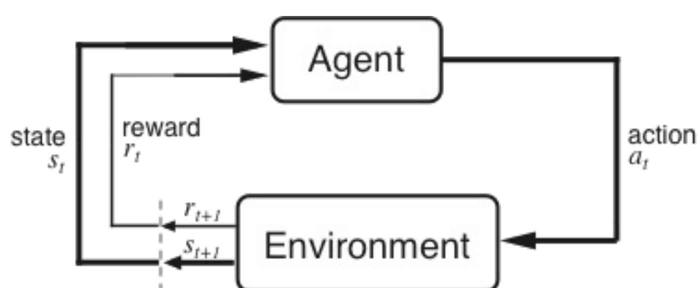
Professur
Künstliche Intelligenz

Kapitel 7: Reinforcement Lernen

Prof. Dr. Fred Hamker
Department of Computer Science

Reinforcement Lernen



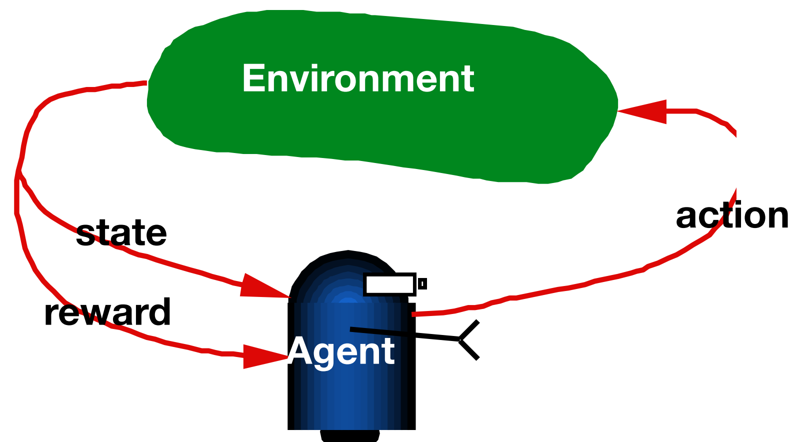
Inhalt:

- Elemente des Reinforcement Lernens
- Die Markov Eigenschaft
- Value Funktionen
- Bellmann Gleichungen
- Policy Evaluation
- TD Methoden
- SARSA
- Q-Learning
- Actor-Critic
- Eligibility traces
- Deep Q-Network

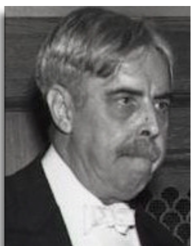
Empfohlene Literatur:
- R. S. Sutton, A. G. Barto: Reinforcement Learning: An Introduction. MIT Press, 1998
- Second edition: MIT Press Cambridge, Massachusetts London, England, 2018, 2020

RL Agent

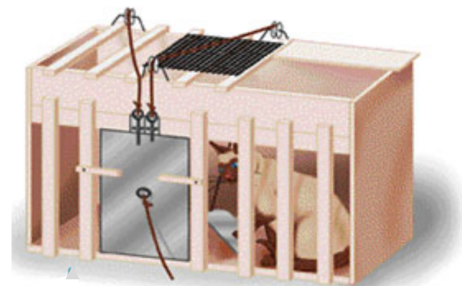
- Zielgerichtetes Lernen ohne Lehrer
- Agent exploriert und nutzt sein Wissen
- Agent verändert seinen Zustand in einer Umwelt durch eine Aktion
- Umwelt ist stochastisch und unsicher

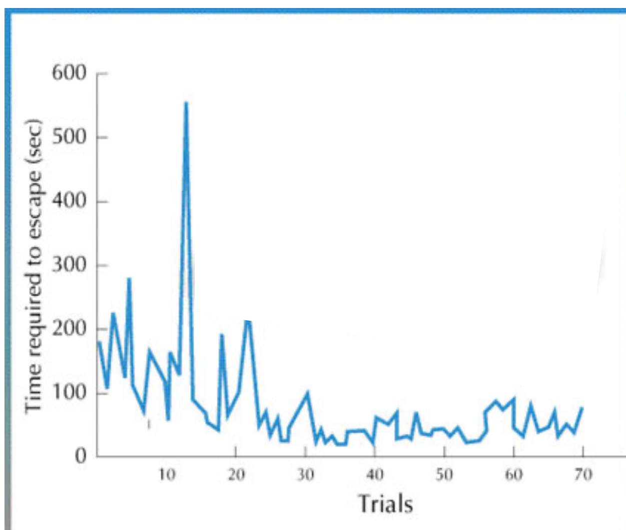


Thorndike's Katzenpuzzle



E. L. Thorndike
(1874 - 1949)





Eine hungrige Katze wird in einen Käfig gesetzt.

Wenn das Tier die angemessene Reaktion zeigte, öffnete sich die Tür der Kammer und die Katze konnte nach draußen, um das dort platzierte Futter zu fressen.

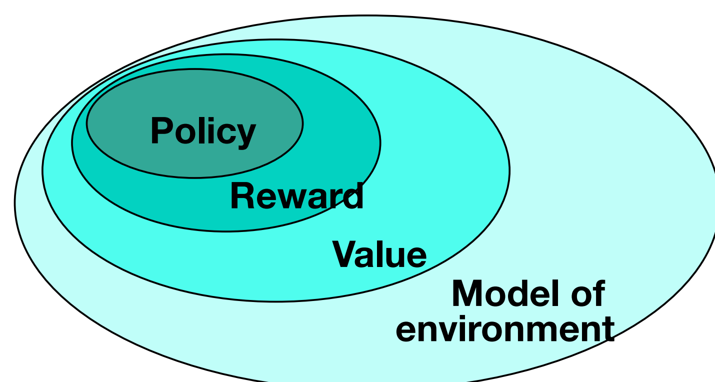
Angemessene Reaktionen waren hier: an einer Schnur ziehen, auf eine Plattform treten und eines der beiden Schnappschlösser an der Vorderseite der Tür drehen.

Operante Konditionierung

B. F. SKINNER
(1904 - 1990)

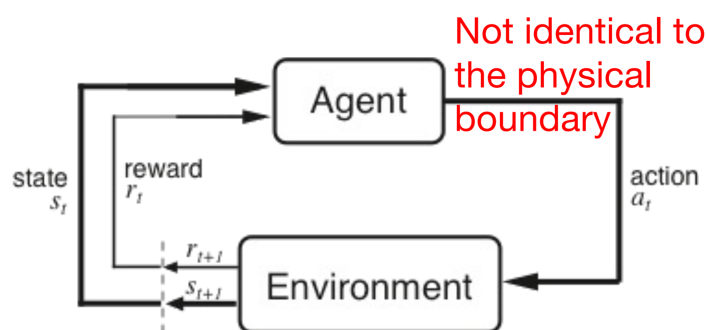
- Verhalten geschieht auch ohne einen externen Stimulus.
- Spontanes Verhalten im Gegensatz zu Reaktionen
- Verstärkung von spontanem Verhalten: pos. Reinforcement führt zur Verstärkung von Verhalten, neg. Reinforcement führt zur Vermeidung.

