



# Reinforcement Learning



Universität Paderborn

Fakultät EIM, PG-Paderkicker III

Wintersemester 2003/04

Andreas Koop

## Motivation

- Lebewesen lernen oft durch Interaktion mit ihrer Umwelt.
- Die Anpassung des Verhaltens wird durch das Feedback der Umwelt beeinflusst.
- Dabei gibt es besonders wünschenswerte oder besonders schlechte Folgezustände der Umwelt – „Belohnung“ und „Bestrafung“.
- Ein „Lehrer“, der die richtige Handlung selbst vorführt ist nicht notwendig – die Handlung wird vielmehr durch eine vorher definierte Erziehungsfunktion „bewertet“.
- Abbildung dieser Fähigkeit auf das Maschinelle Lernen -> Reinforcement Learning.

## Überblick

---

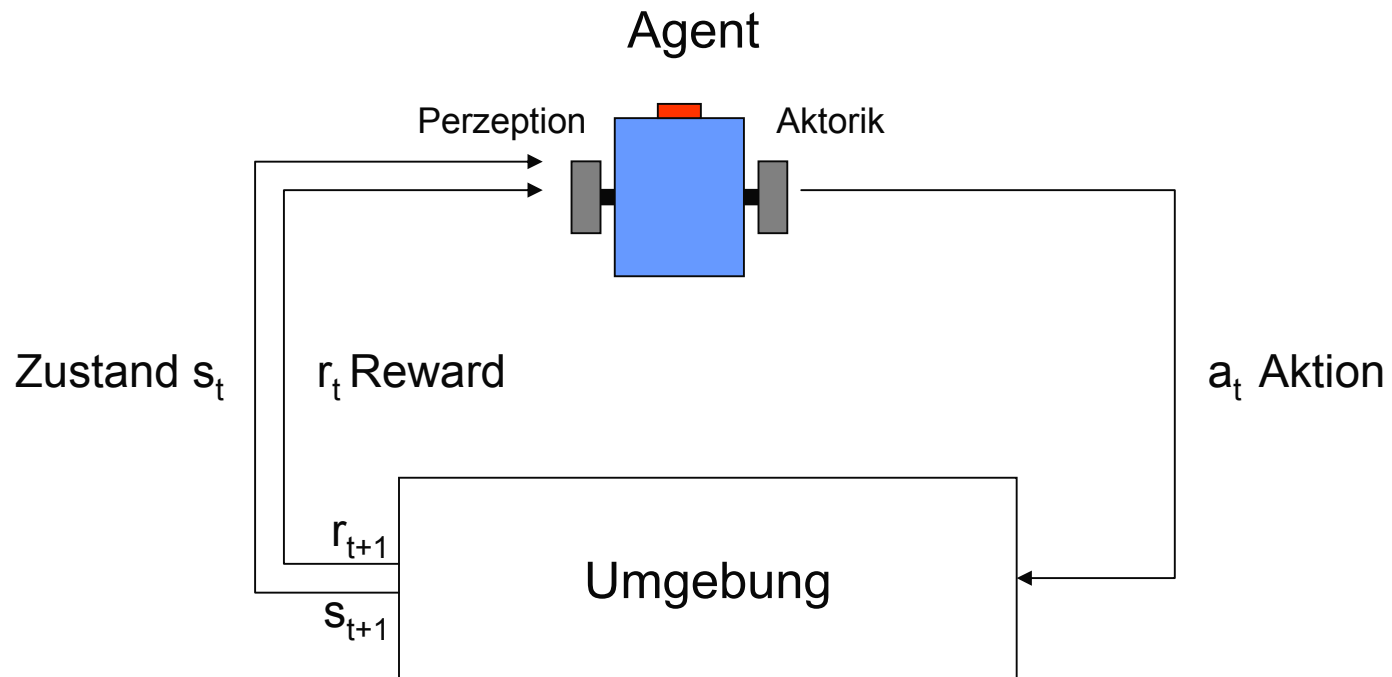
- Einführung
- Hauptbestandteile des Reinforcement Learnings
- Markov-Entscheidungs-Prozess (MDP)
- Lernalgorithmen
  
