Hochschule Bremerhaven

Exposé für eine Bachelorarbeit zum Thema:

Reinforcement Learning

Theoretische Grundlagen der tabellarischen Lernmethoden und praktische Umsetzung am Beispiel eines Ameisen-Agentenspiels

Autor:	Jan Löwenstrom
Matrikelnr.:	34937
Erstprüfer:	Prof. Dr.-Ing. Henrik Lipskoch
Zweitprüfer:	Prof. Dr. Mathias Lindemann

17. Februar 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

//TODO

2 Grundlagen

Bei dem Bestärkenden Lernen (*Reinforcement Learning*) interagiert ein Softwareagent (*Agent*) mit seiner Umwelt (*Environment*), die wiederum nach jeder Aktion (*Action*) Feedback an den Agenten zurückgibt. Dieses Feedback wird als Belohnung (*Reward*) bezeichnet, einem numerischen Wert, der sowohl positiv als auch negativ sein kann. Der Agent beobachtet zudem den Folgezustand (*State*) in dem sich die Umwelt nach der vorigen Aktion befindet, um so seine nächste Entscheidung treffen zu können. Ziel des Agenten ist es eine Strategie (*Policy*) zu entwickeln, so dass die Folge seiner Entscheidungen die Summe aller Belohnungen maximiert.

2.1 Markov Entscheidungsprozess

Die Umwelt wird in dieser Arbeit als Markov'scher Entscheidungsprozess (*Markov Decision Process, MDP*) angesehen. Dieses Framework findet häufig Verwendung in der stochastischen Kontrolltheorie (Gosavi, 2009, S. 3) und bietet im Bezug auf das *Reinforcement Learning* Problem den mathematischen Rahmen, um u.a. präzise theoretische Aussagen treffen zu können Als *MDP* versteht sich die Formalisierung von sequentiellen Entscheidungsproblemen, bei denen eine Entscheidung nicht nur die sofortige Belohnung beeinflusst, sondern auch alle Folgezustände und somit auch alle zukünftigen Belohnungen (Sutton & Barto, 2018, S. 47). Ein Entscheidungsfinder muss somit das Konzept von verspäteten Belohnungen (*delayed rewards*) durchdringen. Vermeintlich schlecht erscheinende Entscheidungen in der Gegenwart können sich im Nachhinein als optimal herausstellen können, angesichts der gesamten Handlung. Ein Skat-Spieler könnte z.B. alle Trümpfe direkt am Anfang spielen, um einen sofortigen Vorteil zu erhalten. Für den Spielausgang ist es aber womöglich besser, die Trümpfe für einen späteren Zeitpunkt aufzubewahren und zu Beginn „schlechte " Entscheidungen zu treffen, die dazu führen, ein paar Stiche zu verlieren.

Probleme, die als *MDP* definiert werden, müssen die Markov-Eigenschaft erfüllen, da diese gewissermaßen als Erweiterung von Markov-Ketten zu betrachten sind, mit dem Zusatz von Aktionen und Belohnungen. Bei den sog. Markov-Ketten führt das System zufällige Zustandswechsel durch (Gosavi, 2009, S. 3). Dabei sind die Übergangswahrscheinlichkeiten zu den einzelnen Folgezuständen ausschließlich von dem aktuellen Zustand abhängig und nicht aufgrund des historischen Verlaufs (Gosavi, 2009, S. 3). Da die Markov-Eigenschaft eine essentielle Voraussetzung bei der Problemmodellierung ist, wird sie in Kapitel X näher erläutert. Die Beziehung zwischen Agent und Umwelt, kann durch folgendes Interface dargestellt werden (Sutton & Barto, 2018, S. 48):

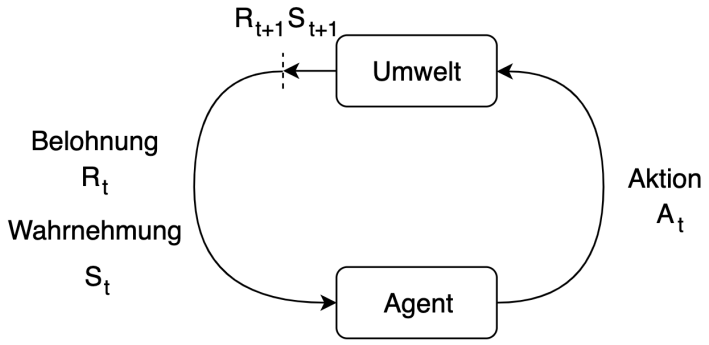

Abbildung 1: Agent-Umwelt Interface

Der Agent interagiert mit dem *MDP* jeweils zu diskreten Zeitpunkten $t = 0, 1, 2, 3, \ldots$. Zu jedem Zeitpunkt t beobachtet der Agent den Zustand seiner Umgebung $S_t \in \mathcal{S}$ und wählt aufgrund dessen eine Aktionen $A_t \in \mathcal{A}$. Als Konsequenz seiner Aktion erhält er einen Zeitpunkt später eine Belohnung $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ und stellt den Folgezustand S_{t+1} fest.

