Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Intelligente Systeme

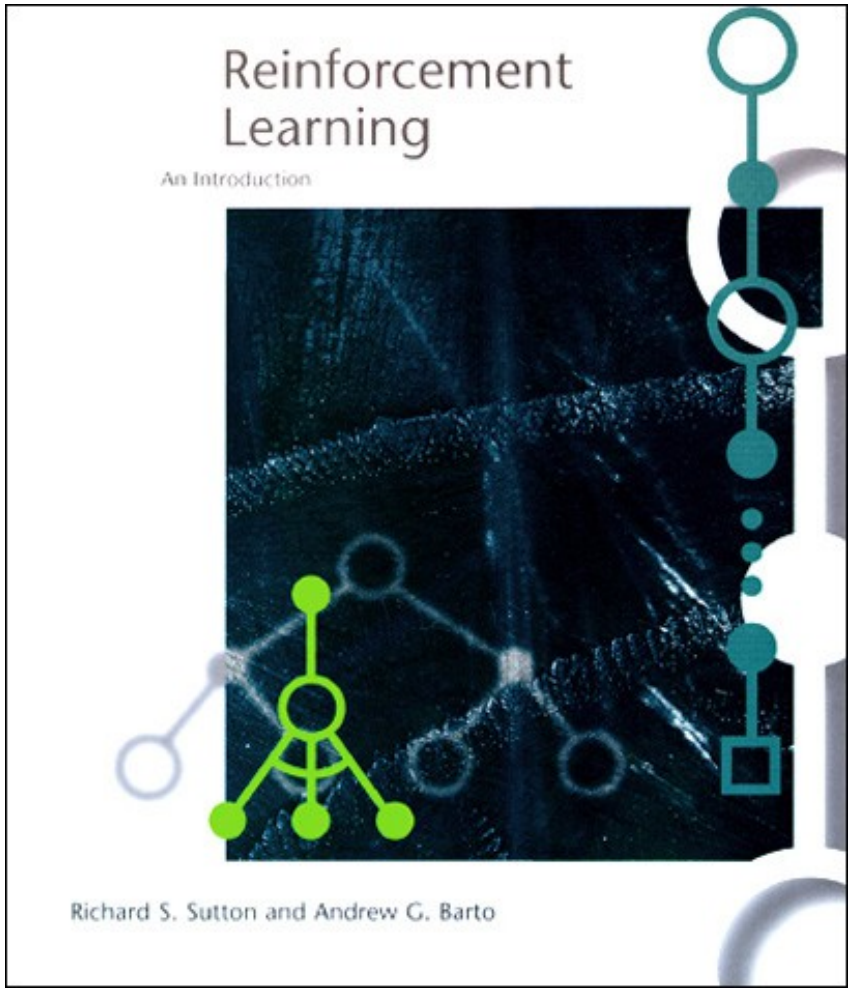
– Verstärkendes Lernen –

Prof. Dr. Michael Neitzke

Demos

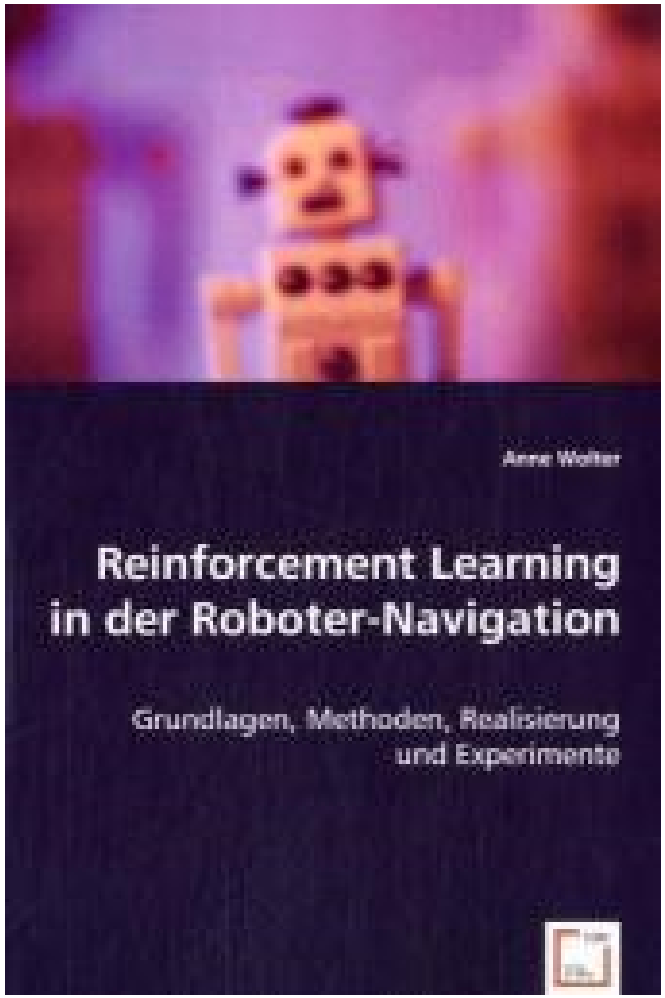
- Vorwärtsbewegung
- Torwart

Literatur: [SB 98]



- Umfassende Einführung
- HTML-Version im Internet:
- http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html
- http://www.cse.iitm.ac.in/~cs670/book/the-book.htm

Literatur: [Wol 08]



- Spezialisiert auf Temporal Difference Learning

Literatur: [RN 04]

- Das Standard-Buch der KI

Literatur: [KLM 96]

Journal of Artificial Intelligence Research 4 (1996) 237-285 Submitted 9/95; published 5/96

- Überblicks-Paper

Reinforcement Learning: A Survey

Leslie Pack Kaelbling LPK@CS.BROWN.EDU
Michael L. Littman MLITTMAN@CS.BROWN.EDU
Computer Science Department, Box 1910, Brown University
Providence, RI 02912-1910 USA

Andrew W. Moore AWM@CS.CMU.EDU
Smith Hall 221, Carnegie Mellon University, 5000 Forbes Avenue
Pittsburgh, PA 15213 USA

Abstract

This paper surveys the field of reinforcement learning from a computer-science perspective. It is written to be accessible to researchers familiar with machine learning. Both the historical basis of the field and a broad selection of current work are summarized. Reinforcement learning is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment. The work described here has a resemblance to work in psychology, but differs considerably in the details and in the use of the word "reinforcement." The paper discusses central issues of reinforcement learning, including trading off exploration and exploitation, establishing the foundations of the field via Markov decision theory, learning from delayed reinforcement, constructing empirical models to accelerate learning, making use of generalization and hierarchy, and coping with hidden state. It concludes with a survey of some implemented systems and an assessment of the practical utility of current methods for reinforcement learning.

Inhalt

- VL1: Grundlagen, Begriffe
- VL2: Methoden der dynamischen Programmierung
- VL3: Monte Carlo Methoden
- VL4: Temporal Difference Learning

VL1: Grundlagen, Begriffe

VL1: Lernziele

- V1: Die rot gekennzeichneten Begriffe erläutern können
- V2: Wechselspiel zwischen Umwelt und Agent erläutern können

Fragen zu den Videos

- Rückwärtsbewegung
- Wie verhält sich der Agent?
- Wie verhält sich der Agent bei einem Hindernis?
- Wie könnte das Lernverfahren funktionieren?

Lösung: Verhalten des Agenten

- Zunächst macht der Agent irgendetwas.
- Dann werden seine Aktionen immer sinnvoller.
- Schließlich ist ein erfolgreiches Handeln erlernt worden.
- Eine ungewohnte Situation (Hindernis) wird zunächst nicht gemeistert. Nach einigen vergeblichen Versuchen probiert der Agent etwas Neues aus. Schließlich ist er erfolgreich.

Lösung: Mögliche Realisierung

- Herumprobieren
- Feststellen, was erfolgreich ist und was nicht
- Erfolgreiche Aktionen bevorzugen
- Wenn Erfolg (entgegen der Erfahrung) wieder ausbleibt, wieder etwas Neues ausprobieren. (Tatsächlich wird dieser Aspekt dadurch realisiert, dass immer mit einer kleinen Wahrscheinlichkeit auch unattraktive Aktionen ausprobiert werden.)