- Probleme
- Erfolgsgeschichten
- Ausblick

## Einführung - Was ist Reinforcement Learning?

- Lernen durch Feedbackfunktion (gut/schlecht)
  - Kein Trainer/ keine Überwachung notwendig
- Es wird keine Information über eine Lösungsstrategie vorausgesetzt
- Eigenständiges Erlernen einer Lösungsstrategie durch (geschickte) Versuch-und-Irrtum Methodik ('Trial and Error')
- RL ist eine Problemklasse, für die viele Lösungsverfahren existieren, RL ist kein spezieller Algorithmus

# Agent/Umgebung-Interaktion

- Agent und Umgebung interagieren zu diskreten Zeitpunkten:  
 $t = 0, 1, 2, \dots$



## Hauptbestandteile

- **Strategie (Policy  $\pi$ )**
  - Abbildung von Zuständen auf Aktionen
- **Vergütungsfunktion (Reward Function)**
  - Definiert übergeordnetes Ziel des RL-Problems
- **Wertefunktion (Value Function)**
  - Summe der Vergütungen vom aktuellen bis zum Endzustand
  - Langfristige Attraktion eines Zustandes
  - Ziel: Gesamtvergütung zu maximieren
  - Zustands-Wertefunktion:  $V^\pi(s)$
  - Aktions-Wertefunktion:  $Q^\pi(s, a)$
- **Optional: Umgebungsmodell**
  - Ermöglicht vorausschauendes Handeln

## Episodische/ kontinuierliche Tasks

- **Episodischer Task:** Interaktion teilt sich in Episoden auf, z.B. Durchgänge eines Spiels, Durchläufe durch ein Labyrinth, etc.
  - Definierter Endzustand
  - Summation zukünftiger Rewards

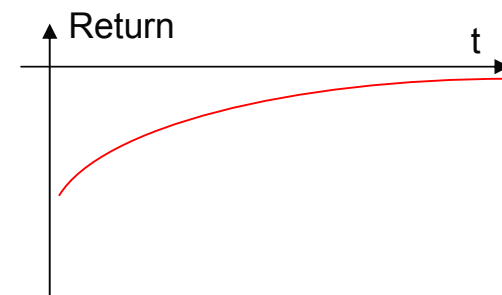
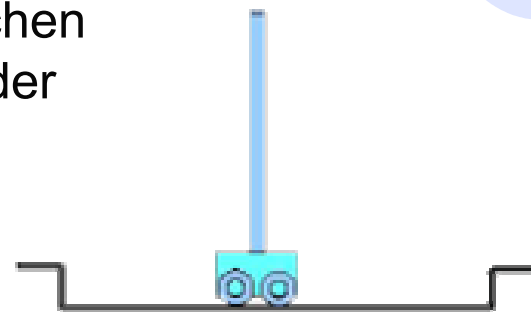
$$R_T = r_{t+1} + r_{t+2} + \dots + r_T$$

- **Kontinuierlicher Task:** Interaktion hat keine Episoden
  - Kein Endzustand vorhanden
  - Notwendigkeit einer Discount Rate  $\gamma$ , wobei  $0 \leq \gamma \leq 1$

$$R_T = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

## Beispiel - Stab balancieren

- **Misserfolg:** Der Stab fällt über einen kritischen Winkel oder der Wagen erreicht das Ende der Strecke
  - **Episodisch** (Ende bei Misserfolg)
    - Reward = +1 für jeden Schritt vor dem Misserfolg
    - Return = #Schritte bis zum Misserfolg
  - **Kontinuierlich** (discounted)
    - Reward = -1 bis Misserfolg, sonst 0
    - Return =  $-g^k$ , für  $k$  Schritte vor dem Misserfolg
- In beiden Fällen Maximierung des Return-Wertes

