Reinforcement Learning

VL Algorithmisches Lernen, Teil 3d

Jianwei Zhang

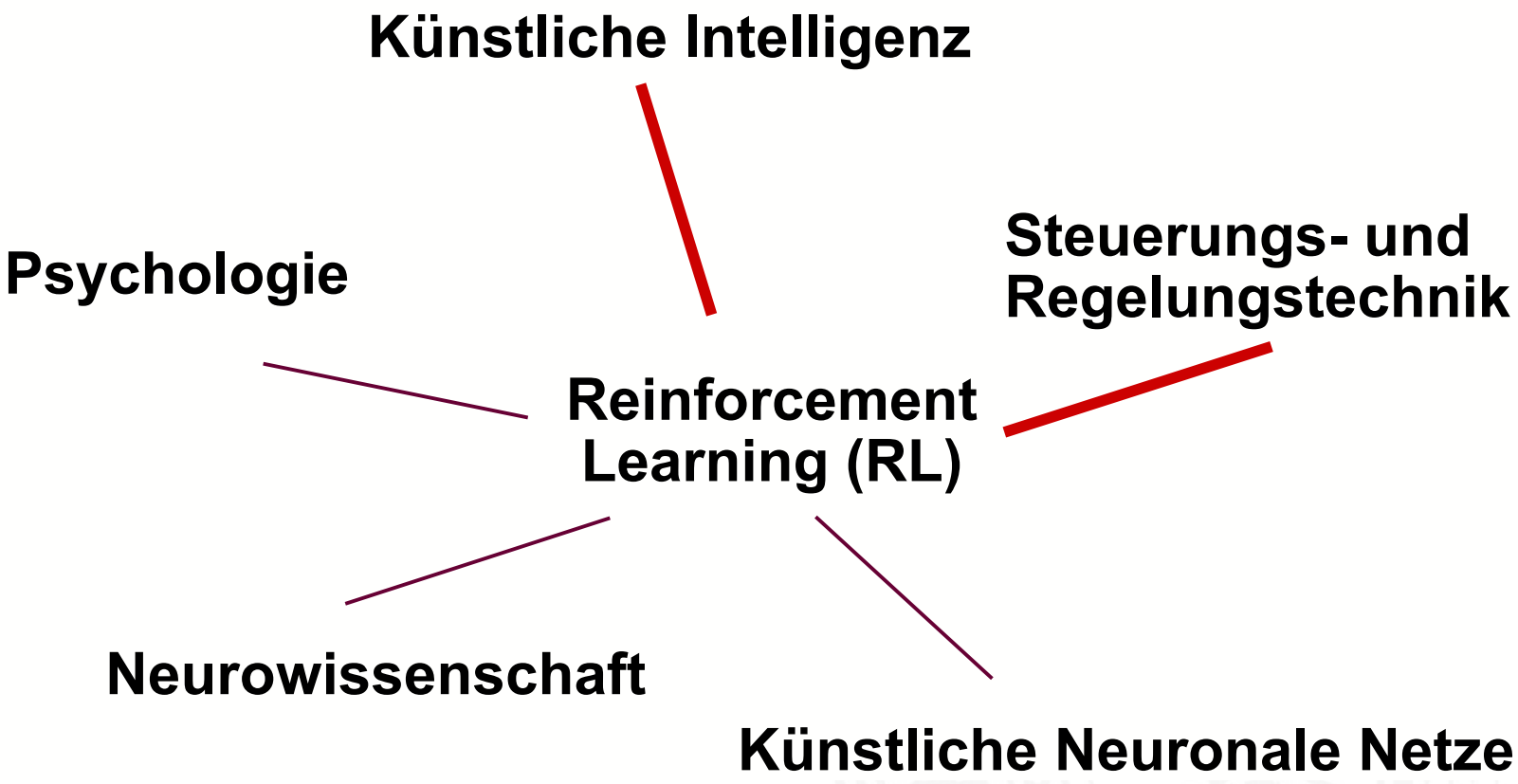
University of Hamburg
MIN Faculty, Dept. of Informatics
Vogt-Kölln-Str. 30, D-22527 Hamburg
zhang@informatik.uni-hamburg.de

08/07/2009

Terminübersicht: Part 3

- 17/06/2009 Dimensionsproblem, PCA
- 18/06/2009 Support-Vektor Maschinen
- 24/06/2009 Support-Vektor Maschinen (4st.)
- 25/06/2009 Übung
- 01/07/2009 Funktionsapproximation, Fuzzy-Logik (4st.)
- 02/07/2009 Übung (Prof. Dr. Menzel)
- 08/07/2009 **Verstärkungslernen (1)**
- 09/07/2009 Verstärkungslernen (2)
- 15/07/2009 Verstärkungslernen (3)
- 16/07/2009 Anwendungen in der Robotik

Einführung

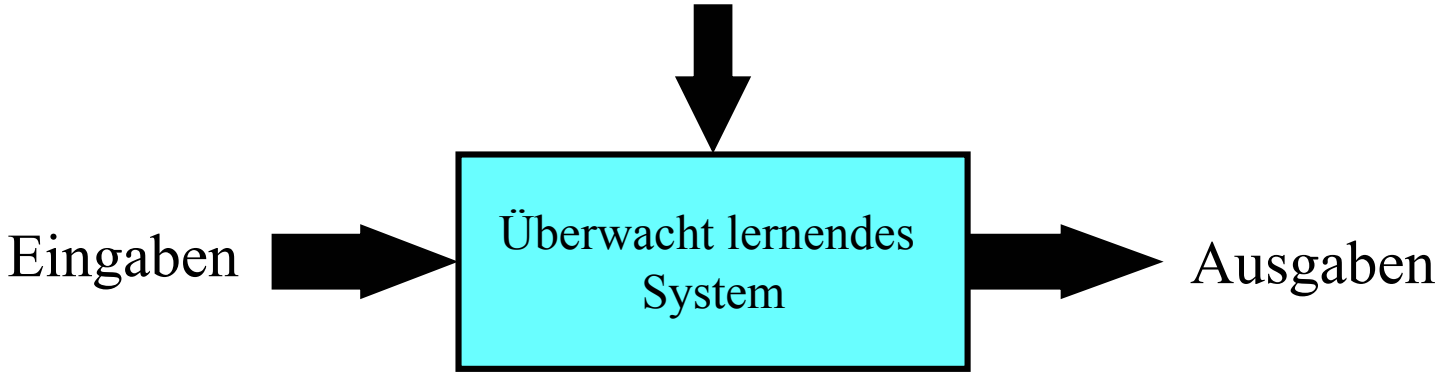

Künstliche Intelligenz
Psychologie
Steuerungs- und Regelungstechnik
Reinforcement Learning (RL)
Neurowissenschaft
Künstliche Neuronale Netze

Was ist Reinforcement Learning?

- Lernen aus Interaktion
- Ziel-orientiertes Lernen
- Lernen durch, von, und während der Interaktion mit einer externen Umgebung
- Lernen "was zu tun ist" — wie man Situationen auf Aktionen abbildet — um ein numerisches Reward-Signal zu maximieren

Überwachtes Lernen

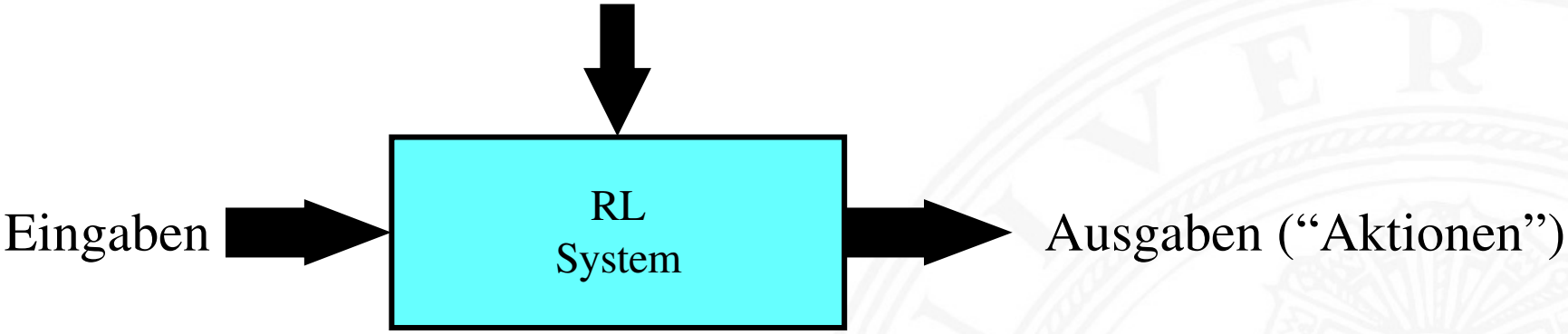
Trainings Info = gewünschte (Soll-) Ausgabe



Fehler = (Soll-Ausgabe – Systemausgabe)

Reinforcement Learning

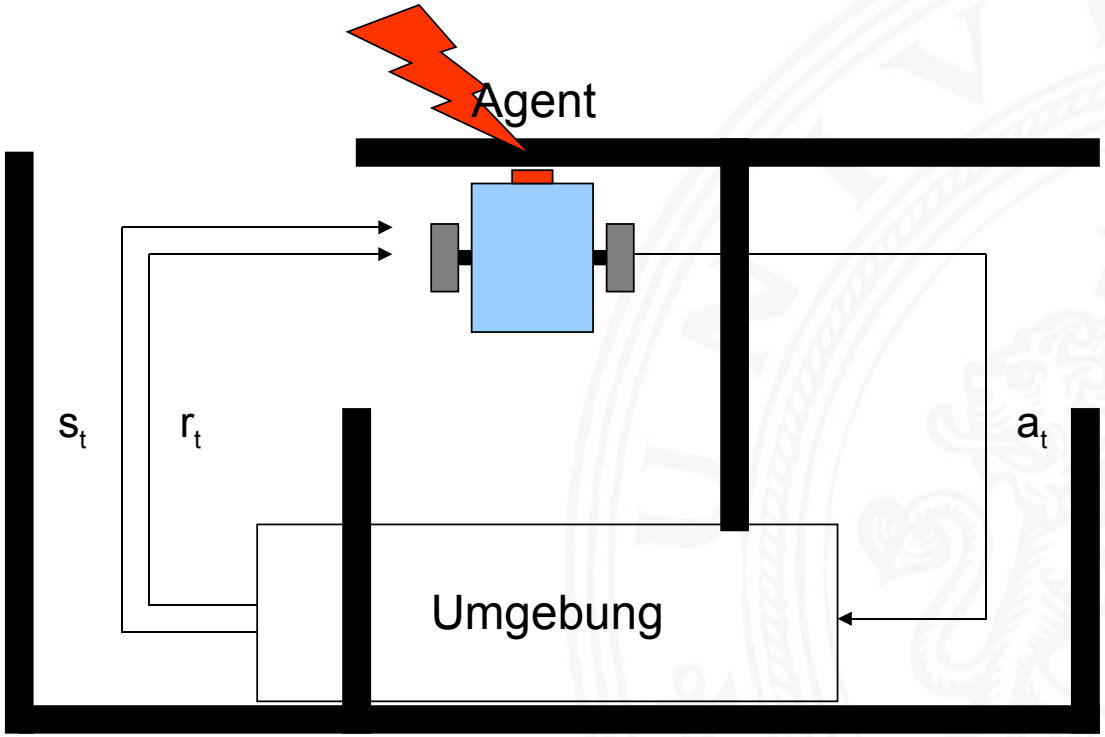
Trainings Info = Bewertungen ("rewards" / "penalties")



