für die Installation der Komplett-Version müssen ggf. zunächst einige zusätzlichen Pakete installiert werden.

```
# Zusätzliche Pakete installieren
sudo apt-get install -y python-numpy python-dev cmake zlib1g-dev libjpeg-
    dev xvfb libav-tools xorg-dev python-opengl libboost-all-dev libsdl2-
    dev swig
# Ins Verzeichnis gym wechseln
pip install -e '.[all]'
```

Listing 8.2: Komplette Installation von OpenAI Gym – in Anlehnung an [Bro+18a]

Einige spezielle Umgebungen basieren auf MuJoCo. Dies ist eine Physik-Engine, mit der realistische Simulationen unter Berücksichtigung physikalischer Gesetzmäßigkeiten durchgeführt werden können. MuJoCo ist jedoch nicht *Open Source*, sondern die Engine benutzt einige proprietäre Bibliotheken. Eine 30-Tage gültige Lizenz zur Nutzung kann man kostenlos bekommen. Dies ist jedoch keine nachhaltige Lösung und aus diesem Grund werden die Umgebungen, die auf MuJoCo basieren nicht weiter betrachtet.

Eine andere Abhängigkeit besteht ggf. zum Python-Paket `pyglet`, welches Multimedia-Funktionen (u.a. Video) anbietet. Hier sollte die Version 1.2.4 verwendet werden. In der Testumgebung war die Version 1.3.0 installiert, sodass bei der Ausführung einiger grafischer Umgebungen Fehler aufgetreten sind. Somit wurde ein Downgrade durchgeführt.

```
# Downgrade pyglet
pip install pyglet==1.2.4
```

Listing 8.3: Downgrade pyglet

Nach der Installation wurden erste Tests in Form von Python-Skripte im Jupyter Notebook durchgeführt, um sicherzustellen, dass sich OpenAI Gym einwandfrei verwenden

lässt. Exemplarisch wird hier die Umgebung Go dargestellt. Go ist ein im asiatischen Raum verbreitetes Brettspiel mit sehr wenigen Regeln, aber sehr vielen Kombinationsmöglichkeiten bezüglich der Stellungen der weißen und schwarzen Spielsteine auf dem Go-Brett. Das normale Brett hat eine Dimension von 19 x 19 Felder. In dieser Umgebung wird ein 9 x 9 Brett benutzt (vgl. Abb. 8.1). Der Python-Quelltext hierzu besteht nur aus wenigen Zeilen und ist weitestgehend selbsterklärend.

```
In [3]: # Board Game
        import gym
        env = gym.make('Go9x9-v0')
        env.reset()
        env.render()

        [2018-01-25 13:24:17,994] Making new env: Go9x9-v0

        To play: black
        Move:   0  Komi: 0.0  Handicap: 0  Captures B: 0 W: 0
              A B C D E F G H J
            +-------------------+
          9 | . . . . . . . . . |
          8 | . . . . . . . . . |
          7 | . . . . . . . . . |
          6 | . . . . . . . . . |
          5 | . . . . . . . . . |
          4 | . . . . . . . . . |
          3 | . . . . . . . . . |
          2 | . . . . . . . . . |
          1 | . . . . . . . . . |
            +-------------------+
```

Abbildung 8.1: Gym-Umgebung Go

Project Malmo Auch zu dieser Softwarelösung sollte man zunächst alle abhängigen Pakete installieren.

```
# Abhängige Pakete installieren
sudo apt-get install libboost-all-dev libpython2.7 openjdk-8-jdk lua5.1
    libxerces-c3.1 liblua5.1-0-dev ffmpeg python-tk python-imaging-tk
# Zertifikate aktualisieren
sudo update-ca-certificates -f
```

Listing 8.4: Vorbereitung der Malmo-Installation – in Anlehnung an [BHH+18]

Dann kann man die aktuelle, vorkompilierte und passende Version als ZIP-Archiv herunterladen und entpacken.

```
# Ins Installationsverzeichnis wechseln
cd /install
# Aktuelles Release downloaden (nicht git benutzen!)
wget https://github.com/Microsoft/malmo/releases/download/0.31.0/Malmo
    -0.31.0-Linux-Ubuntu-16.04-64bit_withBoost_Python3.5.zip
# ZIP-Datei entpacken
unzip Malmo-0.31.0-Linux-Ubuntu-16.04-64bit_withBoost_Python3.5.zip
# Verzeichnis umbenennen (mv = move)
mv Malmo-0.31.0-Linux-Ubuntu-16.04-64bit_withBoost_Python3.5 Malmo
```

Listing 8.5: Aktuelle Malmo-Version installieren – in Anlehnung an [BHH+18]

Zum Testen der Installation wird im ersten Schritt Minecraft in einem Terminal gestartet.

```
# Ins Minecraft-Verzeichnis wechseln
cd /install/Malmo/Minecraft
# Minecraft starten
./launchClient.sh
```

Listing 8.6: Minecraft starten – in Anlehnung an [BHH+18]

Ggf. werden zunächst weitere Dateien aus dem Internet heruntergeladen und installiert. Dann öffnet sich ein Fenster mit Minecraft (vgl. Abb. 8.2).

Abbildung 8.2: Malmo: Minecraft

Nun kann man das eigentliche Anwendungsprogramm, d.h. die Umgebung und den Agenten, starten. Hierzu wird in einem zweiten Terminal ein Beispiel aufgerufen.

```
# Ins Beispielverzeichnis wechseln
cd /install/Malmo/Python_Examples
# Python-Programm starten
python run_mission.py
```

Listing 8.7: Malmo-Beispielprogramm ausführen – in Anlehnung an [BHH+18]

Jetzt ändert sich die Umgebung im ersten Minecraft-Fenster und der Agent wird aktiv (vgl. Abb. 8.3). In diesem Beispiel muss er den Ausweg aus einem Labyrinth finden.

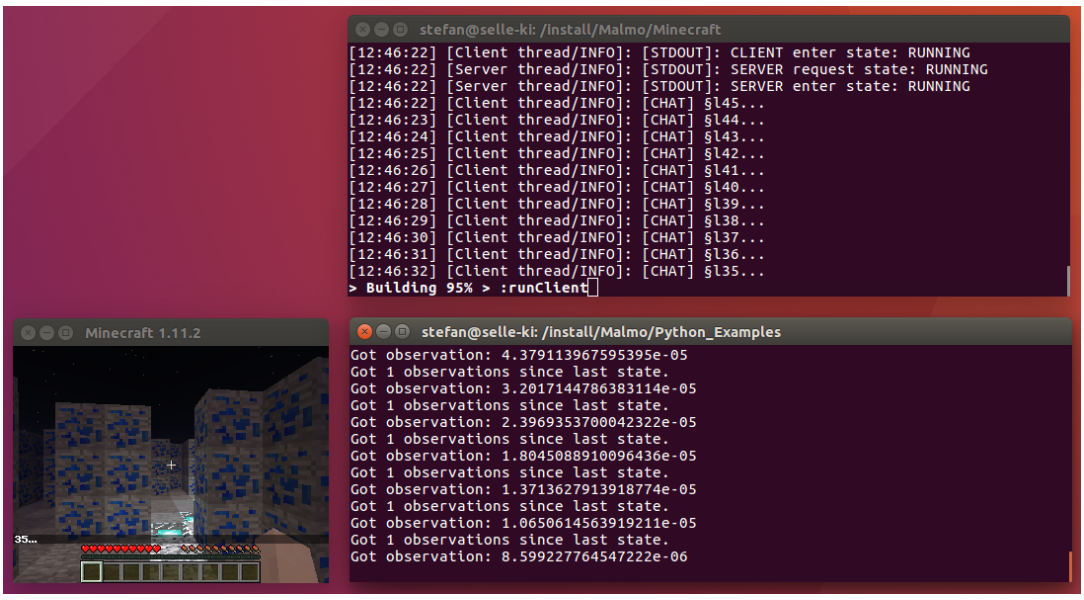


Abbildung 8.3: Malmo: Umgebung und Agent

Coach Zur Installation von Coach wird ein eigenes Skript mitgeliefert, weil sehr viele Abhängigkeiten zu anderen Softwarelösungen bestehen und diese Pakete somit zunächst als Vorbedingungen installiert werden müssen. Dieses Installationsskript startet mit vier Fragen, ob man jeweils die Vorbedingungen für Coach und das Dashboard sowie Schnittstellen zu Gym und Neon unterstützen möchte. Standardmäßig werden diese vier Fragen mit Ja (engl. *Y* für *Yes*) beantwortet.

```
# Ins Installationsverzeichnis wechseln
cd /install
# Aktuelle Version von GitHub holen
git clone https://github.com/NervanaSystems/coach.git
# Ins Verzeichnis coach wechseln
cd coach
# Installationsskript ausführen
./install.sh
```

Listing 8.8: Coach-Installation – in Anlehnung an [CLN+18b]

Es erfolgen interaktive Eingaben:

```
Install Coach requirements? [Y/n] Y
Install Dashboard requirements? [Y/n] Y
Install Gym support? [Y/n] Y
Install neon support? [Y/n] Y
[sudo] Passwort für stefan:
```

Die Installation dauert nun einige Minuten. Am Ende bricht die Installation auf dem Testsystem jedoch leider mit der folgenden Fehlermeldung ab:

```
Please specify the location of python.
Command "python setup.py egg_info" failed with error code 1
  in /tmp/pip-build-55hzjjal/onnx/
Makefile:70: die Regel für Ziel „install“ scheiterte
make: *** [install] Fehler 1
```

Wenn man die Unterstützung für Neon ausschaltet, dann gelingt die Installation. Neon ist wie TensorFlow eine *Deep Learning* Bibliothek. Da TensorFlow bereits auf dem Testsystem installiert ist, wird Neon also auch nicht benötigt. Zum Testen wird nun ein Beispiel-Pythonskript aufgerufen.

```
# Beispiel-Experiment aufrufen
python coach.py -r -p CartPole_PG
```

Listing 8.9: Coach testen – in Anlehnung an [CLN+18b]

Dieses bricht aber direkt mit einer Fehlermeldung ab.

```
ImportError: libcublas.so.8.0: cannot open shared object file:
  No such file or directory
```

Coach hat automatisch ebenfalls TensorFlow installiert, allerdings die Version 1.4.1, die wiederum auf CUDA von Nvidia in der Version 8.0 basiert. Auf dem Testsystem ist aber bereits passend zu TensorFlow 1.5.0 die neuere Versionen CUDA 9.1 installiert. Ein Downgrade dieser Versionen soll aber nicht vorgenommen werden, das auf dem Testsystem noch andere Softwarelösungen zu anderen Einsatzzwecken verwendet werden. Deshalb wird Coach nun vorerst nicht weiter betrachtet.

Ray RLlib Ray und RLlib lassen sich direkt mit dem Paketverwaltungsprogramm für Python-Pakete installieren. Die aktuellen Quellcodes werden aber zusätzlich aus dem GitHub-Repository heruntergeladen, damit Beispiele zum Testen zur Verfügung stehen.

```
# Ray installieren
pip install ray
# RLlib installieren
pip install 'ray[rllib]'
# Ins Installationsverzeichnis wechseln
cd /install
# Aktuelle Version von GitHub holen
git clone https://github.com/ray-project/ray
# Beispiel ausführen
python ray/python/ray/rllib/train.py --run DQN --env CartPole-v0
```

Listing 8.10: Ray RLlib Installation – in Anlehnung an [Nis+18]

Die Installation ist erfolgreich und auch das Beispiel lässt sich ausführen.

```
Waiting for redis server at 127.0.0.1:34109 to respond...
Waiting for redis server at 127.0.0.1:18666 to respond...
Allowing the Plasma store to use up to 26.5785GB of memory.
Starting object store with directory /dev/shm and huge page support disabled
Starting local scheduler with the following resources: {'CPU': 8, 'GPU': 1}.
======================================================================
View the web UI at http://localhost:8888/notebooks/ray_ui60938.ipynb?
token=a22a0d88ec9d221f67d44a7af5522a7152a40bb27ae1eeba
======================================================================
== Status ==
Using FIFO scheduling algorithm.
Result logdir: /home/stefan/ray_results/default
PENDING trials:
 - DQN_CartPole-v0_0:        PENDING
WARNING: Serializing objects of type <class 'ray.tune.registry._Registry'> by expanding
  them as dictionaries of their fields. This behavior may be incorrect in some cases.
== Status ==
Using FIFO scheduling algorithm.
Resources used: 1/8 CPUs, 0/1 GPUs
Result logdir: /home/stefan/ray_results/default
RUNNING trials:
 - DQN_CartPole-v0_0:        RUNNING
Making new env: CartPole-v0
Observation shape is (4,)
Not using any observation preprocessor.
...
TrainingResult for DQN_CartPole-v0_0:
  date: 2018-02-19_16-12-36
  episode_len_mean: 25.0
  episode_reward_mean: 25.0
  episodes_total: 40
  experiment_id: e94a61ad6b784eb6be44043eb9a42112
  hostname: selle-ki
  info:
    exploration: 0.902098
    grad_time_ms: 0.0
    num_target_updates: 1
    sample_time_ms: 0.0
    update_time_ms: 0.0
  pid: 2665
  time_this_iter_s: 0.34363269805908203
  time_total_s: 0.34363269805908203
...
```

TensorForce TensorForce wird ebenfalls mit PIP installiert, wobei die aktuellen Quellcodes vom GitHub-Repository benutzt werden.

```
# Ins Installationsverzeichnis wechseln
cd /install
# Aktuelle Version von GitHub holen
git clone https://github.com/reinforceio/tensorforce.git
# Ins Verzeichnis tensorforce wechseln
cd tensorforce
# TensorForce installieren
pip install -e .
# Beispiel ausführen
python examples/quickstart.py
```

Listing 8.11: TensorForce-Installation – in Anlehnung an [SKF+18]

Die Installation ist erfolgreich, aber das Beispiel bricht direkt mit einem Fehler ab:

```
AttributeError: 'AdamOptimizer' object has no attribute '_beta1_power'
```

Es konnte noch nicht festgestellt werden, was genau die Ursache für diesen Fehler ist.

TensorLayer Die Installation von TensorLayer läuft nach dem gleichen Schema wie bei TensorForce (s.o.) ab.

```
# Ins Installationsverzeichnis wechseln
cd /install
# Aktuelle Version von GitHub holen
git clone https://github.com/zsdonghao/tensorlayer.git
# Ins Verzeichnis tensorlayer wechseln
cd tensorlayer
# TensorLayer installieren
pip install -e .
# Beispiel ausführen
python example/tutorial_cartpole_ac.py
```

Listing 8.12: TensorLayer-Installation – in Anlehnung an [DM+18]

Die Installation ist erfolgreich und auch das Beispiel lässt sich ausführen.

```
Making new env: CartPole-v0
observation dimension: 4
observation high: [  4.80000000e+00   3.40282347e+38   4.18879020e-01   3.40282347e+38]
observation low : [ -4.80000000e+00  -3.40282347e+38  -4.18879020e-01  -3.40282347e+38]
num of actions: 2
I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:898]
  successful NUMA node read from SysFS had negative value (-1),
  but there must be at least one NUMA node, so returning NUMA node zero
I tensorflow/core/common_runtime/gpu/gpu_device.cc:1206] Found device 0 with properties:
  name: GeForce GTX 1080 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.683
  pciBusID: 0000:01:00.0
  totalMemory: 10.91GiB freeMemory: 378.19MiB
I tensorflow/core/common_runtime/gpu/gpu_device.cc:1306] Adding visible gpu devices: 0
I tensorflow/core/common_runtime/gpu/gpu_device.cc:987]
  Creating TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 99 MB memory)
  -> physical GPU (device: 0, name: GeForce GTX 1080 Ti,
      pci bus id: 0000:01:00.0, compute capability: 6.1)
Episode: 0 reward: -10.000000 running_reward -10.000000 took: 0.29932
Episode: 1 reward:    3.000000 running_reward   -9.350000 took: 0.03275
...
```

Andere Softwarelösungen Nur zwei der vier ausgewählten Softwarelösungen konnten erfolgreich installiert und getestet werden. Es stellt sich nun die Frage, wie sich dies mit den anderen sieben Softwarelösungen verhält, die zwar in der Nutzwertanalyse bewertet, aber nicht in die nähere Auswahl gekommen sind (vgl. Kap. 8.2). Also werden auch diese Lösungen noch installiert und getestet (vgl. Tab. 8.7).

Nr	Name	Installation	Test
1	Chainer-RL	o.k.	CUDA-Problem
2	DeepRL	o.k.	Fehlende Module
3	Keras-rl	o.k.	o.k.
4	OpenAI Baselines	MuJoCo-Problem	-
5	OpenAI Lab	o.k.	o.k.
6	RL Lab	Fehler wie bei Coach (s.o.)	-
7	TensorFlow Agents	o.k.	o.k.

Tabelle 8.7: Ergebnis von Installation und Test der übrigen sieben Softwarelösungen

Von diesen sieben Lösungen sind also nur drei einsatzbereit. TensorFlow Agents hat derzeit genau einen Algorithmus implementiert und belegte u.a. deshalb auch in der Nutzwertanalyse mit 1,640 Punkten den vorletzten Platz. Keras-rl kam immerhin auf 2,335 Punkte und OpenAI Lab erreichte 1,949 Punkte. Diese beiden Lösungen sind im Mittelfeld der Nutzwertanalyse zu finden. Sie lassen sich analog zum Vorgehen wie bei TensorLayer (s.o.) installieren. Zusammen mit Ray RLlib und TensorLayer sollen sie im nächsten Kapitel weiter analysiert werden.

9 Experimente

Die vier gefilterten Softwarelösungen Keras-rl, OpenAI Lab, Ray RLlib und TensorLayer, die auf dem Testsystem erfolgreich installiert und getestet werden konnten (vgl. Kap. 8.3) sollen in diesem Kapitel nun genauer analysiert und miteinander verglichen werden. Hierzu werden die Dokumentationen, insbesondere die APIs und *Open Source* Quelltexte betrachtet. Außerdem werden typische *Reinforcement Learning* Experimente durchgeführt und schließlich ein Fazit gezogen.

9.1 Keras-RL

Die Keras-RL-Bibliothek (vgl. 5.4) kennt fünf verschiedene Agenten, die zur Lösung von *Reinforcement Learning* Problemen eingesetzt werden können (vgl. Tab. 9.1). Eine Besonderheit stellt dabei der `DQNAgent` dar, weil zu diesem verschiedene Varianten, d.h. verschiedene Algorithmen, implementiert sind. Alle Agenten können sowohl diskrete als auch kontinuierliche Beobachtungen machen, allerdings entweder nur diskrete oder kontinuierliche Aktionen ausführen [Pla18b].

