Reinforcement Learning

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2026-06 Dozent: Karsten Flügge

Themen des Tages

Tag 3

Bellman-Gleichungen

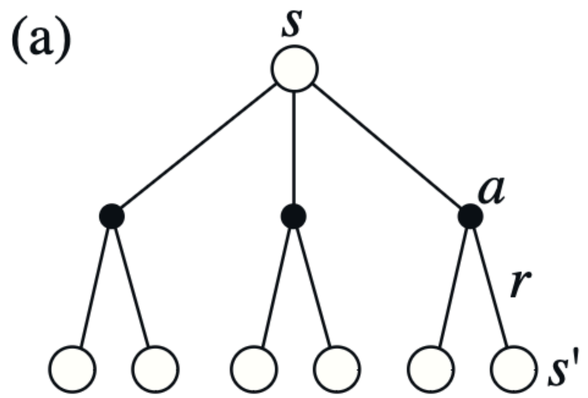
wenn man Umgebung **p** komplett kennt

Dynamic Programming Ansatz

Policy-**Bewertung** (Policy Evaluation)
Policy-**Verbesserung** (Policy Improvement)
Policy-**Iteration** (Policy Iteration)

Praktische Übungen

Backup



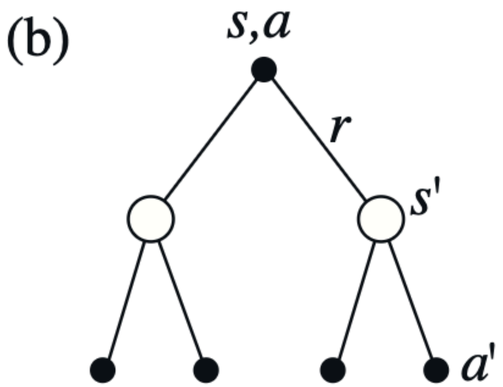


Figure 3.4: Backup diagrams for (a) v_π and (b) q_π.

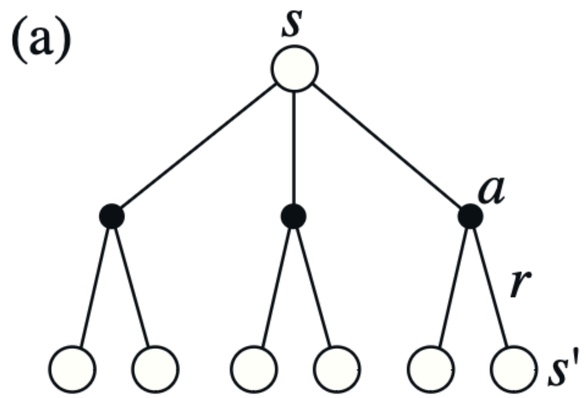
<u>Backup</u> "ermitteln des Wertes einer Position / Aktion anhand der <u>Zukunft!</u>"

Berechnung von v und q ist im wesentlichen gleich, ausser dass man zur Bewertung eines States s im state s anfängt und bei einer Aktion a im state-value-pair s,a.

Wenn man v oder q mit backup verbessert nennt sich dass <u>Update</u>.

💡 bei deterministischem MDP ist b) am Anfang nur <u>eine</u> Kante!

Backup



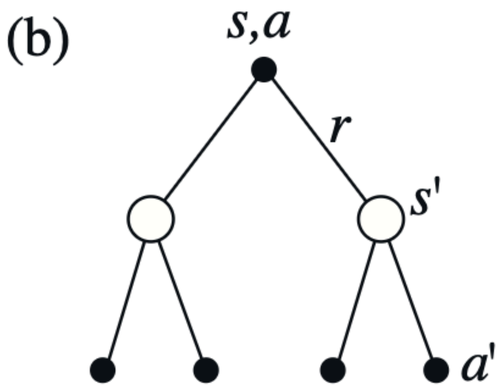


Figure 3.4: Backup diagrams for (a) v_π and (b) q_π.

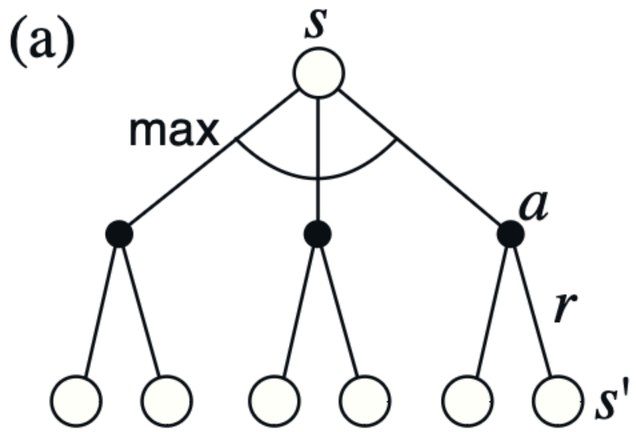
Zwei Zufallseinflüsse

s -> a Aktion Wählen Zufallswahrscheinlichkeit? policy Agent $\pi(a|s)$

a -> s Mit Aktion in neuem Zustand landen, Zufallswahrscheinlichkeit?

Belohnung r und nächster State s' werden 'zufällig' von Umgebung ausgeliefert
p(r,s' | s,a)

Backup



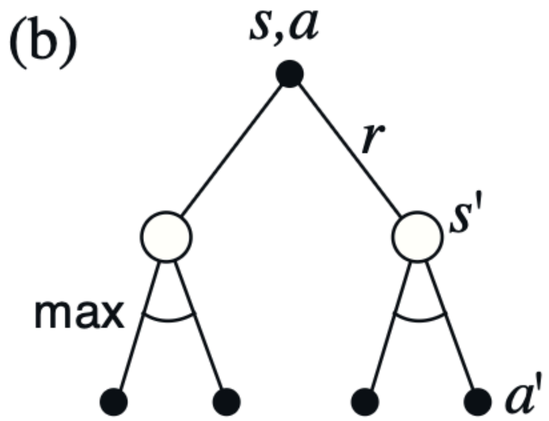
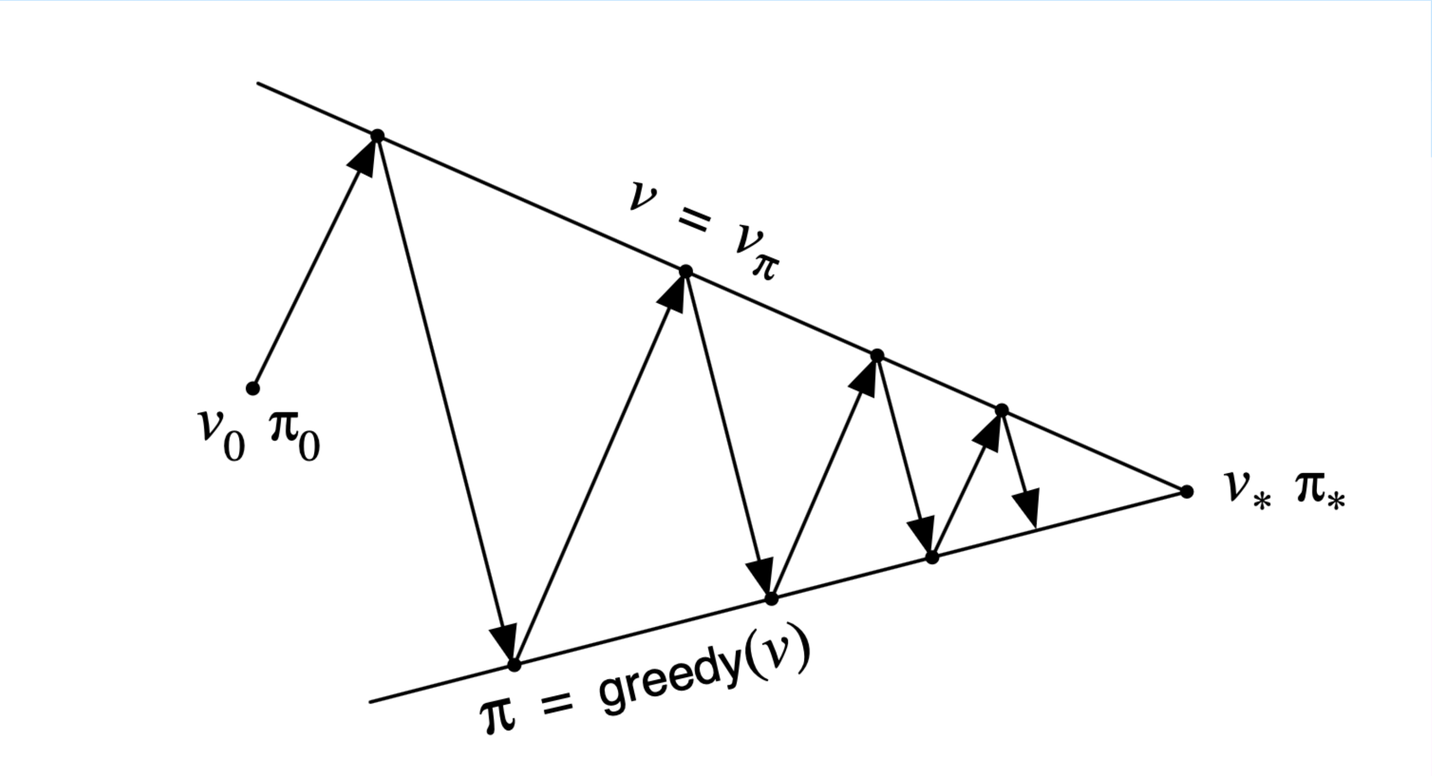


Figure 3.7: Backup diagrams for (a) v_* and (b) q_*

Iterative Näherung π*

Im folgenden werden wir sehen wie wir uns der optimale Strategie π^* nähen können. (*Bei vollständigem Modell der Umgebung)



Philosophisches Framework

In der Zeit bevor Computer auch komplexe Probleme schnell lösen konnten war **Mathematik** das beste Werkzeug. Tatsächlich lassen sich heute nur eine kleine Anzahl von Reinforcement Learning Problemen **exakt / analytisch** lösen.

