Grundlagen der Künstlichen Intelligenz

19. Handeln unter Unsicherheit: Grundlagen

Malte Helmert

Universität Basel

27. Mai 2013

Einordnung

Einordnung:

Handeln unter Unsicherheit

Umgebung:

- statisch vs. dynamisch
- deterministisch vs. nicht-deterministisch vs. stochastisch
- vollständig vs. partiell vs. nicht beobachtbar
- diskret vs. stetig
- ein Agent vs. mehrere Agenten (Gegenspieler)

Lösungsansatz:

- problemspezifisch vs. allgemein vs. lernend

Erinnerung: Klassische Suchprobleme

Erinnerung: Annahmen bei klassischen Suchproblemen

- einzelner Agent in Umgebung (ein Agent)
- kennt immer genauen Weltzustand (vollständig beobachtbar)
- Zustand ändert sich nur durch den Agenten (statisch)
- endlich viele mögliche Zustände/Aktionen (insbes. diskret)
- Aktionen haben deterministischen Einfluss auf Zustand

Erinnerung: Klassische Suchprobleme

Erinnerung: Annahmen bei klassischen Suchproblemen

- einzelner Agent in Umgebung (ein Agent)
- kennt immer genauen Weltzustand (vollständig beobachtbar)
- Zustand ändert sich nur durch den Agenten (statisch)
- endlich viele mögliche Zustände/Aktionen (insbes. diskret)
- Aktionen haben deterministischen Einfluss auf Zustand

In vielen praktische Situationen ist Determinismus nicht gegeben!

Erinnerung: Klassische Suchprobleme

Erinnerung: Annahmen bei klassischen Suchproblemen

- einzelner Agent in Umgebung (ein Agent)
- kennt immer genauen Weltzustand (vollständig beobachtbar)
- Zustand ändert sich nur durch den Agenten (statisch)
- endlich viele mögliche Zustände/Aktionen (insbes. diskret)
- Aktionen haben deterministischen Einfluss auf Zustand

In vielen praktische Situationen ist Determinismus nicht gegeben!

$\rightsquigarrow$ jetzt: Handeln in stochastischen Umgebungen

Markov-Entscheidungsprobleme

Erinnerung: Zustandsraum

Zur Erinnerung unsere Definition im klassischen Fall:

Definition (Zustandsraum)

Ein Zustandsraum ist ein 6-Tupel $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ mit

- S endliche Menge von Zuständen
- A endliche Menge von Aktionen
- $cost : A \to \mathbb{R}_0^+$ Aktionskosten
- $T \subseteq S \times A \times S$ Transitionsrelation oder Übergangsrelation; deterministisch in $\langle s, a \rangle$
- $s_0 \in S$ Anfangszustand
- $S_\star \subseteq S$ Menge der Zielzustände

Erinnerung: Zustandsraum

Zur Erinnerung unsere Definition im klassischen Fall:

Definition (Zustandsraum)

Ein Zustandsraum ist ein 6-Tupel $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ mit

- S endliche Menge von Zuständen
- A endliche Menge von Aktionen
- $cost : A \to \mathbb{R}_0^+$ Aktionskosten
- $T \subseteq S \times A \times S$ Transitionsrelation oder Übergangsrelation; deterministisch in $\langle s, a \rangle$
- $s_0 \in S$ Anfangszustand
- $S_\star \subseteq S$ Menge der Zielzustände

Was ändert sich?

- Deterministische Transitionen $\rightsquigarrow$ Wahrscheinlichkeiten
- Aktionskosten und Zielzustände $\rightsquigarrow$ zustandsabhängige Belohnungen (rewards)

Markov-Entscheidungsproblem

Definition (Markov-Entscheidungsproblem)

Ein Markov-Entscheidungsproblem (Markov decision process, MDP) ist ein 5-Tupel $\mathcal{M} = \langle S, A, T, R, s_0 \rangle$ mit