- Zustandsraum
- Liste möglicher Aktionen
- Bewertung von Zuständen und/oder Aktionen

Grundlegende Modellierung: agentenorientiert

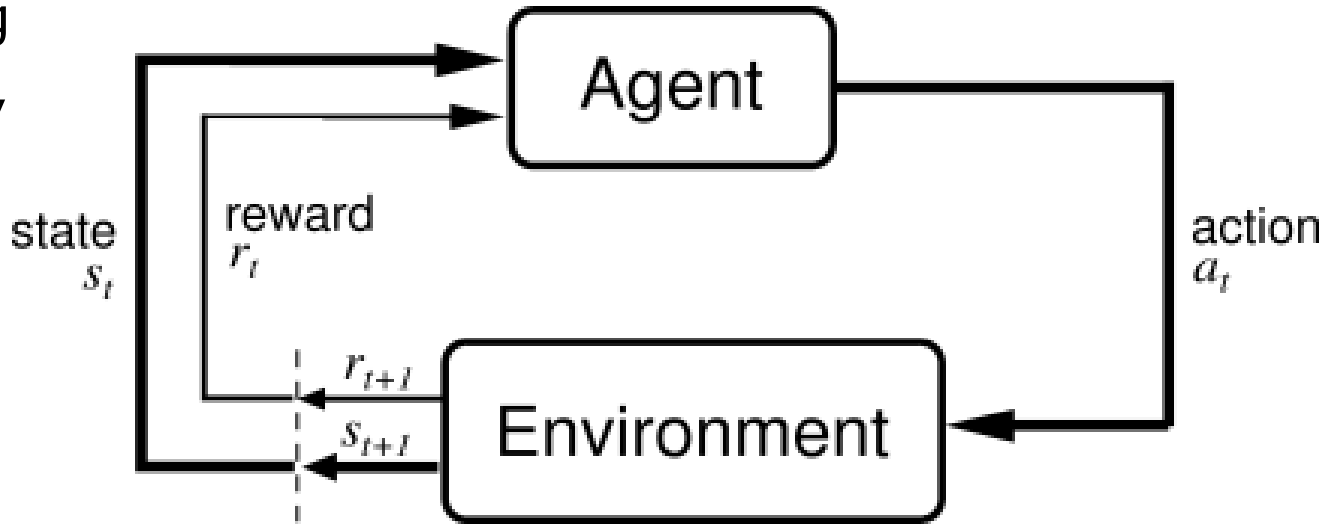
- Agent lebt in **Umwelt**
- Agent nimmt Aspekte der Umwelt über **Sensoren** auf
- Agent verfügt über internes Modell
- Agent wirkt über **Effektoren**/**Aktuatoren** auf Umwelt ein
  - indem er **Aktionen** ausführt
- Handeln ist also Folge von Aktionen

Wie wird gelernt?

- **Zustandsraum**
- Liste möglicher Aktionen, ggf. abhängig vom Zustand
- Anfangs sind alle Zustände und Aktionen gleich gut
- Agent wirkt auf Umwelt über Aktionen ein
  - Teilt der Umwelt mit, welche Aktion er wählt
- Agent nimmt nach Ausführung einer Aktion Zustandsänderung und ggf. **Belohnung** / Bestrafung wahr
  - Umwelt informiert den Agenten über Folgezustand und Belohnung
  - Technisch wird Umwelt meist auch als Agent realisiert
- Agent möchte seine Belohnungen maximieren
  - Bewertung der Zustände gibt Auskunft über zu erwartende Belohnungen

Agenten-Umgebungs-Schnittstelle

- Zu jedem Zeitpunkt **t** nimmt der Agent Merkmale des Zustands s_t wahr, in dem er sich gerade befindet. Er verfügt also über eine bestimmte Repräsentation dieses Zustands.
- Der Agent wählt dann eine mögliche Aktion a_t
- Daraufhin gelangt der Agent in einen Folgezustand s_{t+1} und erhält eine Belohnung (**reward**) r_{t+1}, die auch negativ sein kann.



Diskutieren Sie!

- Wo liegt die Schnittstelle zwischen Umwelt und Agent?
- Der Krabbelroboter hat einen Sensor, der eine Belohnung für Vorwärtskommen signalisiert. Gehört diese Komponente zur Umwelt oder zum Agenten?

Was gehört zur Umgebung?

- Umgebung wird umfassender modelliert, als man intuitiv erwarten würde
  - Belohnung wird als Phänomen der Umgebung modelliert
- Beispiele:
  - Wenn ein Beutetier erfolgreich erlegt wurde, erfolgt die entsprechende Belohnung dieses Zustands offensichtlich durch die Umgebung
  - Fortbewegung kostet Energie
  - Das kann durch negative Belohnung ausgedrückt werden
  - Energiepegel wird also als Konzept der Umgebung modelliert
- **Modell** der Umgebung beinhaltet Belohnung und Folgezustand
  - Belohnung und Folgezustand werden vom Agenten beobachtet
- Geschickte Wahl der Belohnungen ist entscheidend für Lernerfolg!

Diskutieren Sie!

- Welche Schwierigkeit könnte entstehen, wenn ein Schach-Agent eine hohe Belohnung dafür erhält, dass er die gegnerische Dame schlägt?

Lösung

- Dieser Zustand wird so stark angestrebt, dass zu viele Verluste anderer Figuren in Kauf genommen werden, so dass das Spiel letztendlich verloren wird.

Diskutieren Sie!

- Was fällt Ihnen auf?

Environment:	You are in state 65. You have 4 possible actions.
Agent:	I'll take action 2.
Environment:	You received a reinforcement of 7 units. You are now in state 15. You have 2 possible actions.
Agent:	I'll take action 1.
Environment:	You received a reinforcement of -4 units. You are now in state 65. You have 4 possible actions.
Agent:	I'll take action 2.
Environment:	You received a reinforcement of 5 units. You are now in state 44. You have 5 possible actions.
⋮	⋮

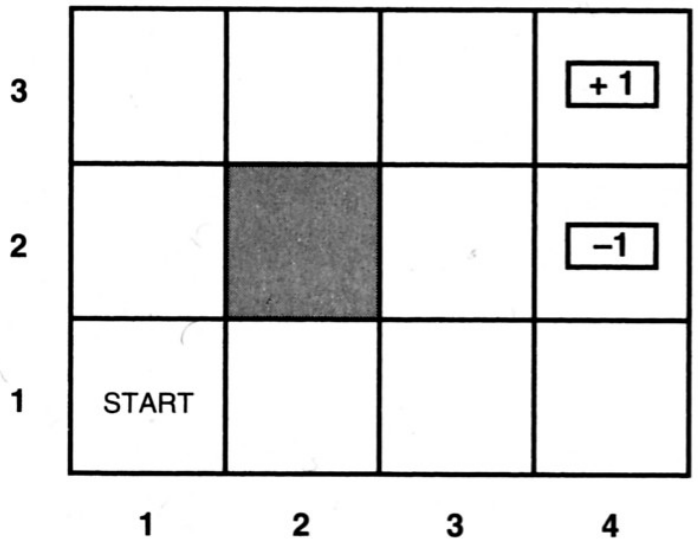
Interaktion mit Umwelt / Umgebung

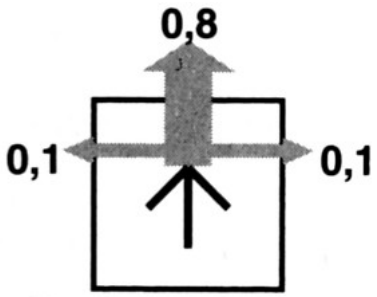
Environment:	You are in state 65. You have 4 possible actions.
Agent:	I'll take action 2.
Environment:	You received a reinforcement of 7 units. You are now in state 15. You have 2 possible actions.
Agent:	I'll take action 1.
Environment:	You received a reinforcement of -4 units. You are now in state 65. You have 4 possible actions.
Agent:	I'll take action 2.
Environment:	You received a reinforcement of 5 units. You are now in state 44. You have 5 possible actions.
⋮	⋮

