

Deep Reinforcement Learning

Das umfassende
Praxis-Handbuch

Moderne Algorithmen für Chatbots, Robotik,
diskrete Optimierung und Web-Automatisierung
inkl. Multiagenten-Methoden

Inhaltsverzeichnis

Über den Autor	17
Über die Korrektoren.....	17
Über den Fachkorrektor der deutschen Ausgabe	18
Einleitung.....	19
Teil I Grundlagen des Reinforcement Learnings.....	24
1 Was ist Reinforcement Learning?.....	25
1.1 Überwachtes Lernen	25
1.2 Unüberwachtes Lernen	26
1.3 Reinforcement Learning	26
1.4 Herausforderungen beim Reinforcement Learning	28
1.5 RL-Formalismen	28
1.5.1 Belohnung	29
1.5.2 Der Agent.....	31
1.5.3 Die Umgebung	31
1.5.4 Aktionen.....	31
1.5.5 Beobachtungen	32
1.6 Die theoretischen Grundlagen des Reinforcement Learnings.....	34
1.6.1 Markov-Entscheidungsprozesse.....	35
1.6.2 Markov-Prozess	35
1.6.3 Markov-Belohnungsprozess	39
1.6.4 Aktionen hinzufügen	42
1.6.5 Policy	44
1.7 Zusammenfassung	45
2 OpenAI Gym	47
2.1 Aufbau des Agenten	47
2.2 Anforderungen an Hard- und Software.....	50
2.3 OpenAI-Gym-API	51
2.3.1 Aktionsraum	52
2.3.2 Beobachtungsraum.....	52
2.3.3 Die Umgebung	54
2.3.4 Erzeugen der Umgebung	55
2.3.5 Die CartPole-Sitzung.....	57
2.4 Ein CartPole-Agent nach dem Zufallsprinzip	59

2.5	Zusätzliche Gym-Funktionalität: Wrapper und Monitor	60
2.5.1	Wrapper	61
2.5.2	Monitor	63
2.6	Zusammenfassung	66
3	Deep Learning mit PyTorch	67
3.1	Tensoren	67
3.1.1	Tensoren erzeugen	68
3.1.2	Skalare Tensoren	70
3.1.3	Tensor-Operationen	71
3.1.4	GPU-Tensoren	71
3.2	Gradienten	72
3.2.1	Tensoren und Gradienten	74
3.3	NN-Bausteine.	76
3.4	Benutzerdefinierte Schichten.	78
3.5	Verlustfunktionen und Optimierer	80
3.5.1	Verlustfunktionen.	81
3.5.2	Optimierer	81
3.6	Monitoring mit TensorBoard	83
3.6.1	Einführung in TensorBoard.	84
3.6.2	Plotten	85
3.7	Beispiel: GAN für Bilder von Atari-Spielen.	87
3.8	PyTorch Ignite	92
3.8.1	Konzepte	93
3.9	Zusammenfassung	97
4	Das Kreuzentropie-Verfahren.	99
4.1	Klassifikation von RL-Verfahren	99
4.2	Kreuzentropie in der Praxis	100
4.3	Kreuzentropie beim CartPole	102
4.4	Kreuzentropie beim FrozenLake	111
4.5	Theoretische Grundlagen des Kreuzentropie-Verfahrens	118
4.6	Zusammenfassung	119
Teil II	Wertebasierte Verfahren	120
5	Tabular Learning und das Bellman'sche Optimalitätsprinzip.	121
5.1	Wert, Zustand und Optimalität	121
5.2	Das Bellman'sche Optimalitätsprinzip	123
5.3	Aktionswert	126
5.4	Wertiteration	128
5.5	Wertiteration in der Praxis	130
5.6	Q-Learning in der FrozenLake-Umgebung.	136
5.7	Zusammenfassung	138

6	Deep Q-Networks	139
6.1	Wertiteration in der Praxis	139
6.2	Tabular Q-Learning	140
6.3	Deep Q-Learning	145
	6.3.1 Interaktion mit der Umgebung	147
	6.3.2 SGD-Optimierung.	147
	6.3.3 Korrelation der Schritte.	148
	6.3.4 Die Markov-Eigenschaft	148
	6.3.5 Die endgültige Form des DQN-Trainings.	149
6.4	DQN mit Pong	150
	6.4.1 Wrapper	151
	6.4.2 DQN-Modell	156
	6.4.3 Training	158
	6.4.4 Ausführung und Leistung	167
	6.4.5 Das Modell in Aktion	170
6.5	Weitere Möglichkeiten	172
6.6	Zusammenfassung	173
7	Allgemeine RL-Bibliotheken	175
7.1	Warum RL-Bibliotheken?	175
7.2	Die PTAN-Bibliothek	176
	7.2.1 Aktionsselektoren	177
	7.2.2 Der Agent.	179
	7.2.3 Quelle der Erfahrungswerte	183
	7.2.4 Replay Buffer für Erfahrungswerte	189
	7.2.5 Die TargetNet-Klasse.	191
	7.2.6 Hilfsfunktionen für Ignite	193
7.3	Lösung der CartPole-Umgebung mit PTAN	194
7.4	Weitere RL-Bibliotheken	196
7.5	Zusammenfassung	197
8	DQN-Erweiterungen	199
8.1	Einfaches DQN	199
	8.1.1 Die Bibliothek common	200
	8.1.2 Implementierung	205
	8.1.3 Ergebnisse	207
8.2	N-Schritt-DQN	208
	8.2.1 Implementierung	211
	8.2.2 Ergebnisse	211
8.3	Double DQN.	212
	8.3.1 Implementierung	213
	8.3.2 Ergebnisse	215
8.4	Verrauschte Netze	216
	8.4.1 Implementierung	217
	8.4.2 Ergebnisse	219

8.5	Priorisierter Replay Buffer	220
8.5.1	Implementierung	221
8.5.2	Ergebnisse	225
8.6	Rivalisierendes DQN	227
8.6.1	Implementierung	228
8.6.2	Ergebnisse	229
8.7	Kategoriales DQN	230
8.7.1	Implementierung	232
8.7.2	Ergebnisse	239
8.8	Alles miteinander kombinieren	241
8.8.1	Ergebnisse	242
8.9	Zusammenfassung	243
8.10	Quellenangaben	244
9	Beschleunigung von RL-Verfahren	245
9.1	Die Bedeutung der Geschwindigkeit	245
9.2	Der Ausgangspunkt	248
9.3	Der Berechnungsgraph in PyTorch	250
9.4	Mehrere Umgebungen	252
9.5	Spielen und Trainieren in separaten Prozessen	255
9.6	Optimierung der Wrapper	259
9.7	Zusammenfassung der Benchmarks	265
9.8	Atari-Emulation: CuLE	265
9.9	Zusammenfassung	266
9.10	Quellenangaben	266
10	Aktienhandel per Reinforcement Learning	267
10.1	Börsenhandel	267
10.2	Daten	268
10.3	Aufgabenstellungen und Grundsatzentscheidungen	269
10.4	Die Handelsumgebung	270
10.5	Modelle	279
10.6	Trainingscode	281
10.7	Ergebnisse	281
10.7.1	Das Feedforward-Modell	281
10.7.2	Das Faltungsmodell	287
10.8	Weitere Möglichkeiten	288
10.9	Zusammenfassung	289
Teil III Policybasierte Verfahren		290
11	Eine Alternative: Policy Gradients	291
11.1	Werte und Policy	291
11.1.1	Warum Policy?	292

11.1.2	Repräsentation der Policy	292
11.1.3	Policy Gradients	293
11.2	Das REINFORCE-Verfahren	294
11.2.1	Das CartPole-Beispiel	295
11.2.2	Ergebnisse	299
11.2.3	Policybasierte und wertebasierte Verfahren	300
11.3	Probleme mit REINFORCE	301
11.3.1	Notwendigkeit vollständiger Episoden	301
11.3.2	Große Varianz der Gradienten	302
11.3.3	Exploration	302
11.3.4	Korrelation zwischen Beispielen	303
11.4	PG mit CartPole	303
11.4.1	Implementierung	303
11.4.2	Ergebnisse	306
11.5	PG mit Pong	310
11.5.1	Implementierung	311
11.5.2	Ergebnisse	312
11.6	Zusammenfassung	313
12	Das Actor-Critic-Verfahren	315
12.1	Verringern der Varianz	315
12.2	Varianz der CartPole-Umgebung	317
12.3	Actor-Critic	320
12.4	A2C mit Pong	322
12.5	A2C mit Pong: Ergebnisse	328
12.6	Optimierung der Hyperparameter	331
12.6.1	Lernrate	332
12.6.2	Beta	333
12.6.3	Anzahl der Umgebungen	333
12.6.4	Batchgröße	333
12.7	Zusammenfassung	333
13	Asynchronous Advantage Actor Critic	335
13.1	Korrelation und Stichprobeneffizienz	335
13.2	Ein weiteres A zu A2C hinzufügen	336
13.3	Multiprocessing in Python	339
13.4	A3C mit Datenparallelität	339
13.4.1	Implementierung	339
13.4.2	Ergebnisse	346
13.5	A3C mit Gradientenparallelität	347
13.5.1	Implementierung	348
13.5.2	Ergebnisse	353
13.6	Zusammenfassung	354
14	Chatbot-Training per Reinforcement Learning	355
14.1	Chatbots – ein Überblick	355

14.2	Chatbot-Training	356
14.3	Grundlagen der Verarbeitung natürlicher Sprache	357
	14.3.1 Rekurrente neuronale Netze	357
	14.3.2 Wort-Embeddings	359
	14.3.3 Encoder-Decoder	360
14.4	Seq2Seq-Training	361
	14.4.1 Log-Likelihood-Training	361
	14.4.2 Der BLEU-Score	363
	14.4.3 RL und Seq2Seq	364
	14.4.4 Self-critical Sequence Training	365
14.5	Das Chatbot-Beispiel	366
	14.5.1 Aufbau des Beispiels	366
	14.5.2 Module: cornell.py und data.py	367
	14.5.3 BLEU-Score und utils.py	368
	14.5.4 Modell	369
14.6	Daten überprüfen	376
14.7	Training: Kreuzentropie	378
	14.7.1 Implementierung	378
	14.7.2 Ergebnisse	382
14.8	Training: Self-critical Sequence Training (SCST)	385
	14.8.1 Implementierung	385
	14.8.2 Ergebnisse	392
14.9	Tests der Modelle mit Daten	395
14.10	Telegram-Bot	397
14.11	Zusammenfassung	401
15	Die TextWorld-Umgebung	403
15.1	Interactive Fiction	403
15.2	Die Umgebung	406
	15.2.1 Installation	407
	15.2.2 Spiel erzeugen	407
	15.2.3 Beobachtungs- und Aktionsräume	409
	15.2.4 Zusätzliche Informationen	411
15.3	Einfaches DQN	414
	15.3.1 Vorverarbeitung von Beobachtungen	416
	15.3.2 Embeddings und Encoder	421
	15.3.3 DQN-Modell und Agent	424
	15.3.4 Trainingscode	426
	15.3.5 Trainingsergebnisse	426
15.4	Das Modell für den Befehlsgenerator	431
	15.4.1 Implementierung	433
	15.4.2 Ergebnisse des Pretrainings	437
	15.4.3 DQN-Trainingscode	439
	15.4.4 Ergebnis des DQN-Trainings	441
15.5	Zusammenfassung	442

16	Navigation im Web	443
16.1	Webnavigation	443
	16.1.1 Browserautomatisierung und RL	444
	16.1.2 Mini World of Bits	445
16.2	OpenAI Universe	446
	16.2.1 Installation	447
	16.2.2 Aktionen und Beobachtungen	448
	16.2.3 Umgebung erzeugen	449
	16.2.4 MiniWoB-Stabilität	451
16.3	Einfaches Anklicken	451
	16.3.1 Aktionen auf dem Gitter	452
	16.3.2 Übersicht der Beispiele	453
	16.3.3 Modell	454
	16.3.4 Trainingscode	455
	16.3.5 Container starten	460
	16.3.6 Trainingsprozess	461
	16.3.7 Überprüfen der erlernten Policy	464
	16.3.8 Probleme mit einfachem Anklicken	465
16.4	Demonstrationen durch den Menschen	467
	16.4.1 Aufzeichnung von Demonstrationen	468
	16.4.2 Aufzeichnungsformat	470
	16.4.3 Training durch Demonstration	473
	16.4.4 Ergebnisse	474
	16.4.5 Tic-Tac-Toe	478
16.5	Hinzufügen von Beschreibungstext	480
	16.5.1 Implementierung	481
	16.5.2 Ergebnisse	486
16.6	Weitere Möglichkeiten	489
16.7	Zusammenfassung	489

Teil IV Fortgeschrittene Verfahren und Techniken

17	Stetige Aktionsräume	491
17.1	Wozu stetige Aktionsräume?	491
17.2	Aktionsraum	492
17.3	Umgebungen	492
17.4	Das A2C-Verfahren	495
	17.4.1 Implementierung	496
	17.4.2 Ergebnisse	499
	17.4.3 Modelle verwenden und Videos aufzeichnen	501
17.5	Deterministisches Policy-Gradienten-Verfahren	502
	17.5.1 Exploration	503
	17.5.2 Implementierung	504
	17.5.3 Ergebnisse	509

17.5.4	Videos aufzeichnen	511
17.6	Distributional Policy Gradients	511
17.6.1	Architektur	512
17.6.2	Implementierung	512
17.6.3	Ergebnisse	517
17.6.4	Videoaufzeichnung	519
17.7	Weitere Möglichkeiten	519
17.8	Zusammenfassung	519
18	RL in der Robotik	521
18.1	Roboter und Robotik	521
18.1.1	Komplexität von Robotern	523
18.1.2	Hardware	524
18.1.3	Plattform	525
18.1.4	Sensoren	526
18.1.5	Aktuatoren	528
18.1.6	Rahmen	528
18.2	Ein erstes Trainingsziel	532
18.3	Emulator und Modell	534
18.3.1	Definitionsdatei des Modells	535
18.3.2	Die robot-Klasse	539
18.4	DDPG-Training und Ergebnisse	545
18.5	Steuerung der Hardware	548
18.5.1	MicroPython	548
18.5.2	Handhabung von Sensoren	552
18.5.3	Servos ansteuern	565
18.5.4	Einrichtung des Modells auf der Hardware	569
18.5.5	Alles kombinieren	577
18.6	Experimente mit der Policy	580
18.7	Zusammenfassung	581
19	Trust Regions – PPO, TRPO, ACKTR und SAC	583
19.1	Roboschool	584
19.2	Standard-A2C-Verfahren	584
19.2.1	Implementierung	584
19.2.2	Ergebnisse	586
19.2.3	Videoaufzeichnungen	590
19.3	Proximal Policy Optimization (PPO)	590
19.3.1	Implementierung	591
19.3.2	Ergebnisse	595
19.4	Trust Region Policy Optimization (TRPO)	597
19.4.1	Implementierung	597
19.4.2	Ergebnisse	599
19.5	Advantage Actor-Critic mit Kronecker-Factored Trust Region (ACKTR)	600
19.5.1	Implementierung	601

19.5.2	Ergebnisse	601
19.6	Soft-Actor-Critic (SAC)	602
19.6.1	Implementierung	603
19.6.2	Ergebnisse	605
19.7	Zusammenfassung	607
20	Blackbox-Optimierung beim Reinforcement Learning.	609
20.1	Blackbox-Verfahren	609
20.2	Evolutionsstrategien (ES)	610
20.3	ES mit CartPole	611
20.3.1	Ergebnisse	616
20.4	ES mit HalfCheetah	617
20.4.1	Implementierung	618
20.4.2	Ergebnisse	622
20.5	Genetische Algorithmen (GA)	624
20.6	GA mit CartPole	624
20.6.1	Ergebnisse	626
20.7	GA-Optimierung	627
20.7.1	Deep GA	628
20.7.2	Novelty Search	628
20.8	GA mit HalfCheetah	628
20.8.1	Ergebnisse	631
20.9	Zusammenfassung	633
20.10	Quellenangaben	633
21	Fortgeschrittene Exploration.	635
21.1	Die Bedeutung der Exploration	635
21.2	Was ist das Problem beim ϵ -Greedy-Ansatz?	636
21.3	Alternative Explorationsverfahren	639
21.3.1	Verrauschte Netze	639
21.3.2	Zählerbasierte Verfahren	640
21.3.3	Vorhersagebasierte Verfahren	641
21.4	MountainCar-Experimente	641
21.4.1	Das DQN-Verfahren mit ϵ -Greedy-Ansatz	643
21.4.2	Das DQN-Verfahren mit verrauschten Netzen	644
21.4.3	Das DQN-Verfahren mit Zustandszählern	646
21.4.4	Das PPO-Verfahren	649
21.4.5	Das PPO-Verfahren mit verrauschten Netzen	652
21.4.6	Das PPO-Verfahren mit zählerbasierter Exploration	654
21.4.7	Das PPO-Verfahren mit Netz-Destillation	656
21.5	Atari-Experimente	658
21.5.1	Das DQN-Verfahren mit ϵ -Greedy-Ansatz	659
21.5.2	Das klassische PPO-Verfahren	660
21.5.3	Das PPO-Verfahren mit Netz-Destillation	661
21.5.4	Das PPO-Verfahren mit verrauschten Netzen	662