Elemente des Reinforcement Lernens



- **Policy**: was soll gemacht werden
- **Reward**: was ist gut
- **Value**: Zustände o.ä. die gut sind, da sie Belohnung vorhersagen
- **Model**: was kommt wann

Agent-Umwelt Interface



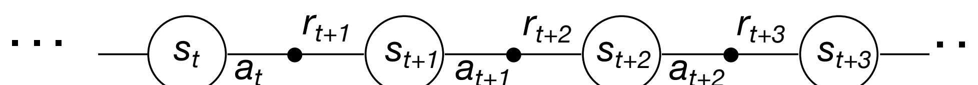
Agent und Umwelt interagieren in Zeitschritten: $t = 0,1,2, \ldots$

Agent "beobachtet" Zustant zur Zeit t: $s_t \in S$

produziert eine Aktion zur Zeit t: $a_t \in A(s_t)$

erhält eine Belohnung : $r_{t+1} \in \Re$

und kommt in den Zustand : s_{t+1}



Belohnungsfunktion

- Bestimmt das Ziel des Lernens.
- Verknüpft einen Zustand S (oder Zustands-Aktions Paar) mit einer Belohnung (*num* Wert)

$$r : S \rightarrow \Re$$

- Das Ziel des Agenten ist die Maximierung der gesamten Belohnung über einen langen Zeitraum.

Policy

- Bestimmt das Verhalten des Agenten zu einem bestimmten Zeitpunkt
- Eine Policy π beschreibt die Zuordung eines Zustandes S zu Aktionen A, die in dem Zustand gewählt werden

$$\pi: S \rightarrow A$$

- Policies können zufällige Komponenten beinhalten
- Ziel: die optimale Policy

Policy

Policy zum Zeitpunkt t, π_t :

eine Zuordung von Zuständen zu Wahrscheinlichkeiten von Aktionen

$\pi_t(s, a) =$ Wahrscheinlichkeit, dass $a_t = a$ wenn $s_t = S$

$$s_t \in S \longrightarrow a_t \in A(s)$$

$\pi^*: S \rightarrow A$ Optimale Policy

Reinforcement Lernmethoden beschreiben, wie der Agent seine Policy anhand von Erfahrung verändert.

Value

- Spezifiziert was auf lange Sicht hin gut ist
- Der *Wert* eines Zustandes ist der Erwartungswert der gesamten Belohnung, die ein Agent erwerben kann, ausgehend von diesem Zustand
- *Belohungen* bestimmen die unmittelbare Gewünschtheit von Zuständen, *Werte* zeigen die langfristige Gewünschtheit von Zuständen an
- Die Methode, nach der der Wert von Zuständen bestimmt wird, ist der zentrale Aspekt der RL Lernens
- Such-Methoden wie genetische Algorithmen suchen direkt im Raum der Policies ohne Value-Funktionen zu generieren

Model

- Ein Modell speichert die Zusammenhänge der Umwelt
- Gegeben die Zustände und Aktionen, sagt ein Modell den nächsten Zustand und die zu erwartende Belohnung vorher
- Modelle können auch für die Planung verwendet werden
- Die Dynamische Programmierung verwendet Modelle

Returns

Sequenz von Belohungen nach dem Zeitschritt t:

$$r_{t+1}, r_{t+2}, r_{t+3}, \dots$$

Ziel:

Maximimierung des **expected return**, $E\{R_t\}$, für jeden Zeitschritt t.

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

Episodische Aufgaben: Interaktion findet im Rahmen von natürlichen Episoden statt, e.g., Spieldurchgang, Labyrinth.

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

T ist der finale Zeitschritt, bei dem ein **terminaler Zustand** erreicht wurde, wo die Episode endet.

Returns

Continuing tasks: Interaktion hat keine natürlichen Episoden (T=∞).

Discounted return:

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

wobei $\gamma, 0 \leq \gamma \leq 1$ die **discount Rate** ist.

kurzsichtig $0 \leftarrow \gamma \rightarrow 1$ weitsichtig

Die Markov Eigenschaft

- Der Zustand zum Zeitpunkt *t*, berücksichtigt alle Information, die bis zum Erreichen des Zeitpunkts *t* über die Umwelt verfügbar ist.
- Idealerweise, frühere Information so aufbereiten, dass die **Markov Eigenschaft** gilt:

$$\Pr\{ s_{t+1} = s', r_{t+1} = r \,|\, s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0 \} = \Pr\{ s_{t+1} = s', r_{t+1} = r \,|\, s_t, a_t \}$$

für alle s', r, und $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0$.

Pr: Probabilty

Auch wenn der Zustand die Markov Eigenschaft nicht erfüllt, macht es häufig Sinn, dieses näherungsweise anzunehmen.

Markov Decision Processes

- Wenn die RL Aufgabe die Markov Eigenschaft erfüllt, handelt es sich um einen **Markov'schen Entscheidungsprozess.**
- Wenn Zustände und Aktionen endlich sind, muss folgendes definiert werden:
  - **Menge von Zuständen und Aktionen**
  - **Zustandsüberganswahrscheinlichkeiten**:

$$\boldsymbol{P}^a_{ss'} = \Pr\{ s_{t+1} = s' \,|\, s_t = s, a_t = a \} \text{ for all } s, s' \in S, a \in A(s).$$

  - **Belohnungswahrscheinlichkeiten**:

$$\boldsymbol{R}^a_{ss'} = \mathrm{E}\{ r_{t+1} \,|\, s_t = s, a_t = a, s_{t+1} = s' \} \text{ for all } s, s' \in S, a \in A(s).$$

deterministisch:

$$P^a_{ss'} = \begin{cases} 1 & \text{if } s' = s(a) \\ 0 & \text{else} \end{cases} \qquad R^a_{ss'} = \mathrm{r(s,a)}$$

Value Functions V und Q

Der **Wert eines Zustands** ist der erwartete Return, ausgehend von diesem Zustand. Er ist von der Policy des Agenten abhängig:

State – value Funktion für die Policy π:

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

Der **Wert einer Aktion in einem Zustand** unter Verwendung der Policy π ist der erwartete Return, ausgehend von diesem Zustand und der gewählten Aktion:

Action – value Funktion für die Policy π:

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

Value Functions V und Q

State-value Funktion und Action-value Funktion sind miteinander über die Wahrscheinlichkeit, bestimmte Aktionen auszuwählen, verknüpft:

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s, a) \cdot Q^{\pi}(s, a)$$

Wahrscheinlichkeit im Zustand *s* die Aktion *a* auszuwählen.

Bellman Gleichungen für eine Policy π

Idee:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$
$$= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots)$$
$$= r_{t+1} + \gamma R_{t+1}$$

$$V^\pi(s) = E_\pi\{ R_t | s_t = s \}$$
$$= E_\pi\{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s \}$$

Wahrscheinlichkeit von *s*‘, falls im Zustand *s* die Aktion *a* gewählt wird.

Belohnung *r* beim Übergang von *s* nach *s*‘ unter Aktion *a*.

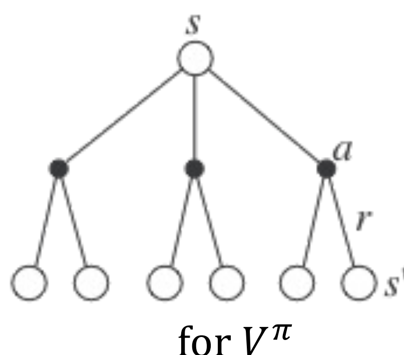
$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^\pi(s')]$$

Bellman Equation for a Policy π

$$V^\pi(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} P^a_{ss'} \cdot [R^a_{ss'} + \gamma \cdot V^\pi(s')]$$

Die Bellman Gleichung beschreibt den Zusammenhang zwischen dem Wert eines Zustands s und den Werten der Nachfolgezustände s’.

Backup diagram:



Wir können den Wert eines Zustandes s unter Kenntnis der Werte der Nachfolgezustände s’ berechnen.

