Johann Wolfgang Goethe Universität
Frankfurt am Main
Fachbereich für Biologie und Informatik

Proseminar *„Komplexe Adaptive Systeme“*

Wintersemester 2004/2005

Das “Reinforcement Learning”-Problem

Alexander Schmid

schmid.kraenkle@t-online.de

Betreuung:
PD Dr. R. Brause

Frankfurt, 5.12.2004

1. Inhaltsverzeichnis

2. Einleitung

Diese Proseminar Ausarbeitung beschäftigt sich mit dem „Reinforcement Learning“-Problem. Das Ziel der Ausarbeitung ist es, eine kurze Einführung in das Thema zu geben und dabei die wesentlichen Aspekte des Themas anzusprechen und so zu erläutern, dass auch Nicht-Fachleute einen Einblick in das Thema erhalten können. Die Ausarbeitung wird bei weitem nicht alle Bereiche des Themas ansprechen. Insbesondere werden keine Methoden zur Lösung des „Reinforcement Learning“-Problems vorgestellt.
Der zweite Teil dieser Einleitung beschreibt zunächst die allgemeinen Ansätze des „Reinforcement Learnings“. Dabei werden, unter anderem, der Ursprung und die Alternativen des „Reinforcement Learnings“ betrachtet. Im dritten Teil der Einleitung wird das „Reinforcement Learning“-Problem formell definiert.
Im Kapitel 3 wird dann das Labyrinthbeispiel eingeführt. Anhand dieses Beispiels werden in den Kapiteln 4-8 die Elemente des „Reinforcement Learnings“ erläutert. In Kapitel 9 folgt ein kurzes Fazit und in Kapitel 10 die Literaturreferenzen.

Definieren wir nun, was gemeint ist wenn man von „Reinforcement Learning“ spricht. Mit „Reinforcement Learning“ wird im Allgemeinen eine bestimmte Art von Lernverfahren bezeichnet. Gegenüber dem allgemein bekannten Lernbegriff ist es in der Informatik notwendig, diesen Lernbegriff auf eine speziellere Art und Weise zu definieren (bzw. genauer gesagt, ihn geeignet einzuschränken). Aus diesem Grund ist es an dieser Stelle (bevor wir näher auf das Prinzip des „Reinforcement Learnings“ eingehen) sinnvoll, kurz zu erwähnen, wie dieser Lernbegriff letztendlich in der Informatik definiert wird.
Die Informatik beschäftigt sich in nicht unerheblichem Maße mit der algorithmischen Berechenbarkeit von Lösungen (und ihrer konkreten Umsetzung) zu konkreten Problemen. Es wäre also zweckmäßig konkrete Beschreibungen für das „Problem des Lernens“ zu formulieren, um dann wiederum konkrete Beschreibungen von Algorithmen zur Lösung dieses Problems zu finden. Diese Algorithmen wären dann also gerade diejenigen, die „das Lernen“ in der Informatik beschreiben und werden aus diesem Grund unter dem Oberbegriff des „maschinellen Lernens“ zusammengefasst. Mit anderen Worten bedeutet das, ein (maschinelles) Lernverfahren kann verwendet werden, um ein künstliches System die Lösung eines konkreten Problems „lernen“ zu lassen.
Die wohl bekannteste Unterkategorie der maschinellen Lernverfahren ist das „überwachte Lernen“ (supervised learning). Die Idee dieser Lernmethoden liegt (grob formuliert) darin, dass eine Lehrer existiert, der dem System zu einer bestimmten Eingabe automatisch die korrekte Ausgabe (also das zu lernende „Verhalten“ des Systems) liefert. Das System lernt also dadurch, indem es sich an diesen korrekten Ausgaben orientiert.
Lassen wir aber diesen Ansatz vorerst außer Acht und machen uns grundlegenden Gedanken über die Natur des Lernens. So ist zum Beispiel die Natur ein Bereich, in dem das Lernen einen völlig selbstverständlichen Vorgang darstellt. Ein Kleinkind braucht keinen Lehrer um festzustellen (bzw. zu lernen) welche Auswirkungen sein Verhalten auf seine Umgebung hat, genauso wie ein frisch geborenes Fohlen keinen Lehrer braucht um Laufen zu lernen. Sowohl das Kind als auch das Fohlen sind in der Lage ihre Umgebung zu beeinflussen und die (hervorgerufene) Veränderung auch wieder wahrzunehmen. Es existiert also eine direkte Verbindung zwischen dem Kind (bzw. dem Fohlen) und seiner Umgebung. Die Konsequenzen, die das Verhalten auf die Umgebung haben, werden dabei einem stetig wachsenden Erfahrungsschatz hinzugefügt. So entsteht nach und nach „Wissen“ über die Umgebung und über die Möglichkeiten diese Umgebung zu beeinflussen. Diese Art von Lernen hängt offensichtlich maßgeblich von der **Interaktion mit der Umgebung** und dem **Sammeln und Anwenden von Erfahrungen** ab. Außerdem ist das Ausprobieren neuer Verhaltensweisen („trial-and-error“) ein sehr wichtiger Aspekt dieser Art von Lernen. Ohne einen neuen Weg auszuprobieren wird man auch nie einen besseren finden. Das Erreichen eines Ziels wird über kleine (Teil-)Erfolge erzielt. Diese können auch als eine kleine Belohnung aufgefasst werden. Wichtig ist es aus der Erfahrung zu schöpfen und so auf diese kleinen Belohnungen hinzuarbeiten (auch wenn sie erst zu einem späteren Zeitpunkt erreicht werden)
Gerade diese Art zu Lernen bezeichnet man als „Reinforcement Learning“. Anstatt allerdings zu analysieren, wie Tiere oder Menschen lernen, wird es für uns darum gehen, Berechnungsmodelle und

Algorithmen zu finden, die in der Lage sind durch Interaktion mit einer Umgebung ein bestimmtes Ziel (also ein bestimmtes Verhalten) zu lernen. Alle wichtigen Aspekte (wie zum Beispiel das Maximieren der Belohnung, die Interaktion mit der Umgebung oder das Prinzip des „trial-and-error“) verlangen eine genau definierte Berücksichtigung in diesen Modellen. Oftmals wird dabei allerdings mit idealisierten Modellen und Verfahren gearbeitet, die weniger einen praktischen als vielmehr ein theoretischen (bzw. wissenschaftlichen) Wert haben. Dabei wird aber immer darauf geachtet, dass praktische Anwendungen auf der Basis dieser theoretischen Modelle realisierbar bleiben. Weitere Ziele werden sein die Effizienz verschiedener Ansätze zu vergleichen oder letztendlich konkrete Verfahren zur Lösung des „Reinforcement Learning“ Problems zu entwickeln (was allerdings den Rahmen dieser Ausarbeitung überschreitet).

