Reinforcement Learning

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2026-06 Dozent: Karsten Flügge

Themen des Tages

Tag 2

Markov Decision Processes (MDPs)
Definition und Eigenschaften von MDPs
Value-Funktionen
Policy

Pause

policy deterministisch
als Funktion
$\pi(s)=a$
$\pi[s]=a$ als map

policy stochastisch
$\pi(a|s)$ bedingte Wahrscheinlichkeiten
wir wollen Aktion wählen
wähle Aktion a im State s mit bestimmter Wahrscheinlichkeit

$\pi(\text{hoch}|\text{mitte}) = 50\%$
$\pi(\text{runter}|\text{mitte}) = 50\%$

Als Neuronales Netz π policy_model => softmax von den Aktionen

Markov Decision Processes (MDPs)

Markov-Entscheidungsproblem

Markow-Annahme

Die aktuelle 'beste Entscheidung' hängt nicht von der Vergangenheit ab.

"Die Umgebung ist gedächnislos"

mathematischer:

Die Wahrscheinlichkeit für einen Zustandsübergang hängt nur von dem aktuellen Zustand und dem Folgezustand ab und nicht von früheren Zustandsübergängen.

Equivalent: Wahrscheinlichkeiten sind Unabhängig $P(x|y)=P(x)*P(y)$

Markov Beispiele

Der 'beste Zug' hängt nur von den erwarteten Zügen des Gegners ab,

nicht von der Vergangenheit.

Z.B.: Roulette im Casino:

- Der Ausgang eines Wurfs (rot, schwarz, oder grün) hängt nur von der aktuellen Drehung und nicht von den vorherigen Ergebnissen ab. Die Kugel hat kein Gedächtnis, und jede Drehung ist unabhängig. Oft hört man so etwas wie "jetzt kam schon dreimal rot jetzt muss wohl schwarz kommen" was natürlich Unsinn ist!!

Z.B.: Schach:

WIE wir in die aktuelle Position gekommen sind, ist IRRELEVANT dafür, was wir als nächstes tun sollten.

(oder? meta: können wir aus dem vorigen Verhalten des Gegners etwas über ihn lernen? Kann aber Blöff sein)

Markov Beispiele

Der 'beste Zug' hängt nur von den erwarteten Zügen des Gegners ab,
nicht von der Vergangenheit.

- **Würfeln / Münze werfen**
- **Krieg Kartenspiel: höher/tiefer**
- **Dame, Schach, Mühle, Halma, Mensch-Ärger-Dich-Nicht viele Brettspiele**
- **Mikado, Jenga**

Gegenbeispiele

- **Jack in the box:** je länger man dreht desto Wahrscheinlicher springt er hoch
- Poker, Skat, Uno (leicht nicht-markovsch)
- Erdbeben (Gas-Ausdünstungen …)

Markov Beispiele

Markov-Eigenschaft:

Die aktuell optimale Entscheidung hängt nicht von der Vergangenheit ab.

<u>Beispiele</u>:

- **Schach**: Wie wir zur aktuellen Position gelangt sind, ist irrelevant für die nächste Entscheidung.

  (Meta-Anmerkung: Zwar lässt sich ggf. etwas über den Gegner lernen, das kann aber Täuschung sein.)
- Der „beste Zug“ hängt nur von den **möglichen gegnerischen Reaktionen** ab, nicht vom Spielverlauf.
- Der **„beste Zug“** basiert typischerweise auf der Annahme einer **optimalen Gegenspielweise**.

  Wenn der Gegner jedoch Fehler macht, kann ein riskanterer Zug optimal sein.
- **Moleküle im Gas**: Atome haben kein Gedächtnis, sie reagieren nur auf ihre Umgebung.

  Brown’sche Bewegung.
- **Roulette**:

  Der Ausgang (rot, schwarz, grün) hängt nur von der aktuellen Drehung ab – nicht von vorherigen Ergebnissen.

  Die Kugel hat kein Gedächtnis.

  Die Aussage „Dreimal rot, jetzt muss schwarz kommen“ ist statistisch Unsinn.

Markov-Modelle dienen der **Vereinfachung**:

Oft ist die Vergangenheit irrelevant oder nur marginal, sodass ein Markov-Modell als **Näherung** genügt.

Markov Gegen-Beispiele

Die folgenden Prozesse sind keine Marcow Prozesse

Der 'beste Zug' hängt nicht nur von allen zu erwartenden Gegen-Zügen des Gegners ab, sondern auch von der Vergangenheit.

1. Poker & Skat

Entscheidungen hängen nicht nur vom sichtbaren Zustand (z. B. Karten auf dem Tisch), sondern auch von der **Setzhistorie** ab. Ein Bluff funktioniert nur durch historisches **Kontextwissen**.

2. Menschliches Verhalten in realen Interaktionen

In empirischen Spielen (z. B. Verhandlungen) reagieren Menschen oft auf **Vertrauensaufbau** oder -bruch über Zeit. Keine Markov-Eigenschaft.

3. Gefängnisermüdung in Gefangenendilemma-Serien

Im iterierten Gefangenendilemma hängt der optimale Zug vom Verhalten des Gegenübers in früheren Runden ab (“**Tit for Tat**”). Der Zustand allein (z. B. die aktuelle Runde) ist nicht hinreichend.

Markov Gegen-Beispiele

Die folgenden Prozesse sind keine Marcov Prozesse

4. Rechtssysteme / Strafmaßentscheidungen

In manchen Systemen hängt das Urteil nicht nur vom aktuellen Delikt, sondern auch von der **Vorstrafen**historie ab. Kein Markov-Prozess.

5. Reinforcement Learning mit *partieller Beobachtung* (POMDPs)

Hier ist der Agent gezwungen, eine **Belief-State**-Verteilung zu führen – also eine Zusammenfassung der bisherigen Beobachtungen. Die optimale Aktion hängt dann nicht nur vom beobachteten Zustand ab.

💡 **jeder endliche Entscheidungsprozess** mit Gedächtnis als Markov-Prozess modellieren, wenn man den Zustand entsprechend **erweitert** – typischerweise als *history-dependent state* oder *belief state* im Fall unvollständiger Information.

💡 Das erinnert an Nondeterministic und Deterministic Automator.

Markov Doch-Beispiele

Prozesse lassen sich in Marcov Prozesse umwandeln

Oft spielt die Vergangenheit eine *Nebenrolle* so dass wir zur **Vereinfachung** der Situation einfach ein Markov Modell annehmen können.
"egal was du in der Vergangenheit für Unsinn gemacht, hast deine Zukunft hängt im wesentlichen von aktuellen Entscheidungen ab, du kannst die Vergangenheit nicht mehr ändern!"

💡 **jeder endliche Entscheidungsprozess** mit Gedächtnis als Markov-Prozess modellieren, wenn man den Zustand entsprechend **erweitert** – typischerweise als *history-dependent state* oder *belief state* im Fall unvollständiger Information.

💡 Das erinnert an Nondeterministic und Deterministic Finite Automates (DFA).

Wenn ein NFA sich die *Vergangenheit* (d. h. mögliche Pfade) merkt, kann man ihn determinisieren.
Analog konstruiert man aus einem spieltheoretischen Prozess mit Historie einen MDP mit Zuständen $S_t=H_t$

⚠️ **Aber**: Die **Komplexitätsexplosion** (Zustandsanzahl, Spielbaumgröße) ist nicht nur analog, sondern oft identisch in Struktur und Inkompressibilität.

Konzept	Nicht-Markovianisch	Markovianisch
Speicher	Explizite Historie oder Kontext	Zustand enthält alles Relevante
Automatentheorie	Nichtdeterministischer Automat (NFA)	Deterministischer Automat (DFA)
Zustand	Momentaner Zustand plus Verlauf	Zustandskombination (Powerset)
Entscheidbarkeit	Teilweise unentscheidbar / nicht endlich	Immer entscheidbar (bei endlicher Zustandsmenge)
Komplexität	Exponentiell (z. B. 2^n Zustände)	Linear in Pfadlänge

Markov Doch-Beispiele

für stark abstrahierte Modelle von Poker ist eine Markovianisierung realistisch – unter bestimmten Einschränkungen.

1. Partielle Information → Belief States (POMDP)

Poker ist kein MDP, da der Agent nicht den vollständigen Zustand (Gegnerkarten) kennt. Stattdessen:
Zustand = public cards + own hand + belief over opponent hands and strategies
→ Das führt zu einem belief-MDP mit kontinuierlichem Zustandsraum (Wahrscheinlichkeitsverteilungen über versteckte Information).
Formell Markovianisch, praktisch hochdimensional.

2. Strategieklassen als Zustände
In vereinfachten Varianten (e.g. Kuhn Poker):

- Man kann Spielhistories (z. B. Bet-Fold-Sequenzen) als abstrakte Zustände kodieren.
- Strategien werden darauf trainiert (z. B. via Counterfactual Regret Minimization).
- Diese Modelle sind tatsächlich Markovianisch mit endlichem Zustandsraum.

Beispiel:
s = (own card, bet history)
→ Zustand ist vollständig zur Strategieentscheidung.

3. Realistische Poker-Engines

State-of-the-art Systeme (DeepStack, Pluribus) approximieren belief-based Markov-Strategien:

- DeepStack führt continual re-solving durch: bei jeder Spielentscheidung wird ein lokales MDP auf Basis des bisherigen Verlaufs gelöst.
- Markovität wird lokal angenommen, aber belief update übernimmt die “Vergangenheit”.

Endliche Markov Prozesse

Endliche Markov-Entscheidungsprozesse

Endlich bedeutet dass die **Episode** (das Spiel) Nach einer bestimmten Anzahl von Schritten (**time** steps t) beendet wird und es eine finale Belohnung gibt (positiv oder negativ). Mit **terminalem** Zustand "Ziel/Ende"

Endlich II bedeutet dass der **Zustandsraum** $\mathcal{S}$ sowie die Menge aller möglichen **Aktionen** $\mathcal{A}$ und **Belohnungen** $\mathcal{R}$ endlich ist.

In diesem Fall spricht man aber eher von **diskret** versus **kontinuierlich.**

💡 wir konzentrieren uns auf **endliche** Episoden und fügen bei unendlichen Episoden ***Zwischenziele*** ein.

Beispiele **endlicher** Marcov Prozesse: alle Computerspiele bei denen Game Over steht, falls nur der aktuelle Situation zählt

Beispiele unendlicher Marcov Prozesse: lebenslanges lernen (aber mit endlichen Zwischenzielen!)

