mitp

Maxim Lapan

Deep Reinforcement Learning

Das umfassende Praxis-Handbuch

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

Inhaltsverzeichnis

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:

```
rl_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.«

Kapitel 1

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 fehlklassifiziert 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 Objektlokalisierung:** 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 wohlbekannte 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.

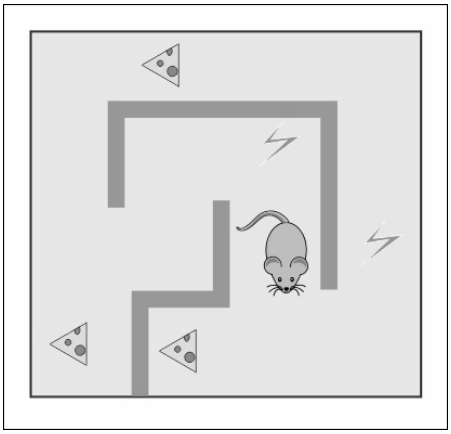

Abb. 1.1: Das Labyrinth der Robotermaus

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 überwachtes und unüberwachtes 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.

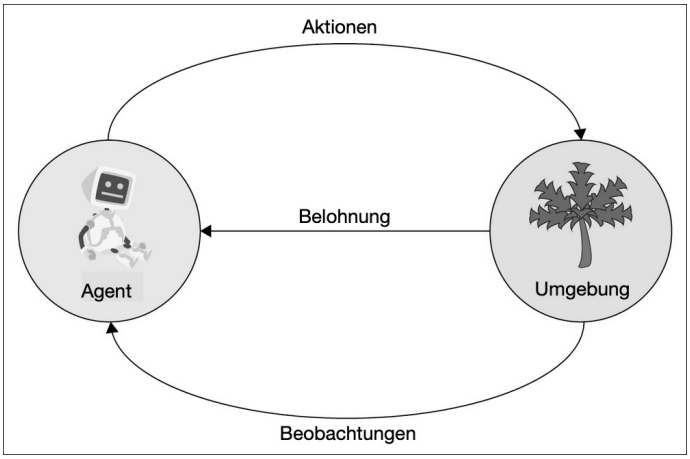

Abb. 1.2: Agent, Umgebung und Kommunikationskanäle

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 sekündlich 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 verschleierter 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 (irgendjemand 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.