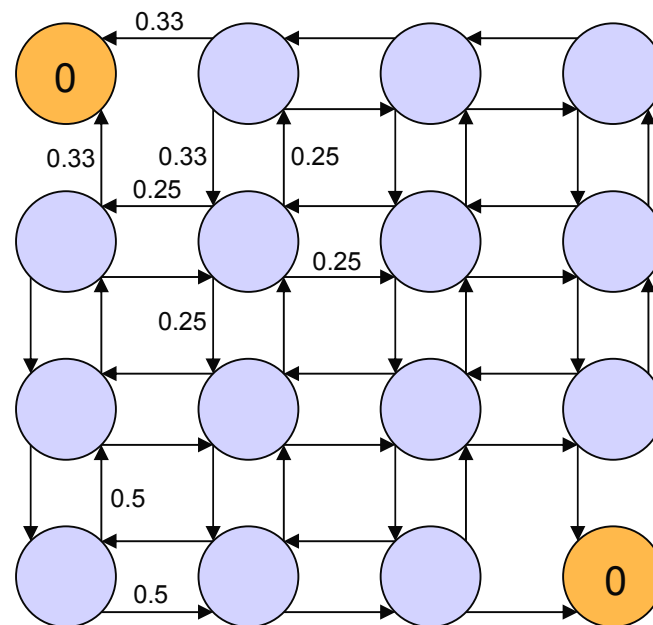


## Exploration vs. Exploitation

- Wann lernt der Agent?
    - **Exploration** (Lernen): Ausprobieren einer neuen Aktion, um mögliche, bessere Strategien zu finden
    - **Exploitation** (Anwenden): Wahl der bislang „besten“ Aktionen
  - „**greedy**“: Wähle *immer* die Aktion mit dem größten erwarteten Reward
  - „**wacky**“: Probiere *immer* neue Aktionen aus
  - **e-greedy**: Meistens „greedy“, wobei  $\epsilon$  für den Anteil steht, zu dem sich der Agent „wacky“ verhält
- Am Anfang „wacky“, mit zunehmender Erfahrung „greedy“

# Markov-Entscheidungs-Prozess (MDP)

- Mathematisches Modell für Reinforcement Learning Probleme
- Quintupel:  $MDP := \{S, A, T, \gamma, R\}$ 
  - $S$  := Menge von Zuständen
  - $A$  := Menge von Aktionen
  - $T := \{P_{sa}(s' | s \in S, a \in A)\}$
  - $\gamma$ : Discountfaktor
  - $R: S \times A \rightarrow R$  (Rewardfunktion)



