Moderne Methoden der KI: Maschinelles Lernen

Prof. Dr. sc. Hans-Dieter Burkhard
Vorlesung Sommer-Semester 2007

Verstärkungs-Lernen
(Reinforcement Learning)

Literatur:
R.S.Sutton, A.G.Barto
Reinforcement Learning
MIT Press 1998

Optimales Verhalten des Agenten

Steuerstrategie lernen:
- Aktionen abhängig von aktuellem Umwelt-Zustand
- Umwelt-Zustand verändert durch Aktionen
- Umwelt-Zustand wird bewertet („rewards“)
- Umwelt nur partiell erfahrbar
- Erfolgsbewertung verzögert

Umwelt i.a. nicht explizit gegeben
(aber bei dynamischer Programmierung/Optimierung)

Explorationsstrategie: Experimentieren mit möglichen Aktionen

Modelle für Interaktion Umwelt/Agent (Wdh.)

Dynamische Beschreibung in diskreter Welt

- Zusammenhänge zwischen Zuständen und Ereignissen
- Wie verändert sich die Welt durch Ereignisse

Agent beeinflusst die Welt durch Aktionen
- Strategie π legt Aktion fest
  (z.B. abhängig vom aktuellen Zustand/Ziel)

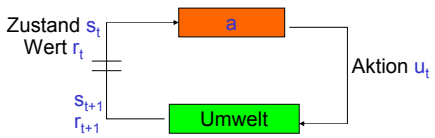
Modelle für Interaktion Umwelt/Agent

Ablaufbeschreibung mit Ereignissen e_t für diskrete lineare Zeit t

$s_0 \ -e_0\rightarrow \ r_1 s_1 \ -e_1\rightarrow \ r_2 s_2 \ -e_2\rightarrow \ ....$

oder mit Aktionen u_t eines Agenten a

$s_0 \ -u_0\rightarrow \ r_1 s_1 \ -u_1\rightarrow \ r_2 s_2 \ -u_2\rightarrow \ ....$



Markov-Bedingung

Abfolge

$s_0 \ -u_0\rightarrow \ r_1 s_1 \ -u_1\rightarrow \ r_2 s_2 \ -u_2\rightarrow \ .... \ u_{t-1}\rightarrow \ r_t s_t \ -u_t\rightarrow$

i.a. Zustand/Wert $r_{t+1} s_{t+1}$ zur Zeit $t+1$
abhängig von gesamter Vergangenheit:

$$P(s_{t+1} = s', r_{t+1}=r \ | \ s_0 u_0 r_1 s_1 u_1 r_2 s_2 u_2 \ldots u_{t-1} r_t s_t u_t)$$

Markov-Bedingung: $= P(s_{t+1} = s', r_{t+1}=r \ | \ s_t u_t)$
d.h. nur aus letztem Zustand/letzter Aktion berechenbar

Markov-Bedingung

Dynamik des Systems für **stochastisch** beschriebene Welt mit Markov-Bedingung :

$F: \quad S \times U \rightarrow$ Verteilungen über $S \times \mathcal{R}$
$P(s' = s_{t+1} \ , r = r_{t+1} \ | \ s_t u_t = su) = \ F(s,u)(s',r)$

daraus abgeleitet

Übergangswahrscheinlichkeit $\mathcal{P}_{sus'} := P(s' = s_{t+1} \ | \ s_t u_t = su)$
Reward-Wahrscheinlichkeit $\mathcal{P}_{sus'r} := P(r = r_{t+1} \ | \ s_t u_t s_{t+1} = sus')$
Erwartungswert für Reward $\mathcal{r}_{sus'} := E(r_{t+1} \ | \ s_t u_t s_{t+1} = sus')$

Werte (rewards), analog: Kosten

Für einen Agenten zur Zeit t : r_t bzw. $r_{sus'}$

Kumulativer Wert
für feste Folge $s_0 \xrightarrow{-u_0} r_1 s_1 \xrightarrow{-u_1} r_2 s_2 \xrightarrow{-u_2} \ldots$

Wert ab Zeit t : $R_t := \sum_{i=0\ldots\infty} \gamma^i r_{t+1+i}$
mit Discount-Faktor $0 < \gamma <= 1$

In stochastischer Welt:
Erwartungswert für R_t bei gegebener Strategie π

Markov-Entscheidungsprozess

Markov Decision Process (MDP) ist gegeben durch

- Diskrete lineare Zeitskala $t=0,1,2,\ldots$
- (Endliche) Menge S von Zuständen s,
  Agent kann s vollständig und zuverlässig beobachten
- (Endliche) Menge U von Aktionen u des Agenten
- Dynamik beschrieben mittels
  - Übergangswahrscheinlichkeiten $\mathcal{P}_{sus'}$
  - Erwartungswerten $r_{sus'} \in \mathcal{R}$

Der Agent muss sich zu jedem Zeitpunkt t für eine Aktion entscheiden.