Wie soeben erwähnt, ist also das Ziel, ein Modell zu entwickeln mit dem sich „Reinforcement Learning“ allgemein beschreiben lässt. Um „Reinforcement Learning“ zu beschreiben, denkt man wahrscheinlich im ersten Moment daran, Lösungswege (also genauer gesagt Reinforcement-Lernmethoden) anzugeben und diese dann so zu verallgemeinern, dass ihre Beschreibung einen größtmöglichen Teil aller denkbaren „Reinforcement Learning“-Anwendungen abdeckt.
Aus praktischen Erfahrungen hat sich aber herausgestellt, dass dieser Ansatz das „Reinforcement Learning“ zu beschreiben gänzlich ungeeignet ist. Es gibt viel zu viele und vor allem viel zu unterschiedliche Anwendungen des „Reinforcement Learnings“, als dass man einen allgemeinen Lösungsweg beschreiben könnte. So kann es zum Beispiel Szenarien geben, in denen zeitliche Vorgaben eher eine untergeordnete Rolle spielen, dafür aber zum Berechnen des Verhaltens ein sehr großer Rechenaufwand erforderlich ist. Denkt man zum Beispiel an einen Roboter, der lernen soll bestimmte Gegenstände korrekt zusammenzubauen, dann kommt es nicht darauf an, wie groß die Reaktionszeit des Roboters auf eine gestellte Aufgabe oder auf eine Veränderung seiner Umgebung ist, sondern wie exakt er am Ende die (komplexe) Aufgabe erfüllt hat. Im Gegensatz dazu kann man sich Szenarien vorstellen die genau gegenteilige Anforderungen besitzen. Nimmt man zum Beispiel eine Sortiermaschine die mit Hilfe von Druckluftventilen lernen soll, Gegenstände von einem laufenden Förderband auszusortieren (wie sie zum Beispiel in der Mülltrennung zur Anwendung kommt), dann darf die Geschwindigkeit der Berechung in keinem Fall der Geschwindigkeit des Förderbandes hinterherhinken. Eine Lernmethode die zwar exakt berechnet und dafür keine zeitlichen Einschränkungen hinnehmen muss, wäre hier also völlig ungeeignet. Viel eher sollte man nach einer Lösung suchen, die einfache Berechnungen zum Beispiel mit Hilfe von Approximation durchführt. Man erkennt also deutlich, dass es viele mögliche Lösungswege gibt, die so unterschiedlich von einander sein können, dass sie nicht unter einer allgemeinen Beschreibung vereinbar sind.
Da also die Lernmethoden zur Beschreibung des „Reinforcement Learnings“ versagen, muss man eine alternative Beschreibung entwickeln. Dazu bietet es sich geradezu an, das „Reinforcement Learning“ über sein grundlegendes Problem zu charakterisieren. Mit anderen Worten heißt dass, man beschreibt zuerst die einzelnen Elemente des „Reinforcement Learnings“, dann wie diese mit einander in Beziehung stehen und schließlich welche allgemeinen Eigenschaften sie erfüllen müssen. Man beschreibt also das Problem, welches es zu lösen gilt. Man bekommt so eine nützliche und allgemeingültige Beschreibung des „Reinforcement Learnings“, die man als „Reinforcement Learning“-Problem (das Thema dieser Ausarbeitung) bezeichnet. Lösungen des Problems bezeichnen wir ab sofort als „Reinforcement Learning“-Methoden. Man stellt fest, dass es durchaus möglich ist das „Reinforcement Learning“-Problem für die beiden genannten Beispiele zu formulieren. Beide Beispiele beinhalten ein System (im nachfolgenden Kapiteln als Agent bezeichnet) das Aktionen ausführen kann, um so die Umgebung zu beeinflussen.
Das Ziel der nachfolgenden Kapitel ist es jetzt also gerade diese Definition des „Reinforcement Learning“-Problems zu machen. Dazu werden wie angesprochen die Elemente und ihre Eigenschaften erläutert. Nicht selten werden mathematische Modelle verwendet, um eben diese Elemente und ihre Eigenschaften in einer präzisen Art und Weise zu beschreiben.

3. Das Labyrinthbeispiel

Bevor wir damit anfangen die einzelnen Elemente des „Reinforcement Learnings“ genauer zu analysieren, wollen wir ein einfaches Beispiel einführen. Anhand dieses Beispiels sollen in den späteren Kapiteln die Elemente des „Reinforcement Learnings“ nachvollzogen werden.
Gegeben sei ein einfaches Labyrinth, das einen Eingang und einen oder mehrere Ausgänge besitzt. Ein Roboter wird nun durch den Eingang in das Labyrinth geschickt und soll einen Ausgang aus dem Labyrinth finden. Skizieren wir zunächst dieses Szenario (der Roboter wird mit den roten Punkt, das Labyrinth durch die dicken schwarzen Linien dargestellt; es gibt 2 Ausgänge):