- **Umgebung** ist im Allgemeinen **nicht-deterministisch**: Action 2 in State 65 führt nicht immer zum selben Folgezustand.
- **Umgebung** ist aber **stationär**: Wahrscheinlichkeiten für Folgezustände bei gegebener Aktion ändern sich nicht im Laufe der Zeit

Beispiel: Nicht-deterministische Umgebung

- Agent muss vom Start zu einem Zielzustand
- Mögliche Aktionen: Links, rechts, oben, unten
  - Gegen Mauer laufen bewirkt Verbleib im aktuellen Zustand
- Umgebung vollständig beobachtbar: Agent weiß, wo er ist
- Umgebung ist nicht-deterministisch: Ergebnis einer Aktion ist nicht vorhersehbar
  - Im Beispiel: 80% Erfolgswahrscheinlichkeit, je 10% für Nachbar-Richtungen





Strategie (Policy)

- Welche Aktion wird abhängig vom Zustand gewählt?
- --> **Strategie** (Policy)
- Allgemeiner Ansatz (**stochastische Strategie**):
  Für einen gegebenen Zeitpunkt **t** gibt es eine Wahrscheinlichkeit $\pi_t(s,a)$ dafür, dass **a** die gewählte Aktion a_t sein wird, falls **s** der aktuell vorliegende Zustand s_t sein sollte.

- **Weshalb ist diese Wahrscheinlichkeit zeitabhängig?**

Lösung

- Die Wahrscheinlichkeit kann zu einem späteren Zeitpunkt anders sein
  - aufgrund eines Lernerfolgs
- **Achtung**: Hier geht es um den Nicht-Determinismus bezüglich des Handelns des Agenten. Nicht aber um den Nicht-Determinismus der Umwelt

Deterministische Strategie, optimale Strategie

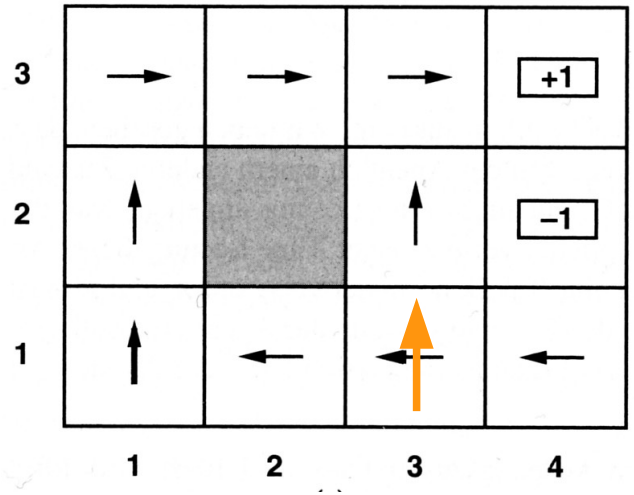
- Spezialfall **deterministische Strategie**: **π: S -> A**
  - **S**: Menge der Zustände, **A**: Menge der Aktionen

- Das Ziel des Agenten besteht darin, seine Belohnungen über die Gesamtlaufzeit zu maximieren, also die **optimale Strategie π^*** zu lernen.

- **A(s)**: Mögliche Aktionen eines Zustands

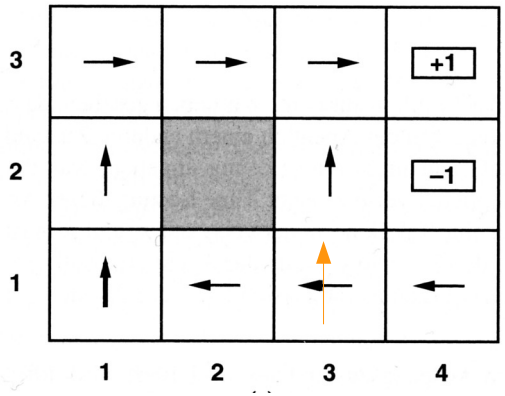
Optimale Strategie für Beispiel

- Optimale Strategie hängt von Belohnungsfunktion ab
- Belohnungen: **+1** und **-1** für Zielzustände, **-0.04** für alle anderen Zustände
- Optimale Strategie für **r(s)=-0.04** :

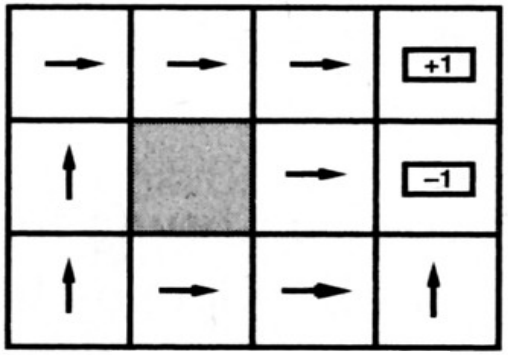


- **Diskutieren Sie:**
  - **Weshalb wird im Feld (2,1) der Umweg gewählt?**
  - **Wie wirken sich höhere, wie wirken sich niedrigere Wegekosten und wie wirkt sich eine Wegebelohnung aus?**

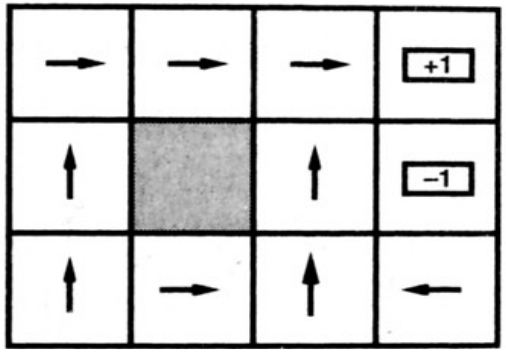
Optimale Strategien für verschiedene Wegekosten



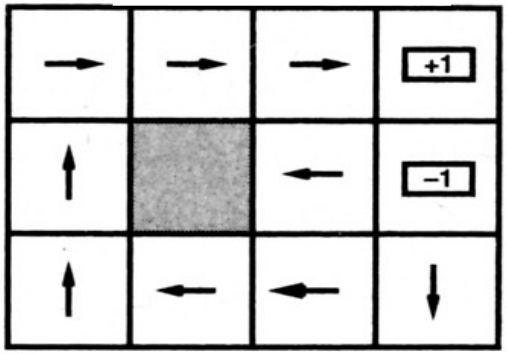
r(s)=-0.04



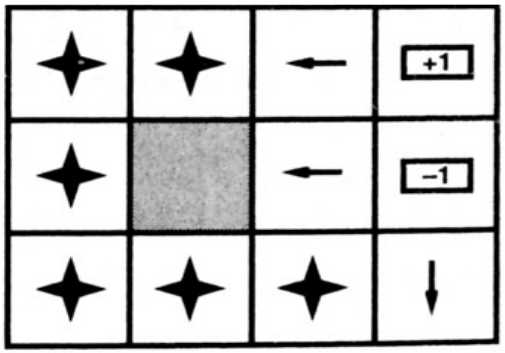
r(s) < -1.628



-0.4278<r(s) < -0.0850



- 0.0221 < r(s) < 0



r(s) > 0

Belohnungen bei episodischen Aufgaben

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T,$$

- **Return R_t** ist Summe der Rewards ab Zeitpunkt **t**
- **T** wäre ein abschließender Schritt
- Handlungsabläufe des Agenten können aus **Episoden** bestehen, die jeweils mit einem abschließenden Schritt enden
- Der Agent führt dann **episodische Aufgaben** durch (episodic tasks)
- Dann unterscheidet man die Menge der **nicht-terminalen** Zustände **S** von der Menge aller Zustände **S^+** (inklusive der **terminalen** Zustände)
  - (Nur nicht-teminale Zustände müssen bewertet werden.)

