Universität Potsdam
Institut für Informatik
Lehrstuhl Maschinelles Lernen

Reinforcement Learning

Uwe Dick

Inhalt

- Problemstellungen
- Beispiele
- Markov Decision Processes
- Planen – vollständige MDPs
- Lernen – unbekannte MDPs
- Monte-Carlo
- Temporal Difference

Problemstellungen des maschinellen Lernens

- Überwachtes Lernen:
  Lernen einer Entscheidungsfunktion aus Beispielen der richtigen Entscheidung.

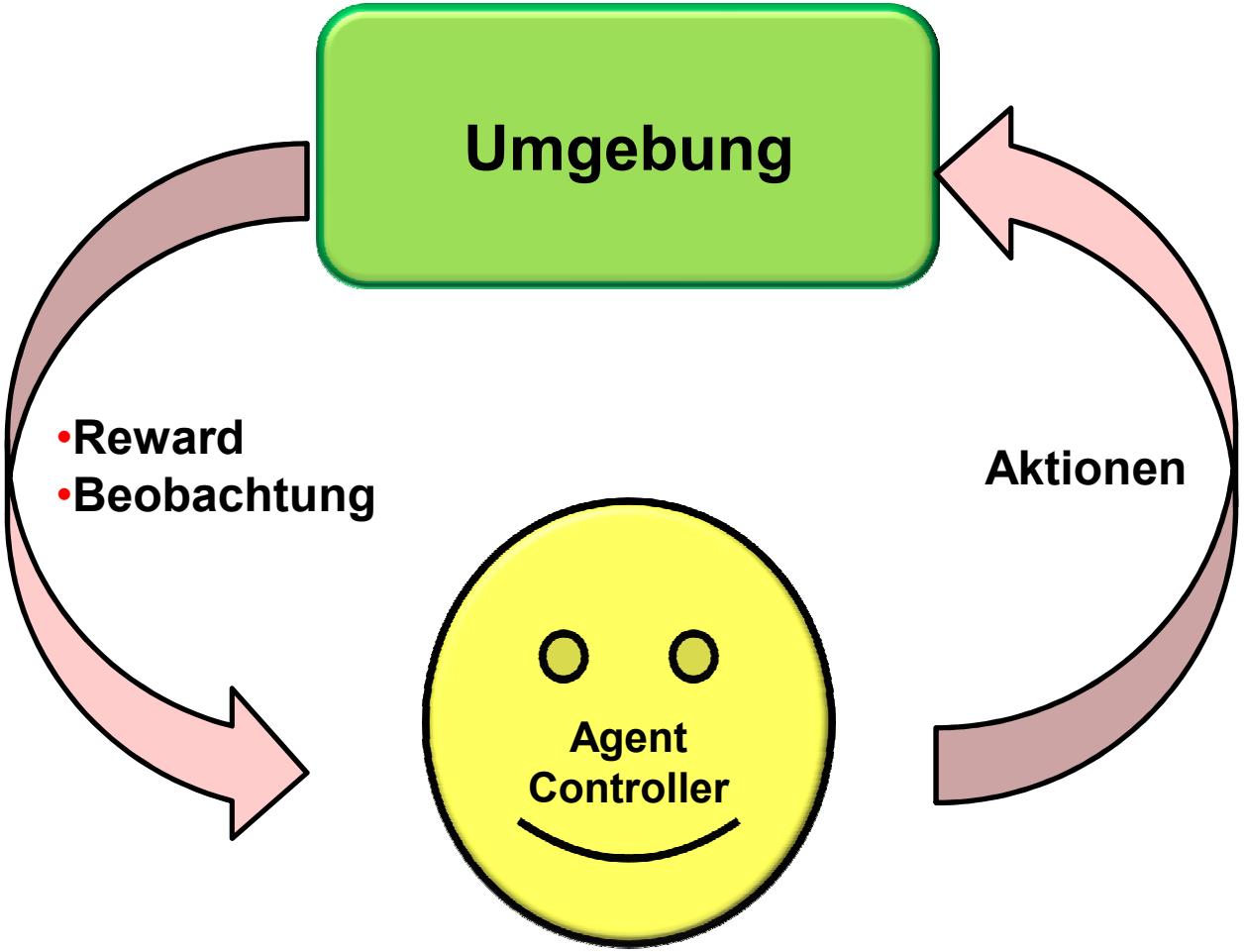
- Unüberwachtes Lernen:
  Lernen von zB. Partitionierungen von Daten (Clustern) ohne Beispiele für die richtige Partitionierung.

- Reinforcement Learning:
  Lernen von sequenziellen Entscheidungen. Die Güte einer Entscheidung wird durch die Güte der Entscheidungssequenz bestimmt.
  $\rightarrow$ Temporal Credit Assignment Problem.

Beispiele

- Schach: Welcher Zug hatte wieviel Einfluss auf das Spielergebnis?
- Robofußball: Ziel ist es, ein Tor zu schießen. Aber welche Bewegungen haben das ermöglicht?
- Helikopterflug: Welche Bewegungen müssen ausgeführt werden, um bei unvorhersehbaren äußeren Bedingungen nicht abzustürzen.

Lernen aus Interaktionen



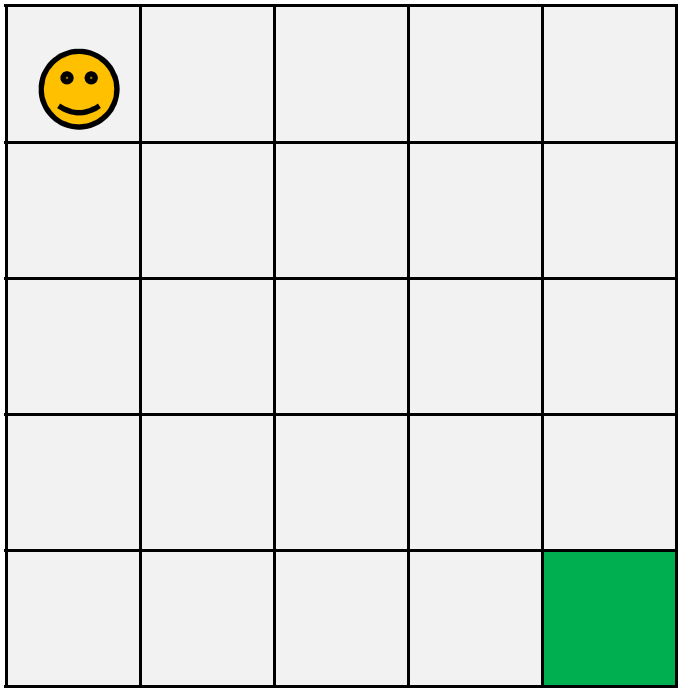
Wann Reinforcement Learning?

- Verzögerte Bewertung von Aktionen.
  (Temporal credit assignment problem)
- Kontrollprobleme
- Agenten – Das volle KI-Problem

Was ist Reinforcement Learning?

- RL-Methoden sind „Sampling based methods to solve optimal control problems “ (Richard Sutton)
- Suche nach einer optimalen Policy: Funktion von Zustand zu Aktion.
- Optimalität des Lernens: Policy mit höchstem erwarteten Reward.
- Aber auch: schnelles Lernen ohne zuviele Fehler zu machen.

Beispiel: Gridworld



Markov Decision Processes

- Markov-Entscheidungsprozess (S,A,R,P)
- S : endliche Zustandsmenge
- A : endliche Aktionsmenge
- P : Übergangswahrscheinlichkeiten

$$P(s'|s,a)\ \ s,s' \in S, a \in A$$

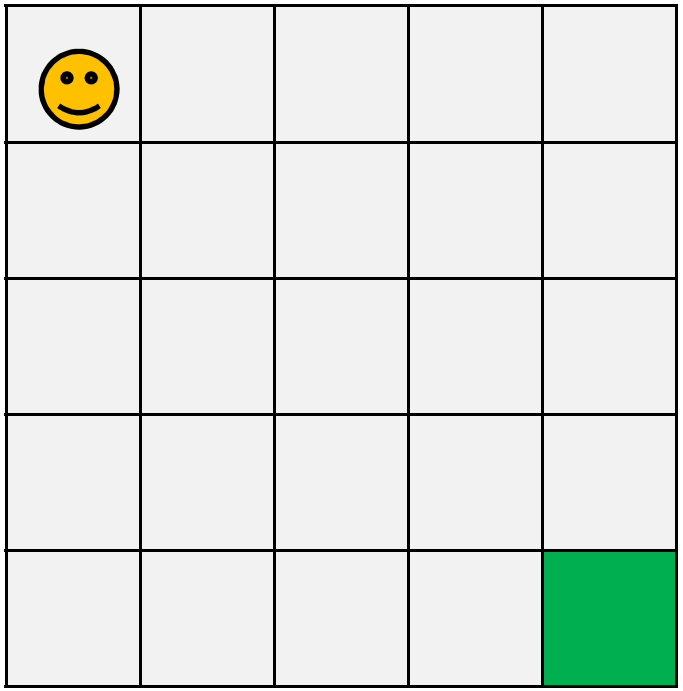
- R : Erwarteter Reward. Beschreibt den sofort erzielten Gewinn.

$$R : (S \times A) \rightarrow \mathbb{R}$$

- Discount factor $0 \leq \gamma < 1$.