In der Literatur findet sich jedoch auch eine abweichende Definition im Bezug auf den Zeitpunkt der Belohnungsvergabe. Watkins (1989), Wiering & van Otterlo (2012) und Yu et al. (2009) z.B. binden die Belohnung R_t an das Zustands-Aktions-Paar (S_t, A_t). Die Definition R_{t+1} bei Aktion A_t von Sutton & Barto (2018) wird allerdings im Verlauf dieser Arbeit verwendet, da sie besser beschreibt, dass die Belohnung und der Folgezustand gemeinsam berechnet werden und einen Zeitpunkt später, nach Aktion A_t, für den Agenten sichtbar sind.

Das Zusammenspiel zwischen Agenten und MDP erzeugt somit folgende Reihenfolge (Sutton & Barto, 2018, S.48):

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \tag{2.1}$$

Wird einfach nur von *MDPs* gesprochen, ist die endliche Variante (*finite MDP*) gemeint, bei dem die Mengen der Zustände, Aktionen und Belohnungen $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ eine endliche Anzahl an Elementen besitzen. In diesem Fall haben die Zufallsvariablen R_t und S_t wohl definierte, diskrete Wahrscheinlichkeitsverteilungen, die nur von dem vorigen Zustand und der vorigen Aktion abhängig sind. Die Wahrscheinlichkeit, dass die bestimmten Werte für diese Variablen $s' \in \mathcal{S}$ und $r \in \mathcal{R}$ eintreten, für einen bestimmten Zeitpunkt t und dem vorigen Zustand s und Aktion a, kann somit durch folgende Funktion beschrieben werden (Sutton & Barto, 2018, S.48):

$$p(s', r \mid s, a) \doteq Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}, \tag{2.2}$$

für alle $s', s \in \mathcal{S}, r \in \mathcal{R}$ und $a \in \mathcal{A}(s)$. Diese Funktion p definiert die sog. Dynamiken (*Dynamics*) eines *MDP*. Sie ist eine gewöhnliche deterministische Funktion mit vier Parametern $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$. Das „|“ Zeichen kommt ursprünglich aus der Notation für bedingte Wahrscheinlichkeiten, soll hier aber andeuten, dass es sich um eine Wahrscheinlichkeitsverteilung handelt für jeweils alle Kombinationen von s und a (Sutton & Barto, 2018, S.49f):

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1 \ \forall s \in \mathcal{S}, a \in \mathcal{A}(s) \tag{2.3}$$

Ist das Entscheidungsproblem nicht stochastischer Natur, sondern deterministisch, so ist p immer nur für ein bestimmtes Triplet (s, a, r) für jedes $s' \in \mathcal{S}$ gleich 1, für alle andere jeweils 0. Mit anderen Worten, wird im Zustand s die Aktion a gewählt, führt dies in jedem Fall zu einem bestimmten Folgezustand s'.

Sutton & Barto (2018) erläutern, dass das MDP Framework als extrem flexibel gilt und es demzufolge auf die unterschiedlichsten Probleme angewendet werden kann. Sie führen weiter aus, dass es die nötige Abstraktion für Probleme bietet, bei denen unter Vorgabe eines Ziels mittels Interaktionen gelernt wird. Einzelheiten über das eigentliche Ziel, die Zustände oder die Form des Agenten sind dabei unerheblich. Letztendlich kommen die zwei Autoren zu dem Schluss, dass „jedes zielgerichtete Lernen auf drei Signale reduziert werden kann, die zwischen dem Agenten und der Umwelt ausgetauscht werden. Ein Signal repräsentiert die Entscheidung, die der Agent getroffen hat (die Aktion), ein Signal repräsentiert die Basis, auf der er zu dieser Entscheidung gekommen ist (der Zustand) und ein Signal definiert das zu erreichende Ziel (die Belohnung)“ (S. 50).

2.2 Markov-Eigenschaft und Zustandsmodellierung

Die Markov-Eigenschaft, obwohl relativ simpel, erhält ein eigenes Kapitel, da sie von fundamentaler Wichtigkeit ist und bei der Modellierung eines Reinforcement Learning Problems eine besondere Rolle spielt. Verbinden lässt sich dies sehr gut mit einem Einblick über die grundsätzliche Modellierung von Zuständen bei einem Reinforcement Learning Problem.

