Dynamic Programming

Ziele dieser Vorlesung:

- Überblick über eine Sammlung klassischer Lösungsmethoden $dynamic programming$ genannt werden
- Demonstration des Einsatzes von DP beim Errechnen von W und somit von optimalen $Policies$
- Diskussion der Effektivität und des Nutzens von DP

Dieser Teil ist aus "Reinforcement Learning: An Introduction", Richard S. Sutt

Policy Evaluierung

***Policy* Evaluierung**: Errechne die Zustands-Wertefunktion V^π gegebene $Policy\,\pi$.

Erinnern wir uns:
Zustands-Wertefunktion für *Policy* π :

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

Bellman-Gleichung für V^π:

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

– ein System von $|S|$ gleichzeitigen linearen Gleichungen

Iterative Methoden

$$V_0 \to V_1 \to \ldots \to V_k \to V_{k+1} \to \ldots \to V^\pi$$
$$\Uparrow \text{ ein "Durchlauf"}$$

Ein Durchlauf besteht darin, auf jeden Zustand eine Backup-Op anzuwenden.

Ein $full - policy$ Evaluierungs-Backup:

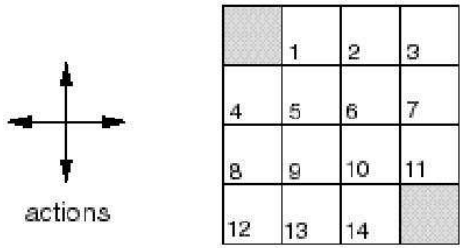
$$V_{k+1}(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]$$

Iterative *Policy* Evaluierung

`Input` π, die zu evaluierende $Policy$
`Initialisiere` $V(s) = 0$, für alle $s \in S^+$
`Wiederhole`
  $\Delta \leftarrow 0$
  Für jedes $s \in S$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]$
    $\Delta <\leftarrow \max(\Delta, |v - V(s)|)$

bis $\Delta < \theta$ (eine kleine positive Zahl)
Output $V \approx V^\pi$

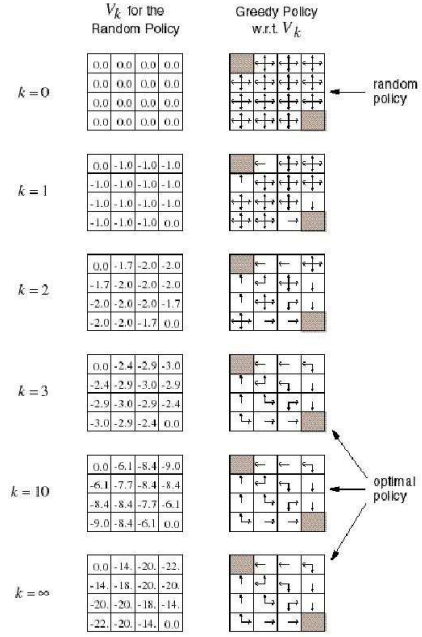
Eine kleine *Gridworld*



$r = -1$
on all transitions

- Eine episodische Aufgabe ($undiscounted$)
- Nicht-terminale Zustände :$1, 2, \ldots, 14$;
- Ein terminaler Zustand (zweimal als schattiertes Quadrat dar
- Aktionen, die den Agenten aus dem $Grid$ nehmen würden, la Zustand unverändert
- Der $Reward$ ist - 1 bis der terminale Zustand erreicht ist

Iterative *Policy* Evaluierung für die kleine *Gr*



π = zufällige (gleichförmige) Aktions-Entscheidungen

Policy Verbesserung

Angenommen, wir haben V^π für eine deterministische $Policy\,\pi$

Wäre es besser für einen gegebenen Zustand s eine Aktion $a \neq$ durchzuführen?

Der Wert, wenn a im Zustand s durchgeführt wird, ist:

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\{r_{r+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a \\ &= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

Es ist genau dann besser, zur Aktion a für den Zustand s zu we

$$Q^\pi(s,a) > V^\pi(s)$$

Policy Verbesserung Forts.

Führe dies für alle Zustände durch, um eine neue $Policy\,\pi$ zu b in Bezug auf V^π ***greedy*** ist:

$$\begin{aligned} \pi'(s) &= \operatorname*{argmax}_a Q^\pi(s,a) \\ &= \operatorname*{argmax}_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \end{aligned}$$

Dann $V^{\pi'} \geq V^\pi$

Policy Verbesserung Forts.

Was wenn $V^{\pi'} = V^{\pi}$?
Z.B. für alle $s \in S$, $\quad V^{\pi'}(s) = \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')]$?

Aber dies ist die optimale Bellman-Gleichung.