Bellman Gleichung für Q und V

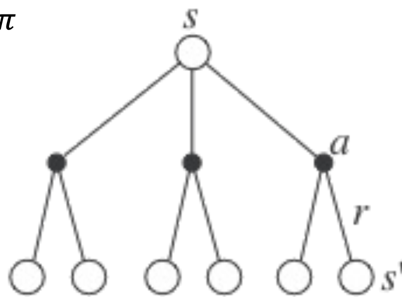
$$Q^{\pi}(s,a) = \sum_{s' \in S} P^a_{ss'} \cdot \left[R^a_{ss'} + \gamma \cdot \sum_{a' \in A(s')} \pi(s',a') \cdot Q^{\pi}(s',a')\right]$$

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} P^a_{ss'} \cdot \left[R^a_{ss'} + \gamma \cdot V^{\pi}(s')\right]$$

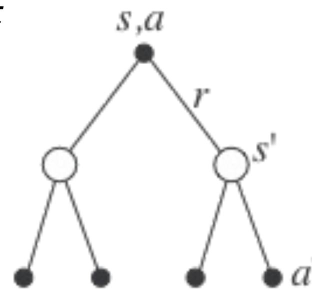
Die Gleichungen beschreiben ein Set von linearen Gleichungen – eine für jeden Zustand.
Die Value Funktion für π ist ihre Lösung.

Backup diagrams:

for V^{π}



for Q^{π}



Bellman Gleichung für Q und V

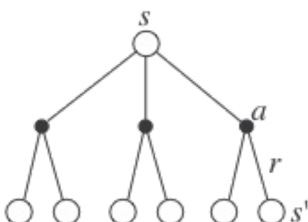
Q^{π} und V^{π} beziehen sich aufeinander:

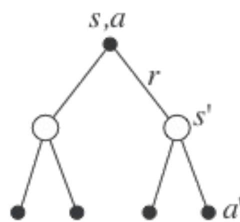
$$Q^{\pi}(s,a) = \sum_{s' \in S} P^a_{ss'} \cdot \left[R^a_{ss'} + \gamma \cdot V^{\pi}(s')\right]$$

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \cdot Q^{\pi}(s,a)$$

$$Q^{\pi}(s,a) = \sum_{s' \in S} P^a_{ss'} \cdot \left[R^a_{ss'} + \gamma \cdot \sum_{a' \in A(s')} \pi(s',a') \cdot Q^{\pi}(s',a')\right]$$

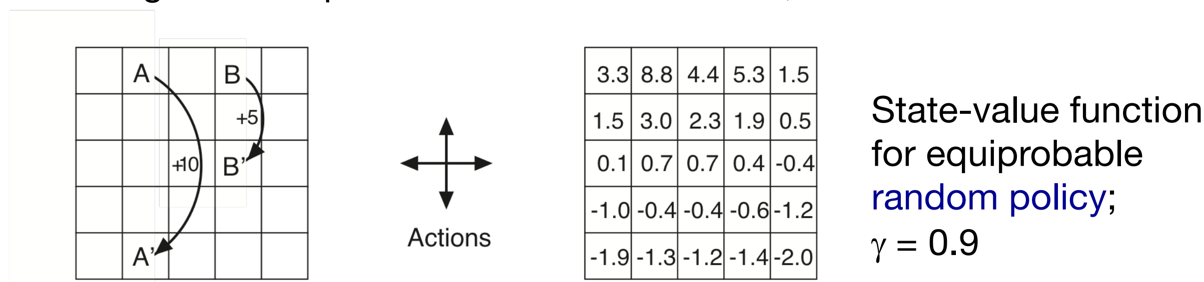
$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \cdot Q^{\pi}(s,a)$$





Gridworld

- Aktionen: `north`, `south`, `east`, `west`; deterministisch.
- Aktion, die den Agenten außerhalb des Grid befördert: keine Bewegung aber Belohnung = –1
- Andere Aktionen ergeben Belohnung = 0, außer die Aktionen, die den Agenten in spezielle Zustände befördern, wie in A und B.



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

- Beachte die negativen Werte an der unteren Kante: Bei einer zufälligen Wahl der Aktion (random policy) überschreitet der Agent häufig die Grenze.
- State A ist der beste Zustand, aber der erwartete Return ist geringer als 10 da der Zustand A' nah an der Kante liegt.
- State B, hat einen Wert höher als 5, da er zu B' führt.

Optimal Value Funktionen

Eine Policy π ist besser oder gleich einer policy π' wenn der erwartete Return größer oder gleich ist (für alle Zustände):

$$\pi \geq \pi' \text{ if and only if } V^{\pi}(s) \geq V^{\pi'}(s) \quad \forall\, s \in S$$

Es gibt zumindest eine Policy, die gleich oder besser als alle anderen ist. Dies ist die **optimal policy** π^*.

Es gilt:

$$V^*(s) := V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s) \quad \forall\, s \in S$$

$$Q^*(s,a) := Q^{\pi^*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \forall\, s \in S, a \in A(s)$$

$$Q^*(s,a) = E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\}$$

Bellman-Optimalitäts-Gleichung für *V**

Der Wert eines Zustands unter Verwendung der besten Policy ist gleich des zu erwartenden Returns für die beste Aktion ausgehend von diesem Zustand:

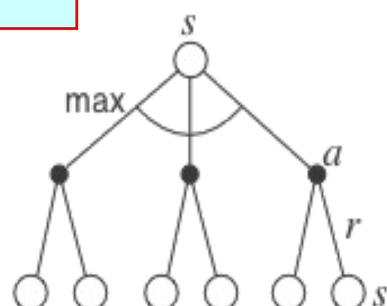
$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\
&= \max_{a \in A(s)} E_{\pi^*}\{R_t \mid s_t = s,\, a_t = a\} \\
&= \max_{a \in A(s)} E_{\pi^*}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s,\, a_t = a\right\} \\
&= \max_{a \in A(s)} E_{\pi^*}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s,\, a_t = a\right\} \\
&= \max_{a \in A(s)} E\{r_{t+1} + \gamma\, V^*(s_{t+1}) \mid s_t = s,\, a_t = a\} \\
&= \max_{a \in A(s)} \sum_{s'} \boldsymbol{P}^a_{ss'} \left[\boldsymbol{R}^a_{ss'} + \gamma\, V^*(s')\right]
\end{aligned}
$$

Bellman-Optimalitäts-Gleichung für *V**

Der Wert eines Zustands unter Verwendung der besten Policy ist gleich des zu erwartenden Returns für die beste Aktion ausgehend von diesem Zustand:

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\
&= \max_{a \in A(s)} E\{r_{t+1} + \gamma\, V^*(s_{t+1}) \mid s_t = s,\, a_t = a\} \\
&= \max_{a \in A(s)} \sum_{s'} \boldsymbol{P}^a_{ss'} \left[\boldsymbol{R}^a_{ss'} + \gamma\, V^*(s')\right]
\end{aligned}
$$

Backup Diagramm:

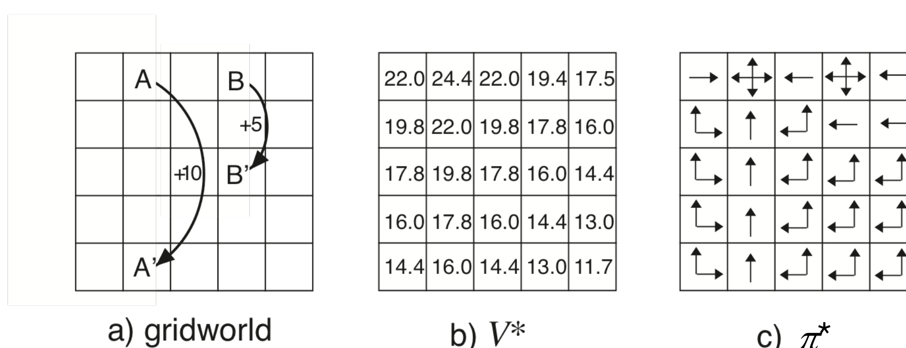


Eine Suche über die Menge der Aktionen in Zustand *s* reicht für ein optimales Verhalten aus

Bellman-Optimalitäts-Gleichung für V^*

Jede greedy Policy mit Bezug auf V^* ist eine optimale Policy.