> The future is independent of the past given the present

Dieser Satz erscheint oft in Büchern und Papern, wenn es um die Markov-Eigenschaft geht, denn er fasst prägnat zusammen, was diese aussagt. Im Zusammenhang von MDPs lässt sich dieser Satz so übersetzen, dass ein Folgezustand nicht abhängig von Aktionen bzw. Zuständen in der Vergangenheit ist, sondern ausschließlich von dem aktuellen Zustand und der aktuell gewählten Aktion.

Sutton und Barto (2018) sehen die Markov-Eigenschaft als Einschränkung für die Zustände und nicht für den Entscheidungsprozess als solches. Ausschlaggebend ist, dass der Zustand, auf dessen Basis der Agent seine Entscheidung trifft, alle notwendigen Informationen der Vergangenheit beinhaltet, die für die Zukunft relevant sind (S.49). Die Umwelt ist somit nicht notwendigerweise gezwungen, Markov-konforme Zustände zu liefern. Brunskill (2019) wählt aufgrunddessen die Bezeichnung „Beobachtung“ (Observation O_t) als Feedback der Umwelt nach einer Aktion. Jene Beobachtungen können anschließend durch eine interne Repräsentation zu Markov-Zuständen verarbeitet werden, die dann dem Entscheidungsfinder zugrunde liegen.

Folgendes Beispiel, basierend auf der Vorlesung der Stanford-Professorin Brunskill (2019), liefert einen guten Einblick in die Zustandsmodellierung und der Problematik, die mit der Markov-Eigenschaft einhergeht.

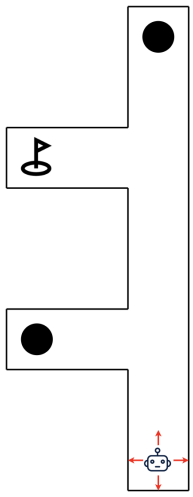

Abbildung 2: Zwei-Wege Beispiel zu der Markov-Eigenschaft

Gegeben ist ein beweglicher Roboter und eine Strecke mit zwei Korridoren. Der Roboter ist mit vier Sensoren ausgestattet, die jeweils eine Himmelsrichtung abdecken. Diese Sensoren sind in der Lage, angrenzende Wände zu erkennen und bilden den Zustand der Umwelt ab. Wahlweise ist der Zustand im Uhrzeiger definiert $\{N, O, S, W\}$, wobei 1 angibt, dass eine Wand erkannt wurde und 0, dass sich keine Wand in der unmittelbaren Nähe befindet. Es ergeben sich folglich 16 unterschiedliche Zustände, die der Agent unterscheiden und auf dessen Basis er Entscheidungen treffen kann (vier Aktionen: Fahrt in jeweils eine Richtungen). Der Roboter soll sein Ziel erreichen, markiert mit einer Flagge, ohne dabei in eine der beiden Fallen zu navigieren.

Eine potentielle Startposition, wie in Abb. 2 dargestellt, liefert somit den Zustand $\{0, 1, 1, 1\}$. Angenommen der Agent hat gelernt in diesem Zustand Richtung Norden zu fahren, dann ist der Folgezustand ebenfalls $\{0, 1, 1, 1\}$. Schließlich erreicht er den ersten Korridor. Der westlicher Sensor liefert folgerichtig 0 und der Zustand ist $\{0, 1, 1, 0\}$. Da der Agent nicht den ersten Korridor folgen darf, sondern dem zweiten, muss der Zustand $\{0, 1, 1, 0\}$ ebenfalls die Aktion „nach Norden fahren “ auslösen. Der Knackpunkt ist jedoch, dass der Zustand bei dem zweiten Korridor identisch mit dem Zustand bei dem ersten Korridor ist und der Agent somit keine Chance hat, zu unterscheiden, vor welchem er sich gerade befindet, siehe Abb. 3. Er würde ebenfalls, wie schon bei dem ersten Korridor, weiter nach Norden und letztendlich in die Falle fahren.

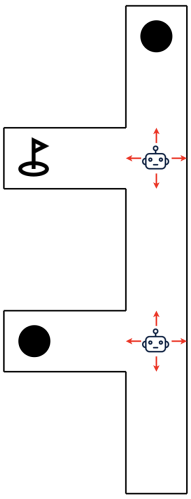

Abbildung 3: Zwei-Wege Beispiel Forts.