Strategie (policy)

Deterministische MDP mit deterministischer Strategie:

$\pi: S \rightarrow U$

Agent entscheidet sich für die Aktion $u=\pi(s)$ in Abhängigkeit vom aktuellen Zustand s

Damit ergibt sich die Abfolge

$s_0 \xrightarrow{-u_0} r_1 s_1 \xrightarrow{-u_1} r_2 s_2 \xrightarrow{-u_2} \ldots. \xrightarrow{-u_{t-1}} r_t s_t \xrightarrow{-u_t} \ldots$

auf der Basis von Reward-/Übergangsfunktionen

$u_t = \pi(s_t)$
$r_{t+1} = f_r(s_t,u_t)$
$s_{t+1} = f_s(s_t,u_t)$

Wertfunktion (value function) V

In deterministischem MDP ist bei gegebener Strategie π für jeden Zustand s der Wert berechenbar:

$V^\pi: S \rightarrow \mathcal{R}$

$V^\pi(s)$ ist der Wert, der bei Start in s und Arbeit gemäß π insgesamt erreicht wird:

$V^\pi(s_t) := R_t = \sum_{i=0\ldots\infty} \gamma^i r_{t+1+i}$

Strategie (policy)

Für stochastische MDP:

$\pi: S \times U \rightarrow [0,1]$

Agent entscheidet sich mit Wahrscheinlichkeit $\pi(s,u)$ für die Aktion u in Abhängigkeit vom aktuellen Zustand s

Daraus ergeben sich Wahrscheinlichkeiten für die Abfolgen

$s_0 \xrightarrow{-u_0} r_1 s_1 \xrightarrow{-u_1} r_2 s_2 \xrightarrow{-u_2} \ldots. \xrightarrow{-u_{t-1}} r_t s_t \xrightarrow{-u_t}$

auf der Basis von

$\pi(s_i,u_i)$, $\mathcal{P}_{sus'}$, $\mathcal{P}_{sus'r}$

Wertfunktion (state-value function) V

Bei gegebener Strategie π ist für jeden Zustand s der Erwartungswert bestimmt:

$V^\pi: S \rightarrow \mathcal{R}$

$V^\pi(s)$ ist der Erwartungswert, der bei Start in s und Arbeit gemäß π insgesamt erreicht wird:

$V^\pi(s) := E(R_t \mid s = s_t, \pi) = E(\sum_{i=0\ldots\infty} \gamma^i r_{t+1+i} \mid s = s_t, \pi)$

Bellmann-Gleichung

$V^\pi(s)$
$:= E(R_t \mid s = s_t, \pi)$
$= E(\sum_{i=0\ldots\infty} \gamma^i r_{t+1+i} \mid s = s_t, \pi)$
$= E(r_{t+1} + \gamma \sum_{i=0\ldots\infty} \gamma^i r_{t+2+i} \mid s = s_t, \pi)$
$= \sum_u \pi(s,u) \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma E(\sum_{i=0\ldots\infty} \gamma^i r_{t+2+i} \mid s' = s_{t+1}, \pi))$
$= \sum_u \pi(s,u) \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma V^\pi(s'))$

Iterative Beziehungen zwischen den Werten $V^\pi(s)$

Bei $n = card(S)$ gibt es n lineare Gleichungen für n Werte.

Nach bekannten Verfahren lösbar.

Optimale Strategie π^*

π besser oder gleichwertig bzgl. π',
falls $V^\pi(s) \geq V^{\pi'}(s)$ für alle $s \in S$

Es gibt maximale Strategien (muss nicht eindeutig sein).

Maximale Strategie: π^*

Wertfunktion für maximale Strategie: V^*
$V^*(s) := \max_\pi V^\pi(s)$

Optimale Strategie π^*

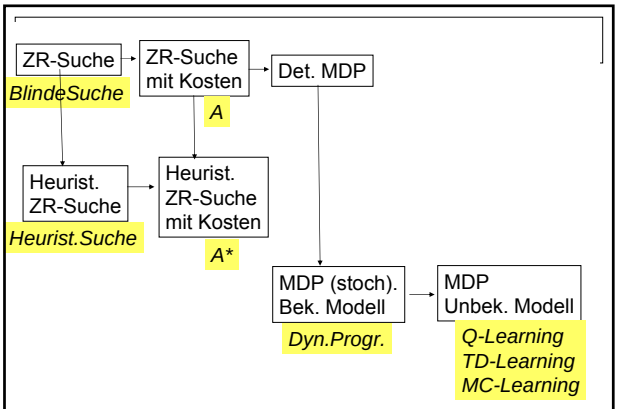
Bellmann-Gleichung
für Wertfunktion V^* der optimalen Strategie π^*

$V^*(s) = \max_u \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma V^*(s'))$

$n := card(S)$
n nicht-lineare Gleichungen für n Werte $V^*(s)$

Lösung mit Methoden der dynamischen Programmierung
Voraussetzung: Modell ist bekannt

Spezialfälle: Suchverfahren (vgl. EKI)



Q-Funktion (action-value function)

$Q^\pi: S \times U \rightarrow \mathcal{R}$

$Q^\pi(s,u)$ ist der Erwartungswert der rewards, bei Start in s mit Aktion u und Arbeit gemäß der π insgesamt erreicht wird:

$Q^\pi(s, u)$
$:= E(R_t \mid s=s_t, u=u_t, \pi) = E(\sum_{i=0\ldots\infty} \gamma^i r_{t+1+i} \mid s=s_t, u=u_t, \pi)$

Bellmann-Gleichung für Q-Funktion

$Q^\pi(s,u) := E(R_t \mid s = s_t, u=u_t, \pi)$
$= E(\sum_{i=0\ldots\infty} \gamma^i r_{t+1+i} \mid s = s_t, u=u_t, \pi)$
$= E(r_{t+1} + \gamma \sum_{i=0\ldots\infty} \gamma^i r_{t+2+i} \mid s = s_t, u=u_t, \pi)$
$= \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma \sum_{u'} \pi(s',u') E(\sum_{i=0\ldots\infty} \gamma^i r_{t+2+i} \mid s'=s_{t+1}, u'=u_{t+1}, \pi))$
$= \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma \sum_{u'} \pi(s',u') Q^\pi(s',u'))$

Iterative Beziehungen zwischen den Werten $Q^\pi(s,u)$

Bei $n = card(S)$ und $m = card(U)$ gibt es $n \cdot m$ lineare Gleichungen für $n \cdot m$ Werte.