- S endliche Menge von Zuständen
- A endliche Menge von Aktionen
- $T : S \times A \times S \to [0, 1]$ Transitionswahrscheinlichkeiten; erfüllen $\sum_{s' \in S} T(s, a, s') = 1$ für alle $s \in S$, $a \in A$
- $R : S \to \mathbb{R}$ Belohnungsfunktion (rewards)
- $s_0 \in S$ Anfangszustand

Name kommt von so genannter Markov-Eigenschaft: nächster Zustand hängt nur von aktuellem Zustand, gewählter Aktion und Zufall ab, nicht von der „Vorgeschichte“

Unterschiede zum klassischen Fall

1. Transitionswahrscheinlichkeiten:
   - wird in Zustand s die Aktion a ausgeführt, hängt Nachfolgezustand vom Zufall ab: Nachfolgezustand ist s' mit Wahrscheinlichkeit $T(s, a, s')$
2. Anwendbare Aktionen:
   - Bei MDPs sind üblicherweise (und in unserer Definition) immer alle Aktionen anwendbar.
   - „Eigentlich“ nicht anwendbare Aktionen oft modelliert als Aktionen, die immer von s zu s zurückführen oder in einen speziellen „Fehlerzustand“ führen.
3. Belohnungen:
   - Statt Zielzuständen ist für jeden Zustand eine Belohnung für das Erreichen definiert.
   - $\rightsquigarrow$ Zielzustände über Belohnungen modellierbar
   - Belohnungen können negativ sein, was Kosten entspricht. $\rightsquigarrow$ Aktionskosten über Belohnungen modellierbar

MDPs: Ziel des Agenten

- Ziel des handelnden Agenten bei MDPs ist,
  so viele Belohnungen wie möglich aufzusammeln.
- Zwei Problemvarianten:
  - endlicher Horizont H:
    Agent führt H Aktionen aus, dann „endet“ das Problem
  - unendlicher Horizont:
    Agent interagiert unbegrenzt lang mit der Umgebung

$\rightsquigarrow$ wir behandeln beides; unendlicher Horizont ist verbreiteter

Policys

- Aktionsfolgen sind hier kein gutes Lösungskonzept:
  beste Aktion im 2. Schritt kann vom zufälligen Ausgang nach Ausführung der Aktion im 1. Schritt abhängen
- beste Aktion hängt von aktuellem Zustand ab
  (den der Agent immer beobachten kann) sowie
  (bei endlichem Horizont) von verbleibender Zeit

$\rightsquigarrow$ berechne Policy (= Strategie) für jeden möglichen Zustand

- stationäre Policy: $\pi : S \to A$
  $\pi(s)$ Aktion in Zustand s
- nichtstationäre Policy: $\pi : S \times \{1, \dots, H\} \to A$
  $\pi(s, t)$ Aktion in Zustand s, wenn noch t Zeitschritte übrig

Explizite Zustandsräume

- In diesem Kapitel betrachten wir Algorithmen, die gesamte Policys (vor-) berechnen, also das Problem komplett lösen.
- Aufwand dafür ist hoch und nur für Probleme geeignet, bei denen alle Zustände in den Speicher passen (explizite Darstellung der Zustandsräume)
- Verallgemeinerungen auf deklarative Zustandsräume kombinieren MDP-Techniken mit Handlungsplanungstechniken („probabilistische Planung“)

Wert einer Policy

Wie gut ist eine Policy π?

- Wie messen wir die von π aufgesammelten Belohnungen?
- Wertfunktion $V_\pi : S \mapsto \mathbb{R}$ (für stationäre Policys) bzw. $V_\pi^t : S \mapsto \mathbb{R}$ ($t \in \mathbb{N}_0$; für nichtstationäre Policys) misst erwartete Belohnung bei Ausführung der Policy von gegebenem Zustand s aus
- hängt von unmittelbarer Belohnung in s ab, aber auch davon, welche Belohnungen später gesammelt werden können
- unsere Aufgabe: berechne optimale Policy (maximiert $V_\pi(s)$ bzw. $V_\pi^t(s)$ in jedem Zustand)

Anmerkung: Anfangszustand spielt in diesem Kapitel keine Rolle, ist aber bei fortgeschrittenen Techniken wichtig.