## Beispiel - Grid World (1)

Zufällige Strategie

0	-14	-20	-22
-14	-18	-22	-20
-20	-22	-18	-14
-22	-20	-14	0

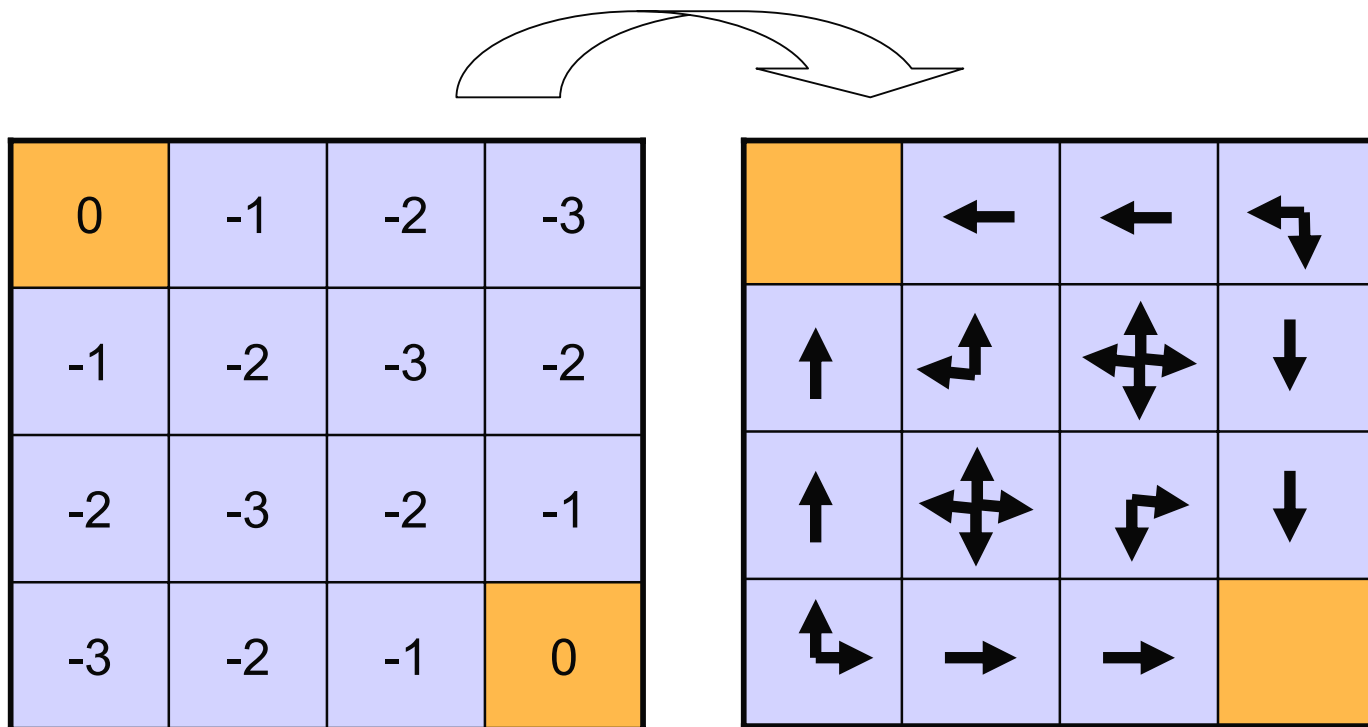
Optimale Strategie

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

- Reward: 0 am Ziel, überall sonst -1

## Beispiel - Grid World (2)

→ Bei bekannter optimaler Wertefunktion ist die optimale Strategie trivial zu berechnen



## Wie finde ich die optimale Wertefunktion?

- Fundamentale Frage des RL Problems

- **Lösungen**

- Dynamisches Programmieren

$$V^\pi(s) = E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) \right\}$$

- Monte Carlo Methoden
- Temporal Difference Methoden

$$Q^\pi(s, a) = E_\pi \left\{ r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) \right\}$$

## Dynamisches Programmieren

- Aktualisierung der Wertefunktion auf Grund der Werte der Folgezustände bis hin zum Endzustand (Bootstrapping)
- Optimale Lösung iterativ berechenbar
- Algorithmen: Policy Evaluation, Policy Improvement, Policy Iteration, Value Iteration
- **Nachteile:**
  - Benötigt komplettes Umgebungsmodell
  - Die komplette Value-Funktion wird zum evaluieren und verbessern eines Zustandes benötigt → Speicherbedarf
  - Die gesamte Wahrscheinlichkeitsverteilung für die Übergänge wird benötigt

## Monte Carlo Methoden

- Gute Approximation der optimalen Strategie
- Kein Bootstrapping, Aktualisierung erst nach Erreichen des Endzustandes
- Erfahrungswerte aus Beispielenisoden notwendig
- **Vorteile zu DP:**
  - Kein Umgebungsmodell notwendig
  - Direktes Lernen durch Interaktion mit der Umgebung
  - Fokussierung auf eine kleine Menge von Zuständen
- **Nachteile:**
  - Keine Garantie für ein globales Optimum der Strategie, da nicht alle Zustände besucht werden müssen
  - Nur für episodische Tasks geeignet

## Temporal Difference Methoden

---

- Kombination aus Monte Carlo Methoden und DP
- Verwendung von Bootstrapping
- Algorithmen: TD-Prediction, Sarsa, Q-Learning
- **Vorteile:**
  - Kein Umgebungsmodell notwendig
  - Für kontinuierliche Tasks geeignet

## TD-Prediction

Initialisiere  $V(s)$  beliebig

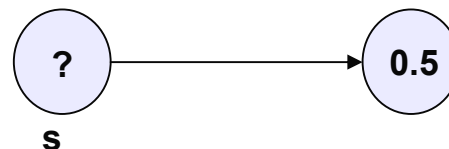
**Repeat** (für jede Episode)