Episode vs Epoche

Episode = ein Spiel (von Umgebung diktiert über *finalen Zustand*)

Epoche = ein Trainingsschritt (beiliebig, z.B. mehrere Episoden)

$\tau = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}$ trajectory

Markov Beispiele

Markov Beispiele

Optimale Entscheidung in gegebener Situation hängt nicht von Vergangenheit ab

- **Lotto**
- **Viele Computerspiele (Pong)**
- was ist, wenn nur Position im State übermittelt wird, nicht Geschwindigkeit.
  Interner State relevant => dann NICHT Markow
- **Optimaler Pfad**
- **objektiv beste Züge**

Markov Gegenbeispiele

- **Viele Computerspiele** (immer wenn nur aktueller Screen als state übermittelt wird)
- **Wetter**

$\tau = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}$ trajectory

Pause

- **Pong ist Markov wenn alles sichtbar ist**

- was ist, wenn nur **Position** im State übermittelt wird, nicht Geschwindigkeit.

- ***Interner State relevant => dann NICHT Markow***

$\tau = \{s_t, a_t, r_t, s_t+1\}$ trajectory
$_{t=0}$

Pause

$\tau = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}$ trajectory

Es geht los

Klassische Formulierung des Reinforcement Problems:

Mathematisch als **Wahrscheinlichkeiten**

Aktion a führt uns von state s zu s' und wir kriegen reward r.

p(s', r | s, a) interne Environment

"Wahrscheinlichkeit dass wir für die Aktion a im Zustand s die **Belohnung r** bekommen und im *neuen* Zustand **s'** landen"

Bei deterministischer Umgebung einfach **p = 1**

Algorithmischer: [r,**s'**] = p(s,a) = environment.**step**(a,s)
Environment Function p gibt uns reward und neuen Zustand!
In aller Regel ist **p** komplett **versteckt**

Frage

p(s', r | s, a) next state und reward zusammen modelliert
p(s'| s, a) R(s,**a**) reward oft extra

"Wahrscheinlichkeit dass wir für die Aktion **a** im Zustand **s** die **Belohnung r** bekommen und im neuen Zustand **s**' landen"

Was sind Szenarien, bei denen nächster State **s**' zufällig ist?

Casino, Blackjack, **viele Spiele**, alle Spiele mit Würfel.

- Mikado? 'praktisch zufällig'
- RNG random number generator im Spiel

~~Immer wenn es andere Agenten gibt welche wir nicht verstehen.~~

Versus Brettspiele ohne Würfel

Es geht los

Klassische Formulierung des Reinforcement Problems:

Mathematisch als **Wahrscheinlichkeiten**

p(s', r | s, a) "Wahrscheinlichkeit dass wir für die Aktion a im Zustand s die Belohnung r bekommen und im neuen Zustand s' landen"

Selbst für deterministische Situationen, also wenn alles Verhalten der Umgebung komplett vorhersagbar ist, wird dieses Framework verwendet (mit **p=1**)

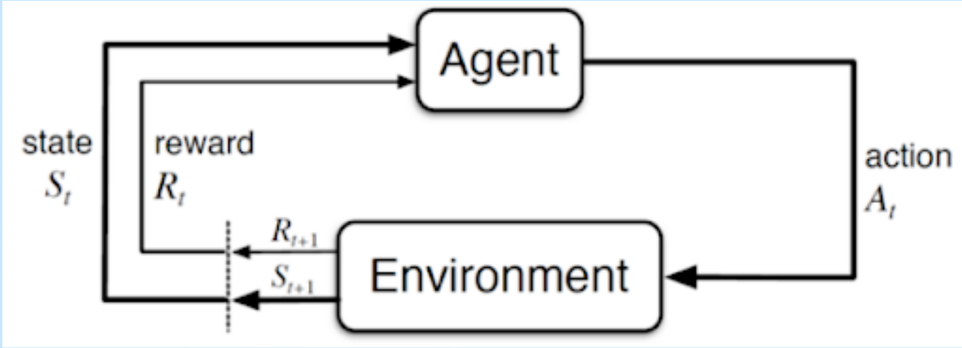
Um das Framework allgemein zu halten sprechen wir bei allen Konzepten (ausser Zustand) von **Funktionen**, auch wenn sie nur statische Einträge in einer Tabelle sind.

In der Tat ist dieser Ausdruck in der Regel viel intuitiver als auf den ersten Blick aussieht, z.B. Ein einarmiger Bandit gibt immer mit 40% W-Keit einen Gewinn aus wenn man ihn wählt => **p=0.4**

Also: Keine Angst vor Mathematik!

Markov Decision Processes (MDPs)

FORMAL:



Ein Markov Decision **Process** (MDP) ist ein **Tupel (**$\mathcal{S}$**,** $\mathcal{A}$**,** $\mathcal{P}$**,** R**, ɣ)**, bestehend aus:

- $\mathcal{S}$: endliche Menge von **States**/Zuständen
- $\mathcal{A}$: endliche Menge von **Aktionen**
- $\mathcal{P}$: Übergangsfunktion ≈ Environment p

  $\mathcal{P}$(s′ | s, a) = Wahrscheinlichkeit, in Zustand s′ zu gelangen, wenn in Zustand s die **Aktion a** gewählt wird $\mathcal{P}$ **P**robability
- $\mathcal{R}$: Reward Belohnungsfunktion

  $\mathcal{R}$(s, a, s′) = erwartete Belohnung (**r**eward) beim Übergang von s nach s′ unter Aktion a
- **ɣ** ∈ [0, 1]: Diskontfaktor für zukünftige Belohnungen

⚠️ oft werden $\mathcal{P}$ **und** R **kombiniert zu p(r, s′ | s, a)** *implizit* in **env.step**

⚠️ $\mathcal{P}$ = **p p**robability **≠ π p**olicy 'intern'

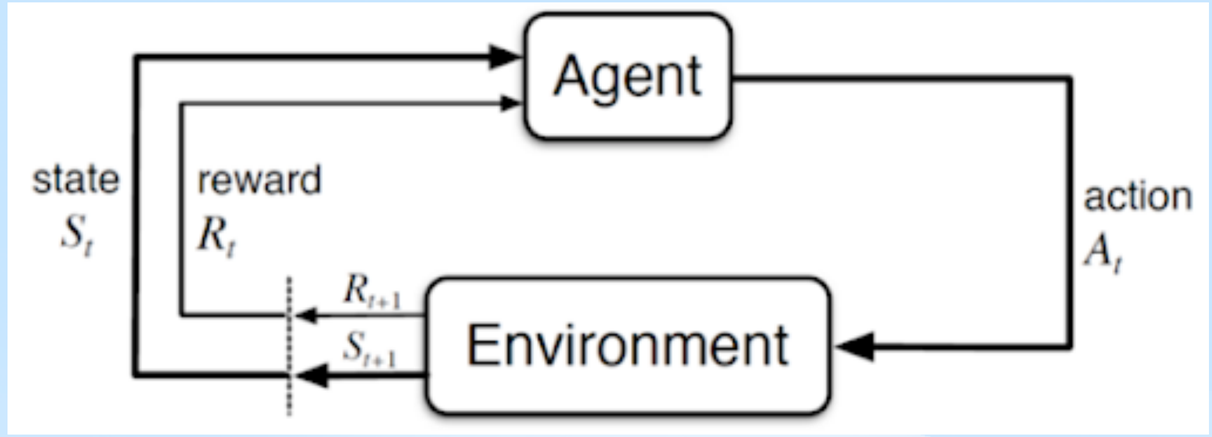
"Wahrscheinlichkeit beim Übergang von einem Zustand s mit einer Aktion a einen neuen Zustand s' und eine Belohnung (reward) r zu erhalten."

Markov Decision Processes (MDPs)

Warum ist π nicht Teil vom offiziellen MDP?

MDP ist eher der Prozess, das **Problem**,

Wir (Agent mit **π**) sind die **Lösung**.



Markov = kein interner Zustand, Zustand vom MDP wird **vollständig ausgeliefert**

Erweiterte Konzepte

Erweiterte (Hilfs)Konzepte

- **Episode:** "Spiel" Ablauf einer Folge von Aktionen und Zuständen ≈ **Trajektorie** $\tau=(s_0,a_0, \ldots a_t,s_t)$
- **Gain** Gewinn **g:** (Erwartete) *Summe der Belohnungen*, bis zum Ende des Spieles
- **Value Function v**: Erwartete zukünftige Belohnung ab einem **Zustand** ≈
- **Qualität (Quality) q**: Erwartete zukünftige Belohnung für eine **Aktion** in einem Zustand.

Value bewertet eine Position

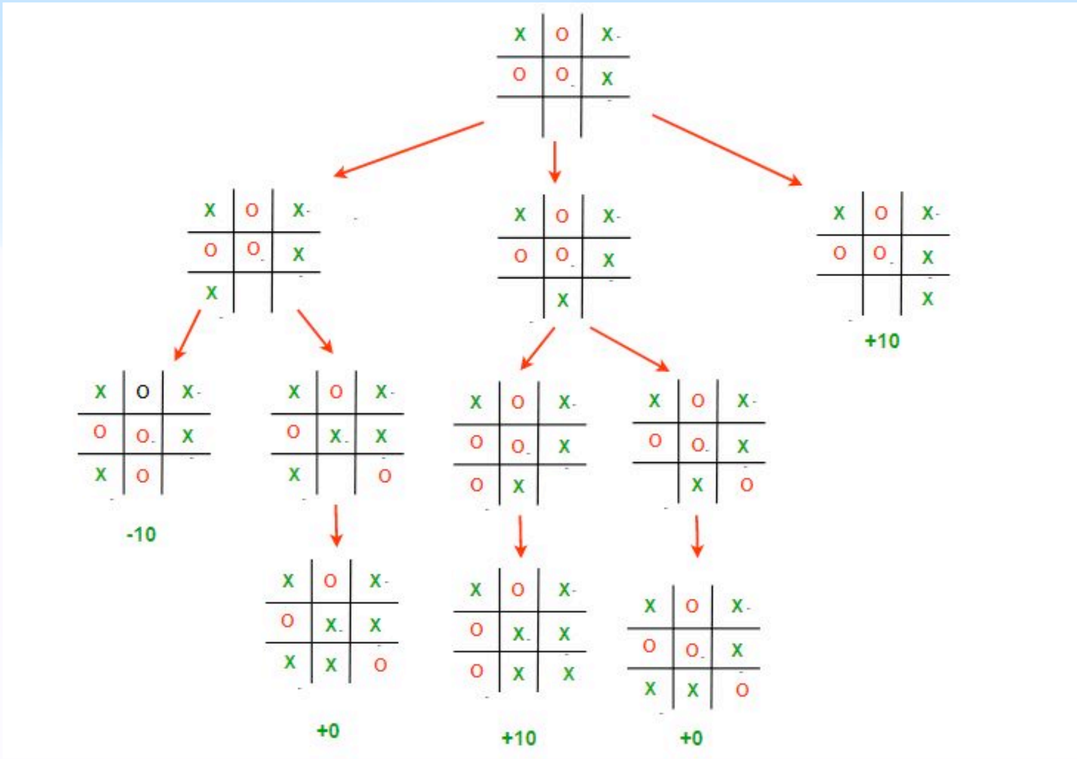
Quality bewertet einen Zug!