Nach bekannten Verfahren lösbar (vgl. Wertfunktion V).

Zusammenhänge zwischen Q und V

$V^\pi(s) := \sum_u \pi(s,u)\ Q^\pi(s,u)$

$Q^\pi(s,u) := \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma V^\pi(s'))$

Optimale Strategie π^*

Q-Funktion für maximale Strategie π^*: $Q^*(s,u) := \max_\pi Q^\pi(s,u)$

Bellmann-Gleichung für Q^* :

$Q^{\pi*}(s,u) = \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma \max_{u'} Q^{\pi*}(s',u'))$

n := card(S), m = card(U) :
n·m nicht-lineare Gleichungen für n·m Werte $Q^*(s,u)$

Lösung mit Methoden der dynamischen Programmierung
Voraussetzung: Modell ist bekannt

Betrachtete Funktionen (Wdh.)

MDP mit
Übergangswahrscheinlichkeit $\mathcal{P}_{sus'} := P(s' = s_{t+1} \mid s_t u_t = su)$
Reward-Wahrscheinlichkeit $\mathcal{P}_{sus'r} := P(r = r_{t+1} \mid s_t u_t s_{t+1} = sus')$
Erwartungswert für Reward $r_{sus'} := E(r_{t+1} \mid s_t u_t s_{t+1} = sus')$

Strategie $\pi: S \rightarrow U$ bzw. stochastisch $\pi: S \times U \rightarrow [0,1]$

Wert-Funktion (state-value-function) $V^\pi: S \rightarrow \mathcal{R}$ für Strategie π
Q-Funktion (action-value function) $Q^\pi: S \times U \rightarrow \mathcal{R}$ für Strategie π

Optimale Funktionen (Wdh.)

Optimale Strategie $\pi^*: S \rightarrow U$
dazu:
Wert-Funktion (state-value-function) $V^*(s) = \max_\pi V^\pi(s)$
Q-Funktion (action-value function) $Q^*(s,u) = \max_\pi Q^\pi(s,u)$

Bestimmung von π^* anhand von V^* bzw. Q^* :
$\pi^*(s) = \max_u Q^*(s,u)$ Anwendbar ohne Kenntnis des Modells
$= \max_u (r_{sus'} + \gamma V^*(s'))$ Kenntnis des Modells notwendig

Statt π^* kann auch V^* bzw. Q^* bestimmt (gelernt) werden

Dynamische Programmierung

Voraussetzung: Modell bekannt, d.h.
Übergangswahrscheinlichkeit $\mathcal{P}_{sus'} := P(s' = s_{t+1} \mid s_t u_t = su)$
Reward-Wahrscheinlichkeit $\mathcal{P}_{sus'r} := P(r = r_{t+1} \mid s_t u_t s_{t+1} = sus')$
Erwartungswert für Reward $r_{sus'} := E(r_{t+1} \mid s_t u_t s_{t+1} = sus')$

Aufgaben:
- Für gegebene Strategie π die Wertfunktion V^π berechnen
- Optimale Strategie π^* bestimmen (alternativ V^* bzw. Q^*)

Strategie-Bewertung: V^π berechnen

Aufgabe:
Gegeben π, berechne V^π in bekanntem MDP
(Vorhersage-Problem)

1. Variante:
Lösen der Bellmann-Gleichung (Lineares System):

$V^\pi(s) = \sum_u \pi(s,u) \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma V^\pi(s'))$

Iterative Strategie-Bewertung

2. Variante: Iterative Strategie-Bewertung

Iteration anhand der Bellmann-Gleichung:

$V_0(s) := 0$

$V_{k+1}(s) := \sum_u \pi(s,u) \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma V_k(s'))$

Konvergenz gegen V^π

(Fixpunkt, Bellmann-Gleichung, Voraussetzung: V^π existiert)

"full backup": alle Nachfolge-Zustände berücksichtigen

Iterative Strategie-Bewertung

Implementation:

a) Zwei Tabellen für V_k und V_{k+1},
Einträge für V_{k+1} anhand der Werte für V_k
(entspricht simultaner Berechnung)

b) Nur eine Tabelle
überschreiben der Werte
konvergiert schneller
(Resultat für V_{k+1} abhängig von Reihenfolge)

Iteration abbrechen,
wenn Abstand zwischen V_k und V_{k+1} hinreichend gering

Strategie-Verbesserung

Zunächst deterministisch:

Greedy-Verfahren

Ausgehend von Strategie π wird neue Strategie π' definiert:

$\pi'(s) := \text{argmax}_u Q^\pi(s,u)$ (falls u nicht eindeutig: wählen)

Dann gilt für alle s : $V^{\pi'}(s) \geq V^\pi(s)$

und $V^{\pi'} = V^\pi \rightarrow V^{\pi'} = V^*$

Für stochastische Strategien analog.

Strategie-Iteration

Sukzessive neue Strategien π^{k+1} aus Strategie π^k definieren:

$\pi^{k+1}(s) := \text{argmax}_u Q^{\pi k}(s,u)$

Dann gilt für alle s und alle k: $V^{\pi k+1}(s) \geq V^{\pi k}(s)$

$V^{\pi 0} \leq V^{\pi 1} \leq V^{\pi 2} \leq V^{\pi 3} \leq \ldots \rightarrow V^*$

Konvergenz nach endlich vielen Schritten für endliche MDP.

