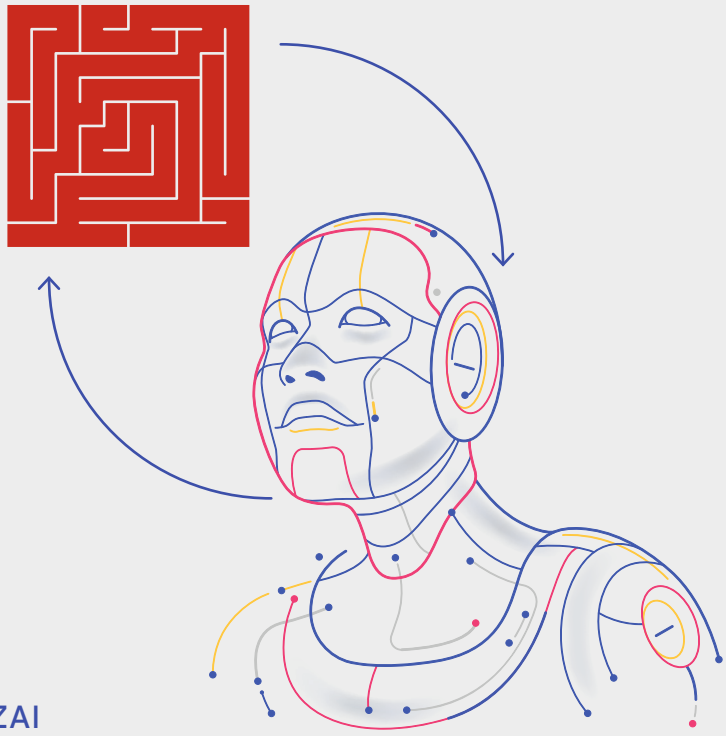

alexander ZAI
brandon BROWN

EINSTIEG IN DEEP REINFORCEMENT LEARNING

KI-Agenten mit Python und PyTorch programmieren

GitHub-Repository zum Buch

HANSER

Zai/Brown

Einstieg in Deep Reinforcement Learning

Alexander Zai
Brandon Brown

Einstieg in Deep Reinforcement Learning

KI-Agenten mit Python und PyTorch programmieren

HANSER

Titel der Originalausgabe: „Deep Reinforcement Learning“, © 2020 by Manning Publications Co.

Authorized translation of the English edition. This translation is published and sold by permission of Manning Publications, the owner of all rights to publish and sell the same.

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

Lektorat: Sylvia Hasselbach
Copy editing: Christian Schneider, Traunstein
Fachlektorat: Peter Seeberg, Neubiberg
Layout: Manuela Treindl, Fürth
Umschlagdesign: Marc Müller-Bremer, München, *www.rebranding.de*
Umschlagrealisation: Max Kostopoulos
Titelmotiv: © *shutterstock.com/ioa*
Datenbelichtung, Druck und Bindung: Kösel, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

Print-ISBN: 978-3-446-45900-7
E-Book-ISBN: 978-3-446-46608-1
E-Pub-ISBN: 978-3-446-46609-8

Inhalt

Vorwort

Deep Reinforcement Learning rückte 2015 ins Rampenlicht, als DeepMind einen Algorithmus entwickelte, der in der Lage ist, eine Reihe von Atari-2600-Spielen besser als der Mensch zu spielen. Die künstliche Intelligenz schien endlich echte Fortschritte zu machen, und wir wollten unseren Teil dazu beitragen.

Wir haben beide einen Hintergrund im Software-Engineering und Interesse an den Neurowissenschaften, und wir interessieren uns schon seit Langem für das breitere Feld der künstlichen Intelligenz (tatsächlich hat einer von uns sein erstes neuronales Netz noch vor der High School in C# geschrieben). Diese frühen Erfahrungen führten zu keinem anhaltenden Interesse, da sie in die Zeit vor der Revolution des Deep Learning um das Jahr 2012 fielen, als die überragende Leistung des Deep Learning deutlich wurde. Aber nachdem wir die erstaunlichen Erfolge des Deep Learning gesehen hatten, wandten wir uns wieder den aufregenden und aufblühenden Bereichen des Deep Learning und dann des Deep Reinforcement Learning zu, und wir beide haben das Machine Learning auf die eine oder andere Weise stärker in unsere berufliche Laufbahn integriert. Alex wechselte in eine Karriere als Machine Learning Engineer und machte sich an wenig bekannten Orten wie Amazon einen Namen, und Brandon begann, Machine Learning in der akademischen neurowissenschaftlichen Forschung einzusetzen. Als wir uns in Deep Reinforcement Learning vertieften, mussten wir uns durch Dutzende von Lehrbüchern und Forschungsartikel kämpfen, wobei wir uns in fortgeschrittene Mathematik und die Theorie des maschinellen Lernens vertieften. Wir stellten jedoch fest, dass die Grundlagen des Deep Reinforcement Learning durchaus zugänglich sind, wenn man einen Hintergrund im Software-Engineering hat. Die gesamte Mathematik lässt sich leicht in eine Sprache übersetzen, die für jeden Programmierer gut lesbar ist.

Wir begannen, über die Dinge zu bloggen, die wir in der Welt des Machine Learning lernten, und über Projekte, die wir bei unserer Arbeit verwendeten. Am Ende bekamen wir eine ganze Menge positives Feedback, was uns auf die Idee brachte, gemeinsam an diesem Buch zu arbeiten. Wir sind der Meinung, dass die meisten Ressourcen, die es zum Lernen komplexer Dinge gibt, entweder zu einfach sind und die fesselndsten Aspekte des Themas auslassen oder für Menschen ohne fortgeschrittenen mathematischen Hintergrund unzugänglich sind. Mit diesem Buch haben wir uns bemüht, ein für Experten geschriebenes Kompendium in einen Kurs für diejenigen zu übersetzen, die nichts weiter als einen Programmierhintergrund und einige Grundkenntnisse über neuronale Netze haben. Wir wenden einige neuartige Lehrmethoden an, von denen wir glauben, dass sie unser Buch auszeichnen und zu einem viel schnelleren Verständnis führen. Wir fangen bei den Grundlagen an, und am Ende werden Sie innovative Algorithmen implementieren, die von industriellen Forschungsprojekten wie DeepMind und OpenAI sowie von leistungsstarken akademischen Einrichtungen wie dem Berkeley Artificial Intelligence Research (BAIR) Lab und dem University College London entwickelt wurden.

Danksagung

Dieses Buch hat viel länger gedauert, als wir erwartet hatten, und wir haben unseren Lektorinnen Candace West und Susanna Kline viel zu verdanken. Sie haben uns in jeder Phase des Prozesses geholfen und uns auf Kurs gehalten. Wenn man ein Buch schreibt, muss man viele Details im Auge behalten, und ohne das professionelle und unterstützende Lektorenteam wären wir ins Schwimmen geraten.

Unser Dank gilt auch unseren technischen Redakteuren Marc-Philippe Huget und Al Krinker sowie allen Gutachtern, die sich die Zeit genommen haben, unser Manuskript zu lesen und uns entscheidendes Feedback zu geben. Unser besonderer Dank gilt den Gutachtern Al Rahimi, Ariel Gamiño, Claudio Bernardo Rodriguez, David Krief, Dr. Brett Pennington, Ezra Joel Schroeder, George L. Gaines, Godfred Asamoah, Helmut Hauschild, Ike Okonkwo, Jonathan Wood, Kalyan Reddy, M. Edward (Ed) Borasky, Michael Haller, Nadia Noori, Satyajit Sarangi und Tobias Kaatz. Wir möchten auch allen bei Manning danken, die an diesem Projekt mitgearbeitet haben: Karen Miller (Developmental Editor), Ivan Martinović (Review Editor), Deirdre Hiam (Project Editor), Andy Carroll (Copy Editor) und Jason Everett (Proofreader).