Bezogen auf diesen Entscheidungsprozess ist die Modellierung der Zustände über den Sensorinput alleine nicht markov. In der Theorie ist es jedoch möglich diesen Entscheidungsprozess als MDP umzumodellieren. Dabei werden die Sensordaten als Beobachtungen der Umwelt betrachtet und eine interne Repräsentation von Markov-Zuständen gepflegt. Möglich ist z.B. die gesamte Historie der Zustände und Aktionen zu speichern, damit der Roboter zurückverfolgen kann, wo er sich zur Zeit befindet. Ein Prozess als MDP zu definiert bedeutet aber gerade darauf zu verzichten, nämlich die gesamte Vergangenheit in einen Zustand zu verarbeitet. Denkbar ist auch, dass der Agent eine interne Repräsentation nach jeder Beobachtung pflegt und die Umwelt sukzessive nachbildet.

//TODO Schlussfolgerung; Modellierung
Letztendlich sollte

2.3 Belohnungen und Zielstrebigkeit

Das Besondere an dem Reinforcement Learning ist das Belohnungssignal (*Reward*), welches der Agent nach jeder Aktion erhält. Zu jedem diskreten Zeitpunkt wird dem Agenten eine Belohnung in Form einer einfachen Zahl $R_t \in \mathbb{R}$ zugestellt. Aufgabe eines jeden RL-Algorithmus ist es, die Summe aller gesammelten Belohnungen zu maximieren. Dabei ist entscheidend, dass der Fokus nicht ausschließlich auf die sofortigen Belohnungen gerichtet ist, sondern auf die erwartbare Summe aller Belohnungen über einen langen Zeitraum. Entscheidungen, die in der Gegenwart eine hohe sofortige Belohnungen versprechen sind verführerisch, können sich aber in der Zukunft in Bezug auf den gesamten Prozess als suboptimal herausstellen. (Sutton & Barto, 2018, S.53)

Eine Belohnungsfunktion wird in der Regeln von einem Menschen definiert und hat den größten Einfluss darauf, wie der Agent sich verhalten soll. Die Festlegung von Belohnung bei bestimmten Events ist die einzige Möglichkeit, die der Agent hat, zu verstehen, welches Ziel er verfolgen soll. Somit ist die Modellierung der passenden Belohnungsfunktion zur korrekten Abbildung der eigentlichen Aufgabenstellung von gravierender Bedeutung.

Grundsätzlich gibt es zwei Ansätze, um eine Belohnungsfunktion zu formulieren. Verständlich werden diese durch ein Beispiel, bei dem ein Agent lernen soll, eine Partie Schach zu gewinnen. Die erste Möglichkeit besteht darin, dem Agenten ausschließlich eine Belohnung aufgrund des Spielausgangs zu geben. Er erhält +1 wenn er gewinnt, -1 bei einer Niederlage und 0 bei Unentschieden (und jeder Aktion zuvor). Auf den ersten Blick erscheint dieser Ansatz trivial, ist aber die direkte Übersetzung des Ziels in eine Belohnungsfunktion. Die größte erwartbare Summe aller Belohnungen erhält der Agent nur, wenn er lernt, das Spiel zu gewinnen. Größter Nachteil dieser Methode ist allerdings, dass der Agent keinerlei Hilfe oder Richtung bei dem Erkunden des Spiels erhält. Je größer der Zustands- und Aktionraum ist, desto länger braucht er um überhaupt einmal ein Spiel gewinnen zu können und zu lernen, welche Aktionen vorteilhaft sind und welche nicht.

Um dem entgegenzuwirken, werden dem Agenten bei der zweiten Möglichkeit feingranularere Belohnungen mitgeteilt, statt diese ausschließlich auf das Endresultat zu reduzieren. Bestimmte Belohnungen zeigen dann, ob der Agent seinem Ziel näher gekommen ist oder eine ungünstige Entscheidung getroffen hat. Zum Beispiel könnte dem Agenten eine hohe Belohnung von +10 gegeben werden, wenn er die gegnerische Dame aus dem Spiel nimmt. Es ist auch denkbar, dass jede Spielfeldkonstellation bewertet wird. Dieser Ansatz benötigt somit spezielles Vorwissen über das Problem und kann sich zugleich sehr negativ auf das Verfolgen des eigentlichen Ziels auswirken. Der Agent könnte zum Beispiel nur lernen in jedem Spiel die Dame des Gegners zu schlagen und dabei trotzdem immer die Partie zu verlieren.

Die korrekte Modellierung der Belohnungsfunktion hat somit eine besondere Bedeu-

tung. Sutton & Barto (2018) sind der Meinung, dass ein Schachagent nur angesichts des Spielaussgangs bewertet werden sollte und nicht aufgrund von Zwischenzielen wie z.B. dem Herausnehmen einer gegnerischen Spielfigur oder der Kontrolle über das Zentrum des Spielfelds (S. 53).