Belohnungen bei kontinuierlichen Aufgaben

- Fortdauernde Aufgaben heißen **kontinuierlich**
- Nicht-endende Aufgaben würden zu einer unendlich hohen Belohnung führen
- Lösung: Abschwächung zukünftiger Belohnungen (**discounting**)

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

- **Discount-Rate**: $0 \leq \gamma \leq 1$
- Durch Discount-Rate ist Gesamt-Belohnung begrenzt:

$$\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \leq \sum_{k=0}^{\infty} \gamma^k \, r_{max} = r_{max} / (1-\gamma)$$

- Grundsätzlich ist Discounting auch bei episodischen Aufgaben möglich

Die Markov-Eigenschaft

- Die Umgebung liefert dem Agenten Zustandsinformation
- Diese sollte komplett sein, also alle relevante Information aller Vorgänger-Zustände enthalten
  - Dann bräuchte man nur die jeweils aktuelle Zustandsinformation und keinen Zugriff auf Vorgänger-Zustände
- Diese Eigenschaft eines Zustands-Signals heißt **Markov-Eigenschaft**
- Beispiel: Konfiguration auf dem Schachbrett
  - Alles Wesentliche für zukünftige Entscheidungen ist enthalten
  - Wie die Konfiguration entstanden ist, ist irrelevant
- Für **Markov-Zustände** sind folgende Wahrscheinlichkeiten gleich:

$$Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\right\},$$

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

Markov-Entscheidungsprozess (MDP)

- Die Wahrscheinlichkeit eines Folgezustands **s'** (die **Übergangswahrscheinlichkeit**) hängt nur vom aktuellen Zustand und der gewählten Aktion ab:

$$\mathcal{P}^a_{ss'} = Pr\left\{s_{t+1} = s' \mid s_t = s, a_t = a\right\}.$$

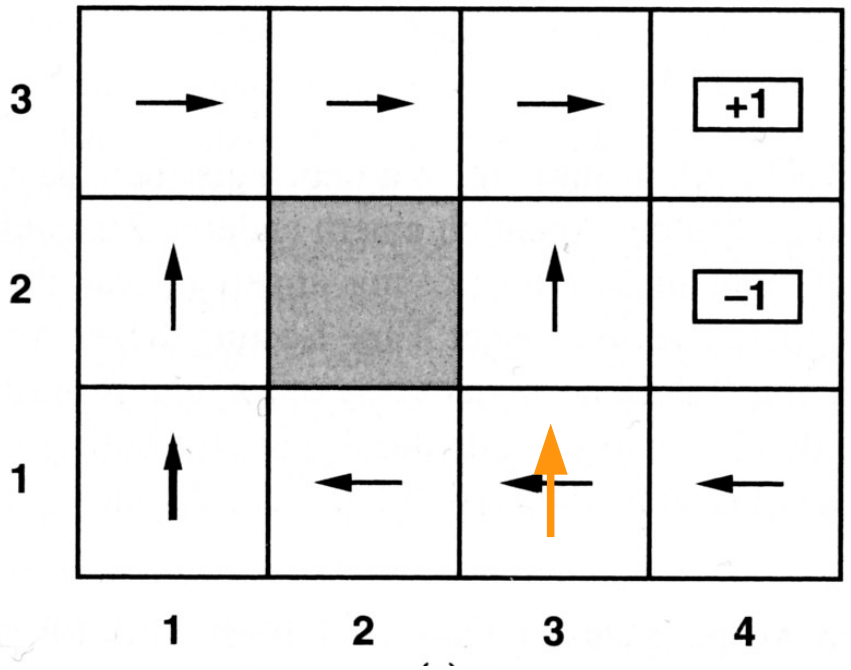
- Der **Erwartungswert für die sofortige Belohnung** hängt nur vom aktuellen Zustand, der gewählten Aktion und vom Folgezustand ab:

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

- Wenn diese beiden Bedingungen für alle Übergänge und Belohnungen erfüllt sind, spricht man von einem **Markov-Entscheidungsprozess** (**Markov Decision Process**, **MDP**)

Diskutieren Sie!

- Wie könnte man einen Zustand bewerten, woran seinen Wert oder Nutzen bemessen?



Bewertungsfunktionen: Zustand-Wert-Funktion

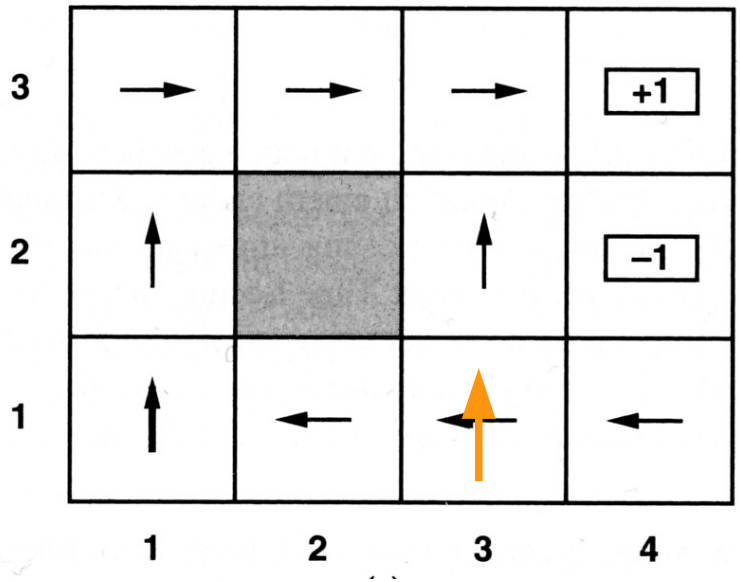
- Der **V-Wert** eines Zustands **s** ist der Erwartungswert der aufsummierten Belohnungen, die, ausgehend von **s** unter einer bestimmten Strategie π erreicht werden:

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

- Funktion V^{π} heißt **Zustand-Wert-Funktion** (**state-value function**)

- **V*** ist **optimale V-Funktion**
  - Also bei optimaler Strategie π^*
  - Gibt maximal erreichbare Belohnung an

Erwartungswert für aufsummierte Belohnungen



Zu bewertende Strategie

V-Werte für $\gamma=1$ und r(s)=-0,04

3	0,812	0,868	0,918	+1
2	0,762		0,660	–1
1	0,705	0,655	0,611	0,388
	1	2	3	4

Link auf Seite 42

Bewertungsfunktionen: Aktion-Wert-Funktion

- Der **Q-Wert** eines Zustands **s** ist der Erwartungswert der aufsummierten Belohnungen, die, ausgehend von **s** unter Auswahl einer bestimmten Aktion **a** und der anschließenden Verfolgung einer Strategie π erreicht werden:

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

- Funktion Q^{π} heißt **Aktion-Wert-Funktion** (**action-value function**)
- **Q*** ist **optimale Q-Funktion**
  - Also bei optimaler Strategie π^*
  - Gibt maximal erreichbare Belohnung an

Abgrenzung zu anderen Lernverfahren

- Überwachtes Lernen
  - Agent bekommt durch einen Trainer Musterlösungen für gegebene Problemstellungen gezeigt, abstrahiert von den Details und erlernt ein Ein-/Ausgabeverhalten für neue Situationen
  - Mathematisch: Lernen einer Funktion
- Unüberwachtes Lernen
  - Agent lernt Unterscheidungen aufgrund von Beispielfällen mit Hilfe von Ähnlichkeitsmaßen
  - Beispielanwendung: Clustering

VL1: Lernziele

- V1: Die rot gekennzeichneten Begriffe erläutern können
- V2: Wechselspiel zwischen Umwelt und Agent erläutern können

Übung: Welche Begriffe sind noch unklar?