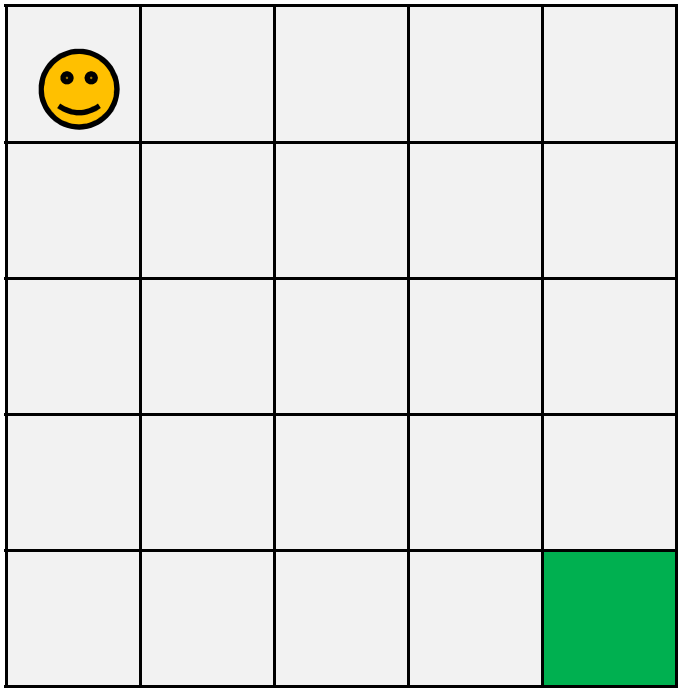
Beispiel: Gridworld

- Zustandsraum S
  - Startzustand $s_s \in S$
  - Zielzustand $s_z \in S$



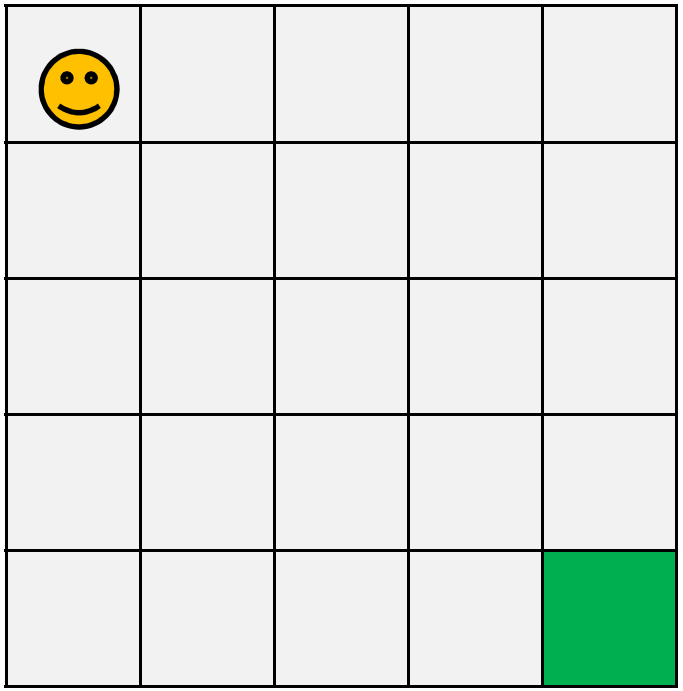
Beispiel: Gridworld

- Zustandsraum *S*
- Aktionsmenge *A*
  - *A*=(links, rechts, oben, unten)



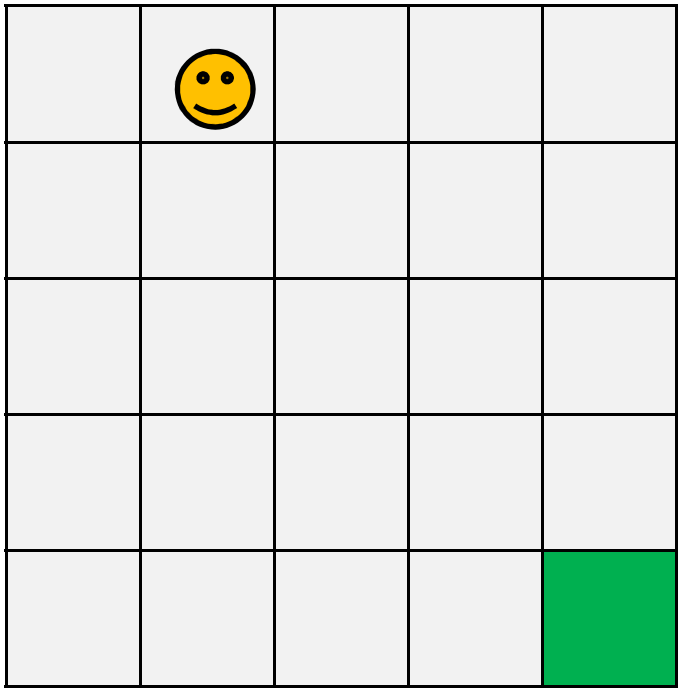
Beispiel: Gridworld

- Zustandsraum *S*
- Aktionsmenge *A*
- Übergangswahrscheinlichkeit *P*
  - *P*((1,2)|(1,1), rechts) = 1



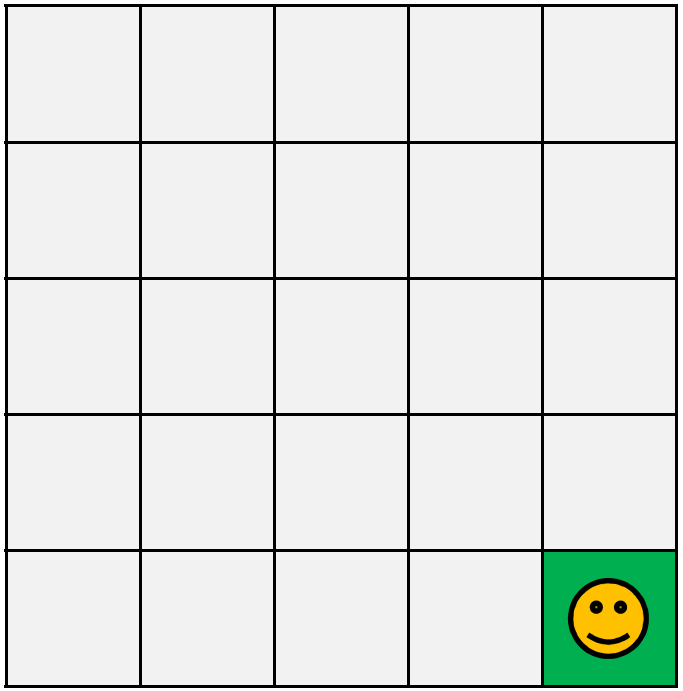
Beispiel: Gridworld

- Zustandsraum *S*
- Aktionsmenge *A*
- Übergangswahrscheinlichkeit *P*
- Erwarter Reward *R*
  - *R*((1,1),rechts) = 0



Beispiel: Gridworld

- Zustandsraum *S*
- Aktionsmenge *A*
- Übergangswahrscheinlichkeit *P*
- Erwarter Reward *R*
  - *R*((4,5),unten) = 1



Markov Decision Processes

- Markov-Entscheidungsprozess (S,A,R,P)
- S : endliche Zustandsmenge
- A : endliche Aktionsmenge
- P : Übergangswahrscheinlichkeiten

$$P(s'|s,a)\ \ s,s' \in S, a \in A$$

- R : Erwarteter Reward. Beschreibt den sofort erzielten Gewinn.

$$R : (S \times A) \rightarrow \mathbb{R}$$

- Discount factor $0 \leq \gamma < 1$.

MDP

- Eine deterministische stationäre Policy bildet Zustände auf Aktionen ab.

$$\pi : S \rightarrow A$$

- Stochastische Policy: Funktion von Zuständen auf eine Verteilung von Aktionen.

