

# Reinforcement Learning

Prof. Dr.-Ing. J. Marius Zöllner



Forschungszentrum Karlsruhe  
in der Helmholtz-Gemeinschaft

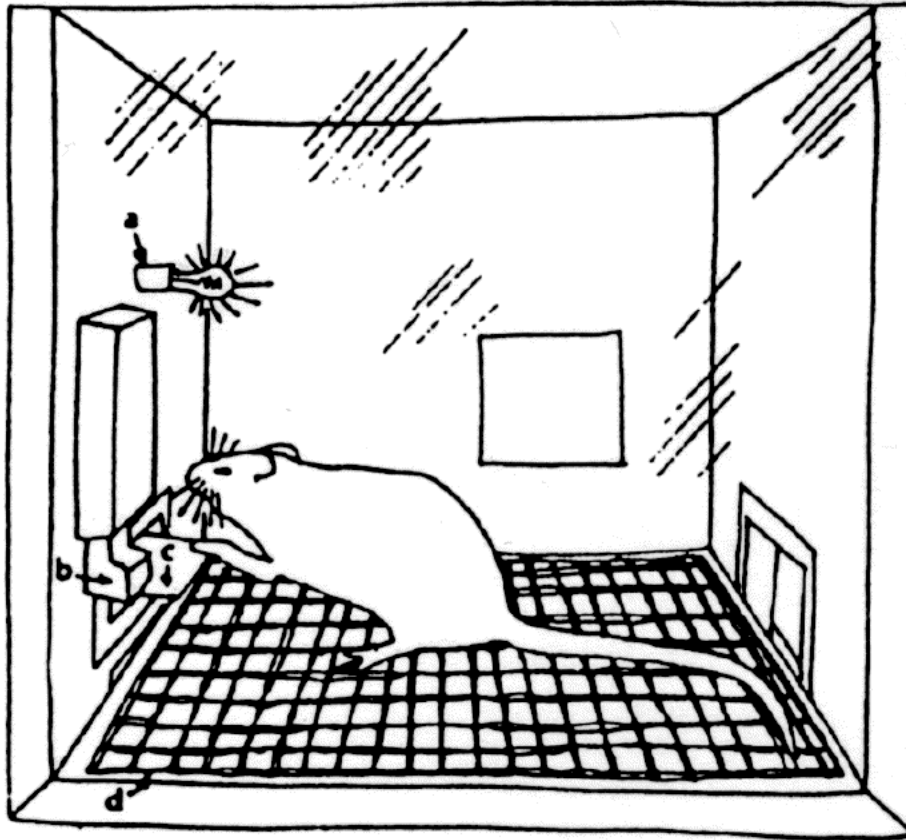


Universität Karlsruhe (TH)  
Forschungsuniversität • gegründet 1825

Thorndike (US-amerikanischer Psychologe, 1911)

Unter verschiedenen Reaktionen, die auf dieselbe Situation hin ausgeführt werden, werden - bei Gleichheit anderer Bedingungen - diejenigen stärker mit der Situation verknüpft, die von einem für das Tier befriedigenden Zustand begleitet oder innerhalb kurzer Zeit gefolgt werden . . . ; diejenigen Reaktionen, die von einem für das Tier unangenehmen Zustand begleitet oder dicht gefolgt werden, werden dagegen - wiederum bei Gleichheit anderer Bedingungen - in ihrer Verknüpfung mit der gegebenen Situation geschwächt.

# Lernen mit Belohnung



- a) Licht
- b) Futtermagazin
- c) Hebel
- d) elektr. Rost

*Abb. 45. Skinner Box*  
(aus: Lefrançois, 1976, S. 63)

# Reinforcement Learning (RL)

Problemstellung

Markov decision process (MDP)

Lernziel

maximale Bewertung (reward)

Problemdimensionen in RL

Strategielernen (Policy-learning)

Optimale Strategie

State Value Function

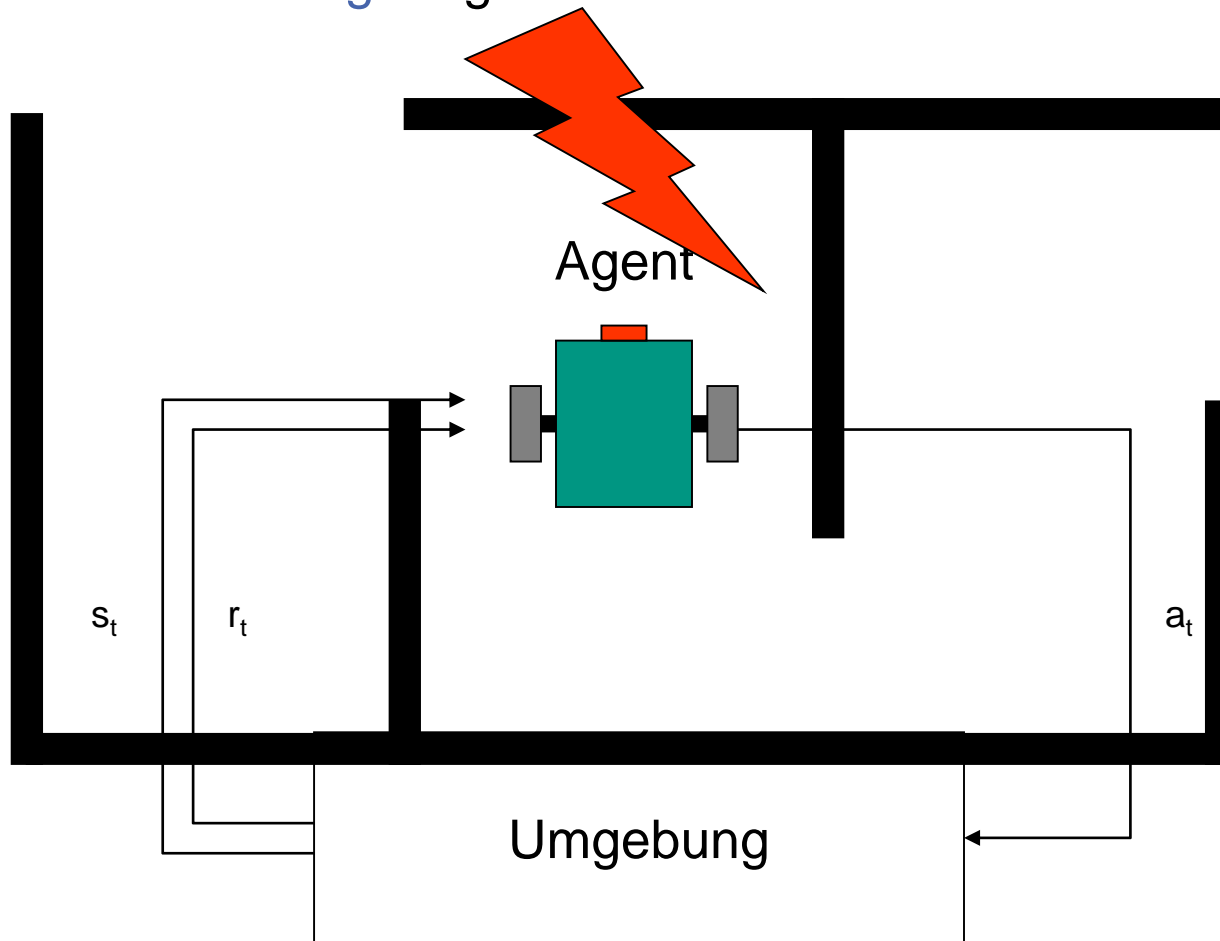
Akkumulierte Bewertung

TD – Lernmethode

(Temporal difference learning)

# Lernziel RL

Finde eine **Aktionssequenz**  $a_1, a_2, \dots, a_n$  so dass dadurch die **maximale Bewertung** aufgesammelt wird



- „Autonomer Agent & Umwelt“:

Zustandsgetriebener Prozess

„Sensorik“ – Erfassung (partiell) von Zuständen  $s_t \in S$

„Aktorik“ – Einwirkung auf die Umwelt durch Aktionen  $a_t \in A$

Zustandsänderungen (meist bekannt oder beobachtbar)