Für eine korrekte Übersetzung der Aufgabenstellung zu einer geeigneten Belohnungsfunktion gibt es keine klaren, formalen Regeln. Ein Designer*in muss auf Erfahrungswerte und einen gewissen Grad an Kreativität zurückgreifen. Soll ein Agent z.B. ein Labyrinth durchlaufen und so schnell wie möglich hinausfinden, dann muss nach jeder Aktion eine negative Belohnung von -1 verteilt werden. Somit wird der Agent gezwungen, auf direktem Wege den Ausgang zu erreichen. Würde lediglich für das Erreichen des Ausgangs eine positive Belohnung vergeben werden, dann wäre die Summe aller Belohnungen für jede Abfolge von Aktionen gleich. Der Agent „trödelt". Hat er durch Zufall aus dem Labyrinth gefunden, so könnte er bei weiteren Durchläufen keinen effektiveren Weg finden, denn für ihn haben alle Aktionsfolgen den gleichen Nutzen.

Prinzipiell gilt, dass „das Belohnungssignal dazu dient, dem Agenten mitzuteilen *was* er erreichen soll, nicht *wie* er es erreichen soll" (Sutton & Barto, 2018, S. 54).

2.4 Gewinn und Episoden

In Kapitel 2.1 wurde gezeigt, dass die Interaktion eines Agenten mit seiner Umwelt als bestimmte Abfolge beschrieben werden kann (2.1). In ihr werden letztendlich alle Triples von Zustand, ausgeführter Aktion aufgrund dieses Zustands und anschließende Belohnung chronologisch aufgezeichnet. Ist diese Reihenfolge endlich, so wird sie auch als Episode (*Episode*) bezeichnet. Eine Episode fasst somit alle Informationen zusammen, die ein Agent erlebt, während er von einem beliebigen Startzustand aus anfängt die Umwelt zu erkunden. Das Ende einer Episode wird durch das Erreichen eines beliebigen Zielzustands erreicht. Ist eine Episode zu Ende, dann wird das Szenario zurückgesetzt und der Agent startet erneut im Startzustand. Episoden sind komplett unabhängig voneinander und erzeugen Abfolgen, die nicht durch vorrige Episoden beeinflusst sind.

Bisher wurde erwähnt, dass das Ziel eines Agenten sei, die Summe der zu erwartenden Belohnungen zu maximieren. Formal betrachtet, versucht er somit die Sequenz der Belohnungen, die er nach dem Zeitpunkt t erhält, den sog. erwarteten Gewinn (*Return*), zu maximieren. Im einfachsten Fall sieht G_t wie folgt aus, wobei T der finale Zeitstempel ist: (Sutton & Barto, 2018, S.55)

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{2.4}$$

Diese simple Addition von nachfolgenden Belohnungen ist ausreichend und sehr

praktikabel bei episodialen Problemszenarien. Jedoch ungeeignet für Probleme, bei denen keine klaren Endzustände definiert sind und daher einen sog. unendlichen Zeithorizont (infinite horizon) besitzen. Folglich ist $T = \infty$, was wiederum bedeutet, dass der Gewinn unendlich ist.

Um episodiale und kontinuierliche Aufgaben im Bezug auf den Gewinn zu vereinheitlichen, wird das Konzept der Diskontierung (*discounting*) verwendet. Dabei gibt der Parameter γ, $0 \leq \gamma \leq 1$, Auskunft darüber, wie die Gewichtung zwischen sofortigen und zukünftigen Belohnungen verteilt ist. Der zukünftige diskontierte Gewinn, der durch die Aktion A_t maximiert werden soll, berechnet sich somit wie folgt (Sutton & Barto, 2018, S.55):

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.5}$$

Eine wichtige Erkenntnis ist, dass Gewinne aufeinanderfolgender Zeitpunkte in Verbindung stehen. Vor allem Algorithmen, die nach jedem Zeitstempel updaten, profitieren von dieser Eigenschaft. Sie verwenden den geschätzten Gewinn des Folgezustands, also G_{t+1}, zur Berechnung von G_t, dem geschätzten Gewinn des aktuellen Zustands. Dieses Verfahren, bei dem ein Schätzwert aufgrund eines anderen Schätzwertes geupdated wird, wird auch als *bootstrapping* bezeichnet.