Ziel: erreiche soviel *Reward* wie möglich

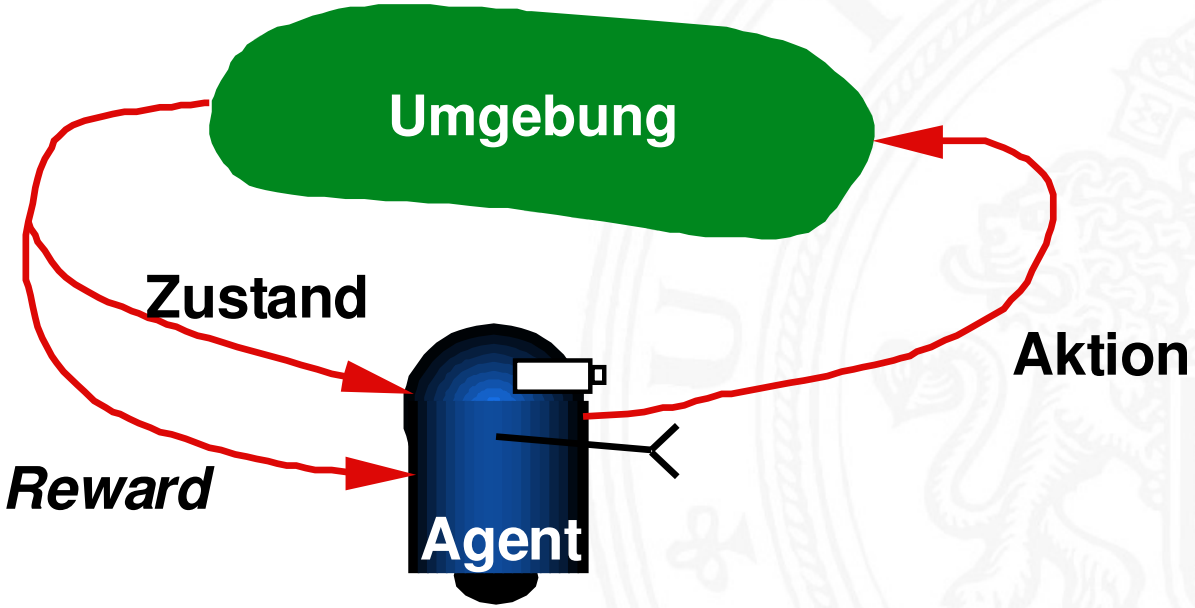
Reinforcement Learning

- Ziel: Möglichst „erfolgreich“ in der Umgebung agieren
- Entspricht Maximierung der Belohnungssequenz R_t

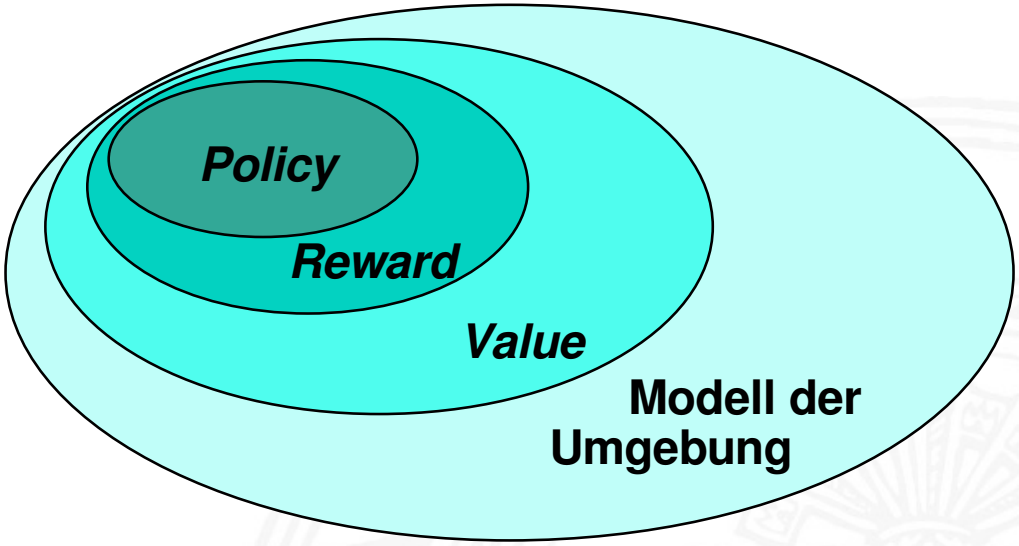


Der vollständige Agent

- Zeitlich situiert
- Beständiges Lernen und Planen
- Beeinflusst die Umgebung
- Umgebung ist stochastisch und ungewiss

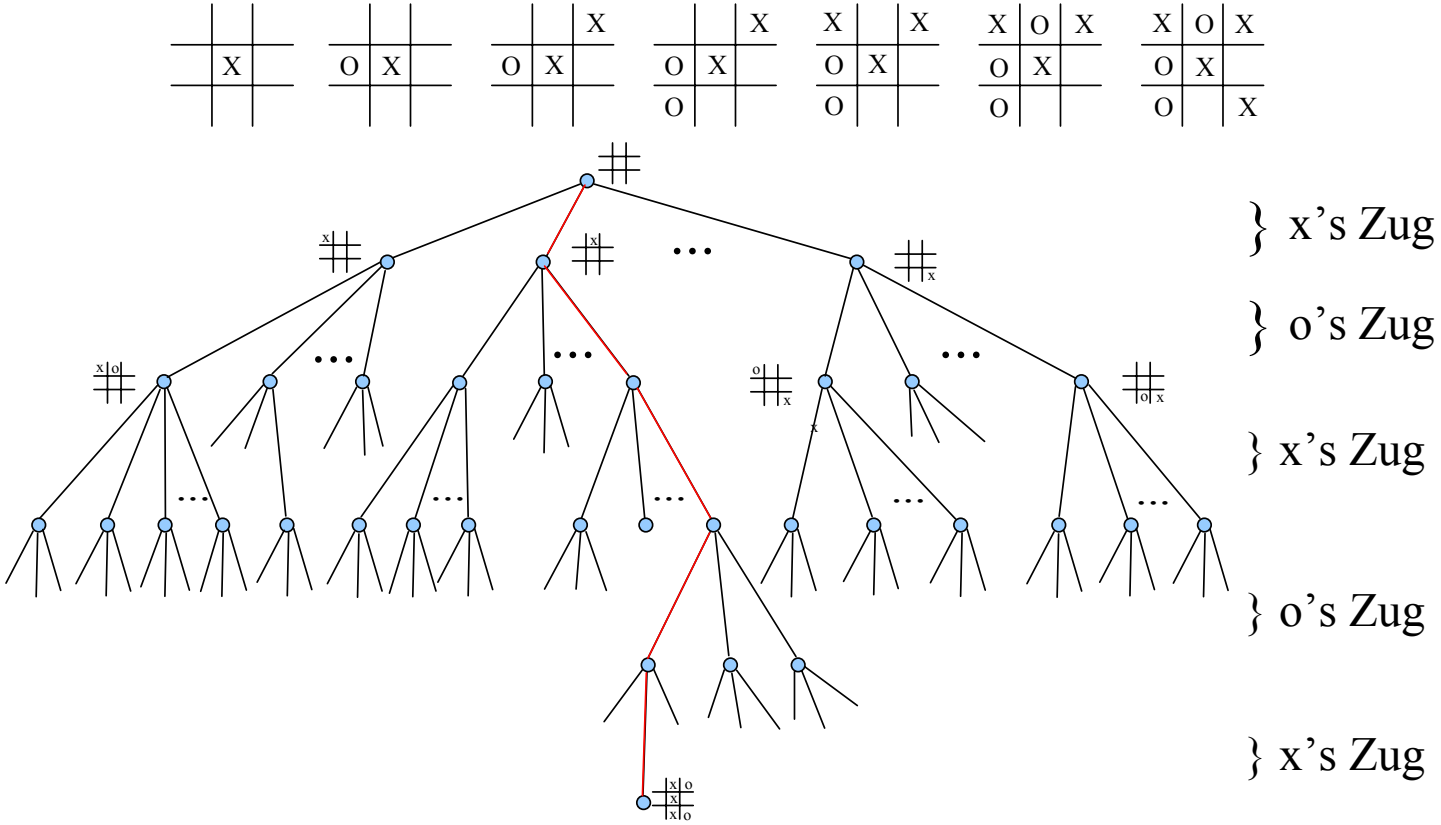


Elemente des RL



- **Policy:** was ist zu tun
- **Reward:** was ist gut
- **Value:** was ist gut, da es Reward vorhersagt
- **Modell:** was folgt auf was

Ein erweitertes Beispiel: TicTacToe



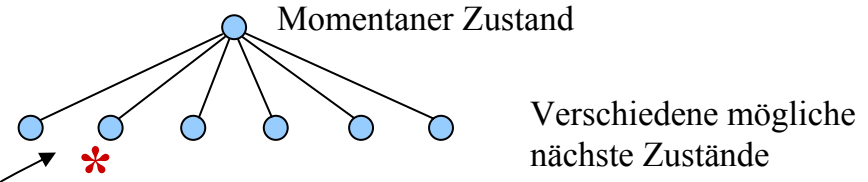
Setzt einen nicht perfekten Gegner voraus: er/sie macht Fehler

Ein RL Ansatz für TicTacToe

1. Erstelle eine Tabelle mit einem Eintrag pro Zustand:

Zustand	$V(s)$ – geschätzte Wahrscheinlichkeit für den Gewinn	
#	.5	
x #	.5	
⋮	⋮	
x x x / o # o	1	gewonnen
⋮	⋮	
x o / x o / o	0	verloren
⋮	⋮	
o x o / o x x / x o o	0	unentschieden

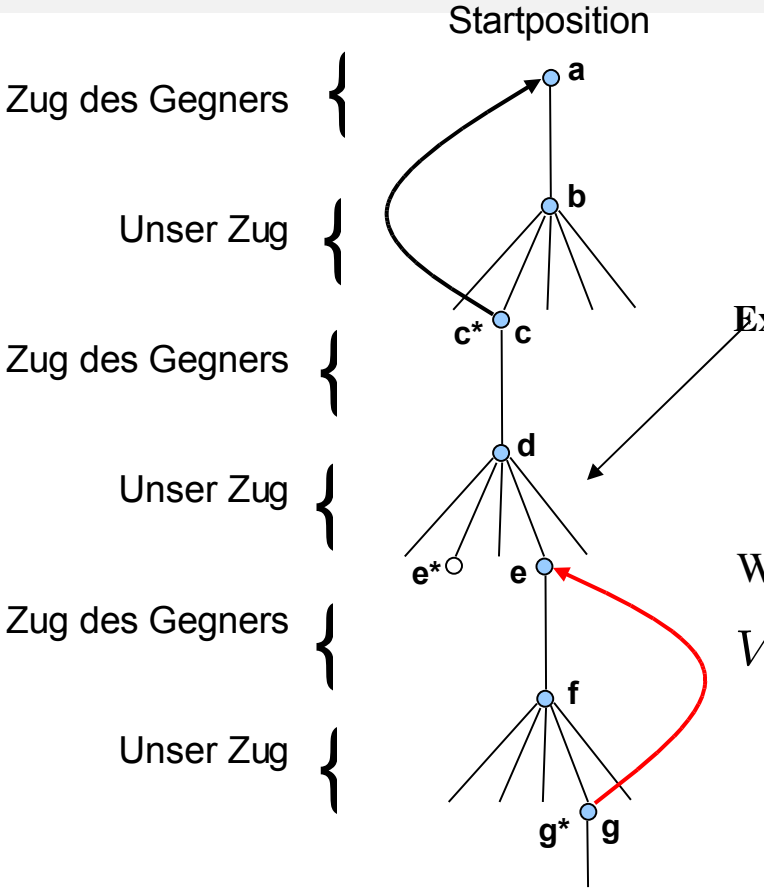
2. Jetzt spiele viele Spiele.
   Um einen Zug zu wählen,
   schaue einen Schritt nach vorne:



Nehme den nächsten Zustand mit der höchsten geschätzten Gewinnwahrscheinlichkeit — das höchste $V(s)$; ein ***greedy*** Zug.