21.6	Zusammenfassung	663
21.7	Quellenangaben.....	663
22	Jenseits modellfreier Verfahren – Imagination	665
22.1	Modellbasierte Verfahren	665
22.1.1	Modellbasierte und modellfreie Verfahren.....	665
22.2	Unzulänglichkeiten der Modelle	666
22.3	Imagination-augmented Agent	668
22.3.1	Das Umgebungsmodell	669
22.3.2	Die Rollout-Policy	670
22.3.3	Der Rollout-Encoder	670
22.3.4	Ergebnisse der Arbeit	670
22.4	I2A mit dem Atari-Spiel Breakout	670
22.4.1	Der Standard-A2C-Agent	671
22.4.2	Training des Umgebungsmodells	672
22.4.3	Der Imagination-Agent	675
22.5	Ergebnisse der Experimente.....	681
22.5.1	Der Basis-Agent	681
22.5.2	Training der EM-Gewichte	683
22.5.3	Training mit dem I2A-Modell	685
22.6	Zusammenfassung	688
22.7	Quellenangaben.....	688
23	AlphaGo Zero	689
23.1	Brettspiele	689
23.2	Das AlphaGo-Zero-Verfahren.....	690
23.2.1	Überblick.....	690
23.2.2	Monte-Carlo-Baumsuche	691
23.2.3	Self-Playing	693
23.2.4	Training und Bewertung	694
23.3	Vier-gewinnt-Bot	694
23.3.1	Spielmodell	695
23.3.2	Implementierung der Monte-Carlo-Baumsuche	697
23.3.3	Modell	702
23.3.4	Training.....	705
23.3.5	Test und Vergleich	705
23.4	Vier gewinnt: Ergebnisse	706
23.5	Zusammenfassung	708
23.6	Quellenangaben.....	708
24	RL und diskrete Optimierung	709
24.1	Die Reputation von Reinforcement Learnings	709
24.2	Zauberwürfel und kombinatorische Optimierung.....	710
24.3	Optimalität und Gottes Zahl.....	711
24.4	Ansätze zur Lösung	712

24.4.1	Datenrepräsentation	712
24.4.2	Aktionen.	712
24.4.3	Zustände	713
24.5	Trainingsvorgang.	717
24.5.1	Architektur des neuronalen Netzes	717
24.5.2	Training	718
24.6	Anwendung des Modells.	719
24.7	Ergebnisse der Arbeit	721
24.8	Code	722
24.8.1	Würfel-Umgebungen	723
24.8.2	Training	727
24.8.3	Suchvorgang	729
24.9	Ergebnisse des Experiments	729
24.9.1	Der 2x2-Würfel	731
24.9.2	Der 3x3-Würfel	733
24.9.3	Weitere Verbesserungen und Experimente	734
24.10	Zusammenfassung	735
25	RL mit mehreren Agenten	737
25.1	Mehrere Agenten.	737
25.1.1	Kommunikationsformen	738
25.1.2	Der RL-Ansatz	738
25.2	Die MAgent-Umgebung	738
25.2.1	Installation	739
25.2.2	Überblick	739
25.2.3	Eine zufällige Umgebung.	739
25.3	Deep Q-Networks für Tiger	745
25.3.1	Training und Ergebnisse	748
25.4	Zusammenarbeit der Tiger	750
25.5	Training der Tiger und Hirsche	754
25.6	Der Kampf ebenbürtiger Akteure	755
25.7	Zusammenfassung	756
	Stichwortverzeichnis	757

Über den Autor

Maxim Lapan ist Deep-Learning-Enthusiast und unabhängiger Forscher. Er verfügt über 15 Jahre Erfahrung als Softwareentwickler und Systemarchitekt. Er hat Linux-Kernel-Treiber entwickelt und verteilte Anwendungen entworfen und optimiert, die auf Tausenden Servern laufen. Er besitzt umfangreiche Erfahrung mit Big Data, Machine Learning und großen HPC- und Nicht-HPC-Systemen und hat das Talent, komplizierte Dinge in einfacher Sprache und mit anschaulichen Beispielen zu erklären. Derzeit beschäftigt er sich insbesondere mit praktischen Anwendungen des Deep Learnings, wie der Verarbeitung natürlicher Sprache (*Natural Language Processing*, NLP) und Deep Reinforcement Learning.

Maxim lebt mit seiner Familie in Moskau.

Ich möchte meiner Frau Olga und meinen Kindern Ksenia, Julia und Fedor für ihre Geduld und ihre Unterstützung danken. Dieses Buch zu schreiben stellte eine Herausforderung dar, und es wäre ohne euch nicht möglich gewesen, danke! Julia und Fedor haben beim Sammeln von Beispielen für MiniWoB (Kapitel 16, Navigation im Web) und beim Testen der Spielstärke des Vier gewinnt-Agenten (Kapitel 23, AlphaZero Go) großartige Arbeit geleistet.

Über die Korrektoren

Mikhail Yurushkins Forschungsgebiete sind High-performance Computing und die Optimierung der Compiler-Entwicklung. Er ist Dozent an der SFEDU-Universität in Rostow am Don. Er gibt Kurse über fortgeschrittenes Deep Learning beim maschinellen Sehen und der Verarbeitung natürlicher Sprache. Er befasst sich seit mehr als acht Jahren mit plattformübergreifender Entwicklung in C++, Machine Learning und Deep Learning. Er ist Unternehmer und Gründer mehrerer Start-ups, unter anderem von BroutonLab, einem Data-Science-Unternehmen, das auf die Entwicklung KI-gestützter Softwareprodukte spezialisiert ist.

Per-Arne Andersen ist Doktorand an der Universität Agder in Norwegen und beschäftigt sich mit Reinforcement Learning. Er hat mehrere technische Arbeiten über den Einsatz von Reinforcement Learning bei Spielen verfasst und wurde für seine Forschung über modellbasiertes Reinforcement Learning von der British Computer Society als bester Student ausgezeichnet. Per-Arne Andersen ist außerdem Experte für Netzwerksicherheit und seit 2012 auf diesem Gebiet tätig. Seine aktuellen Forschungsinteressen sind Machine Learning, Deep Learning, Netzwerksicherheit und Reinforcement Learning.

Sergey Kolesnikov ist in Industrie und Wissenschaft als Forscher tätig und hat mehr als fünf Jahre Erfahrung mit Machine Learning, Deep Learning und Reinforcement Learning. Er arbeitet derzeit an industriellen Anwendungen, die Computervisualistik, Verarbeitung natürlicher Sprache und Empfehlungsdienste nutzen, und beteiligt sich an der Forschung zum Thema Reinforcement Learning. Er ist außerdem an Entscheidungsfindungsprozessen und Psychologie interessiert. Er ist Gewinner eines Wettbewerbs der Conference on Neural Information Processing Systems und Anhänger von Open Source. Darüber hinaus ist er Entwickler von Catalyst, einer High-Level-Umgebung für PyTorch zum Beschleunigen der Forschung und Entwicklung beim Deep Learning/Reinforcement Learning.

Über den Fachkorrektor der deutschen Ausgabe

Friedhelm Schwenker ist Privatdozent für Informatik (Fachgebiet: Machine Learning) an der Universität Ulm. Er hat im Bereich der Angewandten Mathematik promoviert und ist seit vielen Jahren im Bereich Machine Learning in Forschung und Lehre tätig. Seine Forschungsgebiete sind Pattern Recognition, Data Mining und Machine Learning mit Schwerpunkt Neuronale Netze. In jüngster Zeit befasst er sich auch mit Anwendungen des Machine Learnings im Affective Computing. Er ist Editor von 19 Proceedingsbänden und Special Issues sowie Autor von 200+ Journal- und Konferenzartikeln.



Einleitung

Das Thema dieses Buchs ist Reinforcement Learning (verstärkendes Lernen), ein Teilgebiet des Machine Learnings. Es konzentriert sich auf die anspruchsvolle Aufgabe, optimales Verhalten in komplexen Umgebungen zu erlernen. Der Lernvorgang wird ausschließlich durch den Wert einer Belohnung und durch Beobachtung der Umgebung gesteuert. Das Modell ist sehr allgemein und auf viele Situationen anwendbar, von einfachen Spielen bis hin zur Optimierung komplexer Fertigungsprozesse.

Aufgrund der Flexibilität und der allgemeinen Anwendbarkeit entwickelt sich das Fachgebiet Reinforcement Learning sehr schnell weiter und weckt großes Interesse bei Forschern, die vorhandene Methoden verbessern oder neue Methoden entwickeln wollen, und bei Praktikern, die ihre Aufgaben möglichst effizient bewältigen möchten.

Motivation

Dieses Buch stellt den Versuch dar, dem Mangel an praxisnahen und strukturierten Informationen über Verfahren und Ansätze des Reinforcement Learnings (RL) entgegenzuwirken. Es gibt weltweit umfassende Forschungsaktivitäten, und fast täglich werden neue Artikel veröffentlicht. Große Teile von Deep-Learning-Konferenzen wie NeurIPS (Neural Information Processing Systems) oder ICLR (International Conference on Learning Representations) widmen sich RL-Verfahren. Es gibt mehrere große Forschungsgruppen, die sich auf die Anwendung von RL-Verfahren in der Robotik, in der Medizin und auf Multiagenten-Systemen konzentrieren.

Die Informationen über die jüngsten Forschungsergebnisse sind zwar allgemein verfügbar, sie sind aber zu spezialisiert und zu abstrakt, um sie ohne erhebliche Anstrengung zu verstehen. Bei den praktischen Aspekten von RL-Anwendungen ist die Situation noch schlimmer, weil oft nicht klar ist, wie man von der abstrakten mathematischen Beschreibung einer Methode in einem Forschungsartikel zu einer funktionierenden Implementierung gelangt, die eine Aufgabe tatsächlich löst.

Das erschwert es am Fachgebiet Interessierten, die Methoden und Konzepte, die in Artikeln oder Konferenzvorträgen vorgestellt werden, unmittelbar zu verstehen. Es gibt ganz ausgezeichnete Blogbeiträge über verschiedene RL-Methoden, die durch funktionierende Beispiele veranschaulicht werden, aber das eingeschränkte Format eines Blogbeitrags ermöglicht es dem Autor nicht, mehr als ein oder zwei Methoden zu beschreiben, ohne den vollständigen Kontext darzustellen und zu zeigen, in welcher Beziehung die verschiedenen Methoden zueinanderstehen. Dieses Buch ist mein Versuch, dieses Problem in Angriff zu nehmen.

Der Ansatz

Ein weiterer Aspekt des Buchs ist die Praxisorientierung. Alle Methoden werden in verschiedenen Umgebungen implementiert, die von völlig trivial bis zu ziemlich komplex reichen. Ich habe versucht, die Beispiele so zu gestalten, dass sie leicht verständlich sind, was durch die Leistungsfähigkeit von PyTorch ermöglicht wurde. Die Komplexität und die Anforderungen der Beispiele orientieren sich an RL-Interessierten, die keinen Zugang zu sehr großer Rechenleistung haben, wie einem GPU-Cluster oder sehr leistungsstarken Workstations. Dadurch wird, wie ich hoffe, das hochinteressante und spannende Fachgebiet RL einem breiteren Publikum zugänglich, nicht nur Forschungsgruppen oder großen KI-Unternehmen. Allerdings geht es nach wie vor um **Deep** Reinforcement Learning, deshalb empfiehlt sich die Verwendung einer GPU. Etwa die Hälfte der Beispiele im Buch profitiert davon, wenn sie auf einer GPU ausgeführt werden.

Beim Reinforcement Learning kommen oft Umgebungen mittlerer Größe zum Einsatz, beispielsweise bei Atari-Spielen oder für kontinuierliche Steuerungsaufgaben, das Buch enthält aber auch einige Kapitel (Kapitel 10, 14, 15, 16 und 18), in denen größere Projekte beschrieben werden, um zu veranschaulichen, wie sich RL-Verfahren auf kompliziertere Umgebungen und Aufgaben anwenden lassen. Diese Beispiele sind allerdings auch keine vollständigen Projekte aus der Praxis (sonst würden sie ein eigenes Buch erfordern), sondern etwas umfassendere Aufgaben, die veranschaulichen, wie sich das RL-Paradigma auf Bereiche jenseits der üblichen Benchmarks anwenden lässt.

Bei den Beispielen in den ersten drei Teilen des Buchs habe ich versucht, den vollständigen Quellcode zu zeigen, damit die Beispiele eigenständig sind. In einigen Fällen führt das dazu, dass Teile des Codes wiederholt werden (beispielsweise sind die Trainings-Schleifen der meisten Verfahren sehr ähnlich), aber ich denke, direkt zu einer Methode von Interesse springen zu können, ist wichtiger, als einige Wiederholungen zu vermeiden. Alle Beispiele im Buch sind auf Github verfügbar (<https://github.com/PacktPublishing/Deep-Reinforcement-Learning-Hands-On-Second-Edition>) und Sie sind herzlich eingeladen, mit dem Code zu experimentieren und eigene Beiträge zu leisten.

Für wen ist das Buch gedacht?

Das Buch richtet sich vornehmlich an alle, die schon über einige Vorkenntnisse im Bereich Machine Learning verfügen und daran interessiert sind, Reinforcement Learning in der Praxis kennenzulernen. Der Leser sollte mit Python und den Grundlagen von Deep Learning und Machine Learning vertraut sein. Kenntnisse der Statistik und Wahrscheinlichkeitsrechnung sind von Vorteil, aber nicht unbedingt erforderlich, um den Großteil des Buchs zu verstehen.

Zum Inhalt des Buchs

Kapitel 1, Was ist Reinforcement Learning?, stellt eine Einführung in die grundlegenden Ideen des Reinforcement Learnings und der wichtigsten formalen Modelle dar.

Kapitel 2, OpenAI Gym, führt Sie anhand der Open-Source-Bibliothek Gym in die praxisnahen Aspekte des RL ein.

Kapitel 3, Deep Learning mit PyTorch, gibt einen Überblick über die PyTorch-Bibliothek.

Kapitel 4, Das Kreuzentropie-Verfahren, stellt eines der einfachsten RL-Verfahren vor, um Ihnen einen Eindruck von RL-Verfahren und RL-Aufgaben zu vermitteln.

Kapitel 5, Tabular Learning und das Bellman'sche Optimalitätsprinzip, führt in die wertebasierten RL-Verfahren ein.

Kapitel 6, Deep Q-Networks, beschreibt DQNs, die Erweiterung elementarer wertebasierter Verfahren, die es ermöglichen, Lösungen für komplexere Umgebungen zu finden.

Kapitel 7, Allgemeine RL-Bibliotheken, stellt die PTAN-Bibliothek vor, die wir im Buch verwenden werden, um die Implementierung von RL-Verfahren zu erleichtern.

Kapitel 8, DQN-Erweiterungen, gibt einen detaillierten Überblick über moderne Erweiterungen von DQNs, die zur Verbesserung der Stabilität und der Konvergenz in komplexen Umgebungen dienen.

Kapitel 9, Beschleunigung von RL-Verfahren, bietet eine Übersicht über die Möglichkeiten, die Ausführung von RL-Code zu beschleunigen.

Kapitel 10, Aktienhandel per Reinforcement Learning, erläutert das erste konkrete Projekt, die Anwendung des DQN-Verfahrens auf den Aktienhandel.

Kapitel 11, Eine Alternative: Policy Gradients, stellt eine weitere Familie wertebasierter RL-Verfahren vor, die auf Policy Learning beruhen.