Endlicher Horizont

Optimales Verhalten bei endlichem Horizont

- Sie befinden sich in Sydney und Ihr Rückflug nach Hause startet morgen früh. Was ist die beste Aktion?

Optimales Verhalten bei endlichem Horizont

- Sie befinden sich in Sydney und Ihr Rückflug nach Hause startet morgen früh. Was ist die beste Aktion?
- Sie befinden sich in Sydney und Ihr Rückflug nach Hause startet in drei Monaten. Was ist die beste Aktion?

Optimales Verhalten bei endlichem Horizont

- Sie befinden sich in Sydney und Ihr Rückflug nach Hause startet morgen früh. Was ist die beste Aktion?
- Sie befinden sich in Sydney und Ihr Rückflug nach Hause startet in drei Monaten. Was ist die beste Aktion?
- Optimales Verhalten bei endlichem Horizont hängt von verbleibender Restzeit ab.

$\rightsquigarrow$ eine nichtstationäre Policy wird benötigt

Wertfunktion für endlichen Horizont

- $V^k_\pi(s)$ ist Wert von Policy π und Restzeit k im Zustand s
- erwarteter Gesamtnutzen bei Ausführung von π in s, wenn noch k Schritte ausgeführt werden können

$$
\begin{aligned}
V^k_\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{k} R_t \middle| \pi, s\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{k} R(s_t) \middle| a_t = \pi(s_t, k-t), s_0 = s\right]
\end{aligned}
$$

- R_t und s_t sind Zufallsvariablen

Algorithmische Probleme

Policy-Bewertung

Gegeben ein MDP, eine nichtstationäre Policy π und ein endlicher Horizont H, berechne die Wertfunktionen V_π.

Policy-Optimierung

Gegeben ein MDP und ein endlicher Horizont H,
berechne eine optimale Policy π^* für den Horizont H.

Algorithmische Probleme

Policy-Bewertung

Gegeben ein MDP, eine nichtstationäre Policy π und ein endlicher Horizont H, berechne die Wertfunktionen V_π.

Policy-Optimierung

Gegeben ein MDP und ein endlicher Horizont H, berechne eine optimale Policy π^* für den Horizont H.

- Wie viele Policys mit endlichem Horizont gibt es (abhängig von Zustandszahl $|S|$ und Aktionszahl $|A|$)?

Algorithmische Probleme

Policy-Bewertung

Gegeben ein MDP, eine nichtstationäre Policy π und ein endlicher Horizont H, berechne die Wertfunktionen V_π.

Policy-Optimierung

Gegeben ein MDP und ein endlicher Horizont H, berechne eine optimale Policy π^* für den Horizont H.

- Wie viele Policys mit endlichem Horizont gibt es (abhängig von Zustandszahl $|S|$ und Aktionszahl $|A|$)?
- Antwort: $|A|^{|S| \cdot H}$

Algorithmische Probleme

Policy-Bewertung

Gegeben ein MDP, eine nichtstationäre Policy π und ein endlicher Horizont H, berechne die Wertfunktionen V_π.

Policy-Optimierung

Gegeben ein MDP und ein endlicher Horizont H, berechne eine optimale Policy π^* für den Horizont H.

- Wie viele Policys mit endlichem Horizont gibt es (abhängig von Zustandszahl $|S|$ und Aktionszahl $|A|$)?
- Antwort: $|A|^{|S|\cdot H}$

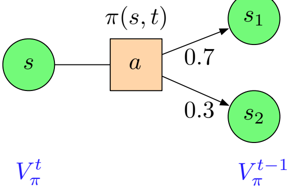
$\rightsquigarrow$ Ausprobieren aller Alternativen nicht möglich!