- Agent, Umwelt/Umgebung
- Sensoren, Effektoren, Aktuatoren
- Aktionen
- Zustandsraum, terminale Zustände, nicht-terminale Zustände
- Belohnung, Reward, Return
- Modell der Umgebung (der Umwelt)
- Umwelt: deterministisch, nicht-deterministisch, stationär
- Strategie: deterministisch, stochastisch/nicht-deterministisch, optimal
- Episoden, episodische Aufgaben, kontinuierliche Aufgaben
- Discounting, Discount-Rate
- Markov-Eigenschaft, Markov-Zustände, Markov-Entscheidungsprozess (MDP)
- Übergangswahrscheinlichkeit, Erwartungswert für sofortige Belohnung
- V-Wert, Zustand-Wert-Funktion (state-value function), optimale V-Funktion
- Q-Wert, Aktion-Wert-Funktion (action-value function), optimale Q-Funktion

VL2: Methoden der dynamischen Programmierung

- Grundidee der Algorithmen aus dem Bereich der dynamischen Programmierung
- Algorithmus zur Strategie-Bewertung
  - Policy Evaluation
- Algorithmen zur Strategie-Verbesserung
  - Policy Iteration
  - Value Iteration

VL2: Lernziele

- V1: Merkmale des Dynamic Programming für Berechnungs-verfahren aus dem verstärkenden Lernen erläutern können
- V2: Die rot gekennzeichneten Begrifflichkeiten erläutern können
- V3: Verfahren Policy Evaluation, Policy Iteration und Value Iteration erklären können
  - Algorithmen in ihren Grundzügen erläutern können

- A1: Die V-Wert-Aktualisierungsformel anwenden können (--> Teil 2 der Klausur)

Dynamische Programmierung

- Voraussetzung: Vollständiges Modell, also
  - Zustände bekannt
  - Belohnungen für alle Zustände bekannt
  - Folgezustände in Abhängigkeit von Ausgangszustand und Aktion bekannt
- Kein Lernen, sondern Vorab-Berechnung
  - Iterativ
- Grundsätzlich zwei Arten von Algorithmen:
  - Strategie-Bewertung
  - Strategie-Verbesserung bis hin zur Berechnung der optimalen Strategie

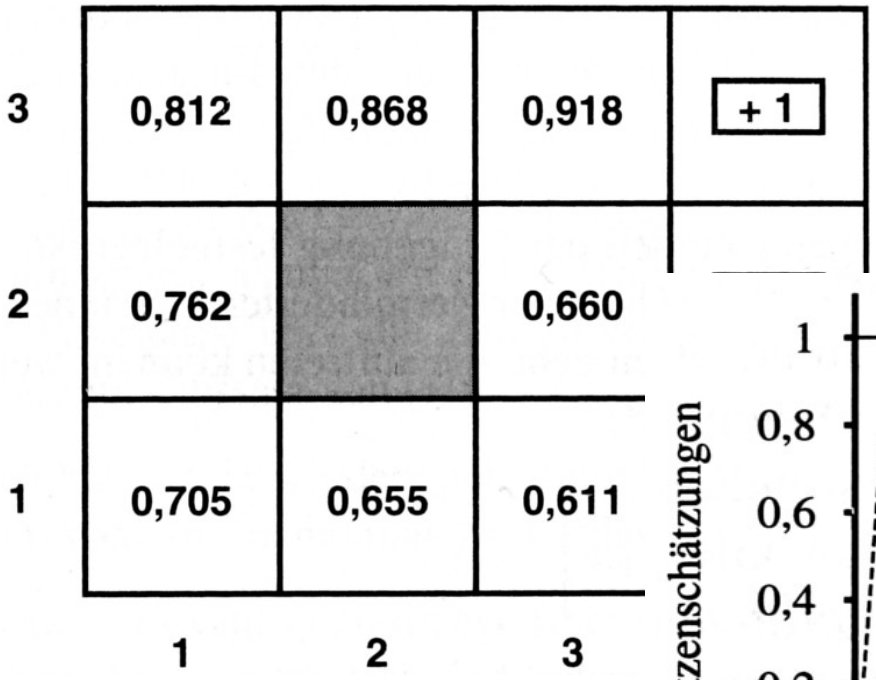
Grundidee bei dynamischer Programmierung

- V-Wert eines Zustands ist **Summe** aus
  - Belohnung für einen einzelnen Schritt
  - V-Wert des aus diesem Schritt resultierenden Folgezustands

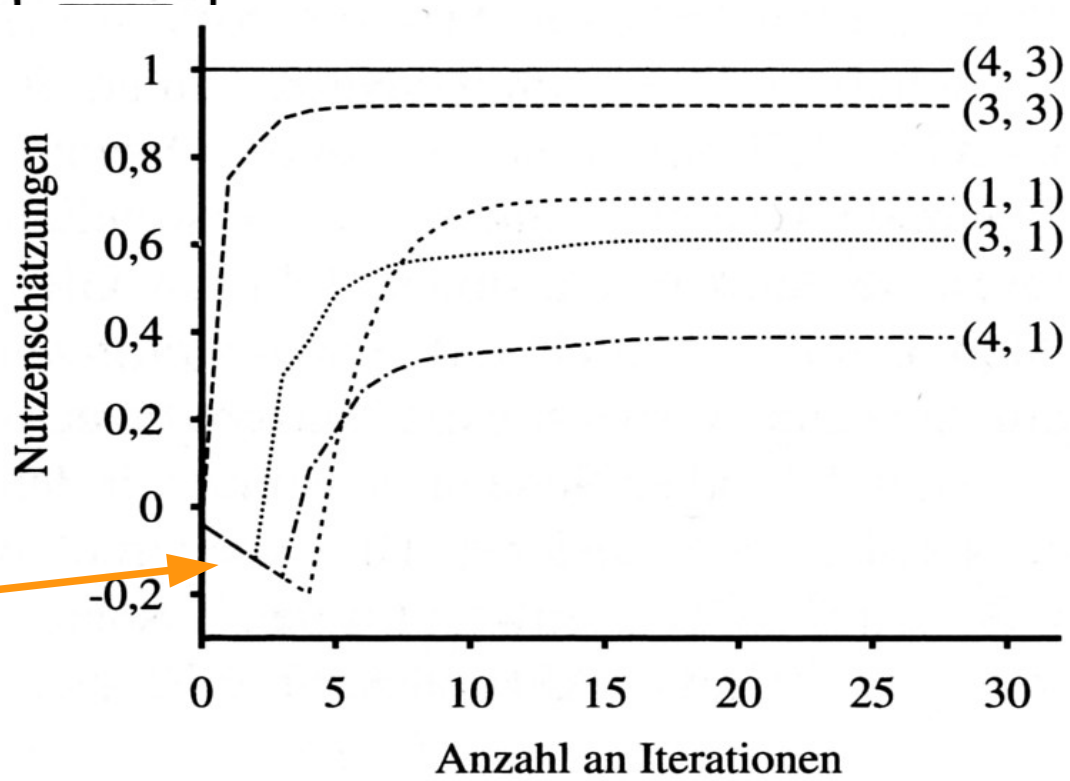
- Also zirkuläre Abhängigkeiten!
  - Wo soll man da mit der Berechnung anfangen?
  - --> Irgendwo, mit irgendwelchen V-Werten
    - Iterativer Prozess, der **konvergiert**!

- Die Abhängigkeiten werden komplizierter
  - bei nicht-deterministischer Strategie
  - bei nicht-deterministischer Umwelt
  - --> **Verallgemeinerung** der Berechnungsformel

Konvergenz des Verfahrens



- Beweis der Konvergenz in [RN 04], Kap. 17.2.3



Wert sinkt bis Pfad zum Ziel gefunden

Allgemeine Formel

- Berechnung der Zustandswerte bzgl. einer fest vorgegebenen Strategie entsprechend Bellman-Gleichung

Erwartungswert für Belohnung bei s, s', a

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right],$$

Wahrscheinlichkeit für Aktion a im Zustand s

Wahrscheinlichkeit für Folgezustand s', wenn im Zustand s Aktion a gewählt wird

- Update-Möglichkeiten aufgrund der Bellman-Gleichung:

$$V_{k+1}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V_{k}(s') \right],$$

Wert aus der Iteration k+1

Wert aus der Iteration k

Erläuterung

- Richard Bellman hat zusammen mit anderen in den 50er Jahren einen Ansatz zur Lösung von Optimierungsproblemen entwickelt, der darauf basiert, dass eine optimale Gesamtlösung aus optimalen Teillösungen berechnet wird. Bezogen auf Reinforcement Learning bedeutet dies, dass der Wert der optimalen V-Funktion für einen Zustand s dadurch berechnet werden kann, dass man, ausgehend von diesem Zustand s die Belohnung für die bestmögliche nächste Aktion und den Wert der optimalen V-Funktion des dadurch erreichten Folgezustands addiert.

$$V^*(s) = \max_a E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\right\}$$
$$= \max_a \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right]$$

Dieser Zusammenhang ist die Basis für die iterativen Berechnungen in den RL-Algorithmen aus dem Bereich der dynamischen Programmierung.