Man kann in **guten** Positionen **schlechte** Züge machen und
in **schlechten** Positionen **optimale** Züge also das 'bestmögliche'.
Ob ein Zug zielführend war entscheidet sich oft erst am Ende des Spiels "Bauernopfer".

Erweiterte Konzepte

Erweiterte (Hilfs)Konzepte

- **Episode:** "**Spiel**" (bei uns endlich: es gibt mindestens einen *terminalen* Zustand)
- Ablauf einer Folge von Aktionen und Zuständen ≈
- **Trajektorie** $\tau=(s_0,a_0, \ldots a_t,s_t)$ ein "Pfad" im Entscheidungsbaum



Beispiele

Aus **schlechten Positionen** gibt es trotzdem **optimale Aktionen** also das 'bestmögliche' Züge:

Poker **blöffen / all in / raise** : bei schlechtem Blatt so tun als hätte man was
einfach **aufhören** (russisches Roulette, tik tok toe : keiner kann Gewinnen)
ich kann nicht gewinnen => **unentschieden** rausholen
Börse: gar **nicht anfangen**
Börse: wir haben investiert, Kurse gehen runter => halten / **cut losses**
Verluste begrenzen, in den sauren Apfel beißen
sunken cost fallacy irreversible Kosten

das echte Leben™

Beispiele

Man kann in **guten Positionen schlechte Züge** (Aktionen) machen:

- Erschiessen nach Lotto Gewinn
- Schachmatt **übersehen**!
- Bei Rot (gute Position weil keine Gefahr) : GAS GEBEN!!
- Blackjack 20 auf der Hand noch eine Karte anfragen 'hit' (schlecht bei Markov ohne Kartenzählen)
- Unforced error in Tennis … (?)

Aus **schlechten Positionen** gibt es trotzdem **optimale Aktionen** also das 'bestmögliche' Züge

- Lotto verloren => aufhören zu Spielen
- Point Break
- Beim Schach: Gegner unter Druck setzen in der Hoffnung das er seinen besten Zug übersieht

- Tic Tac Toe : **man kann nicht Gewinnen, trotzdem verhindern zu verlieren**

Gewinn

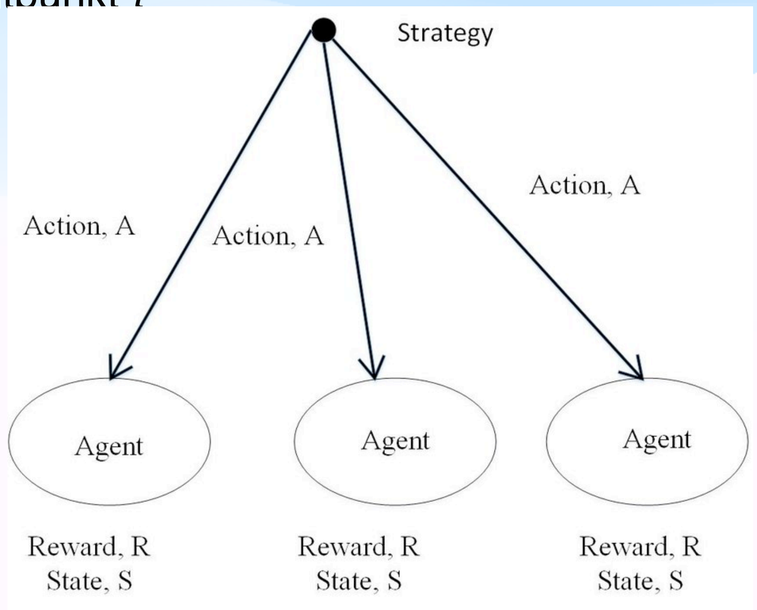
"Gewinn ≈ Summe Belohnungen" (erwartet oder real)

$G_t = \sum_k \mathbf{r}_{k+1}$

Internes *Hilfsmittel* mit welcher wir unsere Policy verfeinern können
Policy: "möglichst viel Gewinn!" "maximieren der Belohnungen"

Gewinn G = kumulierte, **diskont**ierte Belohnung ab einem bestimmten Zeitpunkt *t*
***G**oal / **g**ain = return (cumulative discounted reward) following t*

$G_t = \sum_k \gamma^k \mathbf{r}_{k+1}$ t = Aktueller-Zustand k=Zug# ab jetzt (**r ab jetzt** "lokal")



Warum *discount factor* γ*?* Wir können steuern:
γ=0 "gierig" "myopic" nimm die erstbeste Belohnung, ignoriere Rest (*k=0 => 0^0=1!)*
γ=1 "langfristig" bewerte alle Belohnung gleich

Typisch: **γ≈0.9 oder γ≈0.99** (je nach typischer Spiellänge …)

γ=0 nimmt erste Belohnung (gegebenenfalls aber *vom Ende* rückwärts betrachtet)

Gewinn

Es gibt zwei Formulierungen, je nach dem ab wann wir rewards r zählen

a) ab Anfang t=0 "global" r0 = Belohnung im ersten Zug
b) ab Aktuellem-Zustand t "lokal" r0 = Belohnung im aktuellen Zug

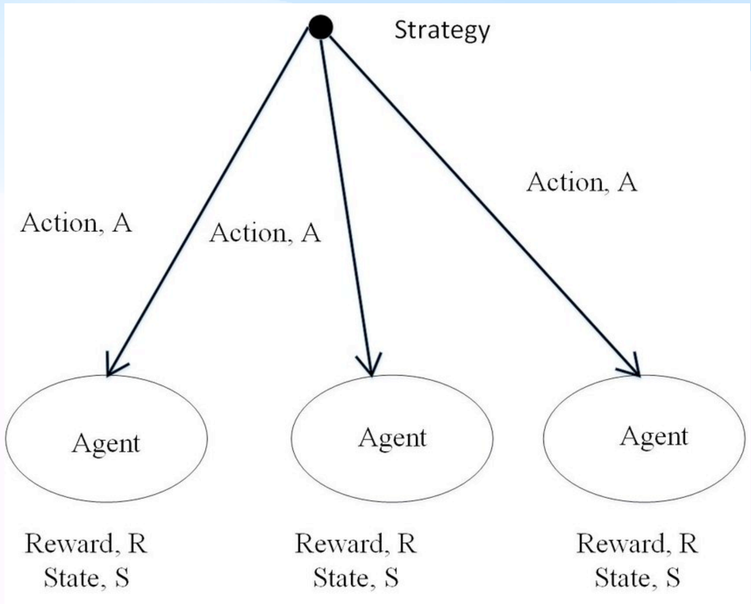
$G_t = \sum_k \gamma^k \mathbf{r}_{t+k+1}$ t = Aktueller-Zustand k=Zug# ab jetzt (**r ab start** "global") oder

$G_t = \sum_k \gamma^k \mathbf{r}_{k+1}$ t = Aktueller-Zustand k=Zug# ab jetzt (**r ab jetzt** "lokal")

*oft t=0 alles 'ab Anfang' Markov oder t was können wir **ab jetzt** erreichen?*

Warum *discount factor* γ*?* Wir können steuern:
γ=0 "gierig" "myopic" nimm die erstbeste Belohnung, ignoriere Rest (*k=0 => 0^0=1!)*
γ=1 "langfristig" bewerte alle Belohnung gleich

Typisch: **$\gamma \approx 0.9$ oder $\gamma \approx 0.99$** (je nach typischer Spiellänge …)



Gewinn

`Alternative 'rekursive Sichtweise / Formulierung':`

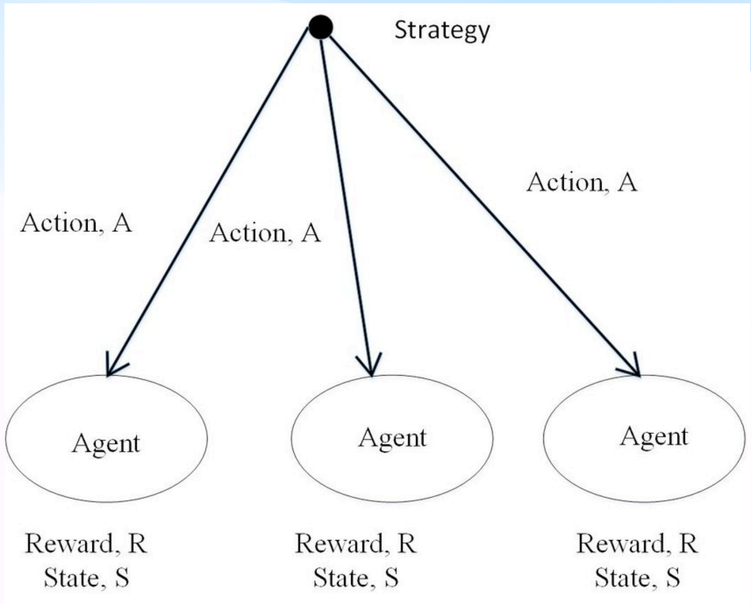
`Gewinn = konkrete Belohnung (reward r) plus zünftiger Gewinn`

$G_t = \mathbf{r}_t + \gamma \cdot G_{t_{+1}}$

führt zu selben Formel:

$G_t = \sum_k \gamma^k \, r_{t_+ k_{+1}}$!!!

Diskont Faktor **γ sagt wie sehr wir an** `zünftige Gewinne 'glauben'`
γ = 0 gar nicht
γ = 1 komplett ($G_t = \sum_k \mathbf{r}_{k_{+1}}$)



Gewinn

G goal / gain / Gesamt Gewinn = ∑ R rewards

"Gewinn ≈ ∑ Belohnungen"

Internes *Hilfsmittel* mit welcher wir unsere Policy verfeinern können

`Gewinn` = `kumulierte, diskontierte Belohnung` ab einem bestimmten Zeitpunkt *t (unserem Aktuellen Zug)*
Goal / gain = return (cumulative discounted reward) following t

$G_t = \sum_k \gamma^k R_{t_+k_{+1}}$ *Zeitpunkte k*

***t = 10** : wir sind in Runde 10*
k = 0 : aktueller Gewinn R11 Belohnung von Runde 10 auf 11 (10+1)
k = 1 : Gewinn nächsten Zug R12 Belohnung von Runde 11 auf 12 (10+1+1)
k = 2 : Gewinn übernächsten Zug R13 Belohnung von Runde 12 auf 13 (10+2+1)
…
k = T : Terminaler Zustand "Ende des Spiels" Finale Belohnung
G10 = R11 + γR12 + γ² R13 + … Gewinn in Runde Zehn = Reward für Runde 10 + γ * Reward für Runde 11 + … bis zum Ende

Gewinn

Lokale Sicht

$G_t = \sum_k \gamma^k R_{k_{+1}}$ *Zeitpunkte k*

t = 10 : wir sind in Runde 10
k = 0 : aktueller Gewinn R10 Belohnung von Runde 10
k = 1 : Gewinn nächsten Zug R11 Belohnung von Runde 11 (10+1)
k = 2 : Gewinn übernächsten Zug R12 Belohnung von Runde 12 (10+2)
…
k = T : Terminaler Zustand "Ende des Spiels" Finale Belohnung
G10 = R10 + γR11 + γ² R12 Gewinn in Runde Zehn = Reward für Runde 10 + γ * Reward für Runde 11 + … bis zum Ende

zukünftige Belohnungen werden exponentiell kleiner gewichtet

Gewinn

Im code wird alles wieder viel einfacher

gain = 0
for each step:
*gain = reward + gamma * gain*

DISCOUNT FAKTOR

DISCOUNT FAKTOR

Exponentieller Verfall:

γ DISCOUNT FAKTOR sagt, wie gierig wir die Belohnungen akzeptieren
γ = 1/2
γ^2 = 1/4 … geht exponentiell runter !