Initialisiere  $s$

**Repeat** (für jeden Schritt der Episode)

1. Wähle Aktion  $a$  aus  $s$  unter Benutzung der zu evaluierenden Strategie  $\pi$
2. Führe Aktion  $a$  aus, beobachte Reward  $r$  und Folgezustand  $s'$
3.  $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
4.  $s \leftarrow s'$

**until**  $s = \text{Endzustand}$



$\alpha$ := Lernrate

$\gamma$ := Discountfaktor

## Sarsa (on-policy)

Initialisiere  $Q(s, a)$  zufällig

**Repeat** (für jede Episode):

Initialisiere Zustand  $s$

Wähle Aktion  $a$  aus  $s$  unter Benutzung der Policy aus  $Q$

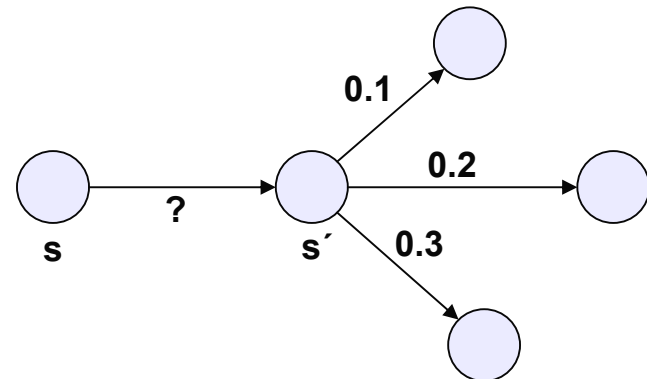
**Repeat** (für jeden Schritt der Episode):

1. Führe Aktion  $a$  aus, beobachte Reward  $r$  und Folgezustand  $s'$
2. Wähle Aktion  $a'$  aus  $s'$  unter Benutzung der Policy aus  $Q$
3.  $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$
4.  $s \leftarrow s'$
5.  $a \leftarrow a'$

**until**  $s = \text{Endzustand}$

$\alpha$  := Lernrate

$\gamma$  := Discountfaktor



## Q-Learning (off-policy)

Initialisiere  $Q(s, a)$  zufällig

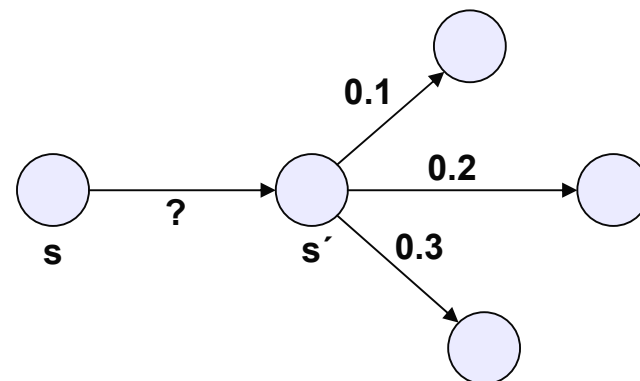
**Repeat** (für jede Episode):

Initialisiere Zustand  $s$

**Repeat** (für jeden Schritt der Episode):

1. Wähle Aktion  $a$  aus  $s$  unter Benutzung der Policy aus  $Q$
2. Führe Aktion  $a$  aus, beobachte Reward  $r$  und Folgezustand  $s'$
3.  $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_i Q(s', i) - Q(s, a)]$
4.  $s \leftarrow s'$