$$\delta : (S \times A) \rightarrow S$$

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

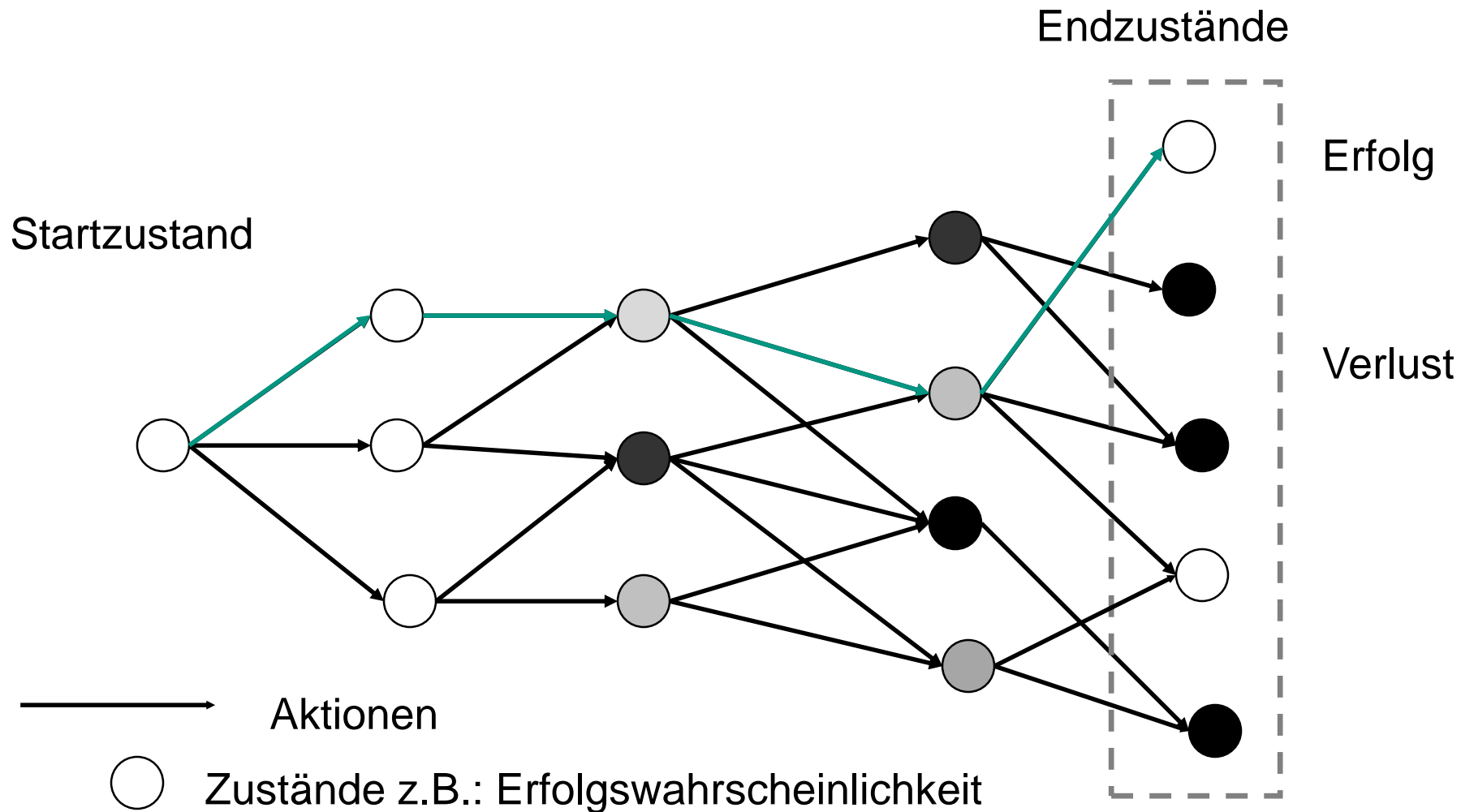
Markov-Bedingung: keine Abhängigkeit von der Vergangenheit

- Bewertung von Aktionen (direkt oder indirekt bekannt/messbar)

$$r : (S \times A) \rightarrow R$$

$$r(s_t, a_t) = r_t$$

# Markov decision process



- Steuerung von Robotern
  - z.B: mobiler Roboter in Büroumgebung
  - Post holen, wenn welche da ist
  - dock-in Station wenn Batterie leer wird .....
- Spiele
  - Brettspiele: Schach, Back-Gammon
- Optimierung und Planung
  - Optimierungen von Fertigungsprozessen
  - Planung Taxizentrale
    - welches Taxi fährt wann wohin,
    - s.d. kurze Wartezeiten für Fahrgäste und wenig Benzinverbrauch
- ....



# Strategielernen - Policy learning

Gesucht:  $s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots \xrightarrow[r_{n-1}]{a_{n-1}} s_n$

↔ finde die (optimale) Zielfunktion (target function)

$$\pi : S \rightarrow A, \quad \pi(s_t) = a_t$$

so dass die akkumulierte Bewertung (zum Ziel hin)

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

maximiert wird

State Value  
Function

Gewichtung der Bewertungen (Diskontierungsfaktor)  $0 \leq \gamma \leq 1$

0 : aktuelle Aktionsbewertung ist wichtig (1-step)

>0: zukünftige (letzte) Bewertungen werden berücksichtigt (n-step)

<1: notwendig um konvergente V-Funktion zu erhalten

Ann.: absorbierender Terminalzustand: Alle Aktionen führen kostenfrei wieder in den Zustand

optimale Zielfunktion

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

maximale akkumulierte Bewertung:

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

rekursive Definition (Bellmann Gleichung):

$$V^*(s_t) = r_t + \gamma V^*(s_{t+1})$$

Problem: Wie erhält man  $V^*(s)$

# Simple Temporal Difference Learning I

## (Simple Value Iteration)

Idee: Lerne  $\hat{V}^*(s)$  als Schätzung von  $V^*(s)$

$$\text{Wähle dann: } \pi^*(s) = \arg \max_a \left[ r(s, a) + \gamma \hat{V}^*(\delta(s, a)) \right]$$

Lernen:

Initialisiere  $\hat{V}^*(s)$  zufällig

do forever

    wähle einen Zustand  $s_t$

    ermittle beste Aktion und den Folgezustand

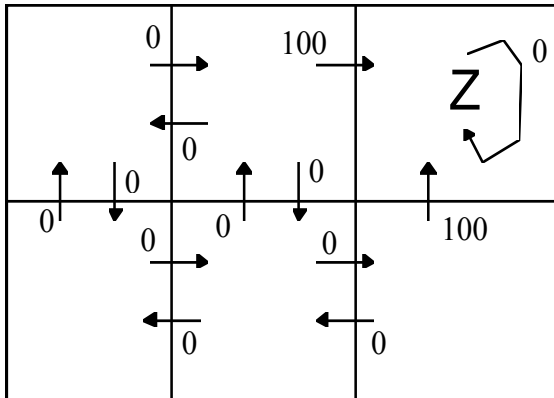
$$s_{t+1} = \delta(s_t, \pi^*(s_t))$$

    ersetze

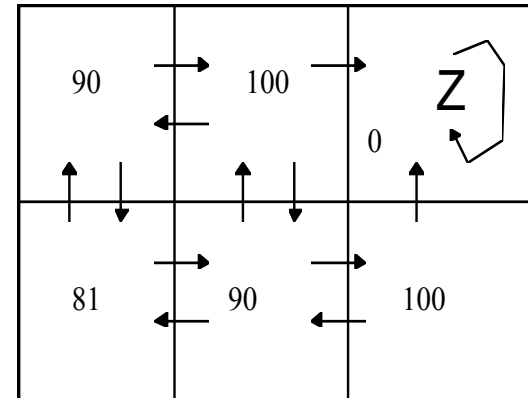
$$\hat{V}^*(s_t) \leftarrow r_t + \gamma \hat{V}^*(s_{t+1})$$

Problem: sehr langsames Lernen („do forever“) und hartes Ersetzen

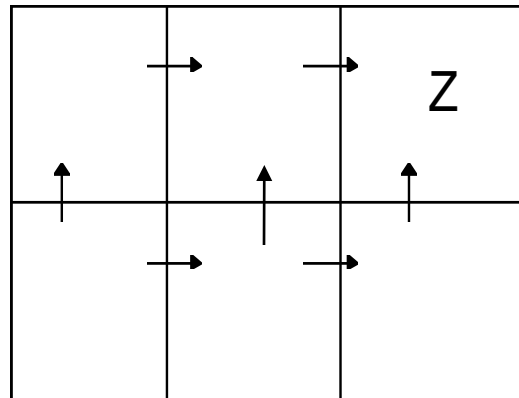
# Beispiel



Zustände, Aktionen  
und Bewertungen,  $\gamma = 0.9$



Maximale akkumulierte  
Bewertung  $V^*(s)$ ,  $\gamma = 0.9$



Optimale Strategie  
 $\pi^*(s)$

# Simple Temporal Difference Learning II

## Optimierung:

Initialisiere  $\hat{V}^*(s)$  zufällig

repeat (für jede Lern - Episode)

wähle einen Zustand  $s_t$

repeat

ermittle Folgezustand  $s_{t+1} = \delta(s_t, \pi^*(s_t))$

ersetze  $\hat{V}^*(s_t) \leftarrow \hat{V}^*(s_t) + \alpha[r_t + \gamma\hat{V}^*(s_{t+1}) - \hat{V}^*(s_t)]$   
( $\alpha = \text{Lernfaktor}$ )

$s_t \leftarrow s_{t+1}$

until  $s_t = \text{terminal}$

## Problem:

$r(s, a)$   $\delta(s, a)$  müssen bekannt sein

on-policy-learning :  $\pi^*(s)$  verwendet → langsam Lernen

Nicht realistisch für echte Anwendungen!!!

- Zielfunktion

Vorhersage:  $V(s_{t+1}) \Leftrightarrow$  Aktionswahl:  $a_t$

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

- Bewertung  $r(s, a)$

direkt  $\Leftrightarrow$  verzögert

- Zustandsübergänge  $\delta(s, a)$

deterministisch  $\Leftrightarrow$  stochastisch

- Modell (Simulation) des Systems

vorhanden  $\Leftrightarrow$  nicht vorhanden

- Zustandsraum und Aktionsraum

eindimensional  $\Leftrightarrow$  hochdimensional

diskret  $\Leftrightarrow$  kontinuierlich

# RL: Einteilung, Q-Lernen, Erweiterungen

## Deterministischer MDP

Q – Funktion (Action-Value Function)

Q - Lernalgorithmus

Suchstrategien

Optimierung

## Generalisierung

z.B.: Neuronale Netze

## Nichtdeterministische MDP

Erweiterung

$Q(s,a)$  maximale Bewertung, die erreicht werden kann  
im Zustand  $s$  durch die Aktion  $a$

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a)) \quad (\text{Bellmann})$$

$$V^*(s) = \max_{a'} Q(s,a')$$

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

Idee: Lerne  $\hat{Q}(s,a)$ ,  $\forall (s,a) \in S \times A$

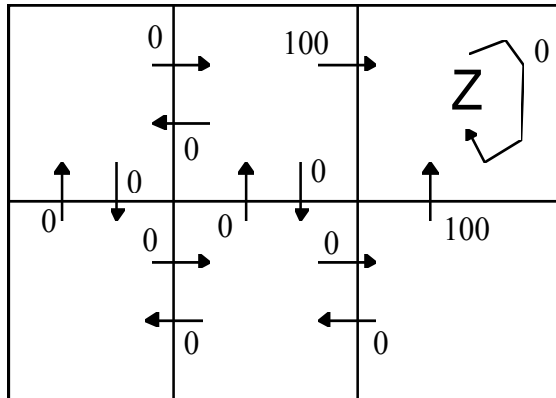
Wähle beste Aktion anhand einer Strategie z.B.:

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

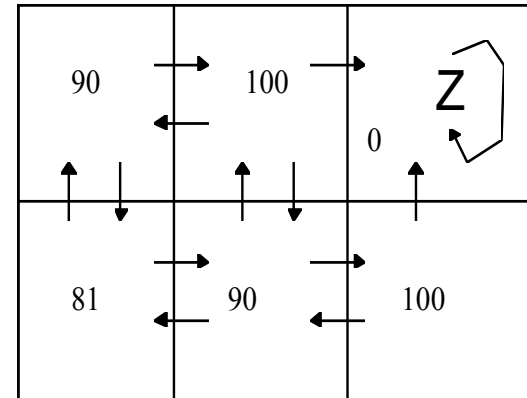
→ In der Anwendung wenig Wissen über Zustandsübergänge nötig



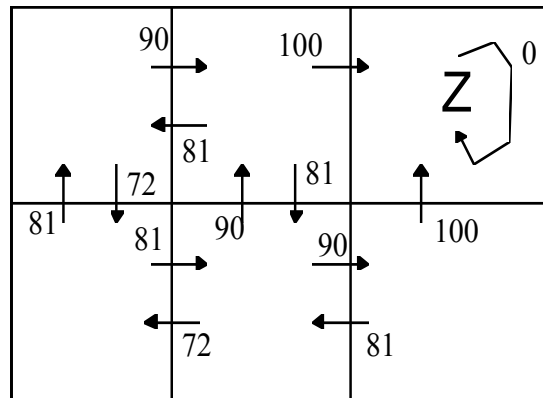
# Beispiel



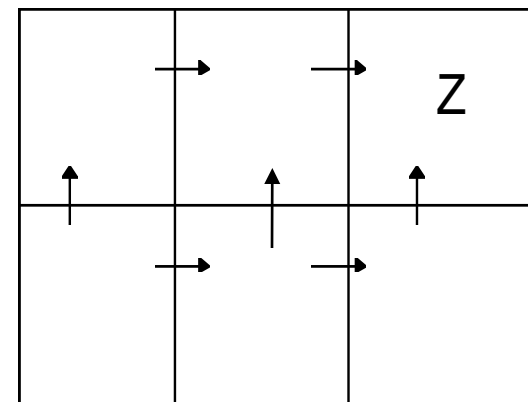
Zustände, Aktionen  
und Bewertungen



Maximale akkumulierte  
Bewertung  $V^*(s)$ ,  $\gamma = 0.9$



$Q(s,a)$



Optimale Strategie  $\pi^*(s)$

# Q-Lernen Algorithmus

**Ziel:** finde Schätzung  $\hat{Q}(s, a)$  der absoluten Funktion  $Q(s, a)$

**Lernen:**

Initialisieren  $\forall s, a \quad \hat{Q}(s, a) = 0$

Wähle Zustand  $s$

do forever:

- wähle Aktion  $a$  und führe aus
- $r \leftarrow$  direkte Bewertung (reward)
- neuer Zustand  $s'$
- update

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

- $s \leftarrow s'$

Nach  $n$  Iterationen gilt

$$\hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a), \forall s, a, n$$
$$\Rightarrow 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$$

Korrektheit und Konvergenz?

JA

wenn: -  $|r(s, a)| < c$   
- deterministisches System  
- wenn alle Paare  $(s, a)$  unendlich oft besucht werden  
 $\Rightarrow \hat{Q} \rightarrow Q$

Fehler wird bei jeder Aktualisierung um den Faktor  $\gamma$  kleiner

**Problem:** Welche Aktion soll in einem Zustand gewählt werden ?

- Aktion mit maximalem  $\hat{Q}(s, a)$  ?
  - „lokales“ Lernen nur bestimmter Aktionen
  - Aktionen die anfangs nicht gewählt wurden, werden nicht weiter ausgewertet obwohl sie ev. bessere Ergebnisse liefern würden

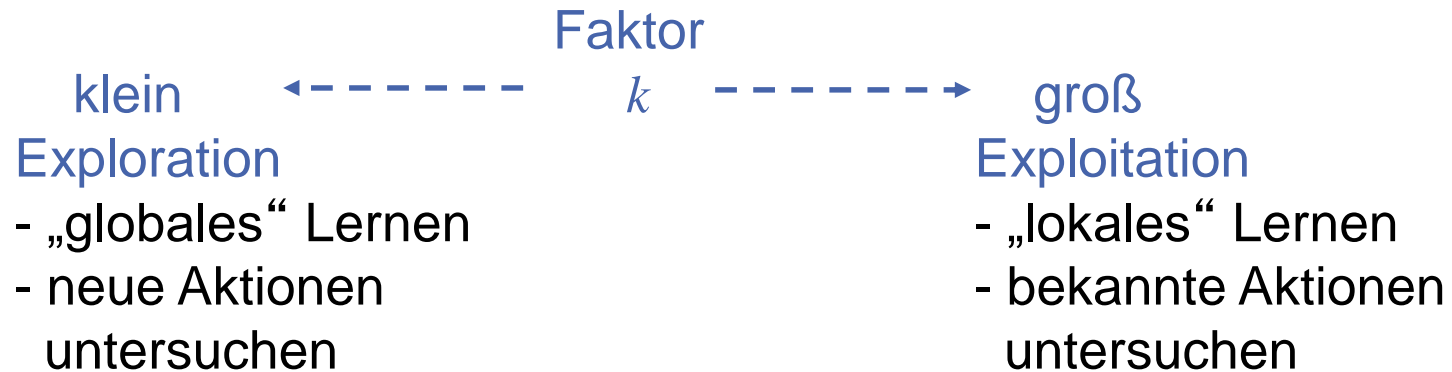
- Probabilistische Auswahl:

$$P(a_i | s) = \frac{k^{\hat{Q}(s, a_i)}}{\sum_i k^{\hat{Q}(s, a_i)}}, k > 0$$

$$\rightarrow \forall a_i \exists P(a_i | s) \neq 0$$

# Exploration vs. Exploitation

Probabilistische Aktionsauswahl je nach:



Beste Lösung:

Änderung der Suchstrategie von global zu lokal während des Lernprozesses

- In jeder Lernepisode alle  $\hat{Q}(s, a)$  vom Zustand  $s$  zum Ziel anpassen

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha[r + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$$

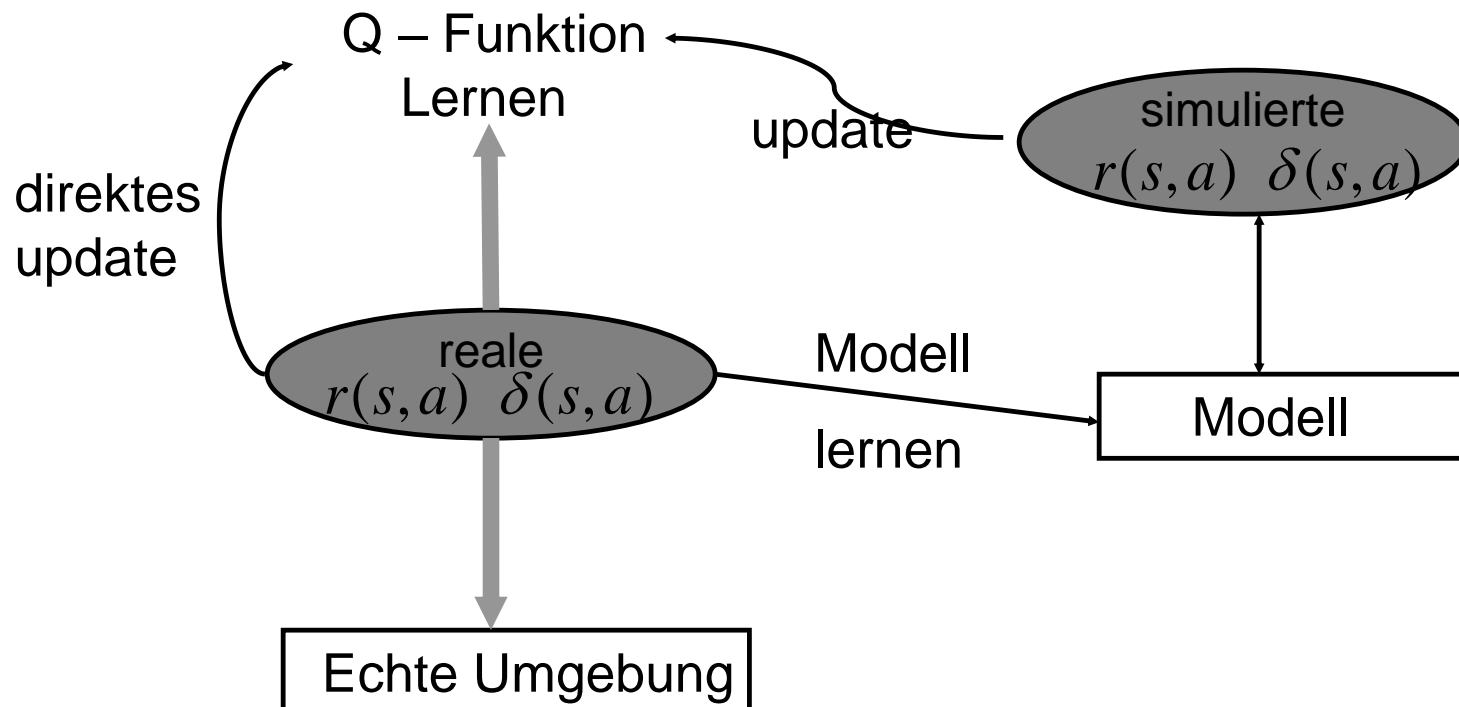
- Speichern von Bewertung  $r$  für jedes Paar  $(s, a)$   
wenn  $\hat{Q}(s, a)$  von  $\hat{Q}(s', a')$  abhängig ist und sich  $\hat{Q}(s', a')$  ändert  
ändere auch  $\hat{Q}(s, a)$

