

Grundlagen der Künstlichen Intelligenz - Informatik

Rapp, DHBW Lörrach

01.12.2023

Inhaltsübersicht

- 1 Motivation
- 2 Reinforcement Learning
- 3 Modellbasierte Wert-Iteration
- 4 Modellfreies Q-Lernen
- 5 SARSA

Lernziele

Meine 3 Lernziele für heute

- ① Ich kenne die Voraussetzungen, um Algorithmen des verstärkenden Lernens einzusetzen.
- ② Ich verstehe die grundlegenden Algorithmen für Wert-Iteration und Q-Lernen.
- ③ Ich kann die nächstbeste Aktion des Agenten aus der optimalen Strategie ableiten.

Literaturempfehlungen

- Grundkurs Künstliche Intelligenz - Eine praxisorientierte Einführung, *Wolfgang Ertel*, Springer Vieweg
- Reinforcement Learning, An Introduction, *Sutton and Barto*, The MIT Press

Motivation und Übersicht

Meilenstein

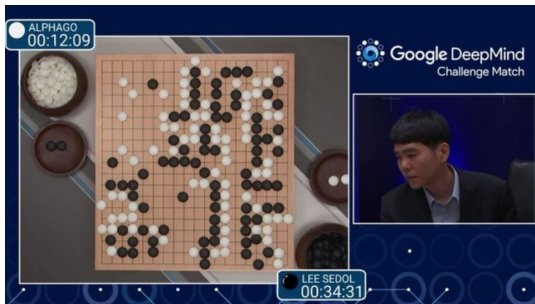
Starcraft II

2019 wurde ein wichtiger Meilenstein der KI erreicht.

Die KI AlphaStar besiegte die elite E-Sportler *TLO* und *MaNa* vom *Team Liquid*.

Meilenstein der KI

AlphaGo gewinnt gegen den Go Master Lee Sedol (2016)



AlphaGo wurde iterativ mit dem strategischen Brettspiel Go trainiert.

Kombination von

- Reinforcement Learning
- Monte Carlo Tree Search
- Künstliche neuronale Netze

Kategorien des maschinellen Lernens

Feedbackarten während des Lernvorgangs Das Feedback, das während dem Lernvorgang zur Verfügung steht, ist ausschlaggebend für die Bestimmung der Lernproblemart:

- **Supervised Learning** (*Überwachtes Lernen*)
Lernt eine Funktion aus den gelabelten Inputdaten
- **Unsupervised Learning:** *Unüberwachtes Lernen*
Der Agent kann lediglich Muster in den Inputdaten erkennen, da keine Labels vorhanden sind
- **Reinforcement Learning:** (*Bestärkendes Lernen*)
Die allgemeinste Form von Lernen. Dem Agenten wird nicht durch einen Lehrer vorgegeben, was er zu tun hat. Er lernt lediglich aus Belohnungen bzw. Bestrafungen, die er aus der Interaktion mit seiner Umgebung erhält.

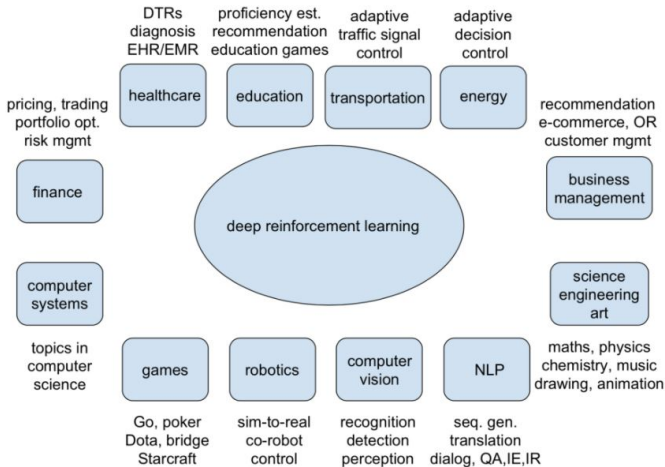
Wesentlicher Unterschied

- Sowohl Supervised, als auch Unsupervised Learning benötigen im Vorhinein bereitgestellte Datensätze für die Modellierung.
- Der Agent im Reinforcement Learning benötigt lediglich einige wenige Anweisungen, um seine Umgebung Schritt für Schritt zu erkunden und daraus die benötigten Daten zu generieren.

Idee von Reinforcement Learning

Der Agent lernt selbständig eine Strategie, die seine Belohnung maximiert.