Aber in 10% aller Fälle wähle einen zufälligen Zug; ein ***explorierender*** Zug.

RL-Lernregel für Tic-Tac-Toe



Explorierender Zug

s – Zustand vor dem *greedy* Zug

s' – Zustand nach dem *greedy* Zug

Wir inkrementieren jedes $V(s)$ zu $V(s')$ – ein „***backup***“

$$V(s) \leftarrow V(s) + \alpha \left[V(s') - V(s) \right]$$

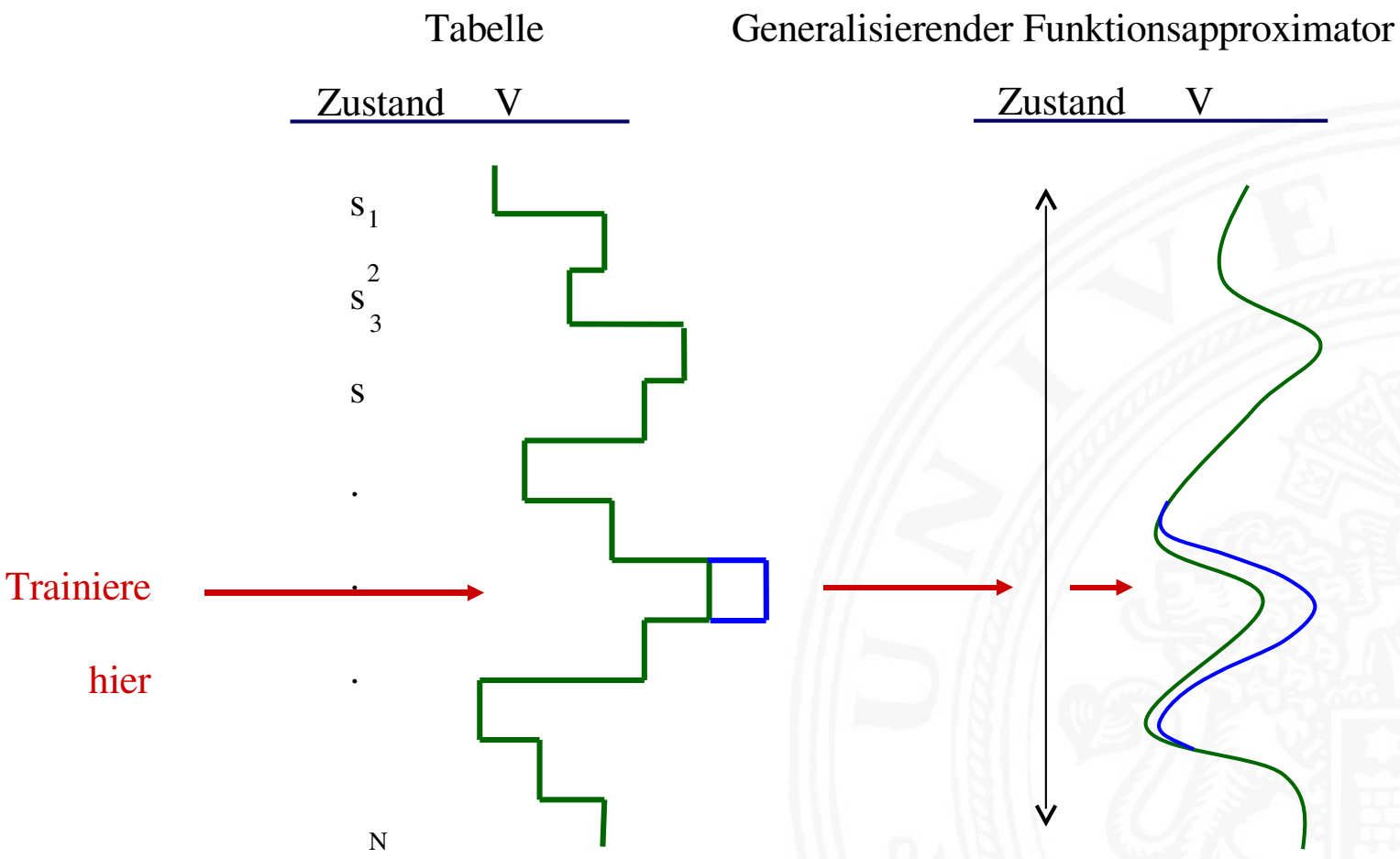
kleiner positiver Wert, z.B. $\alpha = 0.1$

der **„Schrittweitenparameter“**

Verbesserung des Tic Tac Toe Spielers

- Beachten von Symmetrien
  - Darstellung/Generalisierung
  - Wie kann dies fehlschlagen?
- Braucht man "Zufallszüge"? Warum?
  - Braucht man immer die 10 %?
- Kann man von "Zufallszügen" lernen?
- Kann man offline lernen?
  - Vor-Lernen durch Spielen gegen sich selbst?
  - Verwendung von gelernten Modellen des Gegners?
- ...

z.B. Generalisierung



Warum ist TicTacToe einfach?

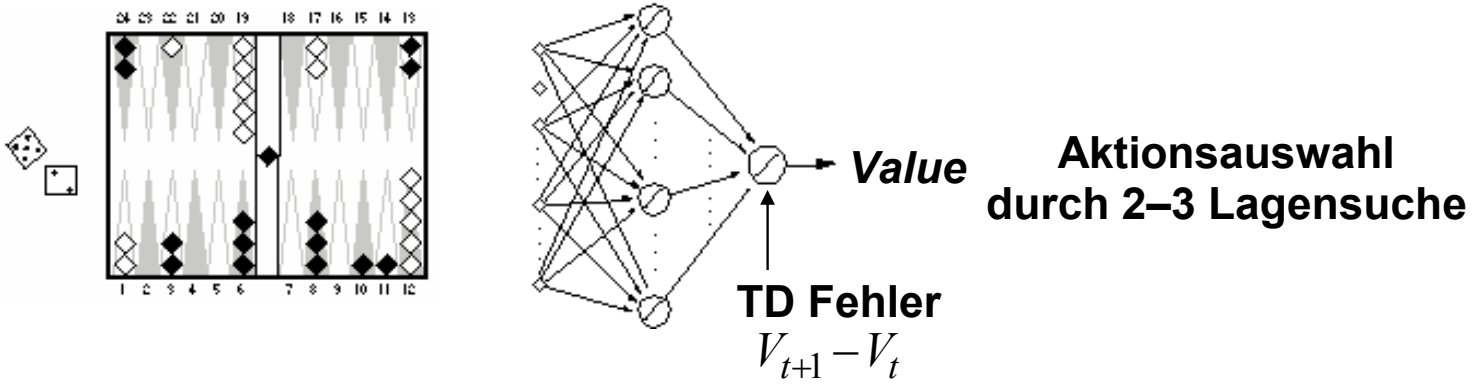
- Endliche, kleine Anzahl an Zuständen
- Es ist immer möglich einen Schritt nach vorne zu gucken (one-step look ahead)
- Zustände komplett wahrnehmbar
- ...

Einige namhafte RL Anwendungen

- **TD-Gammon:** Tesauro
  - weltbestes Backgammon Programm
- **Aufzugssteuerung:** Crites & Barto
  - High Performance "down-peak" Aufzugscontroller
- **Lagerverwaltung:** Van Roy, Bertsekas, Lee & Tsitsiklis
  - 10–15 % Verbesserung gegenüber standard Industriemethoden
- **Dynamische Kanalzuordnung:** Singh & Bertsekas, Nie & Haykin
  - High Performance Zuordnung von Funkkanälen zu Mobiltelefonaten

TD-Gammon

Tesauro, 1992–1995



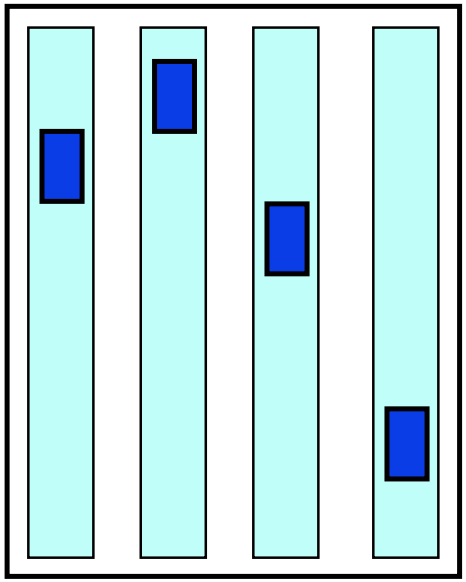
- Starte mit zufälligem Netzwerk
- Spiele sehr viele Spiele gegen dich selbst
- Lerne eine Wertefunktion anhand dieser simulierten Erfahrung

Dies produziert wohl den besten Spieler der Welt

Aufzugseinteilung

Crites and Barto, 1996,

10 Stockwerke, 4 Kabinen



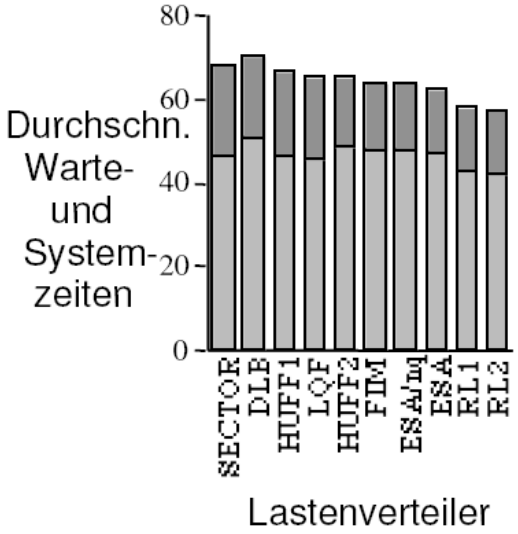
Zustände: Knopfzustände; Positionen, Richtungen, und Bewegungszustände der Kabinen; Personen in Kabinen & in Etagen