Kapitel 12, Das Actor-Critic-Verfahren, beschreibt eines der am häufigsten verwendeten RL-Verfahren.

Kapitel 13, Asynchronous Advantage Actor Critic, erweitert das Actor-Critic-Verfahren durch parallele Kommunikation mit der Umgebung, um Stabilität und Konvergenz zu verbessern.

Kapitel 14, Chatbot-Training per Reinforcement Learning, beschreibt das zweite konkrete Projekt und zeigt, wie RL-Verfahren auf NLP-Aufgaben angewendet werden.

Kapitel 15, Die TextWorld-Umgebung, stellt die Anwendung von RL-Verfahren auf Spiele des Genres *Interactive Fiction* (IF) vor.

Kapitel 16, Navigation im Web, beschreibt ein weiteres größeres Projekt, nämlich die Anwendung von RL auf die Navigation im Web anhand von MiniWoB-Aufgaben.

Kapitel 17, Stetige Aktionsräume, erläutert die Eigenheiten von Umgebungen, die stetige Aktionsräume verwenden, und stellt weitere Verfahren vor.

Kapitel 18, RL in der Robotik, befasst sich mit der Anwendung von RL-Verfahren auf Aufgaben aus dem Gebiet der Robotik. In diesem Kapitel beschreibe ich die Entwicklung und das Training eines kleinen Hardware-Roboters mithilfe von RL-Verfahren.

Kapitel 19, Trust Regions – PPO, TRPO, ACKTR und SAC, ist ein weiteres Kapitel über stetige Aktionsräume und beschreibt die Trust-Region-Verfahren.

Kapitel 20, Blackbox-Optimierung beim Reinforcement Learning, beschreibt Optimierungsverfahren, die keine Gradienten in expliziter Form verwenden.

Kapitel 21, Fortgeschrittene Exploration, erörtert verschiedene Ansätze zur besseren Erkundung der Umgebung.

Kapitel 22, Jenseits modellfreier Verfahren – Imagination, stellt unter Berücksichtigung jüngster Forschungsergebnisse modellbasierte Ansätze für RL vor.

Kapitel 23, AlphaGo Zero, erläutert die Anwendung des AlphaGo-Zero-Verfahrens auf das Spiel »Vier gewinnt«.

Kapitel 24, RL und diskrete Optimierung, beschreibt die Anwendung von RL-Verfahren auf diskrete Optimierungen anhand einer Zauberwürfel-Umgebung (Rubik's Cube).

Kapitel 25, RL mit mehreren Agenten, stellt eine relative neue Entwicklungsrichtung bei RL-Verfahren für Situationen vor, in denen mehrere Agenten vorhanden sind.

Verwendung des Buchs

Alle Kapitel des Buchs, die RL-Verfahren beschreiben, sind identisch aufgebaut: Zunächst werden die Motivation für das Verfahren, die theoretischen Grundlagen und die zugrunde liegenden Ideen erläutert. Anschließend betrachten wir verschiedene Anwendungen des Verfahrens auf unterschiedliche Umgebungen anhand des vollständigen Quellcodes.

Sie können das Buch also auf verschiedene Weise verwenden:

1. Um sich einen schnellen Überblick über ein Verfahren zu verschaffen, können Sie den einführenden Teil des entsprechenden Kapitels lesen.
2. Um ein besseres Verständnis der Implementierung eines Verfahrens zu erlangen, können Sie sich den Quellcode ansehen und die dazugehörigen Kommentare lesen.
3. Um ein tiefer gehendes Verständnis eines Verfahrens zu erlangen, sollten Sie versuchen, es selbst zu implementieren und zum Laufen zu bringen (meiner Ansicht nach die beste Lernmethode). Dabei können Sie den vorhandenen Quellcode als Ausgangspunkt nutzen.

Ich kann mir nur wünschen, dass Ihnen das Buch von Nutzen sein wird!

Codebeispiele herunterladen

Die Codebeispiele aus diesem Buch können Sie unter <http://www.mitp.de/0036> herunterladen.

Farbige Abbildungen herunterladen

Eine farbige Version der Screenshots und Diagramme in diesem Buch finden Sie ebenfalls unter <http://www.mitp.de/0036> zum Download.

Konventionen im Buch

In diesem Buch werden verschiedene Textformatierungen verwendet, um zwischen Informationen unterschiedlicher Art zu unterscheiden. Nachstehend finden Sie einige Beispiele und deren Bedeutungen.

Schlüsselwörter, Datenbanktabellen-, Twitter-, Datei-, Ordner-, Datei- und Pfadnamen sowie URLs und Usereingaben werden im Fließtext in nicht proportionaler Schrift darge-

stellt. Zum Beispiel: »Mounten Sie die heruntergeladene Datei `WebStorm-10*.dmg` als weiteres Laufwerk Ihres Systems.«

Codeblöcke werden wie folgt dargestellt:

```
def grads_func(proc_name, net, device, train_queue):
    envs = [make_env() for _ in range(NUM_ENVS)]

    agent = ptan.agent.PolicyAgent(
        lambda x: net(x)[0], device=device,
        apply_softmax=True)

    exp_source = ptan.experience.ExperienceSourceFirstLast(
        envs, agent, gamma=GAMMA, steps_count=REWARD_STEPS)

    batch = []
    frame_idx = 0
    writer = SummaryWriter(comment=proc_name)
```

Eingaben oder Ausgaben auf der Kommandozeile sehen so aus:

```
r1_book_samples/Chapter11$ ./02_a3c_grad.py --cuda -n final
```

Neue Ausdrücke und **wichtige Begriffe** werden **fett** gedruckt. Auf dem Bildschirm auswählbare oder anklickbare Bezeichnungen, wie z. B. Menüpunkte oder Schaltflächen, werden in der Schriftart **Kapitälchen** gedruckt: »Nach einem Klick auf die Schaltfläche **ABBRECHEN** in der unteren rechten Ecke wird der Vorgang abgebrochen.«

Was ist Reinforcement Learning?

Reinforcement Learning (RL) ist ein Teilgebiet des Machine Learnings, das sich damit beschäftigt, zu lernen, mit der Zeit automatisch optimale Entscheidungen zu treffen. Dabei handelt es sich um ein allgemeines und gängiges Problem, das in vielen wissenschaftlichen und technischen Fachgebieten untersucht wird.

In unserer sich wandelnden Welt sind auch Aufgaben, die auf den ersten Blick wie statische Eingabe-Ausgabe-Aufgaben aussehen, aus übergeordneter Sicht dynamisch. Betrachten Sie beispielsweise die einfache überwachte Lernaufgabe, Bilder von Haustieren entweder als Hund oder als Katze zu klassifizieren. Sie haben die Trainingsdatenmenge zusammengestellt und den Klassifizierer mit dem Deep-Learning-Toolkit Ihrer Wahl implementiert, und nach einiger Zeit liefert das konvergierende Modell ausgezeichnete Ergebnisse. Gut so? Aber sicher! Sie haben das Modell zur Anwendung gebracht und lassen es weiterlaufen. Dann fahren Sie in den Urlaub, und nach Ihrer Rückkehr stellen Sie fest, dass modische Hundehaarschnitte inzwischen völlig anders aussehen und dass ein beträchtlicher Teil Ihrer Bilder fehlerhaft klassifiziert wird. Sie müssen also Ihre Trainingsbilder aktualisieren und den ganzen Vorgang wiederholen. Gut so? Natürlich nicht!

Dieses Beispiel soll zeigen, dass auch ganz einfache Aufgaben beim **Machine Learning (ML)** eine verborgene zeitliche Dimension besitzen, die häufig übersehen wird, bei einem Produktivsystem aber zu einem Problem werden kann. **Reinforcement Learning (RL)** ist ein Ansatz, der diese zusätzliche Dimension (für gewöhnlich die Zeit, das muss aber nicht so sein) in den Gleichungen der Lernregeln berücksichtigt und damit der menschlichen Vorstellung von einer künstlichen Intelligenz schon deutlich näher kommt.

In diesem Kapitel geht es um die folgenden Themen:

- Was RL mit anderen ML-Verfahren, nämlich überwachtem und unüberwachtem Lernen, gemeinsam hat und wodurch es sich von ihnen unterscheidet
- Die wichtigsten RL-Formalismen und in welcher Beziehung sie zueinander stehen
- Die theoretischen Grundlagen des RL: der Markov-Entscheidungsprozess

1.1 Überwachtes Lernen

Überwachtes Lernen dürfte Ihnen bekannt sein, denn es ist die am häufigsten untersuchte und am besten bekannte Machine-Learning-Aufgabe. Die grundlegende Frage lautet: Wie kann man automatisch eine Funktion finden, die einer Eingabe anhand einer Menge von Beispielpaaren eine bestimmte Ausgabe zuordnet? So formuliert klingt die Aufgabe einfach, aber sie ist mit vielen kniffligen Problemen verbunden, die Computer erst in jüngster Zeit einigermaßen erfolgreich lösen konnten. Es gibt eine Vielzahl von Beispielen für überwachte Lernaufgaben, wie beispielsweise diese:

- **Textklassifikation:** Handelt es sich bei einer E-Mail um Spam oder nicht?
- **Bildklassifikation und Objektllokalisierung:** Zeigt ein Bild eine Katze, einen Hund oder etwas anderes?
- **Regressionen:** Wie wird den Messwerten der Wetterstation zufolge das morgige Wetter?
- **Stimmungsanalyse:** Wie zufrieden ist ein Kunde, der eine Rezension geschrieben hat?

Die Fragestellungen können sich unterscheiden, ihnen liegt jedoch die gleiche Idee zugrunde: Es liegen viele Beispiele für Eingaben und der dazugehörigen Ausgaben vor und wir möchten erlernen, die Ausgabe für neue, noch unbekannte Eingaben zu erzeugen. Die Bezeichnung *überwachtes* Lernen ist der Tatsache geschuldet, dass das System anhand bekannter Antworten lernt, denn die Beispiele sind korrekt mit Labels gekennzeichnet.

1.2 Unüberwachtes Lernen

Auf der anderen Seite gibt es das unüberwachte Lernen, bei dem die Daten nicht mit einem Label versehen sind. Das Ziel ist es, in den vorliegenden Daten eine verborgene Struktur aufzuspüren. Ein gängiges Beispiel für diesen Lernansatz ist das Clustering von Daten. Der Algorithmus versucht dabei, die Datenobjekte in Cluster aufzuteilen und auf diese Weise Beziehungen in den Daten aufzudecken. Man könnte beispielsweise nach ähnlichen Bildern suchen oder nach Kunden, die sich ähnlich verhalten.

Eine weitere unüberwachte Lernmethode, die sich zunehmender Beliebtheit erfreut, sind **Generative Adversarial Networks (GANs)**. Dabei gibt es zwei konkurrierende neuronale Netze. Das erste erzeugt gefälschte Daten, und das zweite versucht, zu unterscheiden, ob die Daten gefälscht wurden oder zur echten Datenmenge gehören. Im Lauf der Zeit können die beiden Netze ihre jeweilige Aufgabe immer besser erfüllen, indem sie subtile Muster in der Datenmenge erfassen.

1.3 Reinforcement Learning

RL ist irgendwo zwischen vollständig überwachtem Lernen und dem vollständigen Fehlen vordefinierter Labels einzuordnen. Einerseits verwendet es viele wohlbekannt Methoden des überwachten Lernens, wie tiefe neuronale Netze zur Funktionsapproximation, stochastischem Gradientenabstieg und Backpropagation, um Datenrepräsentationen zu erlernen. Andererseits werden diese Methoden dabei für gewöhnlich auf andere Art und Weise angewendet.

In den nächsten beiden Abschnitten dieses Kapitels haben wir die Gelegenheit, bestimmte Details des RL-Ansatzes zu erkunden, wie die Annahmen und Abstraktionen in mathematisch strenger Form. Fürs Erste verwenden wir eine weniger formale und anschaulichere Beschreibung, um RL mit überwachtem und unüberwachtem Lernen zu vergleichen.

Nehmen wir an, es gibt einen Agenten, der in einer Umgebung bestimmte Aktionen ausführen soll. Eine Robotermaus in einem Labyrinth ist ein gutes Beispiel, wir könnten uns aber auch einen automatischen Helikopter vorstellen, der versucht, ein Flugmanöver zu vollführen, oder ein Schachprogramm, das lernt, wie man einen Großmeister schlägt. Der Einfachheit halber bleiben wir bei der Robotermaus.

Ihre Umgebung ist ein Labyrinth, in dem es an einigen Stellen Futter gibt und an anderen Stromschläge. Die Robotermaus kann verschiedene Aktionen ausführen, wie etwa sich

nach links oder rechts zu drehen oder sich vorwärts zu bewegen. Sie kann zu jedem Zeitpunkt den vollständigen Zustand des Labyrinths beobachten, um zu entscheiden, welche Aktion sie ausführt (Abbildung 1.1). Sie versucht, so viel Futter wie möglich zu finden und Stromschläge möglichst zu vermeiden. Die Signale Futter und Stromschlag sind die *Belohnung*, die der Agent (die Robotermaus) von der Umgebung als zusätzliches Feedback zu den Aktionen erhält. Belohnung ist beim RL ein sehr wichtiges Konzept, auf das ich später in diesem Kapitel noch kommen werde. An dieser Stelle genügt es, zu wissen, dass es das Ziel des Agenten ist, eine möglichst große kumulierte Belohnung zu erhalten. In diesem speziellen Beispiel könnte die Maus einen Stromschlag in Kauf nehmen, um an eine Stelle zu gelangen, an der es viel Futter gibt. Das wäre ein besseres Ergebnis, als einfach nur stehen zu bleiben und gar kein Futter zu finden.

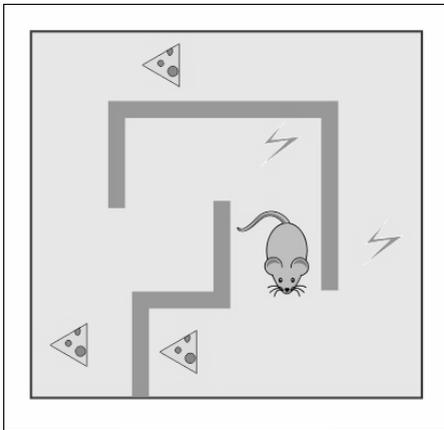


Abb. 1.1: Das Labyrinth der Robotermaus

Wir wollen das Wissen über die Umgebung und die jeweils beste Aktion in einer bestimmten Situation in der Robotermaus nicht fest einprogrammieren – das wäre zu aufwendig und würde nutzlos werden, wenn sich das Labyrinth auch nur leicht ändert. Wir benötigen vielmehr ein paar Methoden, die es dem Roboter ermöglichen, selbst zu erlernen, Stromschläge zu vermeiden und so viel Futter wie möglich zu finden.

Reinforcement Learning bietet hier eine Lösung, die sich von überwachten und unüberwachten Lernmethoden unterscheidet. Es gibt keine vordefinierten Labels wie beim überwachten Lernen. Niemand kennzeichnet die Bilder, die der Roboter sieht, mit *gut* oder *schlecht* oder gibt ihm Hinweise, in welche Richtung er sich am besten bewegen sollte.

Wir sind allerdings auch nicht völlig blind wie beim unüberwachten Lernen – es gibt ein Belohnungssystem. Belohnungen können positiv (beim Finden von Futter), negativ (bei einem Stromschlag) oder neutral sein (wenn nichts Besonderes geschieht). Indem er die Belohnungen beobachtet und in Beziehung zur ausgeführten Aktion setzt, kann unser Agent erlernen, wie er eine Aktion besser ausführen kann, mehr Futter findet und weniger Stromschläge erhält.

Natürlich hat die allgemeine Anwendbarkeit und die Flexibilität des Reinforcement Learnings ihren Preis. RL gilt als sehr viel schwieriger als überwacht und unüberwacht Lernen. Werfen wir einen kurzen Blick darauf, was Reinforcement Learning so knifflig macht.

1.4 Herausforderungen beim Reinforcement Learning

Zunächst einmal ist zu beachten, dass die Beobachtung beim RL vom Verhalten des Agenten abhängt und in gewissem Maße sogar das *Ergebnis* seines Verhaltens ist. Wenn Ihr Agent sich ineffizient verhält, sagt die Beobachtung nichts darüber aus, was er falsch gemacht hat und was man unternehmen sollte, um das Ergebnis zu verbessern (der Agent erhält die ganze Zeit nur negatives Feedback). Wenn der Agent stur ist und weiterhin Fehler begeht, kann die Beobachtung den falschen Eindruck vermitteln, dass es keine Möglichkeit gibt, eine größere Belohnung zu erhalten – das Leben ist ein Leidensweg –, was aber völlig falsch sein könnte.