- ➔ schnelle Konvergenz  $\hat{Q} \rightarrow Q$  durch Anpassung mehrerer Werte
- ➔ Speicheraufwand steigt

## Anwendung

Immer dann, wenn Aktionen hohen Zeitaufwand haben (z.B: in der Robotik)

- Lernen mit (adaptivem) Modell
  - meisten Lernschritte auf simulierter Umgebung
  - wenig Aktionen in realer Umgebung
  - Anpassung des Modells

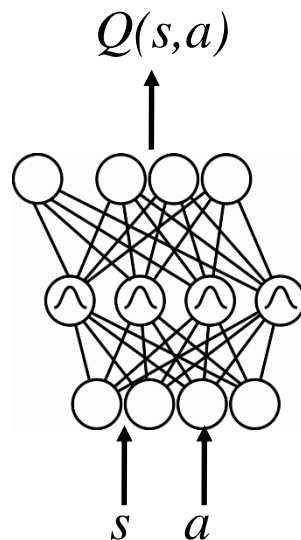


## Problem: kontinuierlicher Zustandsraum

- Speichern der  $Q$  - Werte in einer lookup-Tabelle unmöglich
- sehr hohe Anzahl von Lerniterationen nötig

## Lösung:

Kombination von RL mit Methoden höherer Generalisierung

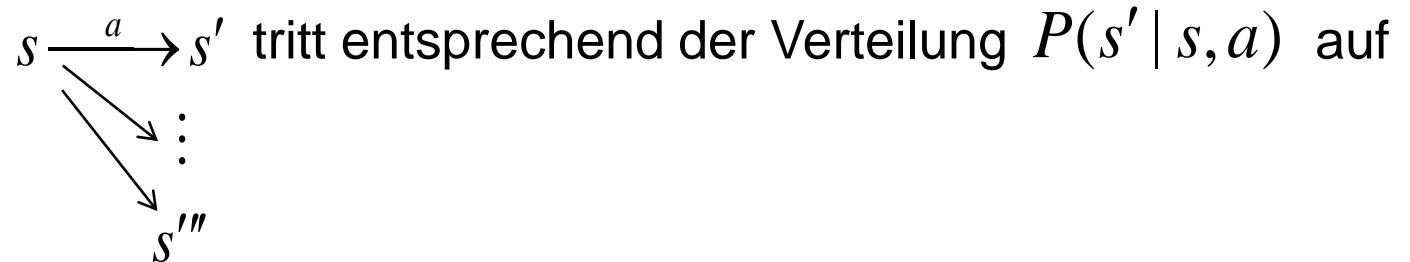


z.B.: Neuronale Netze

- ein Netz für alle Aktionen oder
  - pro Aktion ein Netz oder
  - direktes Lernen der besten Aktion
- 
- update-Werte für  $\hat{Q}(s,a)$  sind Lernbeispiele für das neuronale Netz



## Problem



## Lösung

Verwenden des Erwartungswertes

$$V^{\pi}(s_t) = E \left[ \sum_{i=0}^{\infty} \gamma^i r_{t+i} \right]$$

Berücksichtigen aller Folgezustände

$$Q(s, a) = E[r(s, a)] + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(\delta(s', a), a')$$

# Lernen von Aktionssequenzen

Problemdefinition

$TD(\lambda)$  - Lernen

Vorwärtssicht

Rückwärtssicht

# Warum Lernen von Aktionssequenzen?

- Bewertung (reward) erst nach einer Sequenz von Aktionen bekannt
  - z.B: Schach: selten ist nur ein Zug relevant
- Bewertung erst am Ziel
  - z.B: Spiel gewonnen ?

→ bei langen Aktionssequenzen kann erst am Ende der Sequenz gelernt werden

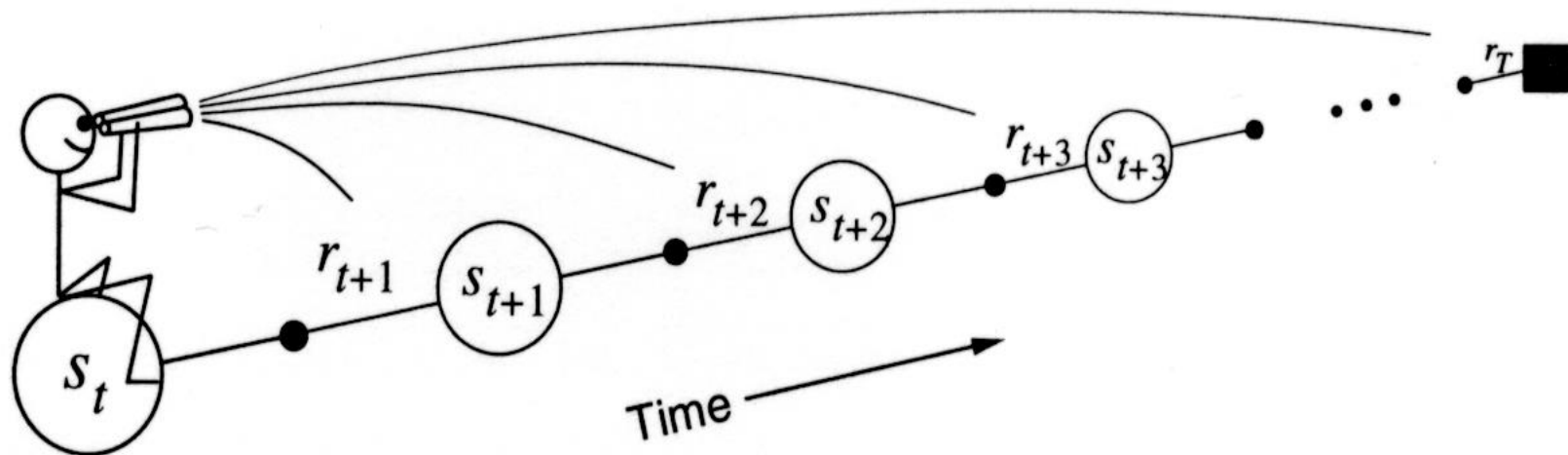
→ nachfolgende Aktionen können für den schlechten Ausgang verantwortlich sein

## Temporal\_Difference\_Learning

die Differenz folgender Schätzungen als Lernsignal für  $\hat{Q}(s, a)$ ,  $\hat{V}^*(s)$

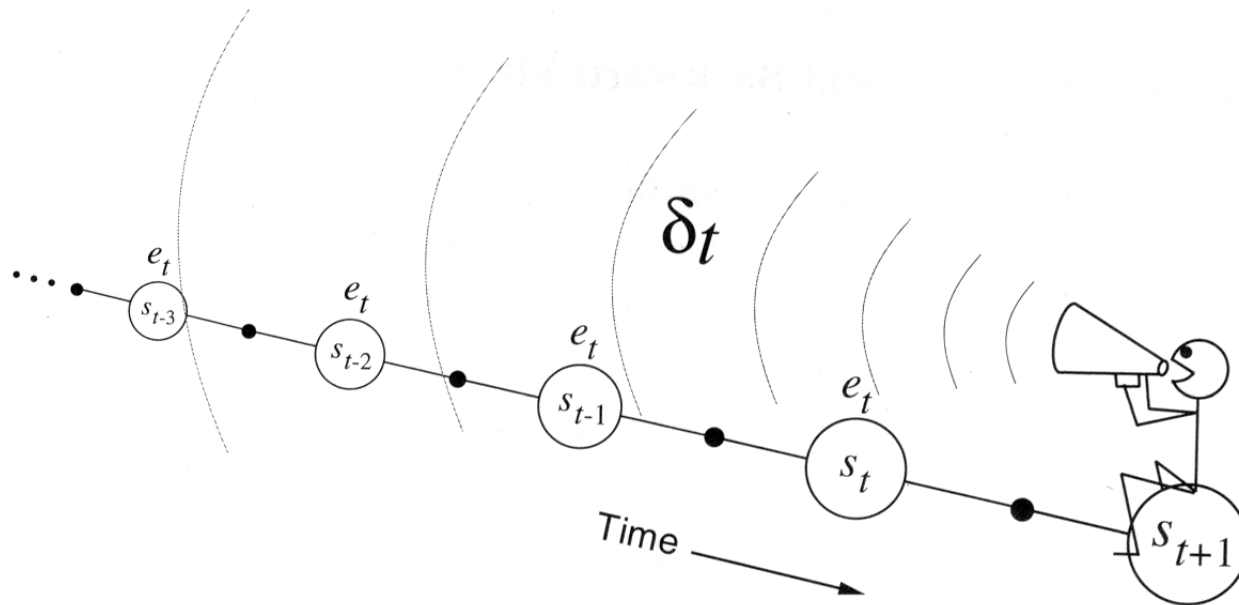
## Vorwärts – Sicht (theoretisch)

- Gewichtete Anpassung an direkt nachfolgender Schätzung (1-step) oder
- Gewichtete Anpassung an n Schritte nachfolgender Schätzung (n-step)