*∑ zickzack Zeichen = Zu**sam**menZählen*
`sum ≈ sammeln`

`Warum` *discount factor* γ*?* Wir können steuern:
γ=0 "gierig" "myopic" nimm die erstbeste Belohnung, ignoriere Rest (*k=0 => 0^0=1!)*
γ=1 "langfristig" bewerte alle Belohnung etwa gleich

Markov Decision Processes (MDPs)

Reinforcement Problem:

Maximiere den erwarteten Gewinn durch eine 'optimale policy'

Gewinn = kumulierte, diskontierte Belohnung ab einem bestimmten Zeitpunkt *t*
Goal / gain = return (cumulative discounted reward) following t

Finde / lerne π^* so, dass $\mathbb{E}[G_t]$ = max ab t=0 (Gewinn G = $\sum_k \gamma^k R_k$)

Zufall vom Reward kommt über

p(s',R | s, a) oder R(s,a)

Value und Quality

Für eine **policy**

Es gibt eine lange Historie von komplexer Theorie, wie man den **Wert** einer Position (**value** of **state**) und die **Qualität** von Aktionen (**quality** of **action**) schätzen kann.

value$_p$**(state)** ist der der erwarteten kumulierten Belohnungswert eines Zustandes

$\mathbf{v}_p(\mathbf{s}) := \mathbb{E}[G_t]$ **"Erwarteter Gewinn"** aus einem state s unter einer **policy π**

$= \sum \mathbf{q(a \mid s)}$ **wir summieren über alle Aktionen (und deren Qualities)**

quality$_p$**(action)** ist der erwarteten **kumulierten Belohnungswert** (Reward)

qπ**(a** | s) = **q**π(s, a) = $\mathbb{E}_{\boldsymbol{\pi}}[G_t]$ state-action-pair einer policy
qπ**(a** | s) = $\mathbb{E}_{\boldsymbol{\pi}}[G_t]$ "**Erwarteter Gewinn**" einer Aktion und der Folgezustände"
qπ**(a** | s) = max(**v**(s')) bester nächster State den ich erreichen kann, bei optimal greedy **policy**

statt diesen mühselig zu **berechnen** wird der **Q-Wert** in Zukunft **geschätzt / gelernt**

Value und Quality

Hilfskonzept

- **Value Function v**: Erwartete zukünftige Belohnung ab einem **Zustand** ≈
- **Qualität (Quality) q(a**,s): Erwartete zukünftige Belohnung für eine **Aktion** in einem Zustand.

Value bewertet eine Position

Quality bewertet einen Zug!

Man kann in **guten** Positionen **schlechte** Züge machen und
in **schlechten** Positionen **optimale** Züge also das '*bestmögliche*'.
Ob ein Zug zielführend war entscheidet sich oft erst am Ende des Spiels "Bauernopfer".

Zwei Seiten der Medaille:
man kann mit **Value die Quality schätzen "Welche Aktion bringt den besten Zustand?"**
man kann mit **Quality die Value schätzen "Wert der besten Aktion."** oder Summe über alle Aktionen

Value und Quality

quality ist der erwarteten **kumulierten Belohnungswert** (Reward)

statt diesen mühselig zu berechnen wird der **Q-Wert** nun **gelernt:**

💡 Der **Zustandsraum** wird bei Spielen schnell quasi **unendlich,** so dass wir auf unser **Netz für Schätzungen** angewiesen sind. (Schätzungen der eventuell theoretisch existierenden optimalen Spielzüge…).

Hier kommt die Kraft von **Mustererkennung** zum tragen!

💡Damit wird ein riesiger Berg von Theorie hinfällig:

💡 Deep-Mind (*https://homl.info/dqn*) konnte im **Jahr 2013** demonstrieren, dass Deep Neural Networks besonders bei komplexen Aufgaben viel besser funktionieren als klassische Schätzverfahren und keinerlei Extraktion von Merkmalen nötig ist.

Value und Quality Beispiel Schach

value$_p$**(state)** ist der der erwarteten kumulierten Belohnungswert eines Zustandes

`gute Position`

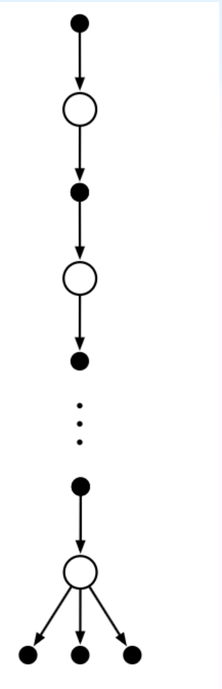
quality$_p$**(action** | state**)** ist der erwarteten

kumulierten Belohnungswert beim Ausführen der **Aktion in** einem **Zustand "state action pair"**

"guter / dummer Zug?"

Aktionen werden oft als kleiner schwarzer Punkt dargestellt

States/Zustände (Positionen) als weisser Kreis.



Pfad Diagramme

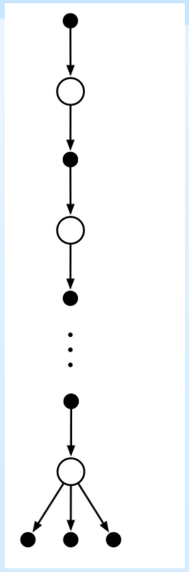
- : Aktionen werden oft als kleiner schwarzer Punkt dargestellt

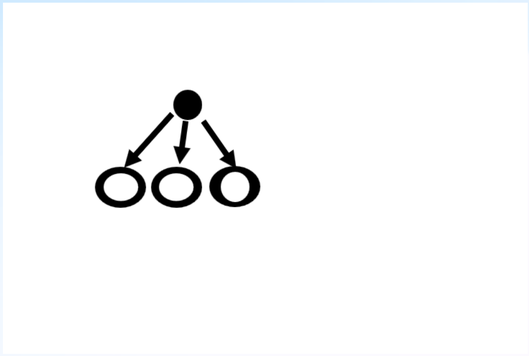
O : States/Zustände (Positionen) als weisser Kreis.

deterministischer Pfad

In einem state können mehrere Aktionen gewählt werden (klar)

Eine **Aktion kann aber auch in verschiedenen Zuständen resultieren** (nicht beim Schach)





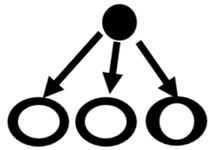
Pfad Diagramme

Beispiele wo mit gleicher Aktion zufällige **Rewards** erreicht werden:

- Ein Würfel
- Lotto

Beispiele wo mit gleicher Aktion zufällige **States** erreicht werden:

- **spread / Streuung beim Schiessen**
- **schlittern auf Eis**
- **die meisten echten kontinuierliche physikalischen Systeme?**

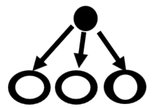


Value und Quality

Für eine policy

quality$_p$(action) ist der **erwartete Gewinn** einer Aktion

quality$_p$(action) ist der erwartete **kumulierte Belohnungswert** einer Aktion



- **$q_\pi(a|s) = \mathbb{E}[G_t]$** state-action-pair einer Policy
  $= \sum p(s|a)\ v(s) = \mathbb{E}\ (\ \sum r + \text{gamma}\ G'\ ) = \mathbb{E}\ (\ \sum \text{gamma}\hat{}k\ r(k)\ )$
  Im ersten Schritt hängt q-Wert nur von Aktion und Umgebung ab, die policy beeinflusst erst den v-Wert.

value$_p$(state) ist der erwarteten kumulierten Belohnungswert eines **Zustandes**

- **$v(s) = \sum_a \pi(a|s)\ q(s, a)$**
  „Wert einer (Schach-)Position unter der Policy“ = “Summe:Werte aller 'qualifizierten' Aktionen”
  policy $\pi(a|s)$ Wahrscheinlichkeit a auszuführen mit policy π
  in/aus einem state s

Wert einer Aktion gewichtet nach Wahrscheinlichkeit, dass wir sie in unserer Strategie auswählen.

Episodic and Continuing Tasks

Endlich nennt man auch '**episodisch**'
Wenn die Folge der Zustände $\mathbf{s}_t$ irgendwann in einem so genannten **terminalen Zustand** endet dann spricht man von **episodischen Problemen.** Das ist unser Standardszenario, bei welchen wir am Ende eine **finale Belohnung** erhalten. Letzter Zustand **T**=S_t(final)

Ansonsten spricht man von **kontinuierlichen Aufgaben**
(werden wir nur am Rande betrachten)
Auch: Nicht endlichen Probleme.

Episode = Tupel aller Zustände (s_i , a_i) **"ein Spiel"**
Synonym **Pfad Trajektorie (s_i , a_i)** "Haufen von Zügen"

⚠️ **Nicht verwechseln mit "Trainingslauf" 'Episode' im ML Epoche**

Pause

$\tau = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}$ trajectory

Optimalität

Optimale policy π*, action **a***, **q***(a), **v***(s)

Optimalität wird gekennzeichnet durch einen Stern*

Optimalität ist in der Regel theoretisch:
wir werden es nie ganz erreichen aber versuchen uns zu nähern!

Optimale policy π* **: wähle immer bestmögliche** action **a***
bestmögliche action **a*** **: höchster erwarteter Gewinn**

Optimalität

Optimale policy π*, action **a***, **q***(a), **v***(s)

Optimale Schätzungen q*(a), v*(s) für quality / value

unsere Bewertung der Aktion **q***(a) und
unsere Bewertung des Zustandes **v***(s) entsprechen **der Wahrheit**

Optimal Value und Quality

optimal $value_p$*(state)

- **v***(s): optimal value "**Wahrer Wert**" eines Zustands s (unter einer Policy)

optimal $quality_p$*(action | state)

- q*(a): optimal quality "**Wahrer Wert**" einer Aktion

optimal actions

Wenn die Qualität aller Aktion bekannt ist also man für alle Aktionen den erwarteten Gewinn geschätzt hat, dann ist *eine* Strategie (**policy**), stets die **Aktion** mit dem höchsten erwarteten Gewinn auszuwählen.

