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. Nadija Syrjakow

2. Februar 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das ist eine Einleitung
(Fedjaev, 2017)

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 Markow Entscheidungsprozess

Die Umwelt wird in den allermeisten Fällen als Markow-Entscheidungsprozess (*Markov Decision Process, MDP*) definiert. 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, bei denen sich vermeintlich schlecht erscheinende Entscheidung in der Gegenwart, später als optimal herausstellen angesichts der gesamten Handlung. //BEISPIEL? Probleme die als *MDP* definiert werden müssen die Markow-Eigenschaft erfüllen, da diese gewissermaßen als Erweiterung von Markow-Ketten zu betrachten sind, mit dem Zusatz von Aktionen und Belohnungen. Bei den sog. Markow-Ketten führt das System zufällige Zustandswechsel durch, bei dem die Übergangswahrscheinlichkeiten von dem aktuellen Zustand zu einem Folgezustand ausschließlich von dem aktuellen Zustand abhängig ist und nicht auf den historischen Verlauf(Gosavi, 2009, S. 3). Da die Markow-Eigenschaft eine essentielle Voraussetzung bei der Problemmodellierung ist, wird sie in Kapitel X näher erläutert. //TODO Ein Universelles Interface könnte so definiert sein

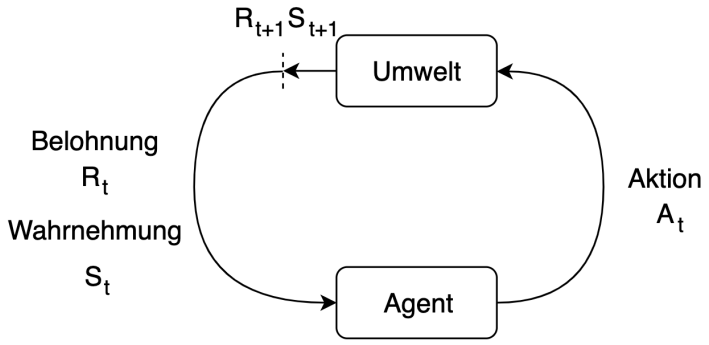

Abbildung 1: Agent-Umwelt Interface

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. Das Zusammenspiel zwischen Agenten und MDP erzeugt somit folgende Reihenfolge:

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

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 zufälligen Variablen R_t und S_t wohl definierte diskrete Wahrscheinlichkeitsverteilungen, die nur von dem vorigen Zustand und der vorigen Aktion abhängig sind (S.48). 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:

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

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:

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

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'.

Das MDP Framework gilt als extrem flexibel und kann auf die unterschiedlichsten Probleme angewendet werden. Es bietet die nötige Abstraktion für Probleme, bei denen unter Vorgabe eines Ziels mittels Interaktionen gelernt wird. Dabei sind die Einzelheiten über das eigentliche Ziel, die Zustände oder die Form des Agenten unerheblich, denn jedes zielgerichtete Lernen kann auf drei Signale reduziert werden, 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)(Sutton & Barto, 2018, S. 50).

2.2 Markow-Eigenschaft und Zustandsmodellierung

Die Markow-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 generelle Modellierung von Zuständen bei einem RL Problem.

> The future is independent of the past given the present

Dieser Satz erscheint oft in Büchern und Papern, wenn es um die Markow-Eigenschaft geht, denn er versucht zusammenzufassen, 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. In der Literatur gibt es unterschiedliche Auffassungen darüber, ob die Markow-Eigenschaft an den MDP direkt geknüpft ist oder an den Zustand, den der Agent zur Abwägung der Entscheidung zur Verfügung hat. Bei der ersten Annahme wird davon ausgegangen, dass der Zustand, der von der Umwelt ausgeliefert wird direkt die Markow-Eigenschaft besitzen muss. (Sutton S.49) hingegen bindet die Eigenschaft an den Zustand und nicht an den Entscheidungsprozess als solches. Ein Zustand ist somit die Menge aller notwendigen Informationen der Vergangenheit, die für die Zukunft relevant sind. Statt den gegebenen Zustand der Umwelt direkt zu übernehmen, werden hier Beobachtungen der Umwelt zu einer internen Repräsentation von Markow-Zuständen verarbeitet.

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 hauptsächlich 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.

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, Schach zu spielen. 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 beim 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, wird dem Agenten bei der zweiten Möglichkeit versucht, durch bestimmte Belohnungen zu zeigen, ob er 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 Bedeutung. Im Beispiel von Schach sollte nur aufgrund des Spielaussgangs bewertet

werden. Durchläuft der Agent allerdings ein Labyrinth und soll er so schnell wie möglich hinausfinden, dann sollte nach jeder Aktion eine negative Belohnung von -1 verteilt werden. Somit wird der Agent gezwungen, auf direktem Wege den Zielzustand zu erreichen.

Prinzipiell gilt: Das Belohnungssignal dient dazu, 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 Szenarie zurückgesetzt und der Agent startet erneut im Startzustand. Episoden sind komplett unabhängig voneinander und erzeugen Trajektorien, die nicht von durch vorrige Episoden beeinflusst werden.

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 Belohnung, 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.