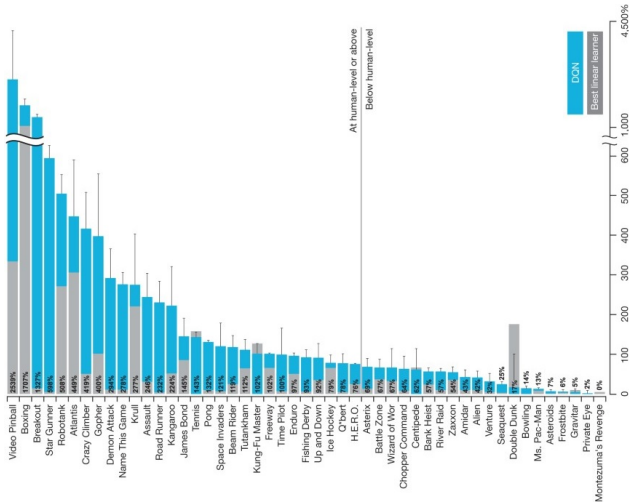
Reinforcement Learning - Anwendungsgebiete



Yuxi Li, Deep Reinforcement Learning, arXiv, 2018

Verstärkendes Lernen eignet sich für den Einsatz in einem breiten Anwendungsspektrum - insbesondere für die Steuerung und Optimierung komplexer Systeme

Deep Q-Learning bei Atari-Spielen



Deep Q-Learning erreicht bzw. übertrifft heutzutage bei einem Großteil der Atari-Spiele die Leistungsfähigkeit menschlicher Spieler. Der Agent erhält als Input lediglich die Pixel sowie Spielpunktzahl. Aus: *Nature, Mnih et al. (2015)*

Reinforcement Learning

Voraussetzungen

Bevor ein Reinforcement-Learning-Algorithmus funktioniert, sind **viele Iterationen** erforderlich.

Das liegt unter anderem daran, dass es **verzögerte Belohnungen** geben kann und diese erst erlernt werden müssen.

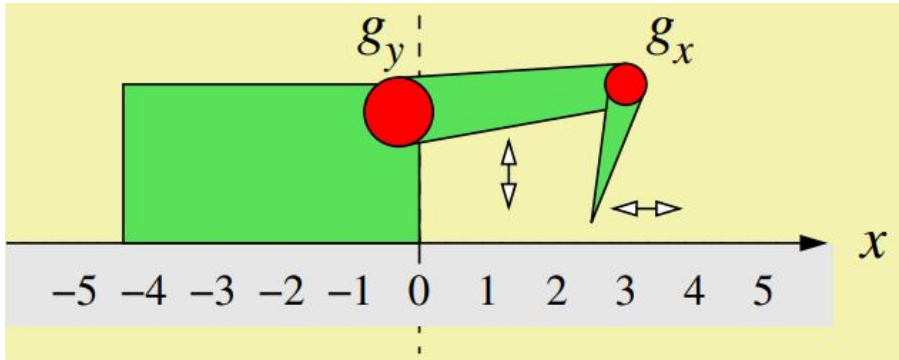
Modellierung des Lernvorgangs als **Markow Entscheidungsprozess** mit:

- ① **Zustandsraum**
- ② **Aktionsraum**
- ③ **Belohnungsfunktion**

Video

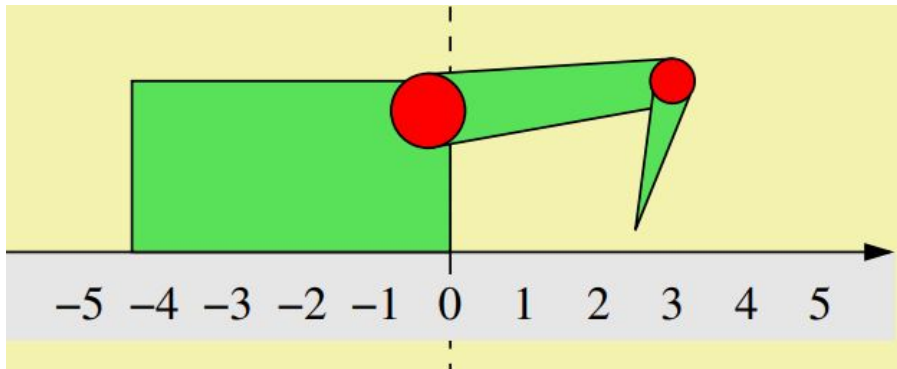
Crawling Robot

Krabbelroboter 1/6

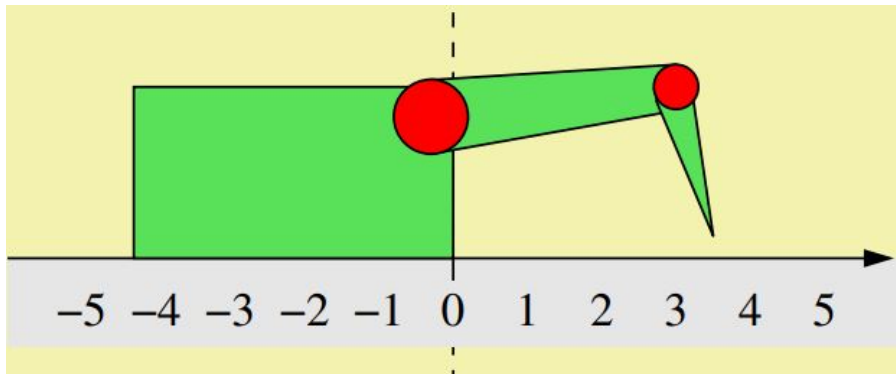


Durch Bewegung der beiden Gelenke kann sich der Krabbelroboter vorwärts und rückwärts bewegen. Das **Feedback** (=Belohnung) für Bewegungen des Roboters nach **rechts ist positiv** und für Bewegungen nach **links negativ**.

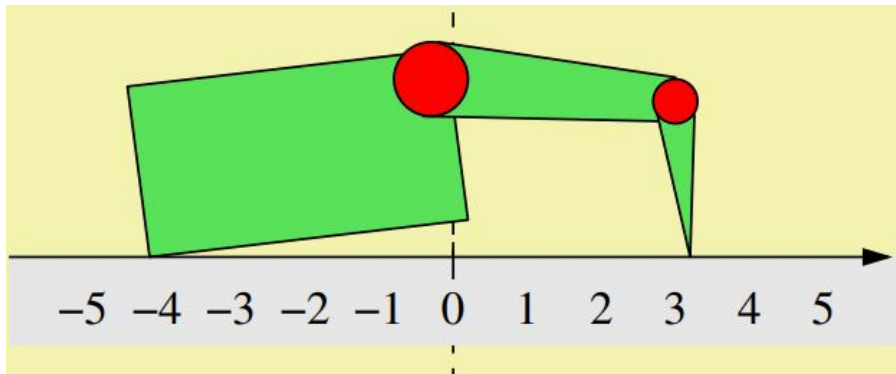
Krabbelroboter 2/6



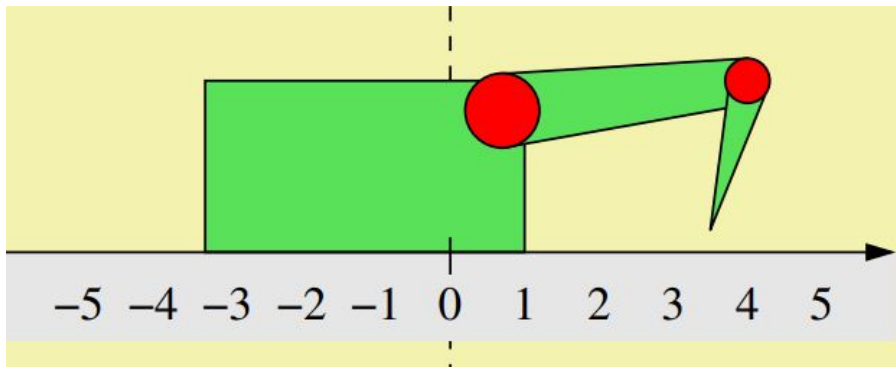
Krabbelroboter 3/6



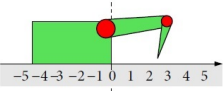
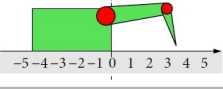
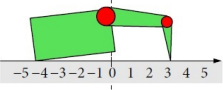
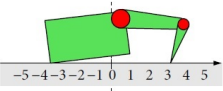
Krabbelroboter 4/6



Krabbelroboter 6/6

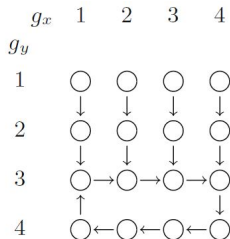
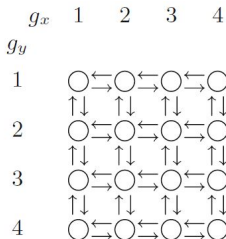
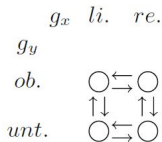


Systematische Vorwärtsbewegung

Krabbleroboter	Zeit t	Zustand		Belohnung x	Aktion a_t
		g_y	g_x		
	0	oben	links	0	rechts
	1	oben	rechts	0	tief
	2	unten	rechts	0	links
	3	unten	links	1	hoch

Ein Zyklus einer periodischen Bewegungsfolge mit systematischer Vorwärtsbewegung.

Zustandsräume des Krabbelroboters



- **Links:** Je zwei mögliche Gelenkpositionen
- **Mitte:** Je vier horizontale und vertikale Gelenkpositionen mit einer entsprechenden **optimalen Strategie** (*rechts*).