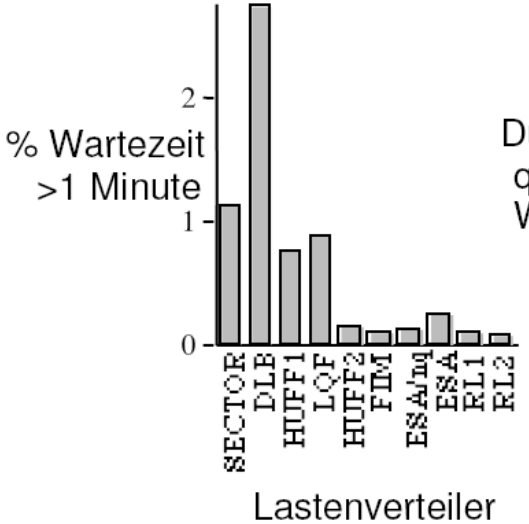
Aktionen: halte an X, oder fahre nach Y, nächste Etage

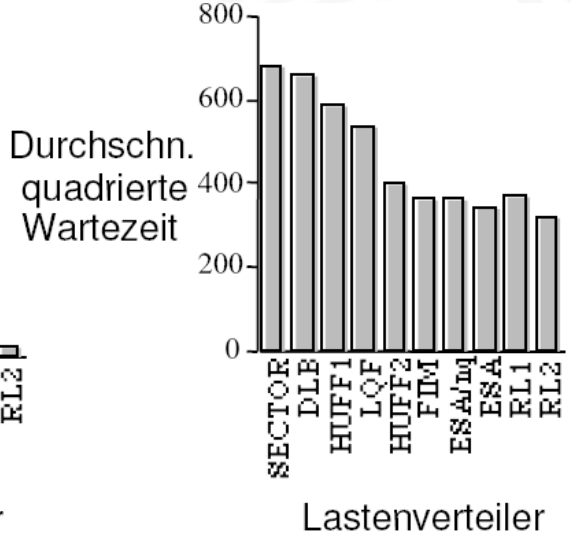
Rewards: geschätzt, –1 pro Zeitschritt für jede wartende Person

Vorsichtige Schätzung: ca. 10^{22} Zustände

Performance Vergleich





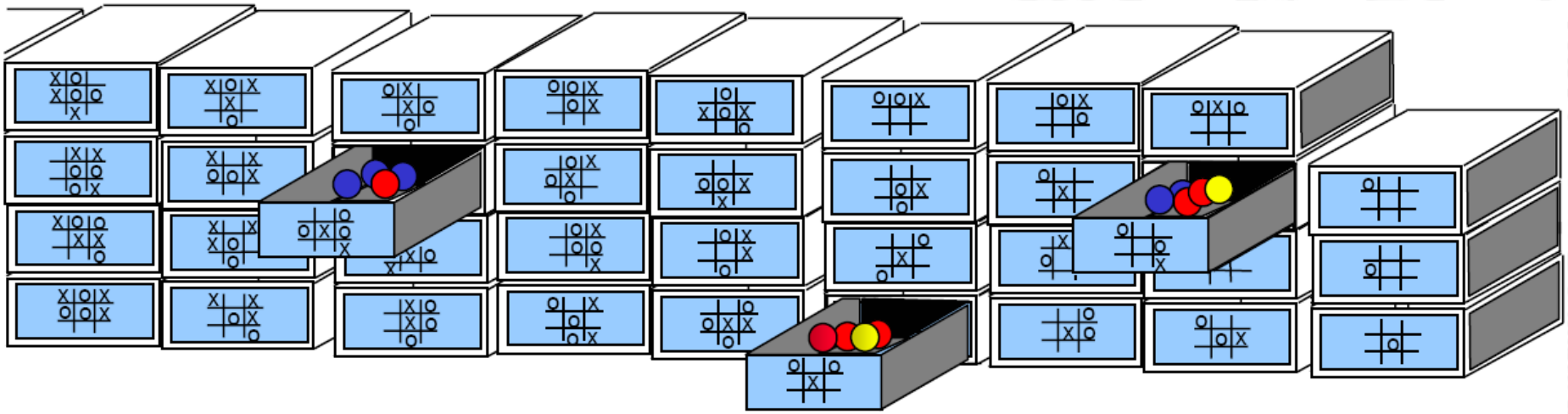


RL Geschichte

Trial-and-Error learning	Temporal-difference learning	Optimal control, value functions
Thorndike (Ψ) 1911	Secondary reinforcement (Ψ)	Hamilton (Physics) 1800s
	Samuel	Shannon
Minsky		Bellman/Howard (OR)
	Holland	
Klopf		
	Witten	
		Werbos
Barto et al.	Sutton	
		Watkins

MENACE (Michie 1961)

"Matchbox Educable Noughts and Crosses Engine"



Bewertendes Feedback

- Aktionen **bewerten** anstatt instruieren durch Vorgabe korrekter Aktionen
- Reines bewertendes Feedback hängt allein von der gewählten Aktion ab. Reines instruiertes Feedback hängt gar nicht von der gewählten Aktion ab.
- Überwachtes Lernen ist instruierend; Optimierung ist bewertend
- **Assoziativ** vs. **Nichtassoziativ**
  - Assoziativ: Eingaben werden auf Ausgaben abgebildet; lerne die beste Ausgabe **für jede** Eingabe
  - Nichtassoziativ: "lerne" (finde) eine beste Ausgabe
- n-armiger Bandit (zumindest wie wir ihn betrachten):
  - Nichtassoziativ
  - Bewertendes Feedback

Der n-armige Bandit

- Wähle wiederholt eine aus n Aktionen; jede Auswahl wird **Spiel** genannt
- Nach jedem Spiel a_t erhält man einen Reward r_t, wobei

$$E\left\langle r_t | a_t \right\rangle = Q^*(a_t)$$

  Dies sind unbekannte **Aktionsbewertungen**
  Verteilung von r_t hängt nur von a_t ab
- Ziel ist es den Reward auf lange Sicht zu maximieren, z.B. über 1000 Spiele

Um die Aufgabe des n-armigen Banditen zu lösen,
muss man eine Reihe von Aktionen **erkunden (explore)**
und die besten von ihnen ausnutzen (exploit).

Das Exploration/Exploitation Problem

- Angenommen man definiert Schätzungen
  $Q_t(a) \approx Q^*(a)$ **Schätzungen für Aktionsbewertungen**
- Die *greedy* Aktion zum Zeitpunkt t ist

$$
\begin{aligned}
a_t^* &= \arg\max_a Q_t(a) \\
a_t &= a_t^* \Rightarrow \textit{exploitation} \\
a_t &\neq a_t^* \Rightarrow \textit{exploration}
\end{aligned}
$$

- Man kann nicht immer erkunden, aber auch nicht immer ausnutzen
- Man kann das Erkunden nie stoppen, sollte es aber reduzieren

Action – Value Methoden

- Methoden die ausschließlich Schätzungen für Aktionsbewertungen betrachten. Angenommen im t-ten Spiel wurde Action a k_a-mal gewählt, was die *Rewards* $r_1, r_2, \ldots, r_a$ produziert hat, dann

$$Q_t(a) = \frac{r_1 + r_2 + \cdots + r_{k_a}}{k_a}$$

"Mittlerer *Reward*"

- $$\lim_{k_a \to \infty} Q_t(a) = Q^*(a)$$

ϵ-greedy Aktionsauswahl

- *greedy* Aktionsauswahl

$$a_t = a_t^* = \arg\max_a Q_t(a)$$

- ϵ-greedy Aktionsauswahl:

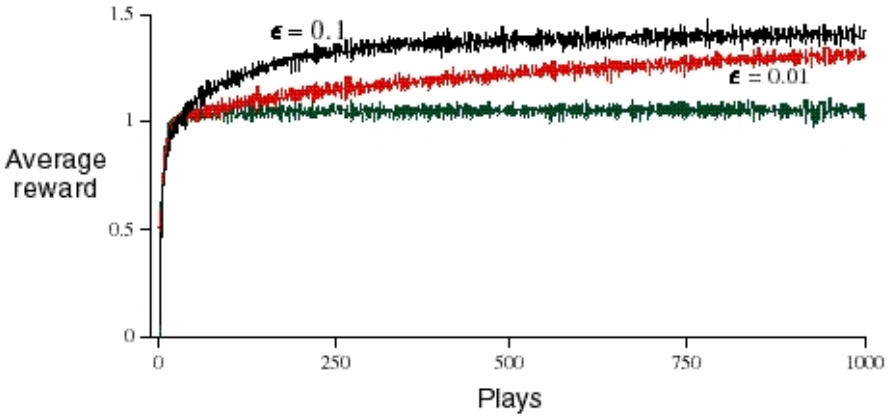
$$a_t = \begin{cases} a_t^* & \text{mit Wahrschinlichkeit} \quad 1-\epsilon \\ & \text{Zufallsaktion mit Wahrscheinlichkeit} \quad \epsilon \end{cases}$$

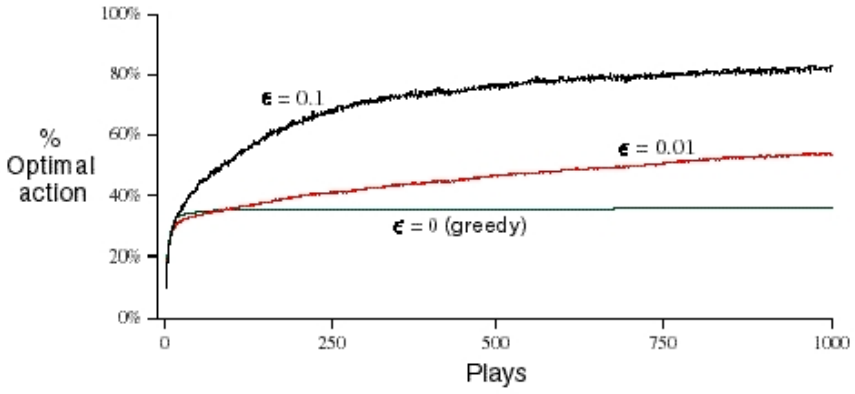
...der einfachste Weg um *exploration* und *exploitation* zu behandeln.

10-armige Testumgebung

- $n = 10$ mögliche Aktionen
- Jedes $Q^*(a)$ wird zufällig aus einer Normalverteilung gewählt: $\eta(0, 1)$
- Jedes r_t ist auch normalverteilt: $\eta(Q^*(a_t), 1)$
- 1000 Spiele
- Wiederhole alles 2000 mal und mittel die Ergebnisse

ϵ-greedy Methode für die 10 armige Testumgebung





Softmax Aktionsauswahl

- *Softmax* Aktionsauswahl-Methode staffelt Aktionswahrscheinlichkeiten nach geschätzten Werten.
- Die gebräuchlichste *softmax* -Methode benutzt eine Gibbs-, oder Boltzmann-Verteilung:
  Wähle Aktion a im Spiel t mit Wahrscheinlichkeit

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}},$$

wobei τ die "Temperatur" ist.