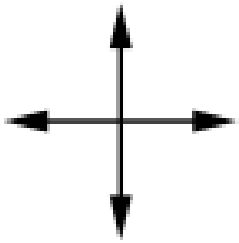
Algorithmus zur Policy Evaluation

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

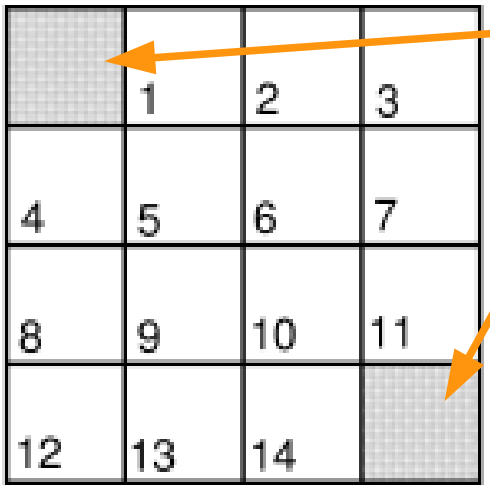
Wahrscheinlichkeit für Aktion a

Wahrscheinlichkeit für Folgezustand s'

Übung (1)



actions



Zielzustand

$r = -1$
on all transitions

$$\mathcal{S} = \{1, 2, \ldots, 14\} \qquad \mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{right}, \texttt{left}\} \qquad \mathcal{R}^a_{ss'} = -1$$

Deterministische Umwelt,
Beispiele für Übergangswahrscheinlichkeiten:

$$\mathcal{P}^{\texttt{right}}_{5,6} = 1 \qquad \mathcal{P}^{\texttt{right}}_{7,7} = 1 \qquad \mathcal{P}^{\texttt{right}}_{5,10} = 0$$

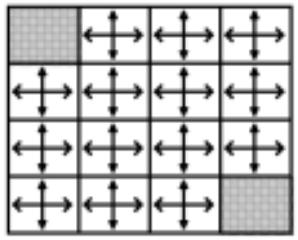
Übung (2)

V_k for the Random Policy

Greedy Policy w.r.t. V_k

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

- **Berechnen Sie V_1 bezüglich der Zufallsstrategie ! (γ=1)**
- **Berechnen Sie V_2 bezüglich der Zufallsstrategie !**

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right],$$

Weitere Iterationsschritte

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

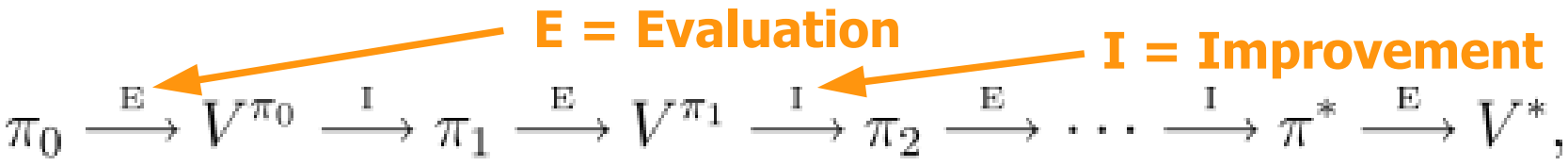
Strategie-Verbesserung: Grundidee

1. Start mit beliebiger Strategie, z. B. Random Policy

2. Bewertung dieser Strategie, also Berechnung der V-Werte

3. Berechnete V-Werte für Verbesserung der Strategie nutzen:
   Welche Aktion führt zum besten Folgezustand?

4. Weiter mit Schritt 2 oder Abbruch, wenn sich die Strategie nicht mehr ändert



Strategie-Iteration (Policy Iteration) Algorithmus

1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

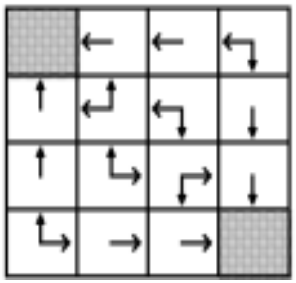
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $b \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$
   If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop; else go to 2

Beispiel Strategie-Iteration

- Links: Ergebnis der Wert-Iteration für Random Policy
- Rechts: Zugehörige Strategie
- **Was ist das Ergebnis der dann folgenden Wert-Iteration?**
- --> Optimale Strategie bereits nach ***zwei*** Iterationen, weil sich Strategie nicht mehr ändert
- Aber Strategie-Bewertung ist aufwändig durch viele Iterationen !

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



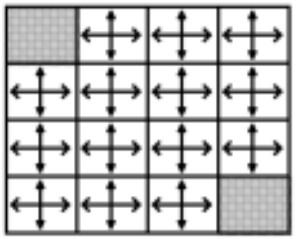
Strategie-Verbesserung sofort

V_k for the Random Policy

Greedy Policy w.r.t. V_k

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

- **Berechnen Sie V_2 !**

$$\sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right],$$

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right],$$

Wert-Iteration Algorithmus

Initialize V arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s') \right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that

$\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s') \right]$

Demo: Value Iteration

- RL-SIM

Rückblick u. Bewertung: Dynamic Programming

- Methoden:
  - Strategie-Bewertung (Policy Evaluation)
  - **Berechnung** einer optimalen Strategie
    - Strategie-Iteration (Policy Iteration)
    - Werte-Iteration (Value Iteration)
- Probleme:
  - Iteration erstreckt sich über alle Zustände
  - Backgammon: > 10^{20} Zustände
    - Wenn 1 Mio Zustände pro Sekunde bewertet, dann Dauer einer Iteration > 1000 Jahre
  - Asynchrone Algorithmen evaluieren gezielt bestimmte Zustände, ermöglichen so Handeln in Realzeit
    - Letztendlich kein verminderter Rechenaufwand

VL2: Lernziele

- V1: Merkmale des Dynamic Programming für Berechnungs-verfahren aus dem verstärkenden Lernen erläutern können
- V2: Die rot gekennzeichneten Begrifflichkeiten erläutern können
- V3: Verfahren Policy Evaluation, Policy Iteration und Value Iteration erklären können
  - Algorithmen in ihren Grundzügen erläutern können

- A1: Die V-Wert-Aktualisierungsformel anwenden können (--> Teil 2 der Klausur)

VL3: Lernziele

- V1: Merkmale der Monte Carlo Methoden erläutern können, das Wesen der Verfahren erklären können, Unterschiede zu DP erläutern können
- V2: Die rot gekennzeichneten Begrifflichkeiten erläutern können
- V3: Verfahren zur Strategie-Bewertung (First-Visit) und Strategie-Verbesserung On-Policy Monte Carlo Control in ihren Grundzügen erläutern können
- V4: Erläutern können, wann V-Werte und wann Q-Werte verwendet werden
- A1: Fragen zu den beiden vorgestellten Algorithmen beantworten können, die Berechnungen beinhalten --> Teil 2 der Klausur. (Die Berechnung der Explorations-Wahrscheinlichkeit muss nicht im Detail beherrscht werden.)