Ziel im Reinforcement Learning

Der Agent lernt die **optimale Strategie** basierend auf seinen Erfahrungen mit der Umgebung.

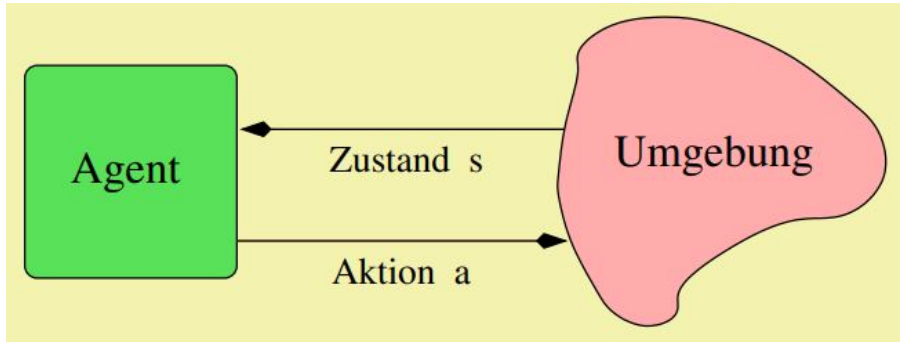
Interaktion des Agenten mit seiner Umgebung

Zustand $s_t \in S$ (=Umgebung)

$$s_t \xrightarrow{a_t} s_{t+1}$$

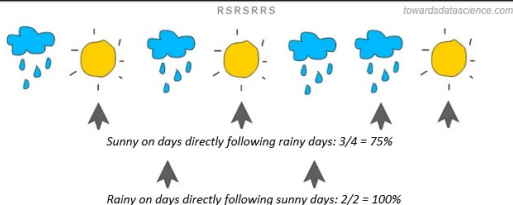
Übergangsfunktion δ

$$s_{t+1} = \delta(s_t, a_t)$$

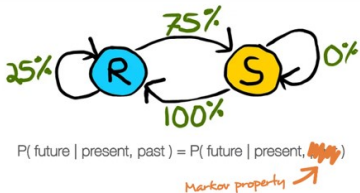


Wettervorhersage als Markov-Prozess erster Ordnung

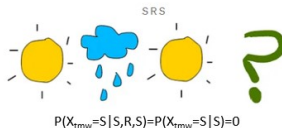
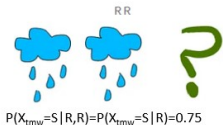
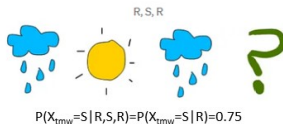
Historical data to „train“ Markov chain



Derived transition probabilities in „memoryless“ Markov chain



Sample predictions for sunny weather



There is a 100% chance of rain tomorrow.
It always rains on days after sun... sad I know...

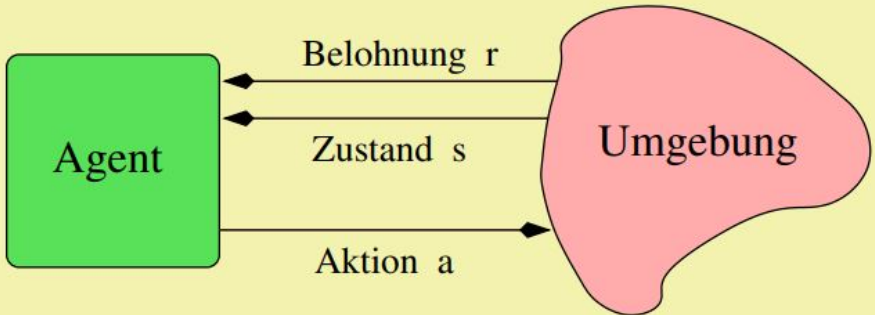
Direkte Belohnung im Markov-Entscheidungsprozess

Nach ausgeführter Aktion a_t erhält der Agent ein Feedback r_t aus der Umgebung in Form einer **direkten Belohnung** r_t (*immediate reward*):

Markov-Eigenschaft

$$r_t = r(s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = r(s_t, a_t)$$

Beim **Lernen** führt $r_t > 0$ zu einer **positiven** und $r_t < 0$ zu einer **negativen Verstärkung der Bewertung** der Aktion a_t im Zustand s_t .



Voraussetzungen für RL

Damit Reinforcement Learning angewendet werden kann, müssen bestimmte Kriterien erfüllt sein:

Verständnis für das zu lösende Problem

- 1 Lässt sich das Problem mit der “Versuch und Irrtum” Methode lösen?
- 2 Kann der Agent durch Interaktion mit der Umgebung Verhaltensstrukturen erlernen?
- 3 Lässt sich das Verhalten belohnen bzw. bestrafen?
- 4 Kann das Problem zu einem Markovschen Entscheidungsprozess modelliert werden?
- 5 Kann die reale Umgebung als abstraktes Modell simuliert werden?