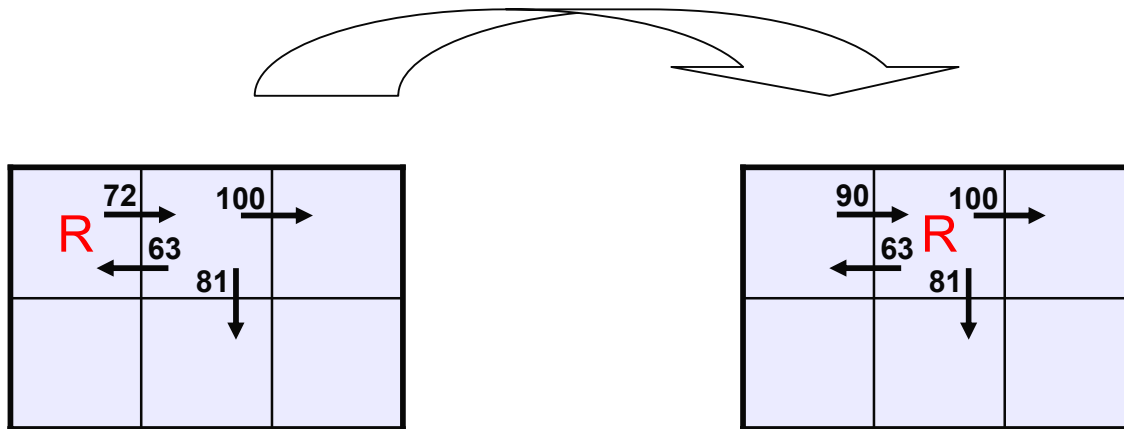
**until**  $s = \text{Endzustand}$



$\alpha$  := Lernrate

$\gamma$  := Discountfaktor

## Beispiel - Q-Learning



- Update der Q-Werte
- Q-Learning:  $Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$
- Im Beispiel:  $0 + 0.9 \cdot \max\{63, 81, 100\} = 0.9 \cdot 100 = 90$

## Probleme

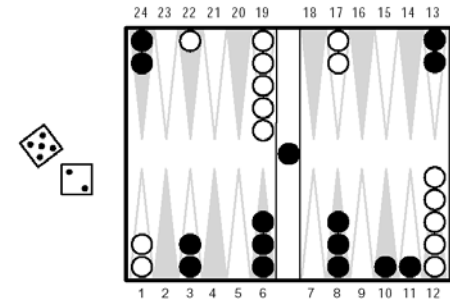
---

- Tatsächliche Probleme sind sehr kompliziert und lassen sich nicht mit einem einfach MDP beschreiben
  - Nichtdeterministische Übergänge
  - Kontinuierliche und/oder unendliche Zustandsmenge
  - Zeitabhängigkeit
- Aufwand steigt mit zunehmendem Zustands- und Aktionsraum
  - Lösung: Berücksichtigung von Vorwissen, etc.
- Multi-Agenten-Systeme
  - Lösung: MMDP (Multi-Agent Markov Decision Process)

# Erfolgsgeschichten

## ■ Backgammon

- kein Vorwissen über das S
- Über  $10^{20}$  Zustände
- nach 300.000 Spielen besser als jedes andere Computerprogramm



## ■ Karlsruher Brainstormer (Simulation League)

- Wertefunktion als Neuronales Netz
- Gelernte Skills: Ball halten, abfangen, schießen, dribbeln

## ■ Fahrstuhlsteuerung: 4 Fahrstühle im 10 stockigem Haus

- Optimierungsziel: Minimierung der Wartezeiten
- Ergebnis: besser als jedes andere Verfahren
- Optimale Policy jedoch immer noch nicht bekannt

## Zusammenfassung

---

- Agent erlernt eine optimale Lösungsstrategie durch Belohnung/Bestrafung in einer unbekanntem Umgebung. Er passt sich dynamisch einer verändernden Umwelt an.
- Modellierung des Reinforcement Learning Problems durch Markov Decision Processes
- Approximationsalgorithmen zur Berechnung der optimalen Wertefunktion → optimale Strategie
- Probleme in Real-World Szenarien

## Ausblick

---

- Bislang erzielte Ergebnisse sind statischer, handcodierter Strategien überlegen
  - Maschinelles Lernen für einzelne Agenten funktioniert gut
  - Für Multi-Agenten-Systeme existieren Lern-Ansätze
- Anwendung von Reinforcement Learning-Methoden ist sinnvoll und Erfolg versprechend.

# Vielen Dank für Euer Interesse

?

?

?

Fragen?

?

?

?

?

?

?

?