Im Machine-Learning-Jargon spricht man davon, dass die **i.i.d.-Annahme** nicht erfüllt ist. Diese Annahme besagt, dass die Daten **unabhängig** voneinander (*independent*) und **identisch verteilt** (*identically distributed*) sind, was für die meisten überwachten Lernmethoden erforderlich ist.

Darüber hinaus wird das Leben unseres Agenten dadurch verkompliziert, dass er nicht nur die erlernte Policy **nutzen** (engl. *exploit*), sondern die Umgebung aktiv **erkunden** (engl. *explore*) muss, denn möglicherweise könnten wir ein erheblich besseres Ergebnis erzielen, wenn wir anders vorgehen. Das Problem ist nur, dass eine zu ausführliche Exploration zu einer beträchtlichen Verringerung der Belohnung führen könnte (ganz zu schweigen davon, dass der Agent auch *vergessen* kann, was er zuvor erlernt hat). Wir müssen also irgendwie ein Gleichgewicht zwischen diesen beiden Vorgehensweisen finden. Wie sich ein Ausweg aus diesem Dilemma zwischen Exploitation und Exploration finden lässt, ist eine der grundlegenden offenen Fragen beim RL.

Menschen müssen solche Entscheidungen ständig treffen: Soll ich zum Abendessen einen bekannten Ort aufsuchen oder doch das schicke neue Restaurant ausprobieren? Wie häufig sollte man den Arbeitsplatz wechseln? Sollte man sich mit einem neuen Fachgebiet befassen oder auf dem jetzigen weiterarbeiten? Eine allgemeingültige Antwort auf diese Fragen gibt es nicht.

Der dritte Faktor, der die Sache verkompliziert, ist die Tatsache, dass die Belohnung einer Aktion unter Umständen mit erheblicher Verzögerung erfolgt. Beim Schach kann ein einziger starker Zug in der Mitte der Partie spielentscheidend sein. Beim Lernen müssen wir so etwas erkennen, was schwierig sein kann, wenn wir mit dem Spielverlauf und den Aktionen beschäftigt sind.

Trotz all dieser Hindernisse und Komplikationen hat RL in den letzten Jahren große Fortschritte erzielt und wird immer mehr zu einem Fachgebiet für Forschung und praktische Anwendungen.

Neugierig geworden? Sehen wir uns die Details an und betrachten die RL-Formalismen und die geltenden Spielregeln.

1.5 RL-Formalismen

Bei jedem wissenschaftlichen Fachgebiet gibt es bestimmte Annahmen und Einschränkungen. Im letzten Abschnitt habe ich überwachtes Lernen erläutert, bei dem die Kenntnis der Eingabe-Ausgabe-Paare vorausgesetzt wird. Es gibt für die Daten keine Labels? Nichts für ungut, Sie müssen die Labels irgendwie beschaffen oder einen anderen Ansatz verfolgen. Überwachtes Lernen ist deswegen nicht besser oder schlechter als andere Verfahren, es ist

einfach nur für Ihre Aufgabe nicht einsetzbar. Es ist wichtig, diese *Spielregeln* für verschiedene Verfahren zu kennen, da Sie dadurch sehr viel Zeit sparen können. Allerdings gibt es auch viele Beispiele für praktische oder theoretische Durchbrüche, die dadurch erzielt wurden, dass die Spielregeln auf kreative Art und Weise missachtet wurden. Man sollte aber auf jeden Fall alle Einschränkungen kennen.

Es gibt eine Reihe solcher Formalismen für RL, und jetzt ist es an der Zeit, sie vorzustellen, denn wir werden sie im verbleibenden Teil des Buchs aus verschiedenen Perspektiven analysieren. In Abbildung 1.2 sind der **Agent**, die **Umgebung** und ihre Kommunikationskanäle dargestellt: **Aktionen**, **Belohnung** und **Beobachtungen**. Ich werde in den folgenden Abschnitten ausführlich darauf eingehen.

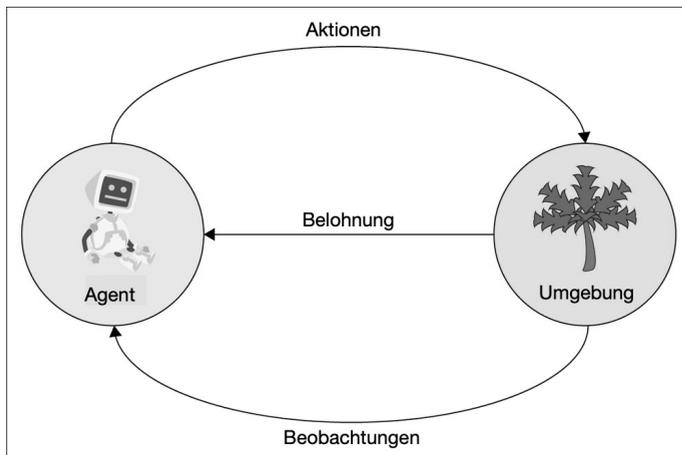


Abb. 1.2: Agent, Umgebung und Kommunikationskanäle

1.5.1 Belohnung

Als Erstes betrachten wir die Belohnung. Beim RL handelt es sich dabei um einen skalaren Wert, den wir regelmäßig von der Umgebung abfragen. Er kann positiv oder negativ sein, groß oder klein, es ist einfach nur eine Zahl. Zweck der Belohnung ist es, unserem Agenten mitzuteilen, wie gut er sich verhalten hat. Wir legen nicht fest, wie oft der Agent eine Belohnung erhält; es könnte sekundlich sein oder aber nur ein einziges Mal während seiner Lebensspanne. Es ist allerdings der Bequemlichkeit halber gängige Praxis, dass der Agent die Belohnung in festen Zeitabständen erhält oder nach jeder Interaktion mit der Umgebung. Bei einem Belohnungssystem, das nur am Ende der Lebensspanne eine Belohnung gewährt, sind alle Belohnungen bis auf die letzte Null.

Wie erwähnt, soll eine Belohnung dem Agenten Feedback über seinen Erfolg liefern; sie spielt beim RL eine wichtige und zentrale Rolle. Der Begriff *Reinforcement* (Verstärkung) wird verwendet, weil die von einem Agenten erhaltene Belohnung sein Verhalten verstärken soll (auf positive oder negative Weise). Die Belohnung ist stets *lokal*, das heißt, sie spiegelt den Erfolg der letzten Aktion des Agenten wider, nicht alle Erfolge in den zurückliegenden Aktionen. Für eine Aktion eine große Belohnung zu erhalten, bedeutet aber nicht, dass nicht eine Sekunde später drastische Folgen der vorangegangenen Entscheidungen eintreten können. Es verhält sich wie bei einem Banküberfall: Die Sache könnte wie eine gute Idee aussehen – bis man über die Konsequenzen nachdenkt.

Ein Agent versucht, durch die Abfolge seiner Aktionen die größte *kumulierte* Belohnung zu erzielen. Zwecks Veranschaulichung sind nachstehend einige konkrete Beispiele nebst Belohnung aufgeführt:

- **Finanzhandel:** Ein erzielter Gewinn ist eine Belohnung für einen Händler, der Aktien kauft und verkauft.
- **Schach:** Hier erhält man die Belohnung am Ende des Spiels in Form eines Siegs, einer Niederlage oder eines Remis. Das ist natürlich Interpretationssache. Für mich wäre ein Remis gegen einen Großmeister beispielsweise eine große Belohnung. In der Praxis müssen wir den genauen Wert der Belohnung ausdrücklich angeben, aber dabei kann es sich um einen ziemlich komplexen Ausdruck handeln. Beim Schach könnte die Belohnung beispielsweise proportional zur Spielstärke des Gegners sein.
- **Dopamin-System im Gehirn:** Ein Teil des Gehirns, das limbische System, produziert Dopamin, wenn es den anderen Teilen des Gehirns ein positives Signal senden muss. Eine höhere Dopamin-Konzentration verursacht ein Wohlgefühl, was Aktivitäten verstärkt, die das System als *gut* beurteilt. Leider ist das limbische System hinsichtlich der Dinge, die es als gut betrachtet, sehr altmodisch: Nahrung, Fortpflanzung und Dominanz, aber das ist eine völlig andere Geschichte.
- **Computerspiele:** Für gewöhnlich erhält der Spieler ein offensichtliches Feedback, etwa die Anzahl der abgeschossenen Gegner oder eine Punktezahl. Beachten Sie bei diesem Beispiel, dass diese Belohnung bereits kumuliert ist. Die Belohnung beim RL sollte bei Computerspielen daher von der Punktzahl abgeleitet werden, also +1, wenn ein weiterer Gegner abgeschossen wird, und bei allen anderen Zeitschritten 0.
- **Navigation im Web:** Es gibt bestimmte Aufgaben, die einen hohen praktischen Nutzen haben, nämlich im Web vorhandene Informationen automatisch zu extrahieren. Suchmaschinen versuchen im Allgemeinen, diese Aufgabe zu lösen, aber in manchen Fällen muss man, um an die gesuchten Daten zu gelangen, ein Formular ausfüllen, mehreren Links folgen oder Captchas lösen, was Suchmaschinen Schwierigkeiten bereitet. Es gibt einen RL-basierten Ansatz zur Lösung solcher Aufgaben, bei dem die Belohnung die gesuchte Information oder das benötigte Ergebnis ist.
- **Suche optimaler neuronaler Netzwerkarchitekturen:** RL wurde erfolgreich zur Optimierung neuronaler Netzwerkarchitekturen eingesetzt. Dabei ist das Ziel, die beste Leistungskennzahl bei einer Datenmenge zu erreichen. Zu diesem Zweck variiert man die Anzahl der Schichten, verändert die Parameter, richtet zusätzliche Verbindungen zwischen Schichten ein oder nimmt andere Änderungen an der Architektur des neuronalen Netzes vor. Die Belohnung ist in diesem Fall die Leistung (die Korrektklassifikationsrate oder ein anderes Maß, das zeigt, wie genau die Vorhersagen des neuronalen Netzes sind).
- **Hundedressur:** Wenn Sie schon einmal versucht haben, einen Hund zu dressieren, dann wissen Sie, dass man ihm jedes Mal etwas Leckeres geben muss (aber nicht zu viel), wenn er richtig gehorcht. Es ist auch üblich, ein Haustier ein wenig zu bestrafen (negative Belohnung), wenn es nicht gehorcht, allerdings haben neuere Studien gezeigt, dass diese Vorgehensweise nicht so effektiv ist wie eine positive Belohnung.
- **Schulnoten:** Damit haben wir alle Erfahrung! Schulnoten stellen ein Belohnungssystem dar, das Schülern Feedback zu ihrem Lernen gibt.

Wie die genannten Beispiele zeigen, ist eine Belohnung ein sehr allgemeiner Indikator für die Leistung des Agenten. Bei vielen alltäglichen Aufgaben gibt es Belohnungen oder man könnte Belohnungen verwenden.

1.5.2 Der Agent

Ein Agent interagiert mit der Umgebung, indem er bestimmte Aktionen ausführt, die Umgebung beobachtet und schließlich eine Belohnung erhält. In der Praxis ist der Agent in den meisten Fällen unsere Software, die eine Aufgabe auf mehr oder weniger effiziente Weise lösen soll. Bei den genannten Beispielen gibt es die folgenden Agenten:

- **Finanzhandel:** Ein Handelssystem oder ein Händler, der Entscheidungen über den An- und Verkauf von Aktien trifft
- **Schach:** Ein Spieler oder ein Computerprogramm
- **Dopamin-System im Gehirn:** Das Gehirn selbst, das Sensordaten zufolge entscheidet, ob die Erfahrung gut oder schlecht war
- **Computerspiele:** Der Spieler, der sich am Spiel erfreut oder das Computerprogramm (*Andrey Karpathy* schrieb in einem Tweet: »Wir wollten eigentlich, dass die KI die Arbeit erledigt und dass wir Spiele spielen, aber wir erledigen die ganze Arbeit und die KI spielt Spiele!«)
- **Navigation im Web:** Die Software, die dem Browser mitteilt, welcher Link angeklickt, wohin der Mauszeiger bewegt oder welcher Text eingegeben werden soll
- **Suche mit neuronalen Netzwerkarchitekturen:** Die Software zur Steuerung der konkreten Architektur des neuronalen Netzes, das beurteilt wird
- **Hundedressur:** Sie treffen die Entscheidung, ob der Hund belohnt oder bestraft wird, also sind Sie der Agent.
- **Schule:** Ein Schüler oder Student

1.5.3 Die Umgebung

Die Umgebung ist alles außerhalb des Agenten. Im weitesten Sinne ist das der Rest des Universums, aber das wäre hier übertrieben und wäre selbst mit zukünftigen Computern nicht zu bewerkstelligen, deshalb bleiben wir bei der üblichen Bedeutung.

Die Umgebung ist für einen Agenten extern und die Kommunikation mit der Umgebung ist auf Belohnungen (die von der Umgebung gewährt werden), Aktionen (die der Agent ausführt) und Beobachtungen (Informationen, die der Agent neben den Belohnungen von der Umgebung erhält) beschränkt. Belohnungen habe ich bereits erläutert, betrachten wir also Aktionen und Beobachtungen.

1.5.4 Aktionen

Aktionen sind Handlungen, die ein Agent in der Umgebung ausführen kann. Aktionen können nach den Spielregeln erlaubte Züge sein (bei einem Spiel) oder die Erledigung der Hausaufgaben (falls es um die Schule geht). Sie können einfach sein (»Ziehe den Bauern ein Feld vorwärts«) oder kompliziert (»Fülle das Steuerformular für morgen Vormittag aus«).

Beim RL unterscheiden wir zwei Arten von Aktionen: diskrete und stetige. Diskrete Aktionen bilden eine endliche Menge von sich gegenseitig ausschließenden Handlungen, die ein Agent ausführen könnte, etwa eine Bewegung nach links oder nach rechts. Zu stetigen Aktionen gehört ein Wert, der beispielsweise bei der Anweisung »Drehe das Steuer« angibt, um welchen Winkel das Steuer eines Autos gedreht werden soll. Verschiedene Winkel

würden dazu führen, dass die Situation eine Sekunde später eine andere ist, deshalb ist die Anweisung »Drehe das Steuer« nicht ausreichend.

1.5.5 Beobachtungen

Neben den Belohnungen sind Beobachtungen der Umgebung der zweite Informationskanal des Agenten. Nun fragen Sie sich vielleicht, weshalb eine eigene Datenquelle benötigt wird. Der Grund ist Bequemlichkeit. Beobachtungen sind Informationen, die dem Agenten von der Umgebung bereitgestellt werden, denen sich entnehmen lässt, was gerade vor sich geht.

Das könnte für die nachfolgende Belohnung von Bedeutung sein (etwa eine Benachrichtigung der Bank, die besagt, dass Geld eingegangen ist) oder auch nicht. Beobachtungen können in verschleierte Form Informationen über Belohnungen enthalten, wie etwa die Punktzahl, die bei einem Computerspiel auf dem Bildschirm angezeigt wird. Die Punktzahlen sind nur Pixel, aber wir könnten sie potenziell in Werte der Belohnung konvertieren; das ist mit modernen Deep-Learning-Verfahren nicht weiter schwierig.

Allerdings sollte die Belohnung nicht als zweitrangig oder unwichtig betrachtet werden, denn sie ist die treibende Kraft beim Lernprozess des Agenten. Wenn die Belohnung falsch, verrauscht oder nicht korrekt am eigentlichen Ziel ausgerichtet ist, kann das dazu führen, dass sich das Training in die falsche Richtung entwickelt.