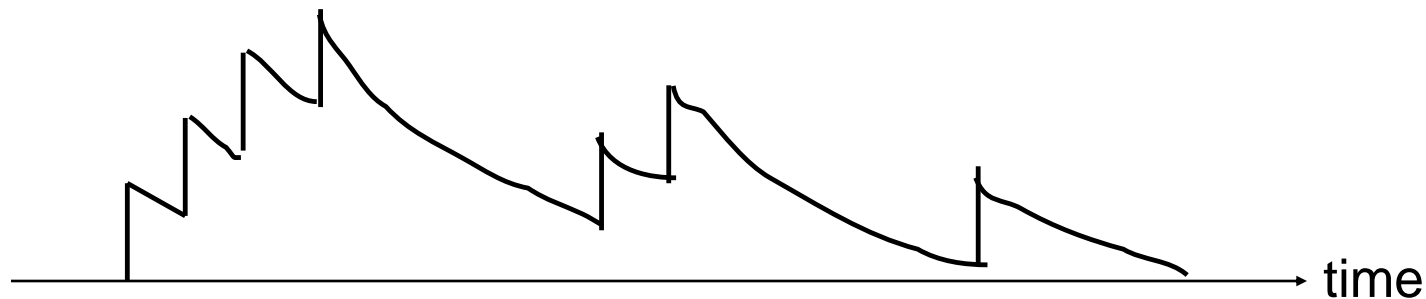
## Rückwärtige Sicht des TD-Lernen (praktisch)

Fehlersignale (temporal differences) in den Schätzungen werden nach hinten weitergegeben



## Eligibility Traces (Verantwortlichkeitsspur)

- Zustände für die Zustandsbewertung  $\rightarrow V$  – Lernen
- (Zustand/Aktion) für die  $Q$ -Wert Bewertung  $\rightarrow Q$  -Lernen



Eligibility trace  $e_t(s)$ : Akkumulativ

z.B. für Zustände:

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) & , s \neq s_t \\ \gamma \lambda e_{t-1}(s) + 1 & , s = s_t \end{cases}, 0 < \lambda < 1$$

# SARSA( $\lambda$ )-Algorithmus mit Eligibility Traces

Lernen:

Initialisiere  $\hat{Q}(s, a)$   $e(s, a) = 0, \forall s, a$

repeat (für jede Lernepisode)

    wähle  $s, a$

    repeat

$$r = r(a, s), s' = \delta(a, s)$$

$$a' = \pi^Q(s'),$$

$$\text{if } r: \Delta_Q \leftarrow r + \gamma \hat{Q}(s', a') - \hat{Q}(s, a)$$

$$e(s, a) \leftarrow e(s, a) + 1$$

für alle  $s'', a''$

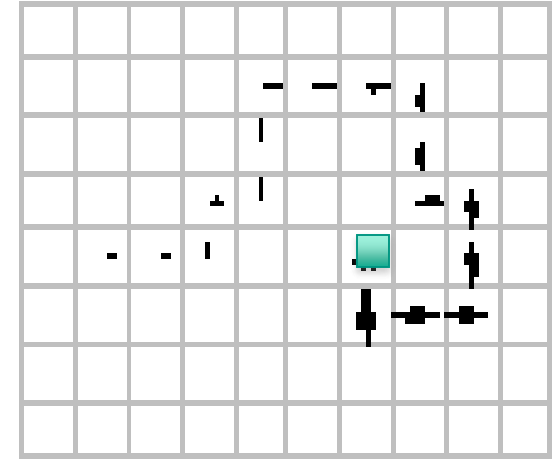
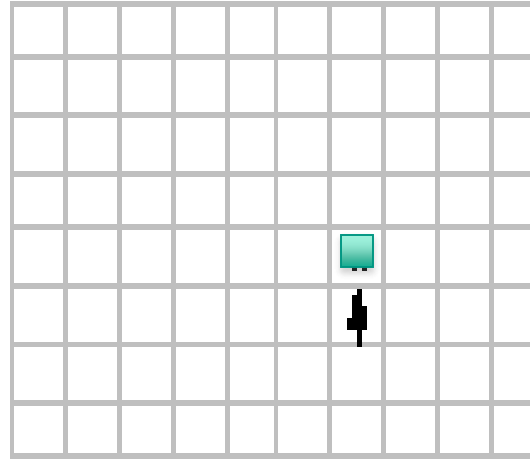
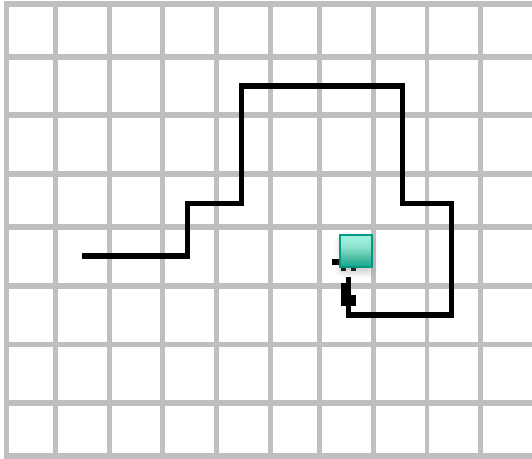
$$\text{if } r: \hat{Q}(s'', a'') \leftarrow \hat{Q}(s'', a'') + \alpha \Delta_Q e(s'', a'')$$

$$e(s'', a'') = \gamma \lambda e(s'', a'')$$

$$s \leftarrow s', \quad a \leftarrow a'$$

until  $s = \text{terminal}$

# SARSA Beispiel



- Ziel – Pfad zum Ziel finden
- Anfangs alle  $Q = 0$
- Reward nur im Ziel
- Mitte – Lernen ohne Eligibility – Trace
  - Nur ein Q- Wert wird angepasst
- Rechts – Einfluss der Anpassung mit Eligibility



## Katz und Maus

<http://www.cse.unsw.edu.au/%7Ecs9417ml/RL1/applet.html>

## Black Jack and Reinforcement Learning

<http://lslwww.epfl.ch/~anperez/BlackJack/classes/RLJavaBJ.html>

## Fahrstuhlsteuerung

## TD – Gammon

## Steuerung von biologisch motivierten Systemen

## Reale praktische Anwendung + Lernen in der Simulation

**Ziel:** Fahrstühle steuern

Optimierungskriterien:

- Mittlere Wartezeit
- Mittlere Systemzeit
- % der Wartezeit über 60 sec

hier: Minierung mittlerer Wartezeit

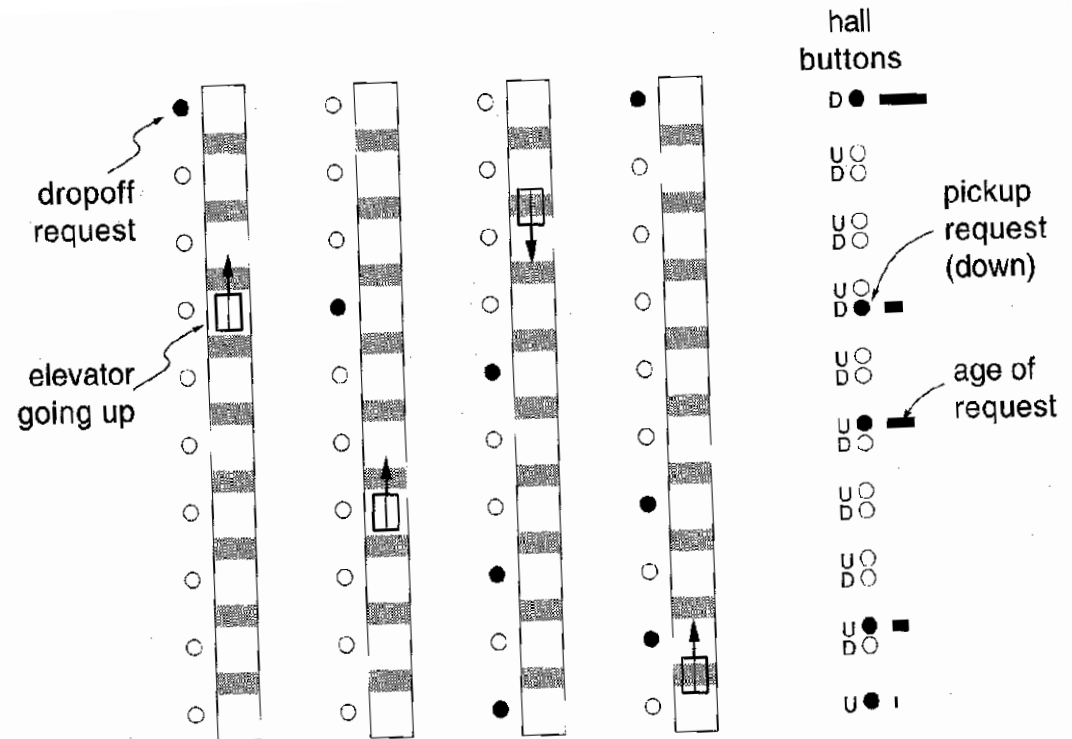


Figure 11.8 Four elevators in a ten-story building.

# Fahrstuhlsteuerung - Randbedingungen des Lernens

Integration von **Vorwissen** zur Vereinfachung des Lernproblems

## Randbedingungen (Regeln):

- Jeder Fahrstuhl trifft selbständig Entscheidungen
- Kein Überfahren eines Stockwerkes, wenn dort Passagiere aussteigen wollen
- Keine Richtungswechsel, solange Passagiere in die aktuelle Richtung fahren wollen.
- Ein Fahrstuhl darf an einem Stockwerk nicht halten, wenn dort niemand ein- oder aussteigen möchte.
- Kein Fahrstuhl soll an einem Stockwerk halten, wenn ein anderer die Fahrgäste schon aufgenommen hat.
- Wenn die Wahl besteht zwischen hoch und runter, sollen die Fahrstühle hoch fahren.