Name	**Implementierung**	**Quelle**	**Aktionen**
Cross-Entropy Method	CEMAgent	[SL06]	Diskret
Deep Deterministic Policy Gradients	DDPGAgent	[Lil+15]	Kontin.
Deep Q-Network	DQNAgent	[Mni+13]	Diskret
		[Mni+15]	
Double DQN		[vGS15]	
Dueling DQN		[WFL15]	
Normalized Advantage Function	NAFAgent	[Gu+16]	Kontin.
State–Action–Reward–State–Action	SARSAAgent	[SB98]	Diskret

Tabelle 9.1: Agenten der Softwarelösung Keras-rl

Jeder dieser Agenten verfügt über die gleichen Funktionen, und zwar `compile`, `fit`, `test`, `get_config`, `reset_states`, `load_weights` und `save_weights`. Die ersten drei genannten Funktionen sind essentiell für die Verwendung der Agenten und deshalb wird das *Application Programming Interface* hierzu mit den folgenden Python-Listings kurz vorgestellt.

```
# Agent für das Tranieren und Testen vorbereiten
compile(optimizer, metrics=[])
```

Listing 9.1: compile()-Funktion

Argumente:

optimizer	Instanz des verwendeten Optimierers (keras.optimizers.Optimizer)
metrics	Liste von Funktionen zwecks Bewertung (wird vom Optimierer benötigt)

```
# Agent trainieren
fit(env, nb_steps, action_repetition=1, callbacks=None, verbose=1,
    visualize=False, nb_max_start_steps=0, start_step_policy=None,
    log_interval=10000, nb_max_episode_steps=None)
```

Listing 9.2: fit()-Funktion

Argumente:

env	Instanz der Umgebung (z.B. von OpenAI Gym)
nb_steps	Anzahl der Trainingsschritte (als ganze Zahl)
action_repetition	Anzahl der Wiederholungsschritte einer Aktion bevor der Agent die Umgebung wieder beobachtet
callbacks	Liste von Callbacks, die im Training ausgeführt werden
verbose	0: keine Ausgabe, 1: Intervallausgabe, 2: Episodenausgabe
visualize	True: Umgebung visualisieren, False: Keine Visualisierung
nb_max_start_steps	Anzahl der max. Anfangsschritte zwecks Exploration
start_step_policy	Policy, der nach nb_max_start_steps gefolgt wird. None für Zufall-Policy
log_interval	Anzahl der Schritte eines Intervalls bei verbose = 1 (s.o.)
nb_max_episode_steps	Anzahl der max. Schritte pro Episode

```
# Agent testen (validieren)
test(env, nb_episodes=1, action_repetition=1, callbacks=None, visualize=
    True, nb_max_episode_steps=None, nb_max_start_steps=0,
    start_step_policy=None, verbose=1)
```

Listing 9.3: test()-Funktion

Argumente:

env	Instanz der Umgebung (z.B. von OpenAI Gym)
nb_episodes	Anzahl der Episoden im Test
action_repetition	Anzahl der Wiederholungsschritte einer Aktion bevor der Agent die Umgebung wieder beobachtet
callbacks	Liste von Callbacks, die während des Tests ausgeführt werden
visualize	True: Umgebung visualisieren, False: Keine Visualisierung
nb_max_episode_steps	Anzahl der max. Schritte pro Episode
nb_max_start_steps	Anzahl der max. Anfangsschritte zwecks Exploration
start_step_policy	Policy, der nach nb_max_start_steps gefolgt wird. None für Zufall-Policy
verbose	0: keine Ausgabe, 1: Intervallausgabe, 2: Episodenausgabe

Für die OpenAI Umgebung *Cart Pole* (vgl. Kap. 2.3 und Kap. 4.4) werden nun die Agenten trainiert. Da die Aktionen in dieser Problemstellung diskret sind, d.h. der Wagen entweder in die eine oder andere horizontale Richtung bewegt werden kann, werden nur die zu dieser Umwelt passenden Agenten bzw. Algorithmen benutzt. Das folgende Listing beschreibt ganz allgemein das Vorgehen. Das Python-Skript ist in vier Blöcke unterteilt: (1) Module importieren, (2) Umgebung initialisieren, (3) Modell erstellen und (4) Agent trainieren und testen.

```
# Allgemeine Module importieren (numpy, OpenAI Gym)
import numpy as np
import gym
# Spezielle Keras-Module importieren
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.optimizers import Adam
# Spezielle Keras-RL-Module importieren
from rl.policy import BoltzmannQPolicy
# TODO 1: Weitere Module importieren (Algorithmus abhängig)

# Umgebung Cart Pole von OpenAI Gym verwenden
ENV_NAME = 'CartPole-v0'
env = gym.make(ENV_NAME)
env.seed(123)
np.random.seed(123)
nb_actions = env.action_space.n # Anzahl möglicher Aktionen

# Modell erstellen
model = Sequential()
model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(nb_actions))
model.add(Activation('linear'))
# model.add(Activation('softmax')) statt linear für CEM
print(model.summary())

# RL-Algorithmus zum Trainieren des Agenten benutzen
ALG_NAME = '' # TODO 2: Namen des Algorithmus verwenden
policy = BoltzmannQPolicy()
agent = None # TODO 3: Instanz des Agenten erzeugen
agent.compile(Adam(lr=1e-3), metrics=['mae']) # ohne Argumente bei CEM
agent.fit(env, nb_steps=50000, visualize=False, verbose=1) # Training
agent.save_weights('{}_{}_weights.h5f'.format(ALG_NAME,ENV_NAME),
    overwrite=True)
agent.test(env, nb_episodes=10, visualize=True) # Ergebnis testen
```

Listing 9.4: Cart Pole – in Anlehnung an [Pla18a]

Dieses Programm ist so noch nicht ausführbar, sondern muss noch ergänzt werden. Dafür ist an drei Stellen im Quelltext des Listings der Begriff `TODO` platziert. An genau diesen Stellen muss der Quelltext komplettiert werden, je nachdem welcher Agent und Algorithmus verwendet wird. In den folgenden Listings werden nur noch diese Programmzeilen für die jeweiligen Agenten / Algorithmen dargestellt.

```
# Spezielle Keras-RL-Module importieren
from rl.agents import SARSAAgent
# SARSA Algorithmus zum Trainieren benutzen
ALG_NAME = 'SARSA'
agent = SARSAAgent(model=model, nb_actions=nb_actions, nb_steps_warmup
    =100, policy=policy)
```

Listing 9.5: Cart Pole & SARSA – in Anlehnung an [Pla18a]

```
# Spezielle Keras-RL-Module importieren
from rl.agents.dqn import DQNAgent
from rl.memory import SequentialMemory
# Deep Q-Network (DQN) zum Trainieren benutzen
ALG_NAME = 'DQN'
memory = SequentialMemory(limit=50000, window_length=1)
agent = DQNAgent(model=model, nb_actions=nb_actions, memory=memory,
    nb_steps_warmup=100, target_model_update=1e-2, policy=policy)
```

Listing 9.6: Cart Pole & DQN – in Anlehnung an [Pla18a]

```
# Spezielle Keras-RL-Module importieren
from rl.agents.dqn import DQNAgent
from rl.memory import SequentialMemory
# Dueling DQN zum Trainieren benutzen
ALG_NAME = 'Dueling-DQN'
memory = SequentialMemory(limit=50000, window_length=1)
agent = DQNAgent(model=model, nb_actions=nb_actions, memory=memory,
    nb_steps_warmup=100, enable_dueling_network=True, dueling_type='avg',
     target_model_update=1e-2, policy=policy)
```

Listing 9.7: Cart Pole & Dueling DQN – in Anlehnung an [Pla18a]

```
# Spezielle Keras-RL-Module importieren
from rl.agents.cem import CEMAgent
from rl.memory import EpisodeParameterMemory
# CEM zum Trainieren benutzen
ALG_NAME = 'CEM'
memory = EpisodeParameterMemory(limit=1000, window_length=1)
agent = CEMAgent(model=model, nb_actions=nb_actions, memory=memory,
    batch_size=50, nb_steps_warmup=2000, train_interval=50, elite_frac
    =0.05)
agent.compile()
```

Listing 9.8: CartPole & CEM – in Anlehnung an [Pla18a]

Wie man in den Listings bereits sieht, wird jeder Agent anders instanziiert, d.h. die Konstruktoren unterscheiden sich hinsichtlich der Parameter bzw. Argumente. Hierzu fehlt allerdings die Dokumentation und die Quellcodes sind leider auch nicht entsprechend kommentiert.

In den folgenden Listings sind die Konstruktoren der drei verwendeten Agenten dargestellt und darunter die Argumente erklärt, soweit hier Informationen zur Verfügung stehen oder sich aus den Quelltexten extrahieren lassen.

```
# Konstruktor zum SARSAAgent
SARSAAgent(model, nb_actions, policy=None, test_policy=None, gamma=0.99,
    nb_steps_warmup=10, train_interval=1, delta_clip=inf)
```

Listing 9.9: SARSAAgent-Konstruktor

Argumente:

model	Ein Keras Modell
nb_actions	Anzahl der möglichen Aktionen des Agenten
policy	Eine Keras-rl Policy
test_policy	Eine Keras-rl Policy zum Testen
gamma	Lernparameter
nb_steps_warmup	Anzahl der Schritte zum Warmmachen
train_interval	Trainingsintervall
delta_clip	Huber Loss Term berechnen [Wik18h]

```
# Konstruktor zum DQNAgent
DQNAgent(model, nb_actions, policy=None, test_policy=None, gamma=0.99,
    batch_size=32, nb_steps_warmup=1000, train_interval=1, memory,
    memory_interval=1,   target_model_update=10000, delta_range=None,
    delta_clip=inf, custom_model_objects={}, enable_double_dqn=True,
    enable_dueling_network=False, dueling_type='avg')
```

Listing 9.10: DQNAgent-Konstruktor

Argumente:

model	Ein Keras Modell
nb_actions	Anzahl der möglichen Aktionen des Agenten
policy	Eine Keras-rl Policy
test_policy	Eine Keras-rl Policy zum Testen
gamma	Lernparameter
batch_size	
nb_steps_warmup	Anzahl der Schritte zum Warmmachen
train_interval	Trainingsintervall
memory	
memory_interval	
target_model_update	
delta_range	
delta_clip	Huber Loss Term berechnen [Wik18h]
custom_model_objects	
enable_double_dqn	Zweites Netzwerk um Overfitting zu reduzieren
enable_dueling_network	Dueling Architektur verwenden
dueling_type	$Q(s,a)$ aus $V(s)$ und $A(s,a)$ berechnen: 'avg', 'max', 'naive'

```
# Konstruktor zum CEMAgent
CEMAgent(model, nb_actions, memory, batch_size=50, nb_steps_warmup=1000,
    train_interval=50, elite_frac=0.05, memory_interval=1, theta_init=
    None, noise_decay_const=0.0, noise_ampl=0.0)
```

Listing 9.11: CEMAgent-Konstruktor

Argumente:

model	Ein Keras Modell
nb_actions	Anzahl der möglichen Aktionen des Agenten
memory	
batch_size	
nb_steps_warmup	Anzahl der Schritte zum Warmmachen
train_interval	Trainingsintervall
elite_frac	
memory_interval	
theta_init	
noise_decay_const	
noise_ampl	

Das Keras-Modell ist eigentlich ein Künstliches Neuronales Netzwerk (KNN), welches aus Schichten besteht, wovon mindestens eine Schicht eine Eingabe und eine Schicht eine Ausgabe darstellt. Eine Schicht (engl. *Layer*) besteht aus künstlichen Neuronen und kann mathematisch als Tensor (eigentlich als Vektor) beschrieben werden. Diese Neuronen benötigen eine Aktivierungsfunktion, typisch sind bspw. 'relu', 'linear' und 'softmax'. Mit dem `Sequential`-Modell können solche Netzwerke aus verschiedenen Schichten konstruiert werden, indem diese einfach hinzugefügt werden [Cho+18]. Die Eingaben sind die Beobachtungen des Agenten und die Ausgaben sind die möglichen Aktionen. Die Schichten dazwischen werden auch als verborgenen Schichten bezeichnet (engl. *Hidden Layers*). Tiefe KNN haben viele solche Schichten und das Training nennt man dann *Deep Learning*.

Für die Strategie bzw. Politik (engl. *Policy*), die der Agent verfolgt, können die folgenden Klassen benutzt werden:

1. LinearAnnealedPolicy
2. EpsGreedyQPolicy
3. GreedyQPolicy
4. BoltzmannQPolicy
5. MaxBoltzmannQPolicy
6. BoltzmannGumbelQPolicy

Einige Agenten benötigen Speicher (engl. *Memory*), um die Ergebnisse aus jedem Trainingsschritt (Beobachtung, Aktion, Belohnung, Endzustand) zu speichern. Zwei verschiedene Klassen können benutzt werden:

1. SequentialMemory
2. EpisodeParameterMemory

Unabhängig von der Wahl des Speichers wird dieser so lange mit neuen Werten gefüllt, bis die Kapazitätsgrenze erreicht ist. Danach werden die ältesten Einträge überschrieben.

Die vier Programme lassen sich auch im Jupyter Notebook ausführen. In Abb. 9.1 bis 9.3 ist dies exemplarisch für SARSA dargestellt.

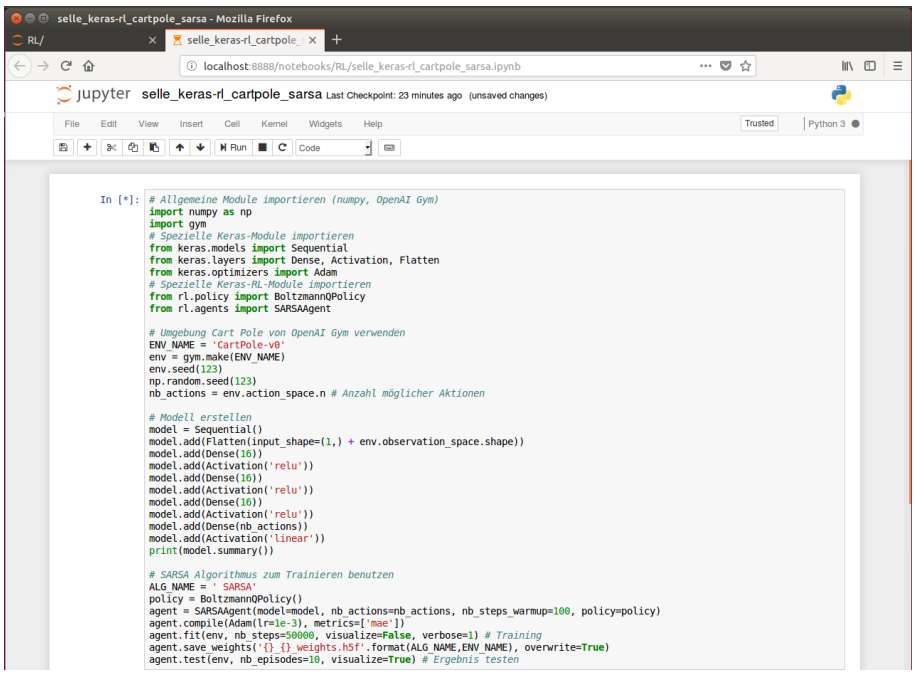


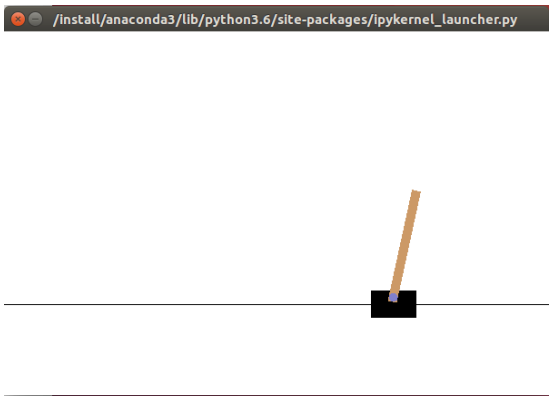
Abbildung 9.1: Jupyter Notebook: Keras-rl Cart Pole SARSA: Python Quelltext

```
Using TensorFlow backend.
[2018-02-20 18:03:49,848] Making new env: CartPole-v0
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 4)	0
dense_1 (Dense)	(None, 16)	80
activation_1 (Activation)	(None, 16)	0
dense_2 (Dense)	(None, 16)	272
activation_2 (Activation)	(None, 16)	0
dense_3 (Dense)	(None, 16)	272
activation_3 (Activation)	(None, 16)	0
dense_4 (Dense)	(None, 2)	34
activation_4 (Activation)	(None, 2)	0

```
Total params: 658
Trainable params: 658
Non-trainable params: 0
_________________________________________________________________
None
Training for 50000 steps ...
Interval 1 (0 steps performed)
10000/10000 [==============================] - 31s 3ms/step - reward: 1.0000
269 episodes - episode_reward: 36.506 [9.000, 200.000] - loss: 7.773 - mean_absolute_error: 14.350 - mean_q: 27.749

Interval 2 (10000 steps performed)
10000/10000 [==============================] - 30s 3ms/step - reward: 1.0000
100 episodes - episode_reward: 100.080 [9.000, 200.000] - loss: 8.567 - mean_absolute_error: 22.765 - mean_q: 45.230

Interval 3 (20000 steps performed)
10000/10000 [==============================] - 31s 3ms/step - reward: 1.0000
81 episodes - episode_reward: 125.383 [15.000, 200.000] - loss: 6.619 - mean_absolute_error: 24.978 - mean_q: 49.787

Interval 4 (30000 steps performed)
10000/10000 [==============================] - 31s 3ms/step - reward: 1.0000
96 episodes - episode_reward: 103.979 [9.000, 200.000] - loss: 5.244 - mean_absolute_error: 22.927 - mean_q: 45.679

Interval 5 (40000 steps performed)
 1392/10000 [===>..........................] - ETA: 26s - reward: 1.0000
```

Abbildung 9.2: Jupyter Notebook: Keras-rl Cart Pole SARSA: Training




```
Testing for 10 episodes ...
Episode 1: reward: 111.000, steps: 111
Episode 2: reward: 103.000, steps: 103
Episode 3: reward: 89.000, steps: 89
Episode 4: reward: 89.000, steps: 89
Episode 5: reward: 86.000, steps: 86
Episode 6: reward: 101.000, steps: 101
Episode 7: reward: 99.000, steps: 99
Episode 8: reward: 89.000, steps: 89
Episode 9: reward: 94.000, steps: 94
Episode 10: reward: 103.000, steps: 103
```

Abbildung 9.3: Jupyter Notebook: Keras-rl Cart Pole SARSA: Testergebnis

Abb. 9.1 zeigt den Quelltext, Abb. 9.2 ist während des Trainings aufgenommen und Abb. 9.3 zeigt die Testergebnisse. Während des Trainings kann man auch beobachten, dass die Rechnungen auf der GPU ausgeführt werden. Hierzu muss nur der Befehl `nvidia-smi` aufgerufen werden.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 387.34                 Driver Version: 387.34                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:01:00.0 Off |                  N/A |
| 28%   54C    P2    72W / 250W |  10672MiB / 11172MiB |     21%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1080      G   /usr/lib/xorg/Xorg                           186MiB |
|    0      2434      G   compiz                                        90MiB |
|    0      7242      C   /install/anaconda3/bin/python              10383MiB |
+-----------------------------------------------------------------------------+
```

Tabelle 9.2 zeigt die Ergebnisse nach dem Training der vier Agenten. Diese wurden jeweils 50.000 Episoden trainiert. In der Testphase werden dann 10 Episoden mit maximal 200 Schritten durchgeführt. Die maximale Belohnung (engl. *Reward*) beträgt 200 Punkte, wenn das Pendel 200 Schritte lang ausbalanciert bleibt. Nach diesen 10 Test-Episoden wird der mittlere Reward berechnet.

	SARSA	**DQN**	**Duel**	**CEM**
Reward	96	200	200	190
Zeit [s]	154,4	202,2	227,3	46,4

Tabelle 9.2: Ergebnisse der Simulationen mit den Agenten / Algorithmen für Cart Pole

Wie man in den Ergebnissen sieht, schneiden die Agenten bzw. Algorithmen DQN und Dueling DQN am besten ab. Sie erreichen beide die maximale Punktzahl. Der SARSA-Agent erreicht nach 50.000 Episoden keine zufriedenstellenden Ergebnisse und kommt im Mittel nur auf 96 von 200 Punkten. Ggf. muss das Training hier verlängert oder andere Parameter verwendet werden. Der CEM-Agent erreicht mit einem Durchschnitt von 190

Punkten ein gutes Ergebnis. Bemerkenswert ist die schnelle Trainingszeit von nur 46,4 Sekunden für die 50.000 Episoden im Vergleich zu den anderen Agenten.

9.2 OpenAI Lab

Wenn man sich die Quelltexte des Frameworks OpenAI Labs ansieht, stellt man schnell fest, dass im Verzeichnis `rl/agent` insgesamt elf Agenten implementiert sind (vgl. Tab. 9.3). Teilweise sind auf der Webseite [KG18b] oder in den Quelltexten [KG18a] auch Quellen zu der jeweiligen Implementierung angegeben.