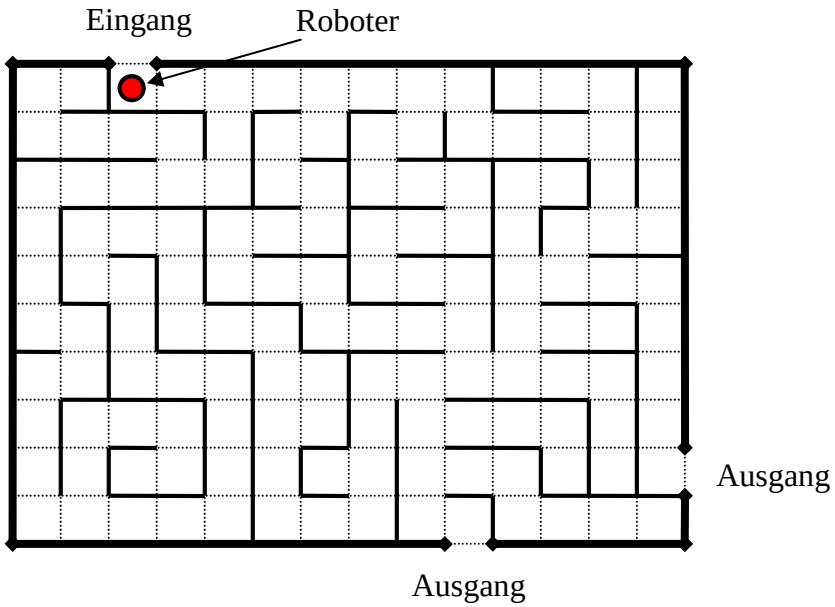


Abbildung 1: Roboter im Labyrinth

Vereinfacht nehmen wir an, dass sich der Roboter von Zelle zu Zelle (gestrichelten Kästchen) bewegen kann, vorausgesetzt es befindet sich keine Wand (dicke schwarze Linie) im weg.
Das Lernziel definieren wir nun wie folgt: Der Roboter soll lernen, so schnell wie möglich aus dem Labyrinth heraus zu finden. Mit anderen Worten soll er gerade den Weg finden (oder lernen), der die minimale Anzahl von Schritten hat, um einen Ausgang zu erreichen.

4. Der Agent und die Umgebung

Erinnern wir uns daran, dass es sich beim „Reinforcement Learning“ um Lernen durch Interaktion handelt. Das heißt also es existiert ein System, das in der Lage ist mit seiner Umwelt zu interagieren. Die Aufgabe besteht nun darin, diesen Vorgang formal zu definieren. Das was wir bisher einfach nur System genannt haben, wird ab jetzt formell als **der Agent** bezeichnet. Der Agent ist also der dynamische Teil des „Reinforcement Learning“ Problems. Er kann Entscheidungen treffen und besitzt somit ein Verhalten. Der Agent ist also genau der Teil, der lernen soll, sein Verhalten zu optimieren, um ein gegebenes Problem möglichst effizient löst. Wir wissen vom Agent außerdem, dass er unbedingt in der Lage sein muss mit seiner Umwelt zu interagieren. Diese Umwelt nennen wir ab jetzt formell **die Umgebung** (des Agenten). Die Umgebung beinhaltet gerade alles das, was nicht explizit dem Agenten zugeordnet werden kann, also alles was „außerhalb“ des Agenten liegt. Im Gegensatz zum Agenten wird die Umgebung als weitgehend statisch betrachtet.
Überlegen wir uns jetzt, wie sich diese formale Definition auf das Labyrinthbeispiel übertragen lässt. Bei dem Roboter handelt es sich offensichtlich um den Agenten, da es der Roboter ist, der lernen soll den Ausgang zu finden. Das Labyrinth ist dann die Umgebung des Agenten. Genauso gehören die Aus- und Eingänge des Labyrinths zur Umgebung.
Wir haben jetzt also die formale Definition des Agenten und der Umgebung auf das Labyrinthbeispiel übertragen.
Kommen wir nun zur Interaktion zwischen dem Agent und seiner Umgebung. Der Agent stellt (wie gesagt) den aktiven Teil der Interaktion dar. Das äußert sich dadurch, dass der Agent eine Aktion tätigt. Diese Aktion beeinflusst seine Umgebung. Genauer formuliert kann man sagen, dass sich durch eine Aktion der Zustand der Umgebung verändern kann. Der neue Zustand den die Umgebung annimmt, wird wiederum dem Agenten mitgeteilt (bzw. kann von diesem erfragt werden; das spielt für die Definition keine Rolle). Die Kommunikation die zwischen Agent und Umgebung stattfindet, kann durchaus als eine Art Signal- oder besser noch als Nachrichtenfluss aufgefasst werden. In diesem Sinne hält die Umgebung noch einen dritten „Nachrichtentyp“ für den Agenten bereit: Den „Reward“. Dabei handelt es sich um die schon in der Einführung kurz angesprochene Belohnung. Auf die „Rewards“ werden wir im Kapitel 6 noch genauer eingehen.
Kommen wir nun zu der Frage, ob sich diese eben beschriebene Interaktion zwischen Agent und Umgebung nicht noch formeller definieren lässt. Zunächst kann man feststellen, dass sich diese Interaktion in diskrete Zeitschritte $t = 0, 1, 2, ...$ aufteilen lässt. In jedem Zeitschritt erhält der Agent zuerst eine Beschreibung des aktuellen Zustandes der Umgebung $s_t \in S$, wobei S die Menge aller möglichen Zustände der Umgebung darstellt. Danach wählt der Agent auf der Basis dieses Zustandes eine Aktion $a_t \in A(s_t)$, wobei $A(s_t)$ die Menge aller Zustände beschreibt, die möglich sind wenn sich die Umgebung in Zustand s_t befindet. Einen Zeitschritt später kann der Agent dann, zusammen mit dem Folgezustand s_{t+1}, den „Reward“ $r_{t+1} \in R$ erhalten, wobei R wiederum die Menge aller „Rewards“ beschreibt. Die Abbildung 2 veranschaulicht diese Interaktion:

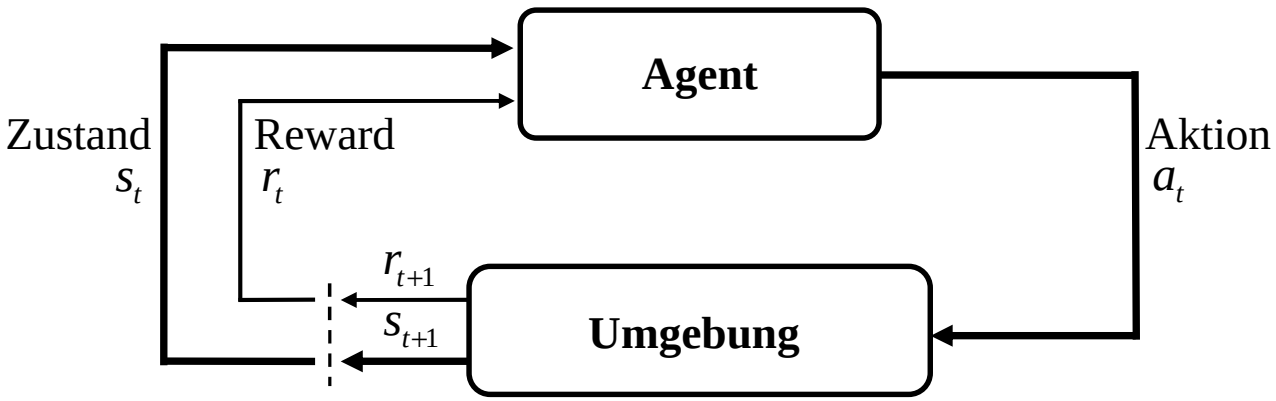


Abbildung 2: Das Interaktionsschema zwischen Agent und Umgebung

Gerade haben wir also zwei neue Elemente kennen gelernt: **Zustände** und **Aktionen**. Es ist an dieser Stelle sinnvoll kurz anzusprechen wie diese Zustände und Aktionen aussehen können.
Aktionen sind in der Regel genau die „Entscheidungen“, von denen wir wollen, dass der Agent ihre korrekte Anwendung lernt, um das spezifische Problem zu lösen. Die Aktionen sind dabei bei weitem weniger eingeschränkt, als man im ersten Augenblick annehmen möchte. So können sie zum Beispiel einfache low-level Entscheidungen sein, wie zum Beispiel das Starten eines Motors zum Bewegen eines Arms eines Roboters. An diesem Beispiel erkennt man aber auch sofort die Möglichkeit komplexere Aktionen zu definieren. So könnte die Aktion zum Beispiel problemlos lauten „bewege den Arm an die Position (x,y)“. Die Steuerung der einzelnen Motoren die für diese Aktion notwendig ist, wird jetzt also nicht mehr explizit vom Agenten ausgeführt, sie in diesem Szenario also in der Umgebung verankert. Diese Idee lässt sich problemlos weiterverfolgen, so kann man sich auch Szenarien vorstellen, in denen sich einzelne Aktionen aus tausenden weiteren Ausführungsschritten (in der Umgebung) zusammensetzen. Insbesondere kann man sich das sehr interessante Szenario vorstellen, dass ein Agent existiert der die high-level Entscheidungen trifft, welche dann von anderen Agenten (die gelernt haben diese high-level Entscheidungen mit Hilfe von low-level Entscheidungen umzusetzen) ausgeführt werden.
Ähnliches gilt für die Zustände. Diese können einfache Formen annehmen, wie zum Beispiel eine einfache Kombination von Sensormesswerten. Die Zustände können aber auch (genauso wie die Aktionen) sehr viel abstraktere Formen annehmen. Denkt man zum Beispiel an einen Roboter, der lernen soll Objekte zu erkennen. Dann könnte dieser durchaus einen Zustand geliefert bekommen, den er folgendermaßen identifiziert: „Ich erkenne ein Objekt, bin aber nicht sicher, ob es sich um ein Stuhl oder einen Tisch handelt“.
Kommen wir wieder zu unserem Labyrinthbeispiel und überlegen uns, wie die Aktionen und die Zustände bei diesem Beispiel aussehen. Der Agent (Roboter) kann sich von Zelle zu Zelle bewegen, also ist eine Aktion gerade die Bewegung zu einer Nachbarzelle. Ein Zustand ist dann also gerade die Zelle in der sich der Agent momentan befindet.
Die lässt sich folgendermaßen darstellen (Die Aktionen werden durch die Pfeile repräsentiert, der aktuelle Zustand durch die hervorgehobene Zelle).

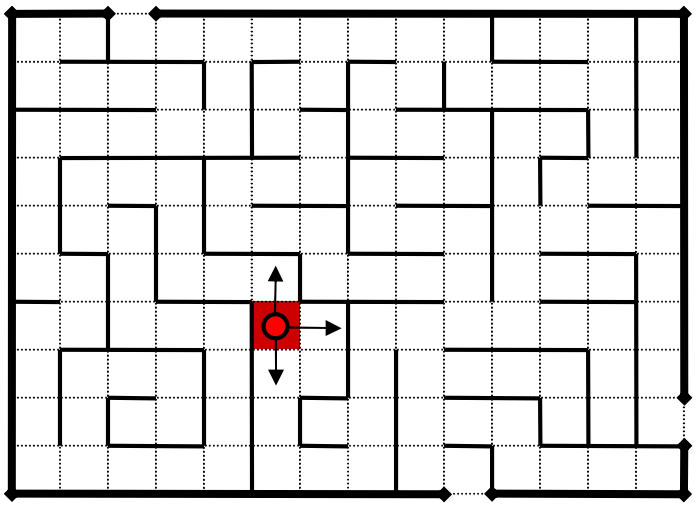


Abbildung 3: Aktion und Zustand im Labyrinthbeispiel

5. Die Policy

Nachdem wir gesehen haben, dass der Agent in der Lage ist Aktionen auszuführen, werden wir jetzt etwas genauer darauf eingehen, wie der Agent entscheidet welche Aktion er als nächstes ausführt. Zu diesem Zweck besitzt jeder Agent eine so genannte „Policy“. **Die „Policy“** ist grob formuliert, eine Abbildung von einem Zustand auf eine Aktion. Mit anderen Worten, es ist allein die „Policy“, die entscheidet, welche Aktion als Reaktion auf einen bestimmten Zustand gewählt wird. Damit ist die „Policy“ also das Herzstück des Agents, da sie direkt das Verhalten des Agents repräsentiert.
Erinnern wir uns: Unser Ziel ist es, dass der Agent ein bestimmtes Verhalten lernt, dann folgern wir daraus, dass die „Policy“ auf keinen Fall statisch sein darf. Vielmehr ist es gerade die „Policy“ die von Schritt zu Schritt neu angepasst werden sollte (da der Agent mit jedem Schritt etwas „schlauer“ wird, er lernt dazu). Diese Überlegung führt uns zu der folgenden formalen Definition: Die „Policy“ eines Agenten zu einem bestimmten Zeitschritt t wird mit π_t notiert.