Strategie-Iteration

Implementation:

Abwechselnd Strategie-Bewertung und Strategie-Verbesserung ausführen (Start beliebig, Abbruch bei Gleichheit bzw. geringer Verbesserung):

$\pi^0 \rightarrow_{Bew.} V^{\pi 0} \rightarrow_{Verb.} \pi^1 \rightarrow_{Bew.} V^{\pi 1} \rightarrow_{Verb.} \pi^2 \rightarrow_{Bew.} V^{\pi 2} \ldots \rightarrow_{Verb.} \pi^* \rightarrow_{Bew.} V^*$

Problem:
Strategie-Bewertung durch Iteration ist unendlicher Prozess.
Man muss aber nicht vollständig bewerten.

Wert-Iteration

Kombinierte Strategie-Bewertung und Strategie-Verbesserung, wobei nur Wertfunktion V explizit iteriert wird:

$V_{k+1}(s) := \max_u E(r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, u_t = u)$
$= \max_u \sum_{s'} \mathcal{P}_{sus'} (r_{sus'} + \gamma V_k(s'))$

Aus Ergebnis V (Abbruch wenn nur noch geringe Verbesserung) dann Strategie π bestimmen.

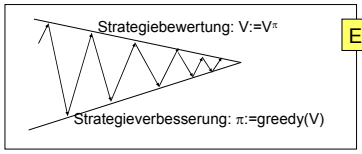
Verallgemeinerte Strategie-Iteration (GPI)

General Poliy Iteration

Interagierende Prozesse:
- Strategie-Bewertung
- Strategie-Verbesserung

Varianten dabei bzgl. Ausführlichkeit der jeweiligen Prozesse.

Sukkzessive Annäherung an optimale Strategie.



Ende bei π = greedy(V$^\pi$)

Lernen ohne Modell

Dynamische Programmierung/Optimierung:
optimale Strategie ermitteln unter Verwendung des Modells:
π^* bzw. V^* oder Q^*

Jetzt: Exploration
Modell unbekannt, aber durch Aktionen erkunden

Voraussetzung: Umwelt (Zustand, Reward) beobachtbar

Iteration ohne Modell

Grundidee:
Mit Aktionen Modell (partiell) erkunden und
mittels Iterationen Funktion bestimmen
Bei stochastischer Umwelt: Wiederholte Erkundung

Entspricht GPI-Ansatz:
- Aktionsauswahl nach jeweils verbesserter Strategie
- (Verkürzte) Bewertung der Strategie

Beispiel

Q-Learning für unbekanntes **deterministisches** MDP

$$f: S \times U \rightarrow S \times \mathcal{R}$$
$$[s_{t+1}, r_{t+1}] = f(s_t,u_t)$$

kann man aufspalten in :
Übergangsfunktion $f_s: S \times U \rightarrow S$
Rewardfunktion $f_r: S \times U \rightarrow \mathcal{R}$

$$s_{t+1} = f_s(s_t,u_t)$$
$$r_{t+1} = f_r(s_t,u_t)$$

Beispiel

Lernziel: optimale Aktions-Wert-Funktion Q^*
daraus ableitbar:
$\pi^*(s) = \text{ArgMax}_u\, Q(s,u)$,
$V^*(s) = \text{Max}_u Q(s,u)$

Bellmann-Optimalitäts-Gleichung (Iteration):
$Q^*(s,u) = f_r(f_s(s,u)) + \gamma\ \text{Max}_{u'} Q^*(f_s(s,u), u')$

ermöglicht iteratives Lernen

Q-Learning-Algorithmus (det. Modell)

Folge von Hypothesen Q_k:
$Q_{k+1}(s,u) = f_r(f_s(s,u)) + \gamma\ \text{Max}_{u'} Q_k(f_s(s,u), u')$

Beschreibung von z.B. in Tabellenform

update durch Auswertung von Explorationen:
$s' = f_s(s, u)$, $f_r(s')$

Lernen in aufeinander folgenden Episoden.

Q-Learning-Algorithmus (det. Modell)

Initial: Q_0: beliebig

Repeat for ever (für jede Episode)
- Q_i : Werte aus voriger Episode
- Anfangs-Zustand s bestimmen

Mit s iterieren:
- Aktion u **bestimmen** und ausführen .
- $s' = f_s(s, u)$, $r = f_r(s,u)$ beobachten
- Aktualisieren: $Q_{i+1}(s,u) := r + \gamma \, Max_{u'} Q_i(s', u')$
- $s := s'$

Q-Learning: Konvergenz (det. Modell)

Konvergenz von Q_k gegen Q^*
gilt auch in unendlichen Modellen, falls
- $0 <= \gamma < 1$
- $f_r(s,u)$ beschränkt durch Konstante c
- beliebige endliche initiale Werte
- alle Paare (s,u) werden unendlich oft betrachtet.

(Beweis)

Bei Initialisierung mit 0 gilt stets $0 <= Q_k(s,u) <= Q(s,u)$

Q-Learning: Aktionsauswahl

Generalized Strategy Iteration (GPI):
Aktionsauswahl nach jeweils verbesserter Strategie
(Verkürzte) Bewertung des erreichbaren Q-Werts

Auswahl-Strategie (nächste Aktion u):

Möglichkeit: „gierige (greedy) Auswahl“
$u:=argmax_{u'} (f_r(s,u') + \gamma \, Q_k(s', u'))$

Damit bestes bekanntes Resultat erzielt.
Problem: evtl. bessere Zustände werden nicht probiert.