Binäre Banditenaufgabe

Angenommen es gibt nur **zwei** Aktionen: $a_t = 1$ oder $a_t = 2$
und nur **zwei** *Rewards* : $r_t = \mathit{Erfolg}$ oder $r_t = \mathit{Fehler}$

Dann könnte man ein **Ziel** oder eine **Soll-Aktion** definieren:

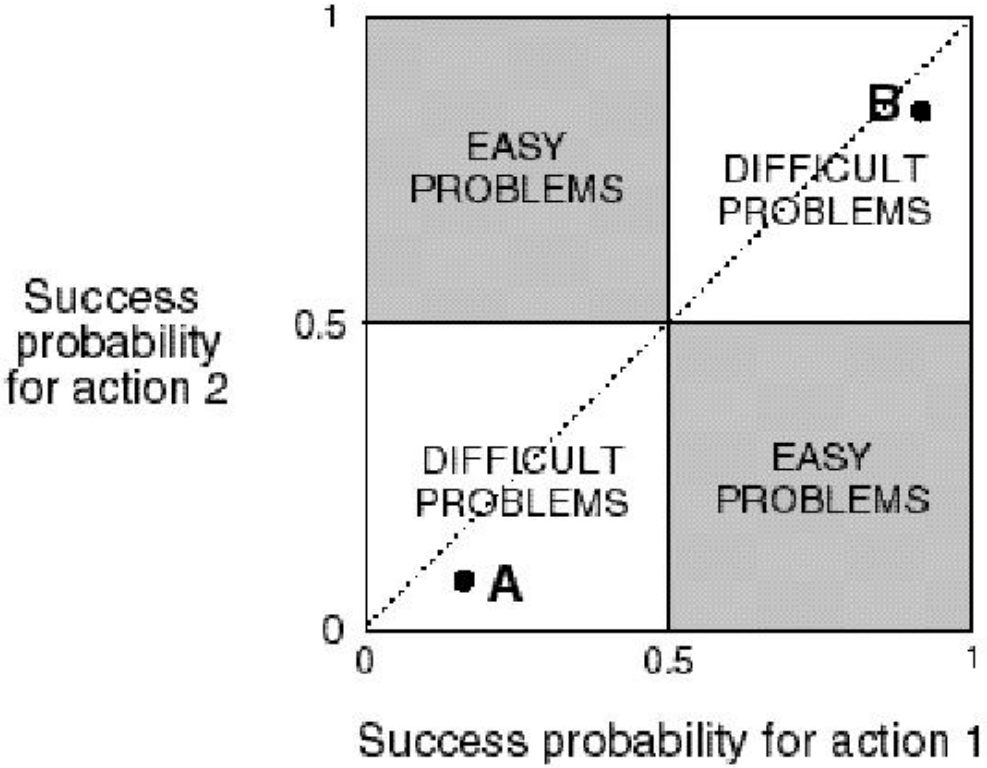
$$d_t = \begin{cases} a_t & \text{wenn} \quad \mathit{Erfolg} \\ \text{Die andere Aktion} & \text{wenn} \quad \mathit{Fehler} \end{cases}$$

und immer die Aktion spielen, die am öftesten das Ziel war

Dies ist ein **überwachter Algorithmus**
Er funktioniert gut bei deterministischen Problemen...

Zufallsraum

Der Raum aller möglichen binären Banditenaufgaben:



Linear Learning Automata

Sei $\pi_t(a) = Pr\{a_1 = a\}$ der einzige zu adaptierende Parameter

L_{R-I} (Linear, reward -inaction):

Bei **Erfolg:** $\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha(1 - \pi_t(a_t)) \quad 0 < \alpha < 1$

Bei **Misserfolg:** keine Änderung

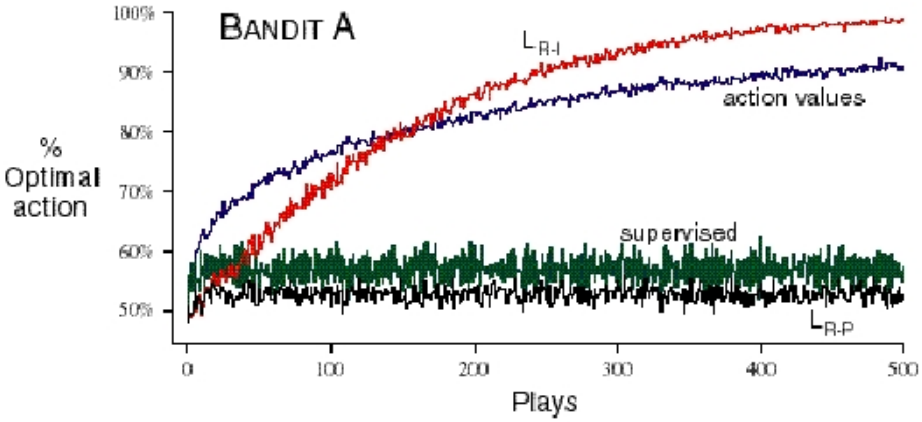
L_{R-P} (Linear, reward -penalty):

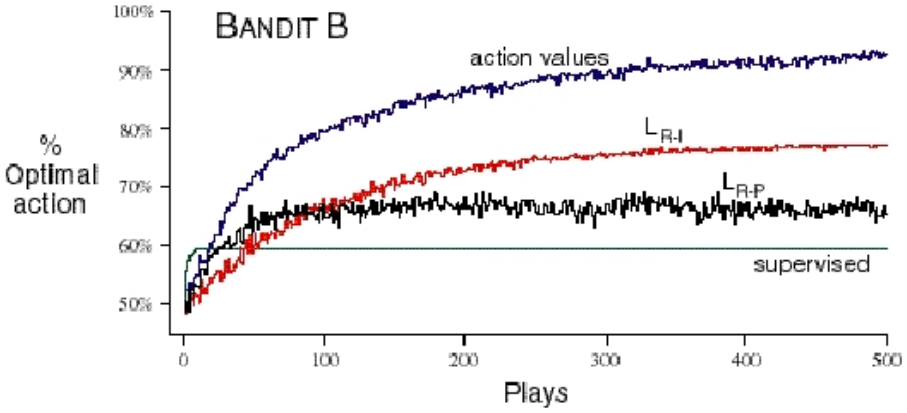
Bei **Erfolg:** $\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha(1 - \pi_t(a_t)) \quad 0 < \alpha < 1$

Bei **Misserfolg:** $\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha(0 - \pi_t(a_t)) \quad 0 < \alpha < 1$

- nach jeder Aktualisierung folgt Anpassung der übrigen Aktionswahrscheinlichkeiten, so dass sie sich zu 1 summieren

Performanz der binären Banditenaufgaben A und B





Inkrementelle Implementierung

Erinnern wir uns an die Bewertungsmethode des mittleren *Rewards*:

Der Durchschnitt der ersten k *Rewards* ist (bei Vernachlässigung der Abhängigkeit von a):

$$Q_k = \frac{r_1 + r_2 + \cdots + r_k}{k}$$

Kann dies inkrementell aufgebaut werden (ohne alle *Rewards* zu speichern)?
Wir könnten eine laufende Summe und einen Zähler benutzen,

oder equivalent:

$$Q_{k+1} = Q_k + \frac{1}{k+1}\left[r_{k+1} - Q_k\right]$$

Dies ist eine übliche Form für *update*-Regeln:

NeueSchätzung = *AlteSchätzung* + *Schrittweite [Ziel - AlteSchätzung]*

Nicht-stationäre Probleme

Q_k als *Reward*-Durchschnitt zu nehmen ist angemessen für ein stationäres Problem,
d.h., wenn sich kein $Q^*(a)$ über die Zeit ändert.

Aber nicht für ein nicht-stationäres Problem.

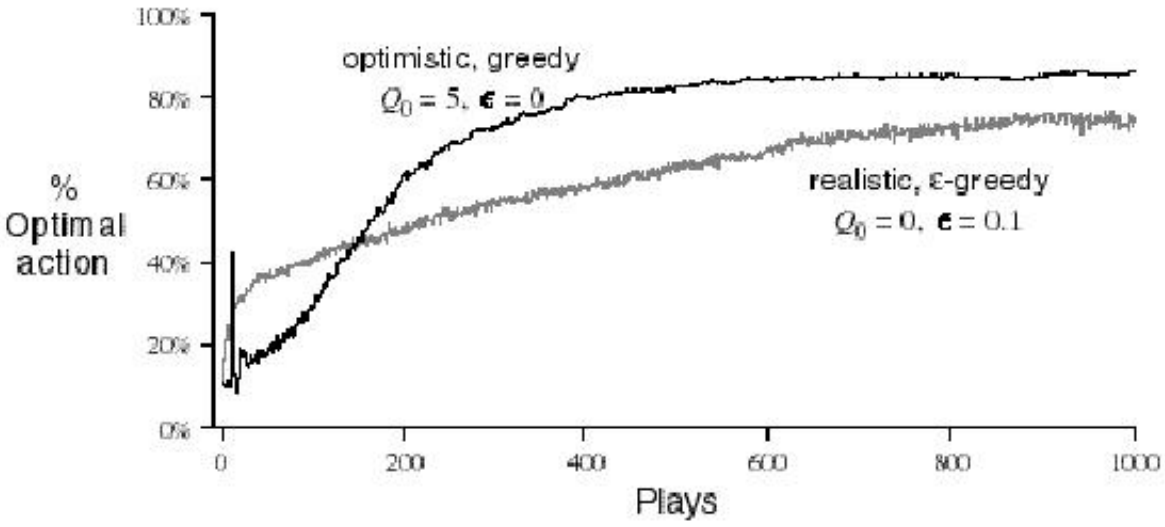
Besser in dem nicht-stationären Fall ist:

$$\begin{aligned} Q_{k+1} &= Q_k + \alpha\left[r_{k+1} - Q_k\right] \quad \textit{fuer konstantes } \alpha, 0 < \alpha \leq 1 \\ &= (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha(1-\alpha)^{k-i} r_i \end{aligned}$$

exponential, recency-weighted average

Optimistische Initialwerte

- Alle bisherigen Methoden hängen von $Q_0(a)$ ab , d.h., sie sind ***biased***.
- Angenommen wir initialisieren die Aktionswerte **optimistisch**, z.B. in der 10-armigen Testumgebung: $Q_0(a) = 5$ für alle a