Darüber hinaus ist es wichtig, zwischen dem Zustand einer Umgebung und Beobachtungen zu unterscheiden. Der Zustand einer Umgebung könnte potenziell sämtliche Atome des Universums umfassen, was es natürlich unmöglich macht, die gesamte Umgebung zu erfassen. Selbst wenn wir die Komplexität der Zustandsbeschreibung der Umgebung beschränken, sodass sie klein genug ist, ist es meistens trotzdem nicht möglich, vollständige Informationen zu erhalten, zudem können die Messungen verrauscht sein. Das ist allerdings unproblematisch, denn RL ist dafür ausgelegt, damit umzugehen. Betrachten wir ein weiteres Mal die Beispiele, um den Unterschied zu veranschaulichen:

- **Finanzhandel:** Hier ist die Umgebung der gesamte Finanzmarkt und alles, was ihn beeinflusst. Das ist eine wirklich lange Liste von Dingen: die neuesten Nachrichten, die wirtschaftlichen und politischen Bedingungen, das Wetter, die Nahrungsmittelversorgung oder Twitter-Trends. Selbst Ihre Entscheidung, heute zu Hause zu bleiben, kann das Finanzsystem potenziell indirekt beeinflussen (sofern Sie an den »Schmetterlingseffekt« glauben). Unsere Beobachtungen sind jedoch auf die Aktienkurse, die Nachrichten usw. beschränkt. Zum größten Teil des Zustands der Umgebung haben wir keinen Zugang, was den Aktienhandel so schwierig macht.
- **Schach:** Die Umgebung ist hier das Spielbrett *und* Ihr Gegner, was seine Schachkenntnisse, seine Stimmung, seinen Gehirnzustand, die ausgewählte Taktik usw. umfasst. Die Beobachtung ist das, was Sie sehen (die Stellung auf dem Schachbrett), aber ich nehme an, dass mit einiger Erfahrung auch psychologische Kenntnisse und die Fähigkeit, den Gegenspieler einzuschätzen, die Gewinnchancen verbessern können.
- **Dopamin-System:** Hier ist die Umgebung Ihr Gehirn *und* das Nervensystem *und* der Zustand der Organe *und* die gesamte Welt, die Sie wahrnehmen können. Beobachtungen sind die inneren Zustände des Gehirns und die Signale, die Ihre Sinne Ihnen übermitteln.
- **Computerspiel:** Hier ist die Umgebung der Zustand des Computers, inklusive der Daten im Arbeitsspeicher und auf der Festplatte. Bei vernetzten Spielen kommen die anderen Computer *und* die Internetinfrastruktur dazu, die sich zwischen den beteiligten

Computern befindet. Beobachtungen sind die Pixel auf dem Bildschirm und der Sound, das war's. Die Informationsmenge, die die Pixel auf dem Bildschirm enthalten, ist nicht gerade klein (jemand hat mal ausgerechnet, dass die Anzahl der möglichen Bilder auf einem Bildschirm mittlerer Größe (1024 × 768 Pixel) beträchtlich größer ist als die Anzahl der Atome in unserer Galaxie), aber die Anzahl der Zustände der gesamten Umgebung ist eindeutig noch größer.

- **Navigation im Web:** Hier ist die Umgebung das Internet inklusive der gesamten Netzwerkinfrastruktur, die sich zwischen unserem Agenten und dem Webserver befindet. Das ist ein wirklich riesiges System, das aus zig Millionen verschiedenen Komponenten besteht. Die Beobachtung ist normalerweise die aktuell geladene Webseite.
- **Suche neuronaler Netzwerkarchitekturen:** Bei diesem Beispiel ist die Umgebung ziemlich einfach und umfasst das NN-Toolkit, das ein bestimmtes neuronales Netz auswertet, sowie die Datenmenge, die verwendet wird, um die Leistungskennzahl zu ermitteln. Im Vergleich zum Internet erscheint diese Umgebung eher winzig.
Die Beobachtungen können unterschiedlich sein und Informationen über Tests oder andere Leistungskennzahlen umfassen, die bei der Bewertung ermittelt wurden.
- **Hundedressur:** Hier ist die Umgebung der Hund (inklusive seiner inneren Reaktion, seiner Stimmung und seiner Lebenserfahrung, die sich aber wohl kaum beobachten lassen) und alles, was sich in seiner Nähe befindet, inklusive anderer Hunde und einer Katze, die sich in einem Busch versteckt. Beobachtungen sind die Signale, die Ihre Sinne Ihnen übermitteln und Ihre Erinnerungen.
- **Schule:** Die Umgebung ist hier die Schule selbst, das Bildungssystem des Landes, die Gesellschaft und das kulturelle Erbe. Die Beobachtungen sind die gleichen wie bei der Hundedressur: Sinneswahrnehmung und Erinnerungen des Schülers.

Das ist unser Umfeld, mit dem wir im verbleibenden Buch herumexperimentieren werden. Ich nehme an, Sie haben bereits bemerkt, dass das RL-Modell extrem flexibel und allgemeingültig ist und auf eine Vielzahl von Szenarien angewendet werden kann. Bevor wir uns eingehender mit dem RL-Modell beschäftigen, werfen wir noch einen Blick darauf, in welcher Beziehung RL zu anderen Verfahren steht.

Es gibt viele weitere Bereiche, die zum RL beitragen oder mit ihm in Beziehung stehen. Die wichtigsten sind in Abbildung 1.3 dargestellt. Sie zeigt sechs große Bereiche, deren Verfahren und Themen sich stark bezüglich der Entscheidungsfindung überschneiden (grauer Kreis im Inneren). Die Schnittmenge dieser verwandten, aber doch unterschiedlichen wissenschaftlichen Fachgebiete bildet RL, das so allgemein und flexibel ist, dass es sich das Beste aus den verschiedenen Fachgebieten zunutze macht:

- **Machine Learning (ML):** RL ist ein Teilgebiet des ML und nutzt viele seiner Mechanismen, Tricks und Verfahren. RL hat im Wesentlichen zum Ziel, zu erlernen, wie ein Agent sich verhalten sollte, wenn ihm nur unvollständige Beobachtungsdaten bereitgestellt werden.
- **Engineering (insbesondere optimale Steuerung):** Ermöglicht es, eine Abfolge der optimalen Aktionen zu ermitteln, um das bestmögliche Ergebnis zu erzielen.
- **Neurowissenschaft:** Wir haben das Dopamin-System als Beispiel hierfür betrachtet. Es konnte gezeigt werden, dass die Funktionsweise des menschlichen Gehirns dem RL-Modell ziemlich nahe kommt.
- **Psychologie:** Hier wird das Verhalten unter verschiedenen Bedingungen untersucht, beispielsweise wie Menschen reagieren und sich anpassen, was einen engen Bezug zum RL aufweist.

- **Wirtschaft:** Zu den wichtigsten Themen gehört, wie sich die Belohnung bei unvollständigem Wissen und wechselnden Bedingungen in der realen Welt maximieren lässt.
- **Mathematik:** Wir verwenden idealisierte Systeme und widmen unsere Aufmerksamkeit im Operations Research dem Aufspüren der optimalen Bedingungen.

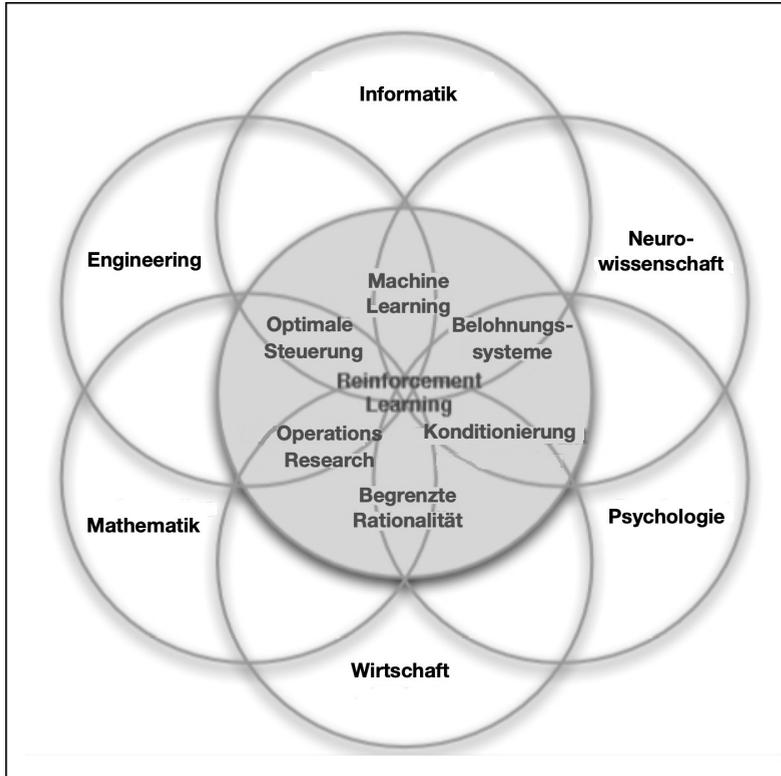


Abb. 1.3: Verschiedene Bereiche des Reinforcement Learnings

Im nächsten Abschnitt dieses Kapitels werden Sie sich mit den theoretischen Grundlagen des RL vertraut machen, die es ermöglichen, Verfahren zum Lösen einer RL-Aufgabe zu entwickeln. Dieser Abschnitt ist für das Verständnis des restlichen Buchs wichtig.

1.6 Die theoretischen Grundlagen des Reinforcement Learnings

In diesem Abschnitt betrachten wir die theoretischen Grundlagen des RL. Zunächst werden die mathematische Darstellung und die Notation der soeben erörterten Formalismen (Belohnung, Agent, Aktionen, Beobachtungen und Umgebung) vorgestellt. Darauf aufbauend betrachten wir die weiterführende RL-Fachsprache mit Begriffen wie *Zustand*, *Episode*, *Verlauf*, *Wert* und *Gewinn* (bzw. *Return*), die ich später im Buch immer wieder zur Beschreibung der verschiedenen Verfahren verwenden werde.

1.6.1 Markov-Entscheidungsprozesse

Die Beschreibung der Markov-Entscheidungsprozesse ähnelt einer russischen Matroschkapuppe: Wir betrachten zunächst den einfachsten Fall eines **Markov-Prozesses** (MP, auch Markov-Kette) und erweitern ihn mit Belohnungen, wodurch er zu einem Markov-Belohnungsprozess wird. Dann fügen wir Aktionen hinzu und verpacken das Ganze ein weiteres Mal, was uns zu einem **Markov-Entscheidungsprozess** (*Markov Decision Process*, MDP) führt.

Markov-Prozesse und Markov-Entscheidungsprozesse kommen in der Informatik und in anderen technischen Fachgebieten häufig zum Einsatz. Die Lektüre dieses Kapitels wird Ihnen also nicht nur im Zusammenhang mit RL von Nutzen sein, sondern auch bei vielen anderen Aufgabenstellungen.

Wenn Ihnen MDPs bereits vertraut sind, können Sie dieses Kapitel schnell überfliegen, sollten dabei aber den Definitionen der Terminologie Beachtung schenken, weil sie später noch verwendet wird.

1.6.2 Markov-Prozess

Wir betrachten zunächst den einfachsten Fall, nämlich einen Markov-Prozess (oder eine Markov-Kette). Stellen Sie sich vor, Sie haben ein System vor sich, das Sie nur beobachten können. Was Sie beobachten, sind die **Zustände** des Systems, die sich gemäß der dynamischen Regeln des Systems ändern. Sie können das System also nicht beeinflussen, sondern nur die Veränderungen beobachten.

Die Gesamtheit der möglichen Zustände eines Systems bildet eine Menge, die als *Zustandsraum* bezeichnet wird. Bei Markov-Prozessen nehmen wir an, dass diese Menge der Zustände endlich ist (sie kann jedoch extrem groß sein, um diese Einschränkung wettzumachen). Ihre Beobachtungen bilden eine Sequenz von Zuständen bzw. eine *Kette* (deshalb werden Markov-Prozesse auch als Markov-Ketten bezeichnet). Wenn wir beispielsweise das einfachste Modell für das Wetter an irgendeinem Ort betrachten, kann der heutige Tag entweder *sonnig* oder *regnerisch* sein – das ist unser Zustandsraum. Eine Folge von Beobachtungen im Laufe der Zeit bildet eine Kette von Zuständen, wie beispielsweise [*sonnig, sonnig, regnerisch, sonnig, ...*], und wird als **Verlauf** bezeichnet.

Damit ein solches System als Markov-Prozess bezeichnet werden kann, muss es die **Markov-Eigenschaft** besitzen. Das bedeutet, dass die zukünftigen Veränderungen an einem System mit einem bestimmten Zustand nur von diesem Zustand abhängen dürfen. Die Markov-Eigenschaft bewirkt, dass jeder beobachtbare Zustand die Zukunft des Systems beschreibt. Mit anderen Worten: Die Markov-Eigenschaft verlangt, dass die Zustände des Systems voneinander unterscheidbar und eindeutig sind. In diesem Fall ist nur ein Zustand erforderlich, um das zukünftige Verhalten des Systems zu modellieren, nicht der gesamte Verlauf oder die letzten N Zustände.

Bei unserem Wetterbeispiel beschränkt die Markov-Eigenschaft unser Modell darauf, nur die Fälle zu repräsentieren, in denen einem sonnigen Tag ein regnerischer immer mit der gleichen Wahrscheinlichkeit folgen kann, unabhängig davon, wie viele sonnige Tage es vorher gegeben hat. Das Modell ist nicht besonders realistisch, denn der gesunde Menschenverstand sagt uns, dass die morgige Regenwahrscheinlichkeit nicht nur von den aktuellen Bedingungen abhängt, sondern auch von vielen anderen Faktoren, wie Jahreszeit, Breitengrad oder die Nähe zu einem Gebirge oder zum Meer. Das Beispiel ist also wirklich naiv,

aber es ist wichtig, die Einschränkungen zu verstehen und bewusste Entscheidungen über sie zu treffen.

Wenn wir ein komplexeres Modell verwenden möchten, können wir das jederzeit tun, indem wir den Zustandsraum erweitern, was es ermöglicht, zu den Kosten eines größeren Zustandsraums mit dem Modell weitere Abhängigkeiten zu erfassen. Wenn Sie beispielsweise die Wahrscheinlichkeit für verregnete Tage im Sommer und im Winter getrennt erfassen möchten, können Sie dem Zustand die Jahreszeit hinzufügen. In diesem Fall ist Ihr Zustandsraum [sonnig+Sommer, sonnig+ Winter, regnerisch+ Sommer, regnerisch+ Winter].

Wenn Ihr System die Markov-Eigenschaft besitzt, können sie Übergangswahrscheinlichkeiten mit einer **Übergangsmatrix** erfassen. Dabei handelt es sich um eine quadratische Matrix der Größe $N \times N$, wobei N die Anzahl der Zustände des Modells angibt. Die Zelle in der Zeile i und der Spalte j der Matrix gibt die Wahrscheinlichkeit dafür an, dass das System vom Zustand i in den Zustand j übergeht.

Bei unserem Wetterbeispiel könnte die Übergangsmatrix beispielsweise so aussehen:

	sonnig	regnerisch
sonnig	0,8	0,2
regnerisch	0,1	0,9

An einem sonnigen Tag beträgt die Wahrscheinlichkeit also 80 Prozent, dass der nächste Tag sonnig ist, und 20 Prozent, dass er regnerisch ist. Wenn wir einen regnerischen Tag beobachten, beträgt die Wahrscheinlichkeit, dass sich das Wetter bessert, 10 Prozent und 90 Prozent, dass der nächste Tag verregnet ist.

Das ist schon alles. Die formale Definition eines Markov-Prozesses lautet:

- Es gibt eine Menge S von Zuständen, die das System annehmen kann.
- Eine Übergangsmatrix T mit Übergangswahrscheinlichkeiten legt das Verhalten des Systems fest.

Ein Graph mit Knoten, die den Zuständen des Systems entsprechen und Kanten, die mit den Übergangswahrscheinlichkeiten gekennzeichnet sind, ist eine nützliche visuelle Repräsentation eines Markov-Prozesses. Wenn die Übergangswahrscheinlichkeit 0 ist, wird die Kante nicht gezeichnet, weil es keine Möglichkeit gibt, von dem einen Zustand zum anderen zu gelangen. Diese Art der Darstellung wird auch zur Repräsentation endlicher Automaten verwendet, die Gegenstand der Automatentheorie sind. Abbildung 1.4 zeigt den Graphen für unser Wettermodell.

Wir sprechen hier wieder nur von Beobachtungen. Wir haben keine Möglichkeit, das Wetter zu beeinflussen, deshalb beobachten wir nur und zeichnen unsere Beobachtungen auf.