Gridworld Beispiel:



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

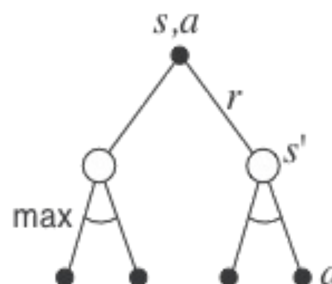
a) gridworld b) V^* c) π^*

Bellman-Optimalitäts-Gleichung für Q^*

$$Q^*(s,a) = E\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\right\}$$

$$= \sum_{s'} \boldsymbol{P}^a_{ss'} \left[\boldsymbol{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right]$$

Backup Diagramm:



Keine Suche notwending!

Bellman-Optimalitäts-Gleichung für *Q**

Unter Verwendung von Q* ist die Wahl der besten Aktion noch einfacher:

Mit Q* benötigt der Agent keine Suche: für jeden Zustand *s* muss lediglich die Aktion gewählt werden, die Q*(*s*,*a*) maximiert.

Die Action-value Gleichung speichert die Ergebnisse einer one-step-ahead Suche.

Mit der optimalen Action-value Funktion wird kein weiteres Wissen über die Umwelt benötigt.

Bellman-Optimalitäts-Gleichung für *Q**

Unter Verwendung von Q* benötigt der Agent nichtmals eine one-step-ahead Suche:

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

$$Q^{\pi^*}(s,a) = \sum_{s' \in S} \boldsymbol{P}^a_{ss'} \cdot \left[\boldsymbol{R}^a_{ss'} + \gamma \cdot V^{\pi^*}(s')\right]$$

$$V^{\pi^*}(s) = \max_a Q^{\pi^*}(s,a)$$

Lösung der Bellman-Optimalitäts-Gleichung

- Die Bestimmung der besten Policy durch Lösung der Bellman-Optimalitäts-Gleichung erfordert:
  - exaktes Wissen über die Dynamik der Umwelt;
  - genug Rechenzeit und Speicherplatz;
  - Die Markov Eigenschaft.
- Wie viel Speicher und Zeit benötigen wir?
  - Eine exakte Lösung muss alle Möglichkeiten durchsuchen. In der Regel ist die Anzahl der Zustände sehr hoch (Backgammon hat ca. 10^{20} Zustände).
- Für typische reale Fälle brauchen wir eine approximative Lösung.
- **Die meisten RL Methoden können so verstanden werden, als dass sie die Bellman-Optimalitäts-Gleichung approximieren.**

Policy Evaluierung

Policy Evaluierung:

für eine gegebene Policy π, wird die state-value function V^π bestimmt

Einfache Monte Carlo Methode:

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

(R_t:) Tatsächlicher Return nach der Zeit *t*

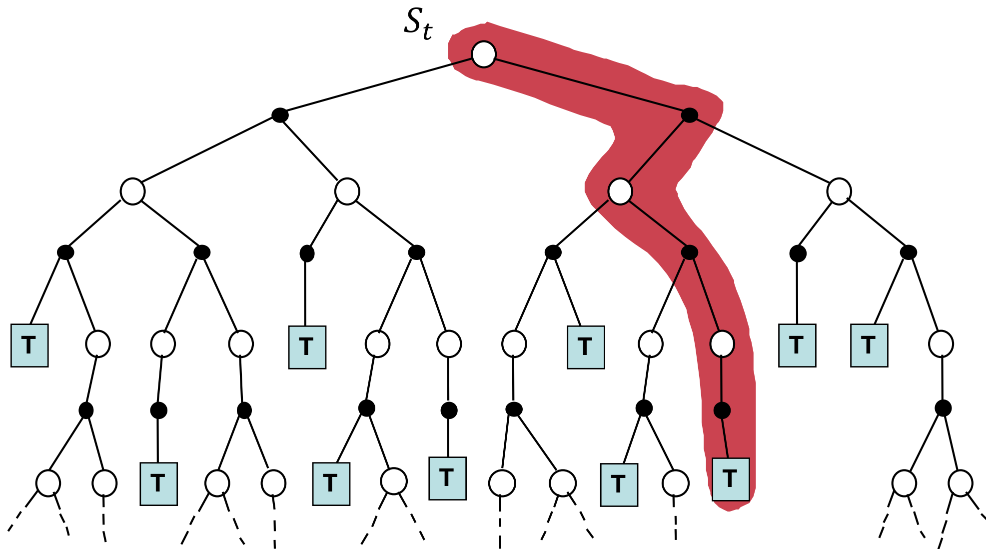
Einfachste TD Methode, TD(0):

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

($r_{t+1} + \gamma V(s_{t+1})$:) Schätzung des Return

Policy Evaluierung

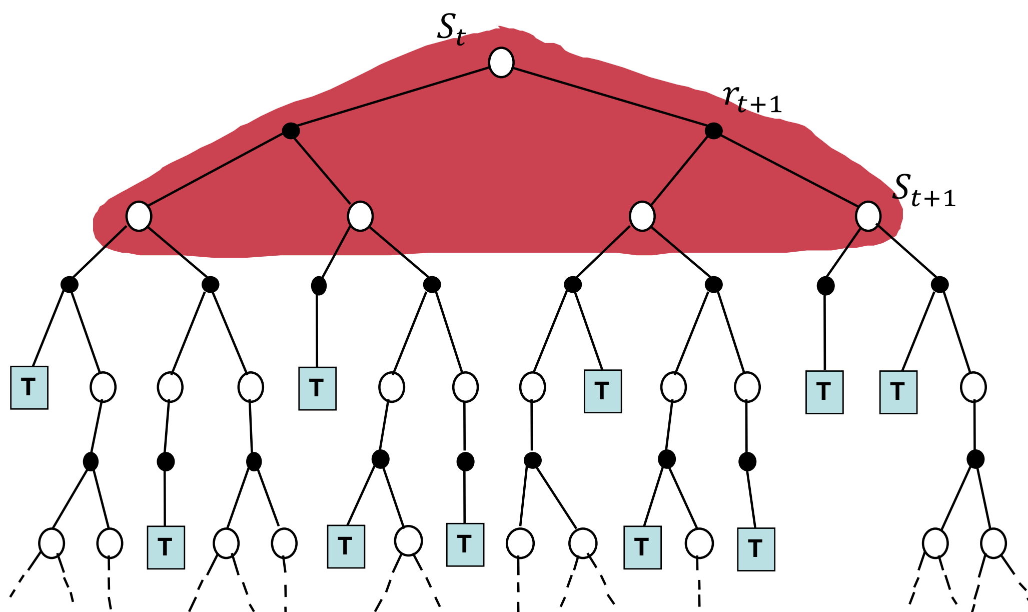
$V(s_t) \leftarrow V(s_t) + \alpha[\, R_t - V(s_t)]$ Einfaches Monte Carlo



Monte Carlo verwendet den tatsächlichen Return zur Schätzung des Wertes.