Durch simple Umformung wird der Zusammenhang von Gewinnen deutlich (Sutton & Barto, 2018, S.55):

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.6}$$

Ist $\gamma = 0$, dann wählt der Agent seine Aktionen ausschließlich aufgrund der sofortigen Belohnung R_{t+1}. Je näher γ an 1 ist, desto „weitsichtiger“ wird der Agent, da der Gewinn für den Zeitpunkt t sich zusätzlich aus zukünftigen Belohnungen zusammensetzt. $\gamma = 1$ führt zu der gleichen Summe wie (2.4) und wird bei Problemen bestimmt, die Episoden erzeugen. Dadurch trifft der Agent seine Entscheidungen immer aufgrund jeglicher Konsequenzen in der Zukunft bzw. bis zum Ende der jeweiligen Episode. Um zu erreichen, dass die unendliche Summe in (2.5) bei kontinuierlichen Aufgaben einen endlichen Wert annimmt, muss $\gamma < 1$ gegeben sein.

Probleme mit unendlichem Zeithorizont können durch die Vergabe einer künstlichen Schranke zu einer episodialen Aufgabe umformuliert werden. Denkbar z.B. durch die Festlegung der maximalen Anzahl an Aktionen oder besuchten Zustände.

//TODO TD-Episodic tasks?! Weglassen?

Die Algorithmen der Monte-Carlo-Methoden, die in Kapitel X vorgestellt werden, können ausschließlich auf Basis von Episoden lernen. Jedoch existieren auch Methoden,

wie das Temporal-Difference-Learning (Kapitel X), die neben dem episodialen Lernen, zusätzlich in der Lage sind, mit kontinuierlichen Aufgaben zurechtzukommen.

2.5 Strategie und Nutzenfunktion

Fast alle Lernalgorithmen des Reinforcement Learning versuchen eine sog. Nutzenfunktion (*Value Function*) zu schätzen. Diese Funktion sagt aus, „wie gut“ es ist, dass sich der Agent in einem bestimmten Zustand befindet oder eine bestimmte Aktion in einem Zustand auszuführen. Dabei bezieht sich das „wie gut“ darauf, welche Belohnungen in der Zukunft erwartbar sind, also wie groß der erwartete Gewinn ist. Zukünftige Belohnungen sind natürlicherweise abhängig davon, wie sich der Agent verhalten bzw. welche Entscheidungen er in der Zukunft treffen wird. Nutzenfunktion sind deshalb immer in Bezug auf eine bestimmte Strategie definiert (Sutton & Barto, 2018, S. 58).

Eine Strategie (*Policy*) kann als Abbildung verstanden werden, die jedem Zustand eine diskrete Wahrscheinlichkeitsverteilung über Aktionen zuordnet. Folgt der Agent einer Strategie π zum Zeitpunkt t, dann gibt $\pi(a \mid s)$ an, mit welcher Wahrscheinlichkeit $A_t = a$ ausgeführt wird, wenn $S_t = s$ (Sutton & Barto, 2018, S. 58). Neben solchen stochastischen Strategien, existieren auch simplere, deterministische Strategien, die jedem Zustand nur eine Aktion zuordnen, $\pi(s) = a$ (Brunskill, 2019).

Wie anfangs erwähnt, gibt es zwei Varianten der Nutzenfunktion. Die erste sagt aus, wie groß der Erwartungswert des Gewinns für den Zustands s ist, wenn in diesem gestartet und anschließend aufgrund der Strategie π gehandelt wird. Dieser *Zustands-Nutzen* kann für alle $s \in \mathcal{S}$ folgendermaßen definiert werden (Sutton & Barto, 2018, S. 58):

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s] \tag{2.7}$$

Die zweiten Variante gibt Auskunft darüber, wie groß der Nutzen ist, wenn im Zustand s gestartet, daraufhin die Aktion a ausgeführt und anschließend der Strategie π gefolgt wird. q_π wird auch als *Aktions-Nutzenfunktion* für die Strategie π bezeichnet und wird formal ausgedrückt durch (Sutton & Barto, 2018, S. 58):

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a] \tag{2.8}$$

//TODO der Erwartungswert auf ? //TODO funktionsapproximation hier?

2.6 Optimalität

Ein Reinforcement Learning Problem zu lösen bedeutet, eine Strategie zu finden, die den größten Gewinn bringt. Dabei lassen sich Strategien vergleichen, insofern, dass

eine Strategie besser ist als eine andere, wenn der erwartete Gewinn für alle Zustände größer oder gleich ist (Sutton & Barto, 2018, S. 62f). Mit anderen Worten, $\pi \geq \pi'$ gilt, wenn $v_\pi(s) \geq v_{\pi'}(s)$ für alle $s \in \mathcal{S}$. Es existiert mindestens eine Strategie die besser oder gleich gegenüber allen anderen Strategien ist. Diese ist die optimale Strategie π_*. Optimale Strategien teilen die selbe (optimale) Zustands-Nutzenfunktion v_* und (optimale) Aktions-Zustands-Nutzenfunktion q_* (Sutton & Barto, 2018, S. 62f):