Ähnlich wie bei der *Error-F*unktion bei neuronalen Netzen, welche wir auch nach eigenem Gutdünken und eigener **Kreativität** frei modifizieren können, dienen die folgenden Formeln also eher als **historische Motivation** oder **Leitfaden** denn als strict zu befolgende Programmieranweisungen.

💡 Viele Entscheidungsverfahren lassen sich als **annähernde** Lösungen der Bellman-Gleichung verstehen:

Heuristische Suche, dynamische Programmierung, Monte Carlo, TD-Learning etc.

Bellmann Gleichungen

Es hat sich schon angedeutet dass die Begriffe **Belohnung, Wert** eines Zustandes und **Qualität**(Wert) einer Aktion sowie eine zugehörige *greedy* **policy** eng miteinander gekoppelt sein können.

Wenn wir mit einem neuen Problem konfrontiert sind und nicht allwissend, dann stehen wir vor dem Problem dass wir zum finden der optimalen Strategie (policy) den Wert unserer Zustände und unserer Aktionen **schätzen** müssen.

Die **Bellmann Gleichung** hilft uns genau beim Lösen dieser Aufgabe:
Wir können **rekursiv** die optimalen **Wertefunktion** schätzen.

Gewinn

"**g**ain is discounted **s**um of **r**ewards"

$G_t = \sum_k \gamma^k r_{k+1}$

$G_t = r_0 + \sum_{k=1} \gamma^k r_{k+1}$ k=0 => γ^0 = 1

$G_t = r_0 + \gamma G_{t+1}$ rekursiv: gain = aktueller reward plus zukünftige

γ zwischen 0 und 1
steuert kurzsichtiges / langfristiges Verhalten

Gewinn rekursiv

$G_t = \mathbf{r_0} + \boldsymbol{\gamma} G_{t+1}$ rekursiv: gain = aktueller reward plus zukünftige

$v(s) := E(G) = \sum_a \boldsymbol{\pi}(a \mid s)\, q(a)$ **value**
je nach policy wählen wir andere Aktionen

$q(a_{,s}) := E(G) = \mathbf{r(a)} + \sum_{s'} \mathbf{p}(s' \mid a_{,s})\, v(s')$ **quality** (action-value)
je nach Environment p landen wir in anderen States

Beides Erwarteter Gewinn, aber mit anderem Start.

$v(s) = \sum_a \boldsymbol{\pi}(a \mid s) \sum_{s'} \mathbf{p}(s' \mid a)\, v(s')$

Gewinn rekursiv

$G_t = \mathbf{r_0 + \gamma G_{t+1}}$ rekursiv: gain = aktueller reward plus zukünftige

v(s) = E(G) = ∑π(sla) **q(a)**
v(s) = E(G) = E($\mathbf{r_0 + \gamma G_{t+1}}$) = E($\mathbf{r_0}$) + E($\mathbf{\gamma G_{t+1}}$)
v(s) = ∑π(s l a)*$\mathbf{r + \gamma}$*∑p(s' l a) v(s')
v(s) = ∑π∑p **[r + γ*v(s')]**

je nach policy wählen wir andere Aktionen

q(a) = E(G) = ∑p(s' l a) v(s')
je nach Environment p landen wir in anderen States

Erwartungswert

`Was heisst eigentlich` **`erwarteter Gewinn?`**

$G_t = \mathbf{r_0} + \gamma G_{t+1}$ rekursiv: gain = aktueller reward plus zukünftige

Reward ist i.d.r nicht deterministisch sondern hängt vom Zufall ab:
Zufällige Aktion via π und zufälliger State via p (Umgebung)

Erwartungswert ≈ Mittelwert ≈ Schwerpunkt

E(X) = ∑p(x)*x alle denkbaren Werte mal deren W-Keit $\int p(x)*x\,dx$

E(Würfel) = 1/6*1+1/6*2+1/6*3+1/6*4+1/6*5+1/6*6 = 3.5
E(Bandit) = bei uns Gleichverteilung random()*max = max/2

v(s) = E(G) = ∑π(a | s) q(a)
v(s) = **E(G)** = $E(\mathbf{r_0} + \gamma G_{t+1})$

Bellmann Gleichungen

Ziel: Schätzungen rekursiv verbessern.

q(a) = … soll geschätzt werden.
q(a) = erste Schätzung haben wir
q(a) = q(a) + update Bewertung von action-state pair (a|s) unter einer Policy

$$q(s, a) = \sum gamma * r$$

v(s) = … gespiegelt Bewertung von state (s) unter einer Policy

$$v(s) = \sum_a \pi(a|s)\, q(s, a)$$

Bewertung von state = Summe der Bewertungen

Summe der erwarteten Gewinne

Decision Tree ≈
Markov Decision Process

Idee: verbinde beides zu
rekursiver Formel.

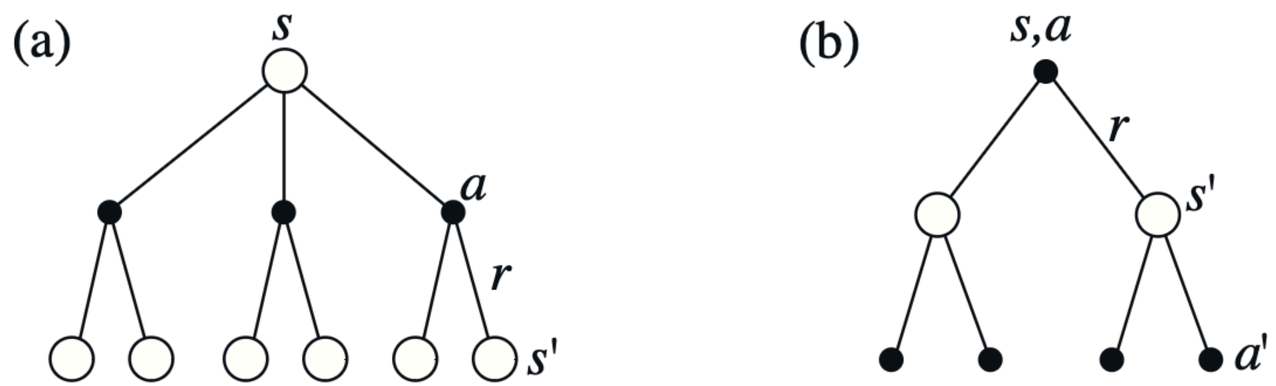


Figure 3.4: Backup diagrams for (a) v_π and (b) q_π.

Gleichungen

Vergleich

Decision Tree ≈
Markov Decision Process

Vorausblick

Wir können die **Bewertung schätzen** auch wenn wir sie noch nie gesehen haben.
Wir können den "**Rest des Baumes**" mit neuronalen Netzen '**abschneiden**'

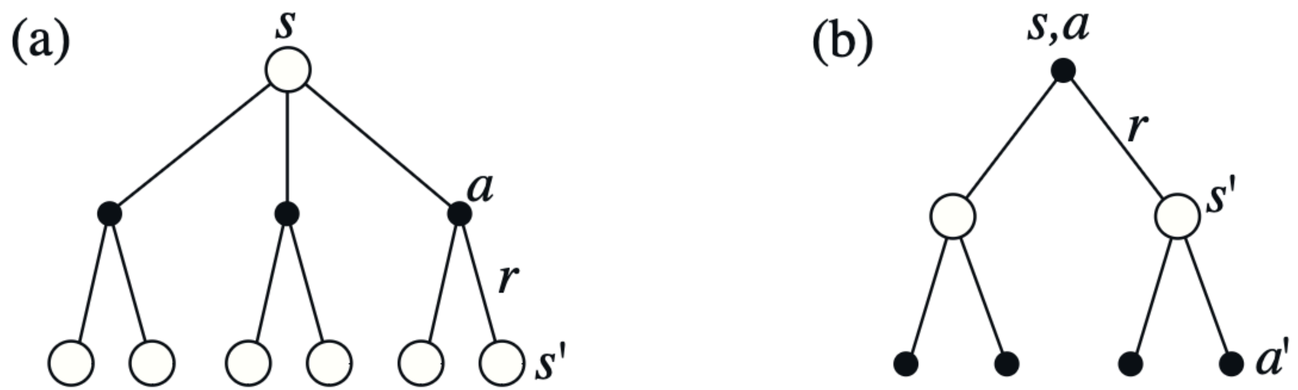


Figure 3.4: Backup diagrams for (a) v_π and (b) q_π.

Bellman vs Baby

Bei Bellman überspringt man immer die mittlere Schicht, dass heisst wir gehen
von State s zu State s'   oder
von Action a zu Action a'

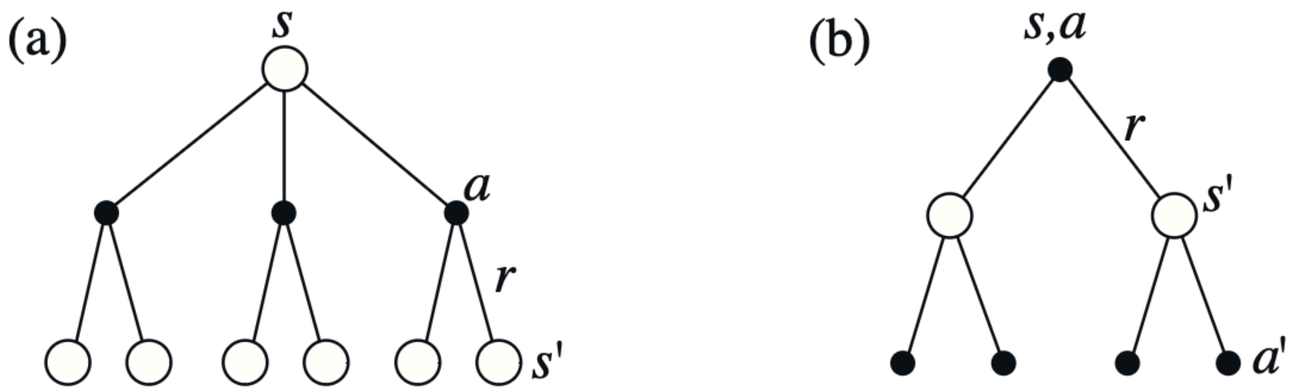


Figure 3.4: Backup diagrams for (a) v_π and (b) q_π.