Policy Evaluierung

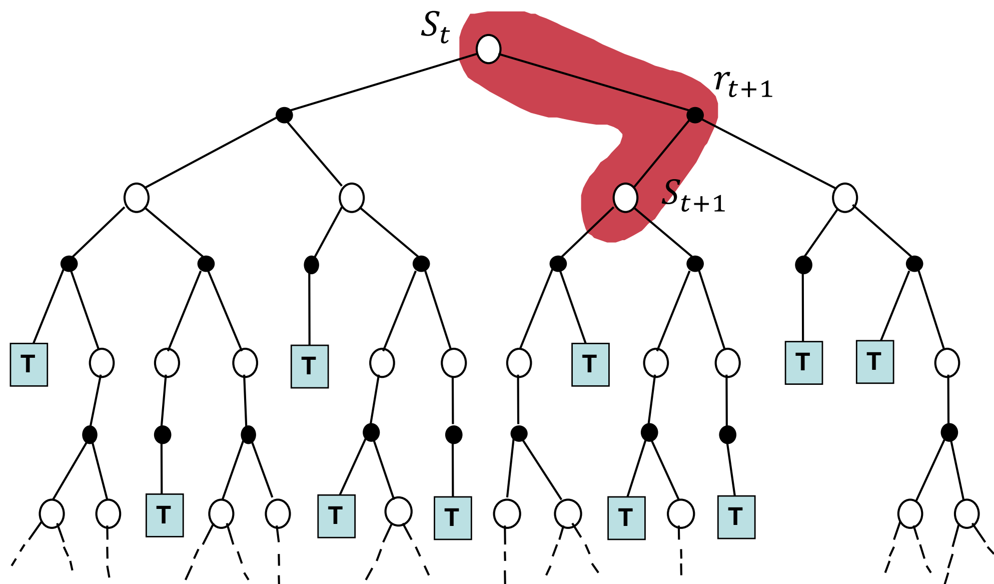
$V(s_t) \leftarrow E_\pi\{r_{t+1} + \gamma V(s_{t+1})\}$ Dynamische Programming



In der DP sind die Zustandsübergänge der Umwelt “bekannt”, allerdings ist V^π nicht exakt bekannt und die aktuelle Schätzung wird stattdessen verwendet.

Policy Evaluierung

$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$ Einfache TD Methode



TD verwendet Stichproben und verwendet die aktuelle Schätzung des Wertes.

Policy Evaluierung

Von Monte Carlo zu einfachem TD

$$
\begin{aligned}
V^\pi(s) &= E_\pi\{R_t \mid s_t = s\} \\
&= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \\
&= E_\pi\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\right\} \\
&= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\}
\end{aligned}
$$

TD(0) - Policy-Evaluierung

Tabular TD(0) for estimating V^{π}

Initialize V(s) arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
Initialize s
Repeat (for each step of the episode):
a ← action given by π for s
Take action a; observe reward, r, and next state, s′

$$V(s) \leftarrow V(s) + \alpha[\underbrace{r + \gamma V(s') - V(s)}_{\text{"temporal Difference"}}]$$

$s \leftarrow s'$

until s is terminal

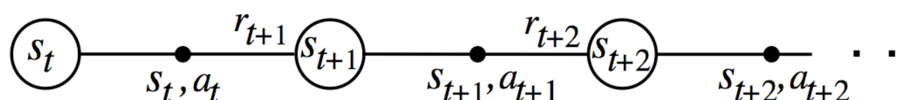
TD Methoden:

- Bootstrapping: Update verwendet eine *Schätzung* des Wertes des Nachfolgezustandes
  - MC: kein Bootstrapping
  - DP: Bootstrapping
  - TD: Bootstrapping
- Sampling: Update verwendet Stichproben der Nachflogezustände
  - MC: Sampling
  - DP: kein Sampling
  - TD: Sampling

Sarsa: On-Policy TD Methode

Sarsa: Lernt eine Action-Value Funktion

Schätzt Q^{π} für die aktuelle Policy π.



Nach jedem Zustandsübergang s_t, update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Falls s_{t+1} ein terminaler Zustand, dann $Q(s_{t+1}, a_{t+1}) = 0$.

Sarsa: On-Policy TD Methode

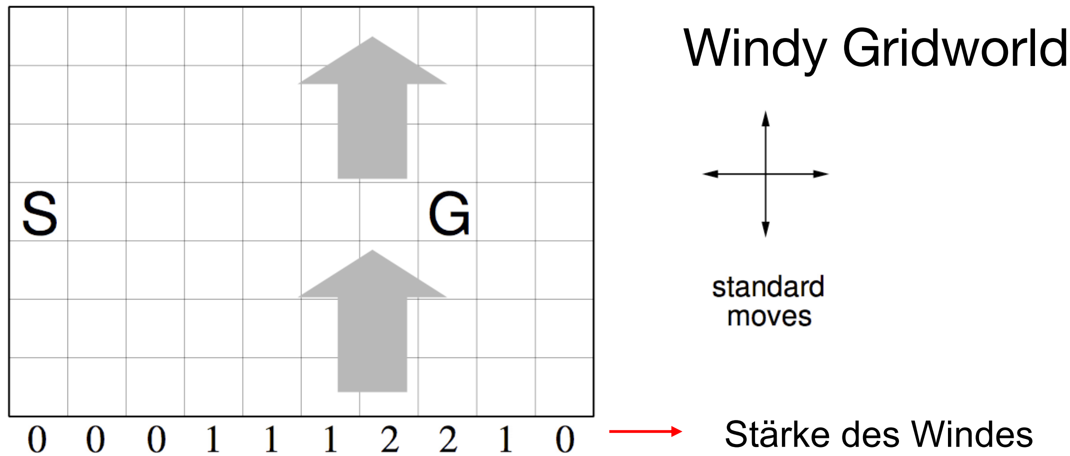
Policy ist immer greedy bezüglich der aktuellen Schätzung.

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from $Q(e.g., \varepsilon - greedy)$
  Repeat (for each step of the episode):
    Take action a, observe r, s′
    Choose a′ from s′ using policy derived from $Q(e.g., \varepsilon\text{–} greedy)$ ← On-Policy
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'; a \leftarrow a';$
  until s is terminal