Weitere Fragestellungen nach der Art des Reinforcement Learning Algorithmus

- 1 Soll der Algorithmus On-Policy oder Off-Policy sein?
- 2 Soll der Algorithmus modellbasiert oder modellfrei sein?
- 3 Q-Learning oder SARSA

Es sollte der Algorithmus gewählt werden, der das Problem am effizientesten löst.

Modellbasierte Wert-Iteration

Wert einer Strategie

Eine **Strategie** π ist eine **Abbildung von Zuständen auf Aktionen**:

$$\begin{aligned}\pi : S &\rightarrow A \\ s &\mapsto a\end{aligned}$$

Wert einer Strategie V^π

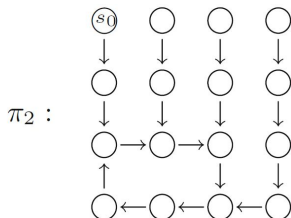
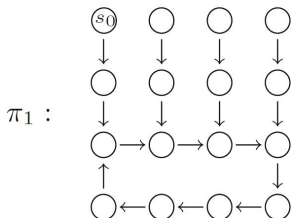
Der Wert einer Strategie ist definiert durch die Funktion:

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

- **Startzustand:** s_t
- **Diskontierungsfaktor:** $0 < \gamma < 1$ (=konstant)

γ sorgt dafür, dass eine Belohnung umso stärker abgeschwächt wird, je weiter sie in der Zukunft liegt. Je größer γ , desto **vorausschauender** die Strategie. Die **direkte Belohnung** r_t wird am stärksten gewichtet.

Vergleich von Strategien



Zwei verschiedene Strategien π_1 und π_2 für den Krabbelroboter

Bewegung nach rechts wird mit 1 **belohnt**, nach links mit -1 **bestraft**.

Die linke Strategie π_1 ist langfristig die bessere, da der mittlere Vortrieb pro Aktion für π_1 gleich $3/8 = 0.375$ und π_2 gleich $2/6 \approx 0.333$ ist.

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

γ	0,9	0,8375	0,8
$V^{\pi_1}(s_0)$	2,52	1,156	0,77
$V^{\pi_2}(s_0)$	2,39	1,156	0,80

größeres γ : größerer Zeithorizont für die Bewertung von Strategien!

Bellman-Gleichung

Optimale Strategie π^*

Eine Strategie ist **optimal**, wenn sie **langfristig**, das heißt über viele Schritte, die **Belohnung maximiert**:

$$V^{\pi^*}(s) \geq V^{\pi}(s), \quad \text{für alle Zustände } s$$

Aus dem **Wert einer Strategie** V^{π} und Optimalitätskriterium $V^{\pi^*}(s) \geq V^{\pi}(s)$ folgt für den **Wert der optimalen Strategie** die

Bellman-Optimalitätsgleichung

$$V^{\pi^*}(s) = \max_a [r(s, a) + \gamma V^{\pi^*}(\delta(s, a))]$$

Der Algorithmus für die
Iterationsvorschrift konvergiert gegen V^{π^*}
(Sutton, Barto)
→ *Dynamische Programmierung*

For all $s \in \mathcal{S}$

$$\hat{V}(s) = 0$$

Repeat

For all $s \in \mathcal{S}$

$$\hat{V}(s) = \max_a [r(s, a) + \gamma \hat{V}(\delta(s, a))]$$

Until $\hat{V}(s)$ sich nicht mehr ändert

Wert-Iteration beim Krabbelroboter

Wert-Iteration mit $\gamma = 0.9$ und zwei optimalen Strategien (unten rechts).

Zahlen an den Pfeilen: direkte Belohnung $r(s, a)$ der jeweiligen Aktion.

Jede Iteration: Zeilenweise Abarbeitung der Zustände von links unten nach rechts oben.

0	$\xrightarrow{0}$	0	$\xrightarrow{0}$	0
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
0	$\xrightarrow{0}$	0	$\xrightarrow{0}$	0
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
0	$\xrightarrow{1}$	0	$\xrightarrow{1}$	0
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow

Initialisierung

0	$\xrightarrow{0}$	0.81	$\xrightarrow{0}$	1.54
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
0	$\xrightarrow{0}$	0.9	$\xrightarrow{0}$	1.71
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
0	$\xrightarrow{1}$	1	$\xrightarrow{1}$	1.9
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow

1. Iteration

0.73	$\xrightarrow{0}$	1.39	$\xrightarrow{0}$	1.54
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
0.81	$\xrightarrow{0}$	1.54	$\xrightarrow{0}$	1.71
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
0	$\xrightarrow{1}$	1	$\xrightarrow{1}$	1.9
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow

2. Iteration

1.25	$\xrightarrow{0}$	1.39	$\xrightarrow{0}$	2.02
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
1.39	$\xrightarrow{0}$	1.54	$\xrightarrow{0}$	2.24
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
0.73	$\xrightarrow{1}$	1.66	$\xrightarrow{1}$	2.49
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow

...