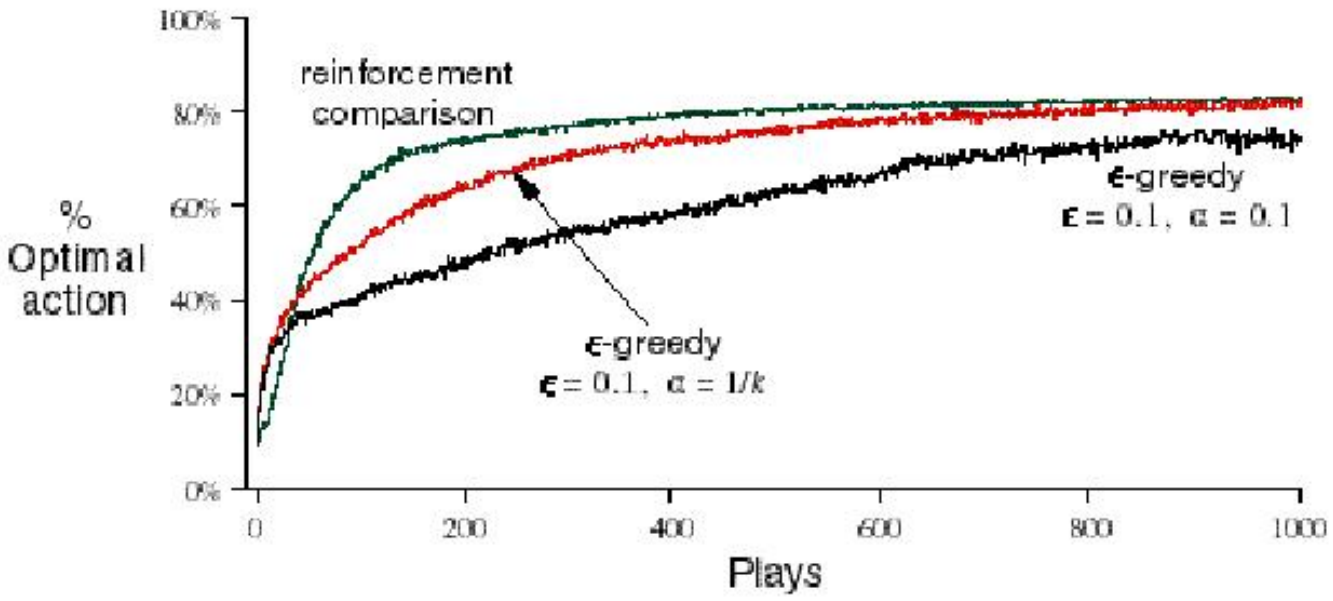
Reinforcement-Vergleich

- Vergleiche Rewards mit einem Referenz - *Reward* $\bar{r}_t$, z.B. ein *Reward*-Durchschnitt aller erhaltenen *Rewards*
- Verstärke oder schwäche die gewählte Aktion in Abhängigkeit von $r_t - \bar{r}_t$ ab
- Sei $p_t(a)$ die **Präferenz** für Aktion a
- Präferenzen legen Aktions-Wahrscheinlichkeiten fest, z.B., durch die Gibbs Verteilung:

$$\pi_t(a) = Pr\{a_t = a\} = \frac{e^{p_t(a)}}{\sum_{b=1}^{n} e^{p_t(b)}}$$

- Dann: $p_{t+1}(a_t) = p_t(a) + \beta\,[r_t - \bar{r}_t]$ *und* $\bar{r}_{t+1} = \bar{r}_t + \alpha\,[r_t - \bar{r}_t]$

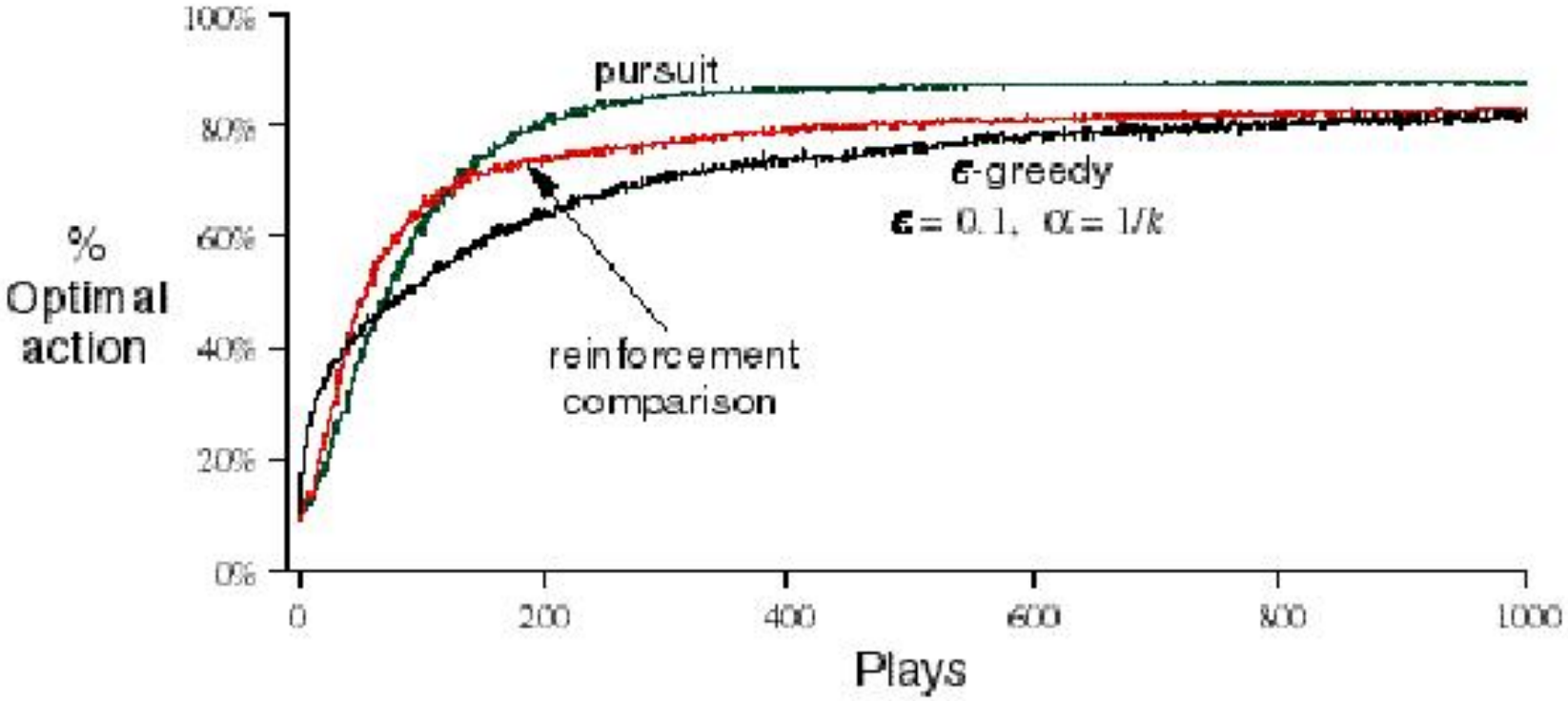
Performanz einer Reinforcement-Vergleich Methode



Pursuit Methoden

- Behandeln sowohl Schätzungen von Aktionswerten sowie Aktions-Präferenzen
- "Verfolge" immer die *greedy* -Aktion, d.h. mache die *greedy* -Aktion wahrscheinlicher für die Aktionsauswahl
- Aktualisiere nach dem t -ten Spiel die Aktionswerte um Q_{t+1} zu erhalten
- Die neue greedy -Aktion ist $a^*_{t+1} = \arg\max_a Q_{t+1}(a)$
- Dann: $\pi_{t+1}(a^*_{t+1}) = \pi_t(a^*_{t+1}) + \beta\left[1 - \pi_t(a^*_{t+1})\right]$ und die Wahrscheinlichkeiten der anderen Aktionen werden reduziert um die Summe von 1 zu erhalten.

Performanz einer Pursuit Methode



Schlussfolgerungen

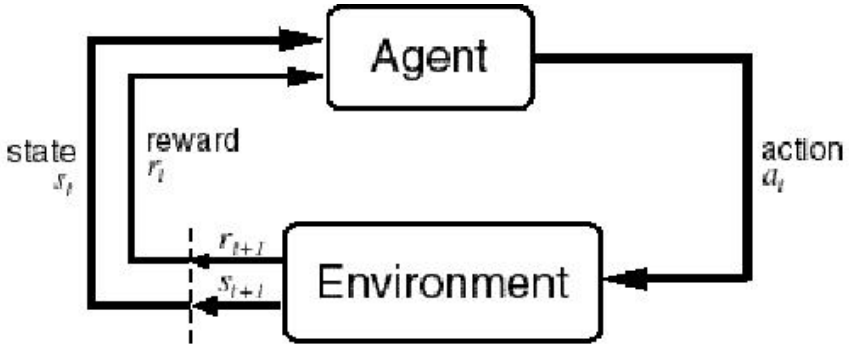
- Dies sind alles sehr einfache Methoden
  - aber sie sind kompliziert genug - wir werden auf ihnen aufbauen
  - Ideen für Verbesserungen:
  - Abschätzung von Unsicherheiten . . . Intervall-Abschätzung
  - Approximierung von *Bayes optimal solutions*
  - Gittens indices (klassische Lösung für n-armige Banditen zur Abwägung von *Exploration* und *Exploitation*)
- Das komplette RL Problem bietet einige Lösungsideen. . .

Das Reinforcement-Lernproblem

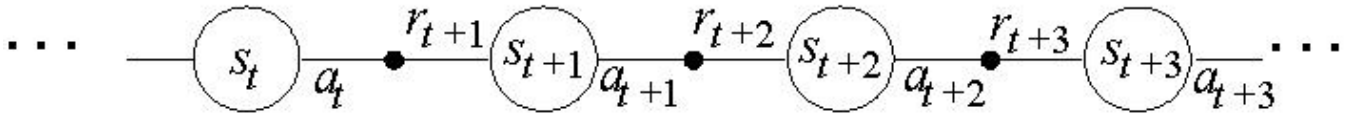
Beschreibung des RL-Problems:

- Präsentation einer idealisierten Form des RL-Problems für die es konkrete theoretische Betrachtungen gibt
- Einführung der wichtigen mathematischen Komponenten: Wertefunktionen und Bellman-Gleichungen
- Beschreibung des trade-offs zwischen Anwendbarkeit und mathematischer Lenkbarkeit

Der Agent in einer Lernumgebung



Agent und Umgebung interagieren zu diskreten Zeitpunkten:	$t = 0,1,2\ldots K$
Agent beobachtet Zustand zum Zeitpunkt t:	$s_t \in S$
erzeugt Aktion zum Zeitpunkt t:	$a_t \in A(s_t)$
erhält *Reward*:	$r_{t+1} \in \mathcal{R}$
und nächsten Zustand:	s_{t+1}



Der Agent lernt eine *Policy*

Policy zum Zeitpunkt t, π_t :

Abbildung von Zuständen auf Aktionswahrscheinlichkeiten
$\pi_t(s, a) =$ Wahrsch., dass $a_t = a$ wenn $s_t = s$

- Reinforcement Lernmethoden beschreiben wie ein Agent seine *Policy* als Resultat seiner Erfahrung ändert.
- Grob gesagt ist das Ziel des Agenten, auf lange Sicht soviel *Reward* wie möglich zu erhalten.

Richtiger Abstraktionsgrad

- Zeitschritte müssen keine festen Intervalle der wirklichen Zeit sein.
- Aktionen können *lowlevel* (z.B., Spannungen von Motoren), oder *highlevel* sein (z.B., nimm ein Jobangebot an), "mental" (z.B., shift in focus of attention), etc.
- Zustände können *lowlevel* "Empfindungen", oder abstrakt sein, symbolisch, basierend auf einem Gedächtnis, oder subjektiv (z.B., der Zustand "überrascht zu sein").
- Ein RL Agent ist nicht wie ein ganzes Tier oder ein kompletter Roboter, welche aus mehreren RL Agenten und anderen Komponenten bestehen.
- Die Umgebung ist dem Agenten nicht notwendigerweise unbekannt, nur unvollständig kontrollierbar.
- *Reward*-Berechnung passiert in der Umgebung, da der Agent sie nicht beliebig ändern kann.