- wir zeigen: Policy-Optimierung kann zurückgeführt werden auf Berechnung der optimalen Wertfunktion

Policy-Bewertung bei endlichem Horizont

Benutze dynamische Programmierung:
Wert mit t verbleibenden Zeitschritten einfach zu berechnen, wenn Wert mit $t-1$ verbleibenden Zeitschritten bekannt.

$$\forall s \in S \, \forall t \geq 1:$$
$$V_\pi^0(s) = R(s)$$
$$V_\pi^t(s) = R(s) + \sum_{s' \in S} T(s, \pi(s,t), s') \cdot V_\pi^{t-1}(s')$$

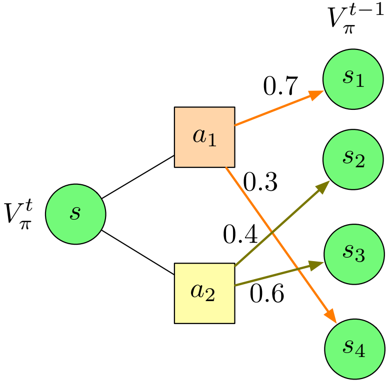


Policy-Optimierung: Bellman-Backups

Wie berechnen wir bestmöglichen Wert $V_*^t(s)$ gegeben $V_*^{t-1}(s)$?

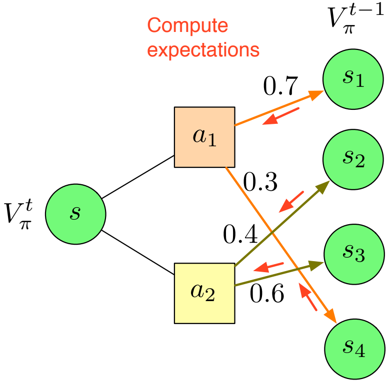
Policy-Optimierung: Bellman-Backups

Wie berechnen wir bestmöglichen Wert $V_*^t(s)$ gegeben $V_*^{t-1}(s)$?



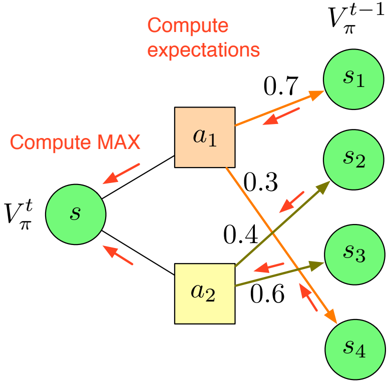
Policy-Optimierung: Bellman-Backups

Wie berechnen wir bestmöglichen Wert $V_*^t(s)$ gegeben $V_*^{t-1}(s)$?



Policy-Optimierung: Bellman-Backups

Wie berechnen wir bestmöglichen Wert $V_*^t(s)$ gegeben $V_*^{t-1}(s)$?



Value Iteration für endlichen Horizont

Dynamische Programmierung kann für Konstruktion der optimalen Policy verwendet werden:

Value Iteration

$$\forall s \in S \, \forall t \geq 1 :$$

$$V_*^0(s) = R(s)$$

$$V_*^t(s) = R(s) + \max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V_*^{t-1}(s')$$

Value Iteration für endlichen Horizont

Dynamische Programmierung kann für Konstruktion der optimalen Policy verwendet werden:

Value Iteration

$$\forall s \in S\,\forall t \geq 1:$$

$$V_*^0(s) = R(s)$$

$$V_*^t(s) = R(s) + \max_{a \in A} \sum_{s' \in S} T(s,a,s') \cdot V_*^{t-1}(s')$$

$$\pi^*(s,t) = \arg\max_{a \in A} \sum_{s' \in S} T(s,a,s') \cdot V_*^{t-1}(s')$$

Value Iteration für endlichen Horizont

Dynamische Programmierung kann für Konstruktion der optimalen Policy verwendet werden:

Value Iteration

$$
\begin{aligned}
&\forall s \in S \, \forall t \geq 1 : \\
&\qquad V_*^0(s) = R(s) \\
&\qquad V_*^t(s) = R(s) + \max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V_*^{t-1}(s') \\
&\qquad \pi^*(s,t) = \arg\max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V_*^{t-1}(s')
\end{aligned}
$$

$V_*^t(s)$ ist die optimale t-Schritt-Wertfunktion

π^* ist optimale Policy (deren Wertfunktion $V_{\pi*}$ ist)