Bellmann Gleichungen

Es hat sich schon angedeutet dass die Begriffe **Belohnung, Wert** eines Zustandes und **Qualität**(Wert) eine Aktion sowie eine zugehörige *greedy* **policy** eng miteinander gekoppelt sein können.

Die **Bellmann Gleichungen**
- optimalen **Wertefunktion** eines *Zustandes* schätzen oder
- optimalen Wertefunktion "**Qualität**" einer *Aktion* schätzen oder

REKURSIVE Beziehung v(s) über s' :

$$V_p(s) = \sum_a \pi(a|s) \sum_r \sum_{s'} p(s',r|s,a) * [r + \gamma V_p(s')]$$

$$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(s')] \quad \text{vereinfacht}$$

Wir updaten unsere value-Schätzung anhand des **r**ewards und aller nächsten value Schätzungen für alle denkbaren **a**ctions und deren folge-**s**tates. (gemäß deren **p**robabilty unter policy π)

s' = nächster state
r = aktueller reward

∑ dreifache Summe über alle Aktionen, rewards und next states

value of state = "Erwartungswert über aktuelle Belohnung plus (γ diskontierter) zukünftiger Wert"

💡 **der kompliziert aussehende formale Ausdruck p(s',r | s,a) ist in Wirklichkeit nur die Wahrscheinlichkeit für eine Belohnung (bei Wahl einer Aktion A im Zustand s)**

💡 **Wie auch schon bei der Gewinn Funktion hängt der aktuelle Erwartungswert von zukünftigen Erwartungswerten ab:**

wenn unsere Position *in zwei Zügen* zu schachmatt führt dann ist unsere Position *jetzt* hervorragend!

Sutton p. 71

Bellmann Gleichungen

Klassisch / Analytisch

$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(s')]$ **vereinfacht**

s' = nächste Zustände

Das ergibt bei fester policy und festem v gibt diese Gleichung
ein System von Linearen Gleichungen (für alle s)

Das liesse sich analytisch lösen mit Gaußschem Eliminierungsverfahren.

Jedes $V_p(s)$ ist eine unbekannte.

Einfaches Bellmann

Vereinfacht wenn determisistisch

$v(s) = \sum_a \boldsymbol{\pi}(a \mid s) \, [\, r + \gamma \cdot v(\mathbf{s}') \,]$

Noch einfacher, ein Schritt und eine Aktion zur Zeit
$q(a,s) = r + \gamma \cdot \text{argmax}\, q_a(\mathbf{s}')$

Optimale Strategien und optimale Wertfunktionen

3.6 Optimale Strategien und optimale Wertfunktionen

Partielle Ordnung über Strategien:

$\pi \geq \pi' \Leftrightarrow v_p(s) \geq v_p'(s) \quad \forall s \in S$

Optimale Strategie π^*, wenn $\pi^* \geq \pi$ für alle π

Alle optimalen Strategien teilen sich die gleiche optimale Zustandswertfunktion:

$v^*(s) := \max_p v_p(s)$ über *alle denkbaren* policies (also sehr theoretisch!)

💡 mit dem Stern * markieren wir stets die **optimal** Eigenschaft für optimale Werte optimale Qualität oder optimale Policy.

💡 Das theoretische Optimum werden wir meist nie ganz erreichen, aber genau dafür sind die Gleichungen da:
dass wenn wir unsere Strategie lange genug verfolgen wir uns theoretisch immer mehr der optimalen Lösung nähern!

Bellmann Gleichungen

Die **Bellmann Gleichungen**
- optimalen **Wertefunktion** eines *Zustandes* schätzen oder
- optimalen Wertefunktion "**Qualität**" einer *Aktion* schätzen oder

REKURSIVE Beziehung v(s) über s':

$$
\begin{aligned}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_*(s')\Big], \text{ or} \qquad (4.1) \\
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big], \qquad (4.2)
\end{aligned}
$$

bei greedy π = policy "best action"

Bellman-Optimalitätsgleichung

Bellman-Optimalitätsgleichung für q*:

$$\mathbf{q^*(a} \mid s) = \sum p(s', r \mid s,a) * [\mathbf{r} + \gamma \cdot \max_{a'} \mathbf{q^*}(s',a')]$$

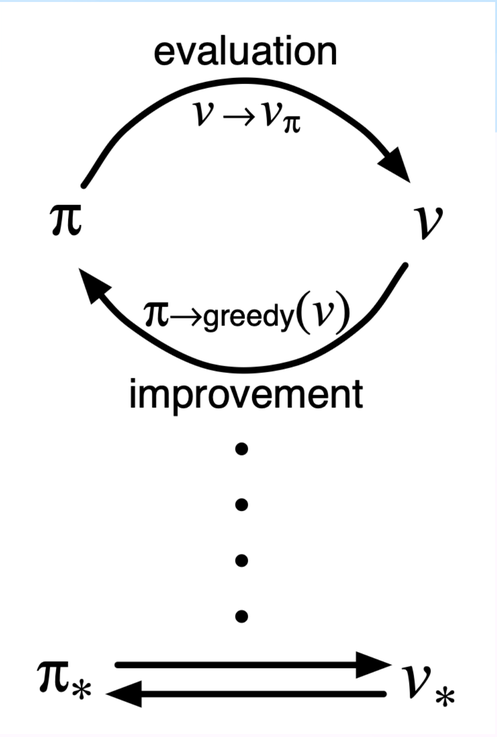
"Eine **Aktion** ist **optimal** wenn die **Summe** aller **folgenden** möglichen **Belohnungen maximiert** wird"
(inklusive direkter Belohnung r)

Bellman-Optimalitätsgleichung für v*:

$$\mathbf{v^*}(s) = \max_a \sum p(s',r \mid s,\mathbf{a}) * [r + \gamma v^*(s')]$$

"Ein **Zustand s ist optimal** durch eine **beste Aktion** welche die Summe aller folgenden möglichen Belohnungen maximiert"

… wird im folgenden iterativ approximiert "dynamic programming"



Pause

Dynamic Programming Ansatz

Dynamische Programmierung

Iterative Lösung der Bellman-Gleichung unter Kenntnis des Modells (Transitionen, Belohnungen)

Der Begriff der **dynamischen Programmierung** (**DP**) bezieht sich auf eine Sammlung von Algorithmen, die verwendet werden können, um optimale Strategien zu berechnen, vorausgesetzt, es liegt ein **perfektes Modell** der Umgebung in Form eines Markov-Entscheidungsprozesses (MDP) vor. Klassische DP-Algorithmen sind im Bereich des Reinforcement Learnings nur begrenzt nützlich, sowohl wegen ihrer Annahme eines perfekten Modells als auch wegen ihres hohen Rechenaufwands, aber sie sind theoretisch dennoch von Bedeutung.

⚠️ der Begriff der dynamischen Programmierung im Kontext von RL ist verwandt aber ungleich der Definition in anderen Programmierkontexten, wo es hauptsächlich darum geht die **Zwischenergebnisse festzuhalten** und zu verwerten, statt immer wieder neu zu berechnen. (**Table der vorherigen Lösungen**)

Dynamic Programming Ansatz

⚠️ der Begriff der dynamischen Programmierung im Kontext von RL ist verwandt aber ungleich der Definition in anderen Programmierkontexten, wo es hauptsächlich darum geht die **Zwischenergebnisse festzuhalten** und zu verwerten, statt immer wieder neu zu berechnen. (Table der vorherigen Lösungen)

bereits_berechnet={}
fib(n) = f(n-1) + f(n-2) # rekursion

```
def fib(n):
  if(n<=1)return 1
  if bereits_berechnet[n-1]:
    a=bereits_berechnet[n-1]
  else:
    a = bereits_berechnet[n-1] = fib(n-1)
  if bereits_berechnet[n-2]:
    b=bereits_berechnet[n-2]
  else:
    b = bereits_berechnet[n-2] = fib(n-2)
  return a+b
// 'entrekursivierung' O(n^2) => O(n)
```

DP : Bootstrapping

Bootstrapping: wir fangen mit vollständig unbekannten Schätzungen an, also alles null :
v(s)=0 q(a)=0 (💡bootstrapping ≈ Schuhe einschnüren von unten) "Von Null anfangen"

Das wird dann **rekursiv verfeinert**!

Also in der ersten iteration fällt V_p(s') weg (=0)

$$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(s')]$$

Erste echte Schätzung:

$$V_p(s)_1 = \sum \pi(a) * p * r$$ einfach Summe über alle rewards

Aktualisiere **Schätzungen** der Werte(funktion) teilweise **basierend auf** vorherigen erlernten **Schätzungen**, ohne auf ein endgültiges Ergebnis zu warten.

Aber: *hier* vollständige Wahrscheinlichkeitsverteilungen aller möglichen Übergänge erforderlich!

Policy-Bewertung (Policy Evaluation)

Bei **fester policy π** wird **v** Schätzung iterativ verbessert

Bootstrapping: wir fangen mit vollständig unbekannten Schätzungen an, also alles null :
v(s)=0 q(a)=0 (💡bootstrapping ≈ Schuhe einschnüren von unten) "Von Null anfangen"

Das wird dann **rekursiv verfeinert**!

Also in der ersten iteration fällt $V_p(s')$ weg (=0)

$$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(s')]$$

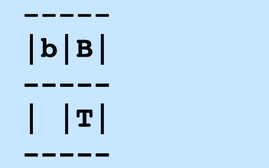
Erste echte Schätzung:

$$V_p(s)_1 = \sum \pi(a) * p * r$$ einfach Summe über alle rewards

Aktualisiere **Schätzungen** der Werte(funktion) teilweise **basierend auf** vorherigen erlernten **Schätzungen**, ohne auf ein endgültiges Ergebnis zu warten.

Aber: *hier* vollständige Wahrscheinlichkeitsverteilungen aller möglichen Übergänge erforderlich!