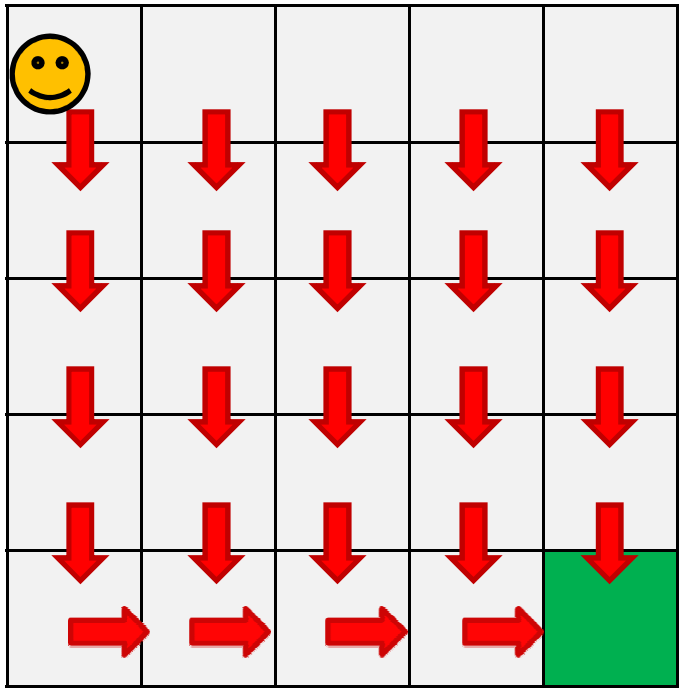
- Ziel: Finde Policy π, die den erwarteten kumulativen (discounted) Gewinn maximieren.

$$E_{\pi,P}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,\pi(s_t))\right]$$

Beispiel: Gridworld

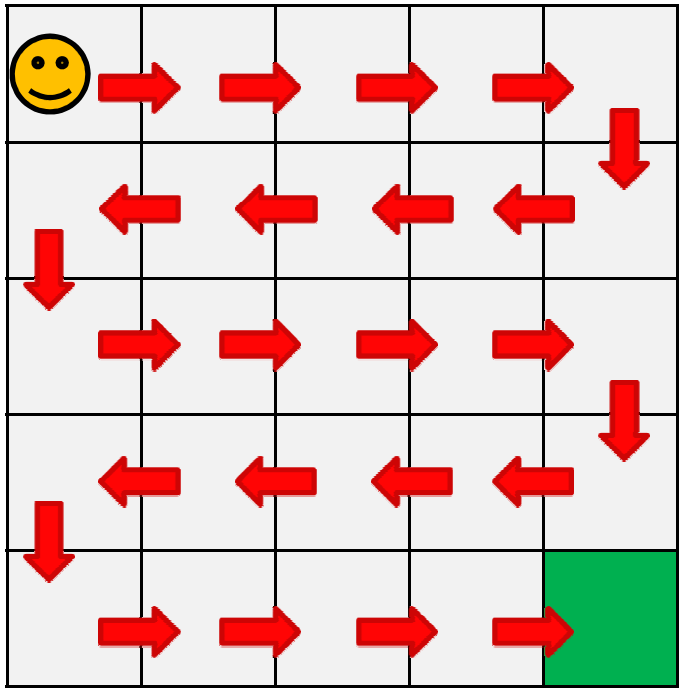
- Zustandsraum *S*
- Aktionsmenge *A*
- Übergangswahrscheinlichkeit *P*
- Erwarter Reward *R*
- Discountfaktor $\gamma = 0,9$
- Policy π
  - Gute Policy π_2
- Erwarteter discounted Reward

$$E_{\pi,P}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,\pi(s_t)) \mid s_0 = s_s\right] = 0,9^7$$



Beispiel: Gridworld

- Zustandsraum *S*
- Aktionsmenge *A*
- Übergangswahrscheinlichkeit *P*
- Erwarter Reward *R*
- Discountfaktor $\gamma = 0,9$
- Policy π
  - Schlechte Policy π_1
- Erwarteter discounted Reward



$$E_{\pi,P}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,\pi(s_t)) \mid s_0 = s_s\right] = 0,9^{23}$$

Markov-Eigenschaft

- Markov-Eigenschaft:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s_{t+1}|s_t, a_t)$$
$$R(s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = R(s_t, a_t)$$

- Aus Sequenz von Beobachtungen und Aktionen wird Zustand.

- Markov-Eigenschaft in Realität selten genau erfüllt.

Value Functions – Bewertungsfunktionen

- Value function $V^{\pi}(s)$ für einen Zustand s und Policy π beschreibt den erwarteten kumulativen Gewinn der von diesem Zustand aus erreicht wird.

$$V^{\pi}(s_t) = E_{\pi,P}\left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, \pi(s_{t+k}))\right]$$

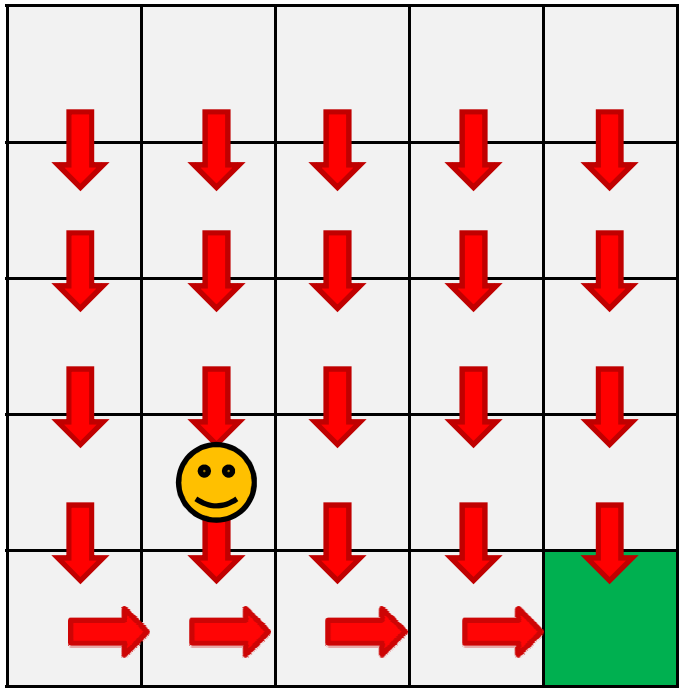
- Es existiert immer eine optimale deterministische stationäre Policy π^*, die die Value Function maximiert.

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$V^{\pi^*}(s) = V^*(s)$$

Beispiel: Gridworld

- Zustandsraum *S*
- Aktionsmenge *A*
- Übergangswahrscheinlichkeit *P*
- Erwarter Reward *R*
- Discountfaktor $\gamma = 0,9$
- Policy π
  - Gute Policy π_2
- Erwarteter discounted Reward



$$V^{\pi_1}(s_t) = E_{\pi,P}\left[\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, \pi(s_{t+k}))\right] = 0,9^3$$

Value Functions

- Bewertungsfunktion für Zustand-Aktion-Paar:

$$Q^{\pi}(s_t, a_t) = R(s_t, a_t) + E_{\pi,P}\left[\sum_{k=1}^{\infty} \gamma^k R(s_{t+k}, \pi(s_{t+k}))\right]$$

- Optimale Bewertungsfunktion

$$\begin{aligned} Q^*(s_t, a) &= R(s_t, a) + \gamma E_P[V^*(s_{t+1})] \\ \text{und } V^*(s_t) &= \max_a Q^*(s_t, a) \\ Q^*(s, a) &= \max_{\pi} Q^{\pi}(s, a) \\ \pi^*(s) &= \arg\max_a Q^*(s, a) \end{aligned}$$

- Annahme: tabellarische Speicherung der Bewertungen aller Zustände

Kontinuierliche Zustandsräume

- In der realen Welt ist der Zustandsraum (meist) kontinuierlich oder aber sehr groß.
- Dasselbe kann für den Aktionsraum gelten.
- Repräsentation von Value Function und/oder Policy durch Funktionsapproximationsmethoden.
  - Z.B. Darstellen der Value Function als parametrisierte Funktion mit Parametervektor θ und Features (Basisfunktionen) $\phi_i : S \times A \to \mathbb{R}$

$$\hat{Q}(s,a;\theta) = \sum_{i=1}^{N} \phi_i(s,a) \cdot \theta_i$$

Kontinuierliche Zustandsräume

- Oder Darstellung der Policy als parametrisierte Funktion von zustandsabhängigen Features

$$\pi(s;\theta) = \sum_{i=1}^{N} \phi_i(s) \cdot \theta_i$$

- Solche Probleme werden nächste Woche behandelt!

- Heute: „Idealisierte“ Probleme mit kleinen diskreten Zustands- und Aktionsräumen.

Bellman-Gleichungen

- Für Bewertungsfunktionen gelten die Bellman-Gleichungen (durch Markov-Eigenschaft)

$$
\begin{aligned}
V^{\pi}(s_t) &= E_{\pi,P}\Big[\sum_{k=0}^{\infty}\gamma^k R(s_{t+k},\pi(s_{t+k}))\Big] \\
&= E_{\pi,P}\Big[R(s_t,\pi(s_t)) + \gamma\sum_{k=0}^{\infty}\gamma^k R(s_{t+k+1},\pi(s_{t+k+1}))\Big] \\
&= E_{\pi,P}\Big[R(s_t,\pi(s_t)) + \gamma V^{\pi}(s_{t+1}))\Big] \\
&= R(s_t,\pi(s_t)) + \gamma\sum_{s_{t+1}\in S} P(s_{t+1}|s_t,\pi(s_t))V^{\pi}(s_{t+1})
\end{aligned}
$$

Bellman-Gleichungen

- Zustand-Aktion-Bewertungsfunktion:

$$Q^{\pi}(s_t, a_t) = R(s_t, a_t) + \gamma E_{\pi,P}\left[Q^{\pi}(s_{t+1}, \pi(s_{t+1}))\right]$$

- Die Bellman-Gleichungen bilden ein lineares Gleichungssystem

$$\begin{aligned} V^{\pi} &= R + \gamma P^{\pi} V^{\pi} \\ \rightarrow V^{\pi} &= (I - \gamma P^{\pi})^{-1} R \end{aligned}$$

Bellman-Operatoren

- In (linearer) Operatorschreibweise:

$$V^\pi = T^\pi V^\pi$$

- Mit linearem Operator T^π:

$$(T^\pi V)(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V(s')$$

➔ V^π ist ein Fixpunkt des Bellman-Operators T^π .