$\pi^* = A_t = \text{argmax}_a \, Q_t(a \mid s)$ "bestmögliche Aktion*" "**greedy policy**"
* bei aktuellem Wissensstand

Diese kompakte Schreibweise bedeutet Algorithmus:
suche diejenige Aktion für welche die **Qualität** maximal ist (zum Zeitpunkt t).

💡 Wenn wir nicht im Voraus planen können wir nur den aktuellen Zeitpunkt betrachten und uns die Zeitpunkte t auch wegdenken.

optimal actions

argmax so ähnlich wie Maximum, aber wir gucken WO das Maximum erreicht wird

```
x = [ 1 , 4 , 2 , 3]
max(x) = 4
argmax_a(x) = x.index(max(x))
argmax_a(x) = np.argmax(x) # numpy
argmax_a(x) = 1 POSITION vom Maximum
```

argmax_a(f(a)) heist bestes a bei dem f(a) maximiert wird

optimal greedy policy

Eng mit dem Begriff der optimalen Wertefunktion ist die **Greedy Policy** verwandt:

Wenn die Qualität aller Aktion bekannt ist also man für alle Aktionen den erwarteten Gewinn geschätzt hat, dann ist eine Strategie (policy), stets die 'optimale' Aktion mit dem höchsten erwarteten Gewinn auszuwählen.

$\pi(a|s)$ = if a == $\text{argmax}_a\ Q_t(a)$ then **1** else 0 // 100% die Beste, 0% die anderen

$\pi(s) = \text{argmax}_a\ Q_t(a)$ # "immer beste Aktion"

Diese kompakte Schreibweise bedeutet Algorithmus: suche diejenige Aktion für welche die Qualität maximal ist (zum Zeitpunkt t) und wähle sie immer.

⚠️ wir schätzen Q nur, Schätzung kann falsch sein => policy nicht optimal

💡 Wenn wir nicht im Voraus planen, können wir nur den aktuellen Zeitpunkt betrachten und uns die Zeitpunkte t auch wegdenken.

ε-greedy policy

Wenn wir einfach nur **gierig** die *nach aktuellem Wissen*(!) **erstbeste Aktion** auswählen kann es sein dass wir verpassen dass es theoretisch **bessere Aktionen** geben kann. Klar wenn wir objektiv die beste Aktion kennen würden dann könnten wir immer gierig sein aber in der Regel können wir die Qualität von Aktionen nur schätzen!

$Q_t(a)$ als heuristische Schätzung:
$Q_t(a)$ = Summe der Belohnungen bei Wahl von a / Anzahl der bisherigen Wahlen von a
$Q_t(a) = (1 / N_i) \cdot \sum R_i$ für $A_i = a, i = 1 \ldots t-1$, mit $N_i := \sum (A_i = a)$

Wären wir stets gierig wäre meist $N_i = 0$ und $Q_t(a)$ unbekannt.

Auf dieses Dilemma gehen wir am Tag vier noch genauer ein, für jetzt können wir vermerken dass die folgende Strategie in vielen Fällen besser ist als die reine gierige Strategie da sie uns immer wieder testen lässt *ob es vielleicht bessere Alternativen gibt*:

ε-greedy policy:
Mit Wahrscheinlichkeit **ε** wird eine **zufällige Aktion** gewählt, sonst die gierige!

Exploration: probiere erst alle möglichen Aktionen um Daten zu sammeln

Beispiel q(a)

Quality = Summe zukünfiger Belohnungen

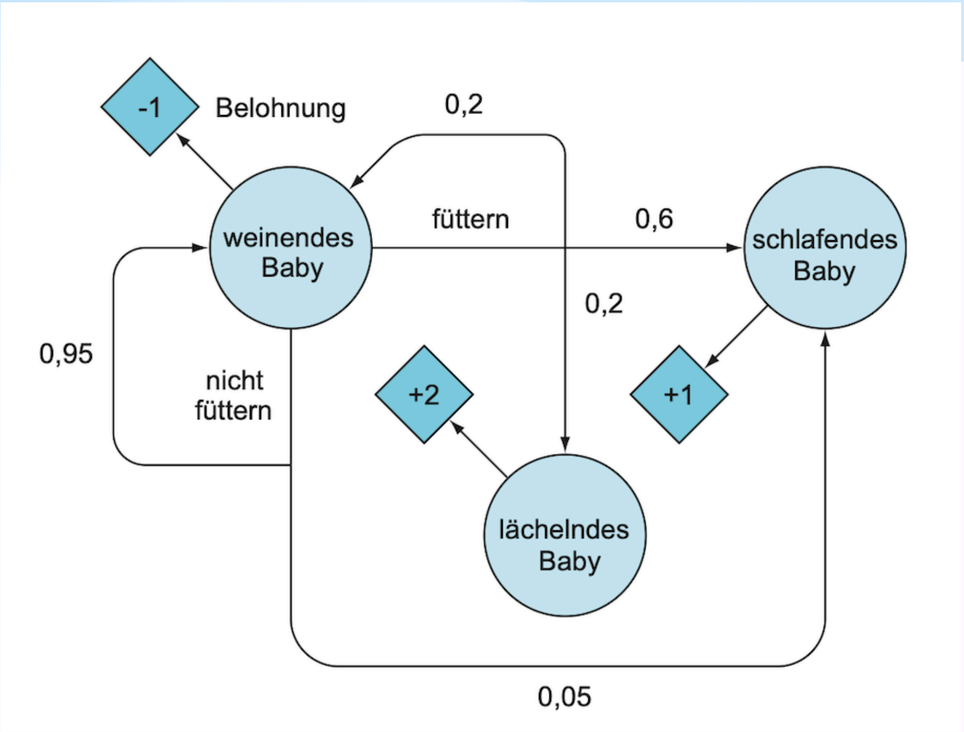
unsere Quality soll 'beste Quality' schätzen

$q_\pi(a|s) = \mathbb{E}_\pi[G_t]$ = Erwarteter Gewinn unter Policy π = $\mathbb{E}_\pi[\sum \gamma^i * r_i]$

reward im Schritt i, γ = discount factor = 0

Zustände S=[weinen,lachen,schlafen]
Aktionen A=[füttern, ignorieren]

abstrakte Belohnung "Freude"
[r,s']=a(f|w) 20% s'=l lächeln r=2
[r,s']=a(f|w) 20% s'=w r=-1
[r,s']=a(f|w) 60% s'=schlaf r=1
[r,s']=a(i|w) 95% s'=w r=-1
[r,s']=a(i|w) 5% s'=schlaf r=1



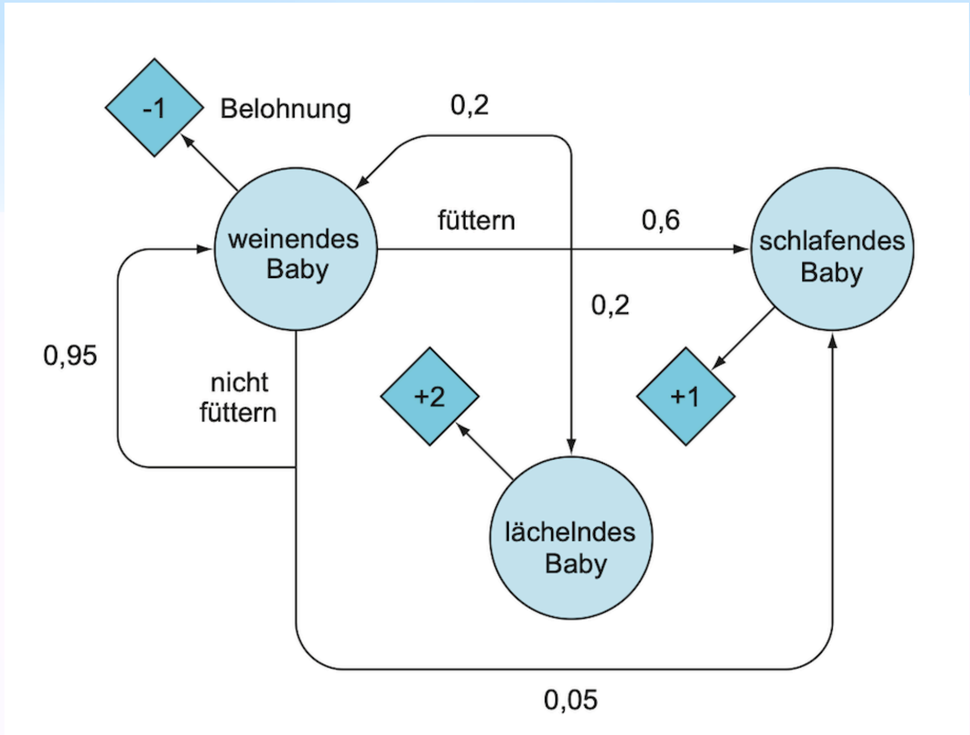
Beispiel q(a)

q(a)=E[G]=∑p·γ^{l}*$\mathbf{r}_i$ reward im Schritt i, γ = discount factor 0

Zustände S=[**w**einen, **l**achen, schlafen]
Aktionen A=[**f**üttern, **i**gnorieren]

abstrakte Belohnung "Freude"
[r,s']=a(f|**w**) 20% s'=l lächeln r=2
[r,s']=a(f|**w**) 20% s'=w r=-1
[r,s']=a(f|**w**) 60% s'=schlaf r=1
[r,s']=a(i|**w**) 95% s'=weinen r=-1
[r,s']=a(i|**w**) 5% s'=schlaf r=1

q(**f**|w)= E(G) = ∑ p(s',r'|s , a) * $\mathbf{r}_i$
= lächeln + weinen + schlafen
= 0.2***2** + 0.2***-1** + 0.6 * **1** = **0.8**
q(**i**|w) = 0.95***-1** + 0.05***1** = **-0.9**



Beispiel v(s)