Damit aber noch nicht genug. Wir hatten in der Einführung festgestellt, dass der Erfolg des „Reinforcement Learnings“ unter anderem maßgeblich davon ab abhängt wie gut Erfahrungen ausgenutzt werden, die der Agent in der Vergangenheit gemacht hat. Man stelle sich also eine „Policy“ vor, die basierend auf Erfahrungen immer den Zustand wählt, der die größte Chance besitzt, in naher Zukunft viel Reward zu bekommen. Solch eine „Policy“ nennt man eine greedy (gierige) „Policy“. Allerdings wird solch eine greedy „Policy“ immer wieder denselben Weg gehen, dieselben Entscheidungen treffen, was dazu führt, dass der Agent keine neuen Erfahrungen sammelt. Das ist aber sehr nachteilig für das Lernverhalten, denn es könnte ja sein das es eine noch bessere Folge von Entscheidungen gibt, die vom Agent nur noch nicht „entdeckt“ wurde. Es ist also notwendig, das der Agent von Zeit zu Zeit Entscheidungen trifft, die nicht notwendigerweise die höchsten Erfolgsausichten versprechen, sonder lediglich dem erforschen von neuen (vielleicht sogar besseren) Möglichkeiten dienen. Dieses Prinzip versteht man auch als das exploitation-exploration Problem (ausnutzen-entdecken Problem). Es ist verständlich das nur ein guter Kompromiss zwischen Ausnutzung und Entdeckung zu einem effizienten Lernen führen kann.
Um dieses Prinzip zu ermöglichen ist eine genauere Definition des Policy-Begriffs notwendig: Sei π_t eine „Policy“ eines Agenten im Zeitschritt t, dann bezeichnet $\pi_t(s,a)$ die Wahrscheinlichkeit dafür, dass die Aktion $a_t = a$ ist, wenn gilt, dass der aktuelle Zustand $s_t = s$ ist. Mit anderen Worten, ist diese Formulierung so zu verstehen, dass $\pi_t(s,a)$ die Wahrscheinlichkeit dafür ist, dass die „Policy“ die Aktion a als Reaktion auf den Zustand s wählt. $\pi_t(s,a)$ definiert also eine Wahrscheinlichkeitsverteilung (die einzelnen Wahrscheinlichkeiten müssen sich, über alle möglichen Aktionen in Zustand s, zu Eins summieren, veranschaulicht in Abbildung 2). Man erkennt also jetzt, dass für einen Zustand nicht zwingend immer dieselbe Aktion gewählt wird. Man kann so exploration-Entscheidungen in die Policy einbauen.

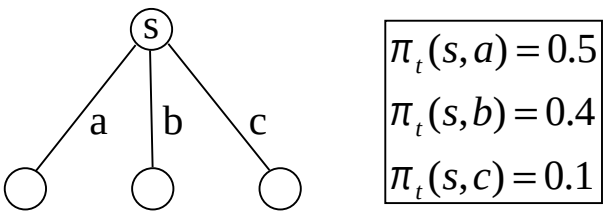


Abbildung 4: „Policy“ als Wahrscheinlichkeitsverteilung

Mit der Abbildung 4 führen wir auch gleichzeitig den Zustandsgraphen ein. Die Knoten sind die Zustände und die Kanten die Aktionen. Der Zustandsgraph wird verwendet um Wege über verschiedene Zustände (Zustandsketten) zu veranschaulichen.

Rewards und Returns

Nachdem wir in den beiden vorangegangenen Kapiteln beschrieben haben, wie der Agent mit seiner Umgebung interagiert und wie er mit Hilfe seiner „Policy“ Entscheidungen trifft, drängt sich nun die Frage auf, wie denn eigentlich das Ziel des Agents formuliert ist.
Wir wissen zwar, dass es das Ziel des Agents ist, ein Problem zu lösen, erinnern wir uns aber an die Beispiele aus den vorangegangenen Kapiteln, dann stellen wir fest, dass sich diese Probleme in ihrer Beschreibung mitunter sehr stark von einander unterscheiden können. Es muss also eine geeignete Verallgemeinerung gefunden werden, mit der alle diese unterschiedlichen Ziele beschrieben werden können. Genau diesen Zweck erfüllen **die „Rewards“**.
Bei einem „Reward“ handelt es sich um einen numerischen Wert $r_t \in R$, wobei R die Menge aller „Rewards“ darstellt. Ein „Reward“ stellt für den Agenten eine Belohnung (bzw. eine Bestrafung bei negativen Werten) dar. Generell kann man sagen, dass es das (einzige) Ziel des Agenten ist, positive Rewards (Belohnungen) zu bekommen und negative „Rewards“ (Bestrafungen) zu vermeiden.
Wie wir in Kapitel 4 schon kurz erwähnt hatten, bekommt der Agent die „Rewards“ aus seiner Umgebung. Es wäre sicher nicht falsch, sich vorzustellen, dass die „Rewards“ auf den Zuständen bereit liegen und vom Agent eingesammelt werden, wenn dieser einen Zustand besucht. Dabei enthält der Zustand s_t den „Reward“ r_t.
Im ersten Moment erscheint diese Methode in ihrer Flexibilität doch sehr eingeschränkt. Man fragt sich, wie es überhaupt möglich sein soll, mit „Rewards“ (Belohnungen oder Bestrafung für den Agenten) ein (möglicherweise sehr komplexes) Ziel zu modellieren.
Damit dieser Ansatz tatsächlich funktioniert, muss folgendes beachtet werden. Das **einzige Ziel** des Agenten muss es sein, über lange Sicht die „Rewards“ zu maximieren. Er muss also eine Folge von Entscheidungen finden (bzw. lernen), die ihm die maximal mögliche Summe an „Rewards“ einbringt. Wichtig hierbei ist, festzustellen, dass es nicht darum geht, den unmittelbar nächsten „Reward“ zu maximieren, sondern einen Weg zu gehen, der im Schnitt den größten „Reward“ abwirft.
Die „Rewards“ müssen jetzt in der Umgebung so platziert werden, dass sie den Agenten immer dann belohnen, wenn er einen Zustand erreicht, der eine Lösung des Problems darstellt. In der Praxis hat sich herausgestellt, dass das Maximieren des „Rewards“ gleichbedeutend damit ist, eine optimale Lösung des Problems zu finden.
Versuchen wir nun dieses Prinzip auf unser Labyrinthbeispiel anzuwenden. Erinnern wir uns daran, dass das Ziel des Agenten (Roboters) sein soll, so schnell wie möglich aus dem Labyrinth herauszufinden. Findet der Agent das Ziel, dann sollte dies belohnt werden, also setzen wir ein Reward auf die beiden Ausgänge. Allerdings soll der Agent so schnell wie möglich aus dem Labyrinth herausfinden, das heißt jeder Schritt muss eine kleine Bestrafung bekommen (negativen Reward), damit der Agent zusätzlich lernt, so wenig Schritte wie möglich zu machen. Also setzen wir die Rewards wie folgt:

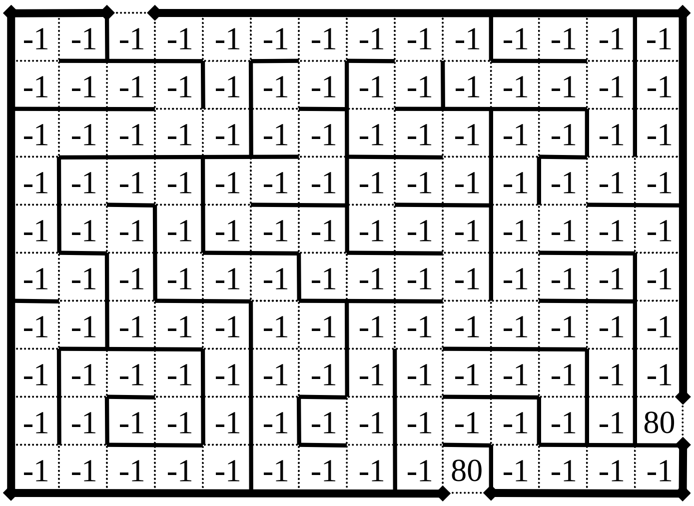


Abbildung 5: Rewards im Labyrinthbeispiel

Ein sehr gutes Beispiel dafür, dass man Rewards auch falsch setzen kann ist folgendes: Man hat einen Agent der Schach lernen soll. Kommt man nun auf die Idee Rewards auf das Schlagen anderer Figuren zu setzen, dann wird der Agent lernen viele Figuren zu schlagen, aber das Spiel wird er verlieren, weil das Schlagen von Figuren nicht notwendigerweise dem eigentlichen Ziel, nämlich dem Spielgewinn, dienlich ist.

Wir hatten im vorherigen Abschnitt also festgestellt, dass der Agent die Rewards über den gesamten (Entscheidungs-)Weg, denn er abläuft, maximieren soll. Um diese Sache noch formaler zu definieren, führen wir den Begriff des Returns ein. **Ein Return** R_t bezeichnet die Summe der Rewards, die ein Agent bekommt, wenn er in Zustand s_t startet, und nach T Zeitschritten sein Ziel erreicht:

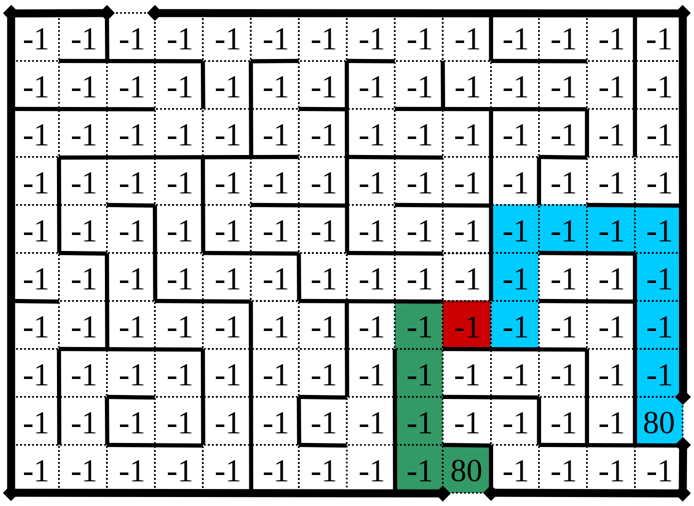
$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T$$

Es ist also gerade der Return (Summe über Rewards), den der Agent maximieren will. Eine Frage die sich jetzt unmittelbar aufdrängt, ist, was passiert, wenn der Agent sein Ziel gar nicht nach endlichen T Zeitschritten erreichen kann (denkt man zum Beispiel an die Steuereinheit der Produktionsanlage, die nie „fertig“ wird). Der Reward würde sich bis ins unendliche summieren. Folgende Formel für den Return löst dieses Problem:

$$R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k\, r_{(t+1)+k}$$

Hier passiert nun folgendes: Je weiter man in der Kette der Rewards in die Zukunft geht, desto stärker wird der Reward $r_{(t+1)+k}$ durch den Faktor γ^k abgeschwächt (es muss gelten $0 \le \gamma < 1$). Man weiß, das diese Summe für $\gamma < 1$ und $k \to \infty$ konvergiert. Es ist also sehr sinnvoll diese Summe in Szenarien zu verwenden bei denen es keine „letzten Zeitschritte“ gibt (man spricht auch von Episoden gegenüber kontinuierlichen Tasks).

Veranschaulichen wir nun die Bedeutung der Returns anhand des Labyrinthbeispiels. Zwei mögliche Returns (ausgehend von Zustand t, der rot hinterlegt ist) werden farbig (blau und grün) hervorgehoben. Die Richtung aus der der Agent gekommen ist (graue Felder), spielt keine Rolle, da ein Return immer in die Zukunft gerichtet ist:



-1-1-1-1+80 = 76

-1-1-1-1-1-1-1-1-1+80 = 71

Abbildung 6: Returns im Labyrinthbeispiel

Man erkennt sofort, dass der grüne Return einen größeren Wert besitzt und somit mehr Reward einbringt. Dieser Weg ist offensichtlich zu bevorzugen.