Als etwas komplizierteres Beispiel betrachten wir ein Modell für einen Büroangestellten (Dilbert, die Hauptfigur in Scott Adams berühmten Cartoons, ist ein gutes Beispiel). Sein Zustandsraum sieht folgendermaßen aus:

- **Zuhause:** Er ist nicht im Büro.
- **Computer:** Er arbeitet im Büro an seinem Computer.
- **Kaffee:** Er trinkt im Büro einen Kaffee.
- **Chat:** Er unterhält sich im Büro mit seinen Kollegen.

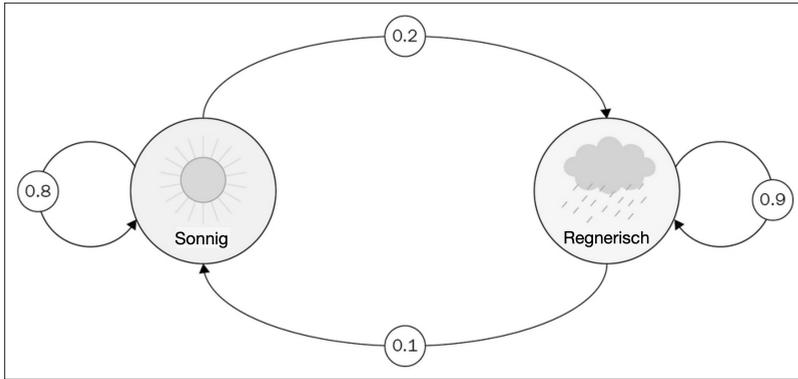


Abb. 1.4: Modell für sonniges oder regnerisches Wetter

Abbildung 1.5 zeigt den Graphen der Zustandsübergänge.

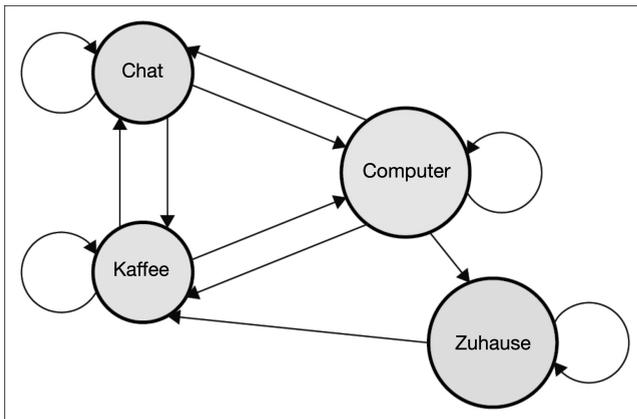


Abb. 1.5: Graph der Zustandsübergänge

Wir nehmen an, dass der Arbeitstag für gewöhnlich mit dem Zustand **Zuhause** beginnt und dass er zunächst immer einen **Kaffee** trinkt – und zwar ausnahmslos, deshalb gibt es die Kanten **Zuhause** → **Computer** bzw. **Zuhause** → **Chat** nicht. Das Diagramm zeigt außerdem, dass der Arbeitstag immer ausgehend vom Zustand **Computer** endet, also wieder der Zustand **Zuhause** erreicht wird. Die Übergangsmatrix für das Diagramm in Abbildung 1.5 sieht folgendermaßen aus:

	Zuhause	Kaffee	Chat	Computer
Zuhause	60%	40%	0%	0%
Kaffee	0%	10%	70%	20%
Chat	0%	20%	50%	30%
Computer	20%	20%	10%	50%

Die Übergangswahrscheinlichkeiten können auch wie in Abbildung 1.6 direkt im Graphen der Zustandsübergänge angegeben werden.

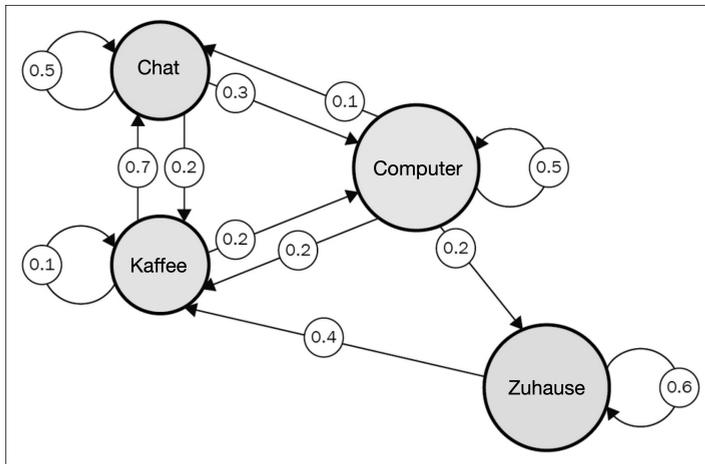


Abb. 1.6: Graph der Zustandsübergänge mit Übergangswahrscheinlichkeiten

In der Praxis haben wir nur selten das Glück, die Übergangsmatrix genau zu kennen. Realistischer ist es, dass uns nur Beobachtungen der Zustände des Systems zur Verfügung stehen, die auch als *Episoden* bezeichnet werden:

- Zuhause → Kaffee → Kaffee → Chat → Kaffee → Computer → Zuhause
- Computer → Computer → Chat → Chat → Kaffee → Computer → Computer → Computer
- Zuhause → Zuhause → Kaffee → Chat → Computer → Kaffee → Kaffee

Es ist nicht weiter kompliziert, die Übergangsmatrix anhand unserer Beobachtung abzuschätzen. Wir zählen einfach die Übergänge aller Zustände und normieren sie, sodass sie sich zu 1 summieren. Je mehr Beobachtungsdaten vorliegen, desto genauer wird unsere Schätzung dem tatsächlichen Modell entsprechen.

An dieser Stelle ist noch erwähnenswert, dass die Markov-Eigenschaft impliziert, dass die Markov-Prozesse stationär sind. Das heißt, dass sich die Wahrscheinlichkeitsverteilung, die den Zustandsübergängen zugrunde liegt, im Lauf der Zeit nicht ändert. Wäre sie nicht stationär, würde das bedeuten, dass es einen verborgenen Faktor gibt, der das Verhalten unseres Systems beeinflusst, und dass dieser Faktor in den Beobachtungen nicht enthalten ist. Das allerdings steht im Widerspruch zur Markov-Eigenschaft, die verlangt, dass die zugrunde liegende Wahrscheinlichkeitsverteilung für einen Zustand die gleiche ist, unabhängig vom Verlauf der Übergänge. Es ist wichtig, den Unterschied zwischen den in einer Episode tatsächlich beobachteten Übergängen und der zugrunde liegenden Verteilung zu verstehen, die durch die Übergangsmatrix gegeben ist. Konkrete Episoden, die wir beobachten, sind zufällige Stichproben der Verteilung des Modells, deshalb können sie sich von Episode zu Episode unterscheiden. Die Wahrscheinlichkeit konkreter Übergänge bleibt unverändert. Ist das nicht der Fall, ist der Formalismus der Markov-Prozesse nicht anwendbar.

Jetzt können wir fortfahren und das Modell der Markov-Prozesse erweitern, um der Lösung unserer RL-Aufgabe näher zu kommen. Wir fügen jetzt Belohnungen hinzu!

1.6.3 Markov-Belohnungsprozess

Zwecks Einführung von Belohnungen müssen wir unser Modell der Markov-Prozesse ein wenig erweitern. Zunächst einmal müssen wir einem Übergang von einem Zustand zum anderen einen Belohnungswert zuweisen. Es gibt schon Wahrscheinlichkeiten, aber die Wahrscheinlichkeiten werden dazu verwendet, das dynamische Verhalten unseres Systems zu erfassen, und auf diese Weise steht uns ein zusätzlicher skalarer Wert zur Verfügung.

Belohnungen können auf verschiedene Weise repräsentiert werden. Die gängigste Methode ist eine weitere quadratische Matrix, die der Übergangsmatrix ähnelt, aber in Zeile i und Spalte j Belohnungen für den Übergang von Zustand i nach Zustand j enthält. Eine Belohnung kann positiv oder negativ sein oder groß bzw. klein – es ist einfach nur eine Zahl. In manchen Fällen ist diese Repräsentation redundant und kann vereinfacht werden. Wenn die Belohnung beispielsweise für das Erreichen eines Zustands unabhängig vom vorhergehenden Zustand vergeben wird, brauchen wir nur »Zustand → Belohnung«-Paare zu speichern, was eine kompaktere Repräsentation darstellt. Das ist allerdings nur anwendbar, wenn der Wert der Belohnung ausschließlich vom Zielzustand abhängt, was nicht immer der Fall ist.

Außerdem fügen wir dem Modell einen Diskontierungsfaktor γ (gamma) hinzu, eine Zahl zwischen 0 und 1 (jeweils inklusive). Die Bedeutung wird später erklärt, nachdem wir die zusätzlichen Eigenschaften des Markov-Belohnungsprozesses definiert haben.

Wir beobachten also eine Kette von Zustandsübergängen bei einem Markov-Prozess. Das trifft auch bei einem Markov-Belohnungsprozess zu, aber bei jedem Übergang gibt es einen zusätzlichen Wert – die Belohnung. Allen Beobachtungen ist jetzt also ein zusätzlicher Wert als Belohnung für Zustandsübergänge des Systems zugeordnet.

Wir definieren den **Return** einer Episode zum Zeitpunkt t wie folgt:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Versuchen wir, zu verstehen, was das bedeutet. Zu jedem Zeitpunkt berechnen wir den **Return** als die Summe nachfolgender Belohnungen, aber zeitlich weiter entfernte Belohnungen werden mit dem zur k -ten Potenz erhobenen Diskontierungsfaktor multipliziert, wobei k die Anzahl der Zeitschritte angibt, die wir von Startzeitpunkt t entfernt sind. Der Diskontierungsfaktor steht für die Voraussicht eines Agenten. Wenn gamma gleich 1 ist, dann ist der Return G_t die Summe aller nachfolgenden Belohnungen. Das entspricht einem Agenten mit perfektem Weitblick, der alle nachfolgenden Belohnungen exakt kennt. Ist gamma gleich 0, dann ist der Return G_t nur die sofortige Belohnung ohne Berücksichtigung irgendwelcher nachfolgenden Zustände, was einem völlig kurzsichtigen Agenten entspricht.

Diese Extremwerte sind nur in Sonderfällen nützlich, und für gewöhnlich liegt der Wert von gamma irgendwo dazwischen, etwa bei 0,9 oder bei 0,99. In diesem Fall berücksichtigen wir zukünftige Belohnungen, die aber nicht in allzu ferner Zukunft liegen.

Der Parameter gamma ist beim RL von großer Bedeutung, und wir werden ihm in den nachfolgenden Kapiteln immer wieder begegnen. Sie können sich gamma als ein Maß dafür vorstellen, wie weit wir in die Zukunft blicken, um den zukünftigen Return abzuschätzen. Je näher der Wert an 1 liegt, desto mehr zukünftige Zeitschritte berücksichtigen wir.

Der **Return** erweist sich in der Praxis als nicht besonders nützlich, weil er für jede einzelne beobachtete Kette des Markov-Belohnungsprozesses definiert ist und deshalb selbst für den

gleichen Zustand stark schwanken kann. Wenn wir jedoch den mathematischen Erwartungswert des Returns für sämtliche Zustände berechnen (durch Mittelwertbildung sehr vieler Ketten), erhalten wir einen weitaus nützlicheren Wert, den **Zustandswert**:

$$V(s) = \mathbb{E}[G|S_t = s]$$

Die Interpretation ist einfach: Der Wert $V(s)$ gibt den durchschnittlichen (den zu erwartenden) Return für den Zustand s an, wenn wir den Markov-Belohnungsprozess ausführen.

Um zu demonstrieren, wie diese theoretischen Überlegungen in der Praxis aussehen, erweitern wir den Dilbert-Prozess um Belohnungen und machen ihn so zu einem **Dilbert-Belohnungsprozess (DBP)**. Die Belohnungen besitzen folgende Werte:

- Zuhause → Zuhause: 1 (weil es schön ist, zu Hause zu sein)
- Zuhause → Kaffee: 1
- Computer → Computer: 5 (harte Arbeit lohnt sich)
- Computer → Chat: -3 (Ablenkung ist nicht gut)
- Chat → Computer: 2
- Computer → Kaffee: 1
- Kaffee → Computer: 3
- Kaffee → Kaffee: 1
- Kaffee → Chat: 2
- Chat → Kaffee: 1
- Chat → Chat: -1 (eine lange Unterhaltung wird langweilig)

Abbildung 1.7 zeigt ein Diagramm mit Belohnungen.

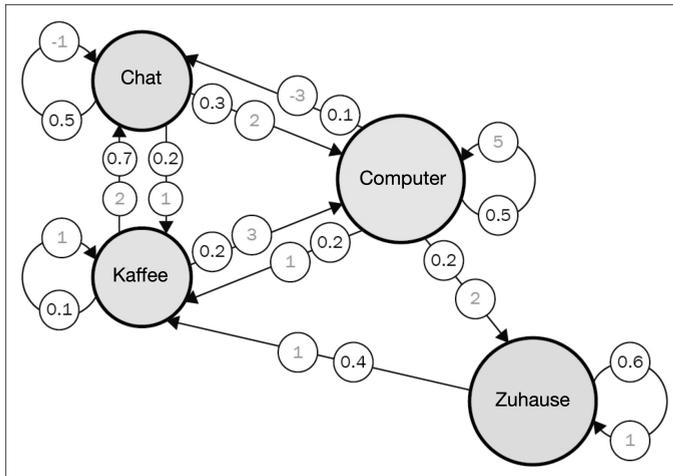


Abb. 1.7: Graph der Zustandsübergänge mit Übergangswahrscheinlichkeiten (dunkel) und Belohnungen (hell)

Kommen wir zurück zum Parameter gamma und betrachten wir die Zustandswerte bei verschiedenen Werten von gamma, zunächst für einen einfachen Fall: gamma = 1. Wie werden hier die Zustandswerte berechnet?

Zwecks Beantwortung dieser Frage betrachten wir den Zustand **Chat**. Was könnte der nächste Übergang sein? Die Antwort lautet: *Es hängt vom Zufall ab*. Laut Übergangsmatrix für den Dilbert-Prozess beträgt die Wahrscheinlichkeit 50 Prozent, dass der nächste Zustand erneut **Chat** ist. Die Wahrscheinlichkeit für **Kaffee** ist 20 Prozent und in 30 Prozent der Fälle ist der nächste Zustand **Computer**. Wenn $\gamma = 0$ ist, erhalten wir als Return lediglich den Wert des unmittelbar nachfolgenden Zustands. Wenn wir also den Zustandswert von **Chat** berechnen möchten, müssen wir die Werte aller Übergänge mit ihrer Wahrscheinlichkeit multiplizieren und sie summieren:

- $V(\text{Chat}) = -1 * 0,5 + 2 * 0,3 + 1 * 0,2 = 0,3$
- $V(\text{Kaffee}) = 2 * 0,7 + 1 * 0,1 + 3 * 0,2 = 2,1$
- $V(\text{Zuhause}) = 1 * 0,6 + 1 * 0,4 = 1,0$
- $V(\text{Computer}) = 5 * 0,5 + (-3) * 0,1 + 1 * 0,2 + 2 * 0,2 = 2,8$

Computer ist also der Zustand mit dem höchsten Wert (wir berücksichtigen nur die unmittelbar nachfolgende Belohnung), was nicht überrascht, denn **Computer** → **Computer** kommt häufig vor, bringt eine hohe Belohnung und die Wahrscheinlichkeit für den Übergang in andere Zustände ist nicht zu hoch.

Jetzt kommt eine kniffligere Frage: Wie groß ist der Zustandswert, wenn $\gamma = 1$ ist? Denken Sie sorgfältig darüber nach.

Die Antwort lautet: Der Wert ist für alle Zustände unendlich. Unser Diagramm enthält keine Senken (Zustände ohne ausgehende Übergänge), und wenn der Diskontierungsfaktor gleich 1 ist, berücksichtigen wir eine potenziell unendlich große Anzahl zukünftiger Übergänge. Wie wir im Fall von $\gamma = 0$ gesehen haben, sind alle kurzfristigen Werte positiv, und die Summe unendlich vieler positiver Werte ergibt einen unendlichen Wert, unabhängig vom Ausgangszustand.