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.9}$$

$$q_*(s,a) = \max_\pi q_\pi(s,a) \tag{2.10}$$

Nutzenfunktionen sind, wie im vorigen Kapitel (2.5) erläutert, immer abhängig von einer bestimmten Strategie, da diese die gesammelte Erfahrung beeinflusst und somit auch die erwarteten, geschätzten Gewinne. Die optimale Nutzenfunktion kann jedoch auch ohne Referenz auf eine bestimmte Strategie beschrieben werden, da der Gewinn eines Zustands unter einer optimalen Strategie gleich dem erwarteten Gewinn für die beste Aktion in diesem Zustand ist. $v_*(s)$ referenziert somit $v_*(s')$, den besten Folgezustand, wodurch eine rekursive Beziehung zustande kommt. Eine optimale Nutzenfunktion v_* kann formal folgendermaßen beschrieben werden (Sutton & Barto, 2018, S. 63):

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}\,\pi_*[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')] \end{aligned} \tag{2.11}$$

Diese Gleichung ist die sog. *Bellman Optimality Equation* und lässt sich auch als Gleichungssystem interpretieren, welches eine Gleichung pro Zustand bestitzt. Für ein Problem mit n Zuständen ergeben sich somit n Gleichung mit n Unbekannten (Sutton & Barto, 2018, S. 63). Eine Berechnung der optimalen Nutzenfunktion ist folglich in der Theorie möglich, jedoch muss die Übergangsfunktion p bekannt sein. Ist p gegeben wird von einem perfekten Modell gesprochen, eine Voraussetzung, die nicht immer erfüllt ist.

Selbst wenn die Dynamiken der Umwelt bekannt sind, kann die benötigte Rechenzeit zur Lösungen jedoch utopische Ausmaße annehmen. Bei einem Spiel wie „Backgammon“ sind die Regeln klar definiert, ein perfektes Modell ist demzufolge vorhanden, aber es existieren 10^{23} Zustände, was die mathematische Berechnung von v_* mittels der *Bellman Optimality Equation* praktisch unmöglich macht (Sutton & Barto, 2018, S. 66). Dennoch stellt sie ein wichtiges Fundament für das Reinforcement Learning dar, da die

meisten Reinforcement Learning Algorithmen als annäherungsweises Lösungsverfahren verstanden werden können (Sutton & Barto, 2018, S. 66).

Methoden des Reinforcement Learnings, die die Umwelt als Blackbox betrachten, werden auch als *model-free* beschrieben. Sie benötigen keinen Zugriff auf die Übergangsfunktion p, denn es wird ausschließlich aufgrund der erhaltenen Belohnungen und Beobachtungen gelernt wird. Hierbei bezieht sich der Lernprozess darauf, wie nah die geschätzte Nutzenfunktion der aktuellen Strategie π an v_* bzw. q_* ist.

Die optimale Strategie lässt sich leicht ermitteln, wenn eine optimale Nutzenfunktion gegeben ist. Ist zum Beispiel v_* gegeben und befindet sich der Agent in Zustand s, dann muss er eine Aktion vorrausschauen, um den Folgezustand s' zu finden, der den maximalen Nutzen hat. Dieses Vorrausschen benötigt jedoch ein perfektes Modell der Umgebung, um die Übergänge für jede Aktion zu berechnen. Das ist der ausschlaggebende Grund, warum bei *model-free* Methoden q_* berechnet wird. Denn dieser Nutzen umfasst implizit den Nutzen der Folgezustände für jede Aktion. Infolgedessen muss der Agent im Zustand s nur schauen, welche Aktion a und somit welches Zustands-Aktions-Paar den größten Nutzen hat und wählt genau jene Aktion.

2.7 Exploration-Exploitation Dilemma

Durch die Vergabe von Belohnungen und dem übergeordnetem Ziel eines Agenten so viele Belohnungen wie möglich zu sammlen, ergibt sich eine spezielle Problematik bei dem Reinforcement Learning, die bei anderen Lernmethoden des Maschinellen Lernens nicht vorhanden ist. Um den Gewinn zu maximieren, muss der Agent auf der einen Seite Aktionen bevorzugen, die sich in der Vergangenheit bereits als gut herausgestellt haben. Er nutzt unvollständige Erfahrung, um so ausbeuterisch wie möglich zu handeln (*Exploitation*). Andererseits ist der Agent dazu gezwungen, neue Aktionen auszuprobieren, damit der Zustands- und Belohungsraum weiter erkundet wird, um bessere oder sogar optimale Entscheidungen in der Zukunft treffen zu können (*Exploration*).