In der heutigen Zeit werden viele Bücher über verschiedene Online-Dienste im Selbstverlag herausgegeben, und diese Option hat uns zunächst gereizt. Nachdem wir jedoch diesen ganzen Prozess durchlaufen haben, können wir den enormen Wert eines professionellen Lektorats erkennen. Insbesondere danken wir dem Copy Editor Andy Carroll für sein aufschlussreiches Feedback, das die Klarheit des Textes wesentlich verbessert hat.

Alex bedankt sich bei seinem PI Jamie, der ihn bereits während seines Grundstudiums an Machine Learning herangeführt hat.

Brandon dankt seiner Frau Xinzhu dafür, dass sie seine langen Nächte am Schreibtisch und seine Abwesenheit von der Familie erduldet und ihm zwei wunderbare Kinder, Isla und Avin, geschenkt hat.

Die Autoren

Alex Zai arbeitete als Chief Technology Officer bei Codesmith, einem immersiven Coding-Bootcamp, wo er weiterhin als Technical Advisor tätig ist, sowie als Software-Ingenieur bei Uber und als Machine Learning Engineer bei Banjo und Amazon. Darüber hinaus wirkt er am Open-Source Deep-Learning-Framework Apache MXNet mit. Er hat zwei Unternehmen mitbegründet, von denen eines durch Y Combinator gefördert wurde.

Brandon Brown beschäftigte sich von klein auf mit dem Programmieren und arbeitete während des Studiums als Teilzeit-Software-Ingenieur, entschied sich aber schließlich für eine Karriere in der Medizin; nebenbei arbeitete er als Software-Ingenieur im Bereich der Gesundheitstechnologie. Heute ist er Arzt und verfolgt seine Forschungsinteressen im Bereich der Computational Psychiatry, inspiriert durch Deep Reinforcement Learning.

Über dieses Buch

Wer dieses Buch lesen sollte

Einstieg in Deep Reinforcement Learning ist ein Kurs, der entwickelt wurde, um Sie von den grundlegenden Konzepten des Reinforcement Learning bis hin zur Implementierung der neuesten Algorithmen zu führen. Als Kurs konzentriert sich jedes Kapitel auf ein Hauptprojekt, das das jeweilige Thema oder Konzept veranschaulichen soll. Wir haben jedes Projekt so konzipiert, dass es effizient auf einem neueren Laptop ausgeführt werden kann; wir erwarten nicht, dass Sie Zugriff auf teure GPUs oder Cloud-Computing-Ressourcen haben (auch wenn der Zugriff auf diese Ressourcen einiges beschleunigt).

Dieses Buch richtet sich an Personen mit einem Programmierhintergrund, insbesondere mit Grundkenntnissen in Python, und an Personen, die zumindest ein grundlegendes Verständnis von neuronalen Netzen haben (auch bekannt als Deep Learning). Mit „grundlegendem Verständnis" meinen wir, dass Sie schon einmal versucht haben, ein einfaches neuronales Netz in Python zu implementieren, auch wenn Sie nicht ganz verstanden haben, was tief im Inneren vor sich geht. Obwohl sich dieses Buch auf die Verwendung von neuronalen Netzen in Bezug auf das Reinforcement Learning konzentriert, werden Sie wahrscheinlich auch viel Neues über Deep Learning im Allgemeinen lernen, das sich auf andere Probleme außerhalb des Reinforcement Learning anwenden lässt, sodass Sie kein Experte für Deep Learning sein müssen, bevor Sie sich in das Deep Reinforcement Learning stürzen.

Wie dieses Buch organisiert ist: Ein Fahrplan

Das Buch hat zwei Teile mit insgesamt 11 Kapiteln.

Teil I erläutert die Grundlagen des Deep Reinforcement Learning.

- *Kapitel 1* bietet eine Einführung in Deep Learning, Reinforcement Learning und die Verbindung von beidem zum Deep Reinforcement Learning.
- *Kapitel 2* führt in die grundlegenden Konzepte des Reinforcement Learning ein, die im weiteren Verlauf des Buches immer wieder auftauchen werden. Wir implementieren auch unseren ersten angewandten Reinforcement-Learning-Algorithmus.
- *Kapitel 3* stellt Deep Q-Learning vor, eine der beiden Hauptklassen von Deep-Reinforcement-Algorithmen. Dabei handelt es sich um den Algorithmus, mit dem DeepMind im Jahr 2015 bei vielen Atari 2600 Spielen die Leistung von Menschen übertroffen hat.
- *Kapitel 4* beschreibt die andere Hauptklasse von Deep-Reinforcement-Learning-Algorithmen, die Policy-Gradient-Methoden. Wir verwenden sie, um einen Algorithmus für ein einfaches Spiel zu trainieren.
- *Kapitel 5* zeigt, wie wir Deep Q-Learning aus Kapitel 3 und Policy-Gradient-Methoden aus Kapitel 4 zu einer kombinierten Klasse von Algorithmen, den sogenannten Actor-Critic-Algorithmen, verbinden können.

Teil II baut auf den Grundlagen auf, die wir in Teil I gelegt haben, um die größten Fortschritte der letzten Jahre im Bereich des Deep Reinforcement Learning zu behandeln.

- *Kapitel 6* zeigt, wie evolutionäre Algorithmen, die Prinzipien der biologischen Evolution nutzen, implementiert werden können, um neuronale Netze zu trainieren.
- *Kapitel 7* beschreibt eine Methode zur signifikanten Verbesserung der Leistung des Deep Q-Learning durch die Einbeziehung probabilistischer Konzepte.
- *Kapitel 8* stellt eine Methode vor, wie man Reinforcement-Learning-Algorithmen ein Gefühl der Neugierde vermittelt, damit sie ihre Umgebung ohne externe Hinweise erforschen können.
- *Kapitel 9* zeigt, wie wir das, was wir beim Trainieren von Reinforcement-Learning-Algorithmen mit einem Agenten gelernt haben, auf Systeme mit mehreren interagierenden Agenten ausweiten können.
- *Kapitel 10* beschreibt, wie man Deep-Reinforcement-Learning-Algorithmen durch den Einsatz von Aufmerksamkeitsmechanismen besser interpretierbar und effizienter machen kann.
- *Kapitel 11* schließt das Buch mit einer Erörterung all der spannenden Bereiche des Deep Reinforcement Learning, die wir aus Platzgründen nicht behandeln konnten, die Sie aber vielleicht interessieren.

Die Kapitel in Teil I sollten der Reihe nach gelesen werden, da jedes Kapitel auf den Konzepten des vorhergehenden Kapitels aufbaut. Die Kapitel in Teil II können mehr oder weniger in beliebiger Reihenfolge angegangen werden, obwohl wir auch hier empfehlen, sie der Reihe nach zu lesen.

Über den Code

Wie bereits erwähnt, ist dieses Buch ein Kurs, daher haben wir den gesamten Code, der für die Durchführung der Projekte erforderlich ist, in den Haupttext des Buches aufgenommen. Grundsätzlich fügen wir kürzere Codeblöcke als Inline-Code ein, der in `dieser Schriftart` formatiert ist, und stellen größere Codeblöcke in separaten, nummerierten Code-Listings dar.