Bellmann Beispiel Grid World 2*2



Bewerte Positionen b (schlechter begin), B (guter Begin) Ziel: T Terminal

policy π = random 25% ← ↑ → ↓

reward r = -1 bei runterfallen (p=1 100%)
reward r = 1 bei erreichen von Terminal state T (p=1 100% reward)
reward r = 0 sonst

γ = 0 : greedy

$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(s')]$ vereinfachte Berechnung der Werte (value) von states

p(r,s'|a,s) = Wahrscheinlichkeit für reward und state bei aktion fällt bei uns komplett zusammen auf Wert 0 oder 1

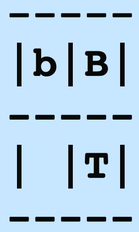
π(a) = 1/4 Wahrscheinlichkeit nach ← ↑ → ↓ zu gehen
p = 0 oder 1 Wahrscheinlichkeit für rewards (-1,1,0) je nachdem wo man hingeht

Trivial greedy

```
v(s=b "begin") = ∑ 0.25 * p [ r + 0 ]   /*          + 0 wegen γ = 0 und wegen bootstrapping v(s)=0 */
               = 1/4 * 1 * -1 /*oben*/   +  1/4 * 1 * -1 /*links*/
              +  1/4 * 1 * 0  /*rechts*/ +  1/4 * 1 * 0  /*unten*/ = -1/2

v(s=B "guter Begin") = ∑ 0.25 * p [ r + 0 ]
                       = 1/4 * 1 * -1 /*oben*/ + 1/4*1*-1 /*rechts*/
                         +1/4 * 1 * 0 /* links zu b */ + 1/4 * 1 * 1 /* unten ZIEL T */ = -1/4
```

Bellmann Beispiel Grid World 2*2



Bewerte Positionen b (schlechter begin), B (guter Begin) Ziel: T Terminal

$$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(s')]$$

Wenn wir Positonen bewertet haben (erst mit random policy),
ergibt sich daraus ein Pfad: Folge immer der nächstbesten Position

Kanonische Policy π : wähle Aktion welche zur nächstbesten Position führt
$\pi(a)=\text{argmax } v(s')$ "greedy"

Bellmann Beispiel Grid World 2*2

```
-----
|b|B|   Bewerte Positionen b (schlechter begin), B (guter Begin) Ziel: T Terminal
-----
| |T|
-----
```

policy π = random 50% ↑ ↓

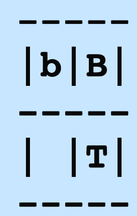
reward r = -1 bei runterfallen (p=1 100%)
reward r = 1 bei erreichen von Terminal state T (p=1 100% reward)
reward r = 0 sonst

γ = 0 : greedy

$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(\underline{s'})]$ vereinfachte Berechnung der Werte (value) von states
p(r,s'|a,s) = Wahrscheinlichkeit für reward und state bei Aktion fällt bei uns komplett zusammen auf Wert 0 oder 1

$\pi(a) = 1/2$ Wahrscheinlichkeit nach ← ↑ → ↓ zu gehen
p = 0 oder 1 Wahrscheinlichkeit für rewards (-1,1,0) je nachdem wo man hingeht

Bellmann Beispiel Grid World 2*2



policy2 π = random 50% ↑ ↓ ändert sich was an value Funktion?
reward r = -1 bei runterfallen (p=1 100%ig)
reward r = 1 bei erreichen von Terminal state T (p=1 100%ig reward)

γ = 0 : greedy

$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(s')]$ **vereinfacht**

Trivial greedy

$$v(s=b \text{ "begin"}) = \sum 0.5 * p\,[\, r_i + 0\,]$$
$$= 1/2 * 1 * -1 \text{ /*oben*/}$$
$$+ 1/2 * 1 * 0 \text{ /*unten*/} = -1/2$$

$$v(s=B \text{ "guter Begin"}) = \sum 0.5 * p\,[\, r + 0\,]$$
$$= 1/2 * 1 * -1 \text{ /*oben*/}$$
$$+ 1/2 * 1 * 1 \text{ /* unten ZIEL T */} = 0$$

besser unter policy2

Bellmann Beispiel Grid World 2*2

```
 -----
 |b|B|
 -----
 | |T|
 -----
```

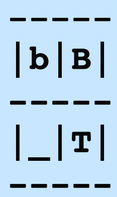
Initialisierung v(s)=0 Wert für ALLE states s ist unbekannt daher erstmal 0. v(s=runterfallen)=0
policy = random 25% ← ↑ → ↓
reward = -1 bei runterfallen (p=1 100%ig)
reward = 1 bei erreichen von Terminal state T (p=1 100%ig reward)

$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(\underline{s'})]$ vereinfacht
$\gamma = 1$: 'planend'

v(s=b "begin") = $\sum$ 0.25 * p [r + 1*$V_p(\underline{s'})$]
= 1/4 * 1 * [-1 + 1* $\underline{0}$] /*oben*/
+ 1/4 * 1 * [-1 + 1*0] /*links*/
+ 1/4 * 1 * [0 + 1*$V_p(\underline{B})$] /* rechts */ + 0 /*unten*/ = -1/2

v(s=B "guter Begin") = $\sum$ 0.25 * p [r + 0]
= 1/4 * 1 * -1 /*oben*/ + 1/4*1*-1 /*rechts*/
+1/4 * 1 * 0 /* links zu b */ + 1/4 * 1 * 1 /* unten ZIEL T */ = -1/4

Bellmann Beispiel Grid World 2*2



Initialisierung v(s')=0 Wert für ALLE states s ist unbekannt daher erstmal 0.
v(s=runterfallen):=0 policy = random 25% ← ↑ → ↓
was passiert eigentlich beim runterfallen genau? Terminaler state x mit v(x):=0

$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(\underline{s'})]$ vereinfacht
γ = 1 : 'planend'

Erster Schritt: v(s')=0
Zweiter Schritt: wir haben v(b)=-1/2 und v(B)=-1/4 v(r)=v(runterfallen):=0
v(T):=0 (1?)

$v(s=b \text{ "begin"}) = \sum 0.25 * p [r + 1 * V_p(\underline{s'})]$
$= 1/4 * 1 * [-1 + 1 * v_p(r)]$ /* oben v(s=runterfallen):=0! */
$+ 1/4 * 1 * [-1 + 1 * v_p(r)]$ /* links */
$+ 1/4 * 1 * [0 + 1 * V_p(\underline{B})]$ /* rechts */
$+ 1/4 * 1 * [0 + 1 * V_p(T)]$ /* unten */
$= -1/4 + -1/4 + 1/4*1*[0 + 1*(-1/4)] + 1/4*1*[0+1*-1/4] = -1/4 -1/4 - 1/8 - 1/8 = -3/4$

Bellmann Beispiel Grid World 2*2

```
-----
|b|B|
-----
|_|T|
-----
```

Initialisierung v(s')=0 Wert für ALLE states s ist unbekannt daher erstmal 0. v(s=runterfallen)=0 policy = random 25% ← ↑ → ↓
was passiert eigentlich beim runterfallen genau? Terminaler state x mit v(x):=0 oder -100 v(T):=0 oder 100

$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(\underline{s'})]$ vereinfacht
$\gamma = 1$: 'planend'

Erster Schritt: v(s')=0
Zweiter Schritt: wir haben v(b)=-1/2 und v(B)=-1/4 v(r)=v(runterfallen):=0
v(T):=0 (1?)

v(s=B "guter Begin") = $\sum$ 0.25 * p [r + 1 * $V_p(\underline{s'})$]
= 1/4 * 1 * [-1 + 1 * v_p(r)] /* oben v(s=runterfallen)=0! */
+ 1/4 * 1 * [0 + 1 * v_p(b)] /* links */
+ 1/4 * 1 * [-1 + 1 * 0] /* rechts */
+ 1/4 * 1 * [1 + 1 * V_p(T)] /* unten */ v(s=Terminal) sei 0
= -1/4 + 1/4 * 1 * [0 + 1*(-1/2)] + -1/4 + 1/4*(1+0) = -1/4 -1/8 - 1/4 + 1/4 = -3/8 > -3/4

Bellmann Beispiel Grid World 2*2

```
-----
|b|B|
-----
| |T|
-----
```

Initialisierung v(s')=0 Wert für ALLE states s ist unbekannt daher erstmal 0. v(s=runterfallen)=0
policy = random 25% ← ↑ → ↓

$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(s')]$ vereinfacht
$\gamma = 1$: 'planend'

Erster Schritt: v(s')=0
Zweiter Schritt: wir haben v(b)=-1/2 und v(B)=-1/4 v(r)=v(runterfallen):=0 v(T):=0 (1?)

v(s=B "guter Begin")
$= \sum 0.25 * p [r + 1*V_p(s')]$
= ↑ + ← + → + ↓
= 1/4*-1 + 1/4*[0 + -1/2] + 1/4*-1 + 1/4*[1 + 1*0]
= -1/4 - 1/8 + -1/4 + 1/4 = -3/8

B ist immer noch besser, aber unser Problem ist etwas schlecht formuliert mit v(runterfallen) = v(T)
Basierend auf unseren Schätzwerten v können wir jetzt einen neue policy definieren $\pi(a|s) = \text{argmax}_a\ v(s')$
s'=state after taking a