Dieses unendliche Ergebnis ist einer der Gründe dafür, bei einem Markov-Belohnungsprozess γ einzuführen, anstatt einfach nur alle zukünftigen Belohnungen zu summieren. In den meisten Fällen kann es unendlich viele (oder zumindest sehr viele) Übergänge geben. Und da es ziemlich unpraktisch ist, unendliche Werte zu handhaben, möchten wir die Anzahl der Übergänge begrenzen, für die wir Werte berechnen. Ein Wert von γ , der kleiner als 1 ist, ermöglicht solch eine Begrenzung. Ich gehe in den Kapiteln über Iterationsverfahren ausführlicher darauf ein. Wenn Sie es allerdings mit einer endlichen Umgebung zu tun haben, beispielsweise beim Spiel Tic-Tac-Toe, das auf höchstens 9 Schritte beschränkt ist, können Sie problemlos $\gamma = 1$ verwenden. Ein weiteres Beispiel ist eine wichtige Klasse von Umgebungen mit nur einem Schritt, die *Multi-Armed Bandit MDP* heißt. Hier muss bei jedem Schritt eine Auswahl zwischen alternativen Aktionen getroffen werden. Die Aktion stellt Ihnen eine Belohnung bereit und beendet die Episode.

Bei der Definition des Markov-Belohnungsprozesses (MBP) hatte ich erwähnt, dass γ für gewöhnlich ein Wert zwischen 0 und 1 zugewiesen wird (0,9 und 0,99 sind gängige Werte). Mit diesen Werten wird es allerdings fast unmöglich, die Werte von Hand zu berechnen, selbst bei einem so kleinen MBP wie in unserem Dilbert-Beispiel, weil es erforderlich ist, Hunderte von Werten zu summieren. Computer sind gut für lästige Aufgaben wie das Summieren Tausender Werte geeignet, und es gibt einige einfache Methoden, die schnell die Werte eines MBP berechnen können, wenn man ihnen Übergangs- und Belohnungsmatrizen übergibt. In Kapitel 5, *Tabular Learning und das Bellman'sche Optimalitätsprinzip*, werden Sie solch eine Methode kennenlernen und sogar implementieren, wenn wir uns mit Q-Learning-Verfahren befassen.

Aber vorher fügen wir dem Markov-Belohnungsprozess eine weitere Ebene der Komplexität hinzu und ergänzen den noch fehlenden Teil: Aktionen.

1.6.4 Aktionen hinzufügen

Vielleicht haben Sie schon eine Vorstellung davon, wie man den Markov-Belohnungsprozess erweitert, sodass Aktionen verfügbar sind. Zunächst einmal fügen wir eine Menge von Aktionen (A) hinzu, die endlich sein muss. Dabei handelt es sich um den *Aktionsraum* unseres Agenten. Anschließend müssen wir unsere Übergangsmatrix mit Aktionen ausstatten, was bedeutet, dass die Matrix eine zusätzliche Aktions-Dimension benötigt, wodurch sie zu einem Würfel wird.

Bei MP und MBP sind die Übergangsmatrizen quadratisch. Dabei ist der Quellzustand in den Zeilen und der Zielzustand in den Spalten gespeichert. Jede Zeile i enthält eine Liste der Wahrscheinlichkeiten, in die anderen Zustände überzugehen (Abbildung 1.8).

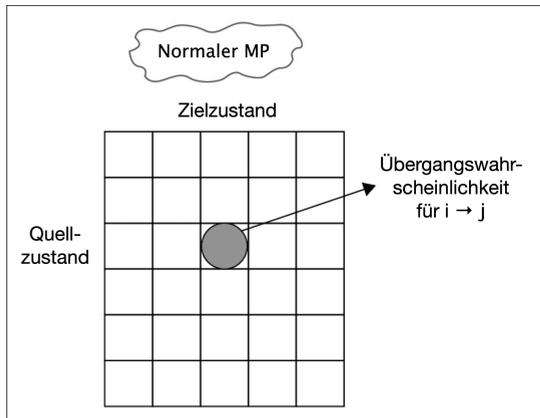


Abb. 1.8: Übergangsmatrix

Jetzt beobachtet der Agent die Zustandsübergänge nicht mehr passiv, sondern kann zu jedem Zeitpunkt aktiv eine Aktion auswählen. Für jeden Zustand gibt es also nicht mehr eine Zahlenliste, sondern eine Matrix, wobei die neue Dimension die Aktionen enthält, die der Agent ausführen kann. Die anderen Dimensionen enthalten den Quellzustand und den Zielzustand, in den das System übergeht, wenn der Agent eine Aktion ausführt. Abbildung 1.9 zeigt die neue Übergangsmatrix, die zu einem Würfel geworden ist, mit dem Quellzustand als Höhe (Index i), dem Zielzustand als Breite (Index j) und den Aktionen, die der Agent auswählen kann, als Tiefe (Index k).

Der Agent kann also im Allgemeinen durch die Auswahl einer Aktion die Wahrscheinlichkeiten der Zielzustände beeinflussen – eine nützliche Fähigkeit.

Stellen Sie sich Folgendes vor, um einen Eindruck davon zu bekommen, weshalb die Sache so kompliziert ist: Ein kleiner Roboter lebt auf einem 3×3 -Felder großem Gitter und kann die Aktionen *nach links drehen*, *nach rechts drehen* und *vorwärts bewegen* ausführen. Der Zustand dieser Welt wird durch die Position des Roboters und durch seine Blickrichtung (nach oben, nach unten, nach links, nach rechts) festgelegt. Es gibt also $3 \times 3 \times 4 = 36$ Zustände (der Roboter darf sich auf allen Feldern aufhalten und in alle Richtungen blicken).

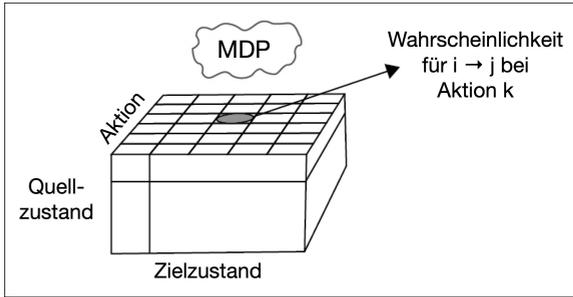


Abb. 1.9: Übergangswahrscheinlichkeiten beim MDP

Nehmen Sie des Weiteren an, dass die Motoren des Roboters nicht ganz in Ordnung sind (das ist in der Realität tatsächlich häufig der Fall). Wenn er eine der Aktionen *nach links drehen* oder *nach rechts drehen* ausführt, funktioniert das in 90 Prozent der Fälle, aber mit einer Wahrscheinlichkeit von 10 Prozent drehen die Räder durch und die Position des Roboters ändert sich nicht. Bei der Aktion *vorwärts bewegen* verhält es sich nicht anders: In 90 Prozent der Fälle funktioniert es, aber in 10 Prozent der Fälle verharrt der Roboter an seiner ursprünglichen Position.

In Abbildung 1.10 ist ein kleiner Teil eines Übergangsdiagramms dargestellt, das die möglichen Übergänge vom Zustand $(1, 1, \text{nach oben})$ zeigt. Der Roboter befindet sich also in der Mitte des Gitters und blickt nach oben. Wenn er versucht, sich vorwärts zu bewegen, beträgt die Wahrscheinlichkeit 90 Prozent, dass er sich anschließend im Zustand $(0, 1, \text{nach oben})$ befindet, aber mit einer Wahrscheinlichkeit von 10 Prozent drehen die Räder durch und er verbleibt an Position $(1, 1, \text{nach oben})$.

Damit all diese Details über die Umgebung und möglichen Reaktionen auf die Aktionen des Agenten richtig erfasst werden, verwendet ein MDP im Allgemeinen eine dreidimensionale Übergangsmatrix mit den Dimensionen (*Quellzustand, Aktion, Zielzustand*).

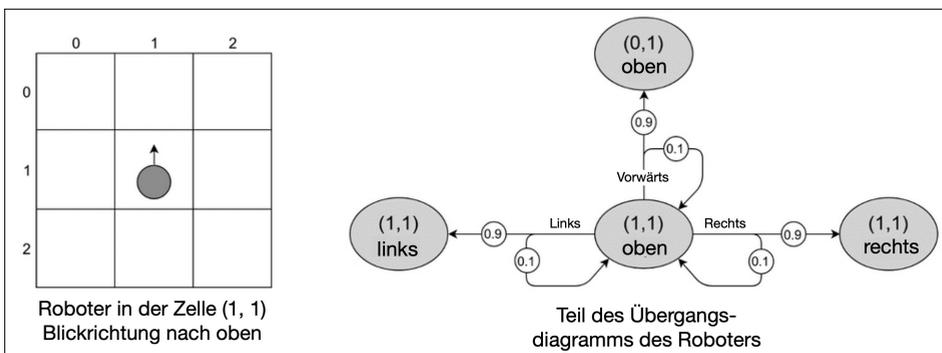


Abb. 1.10: Die Umgebung in der »Grid World«

Um aus unserem MBP ein MDP zu machen, müssen wir unserer Belohnungsmatrix auf die gleiche Weise wie bei der Übergangsmatrix Aktionen hinzufügen: Unsere Belohnungsmatrix hängt nicht nur vom Zustand ab, sondern auch von der Aktion. Mit anderen Worten:

Die Belohnung, die der Agent erhält, hängt jetzt nicht mehr nur von Endzustand ab, sondern auch von der Aktion, die zu diesem Zustand führt.

Es ist damit vergleichbar, dass man an Wissen und Erfahrung gewinnt, wenn man Mühe in eine bestimmte Sache investiert, selbst wenn man nicht allzu erfolgreich ist. Die Belohnung könnte also besser ausfallen, wenn Sie etwas unternehmen, anstatt untätig zu bleiben, selbst wenn das Endergebnis dasselbe ist.

Nun liegt ein formal definierter MDP vor und wir sind bereit, die für RL und MDP wichtigste Sache einzuführen: die **Policy**.

1.6.5 Policy

Die intuitive Definition einer Policy ist, dass es sich um eine Menge von Regeln handelt, die das Verhalten des Agenten steuern. Selbst in ziemlich einfachen Umgebungen kann es eine Vielzahl von Policies geben. Im letzten Beispiel mit dem Roboter in der »Grid World« könnte es beispielsweise verschiedene Policies für den Agenten geben, die zu unterschiedlichen Mengen der besuchten Zustände führen. Der Roboter könnte etwa folgende Aktionen ausführen:

- Sich unter allen Umständen blindlings vorwärts bewegen
- Hindernisse umgehen, indem geprüft wird, ob die vorhergehende Aktion *vorwärts bewegen* gescheitert ist
- Sich um die eigene Achse drehen, um den Programmierer zu unterhalten
- Zufällig eine Aktion auswählen, um ein »Betrunkener Roboter in der Grid World«-Szenario zu modellieren und so weiter ...

Wie Sie wissen, ist es das Hauptziel des Agenten beim RL, einen möglichst großen Return zu erzielen (der als diskontierte Summe der Belohnungen definiert wurde). Verschiedene Policies führen also zu unterschiedlichen Returns, deshalb ist es wichtig, sinnvolle zu finden. Aus diesem Grund sind Policies von so großer Bedeutung und stehen im Mittelpunkt unseres Interesses.

Formal ist eine Policy als die Wahrscheinlichkeitsverteilung der Aktionen für alle möglichen Zustände definiert:

$$\pi(a|s) = P[A_t = a | S_t = s]$$

Sie ist als Wahrscheinlichkeit definiert, nicht als konkrete Aktion, um dem Verhalten eines Agenten eine Zufallskomponente hinzuzufügen. Ich komme später dazu, weshalb das wichtig und nützlich ist. Schließlich ist die deterministische Policy ein Sonderfall, bei dem die Wahrscheinlichkeit für eine erforderliche Aktion 1 betragen muss.

Wenn die Policy **festgelegt** ist und sich nicht ändert, wird aus unserem MDP ein MBP, weil wir die Übergangs- und Belohnungsmatrizen durch die Verwendung der Policy-Wahrscheinlichkeiten vereinfachen und auf die Aktions-Dimension verzichten können.

Herzlichen Glückwunsch, dass Sie es bis hierher geschafft haben! Dieses Kapitel war schwierig, aber der Inhalt ist für die noch folgende praktische Umsetzung wichtig. Nach zwei weiteren einführenden Kapiteln über OpenAI Gym und Deep Learning werden wir endlich die Frage in Angriff nehmen können, wie man einem Agenten beibringt, Aufgaben in der Praxis zu lösen.

1.7 Zusammenfassung

In diesem Kapitel haben wir unsere Reise durch die Welt des Reinforcement Learning angetreten, indem wir betrachtet haben, was das Besondere an RL ist und in welcher Beziehung es zum Paradigma des überwachten und unüberwachten Lernens steht. Sie haben die grundlegenden RL-Formalismen kennengelernt und erfahren, wie sie miteinander interagieren und anschließend habe ich Markov-Prozesse, Markov-Belohnungsprozesse und Markov-Entscheidungsprozesse definiert. Dieses Wissen bildet die Grundlage für das Verständnis der nachfolgenden Teile des Buchs.

Im nächsten Kapitel werden wir die formale Theorie hinter uns lassen und uns der Praxis des RL zuwenden. Ich erörtere, was dazu erforderlich ist, wir betrachten verschiedene Bibliotheken und schreiben unseren ersten Agenten.

Stichwortverzeichnis

`ε`-Greedy-Verfahren 643, 659
2x2-Würfel 731
3D-Druck 529
3x3-Würfel 733
__import__ 578

A

A2C-Verfahren 495
A3C-Verfahren (Asynchronous Advantage Actor-Critic-Verfahren) 335
Abstandssensor 522
Abstrakte Algebra 710
Abstrakte Klasse 53
ACKTR (Advantage Actor-Critic mit Kronecker-Factored Trust Region) 600
ActionWrapper-Klasse 62
Actor 321, 502
Actor-Critic-Verfahren 315
Advantage 227
Agent 31, 47
 Implementierung 497
 zustandsloser 465
Agentengruppe 741
Aktienkurs 272
Aktion 29, 31
 diskrete 31
 stetige 31
Aktionen auf dem Gitter 452
Aktionen, unerlaubte 699
Aktionsraum 42, 52
 stetiger 491
Aktionsselektor 177
Aktionswert 126
Aktivierungsfunktion 496
Aktuator 523, 528
Algebra, abstrakte 710
Algorithmus von Kociemba 712
Algorithmus von Korf 712
Algorithmus, genetischer 624
AlphaGo Zero 689
Analyse, technische 270
Anforderungen an Hard- und Software 50
A-priori-Wahrscheinlichkeit 692
Arbeitsprozess 618
argmax-Modus 398
Assembler 548
Atari-Emulation 265
Atari-Experiment 658
Atari-Spiele 87, 445
Autodidact Iterations (ADI) 718

B

Bandits-Exploration 640
Bar 268
Barometer 522, 527
Barth-Maron, Gabriel 511
Baseline 302
Batchgröße 333
Bellemare, Mark G. 511
Bellman, Richard 123
Bellman'sches Optimalitätsprinzip 123
Bellman-Gleichung 232
 Optimalitätsprinzip 124
Bellman-Operator 511
Bellman-Projektion 514
Belohnung 29
 extrinsische 640
 intrinsische 640
 kumulierte 30
Belohnungsdiagramm 145, 284
Belohnungssystem 27, 273, 544, 545
Beobachtung 29, 32
Beobachtungsraum 52, 746
Berners-Lee, Tim 443
Beschleunigungsmesser 522, 527, 557
Beschreibungstext 480
Bias 524
Bibliothek
 Gym 50
 NumPy 50
 OpenCV Python bindings 50
 PTAN 50
 PyTorch 50
 PyTorch Ignite 93
Bildklassifikation 26
Bitfeld 557
Blackbox-Verfahren 609
 Eigenschaften 609
Blattknoten 698
Blender 529
BLEU (Bilingual Evaluation Understudy) 363
BLEU-Score 368
Börsenhandel 267
Box 492
Box-Klasse 53
Breakout 56, 671
Breitensuche 720
Brettspiel 689
Browserautomatisierung 444
Burda, Yuri 641

C

C 548
 Callback 551
 Candlestick Chart 269
 CartPole-Agent 59
 CartPole-Umgebung 57, 102
 Chatbot 355
 Chatbot-Beispiel 366
 Clock 552
 CMA-ES (Covariance Matrix Adaption Evolution Strategy) 610
 Container starten 460
 Cornell Movie-Dialogue Corpus 366
 cornell.py-Modul 367
 Critic 321, 495, 502
 CrossEntropyLoss 103
 CuLE 265
 Curiosity 523
 Curiosity-driven Exploration 640
 Curriculum Learning 363