Hinweis

Zum Zeitpunkt der Drucklegung sind wir zuversichtlich, dass der gesamte In-Text-Code funktioniert, aber wir können nicht garantieren (insbesondere für diejenigen unter Ihnen, die dies in gedruckter Form lesen), dass der Code auf lange Sicht fehlerfrei bleibt, da sich das Gebiet des Deep Learning und folglich auch seine Bibliotheken schnell weiterentwickeln. Auch wurde der In-Text-Code auf das für das Funktionieren der Projekte notwendige Minimum reduziert, sodass wir Ihnen dringend empfehlen, die Projekte in diesem Buch mit dem Code aus dem GitHub-Repository zu diesem Buch zu verfolgen: *http://mng.bz/JzKp*. Wir beabsichtigen, den Code auf GitHub für die nächste Zeit auf dem neuesten Stand zu halten, und er enthält auch zusätzliche Kommentare und Code, den wir zur Generierung vieler Abbildungen im Buch verwendet haben. Daher ist es am besten, wenn Sie das Buch zusammen mit dem entsprechenden Code in den Jupyter-Notebooks lesen, die auf dem GitHub-Repository zu finden sind.

Wir sind zuversichtlich, dass dieses Buch Ihnen die Konzepte des Deep Reinforcement Learning vermitteln wird und nicht nur, wie man Dinge in Python effektiv programmiert. Wenn Python irgendwie verschwinden würde, nachdem Sie dieses Buch gelesen haben, wären Sie immer noch in der Lage, die Algorithmen in einer anderen Sprache oder einem anderen Framework zu implementieren, da Sie die Grundlagen verstanden haben.

liveBook-Diskussionsforum

Mit dem Kauf von *Einstieg in Deep Reinforcement Learning* erhalten Sie Zugang zu einem privaten Web-Forum von Manning Publications, in dem Sie Kommentare zum Buch abgeben, technische Fragen stellen und Hilfe von den Autoren und anderen Benutzern erhalten können. Um auf das Forum zuzugreifen, nutzen Sie den folgenden Link: *https://livebook.manning.com/#!/book/deep-reinforcement-learning-in-action/discussion*. Mehr über die Manning-Foren und die Verhaltensregeln erfahren Sie unter *https://livebook.manning.com/#!/discussion*.

Mannings Engagement für unsere Leser besteht darin, einen Ort zu schaffen, an dem ein sinnvoller Dialog zwischen einzelnen Lesern sowie zwischen Lesern und Autoren stattfinden kann. Es handelt sich dabei nicht um eine Verpflichtung zu einem bestimmten Umfang der Teilnahme seitens der Autoren, deren Beitrag zum Forum freiwillig (und unbezahlt) bleibt. Wir schlagen deshalb vor, dass Sie versuchen, den Autoren einige herausfordernde Fragen zu stellen, damit ihr Interesse geweckt wird! Das Forum und die Archive früherer Diskussionen sind auf der Website des Verlags zugänglich, solange das Buch im Handel erhältlich ist.

Teil I: Grundlagen

Teil I besteht aus fünf Kapiteln, die die grundlegendsten Aspekte des Deep Reinforcement Learning vermitteln. Nachdem Sie Teil I gelesen haben, können Sie die Kapitel in Teil II in beliebiger Reihenfolge verstehen.

- *Kapitel 1* beginnt mit einer Einführung in das Deep Reinforcement Learning, in der die wichtigsten Konzepte und ihr Nutzen erläutert werden.
- In *Kapitel 2* beginnen wir mit dem Aufbau angewandter Projekte, die die grundlegenden Ideen des Reinforcement Learning veranschaulichen.
- In *Kapitel 3* implementieren wir ein Deep Q-Network – die gleiche Art von Algorithmus, den DeepMind verwendete, um Atari-Spiele auf übermenschlichen Leveln zu spielen.
- *Kapitel 4 und 5* runden die gebräuchlichsten Reinforcement-Learning-Algorithmen ab, nämlich Policy-Gradient-Methoden und Actor-Critic-Methoden. Wir werden die Vor- und Nachteile dieser Ansätze im Vergleich zu Deep Q-Networks betrachten.

1 Was ist Reinforcement Learning?

Dieses Kapitel beinhaltet:

- einen kurzen Überblick über Machine Learning (maschinelles Lernen),
- die Einführung von Reinforcement Learning (bestärkendes Lernen – RL) als Teilbereich,
- das grundlegende Framework (Programmiergerüst) des Reinforcement Learning.

Computersprachen der Zukunft werden sich mehr mit Zielen und weniger mit vom Programmierer vorgegebenen Prozeduren beschäftigen.

Marvin Minksy, 1970 ACM-Turing-Vorlesung

Wenn Sie dieses Buch lesen, sind Sie wahrscheinlich damit vertraut, wie tiefe neuronale Netze für Dinge wie Bildklassifikation oder Vorhersagen verwendet werden (und wenn nicht, lesen Sie einfach weiter; wir haben auch einen Crash-Kurs zu Deep Learning (tiefem Lernen) im Anhang). *Deep Reinforcement Learning* (DRL) ist ein Teilgebiet des Machine Learning, das Deep-Learning-Modelle (d. h. neuronale Netze) für Reinforcement-Learning-(RL)-Aufgaben (siehe Abschnitt 1.2 verwendet. Bei der Bildklassifizierung haben wir eine Menge von Bildern, die einer Reihe von diskreten Kategorien entsprechen, wie z. B. Bilder von verschiedenen Tierarten, und wir möchten, dass ein Machine-Learning-Modell ein Bild interpretiert und die Tierart auf dem Bild klassifiziert (siehe Bild 1.1).

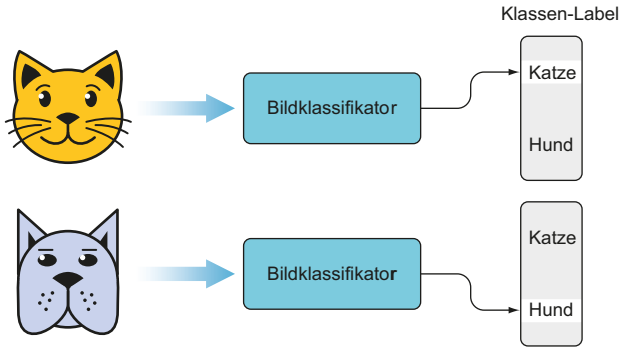

Bild 1.1 Ein Bildklassifikator ist eine Funktion oder ein Lernalgorithmus, der ein Bild aufnimmt und ein Klassenlabel zurückgibt, das das Bild einer Kategorie oder Klasse zuordnet, die aus einer endlichen Anzahl möglicher Kategorien ausgewählt wurde.