Also sind $V^{\pi'} = V^*$ und sowohl π als auch π' optimale $Policies$

Policy Iteration

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \ldots \pi^* \rightarrow V^* \rightarrow \pi^*$$

$Policy$ Evaluierung $\Uparrow$ $\Uparrow$ $Policy$ Verbesserung "Greedificati

Policy Iteration

1. Initialisierung
   $V(s) \in \Re$ und $\pi(s) \in A(s)$ beliebig für alle $s \in S$

2. *Policy* Evaluierung
   Wiederhole
   $\Delta \leftarrow 0$
   Für jedes $s \in S$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s'} P_{ss'}^{\pi(s)} [R_{ss'}^{\pi(s)} + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   bis $\Delta < \theta$ (eine kleine positive Zahl)

3. *Policy* Verbesserung
   Policy-stable $\leftarrow$ wahr
   Für jedes $s \in S$:
   $b \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma V(s')]$
   Wenn $b \neq \pi(s)$, dann *Policy-stable* $\leftarrow$ falsch
   Wenn *Policy-stable*, dann aufhören; ansonsten gehe zu 2

Maschinelles Lernen W24,

Wert Iteration

Erinnern wir uns an das ***full policy*** **Evaluierungs-*Backup***

$$V_{k+1}(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]$$

Hier ist das ***full value*** **Iterations-*Backup***:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]$$

Wert Iteration Forts.

Initialisiere V beliebig, z.B. $V(s) = 0$, für alle $s \in S^+$
Wiederhole

$\Delta \leftarrow 0$
`Für jedes` $s \in S$:
  $v \leftarrow V(s)$
  $V(s) \leftarrow \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$
  $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
bis $\Delta < \theta$ (eine kleine positive Zahl)

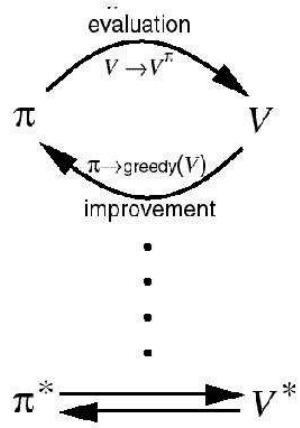
Output ist eine deterministische $Policy$, π, so dass
$\pi(s) = \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

Asynchrones DP

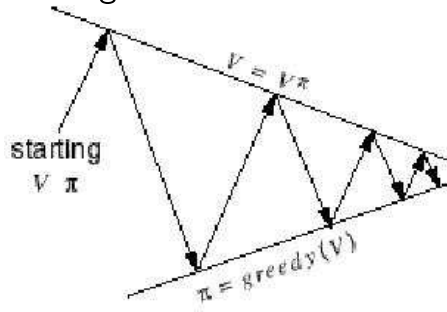
- Alle DP Methoden, die bisher beschrieben wurden, erfordern Durchläufe des gesamten Zustands-Sets
- Das asynchrone DP benutzt keine Durchläufe. Stattdessen fu so:
  - Wiederhole bis das Konvergenzkriterium eingehalten wird:
    - Suche einen Zustand zufällig heraus und wende das pass an
- Benötigt immer noch viel Berechnung, aber hängt nicht in h langen Durchläufen fest
- Kann man Zustände für die Anwendung des $Backup$ intellige JA: die Erfahrung eines Agenten kann als Führer dienen

Generalized $Policy$ Iteration (GPI)

***Generalized Policy* Iteration** (GPI):
jede Interaktion zwischen $Policy$ Evaluierung und $Policy$ Verbesserung, u ihrer Granularität.



Eine geometrische Metapher f Konvergenz von GPI:



Effizienz des DP

- Eine optimale $Policy$ zu finden ist polynomisch mit der Zahl Zustände...
- ABER, die Zahl der Zustände ist oft astronomisch, z.B. wächst exponentiell mit der Zahl der Zustands-Variablen (was Bellma der Dimensionalität" nannte)
- In der Praxis kann das klassische DP auf Probleme mit ein pa Zuständen angewandt werden
- Das asynchrone DP kann auf größere Probleme angewandt w eignet sich für parallele Berechnung
- Es ist überraschend leicht, MDPs zu finden, für die DP Meth unpraktisch sind

Zusammenfassung

- $Policy$ Evaluierung: $Backups$ ohne Maximum
- $Policy$ Verbesserung: bilde eine $greedy\ Policy$, wenn auch nur lokal
- $Policy$ Iteration: wechsle die beiden obigen Prozesse ab
- Wert Iteration: $Backups$ mit Maximum
- Vollständige $Backups$ (die später Beispiel-$Backups$ gegenübergestellt
- Generalized $Policy$ Iteration (GPI)
- Asynchrones DP: eine Methode, um vollständige Durchläufe zu vermeiden
- $Bootstrapping$: Schätzungen durch andere Schätzungen auf den neues bringen