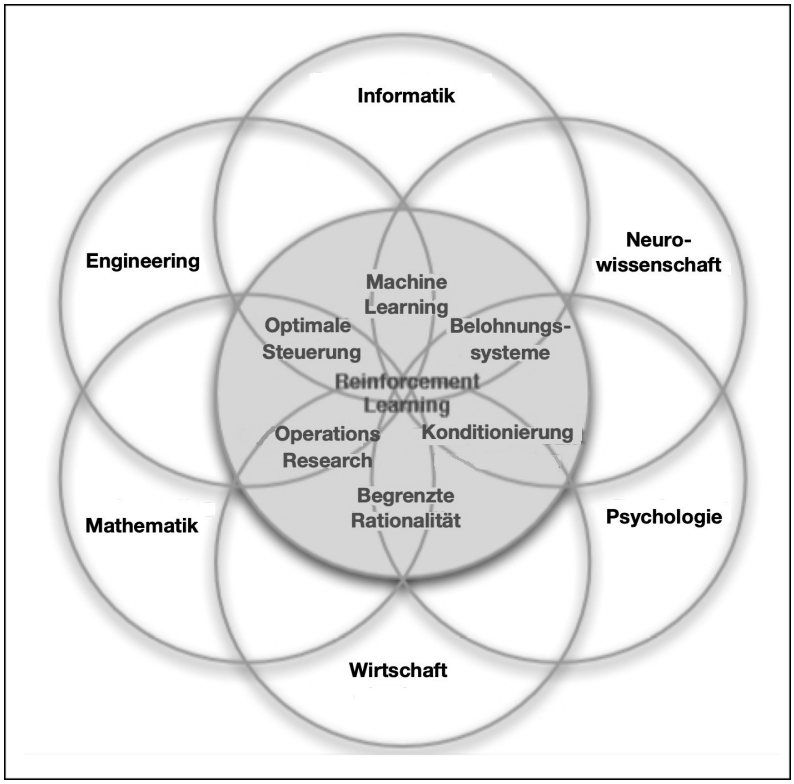

Abb. 1.3: Verschiedene Bereiche des Reinforcement Learnings

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 Matroschka-Puppe: 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 bezeichnen 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.

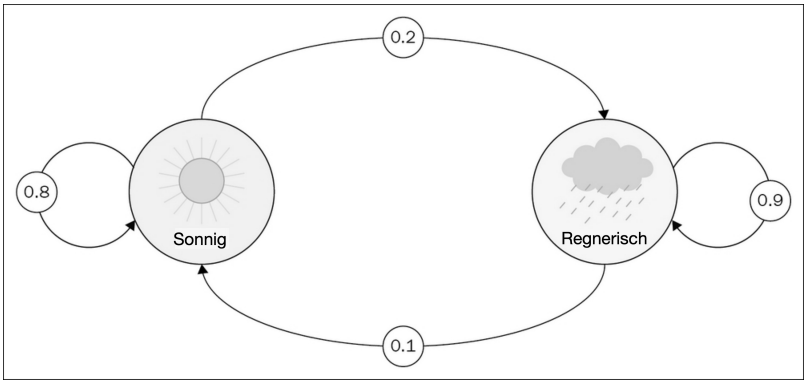

Abb. 1.4: Modell für sonniges oder regnerisches Wetter

Abbildung 1.5 zeigt den Graphen der Zustandsübergänge.

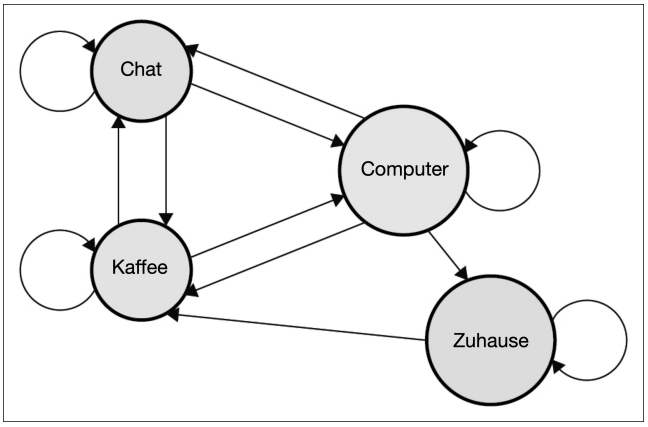

Abb. 1.5: Graph der Zustandsübergänge

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.

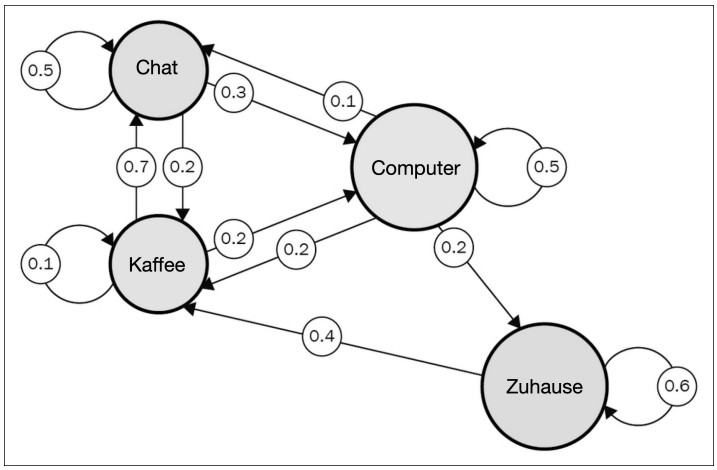

Abb. 1.6: Graph der Zustandsübergänge mit Übergangswahrscheinlichkeiten

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:

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

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.

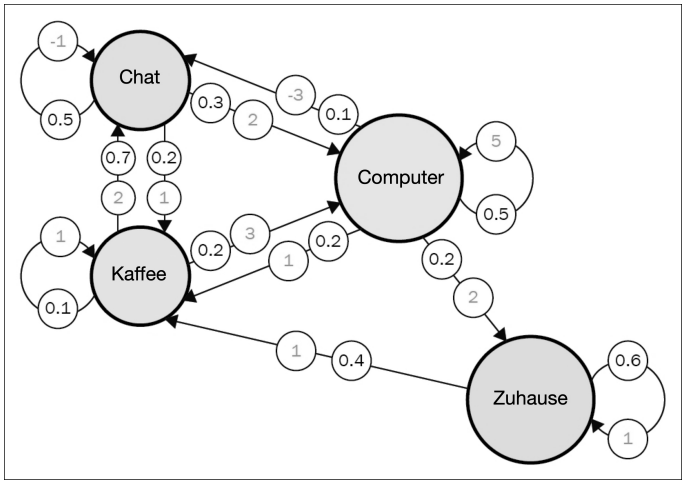

Abb. 1.7: Graph der Zustandsübergänge mit Übergangswahrscheinlichkeiten (dunkel) und Belohnungen (hell)

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(Chat) = -1 * 0,5 + 2 * 0,3 + 1 * 0,2 = 0,3*