$(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ Quintupel

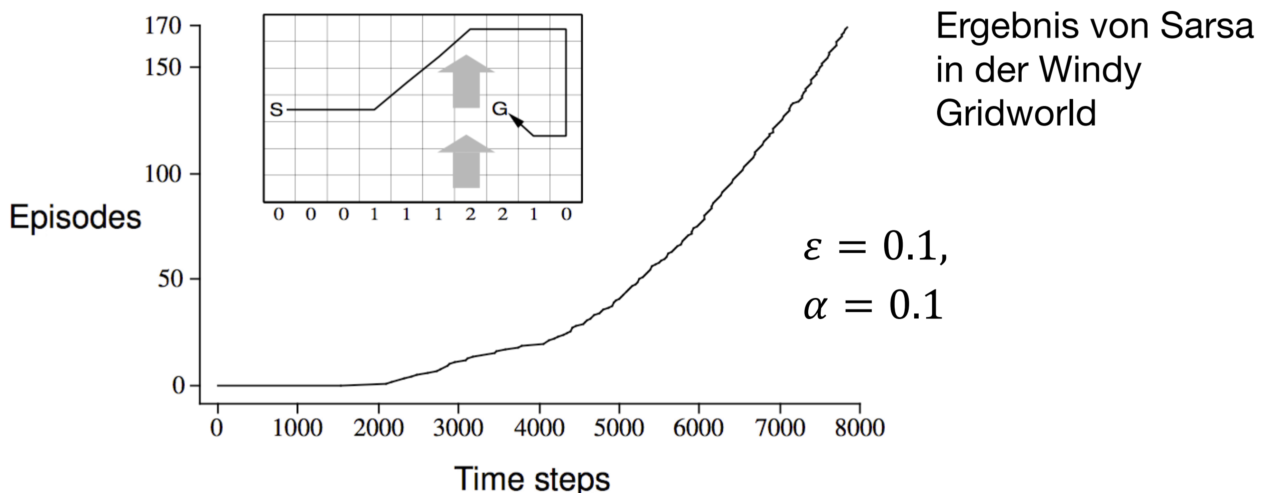
Sarsa: On-Policy TD Methode

Gehe von S nach G, aber der Wind schiebt Dich in der angegebenen Stärke nach oben.



undiscounted, episodische Aufgabe mit konstanten Belohungen
reward = –1 bis Ziel erreicht

Sarsa: On-Policy TD Methode



Können Monte Carlo Methoden verwendet werden?

Problematisch, da eine Terminierung nicht für alle Policies gesichert ist.

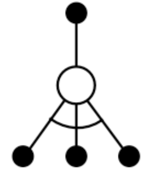
Und Sarsa?

Step-by-step Lernmethoden wie Sarsa haben das Problem nicht, da sie schon während der Episode lernen, dass eine Policy nicht gut ist und daher das Verhalten anpassen.

Q-Learning: Off-Policy TD Methode

$$One\text{-}step\ \ Q\text{-}learning:$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$



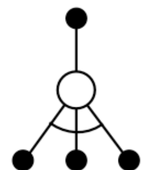
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of the episode):
    Choose a from s using policy derived from $Q(e.g., \varepsilon\text{-}greedy)$
    Take action a, observe r, s′
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s',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

Off-Policy

Q-Learning: Off-Policy TD Methode

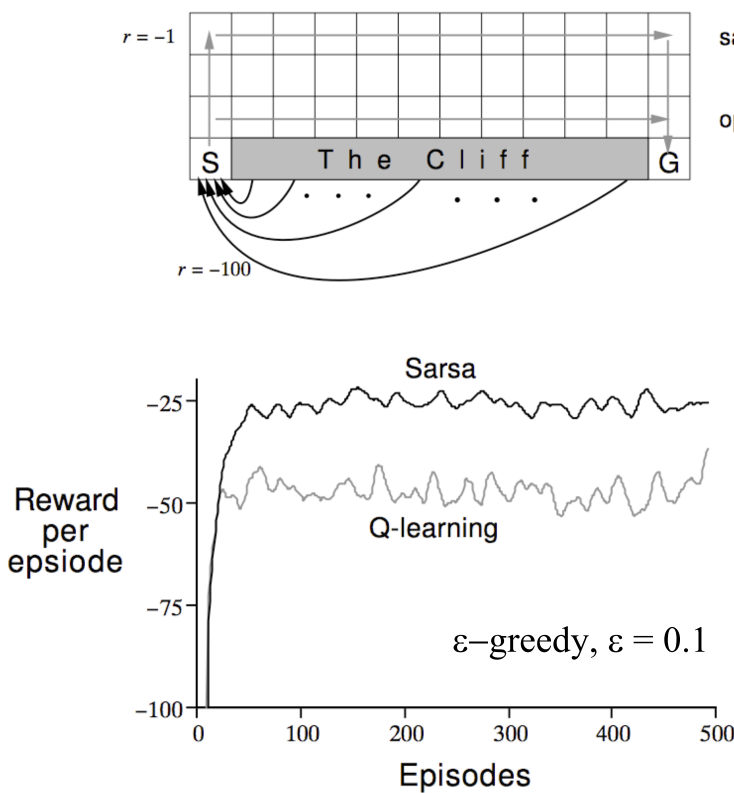
$$One\text{-}step\ \ Q\text{-}learning:$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$



- Die gelernte Action-Value Funktion approximiert direkt die optimale Action-Value Funktion, unabhängig der aktuellen Policy.
- Die Policy hat aber dennoch einen Einfluss insoweit als sie bestimmt, welche Zustands-Aktionspaare aufgesucht werden.

Beispiel: Cliffwalking

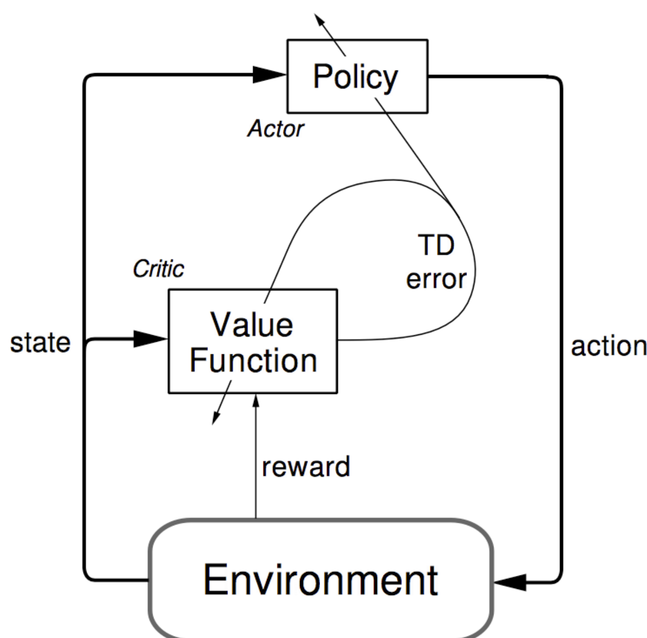


Reward = -1 außer in der Region "The Cliff" (r=-100)

Q-learning lernt schnell die Werte der besten Policy, die entlang der Kante zum Abhang führt. Aufgrund der ε-greedy Selektion führt dieses allerdings manchmal zum Absturz (r=-100).
Sarsa berücksichtigt die aktuelle Policy beim Lernen und lernt daher den längeren aber sichern Pfad.

Wenn der Betrag von ε langsam reduziert wird, dann konvergieren beide Methoden gegen die optimale Policy.