6. Markov-Eigenschaft und Markov-Decision-Process

In diesem Kapitel wird es darum gehen, welche Informationen die Zustände enthalten (bzw. enthalten sollten) und wie man diese Informationen ausnutzen kann.
Wie bereits besprochen, wird die Entscheidung für die Wahl einer Aktion auf der Basis eines einzigen Zustandes gemacht. Deswegen muss überlegt werden, ob ein Zustand auch wirklich genug Informationen liefert, damit diese Wahl sinnvoll getroffen werden kann.
Versuchen wir nun das ganze anhand eines Beispiels zu überlegen: Wir haben wieder den Agenten, der lernen soll Schach zu spielen. Angenommen, der Agent zieht mit einem Bauern vom Feld D3 nach D4. Jetzt bekommt er einen Folgezustand, der folgendermaßen formuliert ist: „Der 6. Bauer steht jetzt auf D4“. Dieser Zustand ist (nach unseren jetzigen) Erkenntnissen nicht unbedingt als „falsch“ zu bezeichnen, allerdings sieht man sofort, das der Agent ein großes Problem bekommt, wenn er (allein aufgrund dieses Zustandes) seine nächste Aktion wählen will. Dem Zustand fehlen offensichtlich wichtige Informationen die für eine sinnvolle Wahl einer Aktion dringend notwendig sind. Würde der Zustand zum Beispiel die Position aller Figuren beschreiben, dann gäbe es dieses Problem nicht.
Man kann an diesem Beispiel sehr schön erkennen, was wir von unseren Zuständen verlangen sollten. Ein Zustand sollte alle für uns wichtigen Informationen vorangegangener Zustände enthalten (bzw. zusammenfassen). Die wichtigen Informationen sind hierbei genau die, die wichtig sind, um in der Zukunft sinnvolle Entscheidungen zu treffen (in unserem Beispiel wären das gerade die Positionen der Figuren auf dem Schachbrett). Besitzt ein Zustand diese Eigenschaft, dann spricht man davon, dass er **die Markov-Eigenschaft** besitzt. Die Markov-Eigenschaft lässt sich auch mathematisch beschreiben. Betrachten wir dazu

$$Ws\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \mathsf{K}, s_0, a_0, r_0\}$$

Diese Formel beschreibt die Wahrscheinlichkeit dafür, dass der Folgezustand s_{t+1} gleich einem bestimmten Zustand s' wird und sich außerdem für diesen Zustand eine Reward von r ergibt. Wie man sieht ist diese Wahrscheinlichkeit abhängig (bedingte Wahrscheinlichkeit) von allen vorangegangenen Zuständen und Aktionen. Die Zustände besitzen also offensichtlich **keine** Markov-Eigenschaft. Würden sie die Markov-Eigenschaft besitzen, dann könnte die Formel für die Wahrscheinlichkeit folgendermaßen beschrieben werden:

$$Ws\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

Man erkennt diesmal klar, dass die Wahrscheinlichkeiten den bestimmten Zustand s' zu erreichen, ausschließlich vom vorherigen Zustand abhängt (dieser enthält ja schon alle Informationen), was gerade unserer Definition der Markov-Eigenschaft entspricht.
In der Praxis ist es oft schwierig Zustände zu finden, die eine **perfekte** Markov-Eigenschaft besitzen, es hat sich aber herausgestellt, dass auch Zustände, die nur annäherungsweise die Markov-Eigenschaft besitzen, zu sehr gute Ergebnissen führen.

Mit **Markov-Decision-Process** (Markov-Entscheidungs-Prozess, abgekürzt MDP) bezeichnet man nun ein „Reinforcement Learning“-Szenario, in dem alle darin definierten Zustände (annähernd) die Markov-Eigenschaft besitzen.
Im Zuge dieser Definition führen wir mit den Übergangswahrscheinlichkeiten zwei weitere mathematische Größen ein. Diese beiden Größen machen sich gerade die Markov-Eigenschaft des MDPs zunutze:

1. $P^a_{s\,s'} = Ws\{s_{t+1} = s' \mid s_t = s, a_t = a\}$

   Der Wert $P^a_{s\,s'}$ beschreibt die Wahrscheinlichkeit dafür, dass der Folgezustand s_{t+1} gleich einem bestimmten Zustand s' ist, unter der Bedingung, dass man aus dem Zustand s_t gekommen ist und die Aktion a_t gewählt hat. Hier kommt möglicherweise zum ersten Mal die Erkenntnis, dass für ein Zustand-Aktions-Paar der Folgezustand nicht notwendigerweise immer derselbe sein muss.

Abbildung 3 veranschaulicht die Bedeutung von $P^a_{s\,s'}$, dabei markieren a und b zwei mögliche Aktionen. Die schwarzen Punkte markieren dann die Möglichkeit durch eine Aktion in einen von mehreren möglichen Zuständen zu gelangen:

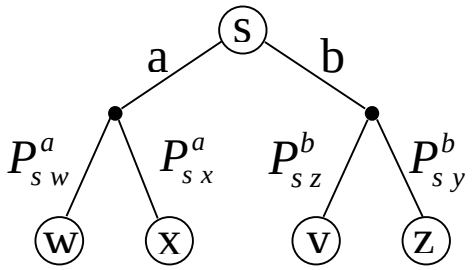


Abbildung 7: Überganswahrscheinlichkeiten für bestimmte Aktionen

2. $R^a_{s\,s'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$

Dieser Wert beschreibt den erwarteten Reward unter der Bedingung, dass man mit der Aktion a aus Zustand s gekommen, und in einem bestimmten Zustand s' gelandet ist. Dieser Wert ist nicht zu verwechseln mit der Definition des Returns. Ein Return ist die Aufsummierung von Rewards über eine Folge von Zuständen, während der soeben definierte Wert einen Erwartungswert für den nachfolgenden Reward darstellt.

Es hat sich in der Praxis herausgestellt, dass es genügt endliche MDPs zu untersuchen um allgemeingültige Aussagen über das „Reinforcement Learning“ treffen zu können. Deswegen werden uns diese beiden Werte im nachfolgenden Kapitel weiterhin beschäftigen.

7. Value Functions

Bisher hatten wir davon gesprochen, dass das Ziel des Agenten darin besteht, auf seinem Weg über eine Kette von Entscheidung die Summe der Rewards zu maximieren. Zu diesem Zweck hatten wir die Returns definiert.
Gehen wir davon aus, dass der Agent sich im Zustand s befindet. Die Policy π muss nun alle infrage kommenden Folgezustände bewerten. Das Problem der Policy ist es also, für jeden Folgezustand herauszufinden, wie groß sein erwarteter Return ist (also wie viel gesamt Reward man erwarten kann, wenn man einen bestimmten Weg weitergeht). Dazu bedient sich die Policy der Value Function.
Die Value Function ist gerade die Funktion, die für einen Zustand den Return zurückgibt, den man erwarten kann, wenn man bei diesem Zustand startet. Also mit anderen Worten die Summe der Rewards die man im Schnitt bekommt, wenn man in diesem Zustand startet; es wird also über alle möglichen Wege gemittelt:

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

Es wird also einfach der Erwartungswert des Returns, unter der Bedingung, dass $s_t = s$ ist, gebildet. Man sieht in der Formel, dass der Erwartungswert und die Value Function wieder von der Policy selbst abhängen. Das hat den Grund, dass es ja gerade die Policy ist, die das zukünftige Verhalten bestimmt, also kann man keinen Erwartungswert berechnen ohne dabei wieder die Policy zu verwenden (das wird auch noch mal im Folgendem deutlich).
Überlegen wir uns jetzt, wie dieser Erwartungswert genau berechnet werden kann:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} \quad (1)$$

$$= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{(t+1)+k} \mid s_t = s\right\} \quad (2)$$

$$= E_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{(t+2)+k} \mid s_t = s\right\} \quad (3)$$

$$= \sum_a \left(\pi(s,a) \sum_{s'} P^a_{s\,s'} \left(R^a_{s\,s'} + \gamma E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{(t+2)+k}\right\}\right)\right) \quad (4)$$

$$= \sum_a \left(\pi(s,a) \sum_{s'} P^a_{s\,s'} \left(R^a_{s\,s'} + \gamma V^{\pi}(s')\right)\right) \quad (5)$$

Diese Formel ist jetzt so zu interpretieren, dass über alle möglichen Aktionen, die Wahrscheinlichkeit für diese Aktion, multipliziert mit den erwarteten Return der möglichen Folgezustände, summiert wird. Das entspricht gerade der bekannten Definition des Erwartungswertes. Noch klarer wird es, wenn man die Formel anhand der folgenden Abbildung nachvollzieht:

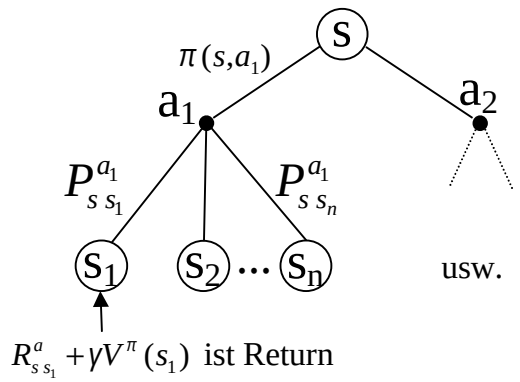


Abbildung 8: Berechnungsprinzip des Erwartungswertes

Die ebene besprochene Value Function wird auch **State Value Function** genannt, da sie gerade einem Zustand den erwarteten Return zuordnet (also diesen bewertet).
Neben den State Value Functions, gibt es auch noch die **Action Value Functions** $Q^{\pi}(s,a)$. Diese bewerten anstatt nur einem Zustand, einen Zustand s gepaart mit einer Aktion a. Das macht durchaus Sinn, da ja verschiedene mögliche Folgezustände für eine Aktion existieren können. In diesem Fall würde es dem Agenten nichts bringen den Folgezustand zu bewerten, weil er ja nicht weiß, in welchem er landen wird. Die Formel für die Action Value Function sieht dann folgendermaßen aus:

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

Diese Formel kann nach exakt demselben Prinzip wie die State Value Function aufgelöst werden (siehe [2]).
Die Action Value Function und die State Value Function hängen miteinander zusammen, da sie dasselbe machen, nämlich die Rewards rekursiv über den Zustandsgraphen zu verteilen. Der einzige Unterschied besteht darin, das sie an unterschiedlichen Punkte ausgewertet werden (für eine Zustande oder aber für Zustand-Aktions Paar).
Es bleibt weiter zu bemerken, dass es sich bei den Formeln lediglich um theoretische Überlegungen handelt, die nicht ohne weiteres in dieser Form angewendet werden können. In speziellen „Reinforcement Learning“ Methoden dienen diese Formeln als Ausgangsbasis, um praktisch anwendbare Berechnungsmodelle zu finden. Betrachtet man zum Beispiel die rekursive Eigenschaft der Value Functions, dann bietet sich hier das Prinzip des dynamischen Programmierens an. Betrachtet man andererseits den sehr hohen Rechenaufwand in Szenarien mit sehr vielen Zuständen, dann ist sicherlich ein approximatives Berechnungsmodelle vorteilhaft. In dieser Ausarbeitung werden wir auf spezielle Methoden für die Berechnung von Value Functions aber nicht weiter eingehen.

Kommen wir nun zu einem weiteren Aspekt der Value Functions, den **optimalen Value Functions**. Im Verlauf der Lernphase werden die Werte für Zustände, hinter denen viel Reward liegt, immer weiter ansteigen, die Werte für Zustände die wenig Reward in Aussicht stellen werden dementsprechend absinken. Die Value Functions konvergieren also gegen Werte, die dem maximal zu erwartenden Return entsprechen.
Eine Value Function, die diese Werte erzeugt, nennen wir optimale Value Function. Die optimale Value Function ist immer eindeutig. Haben wir (in der Praxis) nun Grund zur Annahme, dass sich die Value Function unseres Agents mittlerweile sehr nahe an die optimale Value Function genähert hat, dann können wir problemlos unsere Policy auf eine greedy Policy umschalten (wie wählen einfach immer die Aktion mit dem höchsten Wert). Die Policy nennen wir dann optimale Policy. Der Agent hat dann sozusagen ausgelernt.

8. Fazit

Im ersten Kapitel haben wir das Prinzip und den Ursprung des „Reinforcement Learnings“ kennen gelernt. Danach haben wir im Kapitel zwei den Versuch unternommen, das „Reinforcement Learning“-Problem formal zu beschreiben. Aus diesem Grund haben wir die einzelnen Elemente des „Reinforcement Learnings“ angesprochen. Wir haben außerdem alle wichtigen Eigenschafen dieser Elemente kennen gelernt.
Die gesammelten Erkenntnisse können uns nun als Basis dienen, um spezifische Reinforcment Learning Methoden zu verstehen oder solche sogar selbst zu entwickeln. Dabei handelt es sich zum Beispiel um Monte Carlo Methoden, Dynamisches Programmieren oder das Temporal Difference Learning.

10. Literaturreferenzen

[1] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, Chapter 1 - Introduction, MIT Press, Cambridge, MA, 1998

[2] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, Chapter 3 - The Reinforcement Learning Problem, MIT Press, Cambridge, MA, 1998