Ziele und *Rewards*

- Ist ein skalares *Reward*-Signal eine adäquate Beschreibung für ein Ziel? – vielleicht nicht, aber es ist erstaunlich flexibel.
- Ein Ziel sollte beschreiben **was** wir erreichen wollen und nicht **wie** wir es erreichen wollen.
- Ein Ziel muss außerhalb der Kontrolle des Agenten liegen – also außerhalb des Agenten selbst.
- Der Agent muss Erfolge messen können:
  - explizit;
  - häufig während seiner Lebensdauer.

Returns

Angenommen die Sequenz von Rewards nach Zeitpunkt t ist:

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

Was wollen wir maximieren?

Generell wollen wir den **erwarteten *Return***, $E\{R_t\}$, in jedem Zeitschritt t maximieren.

Episodische Task : Interaktion teilt sich naturgemäß in Episoden auf, z.B. Durchgänge eines Spiels, Durchläufe durch ein Labyrinth.

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

wobei T ein finaler Zeitpunkt ist zu dem ein Zielzustand erreicht und die Episode beendet wird.

Returns für kontinuierliche Tasks

Kontinuierliche Task: Interaktion hat keine natürlichen Episoden.

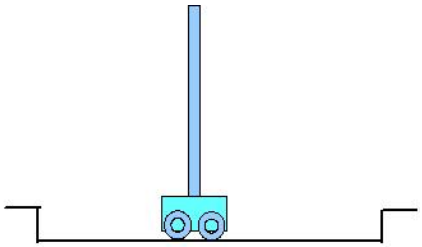
Discounted *Return* :

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

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

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

Ein Beispiel



Misserfolg vermeiden: der Stab fällt über einen kritischen Winkel oder der Wagen erreicht das Ende der Strecke

Als **episodische task** wo die Episoden bei Misserfolg enden:

Reward	$=$	$+1$ für jeden Schritt vor dem Misserfolg
$\Rightarrow$ *Return*	$=$	Anzahl der Schritte bis zum Misserfolg

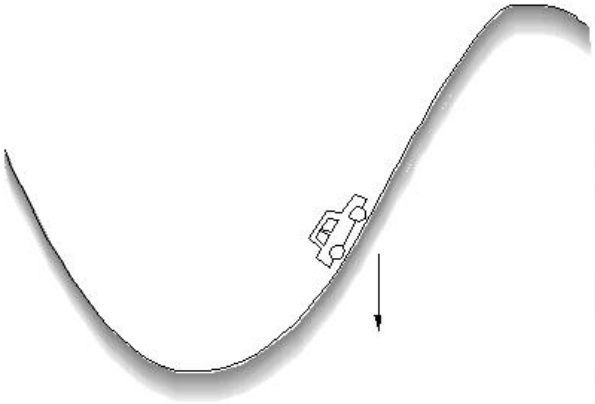
Als **kontinuierliche task** mit *discounted Return*:

Reward	$=$	-1 bei Misserfolg; 0 sonst
$\Rightarrow$ *Return*	$=$	$-\gamma^k$, für k Schritte vor dem Misserfolg

In beiden Fällen wird der Return maximiert, indem Misserfolg so lange wie möglich vermieden wird.

Ein weiteres Beispiel

Fahre so schnell wie möglich auf die Spitze des Berges.

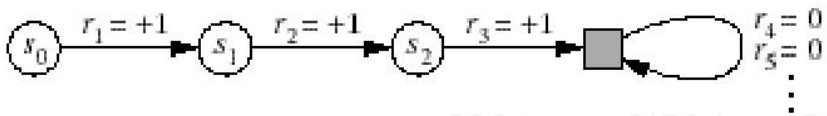


$$\begin{aligned} \textit{Reward} &= -1 \text{ für jeden Schritt an dem man } \textbf{nicht} \text{ auf der Spitze des Berges ist} \\ \textit{Return} &= -\text{Anzahl der Schritte vor Erreichen der Hügelspitze} \end{aligned}$$

Der *Return* wird maximiert indem die Anzahl der Schritte zum Erreichen der Bergspitze minimiert wird.

Vereinheitlichte Notation

- In episodischen Tasks nummerieren wir die Zeitschritte jeder Episode beginnend mit Null.
- Im allgeimeinen differenzieren wir nicht zwischen Episoden. Wir schreiben s_t anstatt $s_{t,j}$ für den Zustand zum Zeitpunkt t in Episode j.
- Betrachte das Ende jeder Episode als einen absorbierenden Zustand der immer einen **Reward** von 0 hat:



- Wir fassen alle Fälle zusammen:

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

wobei γ nur 1 sein kann wenn immer ein absorbierender Zustand erreicht wird.

Die Markov Wahrscheinlichkeit

- Mit "dem Zustand" zum Zeitpunkt t meinen wir alle Informationen die dem Agenten zum Zeitpunkt t über seine Umgebung zur Verfügung stehen.
- Der Zustand kann sofortige "Wahrnehmungen", weiter verarbeitete Wahrnehmungen und Strukturen, die über eine Sequenz von Wahrnehmungen aufgebaut wurden, beinhalten.
- Idealerweise sollte ein Zustand vergangene Wahrnehmungen zusammenfassen, um alle "wichtigen" Informationen zu erhalten, das heißt er sollte die **Markov Wahrscheinlichkeit** besitzen:

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

Für alle s', r,und *histories* $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

Markov Entscheidungsprozesse

- Wenn eine Reinforcement-Lernaufgabe die Markov Wahrscheinlichkeit besitzt ist dies hauptsächlich ein **Markov Entscheidungsprozess (MDP)**.
- Wenn Zustands- und Aktionsräume endlich sind ist es ein endlicher MDP.
- Um einen endlichen MDP zu definieren braucht man:
  - **Zustands- und Aktionsräume**
  - one-step-"Dynamik"definiert über **Übergangswahrscheinlichkeiten:**

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

  - ***Reward* Wahrscheinlichkeiten**:

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

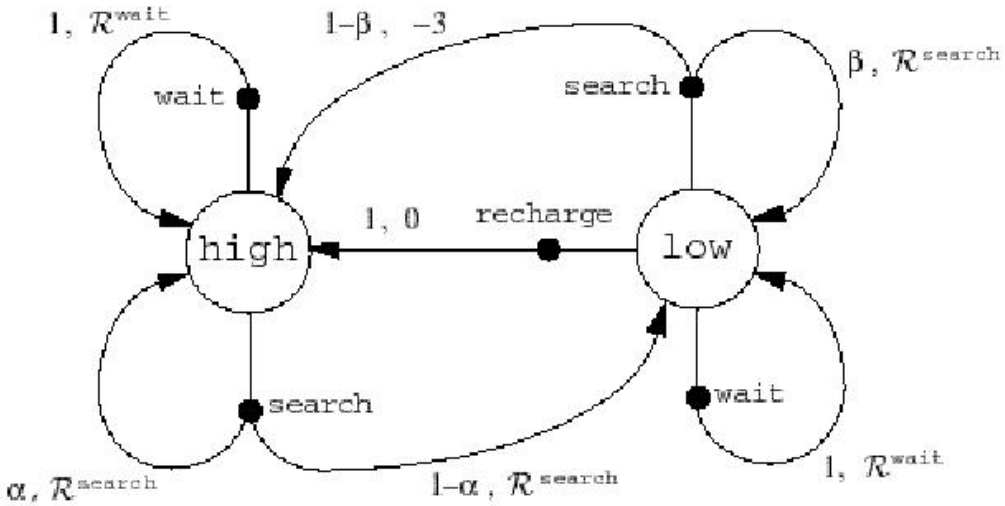
Ein Beispiel endlicher MDP

Recycling-Roboter

- Zu jedem Zeitpunkt muss der Roboter entscheiden, ob er (1) aktiv eine Dose suchen soll, (2) auf jemanden warten soll, der ihm eine Dose bringt, oder (3) zum Aufladen zur Basisstation fahren soll
- Suchen ist besser, verbraucht allerdings die Batterien; wenn die Batterien leer laufen während er sucht, muss er "befreit" werden (schlecht)
- Entscheidungen werden auf Grundlage des momentanen Energielevels getroffen: `high, low`
- *Reward* = Anzahl an gesammelten Dosen

Recycling-Roboter MDP

$S = \{\texttt{high, low}\}$
$A\,(\texttt{high}) = \{\texttt{search, wait}\}$
$A\,(\texttt{low}) = \{\texttt{search, wait, recharge}\}$
$R^{\texttt{search}}$ = erwartete Anzahl Dosen während der Suche
$R^{\texttt{wait}}$ = erwartete Anzahl Dosen während des Wartens
$R^{\texttt{search}} > R^{\texttt{wait}}$



Wertefunktionen

- Der **Wert eines Zustandes** ist der erwartete *Return* beginnend mit diesem Zustand; hängt von der *Policy* des Agenten ab:

  Zustands-Wertefunktion für *Policy* π :

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

- Der **Wert wenn eine Aktion in einem Zustand unter der *Policy* π** gewählt wird ist der erwartete *Return* beginnend mit diesem Zustand, bei Wählen dieser Aktion und weiterem Verfolgen von π :

  Aktions-Wertefunktion für *Policy* π:

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

Bellman-Gleichung für *Policy* π

Grundlegende Idee:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots \\ &= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \dots \right) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

Somit:

$$\begin{aligned} V^\pi(s) &= E_\pi \left\{ R_t | s_t = s \right\} \\ &= E_\pi \left\{ r_{t+1} + \gamma V(s_{t+1}) | s_t = s \right\} \end{aligned}$$

Oder, ohne Erwartungswertoperator:

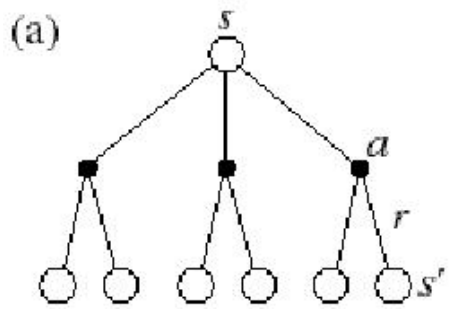
$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

Mehr zur Bellman-Gleichung

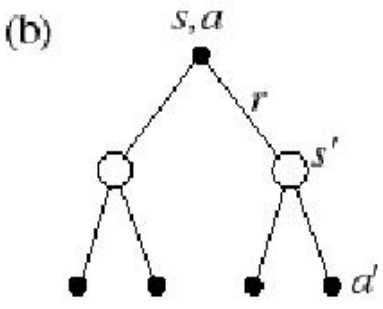
$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

Dies ist eine Menge von Gleichungen (linear), eine für jeden Zustand. Die Wertefunktion für π ist die eindeutige Lösung.

Backup-Diagramme :



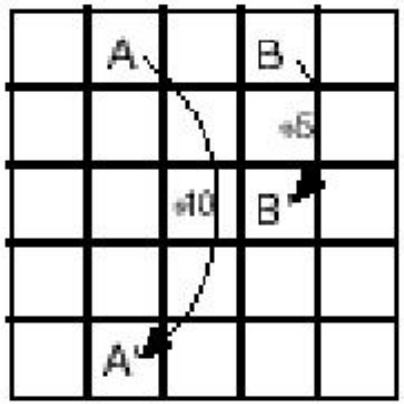
für V^π

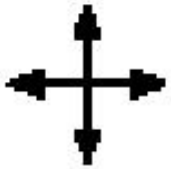


für Q^π

Gridworld

- Aktionen: norden , süden , osten , westen ; deterministisch.
- Wenn der Agent das Feld verlassen würde: kein Zug, aber *Reward* $= -1$
- Andere Aktionen geben *Reward* $= 0$, außer Aktionen die den Agenten aus einem bestimmten Zustand A und B bewegen.