Bellman-Optimalitätsgleichungen

- Bellman-Gleichungen für das Kontrollproblem.

- Rekursive Beziehungen der optimalen Value Functions.

$$V^*(s_t) = \max_a E_{\pi,T}\left[R(s_t,a) + \gamma V^*(s_{t+1})\right]$$

$$Q^*(s_t,a_t) = R(s_t,a_t) + \gamma E_{\pi,T}\left[\max_a Q^*(s_{t+1},a)\right]$$

Modellwissen

- Es ergeben sich unterschiedliche Problemstellungen je nach Wissen über den MDP.

- MDP vollständig bekannt.
  $\rightarrow$ Planen.

- MDP nicht oder teilweise bekannt. Es kann Erfahrung gesammelt werden durch Interaktion mit der Umgebung.
  $\rightarrow$ Reinforcement Learning.

Arten von Reinforcement Learning

- Reinforcement Learning-Methoden können eingeteilt werden bezüglich der Verwendung der Interaktionsbeispiele.

- Indirekte Methoden:
  - Model learning

- Direkte Methoden:
  - Direkte Policy-Suche
  - Value-Function-Schätzung
    - Policy Iteration
    - Value Iteration

MDP vollständig bekannt – Dynamische Programmierung

- 2 Schritte zum Berechnen der optimalen Policy:
  - Policy Evaluation: V^{π} berechnen für festes π_k
  - Policy Improvement: Neues π_{k+1} bestimmen
- Policy Iteration.

- Bellman-Gleichungen bilden ein lineares Gleichungssystem.
- Zustandsmengen sind allerdings in der Realität in der Regel zu groß um Standardlösungsverfahren für LGS zu verwenden.

Policy Iteration

- Iteratives Verfahren: Q^{π} iterativ durch Folge von Approximationen $Q_k{}^{\pi}$ berechnen

$$Q_{k+1}^{\pi}(s,a) = E_{\pi,P}\left[R(s,a) + \gamma Q_k^{\pi}(s',\pi(s'))\right]$$
$$= R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) Q_k^{\pi}(s',\pi(s'))$$

- Greedy Policy Improvement:

$$\pi_{k+1}(s) \quad = \quad \arg\max_a Q^{\pi_k}(s,a)$$

Policy Iteration

- Iteratives Verfahren: V^{π} iterativ durch Folge von Approximationen V_k berechnen

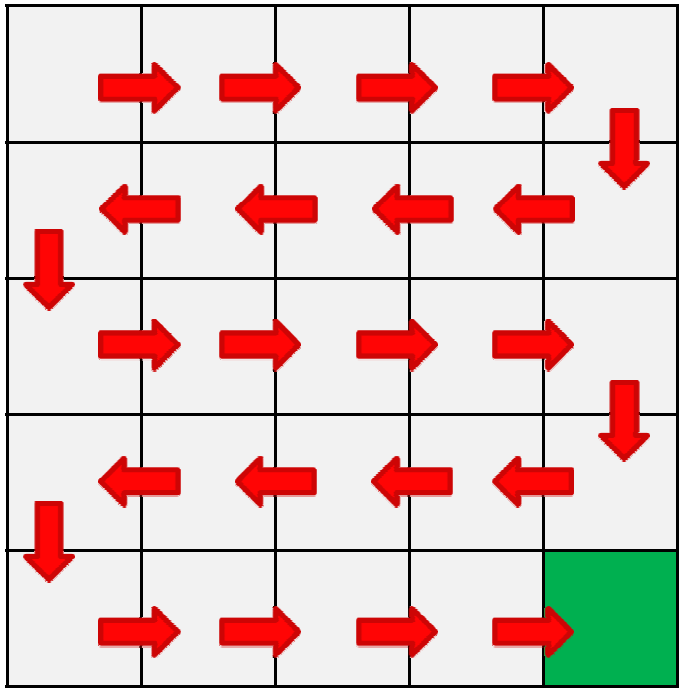
$$\begin{aligned} V_{k+1}(s_t) &= E_{\pi,T}\Big[R(s_t,\pi(s_t)) + \gamma V_k(s_{t+1}))\Big] \\ &= R(s_t,\pi(a_t)) + \gamma \sum_{s_{t+1}\in S} T(s_{t+1}|s_t,\pi(s_t))V_k(s_{t+1}) \end{aligned}$$

- Greedy Policy Improvement:

$$\pi_{k+1}(s) = \arg\max_a Q^{\pi_k}(s,a)$$

Beispiel: Gridworld

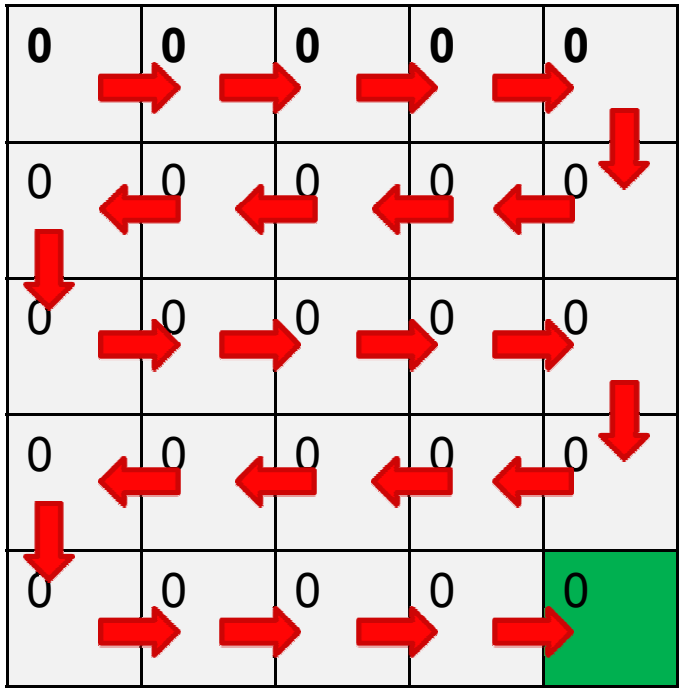
- Discountfaktor $\gamma = 0,9$
- Start Policy π_1 
- Policy Iteration:
  - Berechne V^{π_1} durch Folge von Approximationen V^k



Beispiel: Gridworld

- Discountfaktor $\gamma = 0,9$
- Start Policy π_1 
- Policy Iteration:
  - Berechne V^{π_1} durch Folge von Approximationen V^k

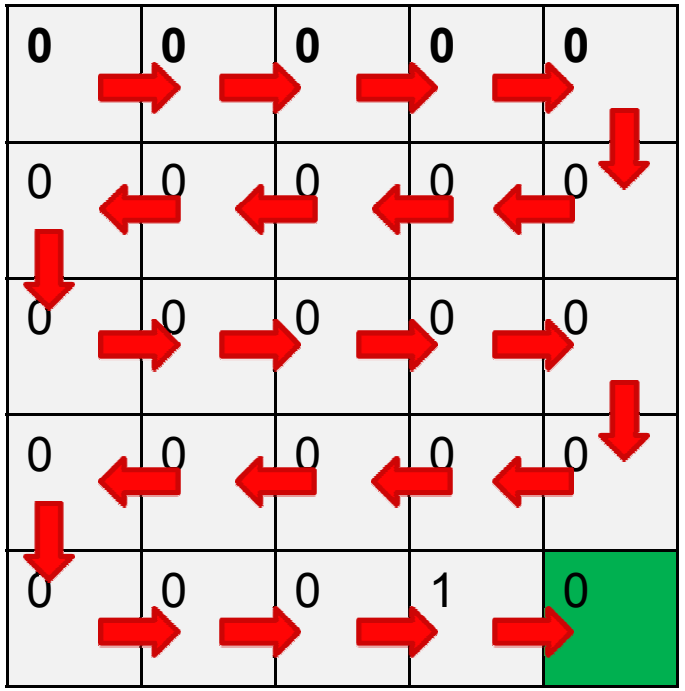
V^0



Beispiel: Gridworld

- Discountfaktor $\gamma = 0,9$
- Start Policy π_1 
- Policy Iteration:
  - Berechne V^{π_1} durch Folge von Approximationen V^k