Actor-Critic Methoden



- Explizite Repräsentation der Policy und der Value Funktion
- Critic bestimmt das Lernen
- On policy Methode

Actor-Critic Details

Die Critic ist eine Zustands-Value Funktion. Nach jeder Selektion einer Aktion evaluiert die Critic den neuen Zustand in der Hinsicht, ob die Lage besser oder schlechter als erwartet ist. Die Evaluation nennt man “TD Error”:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

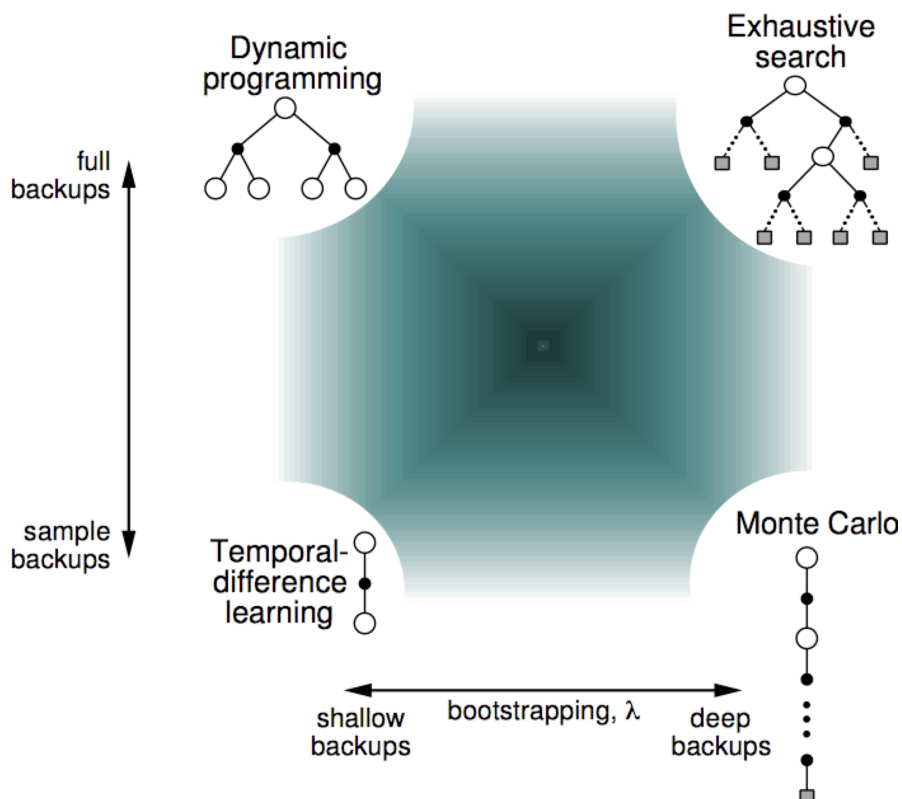
Wenn beispielsweise Aktionen durch die Präferenzen, p(*s*,*a*) bestimmt werden:

$$\pi_t(s,a) = Pr\{\, a_t = a |\, s_t = s \,\} = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}},$$

Dann hängt die Verstärkung oder Abschwächung der Präferenzen, p(*s*,*a*), vom TD Error ab (β – step size parameter):

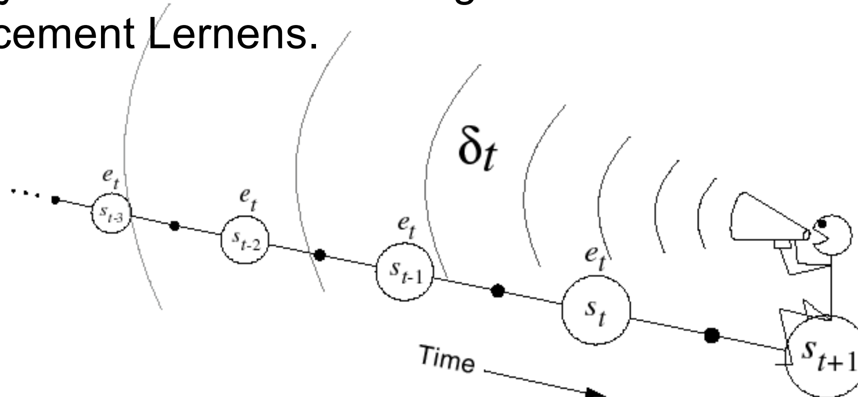
$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t$$

Zusammenfassung



Eligibility Traces der TD(λ) Methode

Eligibility Traces sind ein wichtiger Mechanismus des Reinforcement Lernens.



$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

- Rufe δ_t rückwärts in der Zeit
- Die Stärke der Stimme fällt mir der Distanz durch $\gamma\lambda$

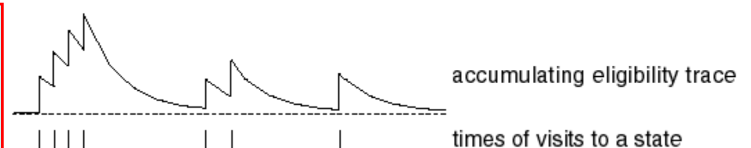
Eligibility Traces der TD(λ) Methode

Eine neue Variable *Eligibility Trace* sichert eine Spur der kürzlich besuchten Zustände *s*, die mit der Zeit *t* aktualisiert wird

$$e_t(s) \in \Re^+$$

Nach jedem Schritt wird die Spur durch $\gamma\lambda$ reduziert und die des neuen Zustands um 1 erhöht.

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & if\ s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & if\ s = s_t \end{cases}$$



γ Discount Rate

λ Trace-decay Parameter

Eligibility Traces der TD(λ) Methode

Die Spuren zeigen auf, inwieweit die jeweiligen Zustände erlaubt sind, am Lernen teilzunehmen.

$$\delta_t = r_{t+1} + \gamma \cdot V_t(s_{t+1}) - V_t(s_t)$$

One-step TD Error

Der TD Error triggert proportional Updates von allen kürzlich besuchten Zuständen:

$$\Delta V_t(s) = \alpha \delta_t e_t(s) \quad \forall s \in S$$

Eligibility Traces der TD(λ) Methode

Initialize $V(s)$ arbitrarily
Repeat (for each episode)
  $e(s) = 0, for\ all\ s \in S$
  Initialize s
  Repeat (for each step of the episode)
    a ← action given by π for s
    Take action a, observe reward r, and next state s′
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    $For\ all\ s$
      $V(s) \leftarrow V(s) + \alpha\delta e(s)$
      $e(s) \leftarrow \gamma\lambda e(s)$
    $s \leftarrow s'$;
  Until s is terminal

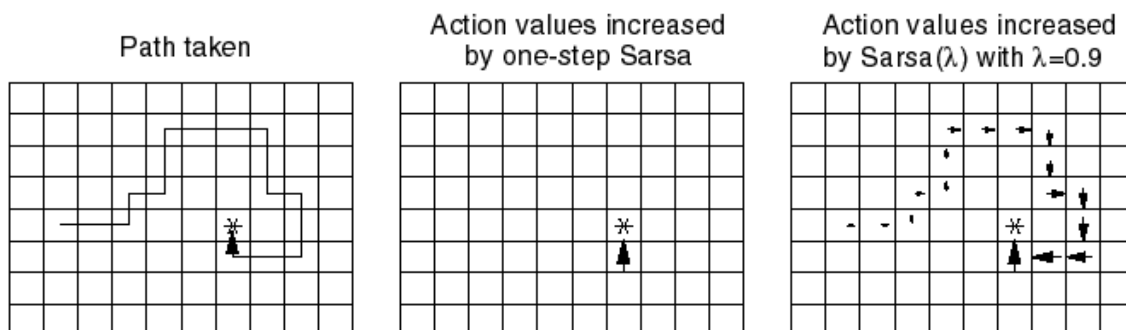
Sarsa(λ)

Algorithm:

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  $e(s,a) = 0, for\ all\ s,a$
  Initialize s, a
  Repeat (for each step of the episode):
    Take action a, observe r, s′
    Choose a′ from s′ using policy derived from $Q(e.g., \varepsilon\text{–}greedy)$
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda e(s,a)$
    $s \leftarrow s'; a \leftarrow a';$
  Until s is terminal

Sarsa(λ)

Beispiel Gridworld :



- Mit nur einem Durchgang hat der Agent deutlich mehr Information, wie er zum Ziel kommen kann
  - nicht notwendigerweise der *beste* Pfad
- Kann das Lernen stark beschleunigen

Q(λ)

Off-policy Methoden wie Q-learning haben ein Problem bei dem Update von zufälligen Aktionen (Exploration), da dann ein Backup über eine non-greedy Policy durchgeführt werden müsste.