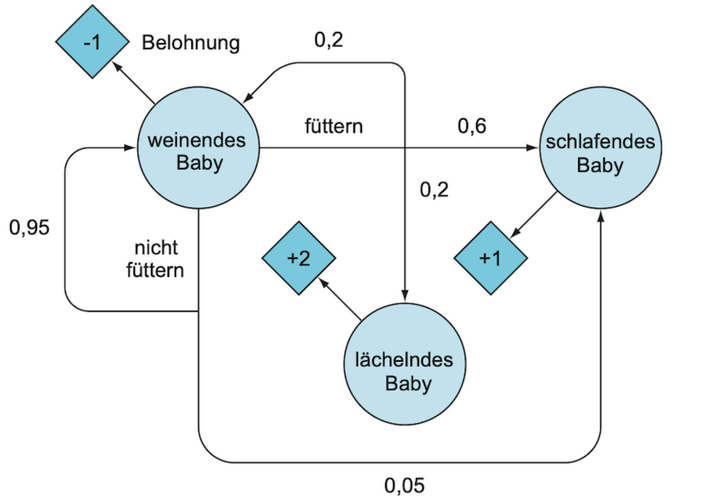
v(s) = ∑**p**(…)**q**(a|s) **policy** diktiert ob wir Aktion wählen
v(s) = ∑ₐ *π*(a|s) q(s, a)

Wert vom Zustand "**W**einen" bei "immer **f**üttern"

v(s=w)= 100% q(füttern|w) + 0% * q(ignorieren|w)
v(s=w)= 1.0* .8 + .0 * -.9 = .8 (value of state 'weinen')
^^ q(füttern|w) hatten wir schon berechnet (Folie davor)
greedy policy: wähle immer die **beste** Aktion

q(f|w)= **0.8**
q(i|w)= **-0.9**

v(s=schlaf) = 1 oder **0** (nichts in der Summe)
v(s=**lächeln**) = 2 oder **0** (nichts in der Summe)



Aufgabe

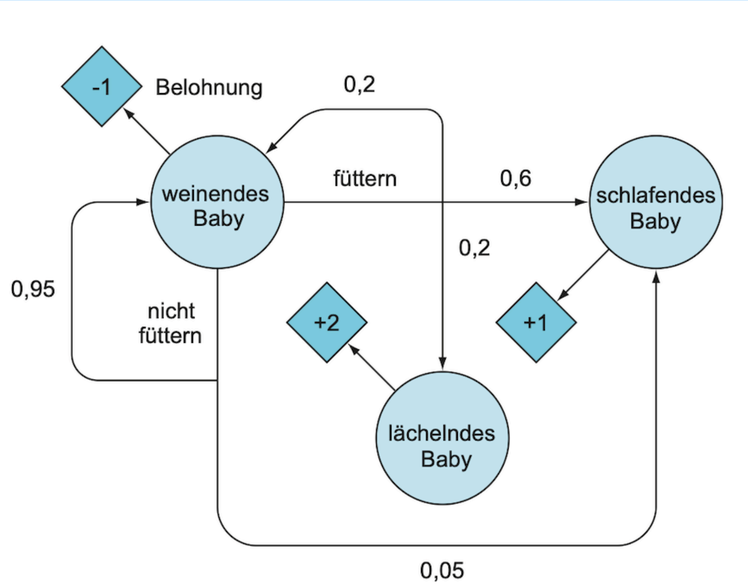
Bewertung vom State "weinen" unter Policy "nie füttern"

v(s)=∑**p**(…)**q**(a|s) **policy** diktiert ob wir Aktion wählen
v(s) = $\sum_a$ π(a|s) q(s, a)

Wert vom Zustand "**W**einen" bei "**NICHT f**üttern"

v(s=w)= 1(100%) * q(i|w) + 0 * q(f|w) = -.9
  q(f|w) hatten wir berechnet

q_π(f|w) = **0.8**
q_π(i|w) = **-0.9**



Bewertungen von state hängen immer von Policy π ab

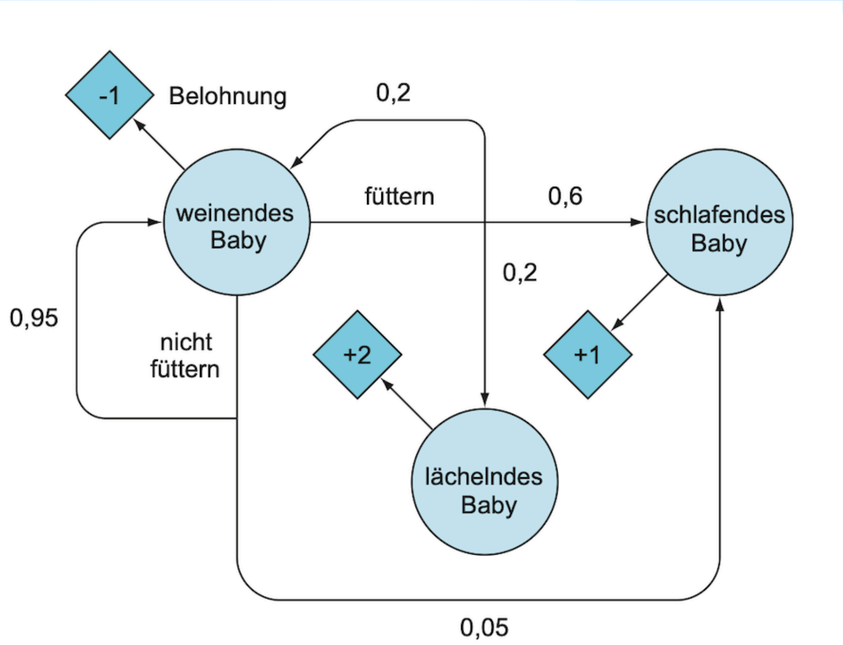
Beispiel v(s)

Aufgabe

a)Wert vom Zustand "Weinen" bei $\pi 1$:"immer ignorieren"
b)Wert von "Weinen" bei $\pi 2$: 50% füttern, 50% ignorieren

v(s)=$\sum\pi$(…)q(a|s) policy diktiert ob wir Aktion wählen

v(s=w)=



q(f|w)=E(g)=$\sum$p(s',r'|s , a) * r_i
=0.2*2 + 0.2*-1 + 0.6 * 1 = **0.8**
q(i|w)=0.95*-1 + 0.05*1 = **-0.9**

Aufgabe (fortgeschritten)

Bewertung vom State "weinen"
unter Policy "50% füttern, 50% ignorieren"

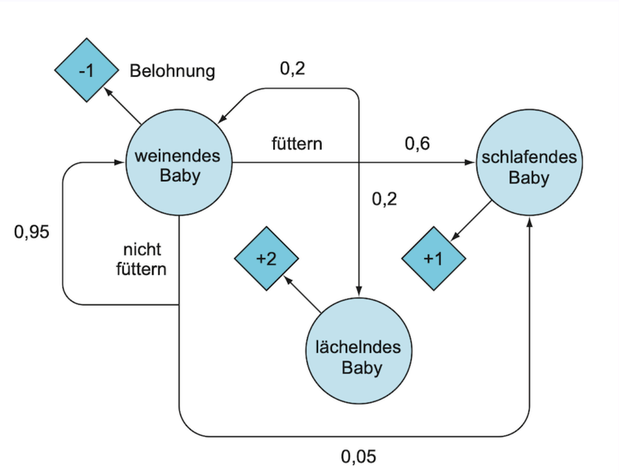
v(s)=∑**p**(…)**q**(a|s) **policy** diktiert ob wir Aktion wählen
v(s) = ∑ₐ *π*(a|s) q(s, a)

Wert vom Zustand "**W**einen" bei "**NICHT f**üttern"

v(s=w) = ***π***(f|w) * q(f|w) + *π*(i|w) * q(i|w)

v(s=w)= .5(50%) * q(f|w) + **.5** * q(i|w) = .4 - .45 = **-.05**

q(f|w)= **0.8**
q(i|w)= **-0.9**
π(f|w) = .5 "50% füttern"
π(i|w) = .5 "50% ignorieren"



Beispiel q kumulativ?

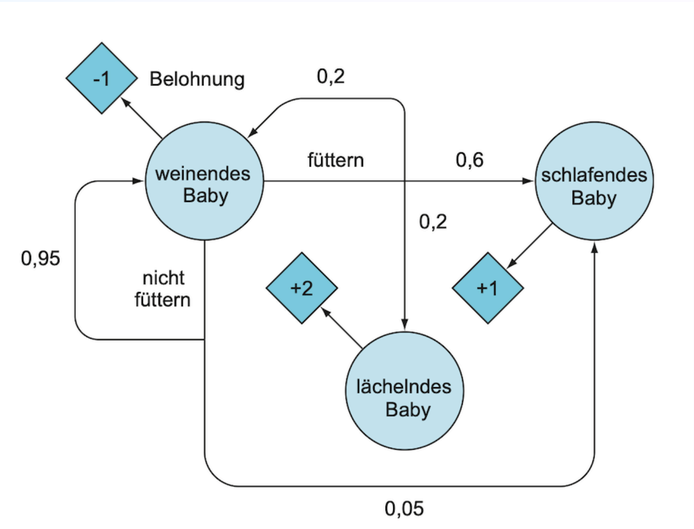
Discount factor 0 nach Schritt 1

v(s)=∑g * p(…)q(a|s) policy diktiert ob wir Aktion wählen

Wert vom Zustand "Weinen" bei "immer füttern"

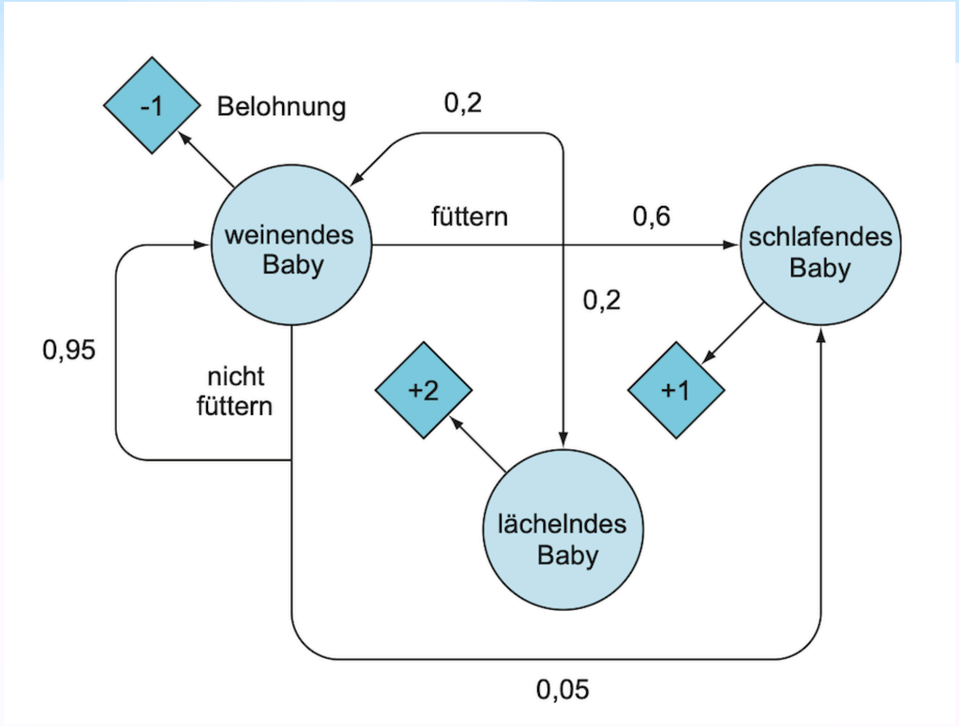
q(f|w)=E(g)=∑p(s',r'|s , a) * r_i
=g * (0.2*2 + 0.2*-1 + 0.6 * 1) + g* …

Endlosschleife weinen => nicht füttern => ...
Ausweg erzwingen: 'abschneiden'
"truncated": nach n Schritten Spiel zu Ende



Aufgabe (fortgeschritten)

```
Was passiert wenn discount Faktor nicht mehr 0 ist, sonder
a) gamma = 1
b) gamma = .9
```



Pause

Aufgaben

Aufgaben Tag 2

a) Markov Decision Processes (MDPs) Markow-Annahme

Die aktuelle 'beste Entscheidung' hängt nicht von der Vergangenheit ab.