Explorationstrategien

Problem (Exploration vs. Exploitation) :
Auswahl zwischen bekannten „guten“ Aktionen und
Experiment mit anderen Aktionen
(Konvergenzbedingung: alle (s,u) unendlich oft)

Auswahl-Wahrscheinlichkeiten (stochastische Strategie!)

z.B. $\pi(s,u) := c^{Qk(s,u)} / \Sigma_v c^{Qk(s,v)}$

Explorationstrategien

Weitere Varianten

Spezielle (günstige) Folgen verwenden
(ausgehend von den Zustandsübergängen mit rewards)

Simulation:
partiell: über bereits bekannte Folgen bei neuen Q_k-Werten
total: bei bekannten f_s und f_r (dynamische Programmierung)

Q-Learning für stochastische Modelle

Einfaches update (Überschreiben) nicht sinnvoll
(keine Konvergenz)

Stattdessen: Inkrementelle Annäherung an Erwartungswert

neu := alt + Schrittweite · (Messwert –alt)

Beispiel: n-armiger Bandit

Beispiel: „n-armiger Bandit“

Ohne Zustand, n Aktionen
$Q^*(u)$: Unbekannter (Erwartungs-)Wert für Aktion u
$Q_t(u)$: geschätzter Wert für u zur Zeit t
z.B. Mittelwert über bisher erhaltene rewards:
$Q_t(u) = 1/k \, \Sigma_{i=1\ldots k} \, r_i$ (konvergiert gegen $Q^*(u)$)

Auswahl einer Aktion zur Zeit t
Exploitation: nächste Aktion gemäß $\text{argmax}_u \, Q_t(u)$
Exploration: andere Aktion
(momentan schlechter, aber falls noch weitere Aktionen folgen sollen: evtl. bessere Alternativen finden)

n-armiger Bandit

greedy-Auswahl: immer Exploitation

ε-greedy : mit Wkt. $\varepsilon > 0$ andere Aktion
Softmax: $P(u) = e^{Qt(u)/\tau} \; / \; \Sigma_v \, e^{Qt(v)/\tau}$
$\tau > 0$ „Temperatur“ (Boltzmann-Verteilung)

Optimistischer Anfangswert
führt dazu, dass zunächst exploriert wird

n-armiger Bandit

Inkrementele Berechnung:

$$Q_{t+1}(u) = (k_u+1)^{-1} \; \Sigma_{i=1\ldots ku+1} \, r_i$$
$$= (k_u+1)^{-1} \, (r_{t+1} + \Sigma_{i=1\ldots ku} \, r_i \,)$$
$$= (k_u+1)^{-1} \, (r_{t+1} + k_u \, Q_t(u) + Q_t(u) - Q_t(u) \,)$$
$$= (k_u+1)^{-1} \, (r_{t+1} + (k_u+1) \, Q_t(u) - Q_t(u) \,)$$
$$= Q_t(u) + (k_u+1)^{-1} \, (r_{t+1} - Q_t(u) \,)$$

Allgemeine Form:
neu := alt + Schrittweite · (Messwert –alt)
= alt + Schrittweite · Fehler

n-armiger Bandit

$Q_{t+1}(u) = Q_t(u) + (k_u+1)^{-1} \, (r_{i+1} - Q_t(u) \,)$

Allgemeiner $Q_{t+1}(u) = Q_t(u) + \alpha_t \, (r_{t+1} - Q_t(u) \,)$

Schrittweite α_t variieren für bessere Konvergenz,

Konvergenz mit Wkt. 1 falls $\Sigma_{i=0\ldots\infty} \, \alpha_i = \infty$ und $\Sigma_{i=0\ldots\infty} \, \alpha_i^2 < \infty$

Q-Learning für stochastisches Modell

Initial: Q_0: beliebig
Repeat for ever (für jede Episode)
– Q_i : Werte aus voriger Episode
– Anfangs-Zustand s bestimmen
Mit s iterieren:
– Aktion u **wählen** (z.B. ε-greedy) und ausführen .
– s‘ , r beobachten
– Aktualisieren:
$Q_{i+1}(s,u) := Q_i(s,u) + \alpha \, (r + \gamma \; \text{Max}_{u'} \, Q_i(s', u') - Q_i(s,u) \,)$
– s := s‘

Dabei ist $r + \gamma \; \text{Max}_{u'} \, Q_i(s',u')$ der neue Messwert
Temporal difference: $r + \gamma \; \text{Max}_{u'} \, Q_i(s',u') - Q_i(s,u)$

TD-Lernen

TD = Temporal difference (entspricht der Abweichung vom bisher angenommen Wert)

(Später mehr dazu)

Auch für andere Probleme anwendbar

Allgemeine Formel: update = Lernrate * Fehler
(vgl. delta-Regel für NN, Gradientenverfahren usw.)

Haben bis jetzt für stochastische Modelle:

	Modell bekannt (DP)	Modell unbekannt (RL)
V^π	• Lin. Glgssytem • Iteration	
Q^π		
π^*	Strategie-Iteration	
V^*	Iteration	
Q^*		Q-Lernen

Für deterministische Modelle auch einfachere Iterationsverfahren

Monte Carlo (MC) Lernen

- Modell nicht bekannt
- Lernen in Episoden (!)
- Mittlung der interessierenden Werte aus Episoden

Prinzip GPI für Steuerung des Agenten

MC: Strategiebewertung für π

Initial: Liste $Returns(s_i,u_i)$ leer, V beliebig
Repeat for ever

- Episode (endl. Zustandsfolge $s_0,s_1,s_2, ..., s_n$ mit rewards) gemäß π generieren und beobachten
- Für alle s_i den dabei erreichten Wert R_i ermitteln und an Liste $Returns(s_i)$ anhängen
- $V(s_i) := Mittelwert(Returns(s_i))$

MC: Strategiebewertung (Policy evaluation) für geg. π

Varianten:
- jedes Auftreten eines Zustands s berücksichtigen
- nur das erste Auftreten („first visit“)

- Evtl. nicht alle Zustände in den Episoden berücksichtigen (dadurch attraktiv)

- Konvergenz gemäß Gesetz der großen Zahlen (unendliche viele Besuche der Zustände)

MC: Strategiebewertung für Q^π-Funktion

Analog zu Strategie-Bewertung.