Name	Implementierung	Quelle	Policy
QTable	QTable	[SB98]	
Actor Critic	ActorCritic	[Sil+14][Ret16]	
Deep Q-Network	DQN	[Mni+13]	1, 2, 3, 4
Double Deep Q-Network	DoubleDQN	[vGS15]	5, 6
Freeze DQN	FreezeDQN	[Sil18]	1, 2, 3, 4
Conv. Deep Q-Network	ConvDQN	-	1, 2, 3, 4
Double Conv. DQN	DoubleConvDQN	-	5, 6
DeepSarsa	DeepSarsa	-	1, 2, 3, 4
DeepExpectedSarsa	DeepExpectedSarsa	-	1, 2, 3, 4
OffPolicySarsa	OffPolicySarsa	-	1, 2, 3, 4
Deep Deterministic Policy Gradients	DDPG	[Lil+15][Ema18]	

Tabelle 9.3: Agenten der Softwarelösung OpenAI Lab

Die folgenden Policies sind implementiert und können ggf. mit den Agenten kombiniert werden (vgl. Tab. 9.3):

1 EpsilonGreedyPolicy
2 OscillatingEpsilonGreedyPolicy
3 TargetedEpsilonGreedyPolicy
4 BoltzmannPolicy
5 DoubleDQNPolicy
6 DoubleDQNBoltzmannPolicy

OpenAI Lab bietet ein Pythonskript `main.py` an, mit dem sich bereits sehr viele Experimente direkt ausführen lassen. Ein Experiment wiederum wird in einer JSON-Datei im Verzeichnis `rl/spec/` beschrieben. Das Skript kann mit dem folgenden Aufruf gestartet werden.

```
# Experiment starten
python main.py <argumente>
```

Listing 9.12: Kommandozeilenskript ausführen

Verschiedene Argumente können benutzt werden, um genauer zu spezifizieren welches Experiment wie ausgeführt werden soll. Mindestens die Option `-e` sollte angegeben werden.

Argument	Wert	Default	Erklärung
-b		False	Keine Grafikausgabe (*blind mode*)
-d		False	Logging der Debug-Ausgabe
-e	<experiment>	dev_dqn	Name oder ID des Experiments
-p		False	Parameterauswahl
-q		False	Nur Warnungen ausgeben (*quiet mode*)
-t	<times>	1	Anzahl Sessions
-x	<max_ep>	-1	Max. Anzahl der Episoden
			-1: benutze Werte in rl/spec/problems.json

Tabelle 9.4: Argumente zum Kommandozeilenskript der Softwarelösung OpenAI Lab

Zur Beschreibung eines Experiments müssen einige Angaben in der JSON-Datei gemacht werden.

experiment_name	Eindeutiger Name des Experiments
problem	Name des Problems (z.B. aus OpenAI Gym)
Agent	Name des Agenten (vgl. Tab. 9.3)
Policy	Name der Policy (s.o.)
Memory	Speicher
Optimizer	Optimierer
HyperOptimizer	Hyper-Optimierer
Preprocessor	Präprozessor
param	Parameter für den Optimierer, Agent usw.
param_range	Parameter für den Hyper-Optimierer

Der Speicher (*Memory*) wird benötigt, um die gemachten Erfahrungen festzuhalten. Es kann eine der folgenden Implementierungen gewählt werden:

1 LinearMemory

2 LinearMemoryWithForgetting

3 LeftTailMemory

4 PrioritizedExperienceReplay

5 RankedMemory

Außerdem können drei Optimierer (*Optimizer*) aus dem Framework Keras [Cho+18] verwendet werden, um die Policy mittels Gradientenabstieg zu verbessern:

1 SGDOptimizer (Stochastic Gradient Descent)

2 AdamOptimizer (Adaptive Moment Estimation)

3 RMSpropOptimizer (Root Mean Square Propagation)

Der Hyper-Optimierer (*HyperOptimizer*) kann benutzt werden, um einige Parameter des Agenten zu optimieren:

1 RandomSearch

2 LineSearch

3 GridSearch

Der Präprozessor (*Preprocessor*) wird nur benötigt, falls die Ausgaben der Umgebung transformiert werden müssen, damit sie der Agent als Input verarbeiten kann (z.B. bei Atari-Spielen). Nachfolgend ist ein Beispiel für die Beschreibung eines Experiments mit dem Namen `dqn` dargestellt.

```
{
  "Agent": "DQN",
  "Policy": "BoltzmannPolicy",
  "Memory": "LinearMemoryWithForgetting",
  "Optimizer": "AdamOptimizer",
  "HyperOptimizer": "GridSearch",
  "PreProcessor": "NoPreProcessor",
  "experiment_name": "dqn",
  "problem": "CartPole-v0",
  "param": {
    "gamma": 0.99,
    "lr": 0.02,
    "hidden_layers": [64],
    "hidden_layers_activation": "sigmoid",
    "exploration_anneal_episodes": 10
  }
}
```

Listing 9.13: Beispiel-Experiment

Für das bekannte RL-Problem *Cart Pole* (vgl. Kap. 2.3 und Kap. 4.4) werden nun wieder Experimente mit der entsprechenden OpenAI Gym Umgebung `CartPole-v0` und mit den Agenten bzw. Algorithmen SARSA, DQN und AC durchgeführt. Der Algorithmus DDQN konnte leider nicht verwendet werden, weil bei der Initialisierung des entsprechenden Agenten immer ein Fehler aufgetreten ist. Die Ursache dafür konnte bislang nicht gefunden werden.

Auf den drei folgenden Seiten sind jeweils die Parametereinstellungen zu diesen drei Experimenten und grafische Visualisierungen des Trainingsverlauf dargestellt (vgl. Abb. 9.4 bis 9.6). Im jeweils oberen Diagramm ist der Reward über die Episoden aufgetragen, im mittleren Diagramm der mittlere Reward (Mittelung über die letzten 100 Episoden) und im unteren Diagramm die Kostenfunktion (*Policy Loss*) zu dem Künstlichen Neuronalen Netzwerk, welches mit dem Optimierer trainiert wird. Meistens ist die Kostenfunktion durch die Summe der mittleren quadratischen Fehler definiert, welche durch ein Gradientenverfahren minimiert werden sollen.

SARSA Parameterwerte:

Agent	OffPolicySarsa
Policy	EpsilonGreedyPolicy
Memory	LinearMemoryWithForgetting
Optimizer	AdamOptimizer
HyperOptimizer	GridSearch
Preprocessor	NoPreProcessor
param	gamma: 0,97; lr: 0,005
	hidden_layers: 32
	hidden_layers_activation: sigmoid
	exploration_anneal_episodes: 20

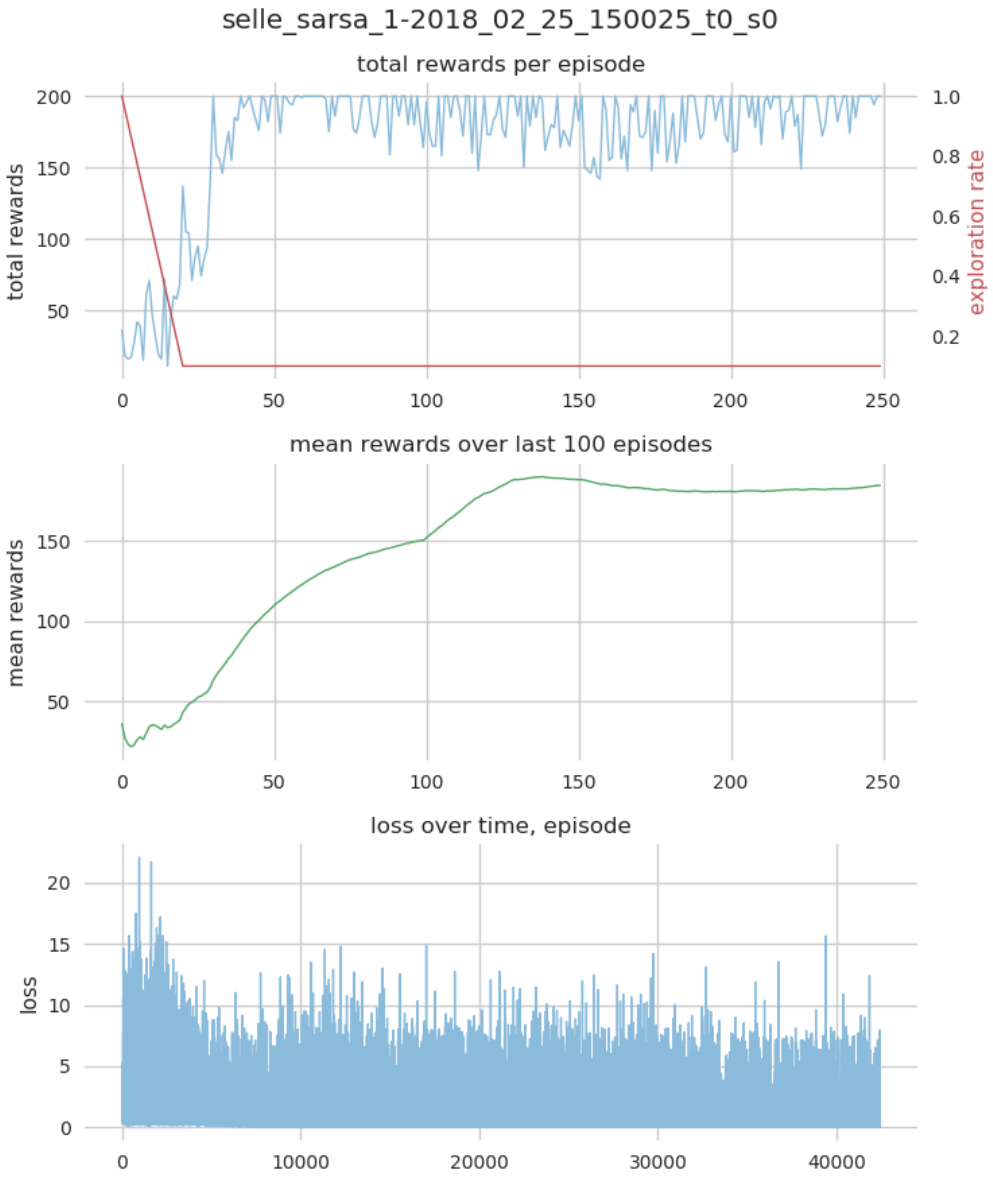


Abbildung 9.4: CartPole & SARSA

DQN Parameterwerte:

Agent	DQN
Policy	BoltzmannPolicy
Memory	LinearMemoryWithForgetting
Optimizer	AdamOptimizer
HyperOptimizer	GridSearch
Preprocessor	NoPreProcessor
param	gamma: 0,99; lr: 0,02
	hidden_layers: 64
	hidden_layers_activation: sigmoid
	exploration_anneal_episodes: 10

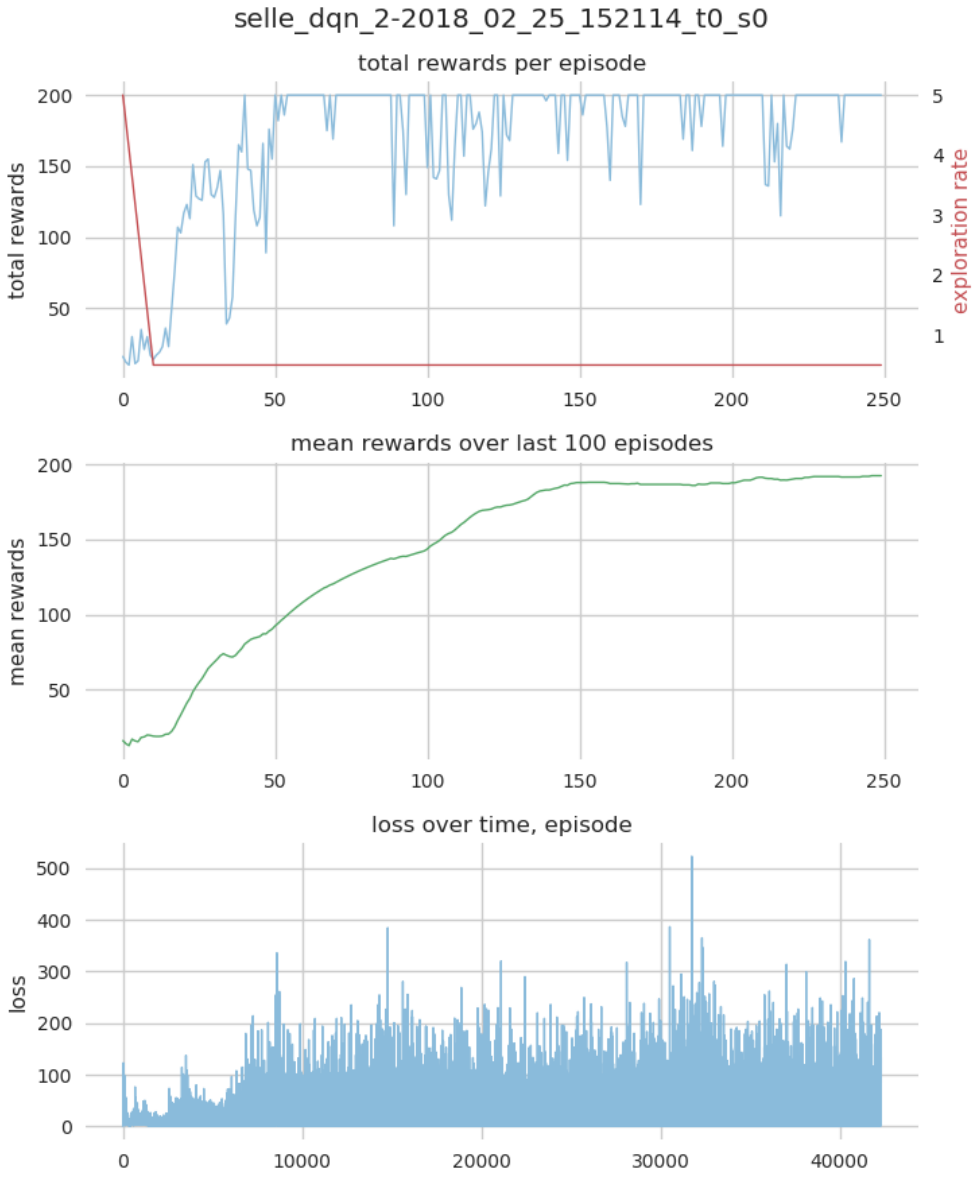


Abbildung 9.5: CartPole & DQN

AC Parameterwerte:

Agent	ActorCritic
Policy	SoftmaxPolicy
Memory	LinearMemoryWithForgetting
Optimizer	AdamOptimizer
HyperOptimizer	GridSearch
Preprocessor	NoPreProcessor
param	gamma: 0,999; lr: 0,05
	hidden_layers: 64
	hidden_layers_activation: sigmoid

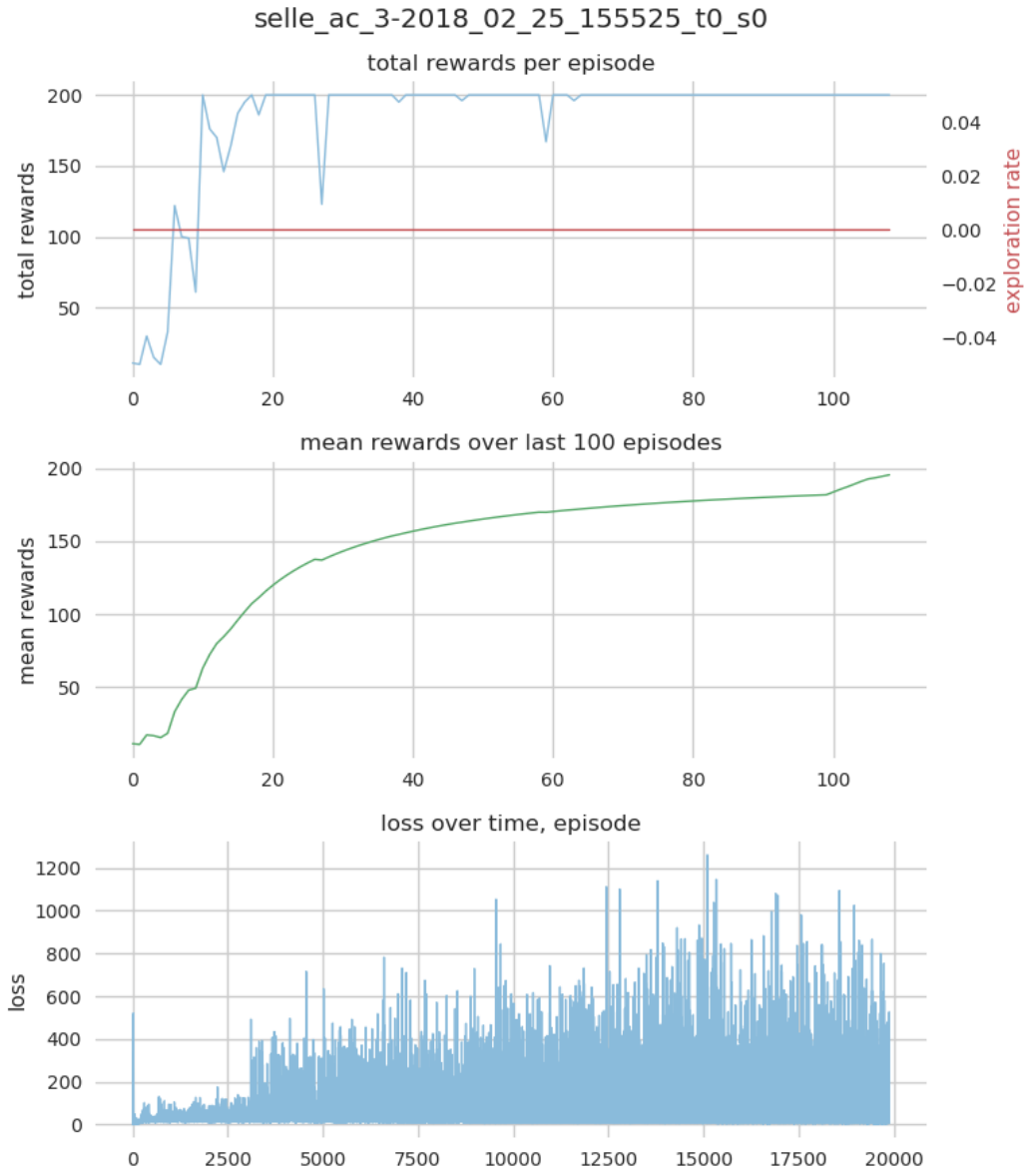


Abbildung 9.6: CartPole & AC

Die Ergebnisse dieser drei Experimente sind in Tabelle 9.5 zusammengefasst.

	SARSA	**DQN**	**AC**
Lösung in Episode	249	249	108
Leistung P	0,74	0,77	1,79
Stabilität S	0,48	0,74	0,91
Fitness Score F	1,10	1,34	13,70
Mittlerer Reward	184,51	192,51	195,62
Benötigte Zeit [s]	456	443	353

Tabelle 9.5: Ergebnisse der Simulationen mit den Agenten / Algorithmen für Cart Pole

Der Fitness Score F wird berechnet durch:

$$F = P \cdot \left[(1+S) \cdot (1+C)^2\right]^{\text{sgn}(P)} \tag{9.1}$$

mit `sgn` als Vorzeichenfunktion (signum). Der Fitness Score wurde entworfen, um mit nur einem Wert das Training von Algorithmen bzw. Agenten in verschiedenen Umgebungen vergleichbar zu machen und um die folgenden Punkte zu erfassen:

- Stärke (*Strength*): Mittlerer Reward der letzten 100 Episoden
- Geschwindigkeit (*Speed*): 1 / Episoden
- Leistung (*Power*) P: Stärke * Geschwindigkeit
- Stabilität (*Stability*) S: gelöste Episoden / (letzte Episode - erste gelöste Episode)
- Konsistenz (*Consistency*) C: gelöste Episoden / Episoden
- Granularität (*Granularity*): Terme $(1+S)$ und $(1+C)$
- Verstärkung (*Amplification*) A: $(1+S) \cdot (1+C)^2$
- Unterscheidungsfähigkeit (*Distinguish ability*): $A^{\text{sgn}(P)}$

Der *Actor Critic* Algorithmus erzielte mit 195,62 den höchsten mittleren Reward. Es ist auch das Experiment, welches am wenigsten Zeit benötigte und den besten Fitnesswert lieferte. Während des Trainings kann man auch beobachten, dass die Rechnungen wieder auf der GPU ausgeführt werden. Diesmal wird mit 242 MiB auch deutlich weniger Speicher benötigt als bei Keras-rl mit 10383 MiB (vgl. Kap. 9.1).