Monte Carlo (und TD-) Methoden: Motivation

- **Was tun, wenn man die Zustandsübergänge nicht kennt?**
- **Was tun, wenn man die zu erwartende Belohnung eines Zustands nicht kennt?**

- Erfahrungen sammeln durch tatsächliches oder simuliertes Interagieren mit der Umwelt

Grundidee der Monte-Carlo Strategie-Bewertung

- Vollständiger Durchlauf entsprechend der zu bewertenden Strategie
  - **Achtung: Monte-Carlo-Methoden sind nur für episodische Aufgaben geeignet !**
- Jeder aufgetretene Zustand bekommt nachfolgende aufsummierte Belohnungen gutgeschrieben
  - Mittelwertbildung über alle Durchläufe
- Erneuter Durchlauf so lange, bis Mittelwertbildung nur noch Änderungen unterhalb Schwellwert bewirkt

First-visit Monte Carlo Methode: Algorithmus

Initialize:

$\pi \leftarrow$ policy to be evaluated

$V \leftarrow$ an arbitrary state-value function

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

(a) Generate an episode using π

(b) For each state s appearing in the episode:

$R \leftarrow$ return following the first occurrence of s

Append R to $Returns(s)$

$V(s) \leftarrow \text{average}(Returns(s))$

Grundidee der Monte-Carlo Strategie-Verbesserung

- Erzeugung einer vollständigen Episode entsprechend der günstigsten Aktionen
  - --> Jetzt brauchen wir Q-Werte, also Bewertungen von Zustands/Aktions-Paaren !
  - Wahl der günstigsten Aktion: **Exploitation**
  - Aber mit kleiner Wahrscheinlichkeit Wahl einer anderen als der günstigsten Aktion: **Exploration**
    - Sonst hätte man keine Möglichkeit, neue, bessere Wege zu entdecken
- Mittelwertbildung für jeden beteiligten Q-Wert der Episode

On-Policy Monte Carlo Control: Algorithmus

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s,a) \leftarrow$ arbitrary

$Returns(s,a) \leftarrow$ empty list

$\pi \leftarrow$ an arbitrary ε-soft policy

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$R \leftarrow$ return following the first occurrence of s, a

Append R to $Returns(s,a)$

$Q(s,a) \leftarrow \text{average}(Returns(s,a))$

(c) For each s in the episode:

$a^* \leftarrow \arg\max_a Q(s,a)$

For all $a \in \mathcal{A}(s)$:

$$\pi(s,a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

ε-greedy und ε-soft Strategien

- Strategie Π ist **soft**, wenn $\Pi(s,a) > 0$ für alle **s**, **a**
- Strategie Π ist **ε-soft**, wenn $\Pi(s,a) \geq \varepsilon / |A(s)|$ für alle **s**, **a**
- Strategie Π ist **ε-greedy**, wenn $\Pi(s,a) = \varepsilon / |A(s)|$ für alle Aktionen außer der günstigsten. (Die günstigste Aktion erhält dann die typischerweise hohe Restwahrscheinlichkeit.)
- **ε-greedy** ist Spezialfall von **ε-soft**

Erläuterungen

- Der Unterschied zwischen **ε-soft** und **ε-greedy** wird deutlich, wenn man sich eine Strategie vorstellt, bei der die Wahrscheinlichkeiten für die Aktionen im selben Verhältnis zueinander stehen wie die entsprechenden Q-Werte (aber mindestens ε betragen). Beispielsweise hätte dann eine doppelt so gut bewertete Aktion eine doppelt so hohe Wahrscheinlichkeit. Diese Strategie wäre **ε-soft**, aber nicht **ε-greedy**.

VL3: Lernziele

- V1: Merkmale der Monte Carlo Methoden erläutern können, das Wesen der Verfahren erklären können, Unterschiede zu DP erläutern können
- V2: Die rot gekennzeichneten Begrifflichkeiten erläutern können
- V3: Verfahren zur Strategie-Bewertung (First-Visit) und Strategie-Verbesserung On-Policy Monte Carlo Control in ihren Grundzügen erläutern können
- V4: Erläutern können, wann V-Werte und wann Q-Werte verwendet werden
- A1: Fragen zu den beiden vorgestellten Algorithmen beantworten können, die Berechnungen beinhalten --> Teil 2 der Klausur. (Die Berechnung der Explorations-Wahrscheinlichkeit muss nicht im Detail beherrscht werden.)

VL4: Temporal Difference Learning

VL4: Lernziele

- V1: TD mit MC und DP vergleichen können
- V2: TD Learning Grundidee erläutern können
- V3: Unterschiede und Gemeinsamkeiten zwischen SARSA und Q-Learning erläutern können
- V4: Die rot gekennzeichneten Begriffe erläutern können
- V5: Besonderheit des SARSA(λ)-Algorithmus erläutern können

- A1: Einzelnen Berechnungsschritt für SARSA und Q-Learning durchführen können (--> Teil 2 der Klausur)

Die TD Learning Grundidee

- Korrektur des Zustandswertes auf Basis von Erfahrungen und Schätzungen

Monte Carlo Konzept

$$V(s_t) \leftarrow V(s_t) + \alpha \Big[R_t - V(s_t) \Big],$$

Auf Basis der bisherigen Erfahrungen geschätzter Wert des erreichten Folgezustands

Erhaltene (beobachtete) Belohnung

$$V(s_t) \leftarrow V(s_t) + \alpha \Big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \Big].$$

Lernrate zur Steuerung der Korrekturstärke

Entspricht R_t bzgl. MC-Konzept und $V(s_t)$ bzgl. DP

Erläuterungen

- Auf der Folie sind die zwei Extremfälle des TD Learnings dargestellt, wobei die untere Variante typisch und üblich ist.
- Oben werden alle Belohnungen aufsummiert, unten wird nur ein Schritt gegangen. Für die untere Variante gibt es daher eine Gemeinsamkeit mit der dynamischen Programmierung, die auch nur einen Schritt bis zum nächsten Zustand berücksichtigt.
- Grundsätzlich kann man auch zwei Schritte gehen und dann den V-Wert des übernächsten Zustands abziehen. Oder drei Schritte und den V-Wert des nach drei Schritten erreichten Zustands, usw.
- Die obere Variante betrachtet wie MC die aufsummierten Belohnungen für alle zukünftigen Schritte. Allerdings findet hier eine Form des Updates statt, die von mehreren aufeinander folgenden Updates die neueren gegenüber den früheren bevorzugt, während in den MC-Methoden alle Durchläufe gleich stark berücksichtigt werden.

SARSA und Q-Learning

- Die beiden wichtigsten Lernverfahren des TD Learning
- Es muss gelernt werden, welche Aktion in welchem Zustand zu wählen ist
  - Also Q-Werte ermitteln
- Im Sinne des TD-Learning:
  - Um einen gewählten Q-Wert zu aktualisieren, ...
  - ... geht man den entsprechenden Schritt, ...
  - ... berücksichtigt die Belohnung als neue Erfahrung ...
  - ... und addiert den Folge-Q-Wert

- **Aber welcher Q-Wert ist der Folge-Q-Wert? (Darin unterscheiden sich SARSA und Q-Learning.)**

Unterschied SARSA und Q-Learning

- Gleich:
  - Nach der entsprechenden Aktion zur Aktualisierung eines Q-Wertes wird gemäß der ε-greedy-Strategie im dann erreichten Zustand eine Folgeaktion gewählt.
- Unterschiedlich:
  - SARSA benutzt den der Folge-Aktion entsprechenden Folge-Q-Wert zur Aktualisierung
    - Das ist meistens der Q-Wert der besten Aktion, also der beste Q-Wert des erreichten Zustands
    - Aber manchmal, d.h. im Fall der Exploration, auch ein schlechterer Q-Wert
  - Q-Learning benutzt immer den besten Q-Wert des erreichten Zustands zur Aktualisierung
    - führt aber genau wie SARSA die gewählte Aktion durch