Problem:
Bei deterministischer Strategie π wird im Zustand s stets nur Aktion $u=\pi(s)$ ausgeführt, d.h. keine vollständige Exploration.

Explorationsproblem (vgl. n-armiger Bandit)

Möglichkeit:
initial kann jedes Zustands-Aktions-Paar (s,u) unendlich oft auftreten

Optimale Strategie

In unbekanntem Modell kann optimale Strategie π^* aus Q^* bestimmt werden:

$$\pi^*(s) = \max_u Q^*(s,u)$$

(nicht möglich aus V^*)

MC: Optimale Q-Funktion lernen

Initial: Returns(s_i,u_i) leer, Q beliebig, π beliebig (stoch.)

Repeat for ever

- Episode (endl. Zustandsfolge $s_0,s_1,s_2, ..., s_n$ mit rewards) gemäß aktueller Strategie π generieren und beobachten
- Für alle (s_i,a_i) den dabei erreichten Wert R_i ermitteln und an Liste Returns(s_i,u_i) anhängen
- $Q(s_i,u_i)$:= Mittelwert(Returns(s_i,u_i))
- Verbesserte Strategie π z.B. gemäß (ε-greedy)
  - $\pi(s,u):=1-\varepsilon+\varepsilon/card(U)$ für $u := argmax_u\ Q(s,u)$
  - $\pi(s,u):= \varepsilon/card(U)$ sonst

folgt dem GPI-Schema

MC: Optimale Q-Funktion lernen

folgt dem GPI-Schema:

Es erfolgen abwechselnd Schritte

- zur Strategieverbesserung (neue Strategie π)
- Strategiebewertung (Q^π), aber nicht vollständig

- ε-greedy statt Start mit unterschiedlichen s_0, u
- Approximation der optimalen Q-Funktion bzw. Strategie.
- Zusätzliche Analyse bzgl. Stabilität des Prozesses für deterministische Strategie (Abbruch bei Stabilität: Optimale Strategie gefunden)

MC: Off-Policy

Bisher:

Wertfunktion/Q-Funktion für Strategie π wird gelernt durch Episoden, die mit π erzeugt werden

Alternative:

Wertfunktion/Q-Funktion für Strategie π wird gelernt durch Episoden, die mit einer anderen Strategie π' erzeugt werden

MC: Off-Policy

Vorteile:

Strategie π kann greedy (deterministisch) sein

Strategie π' erzeugt die notwendigen Beispiele

Dafür notwendig:

1. $\pi(s,u) > 0 \rightarrow \pi'(s,u) > 0$
2. Ausgleich der unterschiedlicher Wahrscheinlichkeiten für die durchlaufenen Folgen bei π bzw. π'

MC: inkrementelles Vorgehen

$V(s_i)$:= Mittelwert(Returns(s_i) inkrementell berechnen

Verallgemeinerung: gewichteter Mittelwert (für off-policy)

$V_n = \Sigma_{i=1...n} w_i R_i \ / \ \Sigma_{i=1...n} w_i$

Inkrementell:

$V_{n+1} = V_n + w_{n+1} / W_{n+1} (R_{n+1} - V_n)$

mit $W_{n+1} = W_n + w_{n+1}$ und $V_0 = W_0 = 0$

$$
\begin{aligned}
& V_n + w_{n+1} / W_{n+1} (R_{n+1} - V_n) \\
&= 1/ W_{n+1} (W_{n+1} V_n + w_{n+1} R_{n+1} - w_{n+1} V_n) \\
&= 1/ W_{n+1} ((W_{n+1} - w_{n+1}) V_n + w_{n+1} R_{n+1}) \\
&= 1/ W_{n+1} ((W_n V_n + w_{n+1} R_{n+1}) \\
&= 1/ W_{n+1} (\Sigma_{i=1...n} w_i R_i + w_{n+1} R_{n+1}) \\
&= 1/ W_{n+1} (\Sigma_{i=1...n+1} w_i R_i) = V_{n+1}
\end{aligned}
$$

Temporal Difference Learning (TD)

Allgemeine Idee:

- Differenz zwischen letzter Schätzung und aktuellem Messwert zur Korrektur verwenden: Richtung der Änderung
- Schrittweite der Änderung durch Wichtungsfaktor α gegeben mit der Zeit abnehmend, z.B.
  α := 1 / (1+Anzahl der bisherigen Aktionen u in s)

TD(0) für Strategiebewertung V^π

Initial: V beliebig
Für jede Episode:
- Startzustand s_0 bestimmen
  V : Werte aus vorheriger Episode
- Wiederhole für jeden Zustand s der Episode bis zum Ende
  Aktion $u := \pi(s)$ ausführen
  Folgezustand s' und reward r beobachten
  $V(s) := V(s) + \alpha\ (r + \gamma V(s') - V(s))$
  $s := s'$