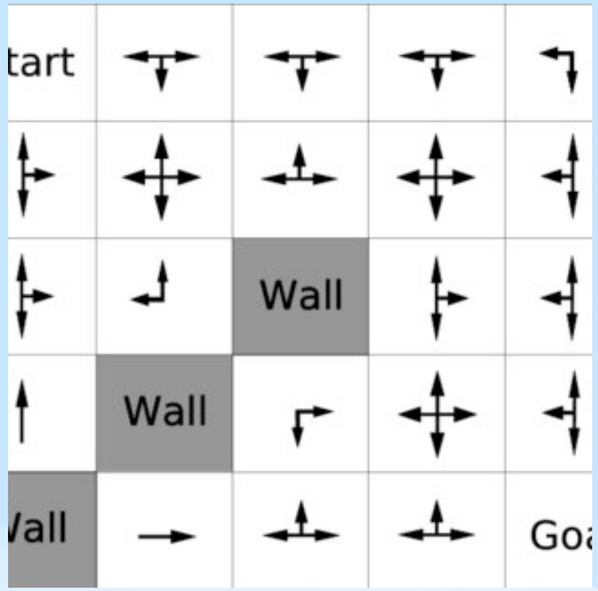
Dynamic Programming Ansatz

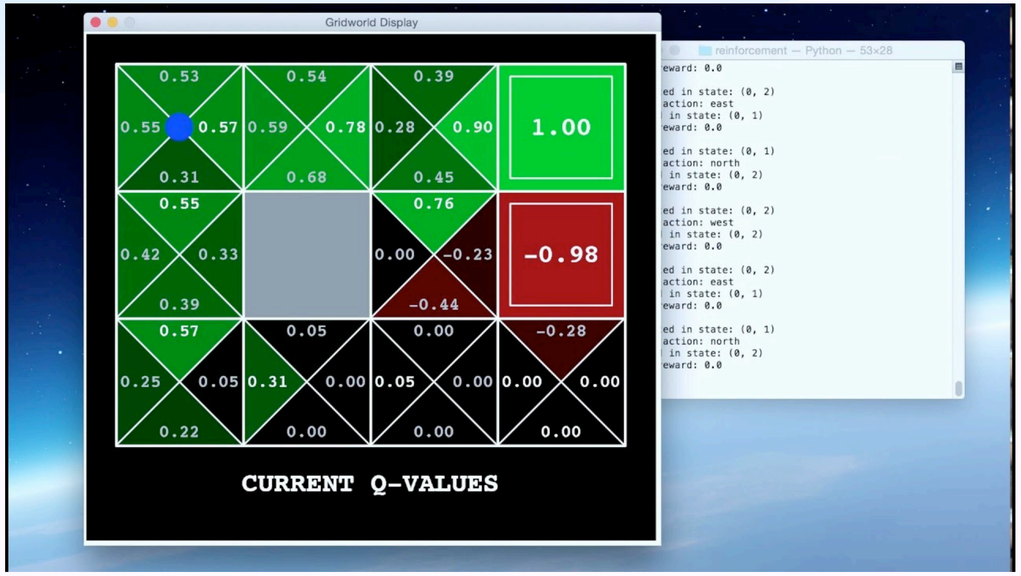
Dynamische Programmierung (ab S. 73/89 im Sutter)
4.1 Policy-**Bewertung** (Policy Evaluation) — S. 74/90
4.2 Policy-**Verbesserung** (Policy Improvement) — S. 94
4.3 Policy-**Iteration** (Policy Iteration) — S. 96
4.4 Wertiteration (Value Iteration) — S. 98
4.5 Asynchrone dynamische Programmierung — S. 101
4.6 Generalized Policy Iteration
4.7 Effizienz der dynamischen Programmierung

Aufgabe: Implementiere **Grid-World**

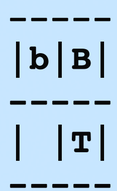
📦 Grid-World

- **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?





Bellmann Beispiel Grid World 2*2



policy π = random 25% ← ↑ → ↓
reward r = -1 bei runterfallen (p=1 100%ig)
reward r = 1 bei erreichen von Terminal state T (p=1 100%ig reward)
reward r = 0 sonst

γ = 0 : greedy

$V_p(s) = \sum \pi(a) * p * [r + \gamma V_p(\underline{s'})]$ vereinfachte Berechnung der Werte (value) von states

$\pi(a) = 1/4$ Wahrscheinlichkeit nach ← ↑ → ↓ zu gehen
p = 0 oder 1 Wahrscheinlichkeit für rewards (-1,1,0) je nachdem wo man hingeht

Trivial greedy
v(s=b "begin") = ∑ 0.25 * p [r + 0]
= 1/4 * 1 * -1 /*oben*/ + 1/4 * 1 * -1 /*links*/
+ 1/4 * 1 * 0 /*rechts*/ + 1/4 * 1 * 0 /*unten*/ = -1/2

1/4 * 1 * -1 /*oben*/ "wir gehen random 25% nach oben und kriegen 100% die Strafe -1"

Dynamic Programming Ansatz

4.1 Policy-Bewertung (**Policy Evaluation**)

Wir nähern uns iterativ (Schritt k)
der wahren Wertefunktion v_π einer Policy π

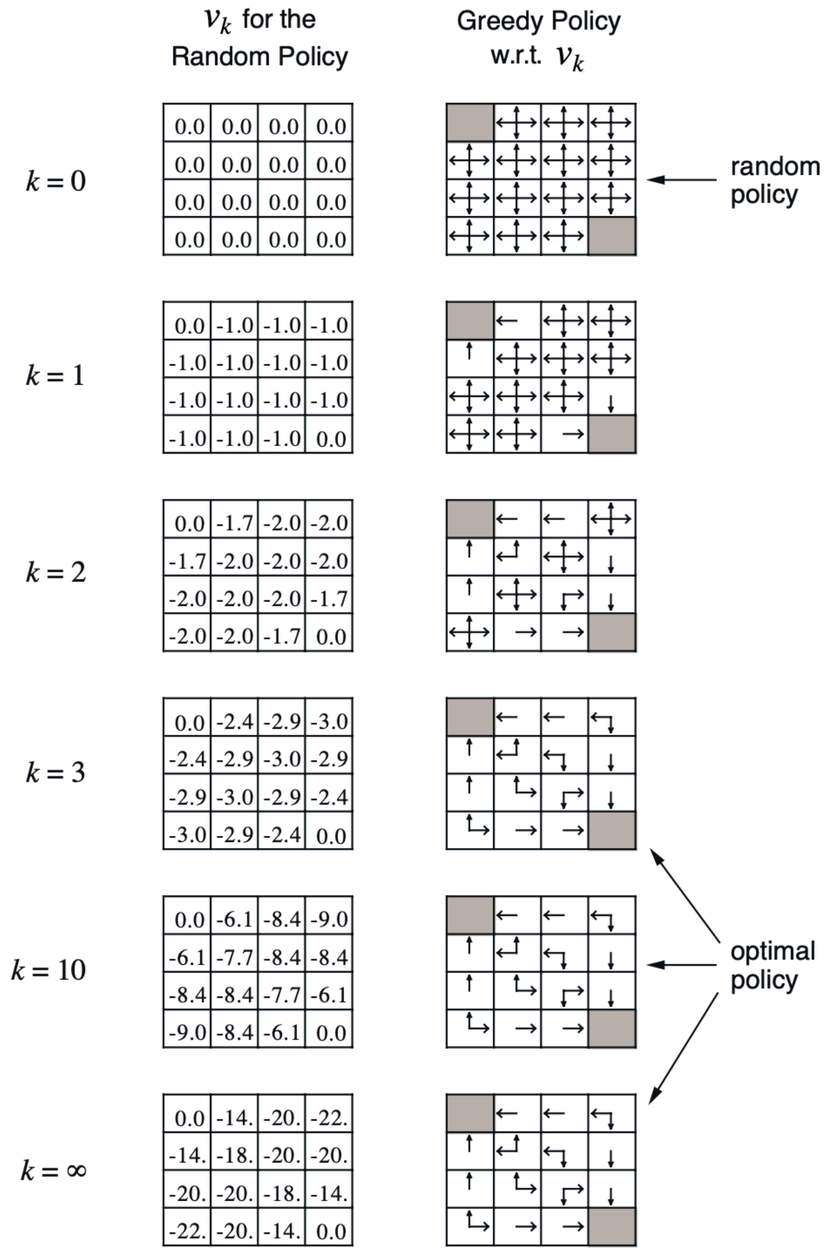
$$v_{k+1}(s) = E\pi[R_{t+1} + \gamma \cdot v_k(S_{t+1}) \mid S_t = s]$$

$$v_{k+1}(s) = \sum_a \pi(a \mid s)\, p(s', r \mid s, a)\, [\, r + \gamma \cdot v_k(\mathbf{s'}) \,]$$

Ein Algorithmus mit zwei 'flavors':

Dieser Ansatz ist **off-Policy wenn** wir v_k *nur* mithilfe der original Policy π verbessern
Dieser Ansatz ist **on-Policy wenn** wir v_k mithilfe einer **neuen Policy π'** verbessern:

Basierend auf dieser Schätzung können wir eine neue Policy π'
definieren: **$\pi'(s) = \text{greedy}(v_k) = \text{argmax}_a\, v_k(s,a)$** besten Nachbarn wählen



Policy-Evaluation

4.1 Policy-Bewertung

eine policy ist 'besser' als eine andere
$\pi >= \pi'$

wenn die erwarteten Werte für alle states besser sind
$v_\pi >= v_\pi'$

Wir verbessern policy π indem wir den besten States / Aktionen **folgen!**

$\pi(a|s) := \text{argmax}(a)\ v(s')$

Policy-Verbesserung (Policy Improvement)

Bei **fester** vorheriger Schätzung **v** wird **policy** iterativ **verbessert**

$\pi(s) := \text{argmax}_a \; R(s,a) + \gamma * V(s')$

Policy Improvement
policy-stable true
For each state s:
  old-action := π(s) via old policy
  π(s) := arg max ∑a p(s0, r ls, a) [r + gamma * V(s')] # greedy new policy: **best neighbor**
  If old-action ≠ π(s), then policy-stable = false
If policy-stable, then stop and return V
else refine **V** via Policy Evaluation (Value Improvement)

Policy improvement gibt eine echt bessere neue policy π es sei denn π war bereits optimal.