D

D4PG (Distributed Distributional Deep Deterministic Gradients) 511
 Damespiel 690
 data.py-Modul 367
 Datenblatt 552
 Datenparallelität 337
 Datenrepräsentation (Zauberwürfel) 712
 Datenübertragung
 serielle 552
 DDPG (Deep Deterministic Policy Gradient) 503, 545
 Decorator 551
 Deep GA 627
 Deep Q-Learning 145
 DeepCube 721
 Definitionsdatei (Modell) 535
 Demonstration 455
 aufzeichnen 468
 Aufzeichnungsformat 470
 Demonstrationen 467
 Deterministisches Policy-Gradienten-Verfahren 502
 Dialoge filtern 367
 Dichtefunktion 640
 Dilbert 36
 Discrete-Klasse 53
 Diskontierungsfaktor 39
 Diskriminator 87
 Docker 448
 Dopamin-System 30
 Double DQN 212
 DQN
 Double 212
 Erweiterung 199
 Implementierung 211, 213
 kategoriales 230
 Konvergenz 211
 rivalisierendes 227
 Verlustfunktion 213

DQNAgent 180
 Drehung 562
 Dynamischer Graph 73

E

Elektrolytkondensator 566
 ELIZA 356
 EM, Training 672
 Encoder-Decoder 360
 Encoder-Klasse 421
 Endzustand 698
 Engine-Klasse 93
 Entropie 118, 303
 Beta 333
 Entropie-Bonus 304
 Entropieregularisierung 602, 660
 Entropieverlust 327
 Environment Model 668
 Env-Klasse 54
 Episode 38, 55
 Epoche 82
 Epsilon-Greedy-Verfahren 147
 Erwartungswertoperator 294
 Evolutionsstrategie 610
 mit CartPole 611
 mit HalfCheetah 617
 ExperienceSourceFirstLast-Klasse 187
 ExperienceSource-Klasse 183, 185
 Exploration 503, 635
 zählerbasierte 654
 Extrinsische Belohnung 640

F

Faltungsmodell 287
 Faltungsnetz 358
 fasttext 360
 Festkommaarithmetik 577
 Feuer-Taste 151
 Finanzhandel 30
 Fitnessfunktion 609, 624
 Fließkommaoperation 560
 FPGA 265
 Frey, Alexander 716
 Fridrich, Jessica 711
 FrozenLake-Umgebung 111
 Fused Deposition Modeling (FDM) 531
 Fusion 360 529

G

gamma 39
 gather()-Funktion 163
 Gauß'sches Rauschen 216
 Gauß-Verteilung 315, 495
 Generation 625
 Generative Adversarial Network 26, 87
 Generator 87
 Genetischer Algorithmus 624
 mit CartPole 624
 mit HalfCheetah 628
 Geometrie 538
 Gerätedatei 549

- Gesamtbelohnung 502
 Gesichtsfeld 741
 Gespiegelte Stichprobenentnahme 613
 get_actions()-Methode 48
 get_observation()-Methode 48
 Gierung 562
 Gleitender Mittelwert 311
 Goodfellow, Ian 87
 Gottes Zahl (Zauberwürfel) 711
 GPIO (General Purpose Input/Output) 551
 GPU-Tensor 71
 Gradient 72, 74
 Gradienten-Clipping 322
 Gradientenparallelität 337, 347
 Gradientenschätzung 614
 Graph
 dynamischer 73
 statischer 73
 Graustufenkonvertierung 154
 Grid World 111, 739
 GridWorld-Klasse 740
 Grosse, Roger 601
 Grundebene 537
 Gruppe 710
 Gruppen-Handle 741
 Gruppentheorie 712
 Gym 51
 Gyroskop 522, 527
- H**
- HalfCheetah 617
 Handelsvolumen 273
 Hardware-Steuerung 548
 Hassel, Mateo 511
 Hesse-Matrix 600
 Hirsche 740
 Ho, Jonathan 610
 Hoffman, Matthew W. 511
 HTML 2.0 443
 Hundedressur 30
- I**
- I2A-Training 680
 i2c.scan() 555
 I2C-Bus 525, 552
 I2C-Operationen 556
 i.i.d.-Annahme 28, 148
 ICLR (International Conference on Learning Representations) 19
 Imagination-Agent 675
 Imagination-augmented Agent 668
 Imagination-Pfad 668
 Importance-Sampling-Theorem 118
 Inertial Measurement Unit (IMU) 526
 Inertiale Messeinheit 526
 Infocom 405
 Interactive Fiction 403
 Interrupt-Handler 559
 Intrinsische Belohnung 640
 item()-Methode 70
- J**
- Jaderberg, Max 667
- K**
- KaiTai 471
 Kalibrierung 524
 Kamera 522
 Kettenregel 502
 K-FAC (Kronecker-Factored Approximation) 601
 K-FAC-Implementierung 601
 Kindprozess 341
 Klasse, abstrakte 53
 Kombinatorische Optimierung 710
 Kommunikationsformen 738
 Komplexität 523
 Konfigurationseinstellungen von Atari-Spielen 200
 Konjugierte-Gradienten-Verfahren 597
 Konkurrenz 738
 Kontextmanager 251
 Korf-Verfahren 722
 Korrekturklassifikationsrate 30
 Kostrikov, Ilya 601
 Kreuzentropie 118
 Training 378
 Kreuzentropie-Verfahren 99, 100
 Kroese, Dirk P. 119
 Krümmung 600
 Kullback-Leibler-Divergenz 118, 232
 Kumulierte Belohnung 30
- L**
- Labyrinth 26
 LED 523
 Lehrlingsmodell 691
 Lernen
 überwachtes 25
 unüberwachtes 26
 Lernrate 142, 332
 LiDAR-Sensor 522
 Lillicrap, Timothy P. 502
 log_softmax-Funktion 238
 Logits 103, 295
 Log-Likelihood-Verlust 362
 Lokales Modell 667
- M**
- Madhavan, Vanisht 624
 MAgent 737
 MAgent-Umgebung 739
 Magnetometer 527
 Manning, Rob 523
 Markov-Belohnungsprozess 39
 Markov-Eigenschaft 35, 148
 Markov-Entscheidungsprozess 35, 42
 Markov-Prozess 35
 stationärer 38
 MARL (Multiagenten-Reinforcement-Learning) 738
 Mars-Rover 523

Martens, James 601
 Maschinensprache 524
 Matrizenmultiplikation 570
 Mauer 741
 McAleer, Stephen 709
 Merkmalsraum 742
 Messbereich 552
 Messeinheit, inertielle 526
 Messfrequenz 557
 Metaprogrammierung 573
 MicroPython 525, 548
 Mini World of Bits 445
 Miniaturkarte 742
 Minimax-Verfahren 689
 MiniPacman 669
 Minitaur 525
 Minitaur-Umgebung 494
 Mittelwert, gleitender 311
 Mnih, Volodymyr 667
 MNIST 56
 Modell

- exportieren 573
- lokales 667

 Modellbasierte Verfahren 665
 Modellfreie Verfahren 100
 Monitoring 83
 Monitor-Klasse 63
 Monte-Carlo-Baumsuche 690, 691, 720

- Implementierung 697

 Motor 523
 MountainCar-Umgebung 641
 mp.Process 339
 mp.Queue 339
 MuJoCo-Paket 493
 Multiprocessing 339
 Mutation 624

N

Navigation im Web 443
 Nebenläufige Programmierung 337
 Neigung 562
 Netz, verdrahtetes 216
 Netz-Destillation-Verfahren 656
 NeurIPS (Neural Information Processing Systems) 19
 nltk-Bibliothek 368
 nn.Module 76
 NoisyLinear-Klasse 644
 Novelty Search 627, 628
 N-Schritt-DQN 208

O

ObservationWrapper-Klasse 62
 Off-policy 100
 Off-Policy-Verfahren 210
 One-hot-Codierung 359
 On-policy 100
 On-Policy-Verfahren 210
 OpenAI Universe 446

OpenGL 501
 OpenSCAD 529
 Optimalität 122
 Optimalitätsprinzip 123
 Optimierer 81, 620
 Optimierung

- kombinatorische 710

 Optimierung der Hyperparameter 331
 Optimierungsaufgabe 709
 Optimierungsverfahren zweiter Ordnung 600
 Ornstein-Uhlenbeck-Prozess 497, 503
 Ostrovski, Georg 640

P

Padding 374
 Pendelzeit 230
 Permutationen 729
 Perzentilwert 106
 Physiksimulation 493
 Physiksimulator 57
 Pinbelegung 554
 Platine 525
 Policy 44

- Definition 44
- Gründe 292
- Repräsentation 292
- überprüfen 464

 Policy Gradient 315
 PolicyAgent 182
 Policybasierte Verfahren 100, 300
 Policy-Destillation 670, 680
 Policy-Gradienten-Verfahren

- deterministisches 502

 POMDP 149
 Pong 146, 150, 322, 739

- Hyperparameter 172

 Population 624
 PPO

- Hyperparameter 592
- Implementierung 591

 PPO (Proximal Policy Optimierung) 590, 649
 PPO-Verfahren 660

- mit Netz-Destillation 661
- mit verdrahteten Netzen 662

 Prädikat-Funktion 724
 Preprocessor-Klasse 421
 Priorisierter Replay Buffer 220
 Programmierung

- nebenläufige 337

 Projektion einer Verteilung 235
 PTAN (PyTorch Agent Net) 175
 Pulsweitenmodulation 565
 PyBullet 493, 534
 PyTorch Agent Net (PTAN) 175
 PyTorch Ignite 92
 PyTorch-Bibliothek 67

Q

Q-Learning 141

Q-Learning-Verfahren 126

R

Rahmen 525, 528
 Raubtier-Beute-Modell 739
 Rauschen 524
 Registrierung 727
 Regression 26
 Regressionsaufgabe 146
 Reibung 536
 REINFORCE-Verfahren 294, 295
 render()-Methode 54
 Replay Buffer 148
 REPL-Eingabeaufforderung 549
 Repräsentation von Spielstellungen 695
 reset()-Methode 52, 55
 RewardWrapper-Klasse 62
 River-Swim-Umgebung 637
 Roboschool 584
 RoboschoolAnt-Umgebung 589
 RoboschoolHalfCheetah-Umgebung 586
 Roboter 521
 Robotermouse 26
 Robotik 521
 robot-Klasse 539
 Rollout 668
 Rollout-Encoder 670
 RolloutEncoder-Klasse 679
 Rotationsinvarianz 715
 Rubik, Ernő 711
 Rubik's Cube *siehe* Zauberwürfel

S

SAC (Soft-Actor-Critic) 602
 SAC-Implementierung 603
 Salimans, Tim 610
 Schach 30
 Schlupf 277
 Schmelzschichtungsverfahren 531
 Schrittmotor 523
 Schulman, John 590, 597
 Schulnoten 30
 screen 549
 Seaquest-Umgebung 658
 Sensor auslesen 558
 Sensorinitialisierung 556
 Sensor-Klassen 559
 Seq2Seq-Training 361
 Sequential-Klasse 77
 Serielle Datenübertragung 552
 Servomotor 523, 525, 528
 Servomotoren ansteuern 565
 Shared Seed 617
 Signal-Rauschen-Verhältnis 219
 Silver, David 502
 Simon, William L. 523
 Singmaster, David 716
 Skalierungsfaktor 315
 Slicer 531
 Slicing 531

Socher, Richard 357
 Softsync 509
 Sokoban 669
 solver-Modi 729
 Speichereffizienz 696, 714
 Spielbaum 691
 Spielregeln 695
 Spieltheorie 738
 Standard-A2C-Agent 671
 Standard-A2C-Verfahren 584
 Standardbibliothek 551
 Statischer Graph 73
 step()-Methode 52
 Stetiger Aktionsraum 491
 Steuerungsaufgaben 56
 Stichprobeneffizienz 135, 666
 Stichprobenentnahme
 gespiegelte 613
 Stichprobenineffizienz 336
 Stimmungsanalyse 26
 STL-Format 530
 StockEnv-Klasse 273
 Stockfish 691
 Such, Felipe Petroski 624
 Suchvorgang 729
 Summer 523
 Support & Resistance 288
 Sutton, Richard 208
 Synchronisation 193
 Synchronisationsmodus 192

T

Taarnoja, Tuomas 602
 Tabular Q-Learning 140, 141
 TargetNet-Klasse 191
 Taster 522
 Tastgrad 565
 Teacher Forcing 362
 Teacher-Forcing-Modus 371
 Technische Analyse 270
 Tensor
 Attribute 74
 Definition 67
 erzeugen 68
 GPU 71
 Gradient 72
 Operationen 71
 skalärer 70
 und Gradient 74
 wiederholen 677
 TensorBoard 83, 84
 Testfunktion 497
 Textadventure 404
 Textklassifikation 26
 TextWorld 403
 Thermometer 522
 Tic-Tac-Toe 478, 691
 Tiefensuche 720
 Tiger 740
 Zusammenarbeit 750

Timer 551, 567
 Timer auslesen 559
 Token 367
 Tokenisierung 367
 torch.nn-Paket 76
 torch.tensor()-Methode 70
 TorchScript 74
 Training durch Demonstration 473
 TRPO (Trust Region Policy Optimization) 597
 tw-make 407
 tw-play 408

U

UART (Universal Asynchronous Receiver/Transmitter) 551
 Übergangendiagramm 43
 Übergangsmatrix 36
 Übertragbarkeit 666
 Überwachtes Lernen 25
 Umgebung 31, 47
 erzeugen 55, 449
 Initialisierung 47
 Interaktion 147
 konfigurieren 449
 mit stetigem Aktionsraum 492
 Umgebung in Gym 54
 Umgebungsmodell 669
 Umgebungszustand 139
 Umkehrfunktion 724
 Unerlaubte Aktionen 699
 UNREAL 667
 Unreal Engine 534
 Unüberwachtes Lernen 26
 utils.py-Modul 368

V

Varianz 232, 315, 524
 Varianz der Gradienten 302, 365
 Vektorisierung 448
 Verarbeitung natürlicher Sprache (NLP) 355
 Verdeckter Zustand 358
 Verfahren
 modellbasierte 665
 modellfrei 100
 policybasierte 300
 policybasiertes 100
 vorhersagebasierte 641
 wertebasierte 300
 zählerbasierte 640
 Verhaltenscharakteristik 628
 Verlauf 35
 Verlust 80
 Verlustfunktion 80
 Verrauschtes Netz 216, 639, 652
 Versteckspiel 737
 Verzweigungsfaktor 691
 Videoaufzeichnung 65, 501, 511, 590
 Vier gewinnt 690, 706

Vier-gewinnt-Bot 694
 VNC-Client 445
 Mauszeiger 470
 VNC-Client manuell verbinden 462
 VNC-Protokoll 448
 VNC-Proxy 469
 VNC-Standardports 456
 Vorhersagebasierte Verfahren 641
 Voyager 1 522

W

Wahrscheinlichkeitsdichte 495
 Wahrscheinlichkeitsverteilung 230
 Warteschlange 344, 620
 Wasserstein-Metrik 232
 Web Scraping 444
 Wert
 Definition 291
 Wertebasierte Verfahren 300
 Wertiteration 128, 130, 139
 Wetterderivat 267
 Word-Klasse 411
 Wort-Embedding 359, 360
 Worthäufigkeit 360
 Wrapper 151
 Wrapper-Klasse 61
 Wu, Yuhuai 600
 Würfel-Umgebung 723

X

X11-Forwarding 65, 66
 Xvfb 65, 501

Y

yield-Funktion 89

Z

Zähler 720
 Zählerbasierte Exploration 654
 Zählerbasierte Verfahren 640
 Zauberwürfel 710, 712
 Zauberwürfel-Modell 719
 Zielnetz 148
 Zork I 405
 Zufälligkeit 640
 Zufallszahlengenerator 275, 617
 Zusammenarbeit 738
 Zustand 35
 verdeckter 358
 Zustand (Zauberwürfel) 713
 Zustandsbehaftung 505
 Zustandsraum 35, 711
 Zustandsübergang 42
 Zustandswert 40, 121, 721
 Zustandszähler 646
 Zweierkomplement 558