1.25	$\xrightarrow{0}$	1.82	$\xrightarrow{0}$	2.36
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
1.39	$\xrightarrow{0}$	2.02	$\xrightarrow{0}$	2.62
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
1.25	$\xrightarrow{1}$	2.12	$\xrightarrow{1}$	2.91
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow

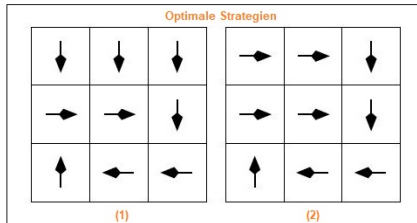
...

2.66	$\xrightarrow{0}$	2.96	$\xrightarrow{0}$	3.28
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
2.96	$\xrightarrow{0}$	3.28	$\xrightarrow{0}$	3.65
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow
2.66	$\xrightarrow{1}$	3.39	$\xrightarrow{1}$	4.05
\downarrow	\downarrow	\downarrow	\downarrow	\downarrow

$$V^{\pi^*}$$

$$\hat{V}(s) = \max_a [r(s, a) + \gamma \hat{V}(\delta(s, a))]$$

Optimale Strategien



Achtung: Es ist falsch, die Aktion zu wählen, welche zum Zustand mit maximalem V^{π^*} -Wert führt. Warum?

Optimale Strategie $\pi^*(s)$

Auswahl der Aktion a im Zustand s in der optimalen Strategie

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} [r(s, a) + \gamma V^{\pi^*}(\delta(s, a))]$$

Aufgabe

Anwendung auf $s = (2, 3)$ in

2.66 0 0	2.96 0 0	3.28 0 0
2.96 0 0	3.28 0 0	3.65 0 0
2.66 -1 -1	3.39 -1 -1	4.05 -1 -1

V^*

Optimale Strategie $\pi^*(s)$

Auswahl der Aktion a im Zustand s in der optimalen Strategie

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} [r(s, a) + \gamma V^{\pi^*}(\delta(s, a))]$$

Anwendung auf $s = (2, 3)$ in

$2.66 \xrightarrow{0} 2.96 \xrightarrow{0} 3.28$
$2.96 \xrightarrow{0} 3.28 \xrightarrow{0} 3.65$
$2.66 \xrightarrow{1} 3.39 \xrightarrow{1} 4.05$

V^*

$$\begin{aligned}
 \pi^*(2, 3) &= \underset{a \in \{\text{links}, \text{rechts}, \text{oben}\}}{\operatorname{argmax}} [r(s, a) + \gamma V^*(\delta(s, a))] \\
 &= \underset{\{\text{links}, \text{rechts}, \text{oben}\}}{\operatorname{argmax}} \{1 + 0.9 \cdot 2.66, -1 + 0.9 \cdot 4.05, 0 + 0.9 \cdot 3.28\} \\
 &= \underset{\{\text{links}, \text{rechts}, \text{oben}\}}{\operatorname{argmax}} \{3.39, 2.65, 2.95\} \\
 &= \text{links}
 \end{aligned}$$

Modellfreies Q-Lernen

Unbekannte Welt?

Zentrale Fragestellung

Was tun, wenn der Agent kein Modell der Welt hat, d.h. wenn er nicht weiß, in welchen Zustand ihn eine mögliche Aktion führt und welche Belohnung er dafür erhält:

$s_{t+1} = \delta(s_t, a_t)$ und $r(s_t, a_t)$ **sind unbekannt?**

Hinweis: Dies ist für die **meisten praktischen Anwendungen** der Fall.

Beispiel: Ein Roboter, der komplexe Objekte greifen soll, kann in vielen Fällen nicht vorhersagen, ob nach einer Greifaktion das zu greifende Objekt fest in seinem Greifer sitzt oder noch an seinem Platz liegt.

Antwort

Wir benötigen ein Verfahren, das **ohne die Kenntnis** von δ und r arbeitet.

→ **Q-Lernen.**

Q-Lernen

Bei fehlendem **Modell der Welt** wird eine Bewertung einer im Zustand s_t ausgeführten Aktion a_t benötigt, auch wenn noch unbekannt ist, wohin diese Aktion führt.