Policy Improvement

4.2 Policy-Verbesserung (Policy **Improvement**) – S. 94

Wir erhalten eine verbesserte Policy anhand einer alten Policy π (mit Wertefunktion $v\pi$)

$\pi'(s) = \text{argmax}_a\ p(s',r \mid s,a)\ [\ r + \gamma \cdot v\pi(s')\ ]$ für alle states

neue policy: gehe einfach dahin wo Position am besten ist v(s′)

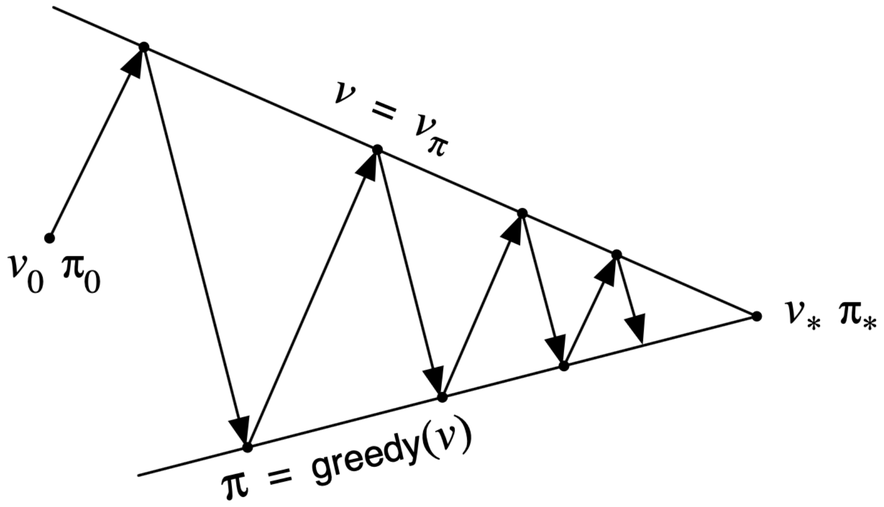
- 💡 Man kann beweisen dass diese garantiert besser ist (S. 94 Sutter)
- 💡 Man kann beweisen dass π' irgendwann nicht mehr besser wird, dann haben wir optimale policy π*

Policy Iteration

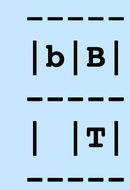
4.3 Policy **Iteration** — S. 96

Abwechselnd Policy **E**valuation und Policy **I**mprovement

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$



Bellmann Beispiel Grid World 2*2 Version 2



policy = random 50% → ↓
reward = 1 bei erreichen von Terminal state T (p=1 100%ig reward), sonst 0
p = 100% bei legalen Zügen, p = 0% bei runterfallen

γ = 0 : greedy

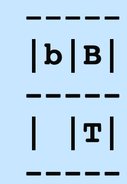
$V_p(s) = \sum \pi(a) * p(s',r|a,s) * [r + \gamma * V_p(s')]$ vereinfacht

Trivial greedy
v(s=b "begin") = ∑ ← ↑ → ↓
 = 0 /* ← */ +
  + 0 /* hoch */
 + 1/2 * 1 * [0 + 0] /* → */
 + 1/2 * 1 * [0 + 0] /* ↓ */
= 0

v(s=B "guter Begin") = ∑ ← ↑ → ↓ = 0 + 0 + 0 + 1/2 * 1 * [1 + 0] = 1/2

π2 policy = 100% ↓
v(s=b "begin") = 1*1*[0 + 0] = 0
v(s=B "guter Begin") = 1*1*[1+0] = 1

Bellmann Beispiel Grid World 2*2 Version 2



policy = random 50% → ↓
reward = 1 bei erreichen von Terminal state T (p=1 100%ig reward), sonst 0
p = 100% bei legalen Zügen, 0% bei runterfallen

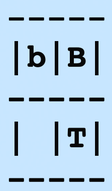
γ = 1 : kumuliert / planend

$V_p(s) = \sum_a \pi(a) * p(s',r|a,s) * [r + \gamma V_p(s')]$ vereinfacht, weil r und s' deterministisch sind

Erster Schätzung so wie greedy v(s)=0 für alle s!
v(s=b "begin") = ∑ ← ↑ → ↓
= 0 /* ← */ +
+ 0 /* hoch */
+ 1/2 * 1 * [0 + $V_p(B)$] /* → */
+ 1/2 * 1 * [0 + $V_p(T)$] /* ↓ */
= 0 v(s=B)=1/2 wie vorher

Zweite Schätzung von v(b)=0 v(B)=1/2
v(b)= /* ↓ */ 0 + /* → */ 1/2 * 1 * [0 + v(B)] = 1/2*1*1/2 = 1/4
v(B)= /* ↓ */ 1/2 * 1 * [1 + $V_p(T)$] = 1/2

Bellmann Beispiel Grid World 2*2 Version 2



policy = random 100% ↓
reward = 1 bei erreichen von Terminal state T (p=1 100%ig reward), sonst 0
p = 100% bei legalen Zügen, 0% bei runterfallen

γ = 1 : kumuliert / planend

$V_p(s) = \sum_a \pi(a) * p(s',r|a,s) * [r + \gamma V_p(s')]$ vereinfacht, weil r und s' deterministisch sind

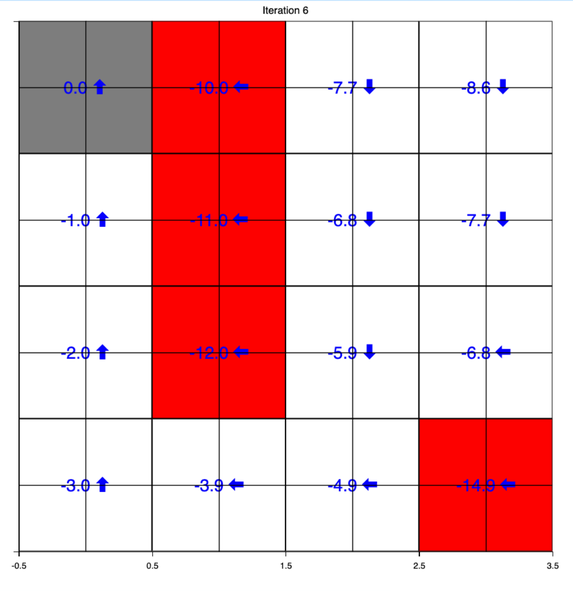
Erster Schätzung so wie greedy v(s)=0 für alle s!
v(s=b "begin") = ∑ ← ↑ → ↓
= 0 /* ← */ +
+ 0 /* hoch */
+ 0 * 1 * [0 + $V_p(B)$] /* → */
+ 1 * 1 * [0 + $V_p(T)$] /* ↓ */
v(s=b)=0 v(s=B)=1

Zweite Schätzung von v(b)=0 v(B)=1
v(b)= /* → */ 0 + /*↓*/ = 0
v(B)= /* ↓ */ 1 * 1 * [1 + $V_p(T)$] = 1

Policy-Iteration

Aufgaben in code/tag3/grid_policy.py:

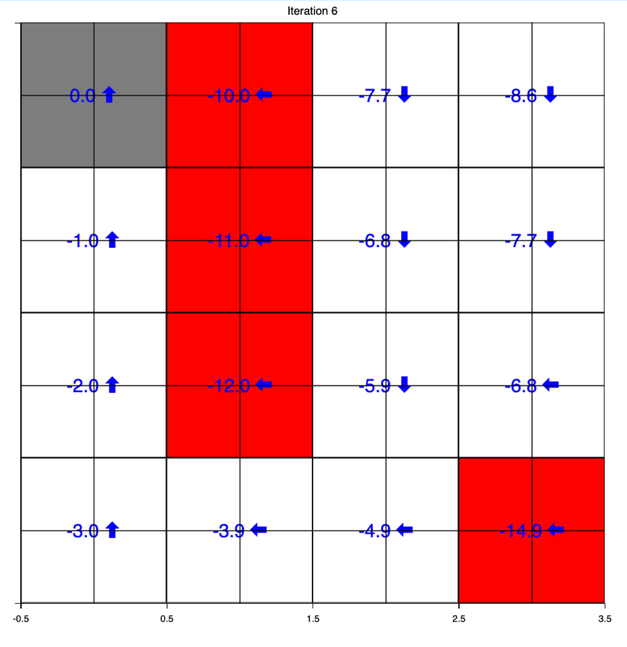
- a) **experimentieren** (verstehen durch Intuition)**:**
- b) wie sehr muss man **Mauer-Strafe** reduzieren bis es sich lohnt da rüber zu **klettern**?
- c) was passiert wenn man den Weg komplett **versperrt**? rewards[3, 1] = -10 … bei DISCOUNT_FACTOR = .9? **warum?**
- d) wie ist der **Rand** implementiert? z.B. was passiert genau wenn man rechts oben weiter nach oben gehen will?
- e) was passiert bei anderem **γ** DISCOUNT_FACTOR? **warum**? bei 0? **bei 1?**
- f) gib den **Wert** der Felder aus (Ziel-Zustand?) done
- g) baue Mauern so, dass es **zwei Pfade** gibt. Welcher wird gewählt? warum?
- h) was ändert sich wenn man value_function[TERMINAL_STATE] = 10 per hand setzt?
- i) **verstehen des codes**



Policy-Iteration

Lösungen Aufgaben in code/tag3/grid_policy.py:

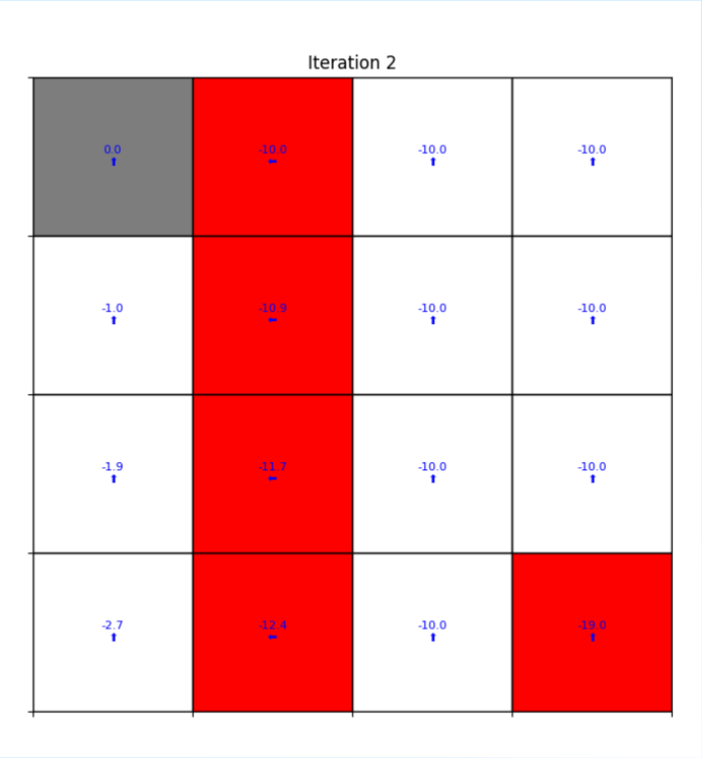
- a) **experimentieren! & verstehen**
- b) wie sehr muss man **Mauer-Strafe** reduzieren bis es sich lohnt da rüber zu **klettern**? 5/6
- c) was passiert wenn man den Weg komplett **versperrt**? rewards[3, 1] = -10 … bei DISCOUNT_FACTOR = .9? **warum?**
- d) wie ist der **Rand** implementiert? z.B. was passiert genau wenn man rechts oben weiter nach oben gehen will? is_**valid_state**
- e) was passiert bei anderem **γ** DISCOUNT_FACTOR? **warum**? bei 0? **bei 1 endlos** *warum*? policy ist mit 'oben' initialisiert. tut sich nichts.
- f) gib den **Wert** der Felder aus (Ziel-Zustand?) done
- g) baue Mauern so, dass es **zwei Pfade** gibt. Welcher wird gewählt? warum?
- h) was ändert sich wenn man value_function[TERMINAL_STATE] = 10 per hand setzt? ist motivierter zu klettern (schon bei -7). warum?