Buch: betrachte jedes Kontrollproblem oder jede Entscheidungsaufgabe in der folgenden Liste und prüfe, ob sie von der Vergangenheit abhängt "+" oder Markov-Eigenschaft besitzt "-":

1) **Autofahren** (idealisiert) (+) -
2) Entscheiden, ob in eine **Aktie** investiert/spekuliert werden soll + oder
3) Auswahl einer medizinischen **Behandlung** für einen Patienten +
4) **Diagnose** der Krankheit eines Patienten (idealisiert objektiv)-±
5) Vorhersage, welche Mannschaft in einem **Fußballspiel** gewinnen wird +
6) Auswahl der **kürzesten Route** zu einem Ziel -
7) Mit der Waffe auf ein entferntes Ziel schießen - (idealisiert)

Aufgaben

Aufgaben Tag 2

a) Markov Decision Processes (MDPs)

Buch: betrachte jedes Kontrollproblem oder jede Entscheidungsaufgabe in der folgenden Liste und prüfe, ob sie die Markov-Eigenschaft "+" besitzt oder nicht "-":

1) **Autofahren** (Markov +)
2) Entscheiden, ob in eine **Aktie** investiert werden soll oder nicht -±
3) Auswahl einer medizinischen **Behandlung** für einen Patienten (-)
4) **Diagnose** der Krankheit eines Patienten (±+ bei vollkommenem Wissen)
5) Vorhersage, welche Mannschaft in einem **Fußballspiel** gewinnen wird -
6) Auswahl der **kürzesten Route** zu einem Ziel +
7) Mit der Waffe auf ein entferntes Ziel schießen + (-verziehen)

Pause

Aufgaben

Suche ein weiteres **gym problem** und schreibe policy.

z.B. "Taxi-v3"
https://gymnasium.farama.org/environments/toy_text/taxi/

zähle Belohnung / Gain mit!

Baue erste Schätzung für die **qualities in bandit** ein:

$Q_t(a) = (1 / N_i) \cdot \sum R_i$ (average pro arm!)

z.B. über hash-map qualities={} # (state, action) => (∑rewards, #visits)

💡 damit kann man 'schlaue' policy definieren!

Pause

Markov Decision Processes (MDPs)

`Reinforcement Lösung 1:`

Heuristik: Tabellen

Einfach **Buch führen**, welche Aktionen zum Erfolg geführt haben.

Wir beschreiben fast erst alle Kernideen der Reinforcement-Learning-Algorithmen in ihrer einfachsten Form: nämlich in derjenigen, bei der die Zustands- und Aktionsräume klein genug sind, sodass die Näherungen der Wertfunktionen als **Arrays oder Tabellen** dargestellt werden können. In diesem Fall können die Methoden oft **exakte Lösungen** finden, das heißt, sie können häufig genau die optimale Wertfunktion und die optimale Strategie bestimmen. Dies steht im Gegensatz zu den **Näherungsmethoden**, die später beschrieben werden und nur ungefähre Lösungen finden, dafür aber auf wesentlich größere Probleme effektiv angewendet werden können.

Model

Ein **Modell** ahmt das Verhalten der Umgebung nach oder erlaubt allgemeiner gesagt Rückschlüsse darauf, wie sich die Umgebung verhalten wird. Zum Beispiel könnte das Modell bei gegebenem Zustand und gegebener Aktion den resultierenden nächsten *Zustand* und die nächste *Belohnung* **vorhersagen.**

Dank Deep Learning bilden sich Modelle **automatisch** aus d.h. die Muster welche unsere neuronalen Netze 'erkennen' spiegeln irgendwann idealerweise die Muster der realen Umgebung wieder!

Modellbasierte Methoden verwenden **Planung**, im Gegensatz zu einfacheren modellfreien Methoden, die explizit durch Versuch und Irrtum lernen.

In Chapter 9 we explore reinforcement learning systems that simultaneously learn by trial and error, learn a model of the environment, and use the model for planning.

Model

1) a) **Model**l = Environment (zB gym env) **p**(r, s' | a, s)
2) b) **Model**l WIR versuchen die Environment zu verstehen, Näherung p' an p

Situtation: wir haben das model **p**
dann nennt sich unser Ansatz **model-based**

Situtation: wir haben das model **p nicht** und es interessiert uns nicht
dann nennt sich unser Ansatz **model-free**

Situtation: wir haben das model **p nicht, aber wollen es rekonstruieren**
dann nennt sich unser Ansatz **model-based (estimate p!)**

Pause

- **On-policy:** Die *Daten* werden unter derselben Policy gesammelt, die im Update benutzt wird.
- **Off-policy:** Die Daten werden unter einer anderen Policy gesammelt als der, die im Update verwendet wird.

Off-policy vs supervised learning

Merkmal	Off-Policy RL	Supervised Learning
Ziel	Policy/V-Funktion optimieren durch Rückschau	Direktes Lernen von Input → Output
Datenquelle	Extern (Replay Buffer, Logging, etc.)	Gegeben (i.i.d., annotiert)
Feedback	Delayed (via Belohnung + Dynamik)	Direktes Label y
Objektivität	Wert-/Politikabhängig	Objektiv: „Ground Truth“
Verteilungsprobleme	Verhaltens-Policy ≠ Ziel-Policy	i.i.d. oder kontrolliert
Zielgrößen	Q^π, V^π, π	$f : X \to Y$
Gradient	Temporal Difference	Loss z. B. $\nabla \ell(f(x), y)$

$$J(\pi) = \mathbb{E}_{\tau \sim P(\tau)} \left[\sum_{t=0}^{T} \gamma^t r_t \right]$$

distribution over episodes

$$P(\tau) = P_0(s_0) \prod_{t=0}^{T} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t).$$

$\tau = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}$ trajectory

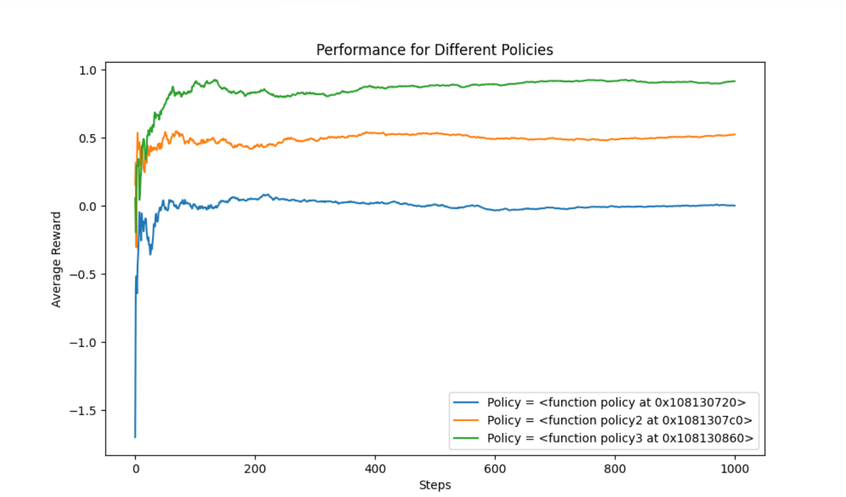
Pause

Aufgabe: Implementiere **Einarmigen Bandit**

🎰 **Bandit-Problem**

- **Umgebung:** Spielhalle mit mehreren Spielautomaten, Typ "**Einarmiger Bandit**"
  - **Zustand**: keiner bzw. konstant
  - **Aktionen**: Auswahl eines Automaten/Hebel
  - **Belohnung**: Sofort, aber stochastisch
  - **Ziel**:
  - a) random **policy** ausführen und Gewinn nach 100 Spielen anzeigen (extra: wohin konvergiert der Wert?)
  - b) irgendeine **<u>andere</u>** policy ausführen und Gewinn nach 100 Spielen anzeigen
  - c) Den Automaten mit der höchsten Auszahlung finden (mehrere Spiele notwendig!)

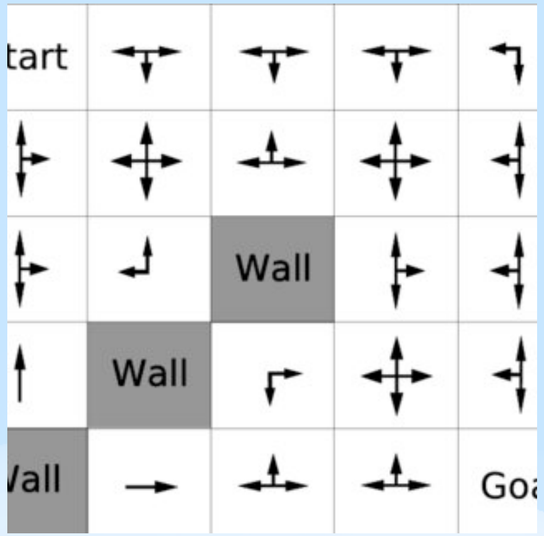
k **Hebel** (engl. *arms*), jeder mit unbekannter Belohnungsverteilung

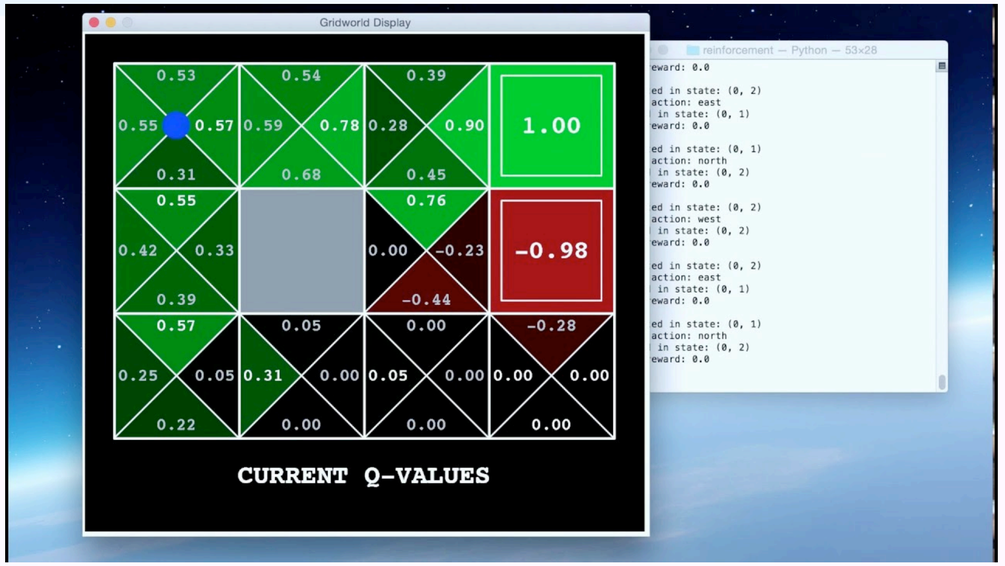


Aufgabe: Implementiere **Grid-World**

📦 Grid-World

- **Umgebung**: Gitter mit Hindernissen und Ziel
- **Zustand**: Position des Agenten
- **Aktion**: Oben, Unten, Links, Rechts
- **Belohnung**: Ziel erreichen = +1, Wand treffen = -1
- **Policy**: Wähle kürzesten Weg zum Ziel anhand von:
- **Value**: Wie gut ist eine bestimmte Position?
- **Quality**: Wie gut ist eine bestimmte Richtung an Position?