Zur Definition der **Bewertungsfunktion Q der Aktion a_t im Zustand s_t** verwenden wir wieder das **schrittweise Abschwächen der Bewertung** für zukünftige Zustands-Aktions-Paare:

$$Q(s_t, a_t) = \max_{a_{t+1}, a_{t+2}, \dots} (r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \dots)$$

und bringen diese Gleichung analog zur *Wert-Iteration* in eine rekursive Form (*Watkins in '89*):

Iterationsvorschrift für Q-Lernen

$$\hat{Q}(s, a) = r(s, a) + \gamma \max_{a'} \hat{Q}(\delta(s, a), a')$$

Zu Beginn **kennt** der Agent die möglichen Zustände s und Aktionen a , jedoch sind ihm die Funktionen r und δ **unbekannt!**

Algorithmus für Q-Lernen

Gelöst wird dieses Problem pragmatisch dadurch, dass wir den **Agenten** in seiner **Umgebung** im **Zustand** s eine **Aktion** a ausführen lassen:

Q-LERNEN()

For all $s \in \mathcal{S}, a \in \mathcal{A}$

$\hat{Q}(s, a) = 0$ (oder zufällig)

Repeat

Wähle (z.B. zufällig) einen Zustand s

Repeat

Wähle eine Aktion a und führe sie aus

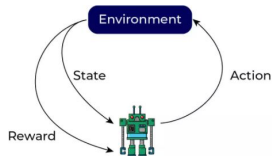
Erhalte Belohnung r und neuen Zustand s'

$\hat{Q}(s, a) := r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$

$s := s'$

Until s ist ein Endzustand Oder Zeitschranke erreicht

Until \hat{Q} konvergiert



Der **Nachfolgezustand** ist dann offenbar $\delta(s, a)$ und die **Belohnung** $r(s, a)$ erhält der Agent von der Umgebung!

Optimale Strategie beim Q-Lernen

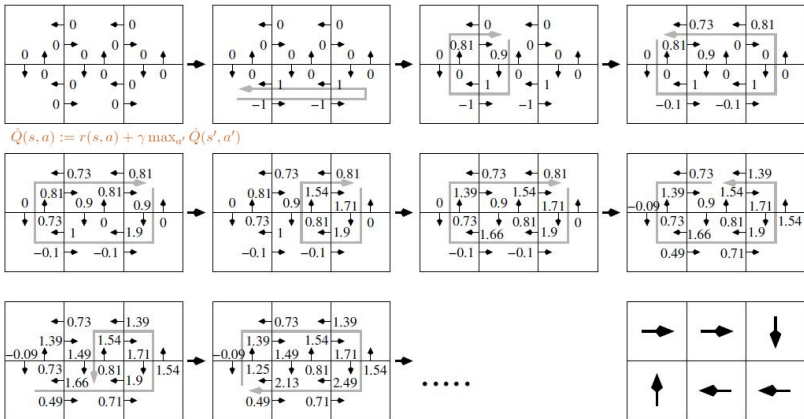
Die Auswahl der **optimalen Strategie** beim Q-Lernen erfolgt durch:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q(s, a)$$

Die optimale Strategie π^* in einem Zustand s ergibt sich somit durch die Wahl derjenigen Aktion a , die die **Bewertungsfunktion Q maximiert**.

Anwendung für $\gamma = 0.9$, $n_x = 3$, $n_y = 2$

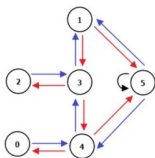
\hat{Q} -Tabelle mit aktualisierten Q -Werten. Die grauen Pfeile stellen die durch den Agenten ausgeführten Aktionen dar und das letzte Bild die optimale Strategie.



Die **Q-Tabelle** ist eine Lookup-Tabelle, die Werte für maximal zukünftig erwartete Belohnungen von Aktionen in den jeweiligen Zuständen enthält.

Python Beispiel

Q-Learning example in Python



1. Labyrinth mit insgesamt 6 Zuständen

	Action					
State	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

$R =$

2. Reward-Matrix (von der Umgebung vorgegeben)

```

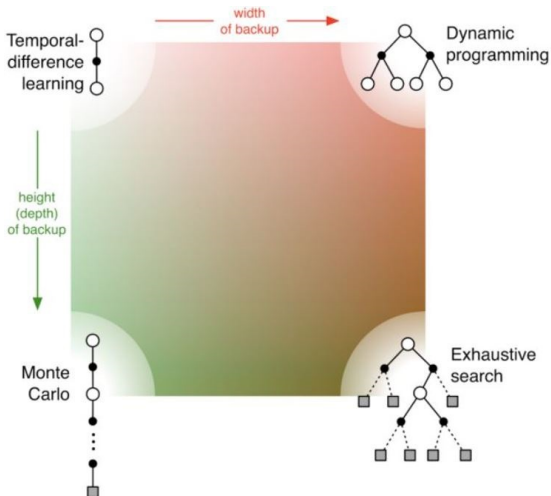
99 # Trained Q matrix:
100 # [[ 0.    0.    0.    0.    80.    0. ]
101 # [ 0.    0.    0.    64.    0.   100. ]
102 # [ 0.    0.    0.    64.    0.    0. ]
103 # [ 0.   80.   51.2  0.   80.    0. ]
104 # [ 0.   80.   51.2  0.    0.   100. ]
105 # [ 0.   80.    0.    0.   80.   100. ]]