1.1 Das „Tiefe“ beim Deep Reinforcement Learning

Deep-Learning-Modelle sind nur eine von vielen Arten von Machine-Learning-Modellen, die wir zur Klassifikation von Bildern verwenden können. Im Allgemeinen brauchen wir nur eine Art Funktion, die ein Bild aufnimmt und ein Klassenlabel zurückgibt (in diesem Fall das Label, das angibt, welche Art von Tier im Bild dargestellt ist), und normalerweise hat diese Funktion einen festen Satz einstellbarer *Parameter* – wir nennen diese Art von Modellen

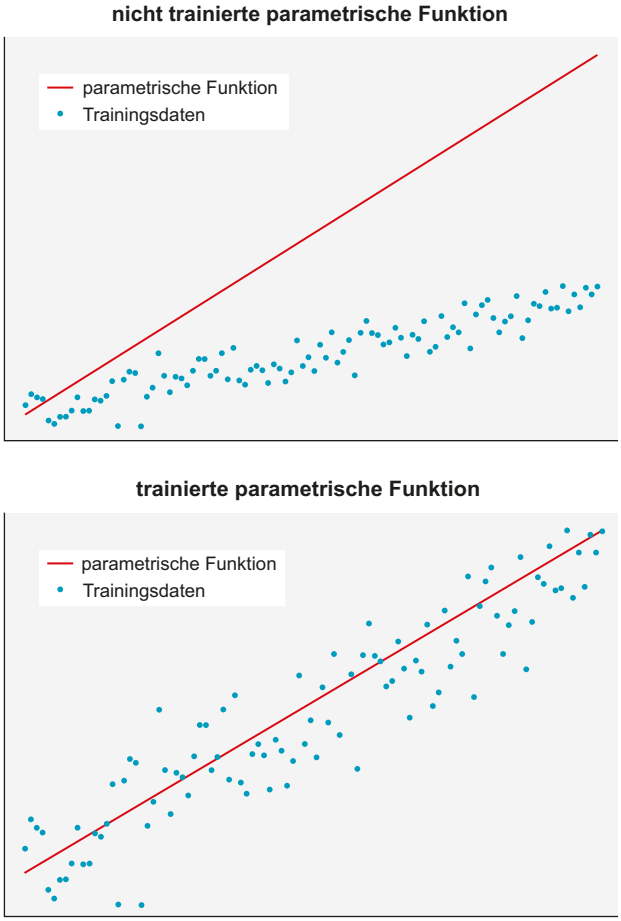

Bild 1.2 Das vielleicht einfachste Machine-Learning-Modell ist eine lineare Funktion der Form $f(x) = mx + b$ mit den Parametern m (die Steigung) und b (der Achsenabschnitt). Da es einstellbare Parameter hat, nennen wir es eine *parametrische* Funktion oder ein *parametrisches* Modell. Wenn wir einige zweidimensionale Daten haben, können wir mit einem zufällig initialisierten Satz von Parametern beginnen, wie z. B. [m = 3,4, b = 0,3], und dann einen Trainingsalgorithmus verwenden, um die Parameter so zu optimieren, dass sie zu den Trainingsdaten passen, wobei der optimale Satz von Parametern nahe bei [m = 2, b = 1] liegt.

parametrische Modelle. Wir beginnen mit einem parametrischen Modell, dessen Parameter mit Zufallswerten initialisiert werden – dies erzeugt zufällige Klassenlabels für die Eingabebilder. Dann verwenden wir eine *Trainingsprozedur*, um die Parameter so anzupassen, dass die Funktion iterativ immer besser darin wird, die Bilder korrekt zu klassifizieren. Irgendwann werden die Parameter bei einem optimalen Satz von Werten liegen, was bedeutet, dass das Modell bei der Klassifizierungsaufgabe nicht mehr besser werden kann. Parametrische Modelle können auch für die *Regression* verwendet werden, bei der wir versuchen, ein Modell an einen Datensatz anzupassen, damit wir Vorhersagen für ungesehene Daten machen können (Bild 1.2). Ein ausgefeilterer Ansatz könnte sogar noch besser funktionieren, wenn er mehr Parameter oder eine bessere interne Architektur hätte.

Tiefe neuronale Netze sind beliebt, weil sie in vielen Fällen die genauesten parametrischen Machine-Learning-Modelle für eine bestimmte Aufgabe, wie z. B. die Bildklassifikation, sind. Dies ist weitgehend auf die Art und Weise zurückzuführen, wie sie Daten repräsentieren. Tiefe neuronale Netze haben viele Schichten (daher die „Tiefe“), was das Modell dazu veranlasst, geschichtete Darstellungen von Eingabedaten zu lernen. Diese Schichtdarstellung ist eine Form der Kompositionalität, was bedeutet, dass ein komplexes Datenstück als Kombination von elementareren Komponenten dargestellt wird, und diese Komponenten können weiter in noch einfachere Komponenten zerlegt werden, und so weiter, bis man zu atomaren Einheiten gelangt.

Die menschliche Sprache ist kompositorisch (Bild 1.3). Ein Buch besteht zum Beispiel aus Kapiteln, Kapitel aus Absätzen, Absätze aus Sätzen usw., bis man zu einzelnen Wörtern kommt, die die kleinsten Bedeutungseinheiten darstellen. Doch jede einzelne Ebene vermittelt Bedeutung – ein ganzes Buch soll Bedeutung vermitteln, und seine einzelnen Absätze sollen kleinere Punkte vermitteln.

Tiefe neuronale Netze können ebenfalls eine kompositorische Darstellung von Daten erlernen – sie können beispielsweise ein Bild als die Zusammensetzung primitiver Konturen und Texturen darstellen, die in elementare Formen zusammengesetzt werden, und so weiter, bis man ein vollständiges, komplexes Bild erhält. Diese Fähigkeit, Komplexität mit kompositorischen Darstellungen zu handhaben, macht Deep Learning so wirkungsvoll.

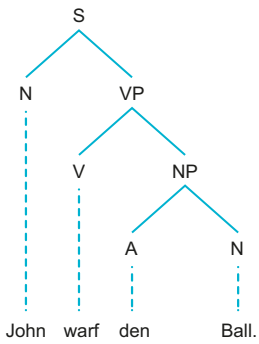

Bild 1.3 Ein Satz wie „John warf den Ball „ kann in immer einfachere Teile zerlegt werden, bis wir die einzelnen Wörter erhalten. In diesem Fall können wir den Satz (mit S bezeichnet) in ein Nomen (N) und eine Verbalphrase (VP) zerlegen. Die VP kann weiter in ein Verb „warf“ und eine Nominalphrase (NP) zerlegt werden. Die NP kann dann in die einzelnen Wörter „den“ und „Ball“ zerlegt werden.

1.2 Reinforcement Learning

Es ist wichtig, zwischen Problemen und ihren Lösungen zu unterscheiden oder, mit anderen Worten, zwischen den Aufgaben, die wir lösen wollen, und den Algorithmen, die wir zu ihrer Lösung entwerfen. Deep-Learning-Algorithmen können auf viele Problemtypen und Aufgaben angewendet werden. Bildklassifikations- und Vorhersageaufgaben sind häufige Anwendungen des Deep Learning, da die automatisierte Bildverarbeitung vor dem Deep Learning angesichts der Komplexität der Bilder sehr begrenzt war. Aber es gibt noch viele andere Aufgaben, die wir vielleicht automatisieren möchten, wie zum Beispiel Autofahren oder das Ausbalancieren eines Portfolios von Aktien und anderen Vermögenswerten. Zum Autofahren gehört ein gewisses Maß an Bildverarbeitung, aber noch wichtiger ist, dass der Algorithmus lernen muss, wie er *sich verhalten* muss, und nicht nur klassifizieren oder vorhersagen kann. Diese Probleme, bei denen Entscheidungen getroffen werden müssen oder bestimmte Verhaltensweisen umgesetzt werden müssen, werden kollektiv als *Steuerungsaufgaben* bezeichnet.