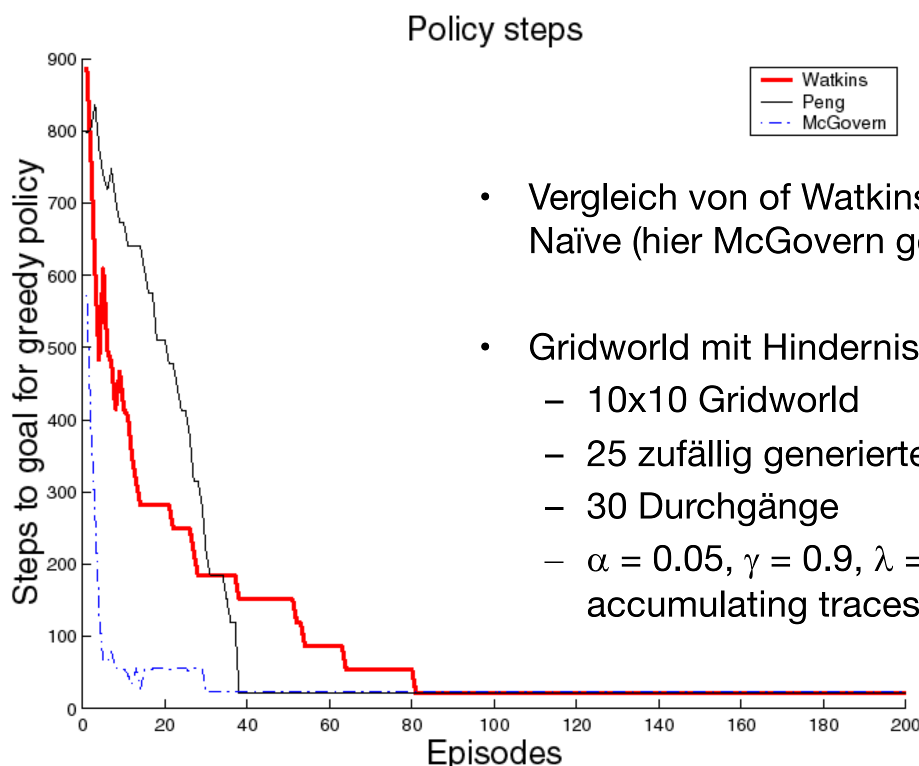
Drei Lösungsansätze von Q(λ):

Watkins: Rücksetzen der Eligibility Traces nach einer non-greedy Aktion. Nachteil: In der Frühphase des Lernens werden die Spuren häufig zurückgesetzt und somit sind die Spuren eher kürzer.

Peng: Frühe Übergänge sind on-policy und spätere verwenden eine greedy Policy. Komplizierter zu Implementieren.

Naïve: Spuren werden einfach nicht zurückgesetzt. Theoretisch falsch, aber in der Praxis häufig nicht problematisch.

Vergleich der Q-Learning Methoden



- Vergleich von of Watkins's, Peng's, und Naïve (hier McGovern genannt) Q(λ)
- Gridworld mit Hindernissen
  - 10x10 Gridworld
  - 25 zufällig generierte Hindernisse
  - 30 Durchgänge
  - $\alpha = 0.05$, $\gamma = 0.9$, $\lambda = 0.9$, $\varepsilon = 0.05$, accumulating traces

Von: McGovern and Sutton (1997). Towards a better Q(λ)

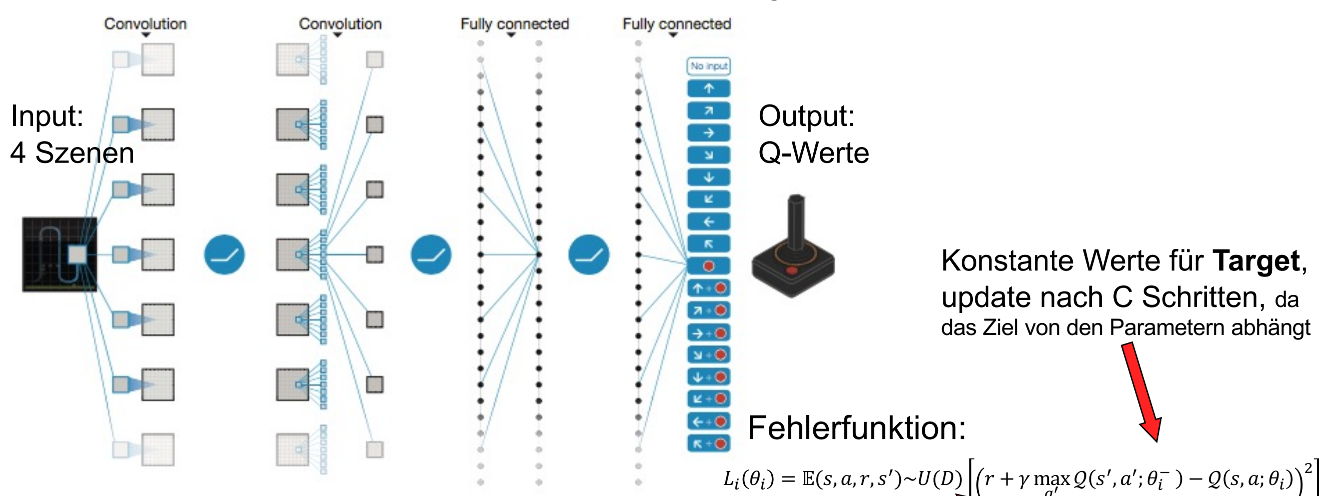
Eligibility Traces

- Eligibility Traces kombinieren die Vorteile von MC and TD Methoden
  - Vorteil MC: robuster wenn Markov Eigenschaft nicht erfüllt)
  - Vorteil TD: Bootstrapping
- Beschleunigt das Lernen
- Erzeugt etwas Kosten durch das Update von den Spuren (Traces)
- Kann auch in den Actor-Critic Ansatz integriert werden

Deep Q-Network

Verwendung eines Neuronalen Netzes zur Schätzung der Q-Werte. $Q(s,a) \rightarrow Q(s,a;\theta_i)$

θ_i: Parameter (Gewichte) des Netzes bzgl. Iteration i



Minibatches: zufällige gewählte Erfahrungen (experience replay)

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540):529–533.

Deep Q-Network

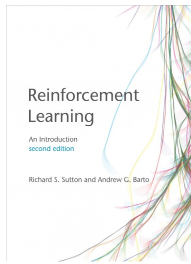
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Experience Replay

Fixed Q-Target

Zusammenfassung

- RL ist die wichtigste Methode für das Lernen in autonomen Agenten.
- Agent exploriert selbständig und kann Lösungen finden, die den Menschen bisher unbekannt sind.
- RL kann mit Deep Learning Methoden kombiniert werden: rohe Sensordaten (Bilder) als Eingabe und Aktionen als Ausgabe.
- In der Robotik dauert das Lernen zu lange. Simulationen oder kommunizierende Roboter werden benötigt.
- Kognitive Prozesse, die die Robustheit gegenüber Änderungen in der Umwelt oder Änderungen in den Zielen verbessern, sind noch nicht ausreichend Implementiert.



Reinforcement Learning: An Introduction, second edition.
Richard S. Sutton and Andrew G. Barto
The MIT Press Cambridge, Massachusetts London, England, 2018, 2020.
http://incompleteideas.net/book/the-book-2nd.html