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 387.34                 Driver Version: 387.34                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:01:00.0 Off |                  N/A |
|  0%   55C    P2    64W / 250W |    492MiB / 11172MiB |     24%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1086      G   /usr/lib/xorg/Xorg                           195MiB |
|    0      2309      G   compiz                                        41MiB |
|    0      5556    C+G   python                                       242MiB |
+-----------------------------------------------------------------------------+
```

9.3 Ray RLlib

Diese Bibliothek zu *Reinforcement Learning* bietet sechs Agenten zum Trainieren an (vgl. Tab. 9.6), vier davon werden im Artikel von Liang et al.[Lia+18a] beschrieben, zwei weitere (BC und PG) sind aber ebenfalls implementiert, wenn man sich den Quelltext zu dieser Bibliothek genauer ansieht [Nis+18].

Name	Implementierung	Quelle	Policy
Asynchronous Advantage Actor Critic	A3CAgent	[Mni+16]	TFPolicy
Behavioral Cloning	BCAgent	[RGB11]	BCPolicy
Deep Q-Network	DQNAgent	[Mni+13]	
Evolution Strategies	ESAgent	[Sal+17]	GenericPolicy
Policy Gradient	PGAgent	[SB98]	PGPolicy
Proximal Policy Optimization	PPOAgent	[Sch+17]	

Tabelle 9.6: Agenten der Softwarelösung Ray RLlib

Jeder dieser Agenten verfügt über einen `Policy Optimizer`. Über Interaktionen mit der Umgebung kann der Agent die Policy trainieren, hierzu wird die Funktion `train` verwendet (vgl. Abb. 9.7).

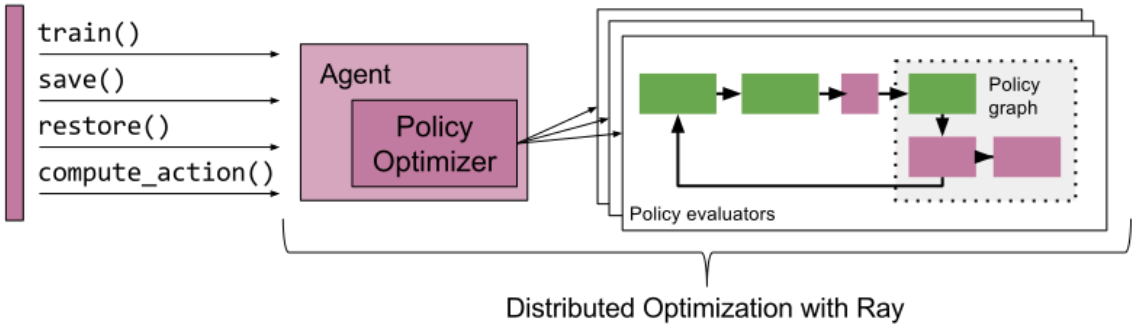


Abbildung 9.7: Ray RLlib API [Lia+18b]

Weitere Funktionen dienen zum Speichern (`save`) und Wiederherstellen (`restore`) des aktuellen Trainingszustands sowie zum Anwenden (`compute_action`) des trainierten Agenten. Geht man etwas weiter ins Detail (vgl. Abb. 9.8), dann startet der Trainingsablauf mit der Umgebung (*environment*). Diese bekommt eine Aktion (*action*) und erzeugt eine Beobachting (*observation*). Über eine Vorverarbeitung (*preprocessor*) und ein Filter wird das beobachtete Signal weiter an das Modell, i.d.R. ein Künstliches Neuronales Netzwerk (KNN), geleitet. Die Ausgabe von diesem Modell wird dann von der Komponente `ActionDistribution` interpretiert, um die nächste Aktion zu bestimmen und der Umgebung mitzuteilen. Zum Trainieren des KNN wird eine spezielle Kostenfunktion (hier: *Policy Loss*) verwendet, die man versucht zu minimieren.

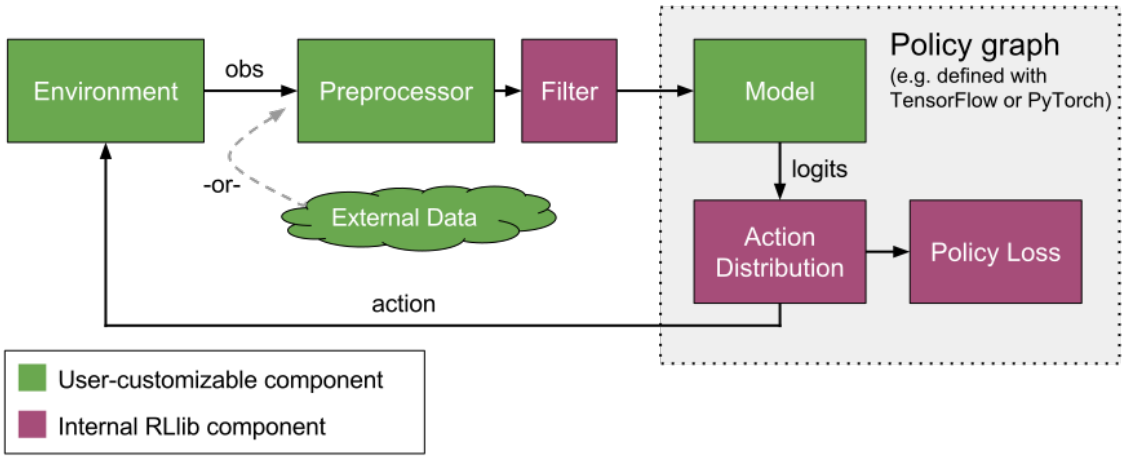


Abbildung 9.8: Ray RLlib Komponenten [Lia+18b]

Alle Komponenten, die in grüner Farbe der Abb. 9.8 dargestellt sind, lassen sich austauschen. Beispielsweise kann man als Umgebung OpenAI Gym *Cart Pole* und als Modell bzw. Agent *DQN* verwenden. Mit dem Python-Skript `train.py` kann das Training bzw. ein Experiment durchgeführt werden:

```
python ray/python/ray/rllib/train.py --env CartPole-v0 --run DQN
```

Listing 9.14: Cart Pole & DQN

Die Argumenten zu diesem Skript sind in Tab. 9.7 dargestellt.

Argument	**Erklärung**	**Beispiel**
env	(OpenAI Gym) Umgebung	CartPole-v0
run	Agent bzw. Algorithmus	DQN
stop	Abbruch-Kriterium	'{"time_total_s":600}'
config	Algorithmus-spezifische Konfiguration	'{'gamma":0.99}'
resources	Computer-Ressourcen	'{"cpu":64, "gpu":8}'
repeat	Anzahl der Wiederholungen je Experiment	1
local-dir	Verzeichnis zum Speichern der Ergebnisse	~/ray_results
upload-dir	Optionale URI zum Hochladen der Ergebn.	
checkpoint-freq	Anzahl Iterationen zwischen 2 Checkpoints	0: keine Checkpoints
scheduler	Art der Einplanung und Ausführung	FIFO
scheduler-config	Konfiguration für den Scheduler	
restore	Wiederaufnahme bei diesem Checkpoint	
redis-address	Redis-Adresse des Clusters	
num-cpus	Anzahl der CPUs	1
num-gpus	Anzahl der GPUs	0
experiment-name	Name des Ergebnisverzeichnis	default

Tabelle 9.7: Argumente zum Kommandozeilenskript der Softwarelösung Ray RLlib

Der Präfix `--` ist bei allen Argumenten aus Darstellungsgründen weggelassen worden.

Jedes Argument erwartet einen Wert. Einige Beispiele sind in der Tabelle bereits angegeben. Möchte man bspw. andere Agenten benutzen, muss man also den Wert `DQN` des Argements `--run` durch `A3C`, `BC`, `ES`, `PG` oder `PPO` ersetzen. Allerdings wird man bei den Agenten `BC` und `PG` feststellen, dass diese nicht als trainierbar registriert wurden und deshalb nicht verwendet werden können. Jeder Agent hat bereits eine Standard-Konfiguration. Diese findet man im Quelltext zum jeweiligen Agenten. Man kann diese Konfigurationseinstellungen aber auch überschreiben, indem man dem Pythonskript das entsprechende Argumente `--config` mit gibt. Standardmäßig benutzt Ray keine GPUs. Mit Hilfe des Arguments `--resources` kann man diese aber zuordnen, also z.B. `--resources '{"cpu":1, "gpu":1}'` verwenden, um auch Rechnungen auf der GPU auszuführen.

Interessant ist bspw. auch das Argument `--checkpoint-freq`. Damit kann eine Frequenz in Trainingsschritten angegeben werden, nach der jeweils der aktuelle Trainingsstand gespeichert wird. Diese Checkpoint-Datei kann dann dazu benutzt werden, um das Training ab dort fortzuführen oder um den bis dahin trainierten Agenten zu testen bzw. für eine Aufgabe zu verwenden. Wenn man nun noch einen Namen für das Experiment angibt, bspw. `--experiment-name test1`, dann wird ein neues Verzeichnis `~/ray_results/test1` erzeugt und in diesem werden dann alle Experimentbezogenen Dateien gespeichert. Das ist sehr nützlich, wenn man viele Experimente durchführen und nicht den Überblick verlieren möchte. Die Ergebnisse werden beim Training automatisch in die Datei `result.json` in dem Experimentverzeichnis gespeichert. Dort wird auch die Datei `params.json` erzeugt, welche die Trainingsparameter enthält.

Zwei Argumente sind nicht in Tab. 9.7 aufgelistet. Diese werden nur mit einem normalen Minuszeichen als Präfix aufgerufen. Mit der Option `-h` bekommt man eine generelle Hilfe zu diesem Pythonskript angezeigt. Die Option `-f` kann benutzt werden, um eine Konfigurationsdatei zu laden und zu verwenden. Durch die Angabe der Option `-f` werden alle anderen Einstellungen bedeutungslos, d.h. nur die Parameter in dieser Konfigurationsdatei werden benutzt.

Mit dem Werkzeug *TensorBoard*, eine Erweiterung von *TensorFlow*, können die Trainings- und Ergebnisdaten visualisiert werden:

```
tensorboard --logdir=~/ray_results
```

Listing 9.15: TensorBoard

Nun sollen Experimente mit Ray RLlib durchgeführt werden. Hierzu wird wieder die Umgebung OpenAI Gym und das Problem *Cart Pole* (vgl. Kap. 2.3 und Kap. 4.4) verwendet. Es werden die vier möglichen Agenten benutzt, meistens in der jeweiligen Standard-Konfiguration. Nur beim `DQNAgent` wird davon abgewichen. Es lassen sich auch Experimente mit der Kommandozeile durchführen (s.o.). Der Nachteil ist allerdings, dass es kein Skript namens `eval.py` gibt, um die Ergebnisse zu testen, wie es eigentlich in der Dokumentation beschrieben steht [Lia+18b]. Aus diesem Grund werden eigene Python-Skripte programmiert, um die Agenten damit zu trainieren und zu evaluieren.

Im folgenden Listing ist der prinzipielle Aufbau gezeigt.

```
# Allgemeine Module importieren (OpenAI Gym und Ray)
import gym
import ray
```

```
# Spezielle Module importieren (Agenten)
# TODO 1

# Ray initialisieren
ray.init(num_workers=0)

# Agent konfigurieren
config = DEFAULT_CONFIG.copy()
# TODO 2

# Agent für CartPole-Problem erstellen
agent = None # TODO 3

# Agent trainieren (n Schritte oder max. 10 min = 600 sec)
n = 100 # TODO 4
time_max = 600
best_reward = 0
for i in range(n):
    signal = ''
    result = agent.train()
    # Neues, bestes Trainingsergebnis ?
    if result.episode_reward_mean > best_reward:
        best_reward = result.episode_reward_mean
        signal = '*'
    # Aktuelles Trainingsergebnis ausgeben
    print("{0:3d}: after {1:7d} timesteps: mean reward = {2:6.2f} (time:
        {3:6.2f} sec) {4:1s}"
          .format((i+1), result.timesteps_total, result.
              episode_reward_mean, result.time_total_s, signal))
    # Trainingsabbruch ?
    if result.time_total_s > time_max:
        break

# Agent speichern
checkpoint_path = agent.save()

# Test vorbereiten
trained_config = config.copy()
test_agent = None # TODO 5
test_agent.restore(checkpoint_path)
env = gym.make('CartPole-v0')

# Agent testen (10 x)
n = 10
test_reward = 0
for i in range(n):
    state = env.reset()
    done = False
    cumulative_reward = 0
    while not done:
        action = test_agent.compute_action(state)
        state, reward, done, _ = env.step(action)
        cumulative_reward += reward
    # Testergebnis ausgeben
    print("Test-Reward {0:2d}: {1:6.2f}".format((i+1), cumulative_reward)
        )
    test_reward += cumulative_reward

# Gesamtergebnis ausgeben
print("Mittlerer Test-Reward: {0:6.2f}".format((test_reward/n)))
```

Listing 9.16: Experiment durchführen

Der Quelltext sollte weitgehend selbsterklärend sein. An fünf Stellen im Quelltext ist Begriff `TODO` verwendet worden. Dieser Begriff kennzeichnet Stellen, an denen der Quelltext noch unvollständig ist. Je nach verwendetem Agenten sieht die Programmierung hier etwas anders aus.

Nach spätestens 10 Minuten wird das Training abgebrochen (`time_max = 600`). Aber auch die Anzahl der Iterationen (`n`) ist ein wichtiges Kriterium. Je nach Agent kann hier ein anderer Wert sinnvoll sein, d.h. man muss hier experimentieren.

Am Ende des Programms findet dann der Test bzw. die Evaluation des Agenten statt. Hierbei muss der trainierte Agent 10 mal das RL-Problem *CartPole* lösen. Der maximal mögliche Reward beträgt dabei wieder 200.

In den nachfolgenden Listings sind nun genau die Programmzeilen für die offenen *Todos* angegeben.

```
# TODO 1
from ray.rllib.dqn import DQNAgent, DEFAULT_CONFIG
#TODO 2
config['dueling'] = False # oder True
config['double_q'] = False # oder True
config['hiddens'] = [64]
config['num_gpus_per_worker'] = 1
# TODO 3
agent = DQNAgent(config, 'CartPole-v0')
#TODO 4
n = 300
# TODO 5
test_agent = DQNAgent(trained_config, 'CartPole-v0')
```

Listing 9.17: Experiment mit DQN durchführen

Durch die Wahl der Konfigurationsparameter `double_q` und `dueling` lassen sich noch zwei weitere Agenten trainieren. Für alle anderen Agenten wird die Standard-Konfiguration benutzt, sodass bei `TODO 2` nichts eingefügt werden muss

```
# TODO 1
from ray.rllib.a3c import A3CAgent, DEFAULT_CONFIG
# TODO 3
agent = A3CAgent(config, 'CartPole-v0')
#TODO 4
n = 300
# TODO 5
test_agent = A3CAgent(trained_config, 'CartPole-v0')
```

Listing 9.18: Experiment mit A3C durchführen

```
# TODO 1
from ray.rllib.es import ESAgent, DEFAULT_CONFIG
# TODO 3
agent = ESAgent(config, 'CartPole-v0')
#TODO 4
n = 30
# TODO 5
test_agent = ESAgent(trained_config, 'CartPole-v0')
```

Listing 9.19: Experiment mit ES durchführen