```

3. Trainierte Q-Matrix
(nach 10.000 Iterationen).
Beispiel für idealen Pfad: 2->3->1->5

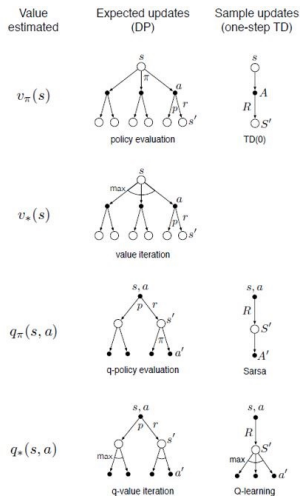
SARSA

Breite und Tiefe bei Reinforcement Learning Methoden



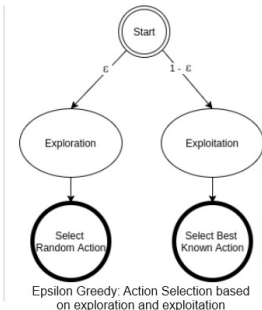
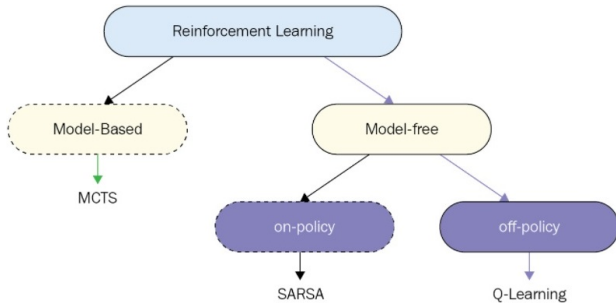
Methods of reinforcement learning vary in width and depth of backup.

a) Top-left: Temporal difference learning e.g. Q-learning b) Top-right: Dynamic programming e.g. value iteration



Q-learning (bottom-right): take an action a in a state s and observe reward R . Then determine next action a' with maximum Q-value.

Off- und On-Policy Reinforcement Learning Algorithmen



Off-Policy

Die Annahme, dass zukünftige Belohnungen zur bestmöglichen Aktion führen, kann von der Strategie (z.B. ϵ -Greedy) abweichen.

On-Policy

Der Agent bleibt bei der Auswahl seiner Folgeaktion seiner Strategie treu.

SARSA - Idee

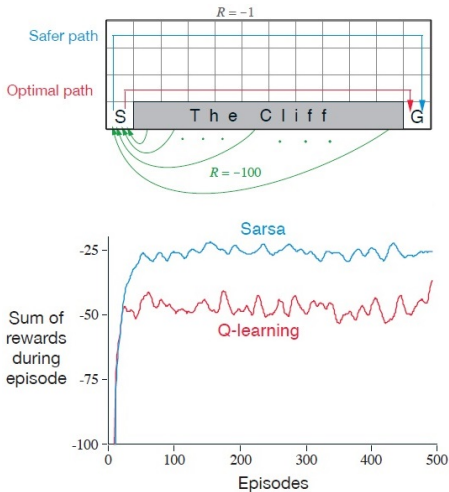
State-Action-Reward-State-Action (SARSA)

Analog zum Q-Lernen ist SARSA ein **Algorithmus zum Erlernen der Bewertungsfunktion Q** der Aktion a_t im Zustand s_t . Im Gegensatz zu Q-Learning bleibt der Agent allerdings bei der Berechnung seiner Folgeaktion seiner Strategie treu (on-policy).

→ *“On-Policy Variante des Q-Lernens“*

- Der Agent führt im aktuellen Zustand s eine Aktion a gemäß seiner Strategie aus und erhält eine Belohnung r .
- Im Folgezustand s' wählt er nun wieder eine **Aktion a' gemäß seiner Strategie** und nimmt dessen Bewertung als zukünftigen Gewinn, um die Bewertungsfunktion anzupassen.

SARSA vs. Q-Lernen



Aus: Sutton and Barto, Reinforcement Learning, An Introduction

Lernkontrolle

Meine 3 Lernziele für heute waren

- ① Ich kenne die Voraussetzungen, um Algorithmen des verstärkenden Lernens einzusetzen.
- ② Ich verstehe die grundlegenden Algorithmen für Wert-Iteration und Q-Lernen.
- ③ Ich kann die nächstbeste Aktion des Agenten aus der optimalen Strategie ableiten.