Vergleich: SARSA - Q-Learning

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

- SARSA-Algorithmus

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until s is terminal

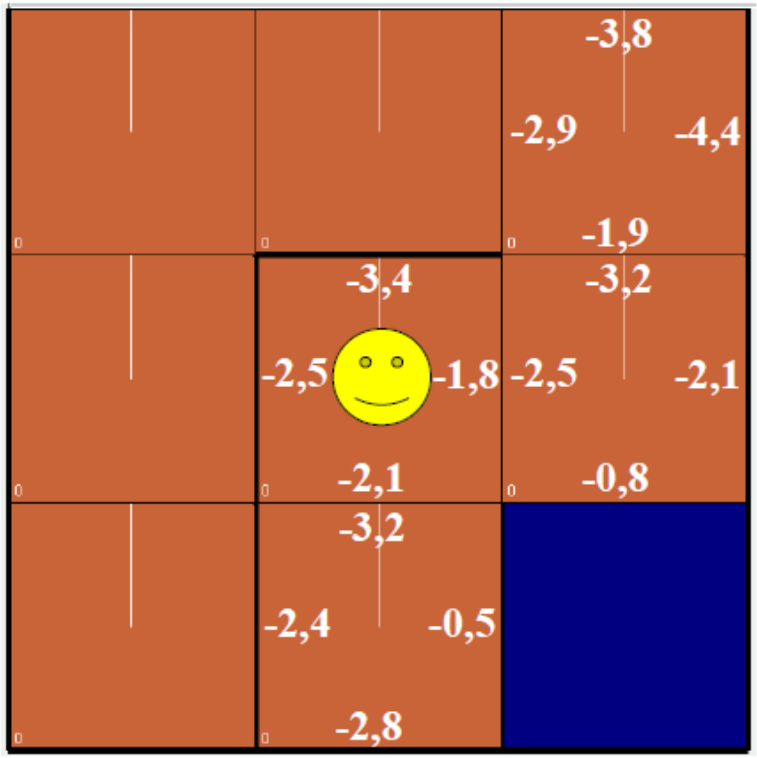
- Q-Learning-Algorithmus

Q-Learning ist ein Off-Policy-Verfahren

- Die den Weg, also das Verhalten bestimmende Aktion, ist also eine andere als die Aktion, die zum Update, also zur Bewertung herangezogen wird.
- Man spricht deshalb auch von einer Behavior Policy und einer Evaluation Policy.
- Wenn beide verschieden sind, liegt ein **Off-Policy**-Verfahren vor.
- Im Falle des Q-Learning betrifft dieses Auseinanderlaufen nur einen Schritt.
- SARSA ist hingegen ein **On-Policy**-Verfahren

Zur Veranschaulichung eine Klausuraufgabe

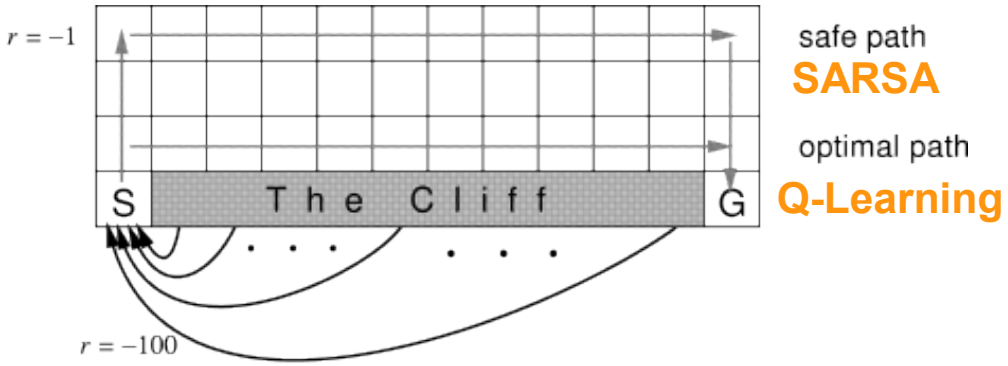
- Welcher Q-Wert wird für SARSA, welcher Q-Wert wird für Q-Learning als nächster aktualisiert, wenn die wahrscheinlichste Aktion gewählt wird?
  - In beiden Fällen der Wert -1,8 rechts vom Agenten
- Wie lautet der neue Q-Wert im SARSA-Verfahren und wie im Q-Learning unter der Annahme, dass die ε-greedy-Strategie ergibt, dass der Agent sich im übernächsten Schritt wieder auf den Platz in der Mitte zurückbewegt. Die Lernrate betrage 0,2, die Discount-Rate 0,8, die Schrittkosten -1,0.



  - SARSA: $Q(s,a) = Q(s,a) + \alpha * [r + \gamma * Q(s',a') - Q(s,a)] =$
    $-1,8 + 0,2 * [-1 + 0,8 * (-2,5) - (-1,8)] = -2,04$
  - Q-Learning: $Q(s,a) = Q(s,a) + \alpha * [r + \gamma * \max_{a'} Q(s',a') - Q(s,a)] =$
    $-1,8 + 0,2 * [-1 + 0,8 * (-0,8) - (-1,8)] = -1,768$

Vergleich: SARSA – Q-Learning am Beispiel

- Belohnungen:
  - Cliff: **-100**
  - Sonst: **-1**
- **ε=0.1**
- **Wie stürzt man über das Kliff in beiden Algorithmen?**
- **Wie entstehen die unterschiedlichen Pfade?**
- **Wie entstehen die unterschiedlichen Rewards pro Episode ?**



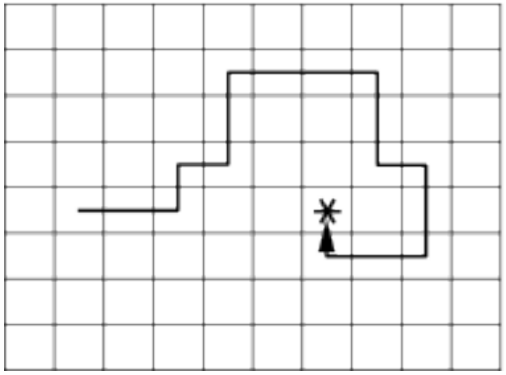
Demo: Q-Learning

SARSA(λ): Grundidee

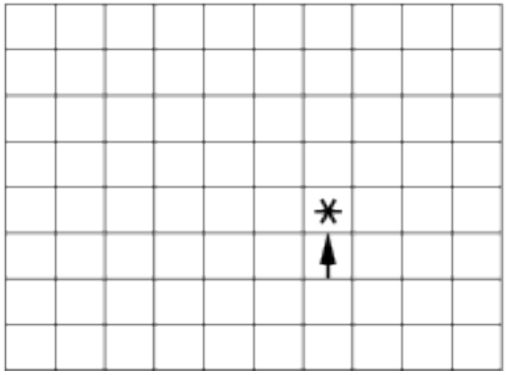
- SARSA(λ) ist das am weitesten verbreitete Lernverfahren im verstärkenden Lernen
- Idee: Für eine Belohnung ist nicht nur die unmittelbar letzte Aktion im letzten Zustand verantwortlich, sondern auch (abgeschwächt) die Vorgänger-Zustände bzw. Vorgänger-Aktionen
  - Beispiel Schachspiel: Der Sieg zeichnet sich meist schon einige Schritte vorher ab
- Belohnungen werden also nicht nur zur Aktualisierung des letzten Q-Wertes benutzt, sondern auch abgeschwächt auf die Vorgänger-Q-Werte übertragen
  - Der Grad der Abschwächung wird durch den Parameter λ gesteuert
  - Für vorigen Schritt λ, für den davor λ^2, usw.
  - $\lambda = 0$: Standard-SARSA
  - $\lambda = 1$: Monte Carlo

SARSA(λ): Veranschaulichung

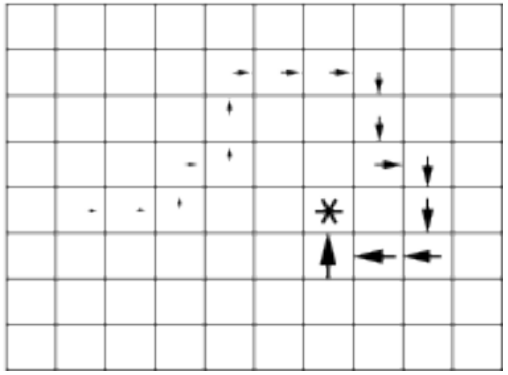
Path taken



Action values increased by one-step Sarsa



Action values increased by Sarsa(λ) with λ=0.9



VL4: Lernziele

- V1: TD mit MC und DP vergleichen können
- V2: TD Learning Grundidee erläutern können
- V3: Unterschiede und Gemeinsamkeiten zwischen SARSA und Q-Learning erläutern können
- V4: Die rot gekennzeichneten Begriffe erläutern können
- V5: Besonderheit des SARSA(λ)-Algorithmus erläutern können

- A1: Einzelnen Berechnungsschritt für SARSA und Q-Learning durchführen können (--> Teil 2 der Klausur)