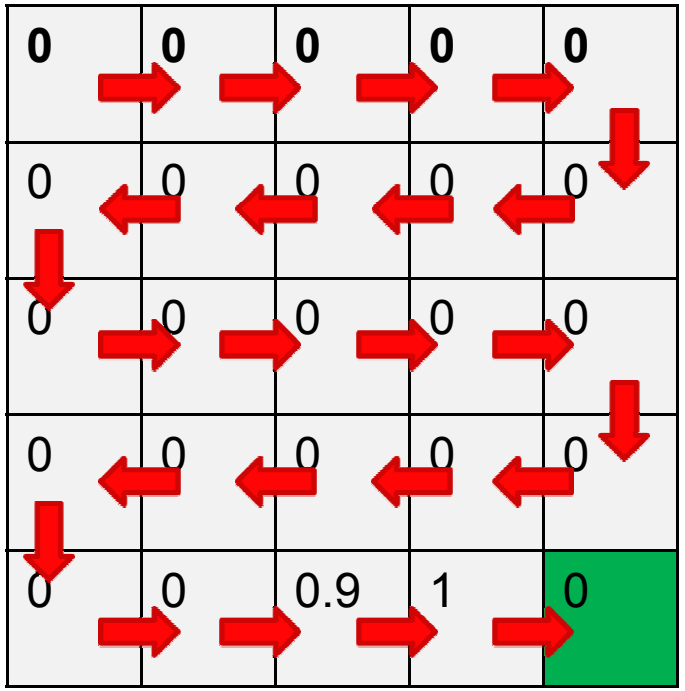
V^1



Beispiel: Gridworld

- Discountfaktor $\gamma = 0,9$
- Start Policy π_1 
- Policy Iteration:
  - Berechne V^{π_1} durch Folge von Approximationen V^k

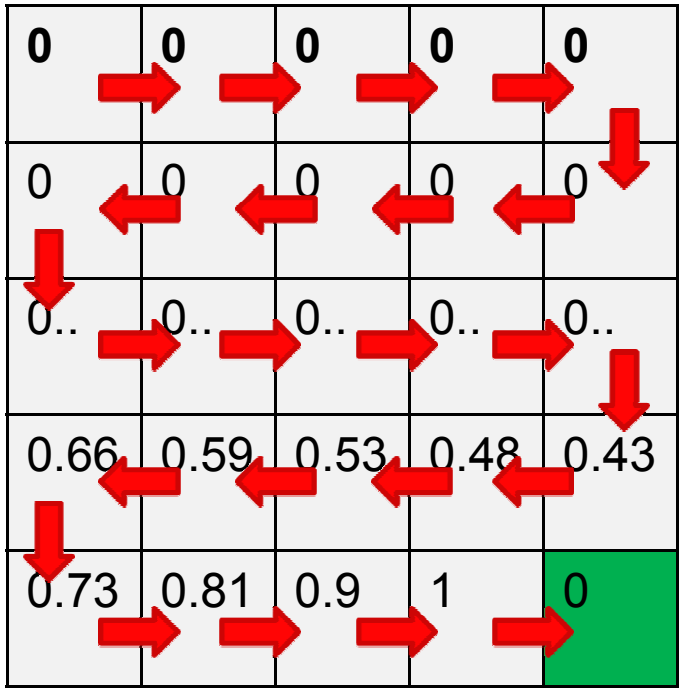
V^2



Beispiel: Gridworld

- Discountfaktor $\gamma = 0,9$
- Start Policy π_1 
- Policy Iteration:
  - Berechne V^{π_1} durch Folge von Approximationen V^k

V^n

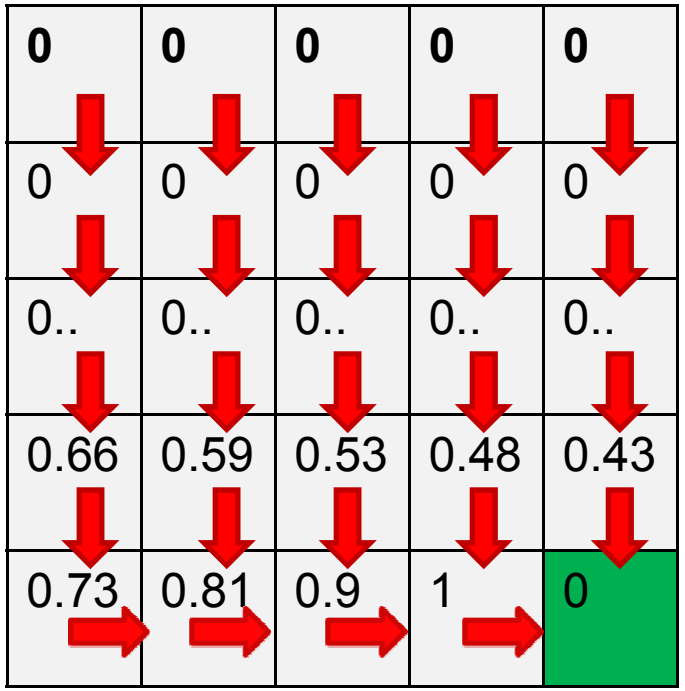


Beispiel: Gridworld

- Discountfaktor $\gamma = 0,9$
- Start Policy π_1 

- Policy Iteration:
  - Berechne V^{π_1} durch Folge von Approximationen V^k
  - Policy Improvement: Berechne greedy Policy π_2 $(Q^{\pi_k}(s,a) = R(s,a) + \gamma E[V^{\pi_k}(s')])$

V^{π_1}



Policy Evaluation

- Im Limit $k \to \infty$ konvergiert V_k zu V^π. Konvergenzrate O(γ^k): $||V_k - V^\pi|| = O(\gamma^k)$

- Beweis z.B. über Banach'schen Fixpunktsatz.
- Sei B=(B,||.||) ein Banachraum.
- Sei T ein Operator T:$B \to B$, so dass $||TU - TV|| \leq \gamma ||U - V||$ mit γ<1.
  T nennt man dann eine γ-Kontraktion.
- Dann hat T einen eindeutigen Fixpunkt V und es gilt, dass für alle $V_0 \in B$ die Folge V_{k+1}=$T\,V_k$, $k \to \infty$ gegen V konvergiert. Außerdem gilt $||V_k - V|| = O(\gamma^k)$

Policy Evaluation

- Es gilt, dass der Bellman-Operator T^π

$$(T^\pi V)(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V(s')$$

eine γ-Kontraktion ist. Daraus folgt, dass die iterative Anwendung des Operators

$$V_{k+1}(s) = (T^\pi V_k)(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V_k(s')$$

gegen V^π konvergiert.

Policy Evaluation: Kontraktion

- Was bedeutet Kontraktion?

$$\|T^\pi V_{k+1} - T^\pi V_k\| \quad \leq \quad \gamma \|V_{k+1} - V_k\|$$

- Anwenden der Iterationsvorschrift:

$$\|V_{k+2} - V_{k+1}\| \quad \leq \quad \gamma \|V_{k+1} - V_k\|$$

- Die Distanz zur echten Value Function verringert sich pro Iteration mit Faktor γ (im sup-Norm Sinne).

$$\|V^\pi - V_{k+1}\| = \|T^\pi V^\pi - T^\pi V_k\| \leq \gamma \|V^\pi - V_k\|$$

Policy Improvement

- Greedy Policy Improvement

$$\pi_{k+1}(s) \quad = \quad \arg\max_a Q^{\pi_k}(s,a)$$

- Policy Improvement Theorem:

  Seien π‘ und π deterministische Policies für die gilt, dass für alle $s \in S$: $Q^{\pi}(s,\pi\text{‘}(s)) \geq V^{\pi}(s)$.

  Dann $V^{\pi\text{‘}}(s) \geq V^{\pi}(s)$

Value Iteration

- Value Iteration für das Kontrollproblem:

$$V_{k+1}(s_t) = \max_a \left[R(s_t, a) + \gamma \sum_{s_{t+1}} T(s_{t+1}|s_t, a) V_k(s_{t+1}) \right]$$

$$Q_{k+1}(s, a) = R(s, a) + \gamma \sum_{s'} T(s'|s, a) \max_{a'} Q_k(s', a')$$

- Konvergiert gegen Q^* für $k \to \infty$
- Ähnlicher Beweis.

Value Iteration

- Algorithmus:
  - Initialisiere Q, z.B. Q = 0
  - for k=0,1,2,…:
    - foreach (s,a) in (S x A):

$$Q_{k+1}(s,a) \quad = \quad R(s,a) + \gamma \sum_{s'} T(s'|s,a) \max_{a'} Q_k(s',a')$$

  - until $Q_{k+1} = Q_k$

- Alternatives Konvergenzkriterium:
  - Max. Entfernung zum optimalen Q^* kleiner als ς
  - Konservative Wahl: $K = \log_\gamma \frac{\varsigma(1-\gamma)^2}{2\|R\|_\infty}$

MDP unvollständig — Reinforcement Learning

- Indirektes Reinforcement Learning: Modelbasiert
- Lerne Modell des MDP:
  - Rewardfunktion R
  - Transitionswahrscheinlichkeiten P
- Anschließend Planen.

Monte-Carlo Methoden

- Lernen von episodischen Interaktionen mit der Umgebung.
- Ziel: Lernen von $Q^{\pi}(s,a)$.

- Monte-Carlo Schätzung von $Q^{\pi}(s,a)$: Mittelwert bilden über gesamplete kumulative Rewards.