```
# TODO 1
from ray.rllib.ppo import PPOAgent, DEFAULT_CONFIG
# TODO 3
agent = PPOAgent(config, 'CartPole-v0')
#TODO 4
n = 20
# TODO 5
test_agent = PPOAgent(trained_config, 'CartPole-v0')
```

Listing 9.20: Experiment mit PPO durchführen

Diese Programme lassen sich nun wieder mit dem Jupyter-Notebook ausführen (vgl. Abb. 9.9). Das hat auch den Vorteil, dass man sehr schnell und einfach Änderungen vornehmen kann.

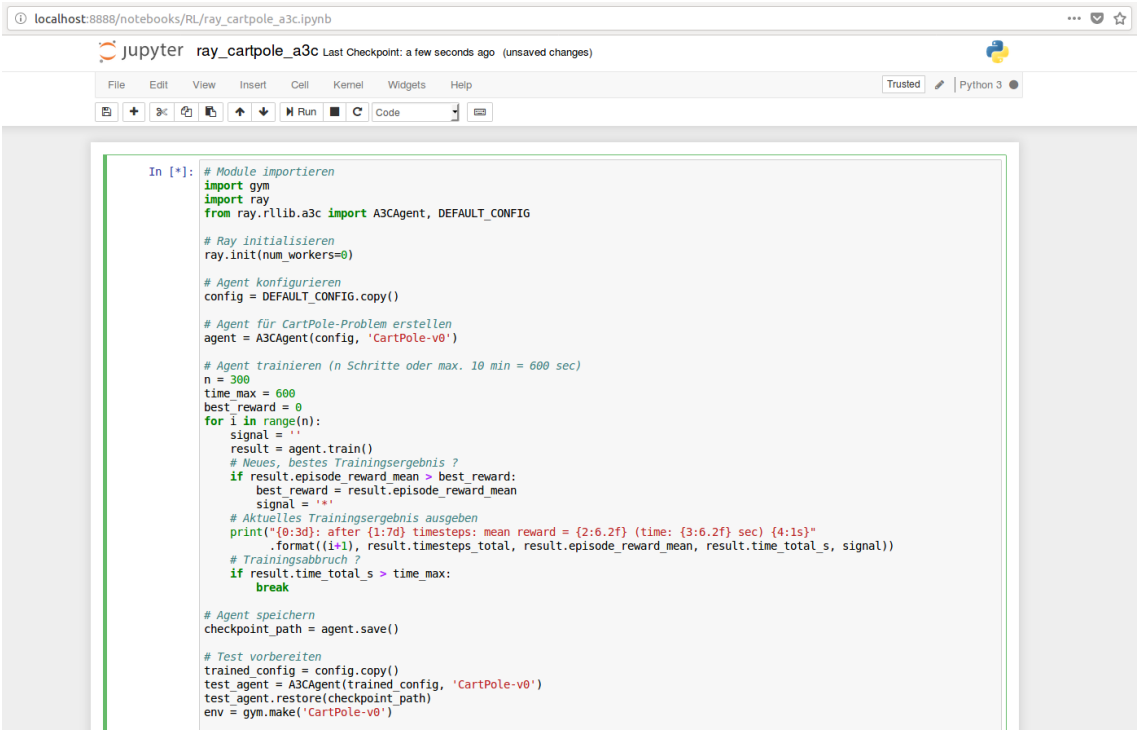


Abbildung 9.9: Ray-Python-Programm im Jupyter-Notebook

Mit Hilfe des TensorBoard-Befehls (s.o.) lässt sich der Lernfortschritt visualisieren und die Agenten miteinander vergleichen. In Abb. 9.10 bis 9.12 sind die Lernkurven zu den Experimenten dargestellt. Das obere Diagramm zeigt jeweils den mittleren Reward pro Episode in Abhängigkeit des Zeitschritts. Das untere Diagramm zeigt die Zeitdauer, die für einen Zeitschritt benötigt wurde.

In Abb. 9.10 sieht man die Ergebnisse zu A3C in blauer Farbe. Zum Vergleich ist auch die Kurve für ES in Rot angegeben. Abb. 9.11 zeigt dann die Ergebnisse zu ES. Dieser Algorithmus ist wesentlich schneller bzw. im Vergleich zu A3C werden sehr viel mehr Zeitschritte gemacht. Auch in dieser Grafik ist der Verlauf von A3C dargestellt. Nun sieht diese Kurve stark gestaucht aus. Die Ergebnisse zu DQN (hellblau), DDQN (rosa) und Dueling (grün) sind in Abb. 9.12 dargestellt.

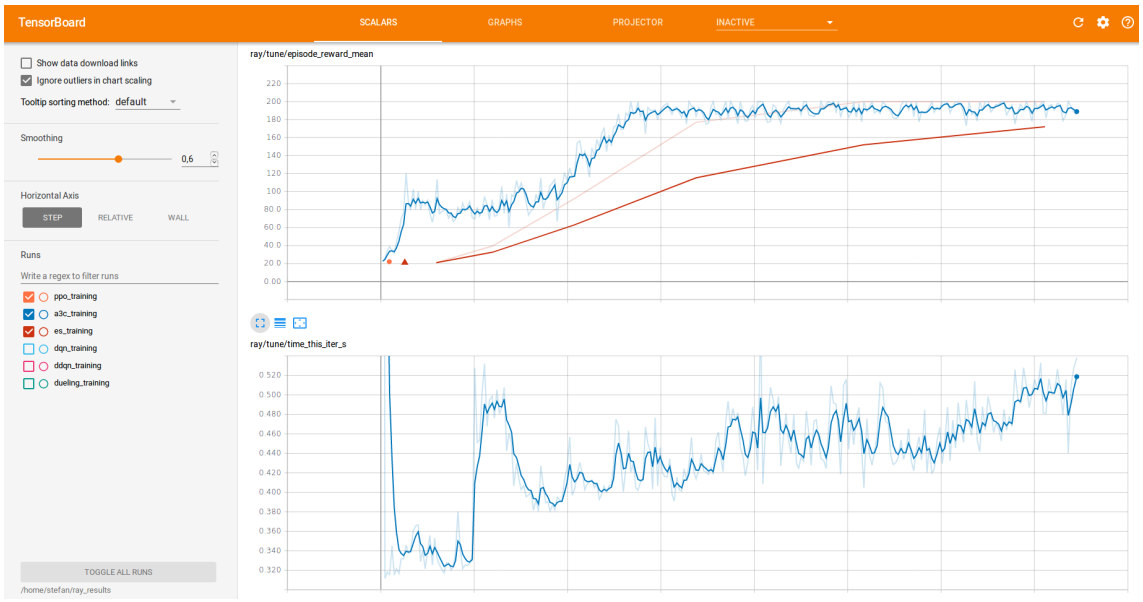


Abbildung 9.10: Ray-TensorBoard mit Ergebnissen zu A3C

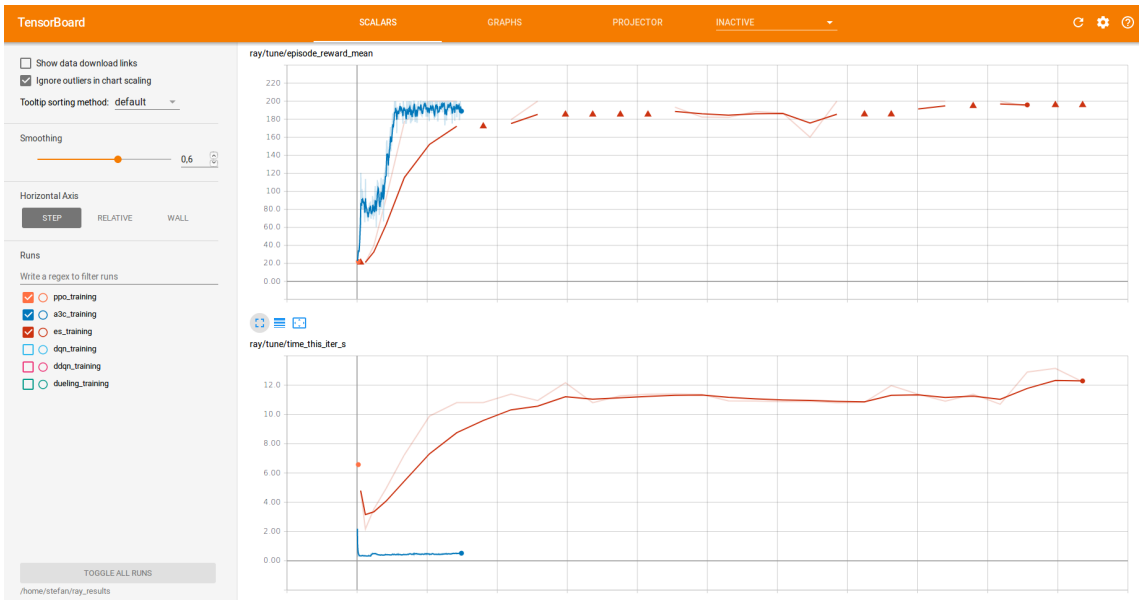


Abbildung 9.11: Ray-TensorBoard mit Ergebnissen zu ES

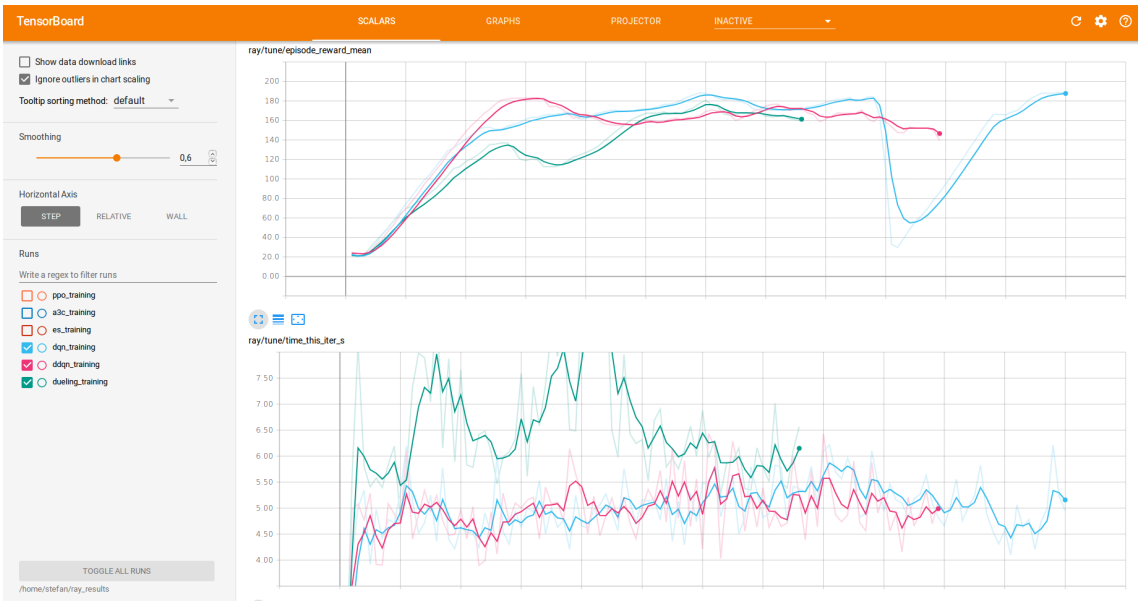


Abbildung 9.12: Ray-TensorBoard mit Ergebnissen zu DQN

Eine Besonderheit des TensorBoards ist die *Mouse-Over*-Funktion. Dadurch kann man mit der Maus in den Diagrammen navigieren und man bekommt in einem überlagerten Bereich Detailinformationen zu dem aktuellen Trainingszeitpunkt angezeigt (vgl. Abb. 9.13).

Abbildung 9.13: Ray-TensorBoard mit Mouse-Over-Funktionalität

Zu PPO konnten die Ergebnisse nicht mit dem TensorBoard angezeigt werden. Die genaue Ursache hierfür muss noch untersucht werden. Es fällt jedoch auf, dass dieser Algorithmus sehr schnell zu einer sehr guten Lösung kommt, wenn man die Ausgabe auf der Konsole betrachtet, die das Programm erzeugt:

```
 1: after    9094 timesteps: mean reward =  22.02 (time:   6.57 sec) *
 2: after   18510 timesteps: mean reward =  48.29 (time:  11.25 sec) *
 3: after   28081 timesteps: mean reward = 127.61 (time:  16.57 sec) *
 4: after   37894 timesteps: mean reward = 181.72 (time:  22.16 sec) *
 5: after   47681 timesteps: mean reward = 188.21 (time:  27.83 sec) *
 6: after   57300 timesteps: mean reward = 196.31 (time:  32.72 sec) *
 7: after   67530 timesteps: mean reward = 196.73 (time:  37.68 sec) *
 8: after   77343 timesteps: mean reward = 196.26 (time:  42.14 sec)
 9: after   86343 timesteps: mean reward = 200.00 (time:  47.55 sec) *
10: after   95797 timesteps: mean reward = 196.96 (time:  52.79 sec)
11: after  104797 timesteps: mean reward = 200.00 (time:  57.51 sec)
12: after  113797 timesteps: mean reward = 200.00 (time:  61.64 sec)
13: after  123333 timesteps: mean reward = 198.67 (time:  66.32 sec)
14: after  132776 timesteps: mean reward = 196.73 (time:  71.43 sec)
15: after  142006 timesteps: mean reward = 196.38 (time:  75.94 sec)
16: after  152009 timesteps: mean reward = 196.14 (time:  80.41 sec)
17: after  161368 timesteps: mean reward = 199.13 (time:  85.36 sec)
18: after  170550 timesteps: mean reward = 199.61 (time:  90.03 sec)
19: after  179550 timesteps: mean reward = 200.00 (time:  94.26 sec)
20: after  188550 timesteps: mean reward = 200.00 (time:  99.03 sec)
```

Bereits nach dem neunten von 20 Schritten wurde der maximale durchschnittliche Reward von 200 erreicht, nach nur knapp 48 Sekunden Training. Die Ergebnisse der sechs Experimente sind in Tabelle 9.8 zusammengefasst.

	A3C	**ES**	**PPO**	**DQN**	**DDQN**	**Dueling**
Lösung in Iteration	104	6	9	-	-	-
Zeitschritte	267746	516728	86343	-	-	-
Benötigte Zeit [s]	42,7	32,5	47,6	-	-	-
Gesamtiterationen	300	30	20	120	120	92
Zeitschritte	814807	5177153	188550	120000	120000	92000
Benötigte Zeit [s]	134,0	304,6	99,0	600,0	600,0	600,0
Mittlerer Reward	200,0	200,0	200	149,1	96,30	200,0
Speicher [MiB]	207	10351	10413	10381	10381	10381

Tabelle 9.8: Ergebnisse der Simulationen mit den Agenten / Algorithmen für Cart Pole

Zunächst fällt auf, dass keiner der DQN-Agenten das RL-Problem während des Trainings löst. Von den drei ersten Algorithmen löst ES das Problem bereits nach 6 Schritten und benötigt dafür nur 32,5 Sekunden. Aber auch A3C und PPO liefern sehr gute Resultate. Bei diesen Algorithmen wirkt die max. Zahl an Iterationen als Abbruchkriterium des Trainings. PPO ist mit 99 Sekunden am Schnellsten. Alle drei Algorithmen können auch im Test überzeugen und erhalten 10 Mal den höchsten Reward von 200 Punkten. Interessant ist, dass A3C der einzige Algorithmus ist, der weniger Speicher auf der Grafikkarte alloziert als die anderen Algorithmen (Auswertung des `nvidia-smi`-Befehls). Die drei Algorithmen zu dem DQN-Agenten müssten nach der schlechten Trainingsleistung eigentlich auch durch die Tests fallen. Umso erstaunlicher ist, dass die Dueling Architektur den Maximalwert von 200 Punkten erreicht. Hierzu müssen noch weitere Untersuchungen durchgeführt werden. Die drei Algorithmen des DQN-Agenten sind vergleichsweise langsam. Ggf. müssen andere Konfigurationen benutzt werden. Da die Dokumentation zur Ray RLlib nicht sehr umfangreich ist und Beispiele fehlen, muss der Quelltext direkt studiert werden. Dies ist aber nicht mehr Bestandteil dieser Arbeit.

9.4 TensorLayer

Die Bibliothek TensorLayer ist nicht speziell für *Reinforcement Learning* Probleme entwickelt worden. Im Gegensatz zu den anderen drei Softwarelösungen Keras-RL, OpenAI Lab und Ray RLlib gibt es bei TensorLayer keine Agenten, Algorithmen oder Experimente. Der Schwerpunkt liegt wie der Name bereits andeutet auf Schichten (engl. *Layers*), die zu Künstlichen Neuronalen Netzwerken (KNN) angeordnet werden können. Diese tiefen KNN lassen sich dann zum Zweck der Funktionsapproximation nutzen, um die Q-Werte eines Agenten in einem RL-Problem zu bestimmen.

Wenn man sich die Quelltexte zu der TensorLayer Bibliothek ansieht, stellt man fest, dass es 6 Tutorials im Verzeichnis Example gibt, die man zum Themenbereich *Reinforcement Learning* zuordnen kann (vgl. auch Kap. 5.12). Das Tutorial `tutorial_cartpole_ac.py` beschreibt, wie mit dem Actor Critic Algorithmus das Problem *Cart Pole* gelöst wird. Wenn man sich den Quelltext etwas genauer ansieht, stellt man fest, dass dieser ursprünglich von Morvan Zhou stammt, der unter dem Namen *Reinforcement Learning with TensorFlow* eine ganze Sammlung von Methoden und Tutorials auf der Plattform GitHub bereitgestellt hat (vgl. Kap. 7.6).

```
# Module importieren
import time
import gym
import numpy as np
import tensorflow as tf
import tensorlayer as tl
from tensorlayer.layers import *

# Zufallsgenerator-Initialisierung (Reproduzierbarkeit)
np.random.seed(123)
tf.set_random_seed(123)

# Parameterwerte setzen
OUTPUT_GRAPH = False            # keine Grafik-Ausgabe
MAX_EPISODE = 3000              # max. Anzahl von Episoden
DISPLAY_REWARD_THRESHOLD = 100  # Schwelle überschritten => Umgebung rendern
MAX_EP_STEPS = 1000             # max. Zeitschritte pro Episode
RENDER = False                  # Kein generelles Rendering (s.o.)
LAMBDA = 0.9                    # Reward Diskontierungsfaktor (TD-Fehler)
LR_A = 0.001                    # Lernrate für den Actor
LR_C = 0.01                     # Lernrate für den Critic

# Umgebung initialisieren
env = gym.make('CartPole-v0')
env.seed(123)
N_F = env.observation_space.shape[0]
N_A = env.action_space.n

# Klasse Actor
class Actor(object):

  def __init__(self, sess, n_features, n_actions, lr=0.001):
    self.sess = sess
    self.s = tf.placeholder(tf.float32, [1, n_features], "state")
    self.a = tf.placeholder(tf.int32, [None], "act")
    self.td_error = tf.placeholder(tf.float32, [None], "td_error")

    with tf.variable_scope('Actor'):
      n = InputLayer(self.s, name='in')
```

```
      n = DenseLayer(n, n_units=30, act=tf.nn.relu6, W_init=tf.
          random_uniform_initializer(0, 0.01), name='hidden')
      n = DenseLayer(n, n_units=n_actions, name='Pi')
      self.acts_logits = n.outputs
      self.acts_prob = tf.nn.softmax(self.acts_logits)

    with tf.variable_scope('loss'):
      self.exp_v = tl.rein.cross_entropy_reward_loss(logits=self.
          acts_logits, actions=self.a, rewards=self.td_error, name='
          actor_weighted_loss')

    with tf.variable_scope('train'):
      self.train_op = tf.train.AdamOptimizer(lr).minimize(self.exp_v)

  def learn(self, s, a, td):
    _, exp_v = self.sess.run([self.train_op, self.exp_v], {self.s: [s],
        self.a: [a], self.td_error: td[0]})
    return exp_v

  def choose_action(self, s):
    probs = self.sess.run(self.acts_prob, {self.s: [s]})
    return tl.rein.choice_action_by_probs(probs.ravel())

  def choose_action_greedy(self, s):
    probs = self.sess.run(self.acts_prob, {self.s: [s]})
    return np.argmax(probs.ravel())

# Klasse Critic
class Critic(object):

  def __init__(self, sess, n_features, lr=0.01):
    self.sess = sess
    self.s = tf.placeholder(tf.float32, [1, n_features], "state")
    self.v_ = tf.placeholder(tf.float32, [1, 1], "v_next")
    self.r = tf.placeholder(tf.float32, None, 'r')

    with tf.variable_scope('Critic'):
      n = InputLayer(self.s, name='in')
      n = DenseLayer(n, n_units=30, act=tf.nn.relu6, W_init=tf.
          random_uniform_initializer(0, 0.01), name='hidden')
      n = DenseLayer(n, n_units=1, act=tf.identity, name='V')
      self.v = n.outputs

    with tf.variable_scope('squared_TD_error'):
      self.td_error = self.r + LAMBDA * self.v_ - self.v
      self.loss = tf.square(self.td_error)

    with tf.variable_scope('train'):
      self.train_op = tf.train.AdamOptimizer(lr).minimize(self.loss)

  def learn(self, s, r, s_):
    v_ = self.sess.run(self.v, {self.s: [s_]})
    td_error, _ = self.sess.run([self.td_error, self.train_op], {self.s:
        [s], self.v_: v_, self.r: r})
    return td_error

# Actor und Critic instanziieren
sess = tf.Session()
actor = Actor(sess, n_features=N_F, n_actions=N_A, lr=LR_A)
critic = Critic(sess, n_features=N_F, lr=LR_C)
tl.layers.initialize_global_variables(sess)
```

```
# Logging
if OUTPUT_GRAPH:
  tf.summary.FileWriter("logs/", sess.graph)

# Training
for i_episode in range(MAX_EPISODE):

  episode_time = time.time()
  s = env.reset()
  t = 0          # Zähler für die Zeitschritte
  all_r = []     # Alle Reward in dieser Episode

  while True:

    if RENDER: env.render()                # Umgebung rendern
    a = actor.choose_action(s)             # Aktion a auswählen
    s_new, r, done, info = env.step(a)     # Aktion a ausführen
    if done: r = -20                       # Problem nicht gelöst
    all_r.append(r)                        # Reward speichern
    td_error = critic.learn(s, r, s_new)   # TD-Fehler berechnen
    actor.learn(s, a, td_error)            # Lern-Strategie anwenden
    s = s_new                              # Neuen Zustand s setzen
    t += 1                                 # Zeitschritt t erhöhen

    # Training abbrechen ?
    if done or t >= MAX_EP_STEPS:
      ep_rs_sum = sum(all_r)
      if 'running_reward' not in globals():
        running_reward = ep_rs_sum
      else:
        running_reward = running_reward * 0.95 + ep_rs_sum * 0.05
      print("Episode: %d reward: %f running_reward %f took: %.5f" % (
          i_episode, ep_rs_sum, running_reward, time.time() -
          episode_time))