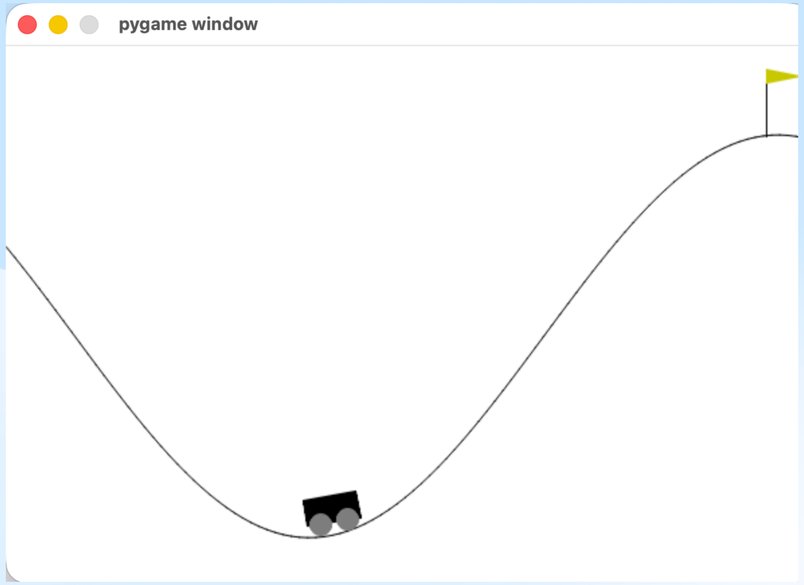
Pause

Aufgabe

https://d.alfanetz.de/index.php/apps/files/files/16611186?dir=/KLR-339/code/tag2

mountainCar-own-policy.py

versucht mit eigener policy die Fahne zu erreichen
(im richtigen moment nach links und rechts beschleunigen)



Wer fertig ist:

Schnappe nächstes Problem auf https://gymnasium.farama.org/environments/classic_control/

Buch

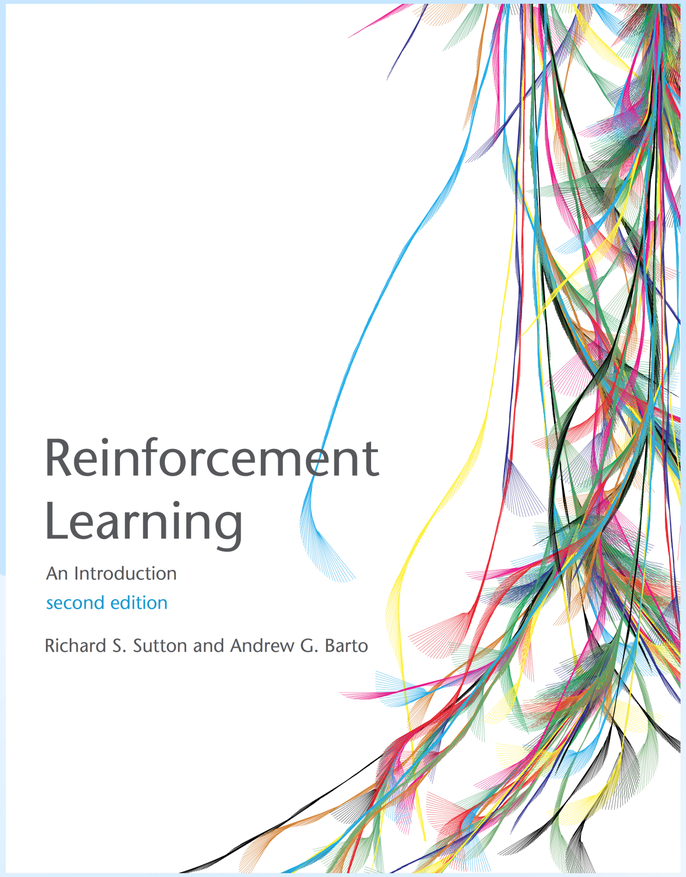
Alexander Zai Brandon Brown

Einstieg in Deep Reinforcement Learning

KI-Agenten mit Python und PyTorch programmieren

Lesen Tag 2:
Teil I: Grundlagen - **Kapitel 2**
Modellierung von Reinforcement-Learning-Problemen: **Markov Decision Processes**

Buch



Sutton: bis Seite 71

Formalisierung der Konzepte

Notation:

- s state aktueller Zustand des Systems / Spiel-Feld / …
- a action, von einem Zustand zum nächsten "Spielzug"
- $\pi(a|s)$ policy Übergangs'Wahrscheinlichkeit' eines Zustandes zum nächsten (zB 100% 'rot')
- R: Reward-/Return-'Funktion', unmittelbare Belohnung einer Aktion a (via π) durch Umgebung
- G: „Goal“ oder „Gain“ Gewinn Summe der Belohnungen $G_t = \sum_k \gamma^k R_{t+k+1}$; Zufallsvariable in:
- V: Value-Funktion, erwarteter Return einer Policy π
- v(s): "Wert" eines Zustands
- v*(s): optimal value "Wahrer Wert" eines Zustands s (unter einer Policy)
- q(a): Qualität (erwartete Belohnung) der **Aktion** a (im Zustand s mit policy π) :
- $q\pi(s, a) = \mathbb{E}\pi[G_t]$ state-action-pair einer Policy
- q*(a): optimal quality "Wahrer Wert" einer Aktion
- $\mathbf{Q}_t$**(a):** Schätzwert von q*(a) zum Zeitpunkt t (“action-values”)
- N_t(a): Anzahl der bisherigen Auswahlvorgänge der Aktion a bis Zeitpunkt t
- H_t(a): Heuristik (heat), erlernte Präferenz zur Auswahl von Aktion a zum Zeitpunkt t ≈> π
- $\boldsymbol{\pi_t}$**(a):** Wahrscheinlichkeit zur Auswahl von Aktion a zum Zeitpunkt t ≈ $P_t(a|s)$

Zentrale Beziehungen:

- $\mathbf{v(s) = \sum_a \pi(a|s)\, q(s, a)}$
  „Wert einer (Schach-)Position unter der Policy“ = “Summe:Werte aller 'qualifizierten' Aktionen”
- $q\pi(s, a) = \mathbb{E}\pi[G_t \mid S_t = s, A_t = a]$ $\quad \mathbf{q_p = \mathbb{E}_p[G_t]}$
  „Wert eines Zugs bei späterem Befolgen der Policy“
- $q\pi(s,a) = \sum_{s',r} p(s', r \mid s, a) \cdot [r + \gamma \sum_a \pi(a|s')\, q(s', a)]$
  (reine Bellman-Gleichung Mittwoch)

Vokabular

Grundlegende Konzepte

- **Agent**: trifft Entscheidungen
- **Aktion (Action) a**: Handlung des Agenten s->**s'** Züge
- **Umgebung (Environment)**: liefert Rückmeldungen (Zustände, Belohnungen)
- **Zustand (State) s**: Beschreibung der aktuellen Situation / Position
- **Belohnung (Reward) r**: Feedback der Umgebung (real, 'abstrakt' oder von uns selbst, Hauptsache quantisierbar, also als Zahl)
  custom reward / reward shaping / intrinsic / extrinsic / sofort / zwischen / final / terminal / sparseπ
- **Policy (π)**: Strategie, die angibt, welche Aktion in welchem Zustand gewählt wird π(s)=a oder π(a | s) = n%
- Spiel = Episode ≈
- **Trajektorie** = Pfad (theoretische Zugabfolge)
- Terminaler Zustand ("Game Over")
- Modell
- `reward` **`shaping`**
- Zug, Runde, Step ≈ eine gewählte Aktion + neuer Zustand

⚠️ Interne Hilfskonzepte / klassische Theorie ⚠️ :

- **Gewinn** *G (goal / gain)* nach Folge von Aktionen (bei policy π) ∑Belohnungen pro Episode / ab Zustand
- **Value Function v**: Erwartete zukünftige Belohnung/Gewinn ab einem **Zustand** ≈
- **Qualität (Quality) q**: Erwartete zukünftige Belohnung/Gewinn für eine **Aktion** in einem Zustand.

Vocabulary

RL as unsupervised learning

MDP

r reward Function, Immediate reward (given action under π) **≠ Return R** ⚠ **:**
G Gewinn “Goal” or “Gain”, random variable used in:
V : Value function, which is the expected Return of a policy π.
v(s) value of state (&policy) ≈ v(s,a) [unused notation] =:
q(a|s) = quality value of action a (at state s) ≈ ∑r = E[G] = ∑ r + v(s)
q*(a) quality true value (expected reward) of action a
Q_t(a) Quality estimate at time t of q*(a) "action-values"
N_t(a) **Number of times action a has been selected up prior to time t**
≈> π policy deterministic π_t(s)=a take action a in state s.
π_t(a|s) probability of selecting action a at time t policy ≈ P_t(a|s) !!
J(π) = **V($s_{t=0}$) expected return of episode / trajectory under policy π**

Vokabular

MDP Markov Decision Process

Episode "ein Spiel", am Ende wird (Haupt)Belohnung vergeben

⚠️ **Nicht verwechseln mit "Trainingslauf" 'Episode' im ML Epoche**

Gewinn

discount factor ɣ

Werte-Funktion v(s) vom state

Qualitäts-Werte q(a,s) von action

Diskret / Kontinuierlich