Value Iteration: Zeitaufwand

Zeitaufwand von Value Iteration:

- H Iterationen

Value Iteration: Zeitaufwand

Zeitaufwand von Value Iteration:

- H Iterationen
- pro Iteration: $|S|$ Zustände (für s)

Value Iteration: Zeitaufwand

Zeitaufwand von Value Iteration:

- H Iterationen
- pro Iteration: $|S|$ Zustände (für s)
- pro Zustand: $|A|$ Aktionen (für a)

Value Iteration: Zeitaufwand

Zeitaufwand von Value Iteration:

- H Iterationen
- pro Iteration: $|S|$ Zustände (für s)
- pro Zustand: $|A|$ Aktionen (für a)
- pro Zustand/Aktions-Paar: $|S|$ Nachfolgezustände (für s')

Value Iteration: Zeitaufwand

Zeitaufwand von Value Iteration:

- H Iterationen
- pro Iteration: $|S|$ Zustände (für s)
- pro Zustand: $|A|$ Aktionen (für a)
- pro Zustand/Aktions-Paar: $|S|$ Nachfolgezustände (für s')

$\leadsto$ Laufzeit $O(H \cdot |S|^2 \cdot |A|)$

Value Iteration: Zeitaufwand

Zeitaufwand von Value Iteration:

- H Iterationen
- pro Iteration: $|S|$ Zustände (für s)
- pro Zustand: $|A|$ Aktionen (für a)
- pro Zustand/Aktions-Paar: $|S|$ Nachfolgezustände (für s')

$\rightsquigarrow$ Laufzeit $O(H \cdot |S|^2 \cdot |A|)$

$\rightsquigarrow$ polynomiell in $|S|$, $|A|$ und H

Frage: Ist das gut?

Zusammenfassung: endlicher Horizont

- Value Iteration berechnet eine optimale Policy:

$$V_{\pi^*}^t(s) \geq V_\pi^t(s), \quad \forall \pi, s, t$$

- Anmerkung: optimale Wertfunktion V_* ist eindeutig; die Policy π^* selbst muss nicht eindeutig sein (Warum?)
- Rechenaufwand ist polynomiell in der Grösse des Problems (gemessen in Zuständen und Aktionen) und des Horizonts

Unendlicher Horizont

Probleme für Wertfunktionen bei unendlichem Horizont

- Für viele MDPs wäre jede feste Zeitschranke willkürlich.
- Beispiel: „Wirf eine Münze, bis Kopf fällt“.
  Welcher endliche Horizont wäre angemessen?
- Daher verwendet man meist einen unendlichen Horizont.
- Problem: bei vielen MDPs ist „Summe über zukünftige Belohnungen“ bei unendlichem Horizont nicht wohldefiniert. (Belohnungen wären unendlich oder divergent.)

Trick: betrachte diskontierte MDPs, bei denen sofortige Belohnungen wertvoller sind als zukünftige.

- Diskontfaktor $0 < \gamma < 1 \in \mathbb{R}$
- zukünftige Belohnungen in jedem Zeitschritt um Faktor γ reduziert

Wertfunktion für unendlichen Horizont

Diskontierte Wertfunktion mit Diskontfaktor γ:

- $V_\pi(s)$ ist Wert von Policy π im Zustand s
- erwarteter Gesamtnutzen bei Ausführung von π in s:

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \,\middle|\, \pi, s\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \,\middle|\, a_t = \pi(s_t), s_0 = s\right]
\end{aligned}
$$

- R_t und s_t sind Zufallsvariablen

Wertfunktion für unendlichen Horizont

Diskontierte Wertfunktion mit Diskontfaktor γ:

- $V_\pi(s)$ ist Wert von Policy π im Zustand s
- erwarteter Gesamtnutzen bei Ausführung von π in s:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \,\middle|\, \pi, s\right] \\ &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \,\middle|\, a_t = \pi(s_t), s_0 = s\right] \end{aligned}$$

- R_t und s_t sind Zufallsvariablen

Warum Diskontierung?