Aufgabe

Extra Aufgabe j): Bei grid size 8*8, nach wie vielen Iteration gibt es für alle Positionen einen Pfad?
(spätestens. erst denken, dann prüfen)

Extra Aufgabe k): warum geht er hier nicht ins Ziel nach links?
/Users/me/Documents/ALFA/RL/code/tag3/grid_policy.py
was hindert ihn am Klettern? reward / value / Regel?



Übungen Sutter

Übung 2.1 In dem Vergleich, der in Abbildung 2.1 gezeigt wird, welche Methode wird langfristig hinsichtlich des kumulativen Belohnungswerts und der kumulativen Wahrscheinlichkeit, die beste Aktion auszuwählen, am besten abschneiden? Wie viel besser wird sie sein? Gib deine Antwort quantitativ an.

Übung 2.2 Gib Pseudocode für einen vollständigen Algorithmus für das n-armige Banditenproblem an. Verwende gierige Aktionsauswahl und inkrementelle Berechnung der Aktionswerte mit einem Schrittweitenparameter $\alpha = 1/k$. Gehe von einer Funktion bandit(a) aus, die eine Aktion entgegennimmt und eine Belohnung zurückgibt. Verwende Arrays und Variablen; indiziere nichts über den Zeitindex t (für Beispiele dieses Pseudocode-Stils siehe Abbildungen 4.1 und 4.3). Gib an, wie die Aktionswerte initialisiert und nach jeder Belohnung aktualisiert werden. Gib an, wie die Schrittweitenparameter für jede Aktion als Funktion der Häufigkeit ihres Ausprobierens gesetzt werden.

Übung 2.3 Wenn die Schrittweitenparameter αk nicht konstant sind, dann ist die Schätzung Qk ein gewichteter Durchschnitt der zuvor erhaltenen Belohnungen mit einer Gewichtung, die sich von der in Gleichung (2.6) angegebenen unterscheidet. Wie ist die Gewichtung jeder vorherigen Belohnung im allgemeinen Fall, analog zu (2.6), in Bezug auf αk?

Übung 2.4 (Programmierung) Entwirf und führe ein Experiment durch, um die Schwierigkeiten zu demonstrieren, die Methoden mit Stichprobendurchschnitt bei nichtstationären Problemen haben. Verwende eine modifizierte Version des 10-armigen Testfelds, bei dem alle q(a) zunächst gleich sind und dann unabhängige Zufallsbewegungen ausführen. Erstelle Diagramme wie in Abbildung 2.1 für eine Aktionswert-Methode, die Stichprobendurchschnitte verwendet, inkrementell berechnet mit $\alpha = 1/k$, und eine andere Aktionswert-Methode mit einem konstanten Schrittweitenparameter $\alpha = 0.1$. Verwende $\varepsilon = 0.1$ und bei Bedarf längere Durchläufe als 1000 Spielzüge.

Übung 2.5 Die in Abbildung 2.2 gezeigten Ergebnisse sollten ziemlich zuverlässig sein, da sie Mittelwerte über 2000 einzelne, zufällig ausgewählte 10-armige Banditenaufgaben darstellen. Warum gibt es dann Oszillationen und Ausschläge im frühen Teil der Kurve für die optimistische Methode? Was könnte dazu führen, dass diese Methode besonders gut oder schlecht abschneidet – im Durchschnitt – bei bestimmten frühen Spielzügen?

Übung 2.6 Angenommen, du stehst vor einer binären Banditenaufgabe, deren wahre Aktionswerte sich von Spiel zu Spiel zufällig ändern. Genauer gesagt, nehmen wir an, dass für jedes Spiel die wahren Werte der Aktionen 1 und 2 jeweils 0.1 und 0.2 mit Wahrscheinlichkeit 0.5 (Fall A), und 0.9 und 0.8 mit Wahrscheinlichkeit 0.5 (Fall B) sind. Wenn du bei keinem Spiel weißt, welcher Fall vorliegt, was ist die beste Erfolgserwartung, die du erreichen kannst, und wie solltest du dich verhalten, um sie zu erreichen? Angenommen, bei jedem Spiel wird dir nun gesagt, ob du Fall A oder Fall B gegenüberstehst (auch wenn du die wahren Aktionswerte noch immer nicht kennst). Dies ist eine assoziative Suchaufgabe. Was ist die beste Erfolgserwartung, die du in dieser Aufgabe erreichen kannst, und wie solltest du dich verhalten, um sie zu erreichen?

Aufgaben Sutton

Übung 3.2 Ist das Verstärkungslern-Framework ausreichend, um alle zielgerichteten Lernaufgaben sinnvoll darzustellen? Fallen dir klare Ausnahmen ein?

Übung 3.3 Betrachte das Problem des Fahrens. Man könnte die Aktionen auf Ebene von Gaspedal, Lenkrad und Bremse definieren – also dort, wo der Körper die Maschine steuert. Oder weiter außen – etwa dort, wo der Reifen die Straße berührt, mit Aktionen wie Drehmomenten an den Rädern. Oder weiter innen – z. B. dort, wo das Gehirn den Körper steuert, mit Muskelzuckungen zur Bewegung der Gliedmaßen. Oder auf sehr hoher Ebene: Entscheidungen darüber, wohin man fährt. Wo liegt die „richtige“ Grenze zwischen Agent und Umgebung? Auf welcher Grundlage ist eine Grenzziehung gegenüber einer anderen vorzuziehen? Gibt es einen fundamentalen Grund für eine bestimmte Wahl, oder ist sie frei?

Übung 3.4 Angenommen, das Stab-Balancierproblem wird als episodische Aufgabe behandelt, jedoch mit Diskontierung, wobei alle Belohnungen null sind, außer -1 beim Scheitern. Wie sieht dann der Return zu jedem Zeitpunkt aus? Wie unterscheidet sich dieser Return von dem bei der diskontierten, fortlaufenden Formulierung?

Übung 3.5 Du entwirfst einen Roboter zum Durchqueren eines Labyrinths und gibst ihm eine Belohnung von +1 fürs Entkommen und sonst stets 0. Die Aufgabe lässt sich natürlich in Episoden unterteilen – aufeinanderfolgende Läufe durch das Labyrinth – also behandelst du sie als episodische Aufgabe mit dem Ziel, die erwartete Gesamtsumme der Belohnungen zu maximieren. Nach einiger Trainingszeit zeigt der Agent jedoch keine Fortschritte beim Entkommen. Was läuft falsch? Hast du dem Agenten effektiv mitgeteilt, was erreicht werden soll?

Aufgaben Sutton

Übung 3.6 Stell dir vor, du bist ein visuelles System. Beim Einschalten siehst du viele Dinge, aber nicht alles – verdeckte oder hinter dir befindliche Objekte bleiben unsichtbar. Hast du nach diesem ersten Bildzugriff Zugang zum Markov-Zustand der Umgebung? Was, wenn deine Kamera defekt ist und du den ganzen Tag kein Bild erhältst – hast du dann Zugang zum Markov-Zustand?

Übung 3.8 Was ist die Bellman-Gleichung für Aktionswerte, also für qπ? Sie soll den Wert q^\pi(s,a) über die Aktionswerte q^\pi(s{\prime},a{\prime}) möglicher Nachfolger ausdrücken. Der entsprechende Backup-Graph ist in Abbildung 3.4b. Zeige die Gleichungsfolge analog zu (3.12), jedoch für Aktionswerte.

Übung 3.9 Die Bellman-Gleichung (3.12) muss für jeden Zustand der Wertfunktion v^\pi aus Abbildung 3.5b gelten. Zeige numerisch, dass die Gleichung für den mittleren Zustand mit Wert +0.7 gilt, wenn die benachbarten Zustände die Werte +2.3, +0.4, −0.4 und +0.7 haben (auf eine Nachkommastelle gerundet).

Übung 3.10 Im Gridworld-Beispiel sind Belohnungen positiv für Ziele, negativ beim Anstoßen an die Weltgrenze und sonst null. Sind die Vorzeichen der Belohnungen wichtig oder nur deren Abstände? Zeige mit (3.2), dass das Hinzufügen einer Konstanten c zu allen Belohnungen eine Konstante v_c zu allen Zustandswerten addiert und somit die relativen Werte unverändert bleiben. Was ist v_c in Abhängigkeit von c und \gamma?

Übung 3.11 Betrachte nun das Hinzufügen einer Konstanten c in einer episodischen Aufgabe wie dem Labyrinthlauf. Hat dies einen Effekt, oder bleibt die Aufgabe – wie im kontinuierlichen Fall – unverändert? Warum oder warum nicht? Gib ein Beispiel.