Reinforcement Learning ist ein generisches Framework für die Darstellung und Lösung von Steuerungsaufgaben, aber innerhalb dieses Rahmens können wir frei wählen, welche Algorithmen wir auf eine bestimmte Steuerungsaufgabe anwenden wollen(Bild 1.4). Deep-Learning-Algorithmen sind dafür prädestiniert, da sie in der Lage sind, komplexe Daten effizient zu verarbeiten. Aus diesem Grund konzentrieren wir uns auf *Deep* Reinforcement Learning, aber vieles von dem, was Sie in diesem Buch lernen werden, ist das allgemeine Reinforcement-Framework für Steuerungsaufgaben (siehe Bild 1.5). Dann werden wir uns ansehen, wie Sie ein geeignetes Deep-Learning-Modell modellieren können, das zum Framework passt und eine Aufgabe löst. Das bedeutet, dass Sie eine Menge über Reinforcement Learning lernen werden, und Sie werden wahrscheinlich auch einiges über Deep Learning lernen, was Sie bisher noch nicht wussten.

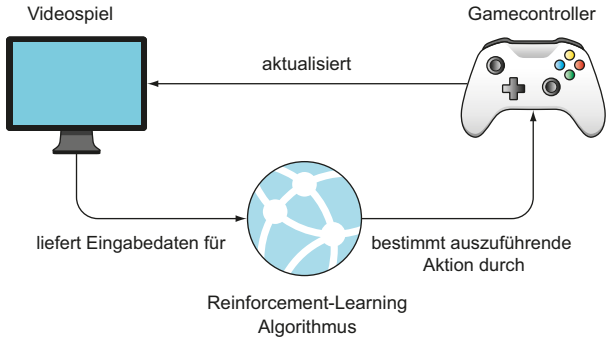

Bild 1.4 Im Gegensatz zu einem Bildklassifikator interagiert ein Reinforcement-Learning-Algorithmus dynamisch mit Daten. Er konsumiert kontinuierlich Daten und entscheidet, welche Aktionen durchgeführt werden sollen – Aktionen, die die ihm nachfolgend präsentierten Daten verändern. Ein Videospielbildschirm könnte die Eingabedaten für einen RL-Algorithmus darstellen, der dann unter Verwendung des Gamecontrollers entscheidet, welche Aktion zu ergreifen ist, und dies bewirkt eine Aktualisierung des Spiels (z. B. der Spieler bewegt sich oder feuert eine Waffe ab).

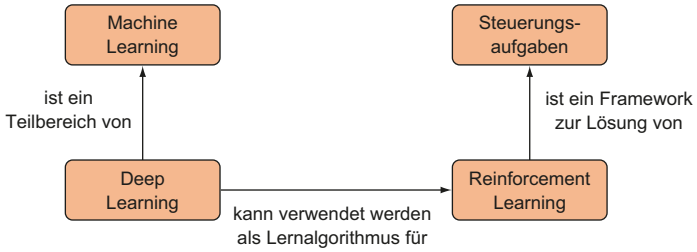

Bild 1.5 Deep Learning ist ein Teilbereich des Machine Learning. Deep-Learning-Algorithmen können zur Unterstützung von RL-Ansätzen zur Lösung von Steuerungsaufgaben verwendet werden.

Eine zusätzliche Komplexität beim Übergang von der Bildverarbeitung zum Bereich der Steuerungsaufgaben bringt das Element der Zeit mit sich. Bei der Bildverarbeitung trainieren wir normalerweise einen Deep-Learning-Algorithmus an einem festen Datensatz von Bildern. Nach einer ausreichenden Menge an Training erhalten wir in der Regel einen Hochleistungsalgorithmus, den wir auf einige neue, ungesehene Bilder anwenden können. Wir können uns den Datensatz als einen „Datenraum“ vorstellen, in dem ähnliche Bilder in diesem abstrakten Raum näher beieinander liegen und unterschiedliche Bilder weiter voneinander entfernt sind (Bild 1.6).

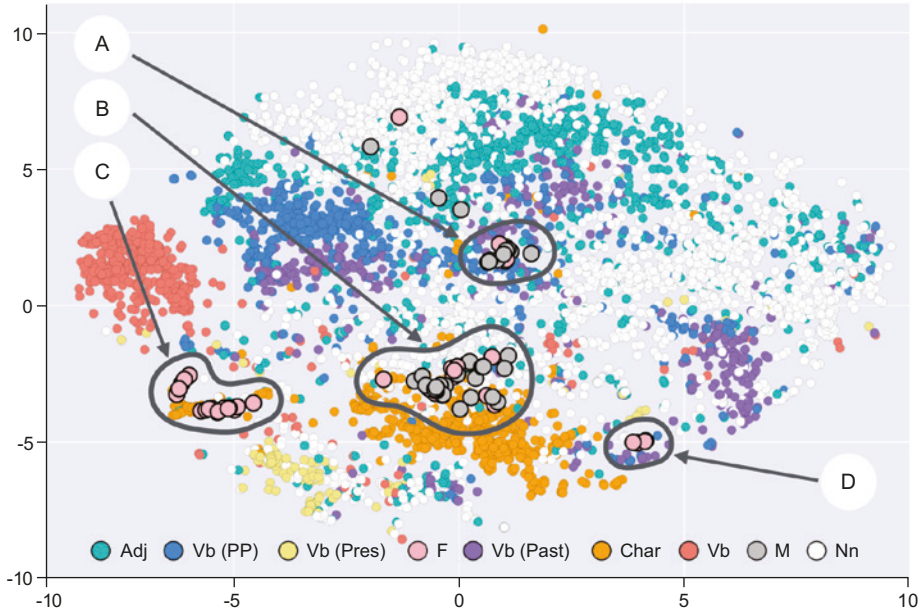

Bild 1.6 Diese grafische Darstellung von Wörtern in einem 2-D-Raum zeigt jedes Wort als einen farbigen Punkt. Ähnliche Wörter gruppieren sich und unähnliche Wörter liegen weiter auseinander. Daten leben natürlich in einer Art „Raum“, wobei ähnliche Daten näher beieinander leben. Die Labels A, B, C und D weisen auf bestimmte Cluster von Wörtern hin, die eine gemeinsame Semantik haben.

Bei Kontrollaufgaben haben wir in ähnlicher Weise einen Raum von zu verarbeitenden Daten, aber jede Dateneinheit hat auch eine zeitliche Dimension – Daten existieren in Zeit und Raum. Das bedeutet, dass das, was der Algorithmus zu einem bestimmten Zeitpunkt entscheidet, von dem beeinflusst wird, was zu einem früheren Zeitpunkt geschah. Dies ist bei gewöhnlicher Bildklassifizierung und ähnlichen Problemen nicht der Fall. Die Zeit macht die Trainingsaufgabe dynamisch – der Datensatz, auf dem der Algorithmus trainiert, ist nicht unbedingt fixiert, sondern ändert sich aufgrund der Entscheidungen, die der Algorithmus trifft.

Gewöhnliche bildklassifikationsähnliche Aufgaben fallen in die Kategorie des *Supervised Learning* (überwachtes Lernen), da der Algorithmus darauf trainiert wird, Bilder richtig zu klassifizieren, indem er die richtigen Antworten gibt. Der Algorithmus stellt zunächst nach dem Zufallsprinzip Vermutungen an, und er wird iterativ korrigiert, bis er die Features (Merkmale) im Bild lernt, die dem passenden Label entsprechen. Dies setzt voraus, dass wir bereits wissen, was die richtigen Antworten sind, was umständlich sein kann. Wenn Sie einen Deep-Learning-Algorithmus trainieren wollen, um Bilder von verschiedenen Pflanzenarten korrekt zu klassifizieren, müssten Sie mühsam Tausende solcher Bilder erfassen, jedem Bild manuell Klassenlabels zuordnen und die Daten in einem Format aufbereiten, mit dem ein Algorithmus für Machine Learning arbeiten kann, in der Regel in einer Art Matrix.