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

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 Horizont (infinite horizon) besitzen. Folglich ist $T = \infty$, was wiederum bedeutet, dass der Gewinn ebenfalls 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:

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

Weiterhin ist zu vermerken, dass Gewinne aufeinanderfolgender Zeitpunkte in Verbindung stehen (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. Je näher γ an 1 ist, desto "weitsichtiger"wird der Agent, da der Gewinn für den Zeitpunkt t sich zusätzlich aus zukünftigen Belohnungen zusammensetzt. Es ist üblich, dass bei Problemen, die Episoden erzeugen $\gamma = 1$ zu bestimmen, sodass der Agent seine Entscheidungen immer aufgrund jeglicher Konsequenzen in der Zukunft bzw. bis zum Ende der jeweiligen Episode trifft. Um zu erreichen, dass die unendliche Summe in (2.5) bei kontinuierlichen Aufgaben einen endlichen Wert annimmt, muss $\gamma < 1$ gegeben sein (Sutton & Barto, 2018, S.55).

Probleme mit unendlichen Horizont 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.

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 gutës ist, dass sich der Agent in einem bestimmten Zustand befindet oder eine bestimmte Aktion in einem Zustand auszuführen. Das "wie gut"bezieht sich darauf, welche Belohnungen in der Zukunft erwartbar sind, also wie der erwartete Gewinn ist. Zukünftige Belohnungen sind natürlicherweise abhängig davon, wie sich der Agent verhalten bzw. welche Entscheidungen er treffen wird. Nutzenfunktion sind deshalb immer in Bezug auf eine bestimmte Strategie definiert(Sutton & Barto, 2018, S. 58).

Eine Strategie (*Policy*) ist die Abbildung von Zuständen auf die Wahrscheinlichkeiten einzelne Aktion auszuführen. 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 $\pi(s) = a$.

Wie anfangs erwähnt, gibt es zwei Varianten der Nutzenfunktion. Die erste sagt aus, wie groß der erwartete Gewinn für den Zustands s ist, wenn in diesem gestartet und anschließend aufgrund der Strategie π gehandelt wird. Dieser *Zustands-Nutzen* kann für alle $s \in \mathcal{S}$ definiert werden:

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

Die zweiten Variante gibt Auskunft darüber wie groß der Nutzen ist, wenn im Zustand s gestartet, daraufhin die Aktion a ausgeführt und anschließend der Stragie π gefolgt wird. q_π wird auch als Aktion-Nutzen-Funktion für die Stragie π bezeichnet und wird formal wie folgt definiert:

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

//TODO der Erwartungswert bezieht sich auf was? //TODO functionsapproximation hier?

2.6 Optimalität

Wofür Nutzenfunktionen? Beste Strategie, beste Nutzenfunktion. Warum actionValue besser ist (pefektes Modell) Bellmann equation, Berechnung. Approximation - Cliffhänger zu den Methoden Prediction und Control.

Ein Reinforcment 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. Mit anderen Worten, $\pi \geq \pi'$ gilt, wenn $v_\pi(s) \geq v_{\pi'}(s)$ für alle $s \in \mathcal{S}$. Es existiert immer eine Stragie 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_*.

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

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

Optimale Nutzenfunktionen sind solche, die den Gewinn im Bezug auf die Dynamiken des *MDP* perfekt wiederspiegeln. Ist ein Modell der Umgebung vorhanden, dann ist es möglich, die optimale Nutzenfunktion zu berechnen, denn sich kann

als Gleichungssystem verstanden werden, welches eine eindeutige Lösung hat. Dieses Gleichungssystem wird auch als *Bellman Optimality Equation* bezeichnet (2.11), wird jedoch im Weiteren nicht geanuer erläutert. Grund hierfür ist, dass zur Lösung ein perfektes Modell vorhanden sein muss, eine Vorraussetzung, die unter normalen Umständen nicht oft gegeben ist. Selbst wenn die Dynamiken bekannt sind, kann die benötigte Rechenzeit zur Lösungen jedoch utopische Ausmaße annehmen. Das Gleichungssystem besitzt eine Gleichung für jeden Zustand, das beudetet, wenn ein Problem n Zustände hat, ergeben sich n Gleichungen mit n Unbekannten (Sutton & Barto, 2018, S. 64).

Bei einem Spiel wie "Backgammonßind die Regeln bekannt, ein perfektes Modell ist somit vorhanden, aber es existieren $10^2 3$ Zustände, was die mathematische Berechnung von v_* mittels der *Bellman Optimality Equation* praktisch unmöglich macht. Dennoch stellt sie ein wichtiges Fundament des Reinforcment Learning dar, da die meisten Reinforcment Learning Algorithmen als annährendes Lösungsverfahren verstanden werden können (Sutton & Barto, 2018, S. 66).

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 ebenfalls ein perfektes Modell der Umgebung, um die Übergange für jede Aktion zu berechnen. Das ist der ausschlaggebende Grund, warum in der Regel q_* berechnet wird. Denn dieser Nutzen umfasst implizit den Nutzen der Folgezustand für jede Aktion. Somit 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.

Literatur

Fedjaev, J. (2017). *Decoding eeg brain signals using recurrent neural networks.* Zugriff auf `https://www.spektrum.de/kolumne/eine-waffe-gegen-malaria/1525481`

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

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`

Bellman Optimality Equation:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{2.11}$$