- Wertfunktion ist wohldefiniert: konvergiert immer
- Belohnungen werden schnell angestrebt: praktisch oft sinnvoll
- ökonomische Argumente

Eigenschaften von diskontierten MDPs

Optimale Policy maximiert Wert jedes Zustands.

- In diskontierten MDPs gibt es immer eine optimale stationäre Policy (Howard, 1960)
- Wir schreiben V_* für die Wertfunktion einer optimalen Policy π^*.

Algorithmische Probleme

Policy-Bewertung

Gegeben MDP mit unendlichem Horizont, Diskontfaktor γ und stationäre Policy π, berechne die Wertfunktion V_π.

Policy-Optimierung

Gegeben MDP mit unendlichem Horizont und Diskontfaktor γ, berechne eine optimale Policy π.

Algorithmische Probleme

Policy-Bewertung

Gegeben MDP mit unendlichem Horizont, Diskontfaktor γ und stationäre Policy π, berechne die Wertfunktion V_π.

Policy-Optimierung

Gegeben MDP mit unendlichem Horizont und Diskontfaktor γ, berechne eine optimale Policy π.

- dieselben Probleme wie bei endlichem Horizont
- Value Iteration über „alle" Zeitschritte reicht nicht mehr aus, da Ausführung der Policy kein Ende nimmt
- Wie können wir dennoch V_π berechnen?
- Wie können wir einen optimale Policy finden?

Policy-Bewertung bei unendlichem Horizont

Erinnerung: Zusammenhang V_π^t und V_π^{t-1} bei endlichem Horizont:

$$V_\pi^t(s) = R(s) + \sum_{s' \in S} T(s, \pi(s,t), s') \cdot V_\pi^{t-1}(s')$$

Bei stationären Policys ist der Zeitschritt für π egal und mit der Diskontierung ergibt sich eine Rekursionsgleichung:

$$V_\pi(s) = R(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \cdot V_\pi(s')$$

Wie können wir V_π berechnen?

Berechnung der Wertfunktion

Wie können wir V_π berechnen?

$$V_\pi(s) = R(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \cdot V_\pi(s') \quad \text{für alle } s \in S$$

- Bei Policy-Bewertung sind R, γ, T und π bekannt.
- unbekannt: die Werte $V_\pi(s)$ für alle $s \in S$

$\rightsquigarrow$ lineares Gleichungssystem (LGS) mit $|S|$ Gleichungen und $|S|$ Variablen

- dieses LGS hat immer genau eine Lösung, wenn $0 < \gamma < 1$ gilt (dabei geht ein, dass T Wahrscheinlichkeiten kodiert)

$\rightsquigarrow$ Lösung des LGS mit Methoden der linearen Algebra (z. B. Gauss'sches Eliminationsverfahren)

Policy-Optimierung bei unendlichem Horizont

- wie im Fall endlichen Horizonts basiert Policy-Optimierung auf Berechnung der optimalen Wertfunktion V_*

Policy-Optimierung bei unendlichem Horizont

- wie im Fall endlichen Horizonts basiert Policy-Optimierung auf Berechnung der optimalen Wertfunktion V_*
- V_* erfüllt Bellman-Gleichung

$$V_*(s) = R(s) + \gamma \cdot \max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V_*(s') \quad \text{für alle } s \in S$$

Policy-Optimierung bei unendlichem Horizont

- wie im Fall endlichen Horizonts basiert Policy-Optimierung auf Berechnung der optimalen Wertfunktion V_*
- V_* erfüllt Bellman-Gleichung

$$V_*(s) = R(s) + \gamma \cdot \max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V_*(s') \quad \text{für alle } s \in S$$

- wieder Gleichungssystem mit $|S|$ Gleichungen und $|S|$ Variablen

Policy-Optimierung bei unendlichem Horizont

- wie im Fall endlichen Horizonts basiert Policy-Optimierung auf Berechnung der optimalen Wertfunktion V_*
- V_* erfüllt Bellman-Gleichung

$$V_*(s) = R(s) + \gamma \cdot \max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V_*(s') \quad \text{für alle } s \in S$$