- Erwartungstreue Schätzung des echten erwarteten Rewards. Varianz fällt mit $1/n$.

Monte-Carlo Methoden

- Berechnungszeit der Schätzung ist unabhängig von der Größe des Zustandsraums.
- Problem: Falls π deterministisch werden viele state-action-Paare $Q(s,a)$ nie beobachtet.
- Probleme beim Policy Improvement-Schritt
- Lösung: stochastische Policies, z.B. ϵ-greedy Policies.

Greedy und ϵ-greedy Policies

- Greedy:

$$\pi = \arg\max_a Q_t(a)$$

- ϵ-greedy

$$\pi = \begin{cases} \arg\max_a Q_t(a) & \text{mit Wahrscheinlichkeit } 1-\epsilon \\ \text{zufällige Aktion} & \text{mit Wahrscheinlichkeit } \epsilon \end{cases}$$

- ϵ-greedy lässt zufällige Explorationsschritte zu.

Stochastische Policy: Softmax

- π stochastische Policy.
- Schätzungen sollen Einfluss auf Auswahlwahrscheinlichkeit haben.
  $\rightarrow$ Softmax

- Beispiel: Gibbs-Verteilung:

$$\pi(a) = \frac{e^{Q_t(a)/\tau_t}}{\sum_{i=1}^{|A|} e^{Q_t(a_i)/\tau_t}}$$

- τ_t ist Temperaturparameter.

Temporal Difference Learning

- Idee: Updates von Zuständen auf Grund von *Schätzungen* von anderen Zuständen.
- Natürliche Formulierung als Online-Methoden.
- Anwendbar auch für unvollständige Episoden.

- Nachteil gegenüber Monte-Carlo:
  - Stärkerer Schaden durch Verletzung der Markov-Eigenschaft.

Q-Learning

- Q-Learning Update-Schritt:

$$Q(s_t, a_t) \leftarrow (1-\alpha_t)Q(s_t, a_t) + \alpha_t \left(R(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') \right)$$

- Konvergiert gegen Q^* falls
  - Jeder Zustand unendlich oft besucht wird
  - Für den Schrittweitenparameter gilt:

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

  - Beweis folgt durch Theorie der stochastischen Approximation aus dem Beweis für DP-Fall.

Q-Learning

- Off-Policy-Methode. Dadurch wird das Exploration / Exploitation Problem gelöst.
- Lernen einer optimalen Policy π^* während nach einer anderen Policy π‘ entschieden wird.
- Die Policy π‘ kann z.B. eine stochastische Policy mit $\pi(s,a)>0$ für alle s und a sein, damit Q konvergiert.

SARSA

- SARSA: On-Policy Temporal Difference Methode.

$$Q(s_t, a_t) \leftarrow (1 - \alpha_t) Q(s_t, a_t) + \alpha_t \left(R(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) \right)$$

- Exploration / Exploitation Problem.

- SARSA vollzieht einen 1-Schritt Temporal-Difference Updateschritt.

N-step Returns

- Allgemeine Updateregel:

$$V(s_t) \leftarrow (1-\alpha_t)V(s_t) + \alpha_t \Delta V(s_t)$$

- Temporal Difference Methoden machen 1-Schritt Updates:

$$\Delta_1 V(s_t) = R_t + \gamma V(s_{t+1})$$

- Monte-Carlo-Methoden machen dagegen Updates, die auf der gesamten Episode basieren:

$$\Delta_\infty V(s_t) = R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + \ldots$$

- N-Schritt-Updates:

$$\Delta_n V(s_t) = R_0 + \gamma R_1 + \gamma^2 R_2 + \ldots + \gamma^{n-1} R_{n-1} + \gamma^n V(s_n)$$

TD(λ)

$$
\begin{array}{lccccccccc}
 & R_0 & & R_1 & & R_2 & & R_3 & \ldots & R_k & & R_{k+1} & \ldots \\
\Delta_1: & R_0 & + & \gamma V(s_1) & & & & & & & & & \\
\Delta_2: & R_0 & + & \gamma R_1 & + & \gamma^2 V(s_2) & & & & & & & \\
\Delta_3: & R_0 & + & \gamma R_1 & + & \gamma^2 R_2 & + & \gamma^3 V(s_3) & & & & & \\
 & & & & & \vdots & & & & & & & \\
\Delta_k: & R_0 & + & \gamma R_1 & + & \gamma^2 R_2 & + & \gamma^3 R_3 & + \ldots & + \gamma^k V(s_k) & & & \\
 & & & & & \vdots & & & & & & & \\
\Delta_\infty: & R_0 & + & \gamma R_1 & + & \gamma^2 R_2 & + & \gamma^3 R_3 & + \ldots & + \gamma^k R_k & + & \gamma^{k+1} R_{k+1} & + \ldots
\end{array}
$$

TD(λ)

$$
\begin{array}{lll}
 & & R_0 \quad R_1 \quad R_2 \quad R_3 \quad \dots \quad R_k \quad R_{k+1} \quad \dots \\
w_1 & \Delta_1: & R_0 + \gamma V(s_1) \\
w_2 & \Delta_2: & R_0 + \gamma R_1 + \gamma^2 V(s_2) \\
w_3 & \Delta_3: & R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 V(s_3) \\
 & & \vdots \\
w_k & \Delta_k: & R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 \quad + \dots + \gamma^k V(s_k) \\
 & & \vdots \\
w_\infty & \Delta_\infty: & R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 \quad + \dots + \gamma^k R_k + \gamma^{k+1} R_{k+1} + \dots
\end{array}
$$

- Idee: gewichtete Summe aller n-step Returns

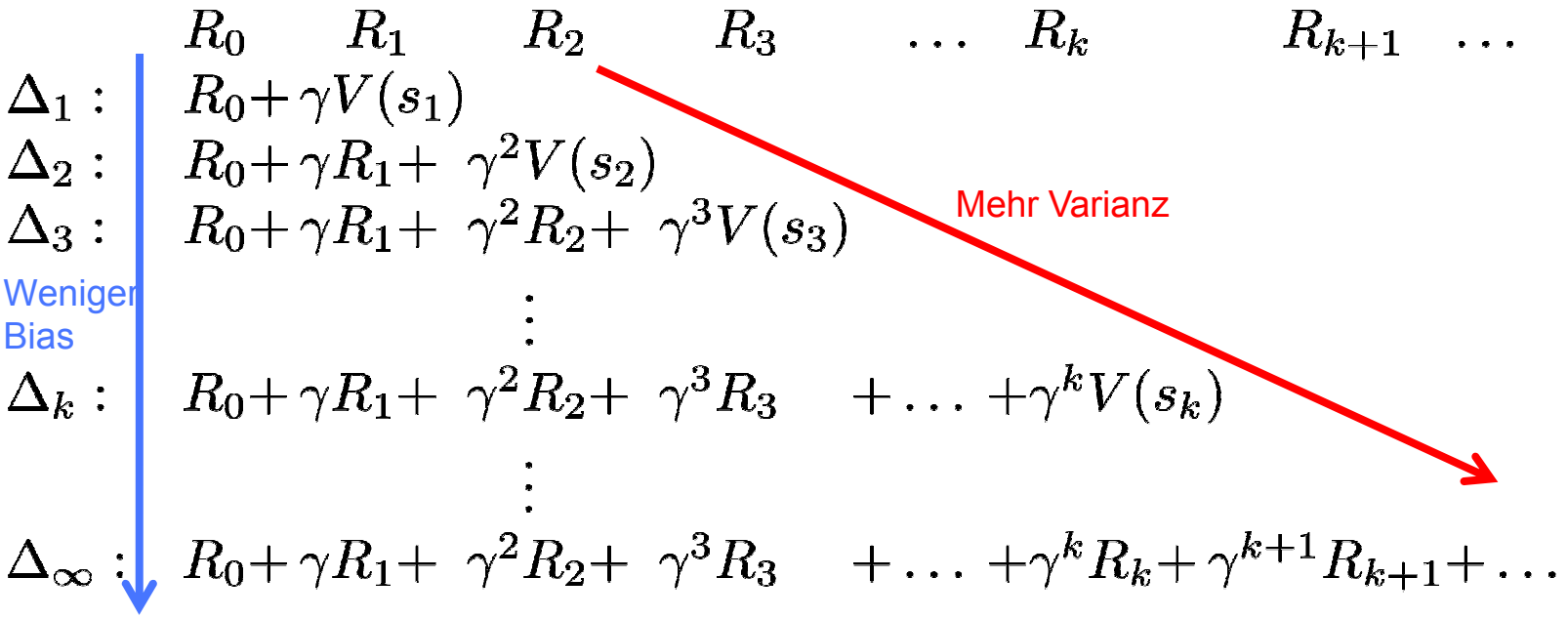
$$\Delta = \sum_{k=1}^{\infty} w_k \Delta_k V(s_0)$$

TD(λ)

$$
\begin{array}{lll}
 & & R_0 \quad R_1 \quad R_2 \quad R_3 \quad \ldots \quad R_k \\
(1-\lambda) & \Delta_1: & R_0 + \gamma V(s_1) \\
(1-\lambda)\lambda & \Delta_2: & R_0 + \gamma R_1 + \gamma^2 V(s_2) \\
(1-\lambda)\lambda^2 & \Delta_3: & R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 V(s_3) \\
 & & \vdots \\
(1-\lambda)\lambda^{k-1} & \Delta_k: & R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 \quad + \ldots + \gamma^k V(s_k) \\
 & & \vdots
\end{array}
$$

- TD(λ) Update: $\Delta(\lambda) = \sum_{k=1}^{\infty} (1-\lambda)\lambda^{k-1} \Delta_k V(s_0)$
- $0 \leq \lambda \leq 1$ interpoliert zwischen 1-step und MC.