Übung 3.12 Der Wert eines Zustands hängt von den Werten der möglichen Aktionen und deren Wahrscheinlichkeiten unter der aktuellen Politik ab. Gib die entsprechende Gleichung zur Intuition und zum Diagramm an für v^π(s) in Abhängigkeit von q^π(s,a), gegeben S_t = s. Gib danach eine zweite Gleichung an, die explizit π(a|s) verwendet, ohne Erwartungsnotation.

Übung 3.13 Der Wert einer Aktion q^π(s,a) hängt vom erwarteten nächsten Reward und dem erwarteten Wert des nächsten Zustands ab. Gib die entsprechende Gleichung gemäß dem Diagramm an für q^\pi(s,a) in Abhängigkeit von R_{t+1} und v^\pi(S_{t+1}), gegeben S_t = s und A_t = a. Gib danach eine zweite Gleichung an, in der der Erwartungswert explizit durch p(s',r|s,a) (vgl. (3.6)) ausgedrückt wird, ohne Erwartungsnotation.

Aufgaben Sutton

Übung 3.14 Zeichne oder beschreibe die optimale Zustandswertfunktion für das Golf-Beispiel.

Übung 3.15 Zeichne oder beschreibe die Konturen der optimalen Aktionswertfunktion für das Putten q^*(s,putter) im Golf-Beispiel.

Übung 3.16 Gib die Bellman-Gleichung für q^* des Recyclingroboters an.

Übung 3.17 Abbildung 3.8 gibt den optimalen Wert des besten Zustands der Gridworld mit 24.4 an (eine Nachkommastelle). Nutze dein Wissen über die optimale Politik und (3.2), um diesen Wert symbolisch auszudrücken und auf drei Dezimalstellen genau zu berechnen.

Übung 3.18 Definiere v^* in Abhängigkeit von q^*.

Übung 3.19 Definiere q^* in Abhängigkeit von v^*.

Übung 3.20 Definiere π^* in Abhängigkeit von q^*.

Übung 3.21 Definiere π^* in Abhängigkeit von v^*.

On Policy / Off Policy

On-Policy Learning:

Die Lernmethode nutzt **Daten**, die mit der aktuellen Policy (Strategie) **erzeugt** wurden.
Das bedeutet: dieselbe Policy wird zum **Handeln** und zum Lernen verwendet.
Beispiel: **SARSA** (State-Action-Reward-State-Action)
Achtung: premature Optimization? Im Extremfall noch während eines Spieles

Off-Policy Learning:

Lernmethode nutzt Daten, die von einer **anderen Policy** erzeugt wurden
– typischerweise einer explorativen oder **älteren** Policy.
Lernen erfolgt anhand einer Ziel-Policy (target policy),
während eine Verhaltens-Policy (behavior policy) die **Daten generiert**.
Beispiel: Q-Learning

Fließender Übergang, oft wird die Policy verzögert Updated: pro **batch**, nach mehreren Episoden …
Auch klassisches off-Policy wird zum latenten on-policy, wenn man den Trainingslauf mehrfach wiederholt.
Fließender Übergang von off policy zu supervised learning

On Policy / Off Policy

Das **Iterative Lösung der Bellman-Gleichung** in der Dynamischen Programmierung nennt ist **On Policy:**

Während der Iteration verändert sich die Policy mit.

Vokabular

Kann nach heute vergessen werden:

`Bellmann Gleichungen`

DP: dynamic programming (in context von MCP):

Policybewertung bezeichnet die (typischerweise) iterative Berechnung der Wertfunktionen für eine gegebene Policy.

Policyverbesserung bezieht sich auf die Berechnung einer verbesserten Policy, gegeben die Wertfunktion dieser Policy.

Wenn man diese beiden Berechnungen kombiniert, erhält man die **Policyiteration** und die **Wertiteration**, die zwei beliebtesten DP-Methoden.

`Policy Iteration mit Bellmann Gleichungen:`
`  abwechselnd Wert-Schätzung verbessern und dann die Policy`

Beide können verwendet werden, um zuverlässig optimale Policen und Wertfunktionen für endliche MDPs zu berechnen, sofern vollständiges Wissen über das MDP vorliegt.

Formalisierung der Konzepte

Notation:

- **s** state aktueller Zustand des Systems / Spiel-Feld / …
- **a** action, von einem Zustand zum nächsten "Spielzug"
- **π**(a|s) policy Übergangs'Wahrscheinlichkeit' eines Zustandes zum nächsten (zB 100% 'rot')
- R/F: 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
- $Q_t(a)$: Schätzwert von q*(a) zum Zeitpunkt t (“action-values”)
- $N_t(a)$: Anzahl der bisherigen Auswahlvorgänge der Aktion a bis Zeitpunkt t
- $\pi_t(a)$: Wahrscheinlichkeit zur Auswahl von Aktion a zum Zeitpunkt t ≈ $P_t(a|s)$

Zentrale Beziehungen:

- $v(s) = \sum_a \pi(a|s)\, q(s, a)$
  „Wert einer (Schach-)Position unter der Policy“ = “Summe:Werte aller 'qualifizierten' Aktionen”
- $q\pi(s, a) = \mathbb{E}\pi[G_t \mid S_t = s, A_t = a]$ $\quad q_p=\mathbb{E}_p[G_t]$
  „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)

Aufgaben

a) schreibt eine eigene Policy in grid_gym_cliff.py

a1) hard coded zum Ziel (einfach)

a2) möglichst unter Ausnutzung von Value ;)

- b) Einbau von **policy iteration** in gymnasium environment "CliffWalking-v1"
- b1) copy paste von grid_policy.py

nutze P = env.unwrapped.P *# transition dynamics:*
Übergangswahrscheinlichkeiten können wir global abfragen!

=> mit Bellmann viel schneller lösen als durch 'probieren'

Zusatz Aufgaben

c) Ein Roboter erhält:

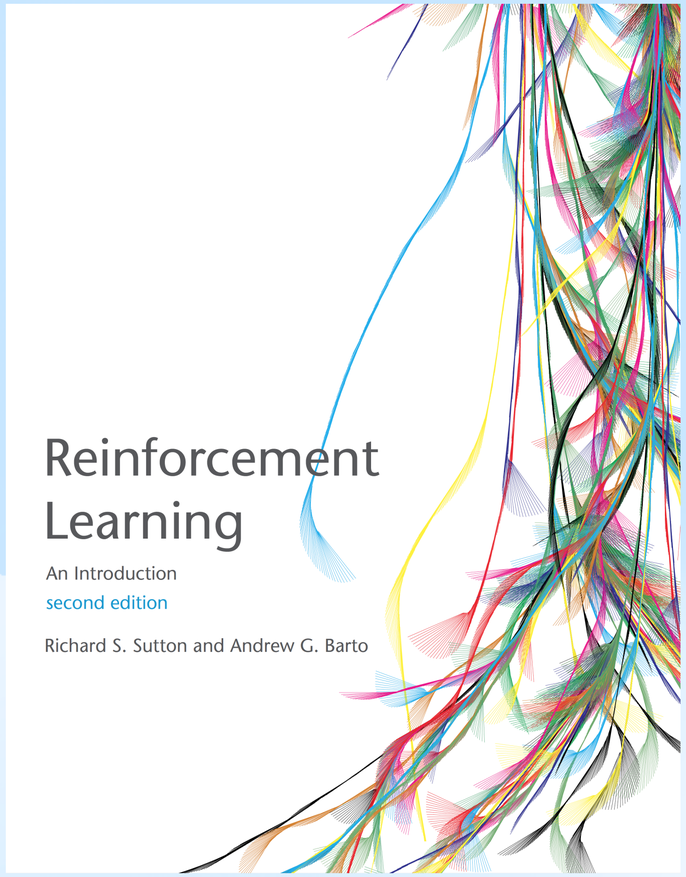
* +100 für das Ziel
* +1 pro Schritt

Der Roboter ist zehn Felder vom Ziel entfernt. Das Spiel bricht nach 20 Zügen ab. Welche Policy wird er am Ende 'optimal' erlernen?

d) Was ist oben der erwartete Gain bei einem Gamma-Faktor von 0.9

e) ändert sich dadurch die Strategie?

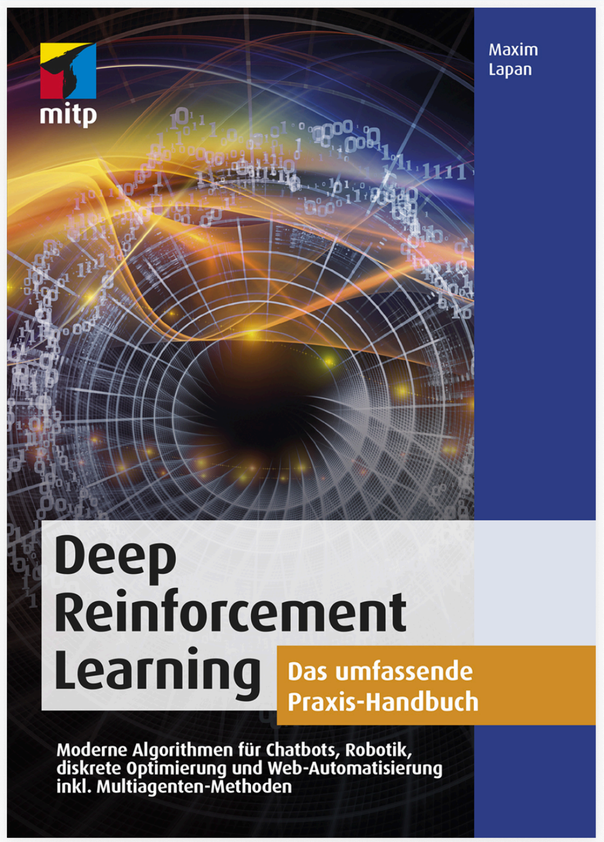
Buch



Sutton: bis Seite 88 (gegebenenfalls nur überfliegen!)

https://www.coursera.org/specializations/reinforcement-learning
https://github.com/ivanbelenky/RL Sutton and Barto book implementation

Buch



Lapan Deep-Reinforcement-Learning: **bis Seite 66** (gegebenenfalls nur überfliegen)