- wieder Gleichungssystem mit $|S|$ Gleichungen und $|S|$ Variablen
- wegen max-Operator ist Gleichungssystem nichtlinear und damit nicht ganz einfach zu lösen

Policy-Optimierung bei unendlichem Horizont

- wie im Fall endlichen Horizonts basiert Policy-Optimierung auf Berechnung der optimalen Wertfunktion V_*
- V_* erfüllt Bellman-Gleichung

$$V_*(s) = R(s) + \gamma \cdot \max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V_*(s') \quad \text{für alle } s \in S$$

- wieder Gleichungssystem mit $|S|$ Gleichungen und $|S|$ Variablen
- wegen max-Operator ist Gleichungssystem nichtlinear und damit nicht ganz einfach zu lösen
- optimale Lösung ist Fixpunkt des Bellman-Operators $\mathcal{B}$, d. h. $\mathcal{B}[V_*] = V_*$, wobei

$$\mathcal{B}[V](s) := R(s) + \gamma \cdot \max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V(s')$$

Value Iteration für unendlichen Horizont

- analog zum endlichen Horizont kann optimale Policy durch Bellman-Updates berechnet werden:
  $V_0 = R;\ V_{t+1} = \mathcal{B}[V_t]$
- ausführlicher also für alle $s \in S$:

$$V^0(s) = R(s)$$
$$V^t(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} T(s,a,s') \cdot V^{t-1}(s')$$

- konvergiert gegen die optimale Wertfunktion!

$$\lim_{t \to \infty} V^t = V_*$$

- Wie viele Iterationen brauchen wir in der Praxis?

Konvergenz von Value Iteration

- Bellman-Backup $\mathcal{B}$ ist ein Kontraktionsoperator für Wertfunktionen, d. h. für alle V und V' gilt:

  $$\|\mathcal{B}[V] - \mathcal{B}[V']\| \leq \gamma \cdot \|V - V'\|$$

  Hierbei ist $\|V\|$ die Maximums-Norm; z. B. $\|V\| = 4.5$ für $V = \{s_1 \mapsto 0.1, s_2 \mapsto -4.5, s_3 \mapsto 3, s_4 \mapsto 2\}$

$\leadsto$ Bellman-Backups auf beliebigen Wertfunktion V und V' bringen diese näher zusammen

Konvergenz von Value Iteration (Fortsetzung)

- Insbesondere gilt für beliebige V:

$$\|V_* - \mathcal{B}[V]\| = \|\mathcal{B}[V_*] - \mathcal{B}[V]\| \leq \gamma \cdot \|V_* - V\|$$

- Fehler von V gegenüber V_* reduziert sich pro Schritt mit Faktor γ

Konvergenz von Value Iteration (Fortsetzung)

- Insbesondere gilt für beliebige V:

$$\|V_* - \mathcal{B}[V]\| = \|\mathcal{B}[V_*] - \mathcal{B}[V]\| \leq \gamma \cdot \|V_* - V\|$$

- Fehler von V gegenüber V_* reduziert sich pro Schritt mit Faktor γ
- mit etwas Rechnung: wenn $\|V^k - V^{k-1}\| \leq \delta$, dann $\|V_* - V^k\| \leq \frac{\delta\gamma}{1-\gamma}$

Konvergenz von Value Iteration (Fortsetzung)

- Insbesondere gilt für beliebige V:

$$\|V_* - \mathcal{B}[V]\| = \|\mathcal{B}[V_*] - \mathcal{B}[V]\| \leq \gamma \cdot \|V_* - V\|$$

- Fehler von V gegenüber V_* reduziert sich pro Schritt mit Faktor γ
- mit etwas Rechnung: wenn $\|V^k - V^{k-1}\| \leq \delta$, dann $\|V_* - V^k\| \leq \frac{\delta\gamma}{1-\gamma}$

$\rightsquigarrow$ um V_* mit einer gewünschten Genauigkeit ε zu approximieren:
  - Löse $\varepsilon = \frac{\delta\gamma}{1-\gamma}$ nach δ auf.
  - Stoppe Value Iteration, sobald Abstand zwischen V^k und V^{k-1} höchstens δ

Was ist die berechnete Policy?