Im Gegensatz dazu wissen wir bei RL nicht bei jedem Schritt genau, was das Richtige ist. Wir müssen nur wissen, was das letztendliche Ziel ist und welche Dinge man vermeiden sollte. Wie bringt man einem Hund einen Trick bei? Man muss ihm Leckerbissen geben. In ähnlicher Weise trainieren wir einen RL-Algorithmus, indem wir ihm einen Anreiz geben, ein hochgestecktes Ziel zu erreichen, und ihn möglicherweise davon abhalten, Dinge zu tun, die wir nicht wollen, dass er sie tut. Im Falle eines selbstfahrenden Autos könnte das hochrangige Ziel sein, „unfallfrei von Ausgangspunkt A aus zu Punkt B zu gelangen“. Wenn er die Aufgabe erfüllt, belohnen wir ihn, und wenn er einen Unfall verursacht, bestrafen wir ihn. Wir würden das alles in einem Simulator machen, statt auf den realen Straßen, sodass wir ihn immer wieder versuchen lassen könnten an der Aufgabe zu scheitern, bis er lernt und belohnt wird.

TIPP

In der natürlichen Sprache bedeutet „Belohnung“ immer etwas Positives, während es im Jargon des Reinforcement Learning eine zu optimierende numerische Größe ist. Eine Belohnung kann also positiv oder negativ sein. Wenn sie positiv ist, entspricht sie dem natürlichsprachlichen Gebrauch des Begriffs, wenn sie negativ ist, entspricht sie dem natürlichsprachlichen Wort „Strafe“.

Der Algorithmus hat ein einziges Ziel – die Maximierung seiner Belohnung – und um dieses zu erreichen, muss er mehr elementare Fähigkeiten erlernen, um das Hauptziel zu erreichen. Wir können auch negative Belohnungen liefern, wenn der Algorithmus Dinge tut, die uns nicht gefallen. Da er versucht, seine Belohnung zu maximieren, wird er lernen, Aktionen zu vermeiden, die zu negativen Belohnungen führen. Aus diesem Grund wird es *Reinforcement Learning* genannt: Wir verstärken (*to reinforce*) bestimmte Verhaltensweisen entweder positiv oder negativ durch Belohnungssignale (siehe Bild 1.7). Das ist ganz ähnlich, wie Tiere lernen: Sie lernen, Dinge zu tun, mit denen sie sich gut oder zufrieden fühlen, und Dinge zu vermeiden, die Schmerzen verursachen.

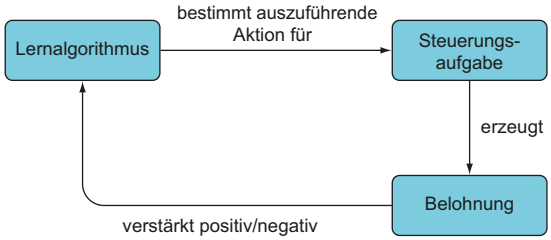

Bild 1.7 Im RL-Framework entscheidet eine Art Lernalgorithmus, welche Aktionen für eine Steuerungsaufgabe (z. B. das Steuern eines Staubsaugerroboters) durchzuführen sind, und die Aktion führt zu einer positiven oder negativen Belohnung, die diese Aktion positiv oder negativ verstärkt und somit den Lernalgorithmus trainiert.

1.3 Dynamische Programmierung versus Monte Carlo

Sie wissen jetzt, dass Sie einen Algorithmus trainieren können, um eine schwierige Aufgabe zu erledigen, indem Sie der Erledigung der Aufgabe eine hohe Belohnung (d. h. positive Verstärkung) zuweisen und Dinge, die wir nicht wollen, negativ bestärken. Lassen Sie uns das konkretisieren. Nehmen wir an, das vorrangige Ziel ist es, einen Staubsaugerroboter so zu trainieren, dass er sich von einem Raum in einem Haus zu seinem Dock, das sich in der Küche befindet, bewegt. Er hat vier Aktionen: nach links, nach rechts, vorwärts und rückwärts gehen. Zu jedem Zeitpunkt muss der Roboter entscheiden, welche dieser vier Aktionen er ausführen soll. Wenn er das Dock erreicht, erhält er eine Belohnung von +100, und wenn er auf seinem Weg an etwas stoßt, erhält er eine negative Belohnung von –10. Angenommen, der Roboter verfügt über eine vollständige 3-D-Karte des Hauses und hat die genaue Lage des Docks, aber er weiß immer noch nicht genau, welche Abfolge einfacher Aktionen er ausführen muss, um zum Dock zu gelangen.

Ein Ansatz zur Lösung dieses Problems heißt *dynamische Programmierung* (DP). Er wurde erstmals 1957 von Richard Bellman formuliert. Die dynamische Programmierung sollte besser als *Zielzerlegung* bezeichnet werden, da sie komplexe Probleme löst, indem sie diese in immer kleinere Teilprobleme zerlegt, bis sie zu einem einfachen Teilproblem gelangt, das ohne weitere Informationen gelöst werden kann.

Anstatt zu versuchen, mit einer langen Sequenz primitiver Aktionen zum Dock zu gelangen, kann der Roboter das Problem zunächst in „in diesem Raum bleiben“ versus „diesen Raum verlassen“ zerlegen. Da er über einen vollständigen Plan des Hauses verfügt, weiß er, dass er den Raum verlassen muss, da sich das Dock in der Küche befindet. Er weiß jedoch immer noch nicht, welche Abfolge von Aktionen es ihm erlaubt, den Raum zu verlassen, also gliedert er das Problem weiter auf in „sich auf die Tür zubewegen“ oder „sich von der Tür wegbewegen“. Da sich die Tür näher am Dock befindet und es einen Weg von der Tür zum Dock gibt, weiß der Roboter, dass er sich auf die Tür zubewegen muss, aber auch hier weiß er nicht, mit welcher Abfolge primitiver Aktionen er sich auf die Tür zubewegt.

Schließlich muss er entscheiden, ob er sich nach links, rechts, vorwärts oder rückwärts bewegen soll. Er sieht, dass sich die Tür vor ihm befindet, also bewegt er sich vorwärts. Er hält diesen Prozess aufrecht, bis er den Raum verlässt. Dann muss er noch einige Zielzerlegungen vornehmen, bis er zum Dock gelangt.

Das ist das Wesen der dynamischen Programmierung. Es handelt sich um einen generischen Ansatz zur Lösung bestimmter Arten von Problemen, die sich in Unter- und Teilprobleme zerlegen lassen, und es gibt Anwendungen in vielen Bereichen, darunter Bioinformatik, Wirtschaft und Informatik.

Um die dynamische Programmierung nach Bellman anwenden zu können, müssen wir in der Lage sein, unser Problem in Teilprobleme zu zerlegen, von denen wir wissen, wie sie zu lösen sind. Aber selbst diese scheinbar harmlose Annahme ist in der realen Welt schwer zu verwirklichen. Wie zerlegt man das hochgesteckte Ziel für ein selbstfahrendes Auto, „unfallfrei von Punkt A aus zu Punkt B zu gelangen“, in kleine, unfallfreie Teilprobleme? Lernt ein Kind das Laufen, indem es zuerst die leichteren Teilprobleme löst? In RL, wo wir oft nuancierte Situationen haben, die ein gewisses Zufallsprinzip enthalten, können wir die dynamische Programmierung nicht genau so anwenden, wie Bellman es beschrieben hat. Tatsächlich kann DP als ein Extrem eines Kontinuums von Problemlösungstechniken betrachtet werden, bei dem das andere Ende zufälliges Ausprobieren wäre.