      # Early Stopping => Agent evaluieren / testen
      if t >= MAX_EP_STEPS:
        print("Early Stopping")
        s = env.reset()
        rall = 0
        while True:
          env.render()
          a = actor.choose_action_greedy(s)
          s_new, r, done, info = env.step(a)
          s_new = np.concatenate((s_new[0:N_F], s[N_F:]), axis=0)
          rall += r
          s = s_new
          if done:
            print("reward", rall)
            s = env.reset()
            rall = 0
break
```

Listing 9.21: Cart Pole & AC – in Anlehnung an [DM+18] bzw. [Zho18]

Die Listings zu den anderen Beispielen sehen ähnlich aus. Die Komponenten Experiment, Agent bzw. Algorithmus und Umgebung sind also nicht wirklich entkoppelt, wenn man überhaupt in diesem Fall von Komponenten sprechen kann. Damit ist eine Wiederverwendendung und Erweiterbarkeit nicht gegeben und deshalb die Nutzung der TensorLayer Bibliothek massiv eingeschränkt. Aus diesem Grund wird auf das Ausführen dieses Beispiels verzichtet.

9.5 Fazit

In den vorangegangenen Abschnitten wurden die vier Softwarelösungen Keras-rl, OpenAI Lab, Ray RLlib und TensorLayer nun genauer analysiert und teilweise getestet, d.h. nur mit den ersten drei Lösungen wurden auch typische *Reinforcement Learning* Experimente durchgeführt. Nach Analyse der Quelltexte der TensorLayer Bibliothek war schnell klar, dass diese Lösung nicht eingesetzt werden sollte, weil die Komponenten Agent bzw. Algorithmus, Umgebung und Experiment nicht ausreichend voneinander entkoppelt sind. Somit lassen sich die Quelltexte schlecht wiederverwenden und weiter entwickeln.

Für die drei anderen Lösungen wurden, wie ursprünglich auch geplant, Experimente durchgeführt. Hierzu wurde die OpenAI Umgebung *Cart Pole* (vgl. Kap. 2.3 und Kap. 4.4) verwendet. Zwei Aktionen sind möglich, um den Wagen in die horizontalen Richtungen zu bewegen: +1 für rechts und -1 für links. Das Pendel ist am Anfang immer in aufrechter, vertikaler Position mit Winkel $\theta = 0$. Das Ziel ist es, dieses Pendel möglichst lange zu balancieren, ohne dass es fällt. Eine Episode ist beendet, falls der Winkel zu groß wird ($\theta > 15$) oder sich der Wagen um mehr als 2,4 Einheiten vom Ausgangspunkt entfernt. Zum Testen werden 200 diskrete Zeitschritte durchgeführt. In jedem Zeitschritt gibt es einen Reward von +1, wenn das Pendel noch ausbalanciert und keine der Abbruchbedingungen eingetreten ist. Somit ist ein maximaler Reward von 200 Punkten möglich. Während des Trainings wird der Reward ebenfalls kontinuierlich gemessen. OpenAI Gym definiert das Problem als gelöst, wenn der mittlere Reward größer als 195 in den letzten 100 Episoden war [Ope18].

Vier Agenten aus der Keras-RL-Bibliothek wurden auf dieses Problem angesetzt: SARSA, CEM, DQN und Dueling-DQN. Die Python-Skripte konnten im Jupyter-Notebook editiert und ausgeführt werden. Jeder Agent verfügt über die gleichen Funktionen wie bspw. `compile`, `fit` und `test`, sie unterscheiden sich dagegen in der Instanziierung, zu der der jeweilige Konstruktor benötigt wird. Die Agenten lassen sich also sehr leicht austauschen. Für 50.000 Episoden wurde das Training für die Standard-Konfiguration der Agenten durchgeführt und anschließend erfolgte der bereits beschriebene Test (s.o.). Die Berechnungen wurden auf der GPU des Rechners ausgeführt (Nvidia Geforce GTX 1080 Ti). Für das Training benötigten die Algorithmen zwischen 46 Sekunden und 3:47 Minuten. Die DQN-Agenten erzielten mit einem mittleren Reward von 200 ausgezeichnete Ergebnisse. Leider gibt es zu Keras-RL kaum Dokumentation und es fehlt eine grafische Visualisierung der Trainingsverläufe. Trotzdem ist es aber eine interessante Softwarelösung, mit der man sehr schnell auch ohne Dokumentation einen Einstieg in das Thema *Reinforcement Learning* findet.

Das Framework OpenAI Lab enthält ein Pythonskript `main.py`, welches zum Ausführen der Experimente verwendet wurde. Die Steuerung erfolgt mit Hilfe einer JSON-Datei, in der die Konfigurationsparameter beschrieben sind, bspw. die Umgebung, der Agent usw. Es wurden drei Agenten getestet: SARSA, AC und DQN. Für DDQN gab es leider einen Fehler bei der Initialisierung. Die Trainingsverläufe werden automatisch während das Trainig läuft laufend visualisiert. Die Agenten wurden in ihren Standard-Konfigurationen belassen. Zu jedem Experiment werden mehrere Dateien in einem eigenen Verzeichnis gespeichert: Die PNG-Datei enthält die grafische Visualisierung, in der JSON-Datei befinden sich die Konfigurationseinstellungen und die Ergebnisse, die CSV-Datei enthält nur das Endergebnis. Eigene Kennzahlen werden berechnet, u.a. ein Fitness Score, damit verschiedene Agenten (Algorithmen) in unterschiedlichen Umgebungen miteinander

verglichen werden können. OpenAI Lab führt die Berechnungen ebenfalls auf der GPU aus, allerdings wird hierzu sehr viel weniger Speicherplatz benötigt. Ein Test wie bei der Keras-RL-Bibliothek konnte nicht durchgeführt werden. Der Agent AC kam im Training auf einen mittleren Reward von 195,62 in den letzten 100 Episoden nach einer Trainingsdauer von 5:53 Minuten. Damit gilt das Problem als gelöst (s.o.).

Diese Bibliothek Ray RLlib bietet ebenfalls ein Pythonskript `train.py` zum Trainieren von vier möglichen Agenten an. Allerdings fehlt das korrespondierende Skript `eval.py`, mit dem sich anschließend die trainierten Agenden testen lassen. Aus diesem Grund wurden eigene Programme erstellt, die dann im Jupyter Notebook editiert und ausgeführt werden konnten. Analog zur Keras-RL-Bibliothek (s.o.) war der Aufbau dieser Skripte sehr ähnlich und die Agenten konnten leicht ausgetauscht werden. Es wurden die Agenten bzw. Algorithmen A3C, ES und PPO in der Standard-Konfiguration benutzt. Der Agent DQN konnte so konfiguriert werden, dass damit drei Algorithmen getestet werden konnten: DQN, DDQN und Dueling-DQN. Während des Trainings konnte der Verlauf mit Hilfe des Werkzeugs TensorBoard visualisiert werden. Für jedes Experiment wurde auch hier ein Verzeichnis angelegt und dort mehrere Dateien automatisch gespeichert: Die Datei `params.JSON` enthält die Konfiguration, während die Datei `results.JSON` die Ergebnisse zu jedem Trainingsschritt enthält. Außerdem sind ähnliche Daten auch in der Datei `progress.CSV` gespeichert. Je nach Konfiguration der sogenannten Checkpoint-Frequenz sind ggf. noch Zwischenergebnisse als Binärdateien gespeichert. Nach dem Training wurde wieder eine Evaluation durchgeführt. Die drei Agenten A3C, ES und PPO führen in ihren Standardkonfigurationen zu ausgezeichneten Ergebnissen (Reward von 200), wobei PPO nur 1:39 Minuten benötigte. Mit den DQN-Agenten konnten keine zufriedenstellenden Ergebnisse erzielt werden. Im Vergleich zu den anderen beiden Softwarelösungen, die ebenfalls DQN-Agenten implementiert haben, waren diese Algorithmen auch vergleichsweise langsam. Hier sind also weitere Untersuchungen nötig. Leider ist hierzu kaum Dokumentation vorhanden, sodass die Quelltexte direkt studiert werden müssen.

Nach diesen experimentellen Untersuchungen und den Ergebnissen der Nutzwertanalyse (vgl. Kap. 8.2) wird die Bibliothek Ray RLlib für weitere Analysen und zum Einsatz in Forschung und Lehre an der htw saar favorisiert. Mit Ray Tune verfügt das Framework Ray außerdem noch über einen Hyper-Optimierer. Dies ist ein interessantes Werkzeug, um eine optimale Konfiguration der Agenten automatisch zu finden. Dieses Werkzeug könnte man somit benutzen, um das schlechte Abschneiden der DQN-Agenten zu analysieren. Als Backup-Lösung kommen Keras-RL und OpenAI Lab in Betracht. Mit Keras-RL lässt sich sehr ähnlich wie mit Ray RLlib arbeiten, allerdings fehlt leider die grafische Visualisierung. OpenAI Lab liefert zwar interessante Kennzahlen, aber es ist unklar wie sich dieses Projekt weiterentwickeln wird. Die Quellcodes sind vergleichsweise alt.

10 Zusammenfassung und Ausblick

Das Thema *Reinforcement Learning (RL)* gehört wissenschaftlich betrachtet in die Kategorie Künstliche Intelligenz (KI). Die Digitalisierung und insbesondere die KI wird zukünftig unser privates, gesellschaftliches und berufliches Leben massiv umgestalten. Je mehr wir über dieses Thema und die damit verbundenen Chancen und Risiken lernen, desto besser können wir unsere Zukunft mitgestalten. Das Ziel dieser Arbeit war es, die theoretischen Grundlagen zu RL zunächst darzustellen und dann mindestens eine Softwarelösung zu diesem Thema zu finden, die zukünftig in der Lehre und Forschung an der Hochschule für Technik und Wirtschaft des Saarlandes (htw saar) eingesetzt werden kann.

Im Grundlagenkapitel wurden deshalb zunächst die Theorie zum Thema *Reinforcement Learning (RL)* vorgestellt. Das Besondere am RL sind die Interaktionen zwischen Agent und Umgebung, aus denen der Agent lernen kann, um ein vorgegebenes Ziel zu erreichen. Somit steht RL in erster Linie nicht für spezielle Algorithmen, sondern vielmehr für spezielle Probleme, die sich mit Hilfe von RL lösen lassen. Hierzu gehören klassische Probleme wie *Cart Pole* oder *Mountain Car*. Aber auch Computerspiele sind typische Lernumgebungen. An die Stelle des menschlichen Spielers tritt hier eben der Softwareagent als Maschine. Der Agent beobachtet die Umwelt und kann autonom Aktionen auswählen und ausführen, die seinen Zustand ändern und möglichst verbessern. Dabei bekommt er ein Feedback in Form eines *Reward*. Er versucht dann eine Strategie bzw. *Policy* zu entwickeln, um das vorgegebene Ziel zu erreichen, d.h. die Summe der *Rewards*, den *Return* zu maximieren. Mathematisch lassen sich viele RL-Probleme mit dem *Markov Decision Process (MDP)* beschreiben. Dabei ist die Bellmann-Gleichung Ausgangspunkt für die meisten Lernalgorithmen. Mittels Q-Learning oder SARSA versucht der Agent eine optimale Strategie zu finden, wobei die sogenannte Q-Wertefunktionen abgeschätzt werden, um die Zustände zu bewerten. Hierzu lassen sich auch Künstliche Neuronale Netzwerke (KNN) einsetzen, die ebenfalls zum Bereich KI zuzuordnen sind, und mit denen beliebige Funktionen approximiert werden können, also auch die Q-Wertefunktionen. In den letzten Jahren bekam die Forschung im Bereich der KNN einen Schub durch die Entwicklung des *Deep Learning (DL)*, mit dem sich auch tiefe KNN trainieren lassen. Werden beide Verfahren miteinander kombiniert, spricht man auch von *Deep Reinforcement Learning (DRL)*, typische Algorithmen sind Deep Q-Network (DQN) oder DeepSARSA.

Mittlerweile gibt es viele *Open Source* Softwarelösungen, die sich mit dem Thema *(Deep) Reinforcement Learning* beschäftigen. Auf der Plattform GitHub wurden am 02.02.2018 5.307 Projekte zu diesem Begriff gefunden. Mit über 60 Prozent waren die meisten dieser Lösungen in der Programmiersprache Python programmiert. Die Architektur einer modernen RL-Software ist komponentenbasiert aufgebaut. Die wesentlichen Komponenten sind Umgebung, Agent bzw. Algorithmus und Experiment. Weitere Komponenten können sein: Policy, Speicher, Optimierer usw. Einige der Softwarelösungen haben sich auf Komponenten spezialisiert, andere bieten das komplette Paket an. Dabei ist es wichtig, ob die Komponenten nur lose gekoppelt sind und sich daher leicht austauschen lassen. Somit wurde eine Klassifikation bzw. Kategorisierung vorgenommen und ausgewählte Lösungen präsentiert. Ein wichtiges Auswahlkriterium war die Popularität des Projekts, ausgedrückt durch die Anzahl von Sternen (*Stars*) durch Mitglieder der GitHub-Community.

In Kap. 4 wurden 10 Lösungen vorgestellt, die sich darauf spezialisiert haben, Umgebungen für Agenten anzubieten: ALE, CARLA, DeepMind Lab, OpenAI Gym, OpenAI Universe, Project Malmo, PLE, RLE, Roboschool und ViZDoom. Im Wesentlichen unterscheiden sich diese Lösungen im Umfang und Art der Problemstellungen. Dieses Spektrum reicht von den klassischen RL-Problemen (s.o.), über Brettspiele (z.B. Go) hin zu Computerspiele (Atari, Nintendo, Flash usw.) und sogar 3D-Egoshooter oder Echtzeit-Simulatoren zum autonomen Fahren. Diese Softwarelösungen besitzen definierte Schnittstellen, um vom Agenten Aktionen entgegenzunehmen und um die aktuellen Umweltzustände und Rewards an den Agenten zurückzuliefern.

Im Fokus des Kap. 5 standen dann Softwarelösungen, die selbst keine Umgebungen enthalten, sondern einige aus Kap. 4 verwenden, um ihre Softwareagenten darin trainieren zu lassen. Diese 13 Lösungen lassen sich in Bibliotheken oder Frameworks unterteilen, je nachdem, ob nur eine reine Sammlung von Algorithmen zur Lösung der RL-Probleme angeboten wird oder ob ein Rahmenwerk existiert, mit dem sich sehr einfach Experimente ausführen lassen. Es sind: Chainer RL, Coach, Deep RL, Keras-RL, OpenAI Baselines, OpenAI Lab, Ray RLlib, RL4J, RL Lab, TF Agents, TensorForce, TensorLayer und Torch-twrl.

Im darauffolgenden Kapitel wurden dann Lösungen präsentiert, die nicht in diese ersten Kategorien fallen. Diese Lösungen beinhalten also entweder beide Komponenten (Umgebungen und Agenten/Algorithmen) oder es sind Speziallösungen zu besonderen RL-Problemen. Außerdem wurden auch Lösungen betrachtet, die sich nicht auf der Plattform GitHub befinden. Insgesamt wurden 8 verschiedene Lösungen betrachtet: BURLAP, MAgent, PyBrain, RLPark, RLPy, MMLF, RL-Glue und RLToolkit.

Kapitel 7 stellte dann abschließend noch 6 Tutorials zum Thema *Reinforcement Learning* vor, die sich auf GitHub befinden. Diese Tutorials haben gemeinsam, dass sie Python-Skripte als Beispiele enthalten, in denen Umgebungen und Algorithmen verwendet werden, um typische RL-Probleme zu lösen. Im weiteren Sinne können diese Tutorials ebenfalls zu Softwarelösungen gezählt werden. Insgesamt wurden also 37 Softwarelösungen zum Thema *Reinforcement Learning* vorgestellt.

In Kap. 8 wurden dann einige dieser Lösungen analysiert, auch im Hinblick auf ein Testsystem, auf dem ausgewählte Lösungen installiert und getestet werden sollten. Auf diesem Testsystem sind das Linux-Betriebssystem Ubuntu 16.04 LTS, die Grafikkarten-Werkzeuge CUDA 9.1 und CuDNN 7.0.5 von Nvidia, die Distribution Anaconda 5.0.1 für Python in der Version 3.6 und die *Deep Learning* Bibliothek TensorFlow 1.5.0 installiert. Vor diesem Hintergrund und der Anforderung eine möglichst modulare Lösung zu verwenden, wurden die 13 Bibliotheken und Frameworks genauer analysiert. OpenAI Gym war als Umgebung praktisch gesetzt, weil alle 13 Lösungen aus Kap. 5 eine Schnittstelle hierzu anbieten. Von den 13 Lösungen bieten 11 ein *Application Programming Interface (API)* für Python an. Zu diesen wurde eine Nutzwertanalyse mit 5 Kategorien (Funktionalität, Popularität, Aktualität, Reputation und Dokumentation), die jeweils mit 20 Prozent gewichtet wurden, durchgeführt. Dabei konnten die Lösungen TensorLayer (3,945 Punkte), TensorForce (3,382 Punkte), Coach (3,090 Punkte) und Ray RLlib (3,086) am meisten überzeugen. Leider konnten TensorForce und Coach nicht erfolgreich auf dem Testsystem installiert werden, TensorLayer und Ray RLlib dagegen schon. Aus diesem Grund wurden auch noch die übrigen Lösungen ausprobiert und es stellte sich heraus, dass nur Keras-RL, OpenAI Lab und TensorFlow Agents installiert und ausgeführt wer-

den konnten. Da TensorFlow Agent bislang nur einen Algorithmus implementiert hat, wird diese Lösung nicht weiter betrachtet. Die anderen vier wurden dann im nächsten Kapitel weiter analysiert.

In Kap. 9 wurden die Dokumentationen und Quelltexte von den verbliebenen vier Softwarelösungen Keras-rl, OpenAI Lab, Ray RLlib und TensorLayer untersucht und es wurden typische Experimente durchgeführt. Hierzu kam die Umgebung OpenAI Gym mit dem Problem *Cart Pole* zum Einsatz. Es stellte sich heraus, dass bei TensorLayer die Komponenten Agent bzw. Algorithmus, Umgebung und Experiment nicht ausreichend voneinander entkoppelt sind. Somit lassen sich die Quelltexte schlecht wiederverwenden und weiter entwickeln. Am meisten konnte die Bibliothek Ray RLlib des Frameworks Ray der kalifornischen Universität Berkeley überzeugen. Die Dokumentation und Quelltexte sind gut. Ein vorbereitetes Pythonskript kann in der Kommandozeile schnell benutzt werden, um Experimente durchzuführen. Mit Hilfe selbst-programmierter Pythonskripte wurden mehrere Experimente im Jupyter Notebook durchgeführt, wobei die Agenten sich leicht austauschen lassen. Der Trainingsfortschritt wird dabei mit dem Werkzeug TensorBoard visualisiert. Außerdem lassen sich die Zwischen- und das Endergebnis zur weiteren Analyse bequem speichern. Insgesamt wurden 6 Experimente zu den folgenden Agenten bzw. Algorithmen durchgeführt: A3C, ES, PPO, DQN, DDQN und Dueling-DQN. Hier konnte PPO in nur 1:39 Minuten ein perfektes Ergebnis erzielen und das Problem lösen. Die beiden anderen Softwarelösungen Keras-RL und OpenAI Lab sind ebenfalls interessant, bieten aber nicht so viele Möglichkeiten wie Ray.