Konvergenz mit Wkt. 1 bei unendlich vielen Besuchen

Vergleich MC und TD

Update jeweils für besuchten Zustand (analog Zustand/Aktion):

Nach Besuch von s_t erfolgt update von $V(s_t)$

- MC: mittels R_t vom Ende der Episode (gemittelt bei mehrmaligem Auftreten)

- TD: mittels r_{t+1} und $V(s_{t+1})$
  („Bootstrapping-Verfahren“)

Vergleich MC, DP und TD

Bellmann-Glg.:

$$V^\pi(s) = E(R_t \mid s = s_t, \pi)$$ Update für MC
$$= E(\sum_{i=0\ldots\infty} \gamma^i r_{t+1+i} \mid s = s_t, \pi)$$
$$= E(r_{t+1} + \gamma \sum_{i=0\ldots\infty} \gamma^i r_{t+2+i} \mid s = s_t, \pi)$$
$$= E(r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s = s_t, \pi)$$ Update für DP

Tatsächlich werden jeweils nur Approximationen verwendet:

MC: Beispielwert für R_t (gemittelt über bisherigen Episoden)

DP: bisherige Schätzung V_i^π für V^π (bootstrapping)

TD: Beispielwert für r_{t+1} und bisherige Schätzung V_i^π

Q-Learning (Wdh.)

Initial: Q: beliebig
Repeat for ever (für jede Episode)
- Q : Werte aus voriger Episode
- Anfangs-Zustand s bestimmen

Mit s iterieren:
- Aktion u **wählen** (z.B. ε-greedy) und ausführen .
- s' , r beobachten
- Aktualisieren (mit bereits neueren Q-Werten):
  $Q(s,u) := Q(s,u) + \alpha (r + \gamma \ Max_{u'} Q(s',u') - Q(s,u))$
- $s := s'$ Off-policy

Sarsa (anderes GPI-Schema)

state action reward state action

Initial: Q beliebig
Repeat for ever (für jede Episode)
- Anfangs-Zustand s bestimmen

Wiederhole für jeden Zustand s der Episode
- Aktion u wählen (z.B. ε-greedy gemäß Q) und ausführen .
- s' , r beobachten
- Aktion u' für s' wählen (z.B. ε-greedy gemäß Q) und ausführen
- Aktualisieren: on-policy
  $Q(s,u) := Q(s,u) + \alpha (r + \gamma \ Q(s',u') - Q(s,u))$
  $s := s'$, $u:=u'$

statt bei off-policy:
$r + \gamma \ Max_{u'} Q(s',u') - Q(s,u)$

TD(λ): Vorwärtsbetrachtung

$$s_0 \ -u_0\rightarrow\ r_1 s_1 \ -u_1\rightarrow\ r_2 s_2 \ -u_2\rightarrow\ \ldots.$$

Wert ab Zeit t : $R_t := \sum_{i=0\ldots\infty} \gamma^i r_{t+1+i}$
Approximation gemäß 1-Schritt-Vorausschau: $r_{t+1} + \gamma \ V_t(s_{t+1})$

Alternativ: Approximation mit Vorausschau von n Schritten ab s_t
n-Schritt-Returns $R_t^{(n}$

n=1: $R_t^{(1)} := r_{t+1} + \gamma \ V(s_{t+1})$
n=2: $R_t^{(2)} := r_{t+1} + \gamma\, r_{t+2} + \gamma^2 V(s_{t+2})$
...
n: $R_t^{(n)} := r_{t+1} + \gamma\, r_{t+2} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n})$
usw.

TD(λ): Vorwärtsbetrachtung

Gleichzeitige Benutzung aller Vorausschaumöglichkeiten bei Kombination mit Wichtungsfaktor λ (recency factor) :

($0 <= \lambda <= 1$, Gesamtgewicht = 1)

$R_t^{\lambda} := (1-\lambda)\,(R_t^{(1)} + \lambda\, R_t^{(2)} + \lambda^2 R_t^{(3)} + \ldots)$

$= (1-\lambda) \sum_{i=1,\ldots} \lambda^{i-1} R_t^{(i)}$

TD(λ): Vorwärtsbetrachtung

Bei einer Episode der Länge T wird abschließender Return R_t erreicht bei : $R_t^{(T-t-1)} = R_t$

Danach weiter für $i>0$: $R_t^{(T-t-1+i)} = R_t$

$R_t^{\lambda} = (1-\lambda) \sum_{i=1,\ldots T-t-1} \lambda^{i-1} R_t^{(i)} + (1-\lambda) \sum_{T-t,\ldots,} \lambda^{i-1} R_t$

$= (1-\lambda) \sum_{i=1,\ldots T-t-1} \lambda^{i-1} R_t^{(i)} + \lambda^{T-t-1} R_t$

Falls $\lambda = 1$: $R_t^{\lambda} = R_t$ entspricht MC-Lernen

Falls $\lambda = 0$: $R_t^{\lambda} = R_t^{(1)}$ entspricht bisherigem TD-Lernen

(daher Bezeichnung TD(0))

Insgesamt bei $0 < \lambda < 1$: Kombination von beidem

TD(λ): Rückwärtsbetrachtung

Nach jeder Aktion erfolgt update der bereits früher besuchten Zustände.