# Fahrsstuhlsteuerung - Netzrepräsentation

**Aktion:** Entscheidung ob am nächsten Stockwerk mit Anfrage gehalten werden soll und ob hoch oder runter gefahren werden soll

**Problem:** Bei relativ grober Diskretisierung  $10^{22}$  mögliche Zustände

**Lösung:** Generalisierung durch Backpropagation Netz

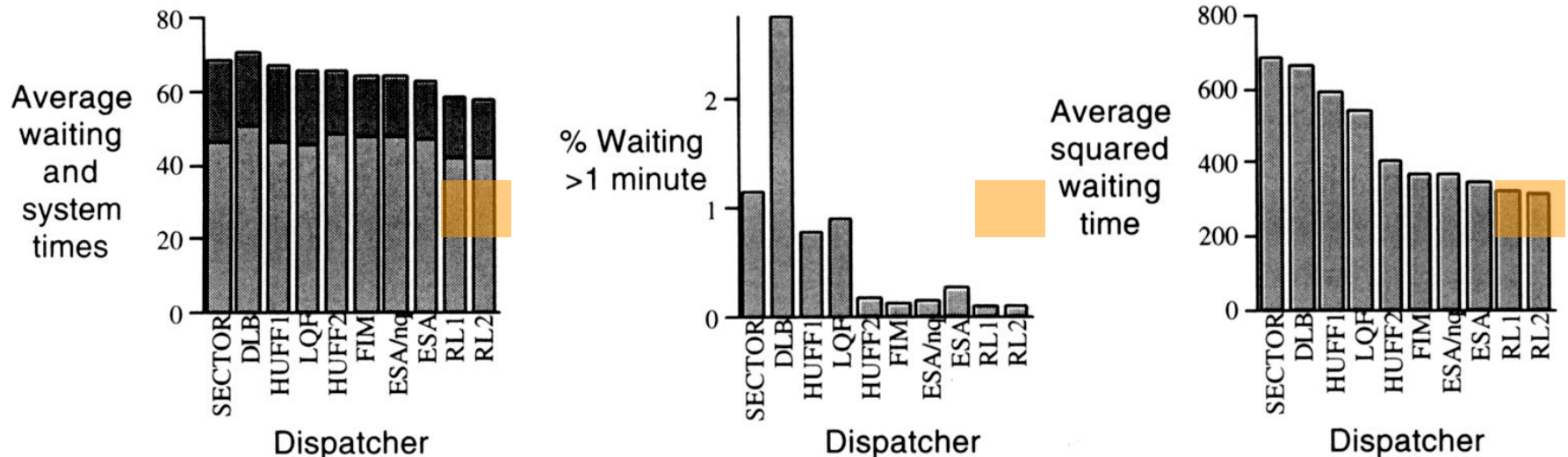
- 47 Eingaben, 20 Hidden Neuronen, 2 Ausgaben
- Ausgaben: 2  $Q$ -Values: für halten / nicht halten, hoch / runter
- Eingaberepräsentation
  - 9 binäre: Runter-Anfragen
  - 9 kont.: Wartezeit der Runter-Anfragen
  - 16 binär: für jede Ort-Richtungs-Möglichkeit Fahrstuhl / Etage
  - 10 diskret: Die Orte der anderen Fahrstühle (Etage)
  - 1 binär: Fahrstuhl auf höchster Etage mit wartendem Passagier
  - 1 binär: Fahrstuhl auf der Etage mit dem am längsten wartenden Passagier
  - 1 Bias Unit ( immer an)

# Fahrstuhlsteuerung - Ergebnisse

## 2 Ansätze mit Neuronalen Netzen

- » Jeder Fahrstuhl eigenes Netz
- » Alle Fahrstühle in einem Netz

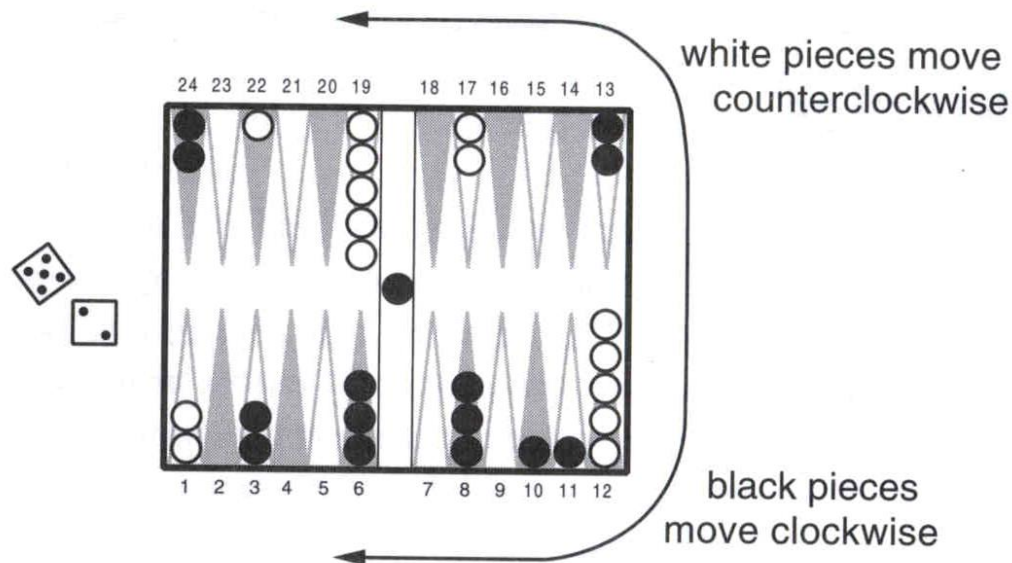
Bei beiden Ansätzen wurden die Bewertungen von allen Fahrstühlen verwendet → keine unabhängigen Systeme



Ergebnisse verschiedener Systeme im Vergleich

# TD – Gammon

- Ziel:** Spiel – Strategie um ein Backgammon Spiel zu gewinnen
- Problem:** Bewertung am Spielende: gewonnen oder verloren  
Zufallskomponente - Würfel
- Zustände:** 24 mögliche Positionen für 30 Steine → hochdimensional
- Aktionen:** Zug des Spielers
- Lösung:** Lerne  $\hat{V}^*(s)$  als Gewinnprognose eines Zustandes



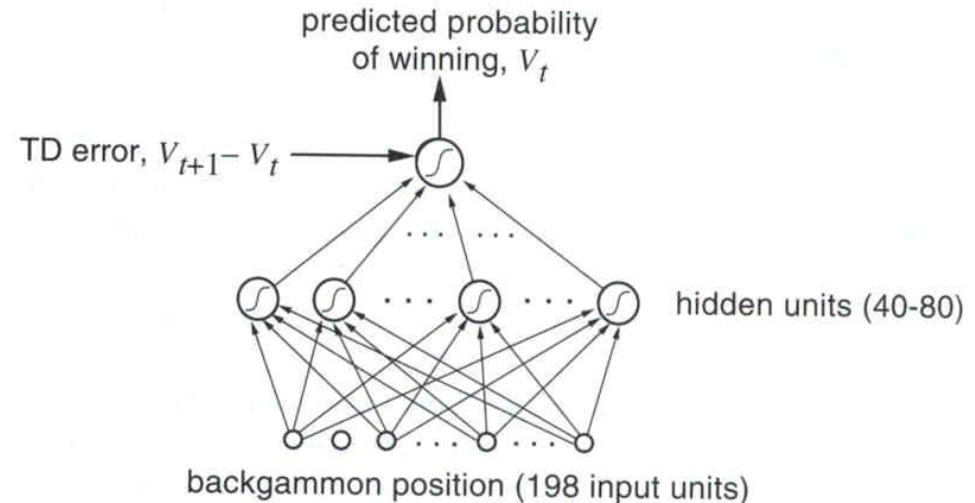
Generalisierung:  
Neuronales Netz

Zustandsbeschreibung  
(Eingabe):

- (schwarz + weiß) \* 4 \* 24 = 192 für die Belegung der Felder
- (schwarz + weiß) = 2 für die Steine, welche draußen sind
- (schwarz + weiß) = 2 für die Steine, welche herausgeholt wurden
- (schwarz + weiß) = 2 binär: nächster Zug schwarz oder weiß

Ausgabe:  $\hat{V}^*(s)$

- Gewinnwahrscheinlichkeit



Lernen:

- TD( $\lambda$ )-Algorithmus mit Eligibility Traces
- Spiel gegen sich selbst
- kein spezifisches Spielwissen nötig (außer Spielregeln)

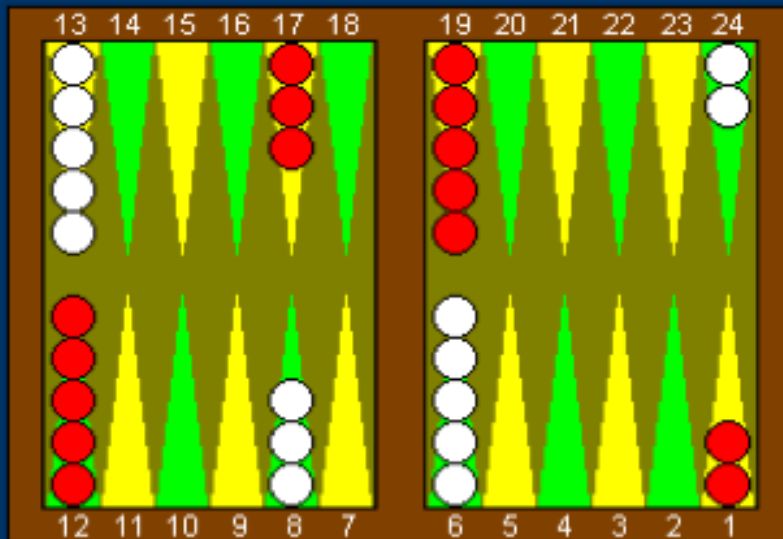
Ergebnisse:

Program	Hidden Units	Training Games	Opponents	Results
TD-Gam 0.0	40	300,000	other programs	tied for best
TD-Gam 1.0	80	300,000	Robertie, Magriel, . . .	−13 points / 51 games
TD-Gam 2.0	40	800,000	various Grandmasters	−7 points / 38 games
TD-Gam 2.1	80	1,500,000	Robertie	−1 point / 40 games
TD-Gam 3.0	80	1,500,000	Kazaros	+6 points / 20 games

- neue Spielzüge wurden vom Programm erzeugt und werden inzwischen allgemein akzeptiert und verwendet



Seit einer Analyse durch TD-Gammon wird ein Eröffnungszug, der bis dahin als selbstverständlich galt, praktisch nicht mehr gespielt.