Als Ergebnis kann somit festgehalten werden, dass die Softwarelösung Ray RLlib für die Behandlung von *Reinforcement Learning* Probleme in Forschung und Lehre der Fakultät für Wirtschaftswissenschaften der Hochschule für Technik und Wirtschaft des Saarlandes (htw saar) zukünftig eingesetzt werden sollte. Allerdings muss hierzu die Lösung noch weiter analysiert und getestet werden. Laut Dokumentation soll es ein Pythonskript `eval.py` geben, mit dem sich trainierte Agenten evaluieren lassen. Dieses Skript ist aber in den Quelltexten nicht enthalten. Des Weiteren konnten mit den DQN-Agenten bislang keine zufriedenstellenden Ergebnisse erzielt werden und die Algorithmen zu diesen Agenten waren vergleichsweise langsam. Ggf. wurden ungünstige Konfigurationseinstellungen gewählt. Hierzu sind also weitere Untersuchungen nötig. Leider ist nur wenig Dokumentation zu diesen speziellem RL-Thema vorhanden, sodass die Quelltexte direkt studiert werden müssen. Mit Ray Tune verfügt das Framework Ray außerdem noch über einen Hyper-Optimierer. Dies ist ein interessantes Werkzeug, um eine optimale Konfiguration der Agenten automatisch zu finden. Dieses Werkzeug könnte man somit ebenfalls benutzen, um das schlechte Abschneiden der DQN-Agenten zu analysieren. Außerdem sollten weitere RL-Probleme studiert werden.

Andererseits konnten einige Lösungen wie bspw. Coach gar nicht getestet werden, weil während der Installation Fehler aufgetreten sind. Gerade Coach ist aber sehr interessant, weil diese Lösung die meisten Algorithmen implementiert hat und ein sehr modernes, Komponentenbasiertes Framework darstellt. Des Weiteren wurden die speziellen und sonstigen Lösungen sowie Tutorials im Rahmen dieser Arbeit nicht detaillierter analysiert. Generell ist eine große Dynamik zum Thema *Reinforcement Learning* zu beobachten, insbesondere weil durch die Erfolge, mit *Deep Learning* tiefe KNN zu trainieren, neue Möglichkeiten geschaffen wurden, beide Techniken zu kombinieren und auf komplexe RL-Probleme anzuwenden. Deshalb sollte der Softwaremarkt im Bereich *Deep Reinforcement Learning* weiterhin und fortlaufend beobachtet werden.

In der Lehre kann das Thema *Reinforcement Learning* ggf. in den folgenden Modulen der Fakultät für Wirtschaftswissenschaften der Hochschule für Technik und Wirtschaft platziert werden:

- Angewandte Methoden der Informationsbeschaffung (MMF-130)
- Angewandte Informatik (DFMMS-222)
- Big Data Analysis (MAMS-120)
- Data Science (MASCM-141)
- Projekt (BBWL-622)

Außerdem können auch Abschlussarbeiten mit Bezug zum Thema *Reinforcement Learning* durch Studierende bearbeitet werden.

In der angewandten Forschung können die Konzepte und Methoden des *Reinforcement Learning* ebenfalls eingesetzt werden. Zwei Ideen werden im Folgenden kurz skizziert.

BWL und VWL: Spieltheorie Spieltheoretische Konzepte werden u.a. benutzt, um strategische Entscheidungen zu treffen. In der Spieltheorie werden verschiedene Spielformen (kooperativ vs. nicht-kooperativ) sowie Gleichgewichts- und Stabilitätskonzepte (z.B. Nash-Gleichgewicht) unterschieden und diese mit quantitativen Methoden analysiert. Ein Spieler kann dabei auch als lernender Softwareagent modelliert werden, der mit Hilfe von *Reinforcement Learning* trainiert wird. Insbesondere Verhandlungsspiele könnten hiermit analysiert werden:

- Preisverhandlungen zwischen Käufer und Verkäufer
- Tariflohnverhandlungen zwischen Gewerkschaften und Arbeitgeberverbänden
- Kapital/Kredit-Verhandlungen zwischen Unternehmen und Banken
- Verhandlungen zwischen Ländern zum Abbau von Handelsschranken

In der Fakultät für Wirtschaftswissenschaften der htw saar beschäftigt sich der Kollege Prof. Dr. Markus Thomas Münter u.a. mit dem Thema *Behavioral Economics*. In Zusammenarbeit mit der Firma EUROKEY Software GmbH hat er ein virtuelles Labor zum Experimentieren ökonomischer Fragestellungen namens VIRTECOLAB entwickelt [Mü+18]. Aus dem Bereich der Spieltheorie ist das Tankstellenspiel implementiert, bei dem zwei Tankstellenbetreiber ihre Preise in mehreren Runden anpassen müssen, um im Wettbewerb bestehen zu können. Per Smartphone-App lässt sich dieses Spiel spielen, wobei bspw. Studierende in die Rollen der Tankstellenbetreiber schlüpfen. Durch dieses Spiel sollen Spieltheoretische Konzepte spielerisch vermittelt werden. Statt eines menschlichen Spielers könnte man auch einen Softwareagenten einsetzen, der mittels *Reinforcement Learning* auf dieses Tankstellenspiel trainiert wurde. In beiden Szenarien könnten die Aktionen der Spieler analysiert und miteinander verglichen werden. Einerseits lernen die Studierenden dann noch etwas zum Thema Künstliche Intelligenz, andererseits könnte analysiert werden, ob sich deren Verhalten ändert, wenn der andere Spieler eine Maschine und kein Mensch ist.

Marketing: Multi-Channel CRM Im Kundenbeziehungsmanagement (engl. *Customer Relationship Management (CRM)*) muss die wiederholt die Entscheidung getroffen werden, welcher Kunde über welchem Kanal (engl. *Channel*) angesprochen werden soll, bzw. per

Post, E-Mail, Telefon usw., wenn eine bestimmte Marketing-Kampagne geplant und durchgeführt wird. Einerseits möchte man keinen Kunden durch diese Maßnahme verärgern und ggf. verlieren, andererseits möchte man den kommerziellen Erfolg der Kampagne optimieren. Langfristig ist man am sogenannten *Customer Lifetime Value (CLV)* interessiert. Man könnte dieses Problem auch im Rahmen des *Reinforcement Learning* beschreiben. Das langfristige Ziel (Return) ist ein maximaler CLV. Nach jeder Kampagne (Episode) kann der Umsatz (Reward) gemessen werden. Somit könnte man Agenten bzw. RL-Algorithmen auf dieses Problem ansetzen, vorausgesetzt die zum Training und zu Analyse benötigten Daten stehen zur Verfügung. In der Fakultät für Wirtschaftswissenschaften der htw saar beschäftigt sich der Kollege Prof. Dr. Frank Hälsig fachlich mit dem Thema CRM. In mehreren anderen Forschungsprojekten wurde bereits erfolgreich zusammengearbeitet. Zu dieser Idee könnte also zukünftig ebenfalls ein kooperatives Forschungsprojekt durchgeführt werden.

Quellenverzeichnis

[BB+18] D. Britz, A. Bailo u. a. *Python implementation of Reinforcement Learning: An Introduction*. 2018. URL: https://github.com/dennybritz/reinforcement-learning (besucht am 02. 02. 2018).

[BHH+18] D. Bignell, T. Hutton, K. Hofmann u. a. *Project Malmo*. 2018. URL: https://github.com/Microsoft/malmo (besucht am 02. 02. 2018).

[BK18] C. Beattie und T. Köppe. *A customisable 3D platform for agent-based AI research*. 2018. URL: https://github.com/deepmind/lab (besucht am 02. 02. 2018).

[BM17] T. Bartscher und G. W. Maier. *Lernen*. 2017. URL: http://wirtschaftslexikon.gabler.de/Definition/lernen.html (besucht am 15. 09. 2017).

[BMH+18] M. G. Bellemare, M. C. Machado, M. Hausknecht u. a. *Arcade Learning Environment*. 2018. URL: https://github.com/mgbellemare/Arcade-Learning-Environment (besucht am 02. 02. 2018).

[BRC+18] N. Bhonker, S. Rozenberg, H. Cerezo u. a. *Retro Learning Environment*. 2018. URL: https://github.com/nadavbh12/Retro-Learning-Environment (besucht am 02. 02. 2018).

[BRH16] N. Bhonker, S. Rozenberg und I. Hubara. „Playing SNES in the Retro Learning Environment“. In: *CoRR* abs/1611.02205 (2016). URL: http://arxiv.org/abs/1611.02205.

[Bea+16] C. Beattie, Leibo J. Z., Teplyashin D., Ward T., Wainwright M., Küttler H., Lefrancq A., Green S., Valdés V., Sadik A., Schrittwieser J., Anderson K., York S., Cant M., Cain A., Bolton A., Gaffney S., King H., Hassabis D., Legg S. und Petersen S. „DeepMind Lab“. In: *CoRR* abs/1612.03801 (2016). URL: http://arxiv.org/abs/1612.03801.

[Bel+13] M. G. Bellemare, Y. Naddaf, J. Veness und M. Bowling. „The Arcade Learning Environment: An Evaluation Platform for General Agents“. In: *Journal of Artificial Intelligence Research* 47 (2013), S. 253–279. URL: https://arxiv.org/abs/1207.4708.

[Ben+17] Y. Bengio, F. Bastien, P. Lamblin, I. Goodfellow, R. Pascanu u. a. *Theano*. 21. Nov. 2017. URL: http://deeplearning.net/software/theano (besucht am 02. 02. 2018).

[Bri16] D. Britz. *Learning Reinforcement Learning (with Code, Exercises and Solutions)*. 2. Okt. 2016. URL: http://www.wildml.com/2016/10/learning-reinforcement-learning (besucht am 02. 02. 2018).

[Bro+16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang und W. Zaremba. „OpenAI Gym“. In: *CoRR* abs/1606.01540 (2016). URL: http://arxiv.org/abs/1606.01540.

[Bro+18a] G. Brockman, J. Tang, J. Schneider, T. Blackwell, J. Schulman, T. B. Brown, R. Cosman, P. Paquette, I. Shcherbatyi u. a. *OpenAI Gym*. 2018. URL: https://github.com/openai/gym (besucht am 02. 02. 2018).

[Bro+18b] T. B. Brown, J. Schlatter, T. Blackwell, G. Brockman u. a. *OpenAI Universe*. 2018. URL: https://github.com/openai/universe (besucht am 02. 02. 2018).

[CB+18] E. Coumans, Y. Bai u. a. *Bullet Physics SDK: Real-time collision detection and multi-physics simulation for VR, games, visual effects, robotics, machine learning etc.* 2018. URL: https://github.com/bulletphysics/bullet3 (besucht am 02. 02. 2018).

[CLN+18a] I. Caspi, G. Leibovich, G. Novik u. a. *Reinforcement Learning Coach Documentation*. 2018. URL: http://coach.nervanasys.com (besucht am 02. 02. 2018).

[CLN+18b] I. Caspi, G. Leibovich, G. Novik u. a. *Reinforcement Learning Coach by Intel AI Lab*. 2018. URL: https://github.com/NervanaSystems/coach (besucht am 02. 02. 2018).

[Cho+18] F. Chollet u. a. *Keras: The Python Deep Learning library*. 2018. URL: https://keras.io (besucht am 02. 02. 2018).

[Col+18] R. Collobert, C. Farabet, K. Kavukcuoglu, S. Chintala u. a. *Torch: A scientific computing framework for LuaJIT*. 2018. URL: http://torch.ch (besucht am 02. 02. 2018).

[DM+18] H. Dong, L. Mai u. a. *TensorLayer: A Versatile Deep Learning and Reinforcement Learning Library for TensorFlow*. 2018. URL: https://github.com/tensorlayer/tensorlayer (besucht am 02. 02. 2018).

[DP+18] Y. Duan, V. Pong u. a. *RL Lab: A framework for developing and evaluating reinforcement learning algorithms, fully compatible with OpenAI Gym*. 2018. URL: https://github.com/rll/rllab (besucht am 02. 02. 2018).

[Dan+18] C. Dann, B. Klein, E. Williams, W. Dabney u. a. *RLPy: Reinforcement Learning Framework*. 2018. URL: https://github.com/rlpy/rlpy (besucht am 02. 02. 2018).

[Dee16a] DeepMind. *Match 1 - Google DeepMind Challenge Match: Lee Sedol vs AlphaGo*. 9. März 2016. URL: https://www.youtube.com/watch?v=vFr3K2DORc8 (besucht am 28. 02. 2018).

[Dee16b] DeepMind. *Match 1 15 min Summary - Google DeepMind Challenge Match*. 11. März 2016. URL: https://www.youtube.com/watch?v=bIQxOsRAXCo (besucht am 28. 02. 2018).

[Dee16c] DeepMind. *Match 2 - Google DeepMind Challenge Match: Lee Sedol vs AlphaGo*. 10. März 2016. URL: https://www.youtube.com/watch?v=l-GsfyVCBu0 (besucht am 28. 02. 2018).

[Dee16d] DeepMind. *Match 2 15 min Summary - Google DeepMind Challenge Match*. 12. März 2016. URL: https://www.youtube.com/watch?v=1aMt7ulL6EI (besucht am 28. 02. 2018).

[Dee16e] DeepMind. *Match 3 - Google DeepMind Challenge Match: Lee Sedol vs AlphaGo*. 12. März 2016. URL: https://www.youtube.com/watch?v=qUAmTYHEyM8 (besucht am 28. 02. 2018).

[Dee16f] DeepMind. *Match 3 15 min Summary - Google DeepMind Challenge Match*. 12. März 2016. URL: https://www.youtube.com/watch?v=6hROM_bxZ9E (besucht am 28. 02. 2018).

[Dee16g] DeepMind. *Match 4 - Google DeepMind Challenge Match: Lee Sedol vs AlphaGo*. 13. März 2016. URL: https://www.youtube.com/watch?v=yCALyQRN3hw (besucht am 28. 02. 2018).

[Dee16h] DeepMind. *Match 4 15 Minute Summary - Google DeepMind Challenge Match 2016*. 13. März 2016. URL: https://www.youtube.com/watch?v=G5gJ-pVo1gs (besucht am 28. 02. 2018).

[Dee16i] DeepMind. *Match 5 - Google DeepMind Challenge Match: Lee Sedol vs AlphaGo*. 15. März 2016. URL: https://www.youtube.com/watch?v=mzpW10DPHeQ (besucht am 28. 02. 2018).

[Dee16j] DeepMind. *Match 5 15 min Summary - Google DeepMind Challenge Match*. 18. März 2016. URL: https://www.youtube.com/watch?v=QxHdPdRcMhw (besucht am 28. 02. 2018).

[Deg+18a] T. Degris u. a. *RLPark: Reinforcement Learning and Robotics in Java*. 2018. URL: https://github.com/rlpark/rlpark (besucht am 02. 02. 2018).

[Deg+18b] T. Degris u. a. *RLPark*. 2018. URL: http://rlpark.github.io (besucht am 02. 02. 2018).

[Deg18] T. Degris. *Zephyr*. 2018. URL: https://github.com/zephyrplugins/zephyr (besucht am 02. 02. 2018).

[Dha+18] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, Y. H. Liao u. a. *OpenAI Baselines: high-quality implementations of reinforcement learning algorithms*. 2018. URL: https://github.com/openai/baselines (besucht am 02. 02. 2018).

[Don+17] H. Dong, A. Supratak, L. Mai, F. Liu, A. Oehmichen, S. Yu und Y. Guo. „TensorLayer: A Versatile Library for Efficient Deep Learning Development“. In: *ACM Multimedia* (2017). URL: http://tensorlayer.org.

[Dos+17] A. Dosovitskiy, D. Ros, F. Codevilla, A. Lopez und V. Koltun. „CARLA: An Open Urban Driving Simulator“. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, S. 1–16. URL: https://arxiv.org/abs/1711.03938.

[Dua+16] Y. Duan, X. Chen, R. Houthooft, J. Schulman und P. Abbeel. „Benchmarking Deep Reinforcement Learning for Continuous Control“. In: *Proceedings of the 33rd International Conference on Machine Learning (ICML) Vol. 48*. New York (NY), USA, 2016, S. 1329–1338. URL: https://arxiv.org/abs/1604.06778.

[Dud17] Duden. *Lernen*. 2017. URL: http://www.duden.de/rechtschreibung/lernen (besucht am 15. 09. 2017).

[Ema18] P. Emami. *Deep-RL*. 2018. URL: https://github.com/pemami4911/deep-rl (besucht am 02. 02. 2018).

[FAG+18] R. Fiszel, S. Audet, F Garillot u. a. *RL4J: Deep Reinforcement Learning for the JVM*. 2018. URL: https://github.com/deeplearning4j/rl4j (besucht am 02. 02. 2018).

[FK+18] Y. Fujita, T. Kataoka u. a. *ChainerRL: A deep reinforcement learning library built on top of Chainer*. 2018. URL: https://github.com/chainer/chainerrl (besucht am 02. 02. 2018).

[Fou18] The Apache Software Foundation. *Apache Maven Project*. 2018. URL: https://maven.apache.org (besucht am 02. 02. 2018).

[GBC16] I. Goodfellow, Y. Bengio und A. Courville. *Deep Learning*. 1. Aufl. Cambridge, Massachusettts, USA: The MIT Press, 2016.

[GDH16] M. Ganger, E. Duryea und W. Hu. „Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning“. In: *Journal of Data Analysis and Information Processing* 4 (2016), S. 159–176. URL: https://file.scirp.org/pdf/JDAIP_2016101714072270.pdf.

[GH03] A. Greenwald und K. Hall. „Correlated-Q learning“. In: *Proceedings of the 20th International Conference on Machine Learning*. Washington D.C., USA, 2003, S. 242–249. URL: https://www2.cs.duke.edu/courses/spring07/cps296.3/correlated_q.pdf.

[GNP+18] A. Gibson, C. Nicholson, J. Patterson u. a. *DL4J: Deep Learning for Java*. 2018. URL: https://deeplearning4j.org (besucht am 02. 02. 2018).

[Ger+13] A. Geramifard, R. H. Klein, C. Dann, W. Dabney und J. P. How. *RLPy: The Reinforcement Learning Library for Education and Research*. 1. Apr. 2013. URL: http://acl.mit.edu/RLPy (besucht am 02. 02. 2018).

[Ger+15] A. Geramifard, R. H. Klein, C. Dann, W. Dabney und J. P. How. „RLPy: A Value-Function-Based Reinforcement Learning Framework for Education and Research“. In: *Journal of Machine Learning Research* 16 (2015), S. 1573–1578. URL: http://www.jmlr.org/papers/v16/geramifard15a.html.

[Gig16] Gigazine. *DeepMind Lab*. 6. Dez. 2016. URL: https://gigazine.net/news/20161206-deepmind-lab (besucht am 02. 02. 2018).

[Git18] GitHub. *GitHub*. 2018. URL: https://github.com (besucht am 02. 02. 2018).

[Gu+16] S. Gu, T. P. Lillicrap, I. Sutskever und S. Levine. „Continuous Deep Q-Learning with Model-based Acceleration“. In: *CoRR* abs/1603.00748 (2016). URL: http://arxiv.org/abs/1603.00748.

[HDV17] D. Hafner, J. Davidson und V. Vanhoucke. „TensorFlow Agents: Efficient Batched Reinforcement Learning in TensorFlow“. In: *ArXiv E-Prints* (2017). URL: https://arxiv.org/pdf/1709.02878.pdf.

[Haf+18] D. Hafner u. a. *TensorFlow Agents: Efficient Batched Reinforcement Learning in TensorFlow*. 2018. URL: https://github.com/tensorflow/agents (besucht am 02. 02. 2018).

[Hen+18] P. Henderson u. a. *Gym-Extensions: An extension for OpenAI Gym for auxiliary tasks*. 2018. URL: https://github.com/Breakend/gym-extensions (besucht am 02. 02. 2018).

[KCC10] P. Kormushev, S. Calinon und D. G. Caldwell. *Robot Learns to Flip Pancakes*. 26. Juli 2010. URL: https://www.youtube.com/watch?v=W_gxLKSsSIE (besucht am 28. 02. 2018).

[KG18a] W. L. Keng und L. Graesser. *OpenAI Lab: An experimentation framework for Reinforcement Learning using OpenAI Gym, Tensorflow, and Keras*. 2018. URL: https://github.com/kengz/openai_lab (besucht am 02. 02. 2018).

[KG18b] W. L. Keng und L. Graesser. *OpenAI Lab*. 2018. URL: http://kengz.me/openai_lab (besucht am 02. 02. 2018).

[KK10] A. T. Kalai und E. Kalai. „Cooperation and competition in strategic games with private information“. In: *Proceedings of the 11th ACM conference on Electronic commerce*. New York (NY) USA, 2010, S. 345–346.

[KS17] O. Klimov und J. Schulman. *Roboschool*. 15. Mai 2017. URL: https://blog.openai.com/roboschool (besucht am 02. 02. 2018).

[KSG17] N. S. Kuno und S. Schwiderski-Grosche. *Presenting the winners of the Project Malmo Collaborative AI Challenge*. 5. Juni 2017. URL: https://www.microsoft.com/en-us/research/blog/malmo-collaborative-ai-challenge-winners (besucht am 02. 02. 2018).

[KW+18] M. Kempka, M. Wydmuch u. a. *ViZDoom*. 2018. URL: https://github.com/mwydmuch/ViZDoom (besucht am 02. 02. 2018).

[Kem+16] M. Kempka, M. Wydmuch, G. Runc, J. Toczek und W. Jaśkowski. „ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning". In: *IEEE Conference on Computational Intelligence and Games*. Santorini, Griechenland: IEEE, 2016, S. 341–348. URL: http://arxiv.org/abs/1605.02097.

[Kli+18] O. Klimov u. a. *Roboschool: Open-source software for robot simulation, integrated with OpenAI Gym*. 2018. URL: https://github.com/openai/roboschool (besucht am 02. 02. 2018).