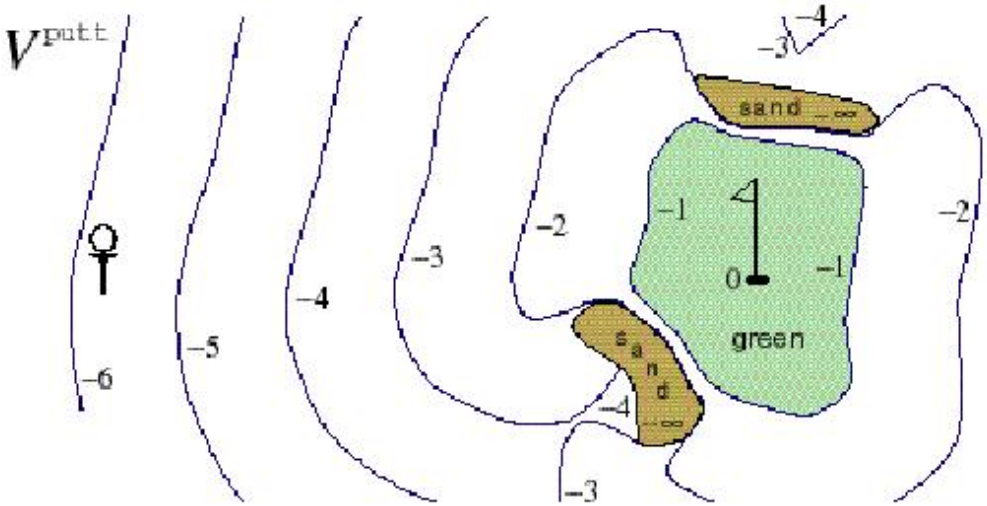


3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Zustands-Wertefunktion für gleichverteilte Zufalls-*Policy*; $\gamma = 0.9$

Golf

- Zustand ist die Ballposition
- Reward von -1 für jeden Schlag bis der Ball im Loch ist
- Wert eines Zustandes?
- Aktionen: `putt` (benutze Putter) `driver` (benutze Driver)
- `putt` führt auf dem Grün immer zum Erfolg



Optimale Wertefunktionen

- Für endliche MDPs, können die *Policies* **partiell geordnet** werden:

$$\pi \geq \pi' \quad \textit{genau wenn} \quad V^{\pi}(s) \geq V^{\pi'}(s)\ \forall s \in S$$

- Es gibt immer mindestens eine (vielleicht mehrere) *Policies* die besser als oder gleich allen anderen ist. Dies ist eine **optimale *Policy***. Wir bezeichnen sie alle mit π^*.
- Optimale *Policies* teilen sich die gleiche **optimale Zustands-Wertefunktion**:

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

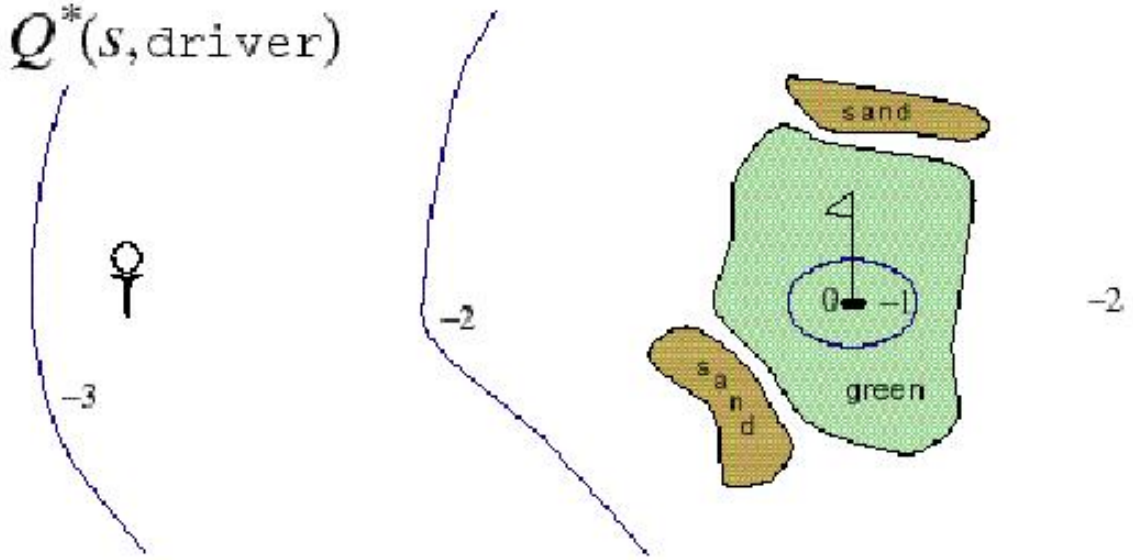
- Optimale *Policies* teilen sich auch die gleiche **optimale Aktions-Wertefunktion:**

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

Dies ist der erwartete *Return* nach Wahl der Aktion a in Zustand s und Weiterverfolgen einer optimalen *Policy*.

Optimale Wertefunktionen beim Golf

- Wir können den Ball mit `driver` weiter schlagen als mit `putter`, allerdings mit weniger Genauigkeit
- Q *(s,`driver`) gibt die Werte für die Auswahl von `driver` an, wenn nachfolgend immer die jeweils beste Aktion gewählt wird

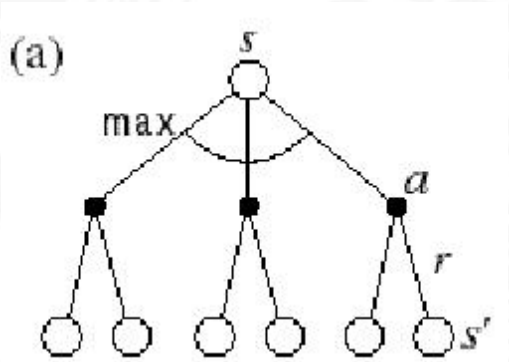


Optimale Bellman-Gleichung für V^*

Der Wert eines Zustandes unter einer optimalen *Policy* ist gleich dem erwarteten *Return* für die beste Aktion von diesem Zustand an.

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

Das entsprechende Backup-Diagramm:

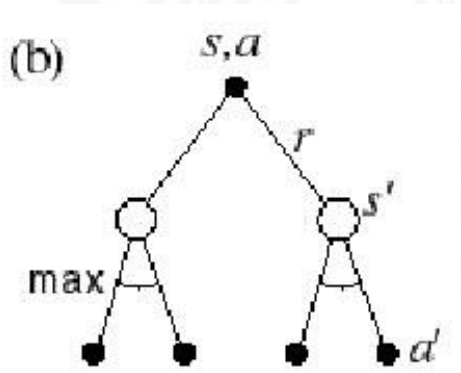


V^* ist die eindeutige Lösung dieses Systems der nichtlinearen Gleichungen.

Optimale Bellman-Gleichung für Q^*

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

Das entsprechende Backup-Diagramm:



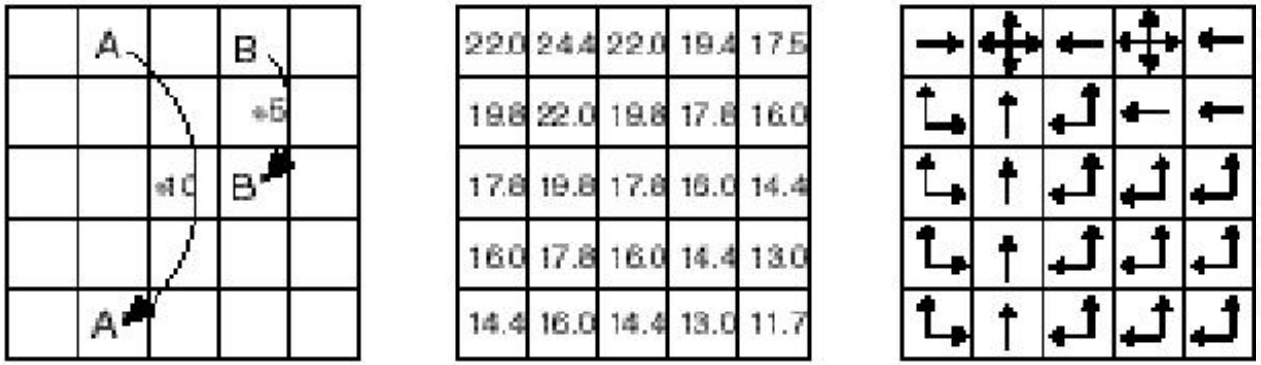
Q^* ist die eindeutige Lösung dieses Systems der nichtlinearen Geichungen.

Warum optimale Zustands-Wertefunktionen nützlich sind

Eine *Policy* die mit Rücksicht auf V^**greedy* ist, ist eine optimale *Policy*.

Deshalb produziert die *one-step-ahead*-Suche bei gegebenem V^* auf lange Zeit optimale Aktionen.

z.B., in der gridworld:



a) gridworld b) V^* c) π^*

Was ist mit optimalen Aktions-Wertefunktionen?

Bei gegebenem Q^* muss der Agent sogar keine *one-step-ahead*-Suche machen:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s, a)$$

Lösen der optimalen Bellman-Gleichung

- Um eine optimale *Policy* über die Lösung der optimalen Bellman-Gleichung zu finden benötigt man folgendes:
  - exakte Kenntnis der Dynamik der Umgebung;
  - genug Speicherplatz und Berechnungszeit;
  - die Markov-Wahrscheinlichkeit.
- Wieviel Speicherplatz und Zeit benötigen wir?
  - polynomiell mit der Anzahl der Zustände (über *dynamic programming* Methoden; spätere Vorlesung),
  - ABER, normalerweise ist die Anzahl der Zustände groß (z.B., hat Backgammon ungefähr 10^{20} Zustände).
- Wir müssen normalerweise auf Approximationen zurückgreifen.
- Viele RL-Methoden können als approximative Lösung der optimalen Bellman-Gleichung verstanden werden.

Zusammenfassung

- Agent-environment Interaktion
  - Zustände
  - Aktionen
  - *Rewards*
- ***Policy***: stochastiche Aktionsauswahl-Regel
- ***Return***: die Funktion der zukünftigen *Rewards*, die der Agent zu maximieren versucht
- Episodische und kontinuierliche Aufgaben
- Markov Wahrscheinlichkeit
- Markov Entscheidungsprozess
  - Übergangswahrscheinlichkeiten
  - Erwartete *Rewards*

Zusammenfassung (cont.)

- **Wertefunktionen**
  - Zustands-Wertefunktion für eine *Policy*
  - Aktions-Wertefunktion für eine *Policy*
  - Optimale Zustands-Wertefunktion
  - Optimale Aktions-Wertefunktion
- Optimale Wertefunktionen
- Optimale *Policies*
- Bellman-Gleichungen
- Die Erfordernis von Approximation