- Wurf: 4/1
- weiß ist am Zug

Move	Estimate	Rollout
13-9, 6-5	-0.014	-0.040
13-9, 24-23	+0.005	+0.005

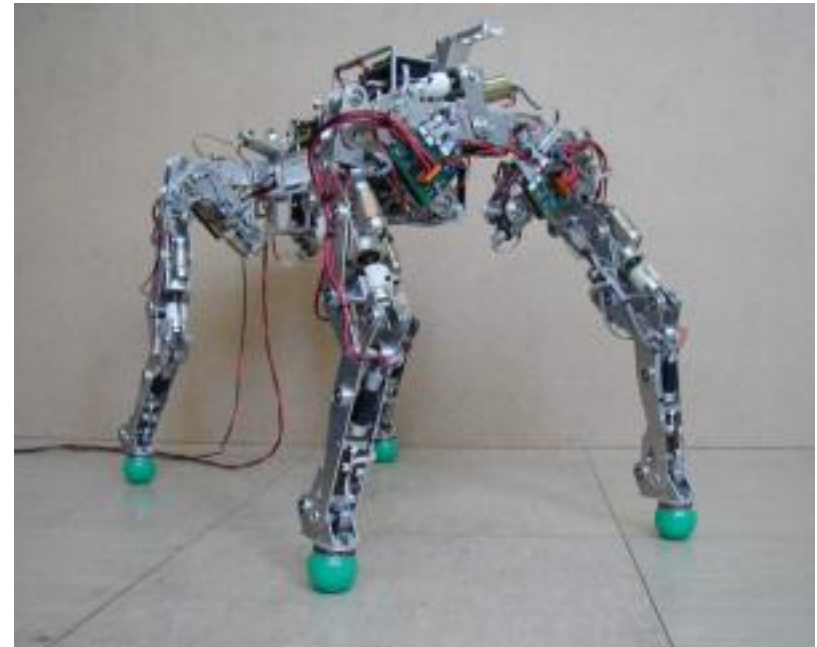
# Online-Lernen auf der vierbeinigen Laufmaschine BISAM (Ilg00/Albiez00)

## Ziele

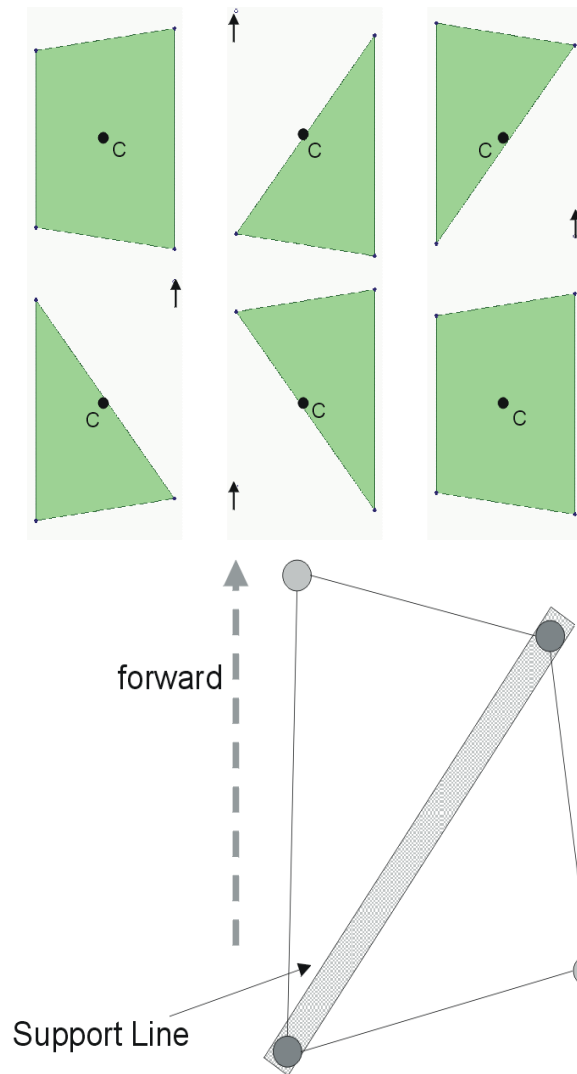
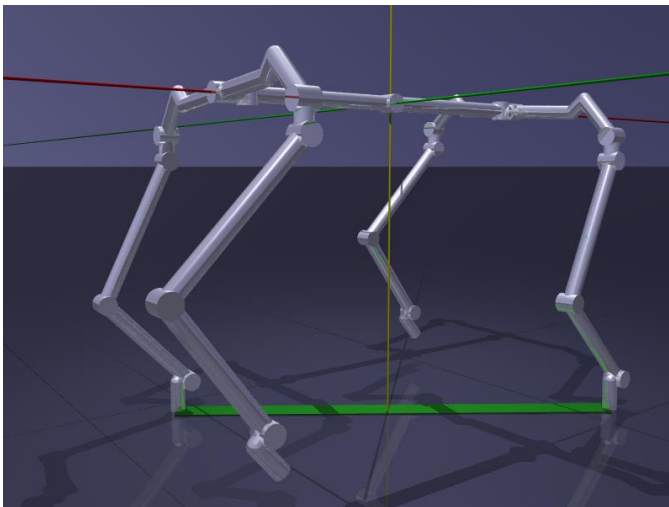
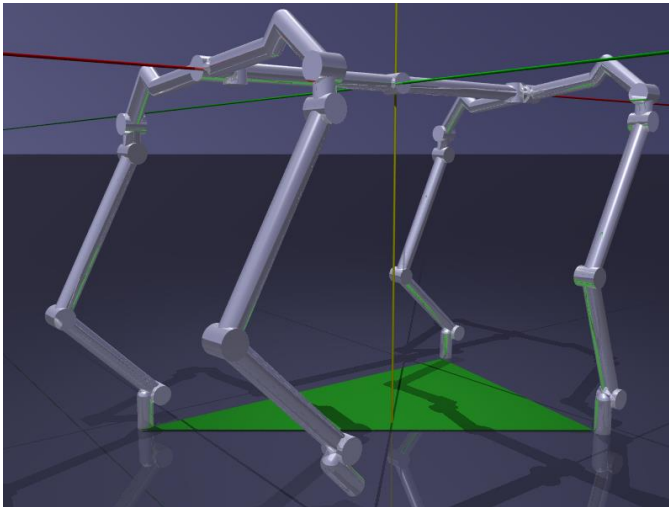
Experimentelles Lernen zur  
Optimierung der Haltung  
(Schwerpunkt) im Kreuzgang

## Roboter BISAM

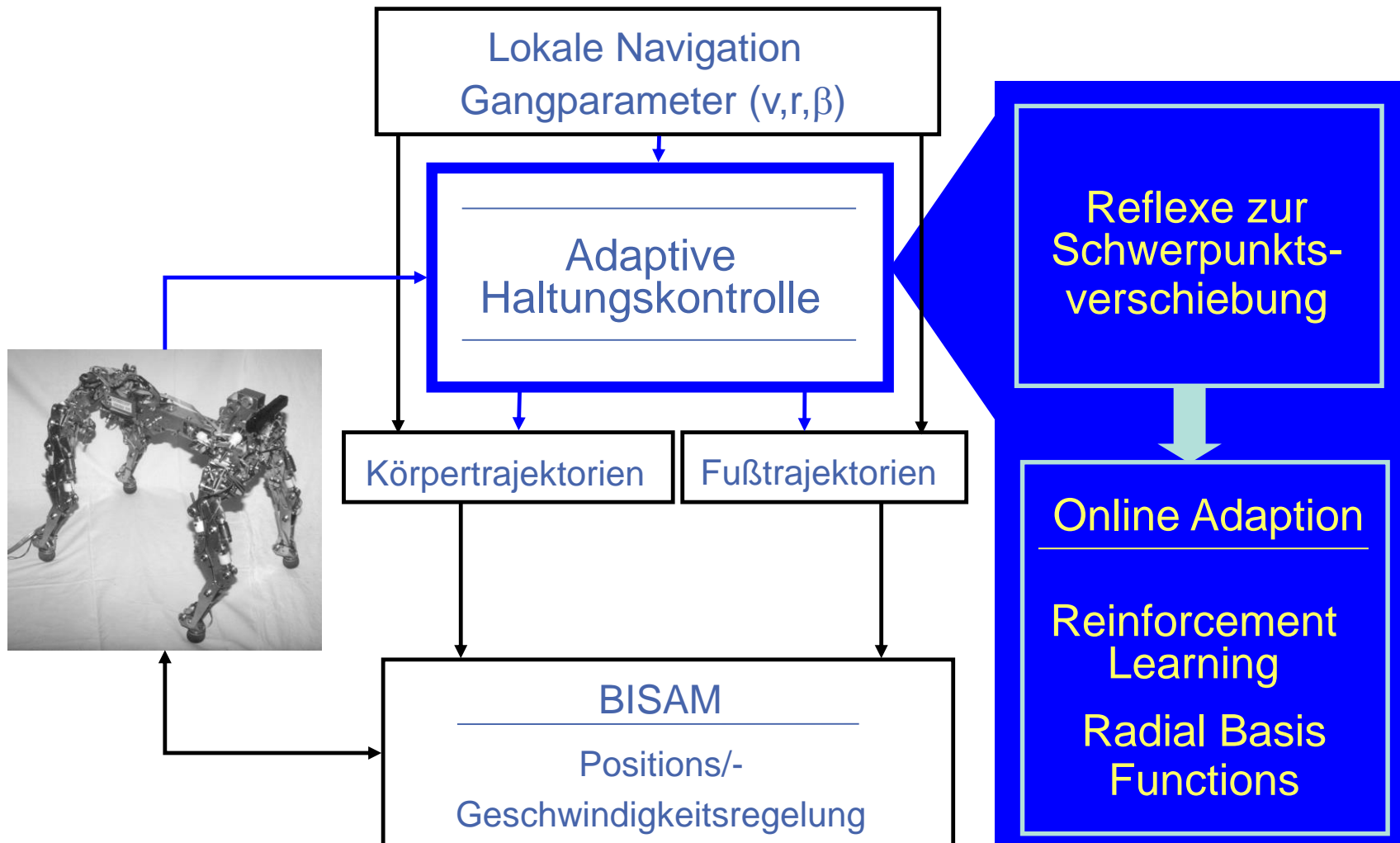
21 Aktive Freiheitsgrade (4 pro  
Bein 5 im Körper)  
2 Achsen Lagesensor, 3 Achsen  
Winkelbeschleunigung  
3 Komponenten Kraftsensor pro  
Fuß  
Onboard PC, 6 C167  $\mu$ C