[Kun+15] N. S. Kuno u. a. *Project Malmo*. 1. Juni 2015. URL: https://www.microsoft.com/en-us/research/project/project-malmo (besucht am 02. 02. 2018).

[Lab15] Aerospace Controls Lab. *Autonomous Drifting using Machine Learning*. 11. Aug. 2015. URL: https://www.youtube.com/watch?v=opsmd5yuBF0 (besucht am 28. 02. 2018).

[Lab18] VUB Artificial Intelligence Lab. *Multi-agent Learning Seminar*. 2018. URL: https://ai.vub.ac.be/courses/2015-2016/multi-agent-learning-seminar (besucht am 02. 02. 2018).

[Lee+18] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman und C. K. Liu. *Dynamic Animation and Robotics Toolkit*. 2018. URL: https://github.com/dartsim/dart (besucht am 02. 02. 2018).

[Lia+18a] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg und I. Stoica. „Ray RLlib: A Composable and Scalable Reinforcement Learning Library". In: *CoRR* abs/1712.09381 (2018). URL: https://arxiv.org/abs/1712.09381.

[Lia+18b] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg und I. Stoica. *Ray RLlib: A Scalable Reinforcement Learning Library*. 2018. URL: http://ray.readthedocs.io/en/latest/rllib.html (besucht am 02. 02. 2018).

[Lib18] LibRetro. *LibRetro API*. 2018. URL: https://www.libretro.com/index.php/api/ (besucht am 02. 02. 2018).

[Lil+15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver und D. Wierstra. „Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015). URL: http://arxiv.org/abs/1509.02971.

[Lit01] M. L. Littman. „Friend-or-Foe Q-learning in General-Sum Games". In: *Proceedings of the 18th International Conference on Machine Learning*. Williamstown (MA), USA, 2001, S. 322–328. URL: https://www.cs.rutgers.edu/~mlittman/papers/icml01-ffq.pdf.

[Lua18] LuaRocks.org. *LuaRocks: The package manager for Lua modules*. 2018. URL: https://luarocks.org (besucht am 02. 02. 2018).

[M.+16] Johnson M., Hofmann K., Hutton T. und Bignell D. „The Malmo Platform for Artificial Intelligence Experimentation". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. Palo Alto (CA), USA: AAAI Press, 2016, S. 4246–4247. URL: http://www.ijcai.org/Proceedings/16/Papers/643.pdf.

[ME11] J. H. Metzen und M. Edgington. *Maja Machine Learning Framework*. 1. Jan. 2011. URL: http://mmlf.sourceforge.net (besucht am 02. 02. 2018).

[MS10] H. Maei und R. Sutton. „GQ(lambda): A general gradient algorithm for temporal-difference prediction learning with eligibility traces“. In: *Proceedings of the 3d Conference on Artificial General Intelligence*. Washington D.C., USA, 2010, S. 91–96. URL: http://download.atlantis-press.com/php/download_paper.php?id=1920.

[Mac+18a] J. A. MacGlashan u. a. *BURLAP*. 2018. URL: http://burlap.cs.brown.edu (besucht am 02. 02. 2018).

[Mac+18b] J. A. MacGlashan u. a. *BURLAP*. 2018. URL: https://github.com/jmacglashan/burlap (besucht am 02. 02. 2018).

[Mac13] J. A. MacGlashan. „Multi-Source Option-Based Policy Transfer“. Diss. University of Maryland, 2013.

[Mae+10] H. Maei, C. Szepesvári, S. Bhatnagar und R. Sutton. „Toward off-policy learning control with function approximation“. In: *Proceedings of the 27th International Conference on Machine Learning*. Haifa, Israel, 2010, S. 719–726. URL: http://www.ics.uci.edu/~dechter/courses/ics-295/winter-2018/papers/2010-sutton-ICML10_controlGQ.pdf.

[Mat+18] K. Mathewson u. a. *Torch-twrl: A package that enables reinforcement learning in Torch*. 2018. URL: https://github.com/twitter/torch-twrl (besucht am 02. 02. 2018).

[May18] V. Mayoral. *Basic Reinforcement Learning*. 2018. URL: https://github.com/vmayoral/basic_reinforcement_learning (besucht am 02. 02. 2018).

[Med18] M. Meder. *MMLF: Maja Machine Learning Framework*. 2018. URL: https://robotik.dfki-bremen.de/de/forschung/softwaretools/mmlf.html (besucht am 02. 02. 2018).

[Mni+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra und M. A. Riedmiller. „Playing Atari with Deep Reinforcement Learning“. In: *CoRR* abs/1312.5602 (2013). URL: http://arxiv.org/abs/1312.5602.

[Mni+15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg und D. Hassabis. „Human-level Control through Deep Reinforcement Learning“. In: *Nature* 518 (2015), S. 529–533.

[Mni+16] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver und K. Kavukcuoglu. „Asynchronous Methods for Deep Reinforcement Learning“. In: *CoRR* abs/1602.01783 (2016). URL: http://arxiv.org/abs/1602.01783.

[Mor+17] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, W. Paul, M. I. Jordan und I. Stoica. „Ray: A Distributed Framework for Emerging AI Applications“. In: *CoRR* abs/1712.05889 (2017). URL: https://arxiv.org/abs/1712.05889.

[Mot+18] B. W. Mott, S. Anthony, C. Speckner, T. Jentzsch, E. Stolberg u. a. *Stella*. 2018. URL: https://stella-emu.github.io (besucht am 02. 02. 2018).

[MuJ18] MuJoCo. *Advanced Physics Simulation*. 2018. URL: http://www.mujoco.org (besucht am 02. 02. 2018).

[Mue+12] K. Muelling, J. Kober, O. Kroemer und Peters. J. *Towards Learning Robot Table Tennis*. 24. Mai 2012. URL: https://www.youtube.com/watch?v=SH3bADiB7uQ (besucht am 28. 02. 2018).

[Mü+18] M.T. Münter u. a. *VIRTECOLAB*. 2018. URL: http://virtecolab.com (besucht am 28. 02. 2018).

[Net18] Preferred Networks. *Chainer: A Powerful, Flexible, and Intuitive Framework for Neural Networks*. 2018. URL: https://chainer.org (besucht am 02. 02. 2018).

[Nis+18] R. Nishihara, P. Moritz, E. Liang, R. Liaw, S. Wang u. a. *Ray: A high-performance distributed execution engine*. 2018. URL: https://github.com/ray-project/ray (besucht am 02. 02. 2018).

[Ols+18] C. Olsson, J. Smith, S. Stites, A. Schreiber u. a. *gym-http-api: API to access OpenAI Gym from other languages via HTTP*. 2018. URL: https://github.com/openai/gym-http-api (besucht am 02. 02. 2018).

[Ope18] OpenAI. *Gym: CartPole-v0*. 2018. URL: https://gym.openai.com/envs/CartPole-v0 (besucht am 20. 02. 2018).

[Pap15] Two Minute Papers. *Google DeepMind's Deep Q-learning playing Atari Breakout*. 7. März 2015. URL: https://www.youtube.com/watch?v=V1eYniJ0Rnk (besucht am 28. 02. 2018).

[Par+18] A. Park, S. Leishman, A. Thomas, U. Köster u. a. *Neon: Reference Deep Learning Framework*. 2018. URL: https://github.com/NervanaSystems/neon (besucht am 02. 02. 2018).

[Pet+16] L. Petterson, J. Tang, T. Brown, A. Radford, J. Schneider, S. Sidor u. a. *Universe*. 5. Dez. 2016. URL: https://blog.openai.com/universe (besucht am 02. 02. 2018).

[Pla18a] M. Plappert. *Keras-RL: Deep Reinforcement Learning for Keras*. 2018. URL: https://github.com/matthiasplappert/keras-rl (besucht am 02. 02. 2018).

[Pla18b] M. Plappert. *Keras-RL Documentation*. 2018. URL: http://keras-rl.readthedocs.io (besucht am 02. 02. 2018).

[PyT18] PyTorch. *PyTorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration*. 2018. URL: http://pytorch.org (besucht am 02. 02. 2018).

[RGB11] S. Ross, G. J. Gordon und J. A. Bagnell. „No-Regret Reductions for Imitation Learning and Structured Prediction". In: *CoRR* abs/1011.0686 (2011). URL: http://arxiv.org/abs/1011.0686.

[RN94] G. A. Rummery und M. Niranjan. *On-line Q-Learning using connectionist systems*. Techn. Ber. CUED/F-INFENG/TR 166. Engineering department, Cambridge University, 1994. URL: ftp://mi.eng.cam.ac.uk/pub/reports/auto-pdf/rummery_tr166.pdf.

[Rat+18] F. Ratnikov, O. Vasilev, P. Shvechikov und A. Fritsler. *Practical-RL: A course in reinforcement learning in the wild*. 2018. URL: https://github.com/yandexdataschool/Practical_RL (besucht am 02. 02. 2018).

[Ret16] G. Retkowski. *Actor Critic with OpenAI Gym*. 5. Juli 2016. URL: http://www.rage.net/~greg/2016-07-05-ActorCritic-with-OpenAI-Gym.html (besucht am 02. 02. 2018).

[Rie10] M. Riedmiller. *Cart Pole Balancing: Learning to Swing-Up and Balance from Scratch*. 16. März 2010. URL: https://www.youtube.com/watch?v=LtKLtkDlh8 (besucht am 28. 02. 2018).

[Rö09] M. Röttger. „Reinforcement Learning für kontinuierliche Zustands- und Aktionsräume unter Berücksichtigung der wissenschaftlichen Informationsverarbeitung“. Diss. Albert-Ludwigs-Universität Freiburg i. Brsg., 2009.

[SB18] R. S. Sutton und A. G. Barto. *Reinforcement Learning: An Introduction*. 2. Aufl. Cambridge (MA), USA: MIT Press, 2018. URL: http://incompleteideas.net/book/bookdraft2018jan1.pdf (besucht am 02. 02. 2018).

[SB98] R. S. Sutton und A. G. Barto. *Reinforcement Learning: An Introduction*. 1. Aufl. Cambridge (MA), USA: MIT Press, 1998.

[SBR+18] T. Schaul, J. Bayer, T. Rückstieß u. a. *PyBrain*. 2018. URL: https://github.com/pybrain/pybrain (besucht am 02. 02. 2018).

[SKF+18] M. Schaarschmidt, A. Kuhnle, K. Fricke u. a. *TensorForce: A TensorFlow library for applied reinforcement learning*. 2018. URL: https://github.com/reinforceio/tensorforce (besucht am 02. 02. 2018).

[SL06] I. Szita und A. Lőrincz. „Learning Tetris Using the Noisy Cross-Entropy Method“. In: *Neural Computation* 18 (2006), S. 2936–2941. URL: https://www.researchgate.net/publication/6743957_Learning_Tetris_Using_the_Noisy_Cross-Entropy_Method.

[Sal+17] T. Salimans, J. Ho, X. Chen, S. Sidor und I. Sutskever. „Evolution Strategies as a Scalable Alternative to Reinforcement Learning“. In: *CoRR* abs/1703.03864 (2017). URL: https://arxiv.org/abs/1703.03864.

[Sch+10] T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß und J. Schmidhuber. „PyBrain“. In: *Journal of Machine Learning Research* 11 (2010), S. 743–746. URL: http://www.jmlr.org/papers/volume11/schaul10a/schaul10a.pdf.

[Sch+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford und O. Klimov. „Proximal Policy Optimization Algorithms“. In: *CoRR* abs/1707.06347 (2017). URL: http://arxiv.org/abs/1707.06347.

[Sch17] C. Schoeman. *Baby antelope's first steps*. 30. Jan. 2017. URL: https://www.youtube.com/watch?v=fLZOI0CqBm8 (besucht am 28. 02. 2018).

[Shi18] P. Shinners. *PyGame*. 2018. URL: http://www.pygame.org (besucht am 02. 02. 2018).

[Sil+14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra und M. Riedmiller. „Deterministic Policy Gradient Algorithms“. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, S. I–387–I–395. URL: http://proceedings.mlr.press/v32/silver14.pdf.

[Sil15] D. Silver. *UCL Course on RL*. 1. Jan. 2015. URL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html (besucht am 02. 02. 2018).

[Sil18] D. Silver. *Deep Reinforcement Learning*. 2018. URL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Resources_files/deep_rl.pdf (besucht am 24. 02. 2018).

[Sod+13] E. Sodomka, E. Hilliard, M. Littman und A. Greenwald. „Coco-Q: Learning in Stochastic Games with Side Payments“. In: *Proceedings of the 30th International Conference on Machine Learning*. Atlanta (GA), USA, 2013, S. 1471–1479. URL: http://proceedings.mlr.press/v28/sodomka13.pdf.

[Sub+18] F. Subirón N. Codevilla u. a. *CARLA: Open-source simulator for autonomous driving research*. 2018. URL: https://github.com/carla-simulator/carla (besucht am 02. 02. 2018).

[Sut16] R. S. Sutton. *Reinforcement Learning Toolkit*. 1. Okt. 2016. URL: http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/RLtoolkit/RLtoolkit1.0.html (besucht am 02. 02. 2018).

[Sut96] R. S. Sutton. „Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding“. In: *Advances in Neural Information Processing Systems* 8 (1996), S. 1038–1044. URL: https://papers.nips.cc/paper/1109-generalization-in-reinforcement-learning-successful-examples-using-sparse-coarse-coding.pdf.

[Sze10] C. Szepesvári. *Algorithms for Reinforcement Learning*. 1. Aufl. San Rafael (CA), USA: Morgan & Claypool, 2010.

[TU18] N. Tasfi und E. Urtans. *PyGame Learning Environment (PLE): Reinforcement Learning Environment in Python*. 2018. URL: https://github.com/ntasfi/PyGame-Learning-Environment (besucht am 02. 02. 2018).

[TW09] B. Tanner und A. White. „RL-Glue: Language-Independent Software for Reinforcement-Learning Experiments“. In: *Journal of Machine Learning Research* 10 (2009), S. 2133–2136. URL: http://glue.rl-community.org/images/c/c2/Rlglue-mloss-jmlr-2009-PREPRINT.pdf.

[TW10] B. Tanner und A. White. *RL-Glue*. 19. Nov. 2010. URL: http://glue.rl-community.org (besucht am 02. 02. 2018).

[Ten18] TensorFlow. *An open-source software library for Machine Intelligence*. 2018. URL: https://www.tensorflow.org (besucht am 02. 02. 2018).

[Tok+15] S. Tokui, K. Oono, S. Hido und J. Clayton. „Chainer: a Next-Generation Open Source Framework for Deep Learning“. In: *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*. 2015. URL: http://learningsys.org/papers/LearningSys_2015_paper_33.pdf.

[Tok09] M. Tokic. *A crawling robot learning a forward walking policy on a carpet*. 14. Mai 2009. URL: https://www.youtube.com/watch?v=2VjNTQ5cGzM (besucht am 28. 02. 2018).

[WFL15] Z. Wang, N. de Freitas und M. Lanctot. „Dueling Network Architectures for Deep Reinforcement Learning“. In: *CoRR* abs/1511.06581 (2015). URL: http://arxiv.org/abs/1511.06581.

[WYL+18] L. W. Woongwon, H. Yang, B. Lee u. a. *Minimal and Clean Reinforcement Learning Examples*. 2018. URL: https://github.com/rlcode/reinforcement-learning (besucht am 02. 02. 2018).

[Wat89] C. J. C. H. Watkins. „Learning from Delayed Rewards“. Diss. Cambridge University, 1989. URL: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

[Wik18a] Wikipedia. *Atari 2600*. 2018. URL: https://de.wikipedia.org/wiki/Atari_2600 (besucht am 02. 02. 2018).

[Wik18b] Wikipedia. *C++*. 2018. URL: https://de.wikipedia.org/wiki/C\%2B\%2B (besucht am 09. 02. 2018).

[Wik18c] Wikipedia. *Docker*. 2018. URL: https://de.wikipedia.org/wiki/Docker_(Software) (besucht am 02. 02. 2018).

[Wik18d] Wikipedia. *Doom*. 2018. URL: https://de.wikipedia.org/wiki/Doom (besucht am 02. 02. 2018).

[Wik18e] Wikipedia. *Flappy Bird*. 2018. URL: https://de.wikipedia.org/wiki/Flappy_Bird (besucht am 02. 02. 2018).

[Wik18f] Wikipedia. *Git*. 2018. URL: https://de.wikipedia.org/wiki/Git (besucht am 09. 02. 2018).

[Wik18g] Wikipedia. *GitHub*. 2018. URL: https://de.wikipedia.org/wiki/GitHub (besucht am 09. 02. 2018).

[Wik18h] Wikipedia. *Huber Loss*. 2018. URL: https://en.wikipedia.org/wiki/Huber_loss (besucht am 21. 02. 2018).

[Wik18i] Wikipedia. *Inverses Pendel*. 2018. URL: https://de.wikipedia.org/wiki/Inverses_Pendel (besucht am 13. 02. 2018).

[Wik18j] Wikipedia. *Java*. 2018. URL: https://de.wikipedia.org/wiki/Java_(Programmiersprache) (besucht am 09. 02. 2018).

[Wik18k] Wikipedia. *Lua*. 2018. URL: https://de.wikipedia.org/wiki/Lua (besucht am 09. 02. 2018).

[Wik18l] Wikipedia. *Python*. 2018. URL: https://de.wikipedia.org/wiki/Python_(Programmiersprache) (besucht am 09. 02. 2018).

[Wik18m] Wikipedia. *Super Nintendo Entertainment System*. 2018. URL: https://de.wikipedia.org/wiki/Super_Nintendo_Entertainment_System (besucht am 02. 02. 2018).

[Wik18n] Wikipedia. *Unreal Engine*. 2018. URL: https://de.wikipedia.org/wiki/Unreal_Engine (besucht am 02. 02. 2018).

[Wik18o] Wikipedia. *Virtual Network Computing*. 2018. URL: https://de.wikipedia.org/wiki/Virtual_Network_Computing (besucht am 02. 02. 2018).

[Wik18p] Wikipedia. *WebSocket*. 2018. URL: https://de.wikipedia.org/wiki/WebSocket (besucht am 02. 02. 2018).

[Wyf10] F. Wyffels. *A crawling robot: a Q-learning example*. 29. Jan. 2010. URL: https://www.youtube.com/watch?v=2iNrJx6IDEo (besucht am 28. 02. 2018).

[ZC18] S. Zhang und M. Clark. *DeepRL: Highly modularized implementation of popular deep RL algorithms by PyTorch*. 2018. URL: https://github.com/ShangtongZhang/DeepRL (besucht am 02. 02. 2018).

[Zan70] C. Zangemeister. „Nutzwertanalyse in der Systemtechnik". Diss. Technische Universität Berlin, 1970.

[Zha+18] S. Zhang u. a. *Python implementation of Reinforcement Learning: An Introduction*. 2018. URL: https://github.com/ShangtongZhang/reinforcement-learning-an-introduction (besucht am 02. 02. 2018).

[Zhe+17] L. Zheng, J. Yang, H. Cai, W. Zhang, J. Wang und Y. Yu. „MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence". In: *ArXiv E-Prints* (2017). URL: https://arxiv.org/abs/1712.00600v1.

[Zhe+18] L. Zheng u. a. *MAgent: A Platform for Many-agent Reinforcement Learning*. 2018. URL: https://github.com/geek-ai/MAgent (besucht am 02. 02. 2018).

[Zho18] M. Zhou. *Simple Reinforcement learning tutorials*. 2018. URL: https://github.com/MorvanZhou/Reinforcement-learning-with-tensorflow (besucht am 02. 02. 2018).

[vGS15] H. van Hasselt, A. Guez und D. Silver. „Deep Reinforcement Learning with Double Q-Learning“. In: *29th AAAI Conference on Artificial Intelligence*. Austin (TX), USA, 2015, S. 2094–2100. URL: https://arxiv.org/abs/1509.06461.

[van+09] H. van Seijen, H. van Hasselt, S. Whiteson und M. Wiering. „A Theoretical and Empirical Analysis of Expected Sarsa“. In: *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. Nashville (TN), USA, 2009, S. 177–184. URL: http://www.cs.ox.ac.uk/people/shimon.whiteson/pubs/vanseijenadprl09.pdf.

[van10] H. van Hasselt. „Double Q-learning“. In: *Advances in Neural Information Processing Systems 23*. Hrsg. von J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel und A. Culotta. Curran Associates, Inc., 2010, S. 2613–2621. URL: http://papers.nips.cc/paper/3964-double-q-learning.pdf.

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar der Fakultät für Wirtschaftswissenschaften im Bereich Wirtschaftsinformatik erstellt (Version 1.0). Die Vorlage wurde von Stefan Selle erstellt und basiert weitestgehend auf der entsprechenden Vorlage der Fakultät für Ingenieurswissenschaften, die von Yves Hary, André Miede, Thomas Kretschmer, Helmut G. Folz und Martina Lehser entwickelt wurde.