Dabei Wichtung der Zustände entsprechend der Häufigkeit der Besuche („eligibility trace“ = Spur) mit den Parametern γ , λ

Für jeden Zeitpunkt t : $e_t: S \rightarrow \mathcal{R}$

$e_t(s) := \gamma \lambda\, e_{t-1}(s)$ für $s \neq s_t$

$e_t(s) := \gamma \lambda\, e_{t-1}(s) + 1$ für $s = s_t$

TD(λ): Strategiebewertung V^{π}

Initial: V(s) beliebig, e(s)=0 für alle $s \in S$

Repeat (für jede Episode)

Initial: s wählen

Repeat (für jeden Schritt der Episode)

Aktion $u=\pi(s)$ ausführen,

reward r und Folgezustand s‘ beobachten

Korrekturwert $\delta := r + \gamma\, V(s') - V(s)$

e-Wert für s erhöhen: $e(s) := e(s)+1$

α ist Lernrate

Für alle $s \in S$ neue Schätzung: $V(s) := V(s) + \alpha \cdot \delta \cdot e(s)$

Alle e-Werte anpassen: $e(s) := \gamma \cdot \lambda \cdot e(s)$

$s := s'$

TD(λ)

TD(1) d.h. $\lambda = 1$:

entspricht MC-Lernen, aber inkrementelle Realisierung

Variationen von λ

- Zeitlich
- Zustandsabhängig

$e(s) := \gamma \cdot \lambda(s) \cdot e(s)$

Bei guter Schätzung in s : $\lambda(s) \rightarrow 0$ (hoher Einfluss)

Bei schlechter Schätzung: $\lambda(s) \rightarrow 1$ (geringer Einfluss)

Sarsa(λ)

Initial: Q beliebig, e(s,u)=0 für alle s, u

Repeat (für jede Episode)

– s, u bestimmen

Repeat (für jeden Schritt der Episode)

– Aktion u wählen (z.B. ε-greedy gemäß Q) und ausführen .

– s‘ , r beobachten

– u‘ für s‘ wählen (z.B. ε-greedy gemäß Q) und ausführen

– Korrektur-Wert $\delta := r + \gamma\, Q(s',u') - Q(s,u)$

– e-Wert für s erhöhen: $e(s) := e(s)+1$

– für alle s, u neue Schätzung: $Q(s,u) := Q(s,u) + \alpha\, \delta\, e(s,u)$

– Alle e-Werte anpassen: $e(s) := \gamma \cdot \lambda \cdot e(s)$

$s := s'$, $u := u'$

Q(λ): Q-Learning erweitert

Off-Policy-Verfahren

Problem:
„Exploitation"-Aktionen (nicht greedy gewählt)
können Verfahren verfälschen

Eligibility-Trace entsprechend modifizieren:

Verfahren von Watkins

Q(λ): Verfahren von Watkins

Initial: Q beliebig, e(s,u)=0 für alle s, u
Repeat (für jede Episode)
- s, u bestimmen

Repeat (für jeden Schritt der Episode)
- Aktion u wählen (z.B. ε-greedy gemäß Q) und ausführen .
- s', r beobachten
- u' für s' wählen (z.B. ε-greedy gemäß Q) und ausführen
- Greedy-Aktion bestimmen: $u^* := \text{argmax}_v Q(s',v)$
- Korrektur-Wert $\delta := r + \gamma\, Q(s',u^*) - Q(s,u)$
- e-Wert für s erhöhen: $e(s) := e(s)+1$
- für alle s, u neue Schätzung: $Q(s,u) := Q(s,u) + \alpha\, \delta\, e(s,u)$
- e-Werte anpassen: falls $u'=u^*$: $e(s,u) := \gamma \cdot \lambda \cdot e(s,ua)$
  sonst: $e(s,u) := 0$

$s := s'$, $u := u'$

Für große/kontinuierliche Modelle

Tabelle für V(s) bzw. Q(s,a) ersetzen durch Approximation
- Funktionen
- Neuronale Netze, …

Statt verbessern der Tabellenwerte:
Lernverfahren für die Approximationen
(Gradientenabstieg, BackPropagation, …)
Trainingsbeispiele aus den beobachteten Werten

Konvergenz nicht gesichert

Behandelte Probleme

	Mit Modell DP	Ohne Modell MC	Ohne Modell Temporal Difference	Ohne Modell Eligibility Traces
Bewertung (prediction) V^π	Lin-.Glgen. Iteration	Episoden	TD(0)	TD(λ)
Bewertung Q^π				
Strategie-Iteration π^*	Iteration			
Wert-Iteration V^*	Iteration			
Q-Learning Q^*, Control (GPI)		Episoden	Q-Learning Sarsa	Watkins Sarsa(λ)

Modellierung beim Verstärkungslernen

MDP festlegen:
- Beschreibungen für Zustände, Aktionen: S, U
- Übergangsfunktion: Tabelle, lin. Funktion, Neuron. Netz, …
- Rewardfunktion: Tabelle, lin. Funktion, Neuron. Netz, …

Lernverfahren:
- Episoden
- Akkumulation von rewards: Parameter γ
- on-policy vs. off-policy
- Exploration vs. Exploitation: ε-greedy, Parameter ε
- Schrittweite α für TD-Lernen
- Recency-Faktor λ für TD(λ)-Lernen

Konvergenz-Aussagen

Notwendig:
Alle Zustands-Aktionspaare werden hinreichend oft besucht
- in endl. stochastischen MDP: unendlich oft
- in endl. Determinist. MDP: endlich Anzahl von Besuchen
- in unendl. MDP: Problem der Approximationsmodelle für Zustandsübergangs-Funktion/Reward-Funktion

Beispiel:
Konvergenz beim Q-Lernen gegen Q^* , falls
- $0 <= \gamma < 1$
- $f_r(s,u)$ beschränkt durch Konstante c
- beliebige endliche initiale Werte
- alle Paare (s,u) werden unendlich oft betrachtet