Eine andere Möglichkeit, dieses Lernkontinuum zu betrachten, besteht darin, dass wir in einigen Situationen ein maximales Wissen über die Umgebung haben und in anderen ein minimales Wissen über die Umgebung, und wir müssen in jedem Fall unterschiedliche Strategien anwenden. Wenn Sie die Toilette in Ihrem eigenen Haus benutzen müssen, wissen Sie genau (na ja, zumindest unbewusst), welche Abfolge von Muskelbewegungen Sie von jeder Ausgangsposition aus (d. h. von der dynamischen Programmierung her) zur Toilette bringen wird. Das liegt daran, dass Sie Ihr Haus sehr gut kennen – Sie haben ein mehr oder weniger perfektes *Modell* Ihres Hauses im Kopf. Wenn Sie auf einer Party in einem Haus sind, in dem Sie noch nie zuvor gewesen sind, müssen Sie sich vielleicht lange umschauen, bis Sie die Toilette alleine finden (durch Trial-and-Error), weil Sie kein gutes Modell des Hauses dieser Person haben.

Die Trial-and-Error-Strategie wird im Allgemeinen zu den *Monte-Carlo-Methoden* gezählt. Eine Monte-Carlo-Methode ist im Wesentlichen eine Stichprobe nach dem Zufallsprinzip aus der Umgebung. Bei vielen Problemen der realen Welt haben wir zumindest ein gewisses Wissen über die Umgebungssituation, sodass wir am Ende eine gemischte Strategie aus Trial-and-Error und einer Ausnutzung dessen, was wir bereits über die Umgebung wissen, anwenden, um die einfachen Teilziele direkt zu lösen.

Ein etwas albernes Beispiel für eine gemischte Strategie wäre, wenn man Ihnen die Augen verbindet, Sie an einen unbekannten Ort in Ihrem Haus bringt und Ihnen sagt, Sie sollen das Badezimmer finden, indem Sie Kieselsteine werfen und auf das Geräusch achten. Sie könnten damit beginnen, das übergeordnete Ziel (das Badezimmer zu finden) in ein leichter zugängliches Unterziel zu zerlegen: Finden Sie heraus, in welchem Raum Sie sich gerade befinden. Um dieses Unterziel zu lösen, könnten Sie nach dem Zufallsprinzip ein paar Kieselsteine in zufällige Richtungen werfen und die Größe des Raumes abschätzen, was Ihnen genügend Informationen liefert, um daraus abzuleiten, in welchem Raum Sie sich befinden – sagen wir im Schlafzimmer. Dann müssten Sie zu einem weiteren Unterziel schwenken: zur Tür navigieren, damit Sie den Flur betreten können. Sie würden wieder anfangen, Kieselsteine

zu werfen, aber da Sie sich an die Ergebnisse Ihres letzten nach dem Zufallsprinzip ausgeführten Kieselsteinwurfs erinnern, könnten Sie Ihren Wurf auf weniger sichere Bereiche ausrichten. Durch Iteration dieses Vorgangs könnten Sie schließlich Ihr Badezimmer finden. In diesem Fall würden Sie sowohl die Zielzerlegung der dynamischen Programmierung als auch die Zufallsprinzipien der Monte-Carlo-Methode anwenden.

1.4 Das Reinforcement-Learning-Framework

Richard Bellman führte die dynamische Programmierung als eine allgemeine Methode zur Lösung bestimmter Arten von Kontroll- oder Entscheidungsproblemen ein, aber sie nimmt einen der äußersten Punkte des RL-Kontinuums ein. Bellmans wichtigerer Beitrag bestand wohl darin, zur Entwicklung des Standard-Frameworks für RL-Probleme beizutragen. Das RL-Framework ist im Wesentlichen der Kernsatz von Begriffen und Konzepten, in dem jedes RL-Problem formuliert werden kann. Damit steht nicht nur eine standardisierte Sprache für die Kommunikation mit anderen Ingenieuren und Forschern zur Verfügung, sondern wir sind auch gezwungen, unsere Probleme so zu formulieren, dass sie sich einer dynamischen Programmierung – ähnlich der Zerlegung von Problemen – unterwerfen lassen, sodass wir über lokale Unterprobleme hinweg iterativ optimieren und Fortschritte bei der Erreichung des globalen Ziels auf hoher Ebene erzielen können. Glücklicherweise ist das auch ziemlich einfach.

Um das Framework konkret zu veranschaulichen, betrachten wir die Aufgabe des Aufbaus eines RL-Algorithmus, der lernen kann, den Energieverbrauch in einem großen Rechenzentrum zu minimieren. Computer müssen kühl gehalten werden, damit sie gut funktionieren, sodass in großen Rechenzentren erhebliche Kosten durch Kühlsysteme entstehen können. Der schlichte Ansatz, ein Rechenzentrum kühl zu halten, bestünde darin, die Klimaanlage ständig auf einem Niveau zu halten, das dazu führt, dass kein Server jemals zu heiß läuft; dies würde kein ausgefallenes Machine Learning erfordern. Aber das ist ineffizient, und Sie könnten es besser machen, da es unwahrscheinlich ist, dass alle Server im Rechenzentrum zur gleichen Zeit heiß laufen und dass die Auslastung des Rechenzentrums immer auf dem gleichen Niveau ist. Wenn Sie die Kühlung dorthin lenken würden, wo und wann es am wichtigsten ist, könnten Sie dasselbe Ergebnis für weniger Geld erzielen.

Der erste Schritt im Framework ist, Ihr Gesamtziel zu definieren. In diesem Fall besteht unser Gesamtziel darin, die für die Kühlung aufgewendeten Mittel zu minimieren, mit der Einschränkung, dass kein Server in unserem Zentrum eine bestimmte Schwellentemperatur überschreiten darf. Obwohl dies zwei Ziele zu sein scheinen, können wir sie zu einer neuen zusammengesetzten *Zielfunktion* bündeln. Diese Funktion gibt einen Ertrag zurück, der angibt, wie weit wir bei der Erreichung der beiden Ziele angesichts der aktuellen Kosten und der Temperaturdaten für die Server vom Ziel abweichen. Die tatsächliche Zahl, die unsere Zielfunktion zurückgibt, ist nicht wichtig; wir wollen sie nur so niedrig wie möglich halten. Daher brauchen wir unseren RL-Algorithmus, um den Ertrag dieser Zielfunktion (Fehler) in Bezug auf einige Eingabedaten zu minimieren, die definitiv die laufenden Kosten und die

Temperaturdaten umfassen, aber auch andere nützliche Kontextinformationen enthalten können, die dem Algorithmus helfen können, die Nutzung des Rechenzentrums vorherzusagen.

Die Eingabedaten werden von der *Umgebung* erzeugt. Im Allgemeinen ist die Umgebung einer RL- (oder Steuerungs-) Aufgabe jeder dynamische Prozess, der Daten erzeugt, die für das Erreichen unseres Ziels relevant sind. Obwohl wir „Umgebung" als technischen Begriff verwenden, ist er nicht allzu weit von seiner alltäglichen Verwendung abstrahiert. Da Sie selbst ein Beispiel für einen sehr fortschrittlichen RL-Algorithmus sind, befinden Sie sich immer in irgendeiner Umgebung, und Ihre Augen und Ohren nehmen ständig Informationen auf, die von Ihrer Umgebung produziert werden, damit Sie Ihre täglichen Ziele erreichen können. Da die Umgebung ein *dynamischer Prozess* (eine Funktion der Zeit) ist, kann sie einen kontinuierlichen Strom von Daten unterschiedlicher Größe und Art produzieren. Um die Dinge algorithmusfreundlich zu gestalten, müssen wir diese Umgebungsdaten in diskrete Pakete bündeln, die wir den *Zustand* (der Umgebung) nennen, und sie dann in jedem ihrer diskreten Zeitschritte an unseren Algorithmus liefern. Der Zustand spiegelt unser Wissen über die Umgebung zu einem bestimmten Zeitpunkt wider, so wie eine Digitalkamera zu einem bestimmten Zeitpunkt einen einzelnen Schnappschuss einer Szene aufnimmt (und ein konsistent formatiertes Bild erzeugt).