Bias-Variance-Tradeoff



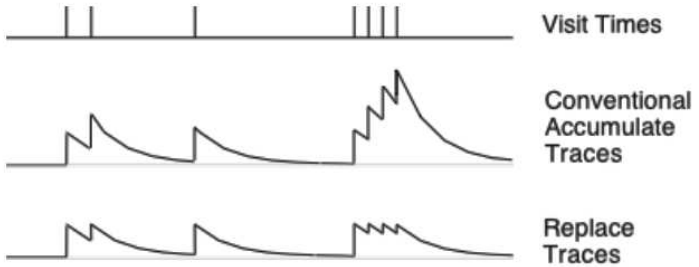
Eligibility Traces

- Algorithmische Sicht auf TD(λ)
- Einführung eines zusätzlichen Speichers $e(s)$ für jeden Zustand $s \in S$.
- Nach Beobachtung <s_t, a_t, R_t, s_{t+1}>, berechne

$$\begin{aligned} \delta_t &\leftarrow R_t + \gamma V(s_{t+1}) - V(s_t) \\ e(s_t) &\leftarrow e(s_t) + 1 \end{aligned}$$

- Update für alle Zustände

$$\begin{aligned} V(s) &\leftarrow V(s) + \alpha_t \delta_t e(s) \\ e(s) &\leftarrow \lambda \gamma e(s) \end{aligned}$$



Problemstellungen

- Lernen einer optimalen Policy.
  - Oder bestmögliche Approximation.
- Optimales Lernen: Möglichst wenige Fehler während des Lernen.
  - Exploration / Exploitation Problem.

Exploration / Exploitation Problem

- Tradeoff zwischen
  - Verfolgen der derzeit besten Policy, um den (greedy) Gewinn zu maximieren.
    (Exploitation)
  - und Erkunden derzeit suboptimaler Aktionen, über deren Wert noch Unsicherheit besteht, um eine potentiell bessere Policy zu finden.
    (Exploration)

Bandit Problem

- n-armed bandit Problem:
  - n Aktionen (Hebel) .
  - Jede Aktion anderen erwarteten Gewinn.
  - Erwartete Gewinne unbekannt.
  - Problem: Finde beste Aktion durch Ausprobieren, ohne dabei zuviel zu verlieren.

- Erwarteter Gewinn für Aktion a ist $Q^*(a)$.
- Schätzung des erwarteten Gewinns nach t Versuchen:

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{n_a(t)}}{n_a(t)}$$

Greedy und ϵ-greedy Policies

- Greedy:

$$\pi = \arg\max_a Q_t(a)$$

- ϵ-greedy

$$\pi = \begin{cases} \arg\max_a Q_t(a) & \text{mit Wahrscheinlichkeit } 1-\epsilon \\ \text{zufällige Aktion} & \text{mit Wahrscheinlichkeit } \epsilon \end{cases}$$

- ϵ-greedy lässt zufällige Explorationsschritte zu.

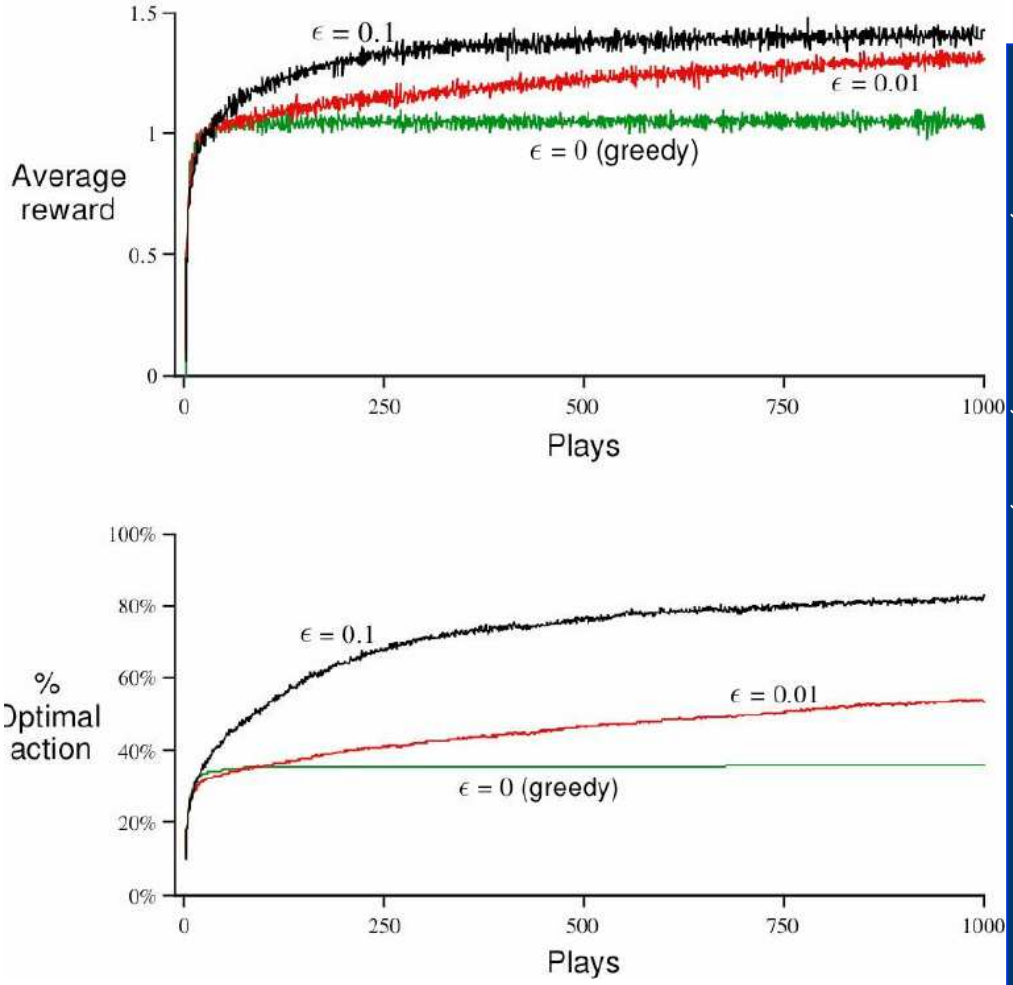
ϵ-greedy Policies

- 10-armed bandit
- 2000 Wiederholungen
- Zufälliges Ziehen von $Q^*(a)$ für alle a:

$$Q^*(a) \sim \mathcal{N}(0,1)$$

- Rewards werden gezogen aus

$$R_t(a) \sim \mathcal{N}(Q^*(a),1)$$



Optimismus bei Unsicherheit

- Ein Prinzip zur Auflösung des Exploration/Exploitation Dilemmas ist „Optimismus bei Unsicherheit“.
- Existieren auch Umgebungen, in denen Performance schlecht ist.
- Implementierbar z.B. über sehr hohe Initialisierungswerte von Q.

Optimismus bei Unsicherheit

- Upper Confidence Bound (UCB): [Auer et al. 02]
  - Angenommen, Rewards sind in [0,1].

$$\pi = \arg\max_a \left[Q_t(a) + \sqrt{\frac{c_e \log(t)}{2 n_a(t)}} \right] \quad , c_e \geq 2$$

- Für stationäre Umgebungen und iid Rewards sehr gute Ergebnisse.

Problemstellungen

- P,R bekannt. $P(s'|s,a)$ können abgefragt werden.
- P,R nicht explizit bekannt. Aber aus den Verteilungen $P(s'|s,a)$ kann gesamplet werden. Annahme: Generatives Modell von P und R.
- P,R nicht oder teilweise bekannt. Es kann Erfahrung gesammelt werden durch Interaktion mit der Umgebung.
  $\rightarrow$ Reinforcement Learning.
- Batch Reinforcement Learning: Es muss von einer fixen Menge von Beispielepisoden gelernt werden.

Große und unendliche Zustandsräume

- In realistischen Anwendungen sind Zustandsräume i.A. sehr groß bzw. kontinuierlich.
- Bisherige Annahme: tabellarische Repräsentation der Value Function.

- Mögliche Lösungen:
  - Planen:
    - Monte-Carlo Sampling
    - Diskretisierung und anschließend z.B. Value Iteration
  - Approximation der Value Function durch Funktionsapproximationsmethoden.
  - Direktes Lernen der Policy.