- *V(Kaffee) = 2 * 0,7 + 1 * 0,1 + 3 * 0,2 = 2,1*
- *V(Zuhause) = 1 * 0,6 + 1 * 0,4 = 1,0*
- *V(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 gamma 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 gamma, 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 gamma 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).

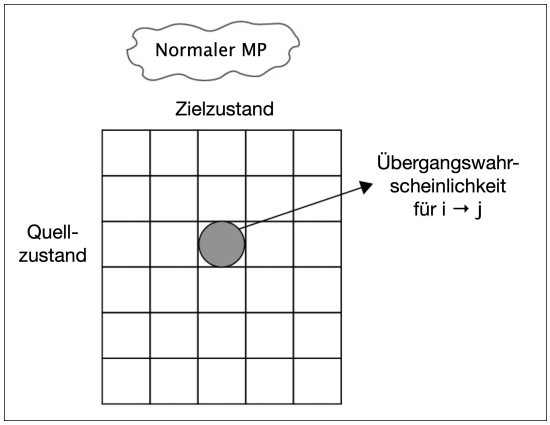

Abb. 1.8: Übergangsmatrix

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 × 3 × 4 = 36 Zustände (der Roboter darf sich auf allen Feldern aufhalten und in alle Richtungen blicken).

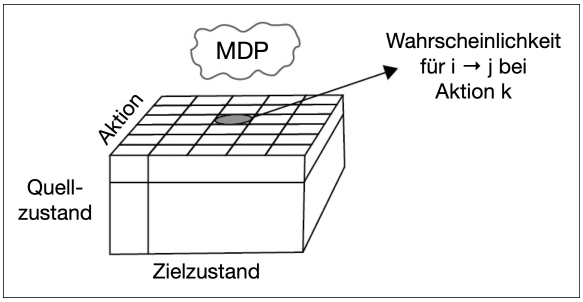

Abb. 1.9: Übergangswahrscheinlichkeiten beim MDP

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, 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, nach oben*) befindet, aber mit einer Wahrscheinlichkeit von 10 Prozent drehen die Räder durch und er verbleibt an Position (*1, 1, 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*).

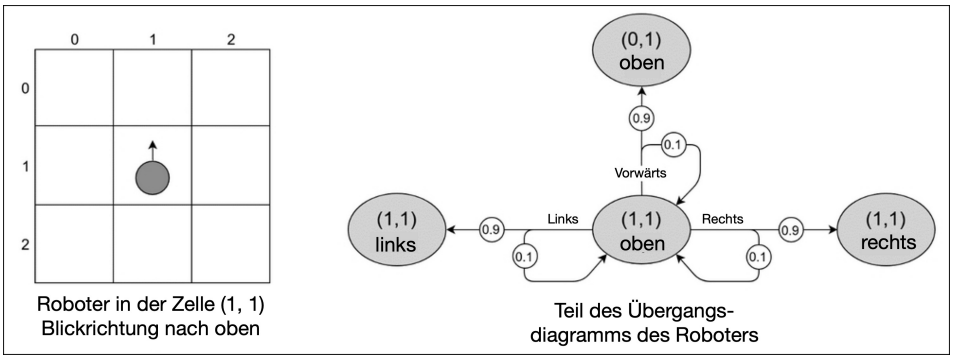

Abb. 1.10: Die Umgebung in der »Grid World«

Um aus unserem MBP einen 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