Zusammenfassend lässt sich sagen, dass wir bisher eine Zielfunktion (Minimierung der Kosten durch Optimierung der Temperatur) definiert haben, die eine Funktion des Zustands (aktuelle Kosten, aktuelle Temperaturdaten) der Umgebung (des Rechenzentrums und aller damit verbundenen Prozesse) ist. Der letzte Teil unseres Modells ist der RL-Algorithmus selbst. Dies könnte *jeder* parametrische Algorithmus sein, der aus Daten lernen kann, um eine Zielfunktion zu minimieren oder zu maximieren, indem seine Parameter modifiziert werden. Es muss *kein* Deep-Learning-Algorithmus sein; RL ist ein eigenständiges Gebiet, das von den Belangen eines bestimmten Lernalgorithmus getrennt ist.

Wie wir bereits bemerkt haben, besteht einer der Hauptunterschiede zwischen RL (oder Supervised-Learning-Aufgaben im Allgemeinen) und gewöhnlichem Supervised Learning darin, dass der Algorithmus bei einer Kontrollaufgabe Entscheidungen treffen und Aktionen ausführen muss. Diese Aktionen haben eine kausale Auswirkung auf das, was in der Zukunft geschieht. Eine Aktion zu ergreifen ist ein Schlüsselwort im Framework, und es bedeutet mehr oder weniger das, was man erwarten würde. Jede Aktion ist jedoch das Ergebnis der Analyse des aktuellen Zustands der Umgebung und des Versuchs, auf der Grundlage dieser Informationen die beste Entscheidung zu treffen.

Das letzte Konzept im RL-Framework besteht darin, dass der Algorithmus nach jeder Aktion eine *Belohnung erhält*. Die Belohnung ist ein (lokales) Signal dafür, wie gut der Lernalgorithmus bei der Erreichung des globalen Ziels abschneidet. Die Belohnung kann ein positives Signal sein (d. h. es läuft gut, weitermachen) oder ein negatives Signal (d. h. das nicht tun), auch wenn wir beide Situationen als „Belohnung" bezeichnen.

Das Belohnungssignal ist das einzige Signal, an dem sich der Lernalgorithmus orientieren muss, während er sich in der Hoffnung auf eine bessere Leistung im nächsten Zustand der Umgebung aktualisiert. In unserem Rechenzentrumsbeispiel könnten wir dem Algorithmus eine Belohnung von +10 (ein willkürlicher Wert) gewähren, wenn seine Aktion den Fehlerwert reduziert. Oder, was vernünftiger ist, wir könnten ihm eine Belohnung gewähren, die proportional dazu ist, wie sehr er den Fehler verringert. Wenn er den Fehler erhöht, würden wir ihm eine negative Belohnung gewähren.

Schließlich sollten wir unserem Lernalgorithmus einen schickeren Namen geben, indem wir ihn *Agent* nennen. Der Agent ist der handlungs- oder entscheidungsorientierte Lernalgorithmus bei jedem RL-Problem. Wir können das alles zusammensetzen, wie in Bild 1.8 gezeigt.

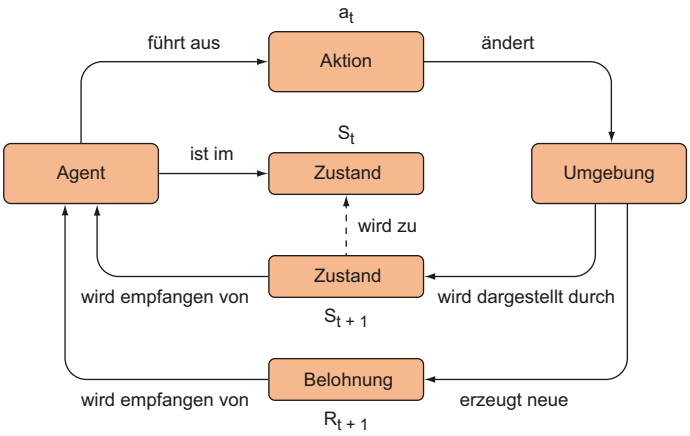

Bild 1.8 Das Standard-Framework für RL-Algorithmen. Der Agent führt eine Aktion in der Umgebung aus, wie z. B. das Ziehen einer Schachfigur, die dann den Zustand der Umgebung aktualisiert. Für jede Aktion, die er ausführt, erhält er eine Belohnung (z. B. +1 für das Gewinnen der Partie, –1 für das Verlieren der Partie, sonst 0). Der RL-Algorithmus wiederholt diesen Prozess mit dem Ziel, die Belohnungen langfristig zu maximieren, und er lernt schließlich, wie die Umgebung funktioniert.

In unserem Beispiel für ein Rechenzentrum hoffen wir, dass unser Agent lernt, wie wir unsere Kühlungskosten senken können. Wenn wir ihn nicht mit vollständigem Wissen über die Umgebung versorgen können, wird er ein gewisses Maß an Trial-and-Error anwenden müssen. Wenn wir Glück haben, lernt der Agent vielleicht so gut, dass er in einer anderen Umgebung als der, in der er ursprünglich trainiert wurde, eingesetzt werden kann. Da der Agent der Lernende ist, wird er als eine Art Lernalgorithmus implementiert. Und da es sich um ein Buch über *Deep* Reinforcement Learning handelt, werden unsere Agenten unter Verwendung von *Deep-Learning-Algorithmen* (auch bekannt als tiefe *neuronale Netze*, siehe Bild 1.9) implementiert. Aber denken Sie daran, dass es bei RL mehr um die Art des Problems und der Lösung geht als um einen bestimmten Lernalgorithmus, und Sie könnten sicherlich Alternativen zu tiefen neuronalen Netzen verwenden. Tatsächlich werden wir in Kapitel 3 damit beginnen, einen sehr einfachen nicht-neuronalen Netz-Algorithmus zu verwenden, und am Ende des Kapitels werden wir ihn durch ein neuronales Netz ersetzen.

Das einzige Ziel des Agenten besteht darin, seine erwartete Belohnung auf lange Sicht zu maximieren. Er wiederholt einfach diesen Zyklus: die Zustandsinformationen verarbeiten, entscheiden, welche Aktion zu ergreifen ist, sehen, ob er eine Belohnung erhält, den neuen Zustand beobachten, eine weitere Aktion durchführen und so weiter. Wenn wir all dies richtig einrichten, wird der Agent schließlich lernen, seine Umgebung zu verstehen und bei jedem Schritt verlässlich gute Entscheidungen zu treffen. Dieser allgemeine Mechanismus lässt sich auf autonome Fahrzeuge, Chatbots, Robotik, automatisierten Aktienhandel, Gesundheitswesen und vieles mehr anwenden. Einige dieser Anwendungen werden wir im nächsten Abschnitt und an weiteren Stellen im Buch erkunden.