Das Dilemma besteht darin, dass weder Exploration noch Exploitation ausschließlich verfolgt werden kann, ohne dabei die eigentliche Lernaufgaben zum Scheitern zu bringen. Dieses Exploration-Exploitation Dilemma wird von Mathematikern seit Jahrzehnten intensiv untersucht, bleibt allerdings ungelöst (Sutton & Barto, 2018, S. 3). Grundsätzlich muss ein Entscheidungsfinder eine Reihe von unterschiedlichen Aktionen ausführen und zunehmend jene bevorzugen, die sich als gut herausstellen. Dementsprechend muss eine Balance zwischen den beiden Prozessen gefunden werden. Eine Strategie, die ausschließlich ausbeuterisch handeln, wird auch als gierig (*greedy*) bezeichnet. Der Begriff „gierig“ bezeichnet in der Informatik eine Vorgehenweise, bei der immer die, zum Zeitpunkt der Wahl, vermeintlich beste Entscheidungen getroffen wird (Möller & Struth, 2004, S. 203). Dabei wird die Suche nach einem globalen

Maximum komplett vernachläsigt. Auf den Kontext des Reinforcement Learning übertragen, wählt eine gierige Strategie für jeden Zustand immer jene Aktion, die den derzeitigen größten Nutzen besitzt. Nur wenn die Nutzenfunktion zu einer optimalen Nutzenfunktion konvergiert ist, ist eine gierige Nutzenfunktion auch gleichzeitig die optimale Strategie. Um jedoch die optimale Nutzenfunktion zu finden, muss erkundet werden.

Ein trivialer, aber denoch effektiver Ansatz ist es, die meiste Zeit gierig zu handeln, aber mit einer geringen Wahrscheinlichkeit ϵ eine zufällige Aktion auszuführen. Dabei spielen die geschätzen Nutzen der Aktionen keine Rolle und jede Aktion hat die gleiche Wahrscheinlichkeit ausgewählt zu werden. Zu vermerken ist, dass die gierige Aktion A_* ebenfalls in der Menge $\mathcal{A}(S_t)$ enthalten ist. Solche Strategien werden entsprechend als $\epsilon - greedy$ bezeichnet (Sutton & Barto, 2018, S. 28):

$$\pi(a|S_t) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)| & \text{wenn } a = A_* \\ \epsilon/|\mathcal{A}(S_t)| & \text{wenn } a \neq A_* \end{cases} \tag{2.12}$$

3 Lernmethoden

3.1 Monte-Carlo Methoden

3.1.1 First Visit

//TODO englisch oder deutsch?

Algorithm 1 On-policy first-visit MC control (for ϵ-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\epsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitary ϵ-soft policy
    $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
    $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat forever (for each episode):
    Generate an episode following $\pi : S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T - 1, T - 2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}$:
            Append G to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow$ average$(Returns(S_t, A_t))$
            $A^* \leftarrow \text{argmax}_a\, Q(S_t, a)$ (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(S_t)$:
                $$\pi(a|S_t) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \epsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

3.1.2 Every Visit*

3.1.3 Off-Policy

3.2 Temporal Difference Learning

3.2.1 Sarsa

Algorithm 2 Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameter: step size $\alpha \in (0, 1])$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in S^+, a \in \mathcal{A}(s)$, arbitrarily except that
$Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'; A \leftarrow A';$
    until S is terminal

3.2.2 Q-Learning

Algorithm 3 Q-Learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameter: step size $\alpha \in (0, 1])$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in S^+, a \in \mathcal{A}(s)$, arbitrarily except that
$Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S';$
    until S is terminal

Literatur

Brunskill, E. (2019). *Stanford cs234: Reinforcement learning - winter 2019.* Zugriff am 2020-02-10 auf https://www.youtube.com/watch?v=FgzM3zpZ55o

Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, *21*. doi: 10.1287/ijoc.1080.0305

Möller, B. & Struth, G. (2004). Greedy-like algorithms in modal kleene algebra. In R. Berghammer, B. Möller & G. Struth (Hrsg.), *Relational and kleene-algebraic methods in computer science.* Berlin, Heidelberg: Springer Berlin Heidelberg.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second Aufl.). The MIT Press. Zugriff auf http://incompleteideas.net/book/the-book-2nd.html

Watkins, C. J. C. H. (1989). Learning from delayed rewards.

Wiering, M. & van Otterlo, M. (2012). *Reinforcement learning: State-of-the-art.* Zugriff auf https://ebookcentral.proquest.com

Yu, J. Y., Mannor, S. & Shimkin, N. (2009). Markov decision processes with arbitrary reward processes. *Mathematics of Operations Research* (3), 737–757. Zugriff auf http://www.jstor.org/stable/40538443

Bellman Optimality Equation:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{6.1}$$