# BISAM – Dynamisch und Statisch stabiles Laufen



# BISAM - Steuerungsarchitektur



# BISAM – Lernen der Gewichtsverlagerung im Schritt

## ■ Problemstellung

- Optimierung der Schwerpunktsverschiebung zur Unterstützung des Schritts

## ■ Bewertungskriterien

- Anhand der Lage des Schwerpunktes ( $SSM$ )

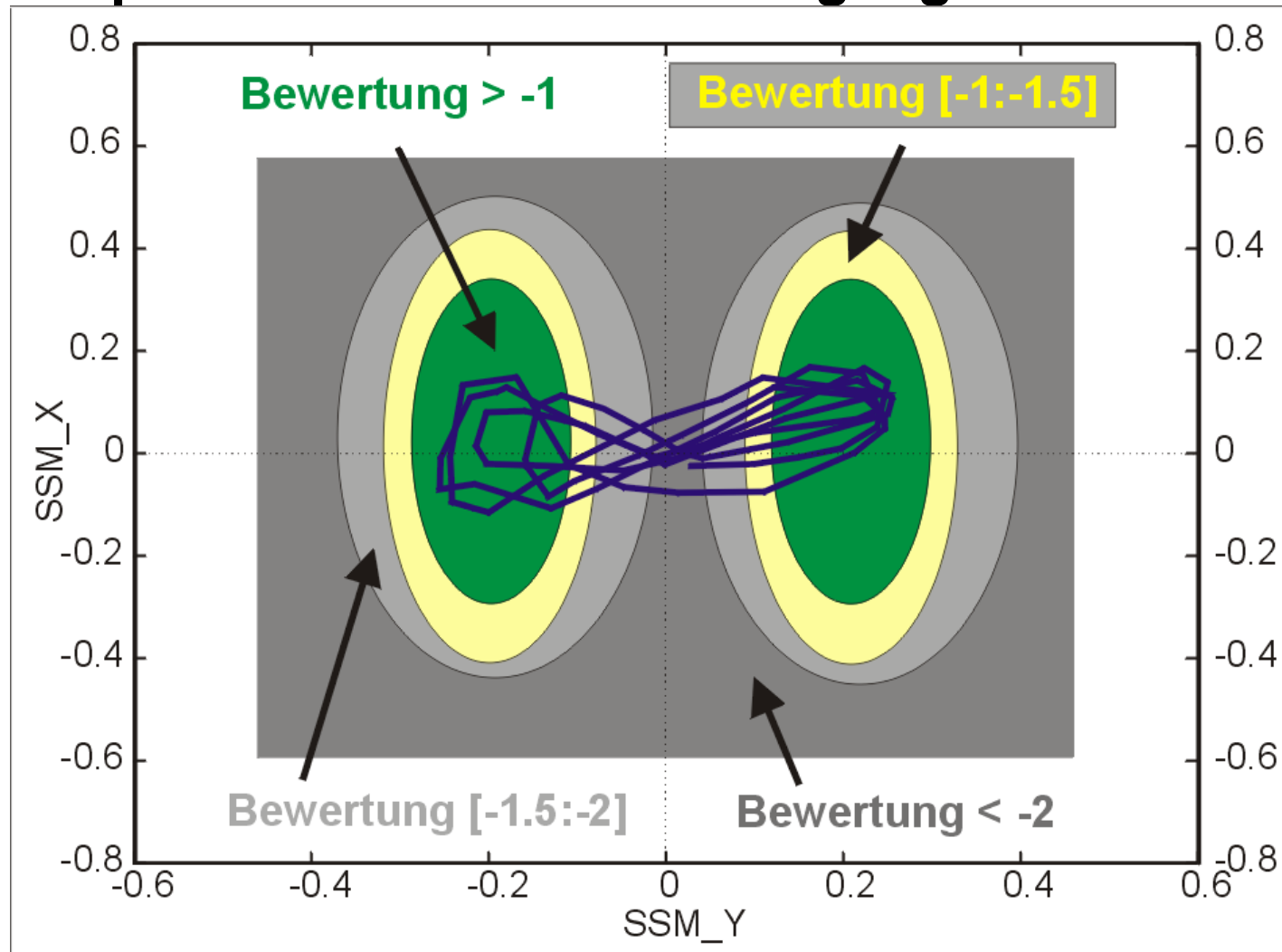
## ■ Netzeingaben

- Zustand:  $s = (SSM_x, SSM_y)$
- Letzte Aktion: Verschiebung  $a_{t-1} = (a_x, a_y)$

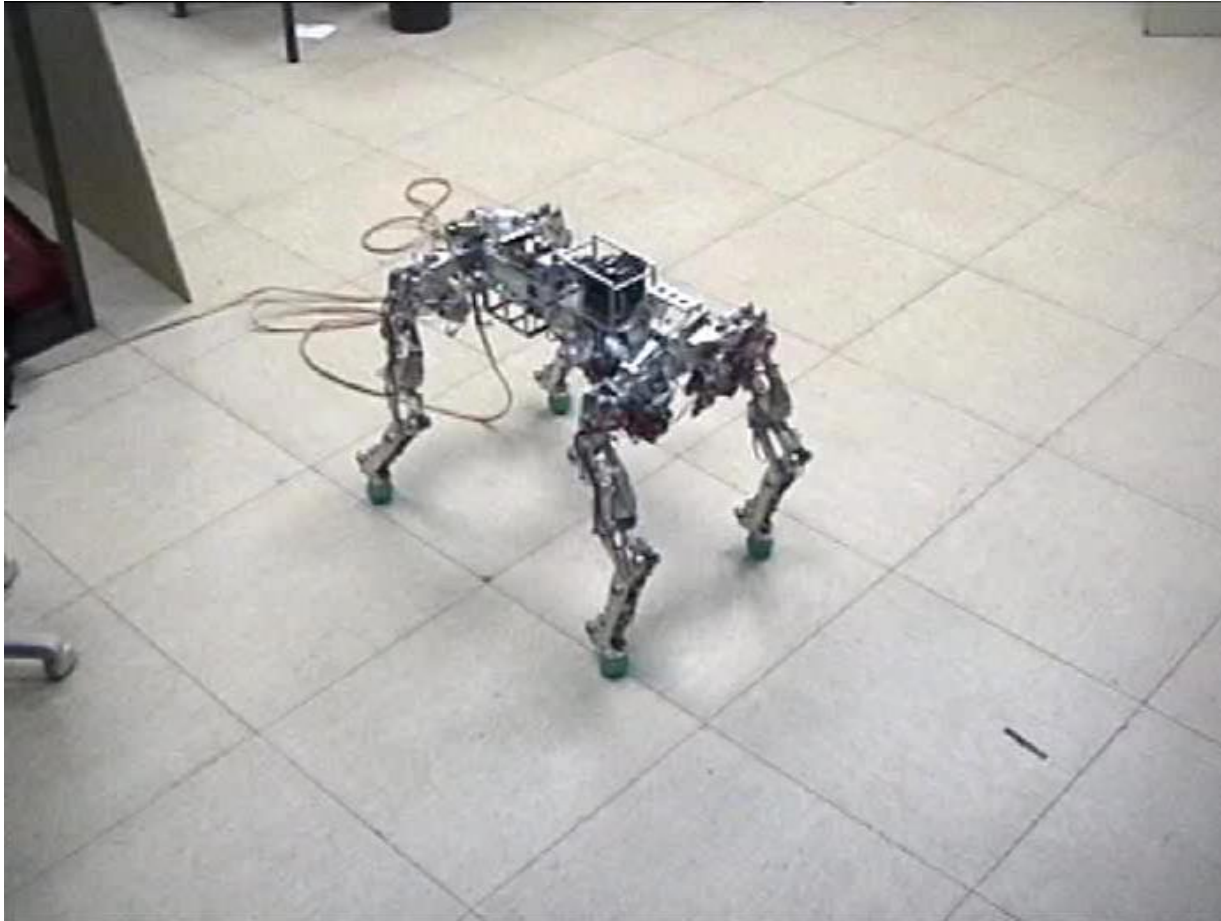
## ■ Netzausgaben

- Kritiker:  $V(s)$
- Neue Aktion: Verschiebung  $a_t = (a_x, a_y)$

# BISAM – Bewertung des Schwerpunktsverlaufes im Kreuzgang



# BISAM – Lernverlauf Kreuzgang



- [1] Tom Mitchell: **Machine Learning**. McGraw-Hill, New York, 1997.
- [2] R.S. Sutton, A.G. Barto: **Reinforcement Learning – An Introduction**. MIT Press, 1998.
- [3] L.P. Kaelbling, M.L. Littman, A.W. Moore: **Reinforcement Learning – a survey**. Journal of Artificial Intelligence Research, Vol. 76, 1995.
- [4] Vorlesung: **Neuronale Netze 2006**.  
<http://isl.ira.uka.de/>
- Gegenstand aktueller Forschung z.B. Prof. Riedmiller (Freiburg), Prof. Jan Peters (Darmstadt/Tübingen)



# Motor Skill Learning for Robotics

Jan Peters, Jens Kober, Katharina Mülling  
Department of Empirical Inference and Machine Learning  
Max Planck Institute for Biological Cybernetics