- Angenommen, wir können V_* gut durch V^k approximieren.
- Welche Policy sollen wir ausführen?

Was ist die berechnete Policy?

- Angenommen, wir können V_* gut durch V^k approximieren.
- Welche Policy sollen wir ausführen?
- Antwort: gierige Policy bezüglich V^k (Ein-Schritt-Lookahead):

$$\text{greedy}[V^k](s) = \arg\max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V^k(s')$$

Was ist die berechnete Policy?

- Angenommen, wir können V_* gut durch V^k approximieren.
- Welche Policy sollen wir ausführen?
- Antwort: gierige Policy bezüglich V^k (Ein-Schritt-Lookahead):

$$\text{greedy}[V^k](s) = \arg\max_{a \in A} \sum_{s' \in S} T(s, a, s') \cdot V^k(s')$$

- Anmerkung: Wertfunktion von greedy$[V^k]$ ist nicht unbedingt gleich V^k!
- Wenn Fehler von V^k gegenüber V_* höchstens ε ist, dann ist Fehler von $V_{\text{greedy}[V^k]}$ gegenüber V_* höchstens $\frac{2\varepsilon\gamma}{1-\gamma}$

Diskussion von Value Iteration

- Value Iteration ist oft ein guter praktischer Algorithmus zum Finden einer optimalen Policy.
- Ein Nachteil ist, dass er nur approximativ ist, da V_* numerisch angenähert statt exakt berechnet wird. (Allerdings können wir V_* beliebig nahe annähern.)
- Wir stellen noch einen zweiten Algorithmus vor, der sich für exakte Optimierung eignet.
- Bei diesem Algorithmus wird nicht über (numerische) Wertfunktionen, sondern über (diskrete) Policys iteriert.

Policy-Optimierung durch Policy Iteration

- Gegeben feste Policy π können wir die Wertefunktion exakt berechnen (Policy-Bewertung durch Lösung eines LGS).
- Policy Iteration nutzt dies aus:
  wechsle Policy-Bewertung mit Policy-Verbesserung ab,
  bis die Wertfunktion sich nicht mehr ändert.

Policy Iteration

$\pi :=$ any policy, e.g. initialized randomly
repeat:
  Evaluate V_π by solving linear equations.
  if V_π is the same as in the previous iteration:
    return π
  for each $s \in S$:
    $\pi'(s) := \arg\max_{a \in A} \sum_{s' \in S} T(s, a, s') V_\pi(s')$
  $\pi := \pi'$

Eigenschaften von Policy Iteration

- Jeder Policy-Iteration-Schritt führt zu einer strikten Verbesserung der Policy in mindestens einem Zustand.
- Konvergenz nach endlich vielen Schritten ist garantiert (da es nur endlich viele verschiedene Policys gibt, sind nur endlich viele Verbesserungen möglich).
- Die berechnete Policy ist immer optimal.

Eigenschaften von Policy Iteration

- Jeder Policy-Iteration-Schritt führt zu einer strikten Verbesserung der Policy in mindestens einem Zustand.
- Konvergenz nach endlich vielen Schritten ist garantiert (da es nur endlich viele verschiedene Policys gibt, sind nur endlich viele Verbesserungen möglich).
- Die berechnete Policy ist immer optimal.

Wie viele Iterationen sind nötig?

- In der Praxis scheinen $O(|S|)$ Iterationen auszureichen...
- ...aber es sind keine in $|S|$ polynomiellen Garantien zur Iterationszahl bekannt ($\rightsquigarrow$ wichtiges offenes Problem!).

Value Iteration oder Policy Iteration

- Was ist schneller: Value Iteration oder Policy Iteration?
- kein Verfahren dominiert das andere: es hängt vom MDP ab
- Value Iteration benötigt mehr Iterationen als Policy Iteration, aber Policy Iteration benötigt mehr Zeit pro Iteration, weil Policy-Bewertung teuer ist (Lösung des LGS)