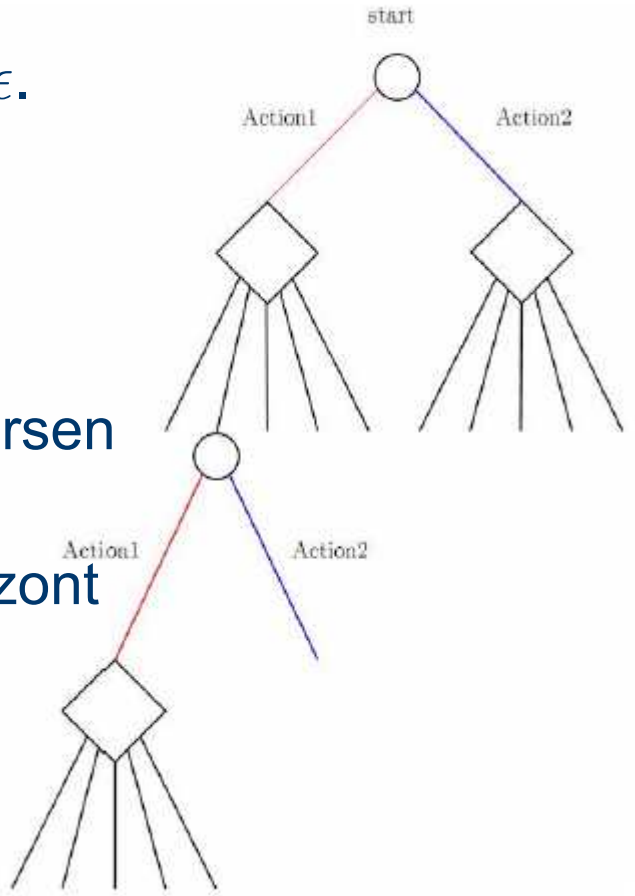
Approximation

- Arten von Approximation
  - Repräsentation, z.B.
    - Value Function $\hat{Q}(s,a;\theta) = \phi^T(s,a)\theta$
    - Policy $\pi(s,a;\theta) = h(\phi^T(s,a)\cdot\theta)$
  - Sampling
    - Online Lernen durch Interaktion
    - Samplen aus generativem Modell der Umgebung
  - Maximierung
    - Finden der besten Aktion in einem Zustand

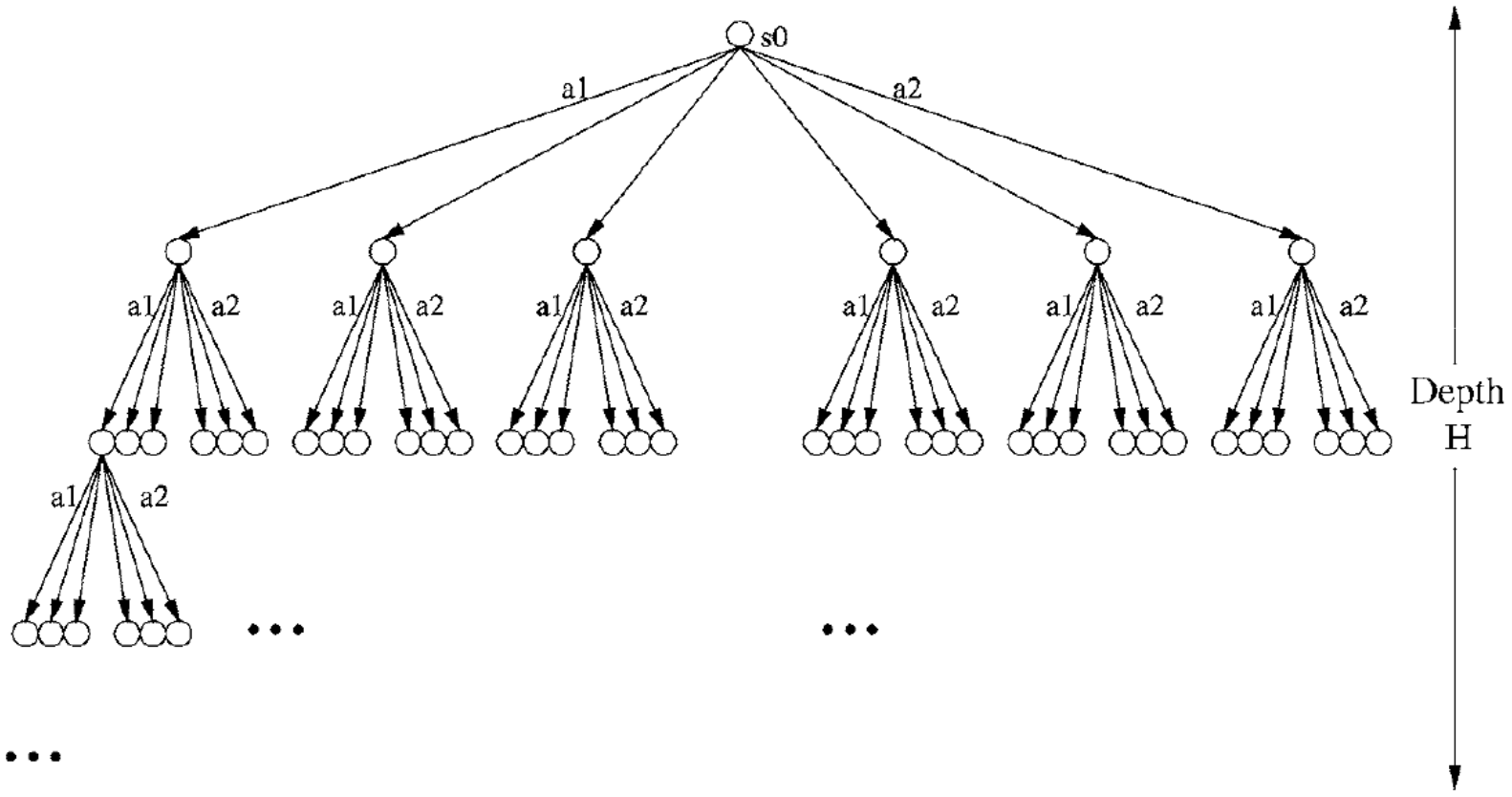
Monte-Carlo Sampling

- Angenommen, S sehr groß
- Ziel: Finde Q, so dass $||Q\text{-}Q^*||_\infty < \epsilon$.

- Sparse Lookahead Trees: [Kearns et al. 02]
  - Monte-Carlo: Samplen eines sparsen Aktions-Zustands-Baums.
  - Tiefe des Baums: Effektiver Horizont $H(\epsilon)$ = O(1/(1-γ) log(1/ϵ(1-γ)))
  - MC unabhängig von |S|
  - Aber exponentiell in $H(\epsilon)$: min. Größe des Baums $c|A|^{H(\epsilon)}$



Sparse Lookahead Trees



Upper Confidence Bounds for Trees

- Besser: Nur solche Teilbäume genauer untersuchen, die vielversprechend sind.
- Optimismus bei Unsicherheit!
  - Nutze das gleiche Prinzip wie bei Bandit Problem.
  - UCT: UCB for Trees.
    [Kocsis & Szepesvári 06]

$$\pi(s,a) = \arg\max_a \left[Q_t(s,a) + \sqrt{\frac{c_e \log(t)}{2n_{s,a}(t)}} \right] \quad , c_e \geq 2$$

UCT Performance: Go

- Sehr gute Resultate in Go.
- 9x9 & 19x19

- Computer Olympiade 2007 - 2009:
  - 2007 & 2008: 1.-3. Platz verwenden Varianten von UCT.
  - Im Allgemeinen: Monte-Carlo Search Trees (MCST).
  - 2009: Mindestens 2. und 3. verwenden Varianten von UCT.

Diskretisierung

- Kontinuierlicher Zustandsraum S.
- Random Discretization Method: [Rust 97]
  - Sampling von Zuständen S‘ nach uniformer Verteilung über den Zustandsraum.
  - Value Iteration.
- Kontinuierliche Value Iteration:

$$V_{t+1}(s) = \max_a \left[R(s,a) + \gamma \int_{s'} p(s'|s,a) V_t(s') ds' \right]$$

- Diskretisierung: Weighted Importance Sampling

$$\frac{\sum_{i=1}^{N} p(s_i|s,a) V(s_i)}{\sum_{j=0}^{N} p(s_j|s,a)} \rightarrow \int_{s'} p(s'|s,a) V(s') \text{ , für N} \rightarrow \infty$$

Diskretisierung

- Berechnen der Value Function $V(s)$ für Zustände, die nicht in der Samplingmenge S‘ sind:
- Bellman-Update –Schritt

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{i=1}^{N} \frac{p(s_i|s,a)}{\sum_{j=0}^{N} p(s_j|s,a)} V^*(s_i) \right]$$

- Garantierte Performance: [Rust97]
  Annahme: S=[0,1]^d

$$E[\|V_N(s) - V^*(s)\|_\infty^2] \leq \frac{Cd|A|^{5/4}}{(1-\gamma)^2 N^{1/4}}$$