Seminar Lernende Agenten im WWW (WS 1998/99)

Reinforcement Learning

Referat über den Artikel
"An Introduction to Reinforcement Learning"
von Leslie Pack Kaelbling, Michael L. Littman und Andrew W. Moore

Volkher Kaschlun
Fachbereich Informatik

6 November 1998

Gesamtübersicht

Einführung

reinforcement learning

beliebtes Modell eines Agenten,
lernt durch trial-and-error in Wechselwirkung mit einer dynamischen Umgebung

hat starke Ähnlichkeit mit Arbeiten in der Psychologie, aber Unterschiede in den Details und im Gebrauch des Wortes *reinforcement*

Unterschiede zum supervised learning:

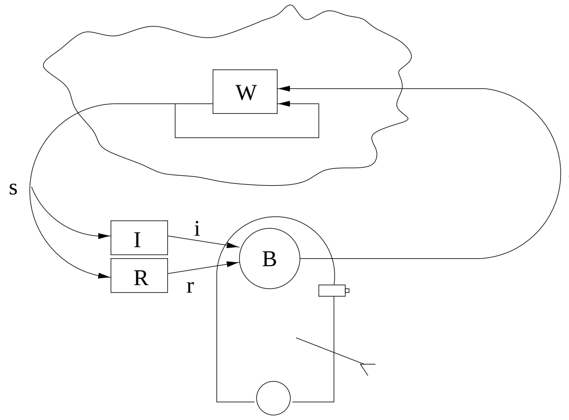
- keine input/output-Paare
- Agent erfährt sofort nach Wahl einer Aktion den "Lohn" und den Folgezustand
- er erfährt nicht, welches die beste Aktion für maximal lange Laufzeit wäre
- Notwendigkeit aktiv Wissen über mögliche Systemzustände, -Aktionen,-Übergänge und -Belohnungen zu sammeln
- online-performance ist wichtig
- möglichst keine Aufteilung in Lern- und Performance-Phasen

reinforcement learning Modell

Standard-Modell:
Agent ist mit seiner Umgebung via Wahrnehmung und Aktionen verbunden.

* Agent empfängt Hinweise auf den aktuellen Zustand der Umgebung
* Agent wählt daraufhin Aktion, generiert als Ausgabe

Aktion ändert Zustand der Umgebung; Wert des neuen Zustandes als spezieller skalarer *reinforcement input*
Ziel: Erhöhung der Summe der Werte des reinforcement Signals



Das Standard reinforcement learning Modell

Formal:

- diskrete Menge von Umgebungszuständen S
- diskrete Menge von Agenten-Aktionen A
- eine skalare Menge von reinforcement-Werten; meist $\{0, 1\}$ oder reele Zahlen
- Eingabefunktion I; bestimmt die Sicht des Agenten auf den Umgebungszustand, im weiteren nehmen wir die Identitätsfunktion an. (d.h. der Agent erkennt den genauen Zustand)

Beispiel:

Umgebung: Du bist in Zustand 65. Du hast 4 mögliche Aktionen

Agent: Ich nehme Aktion 2.

Umgebung: Du erhältst ein reinforcement von 7 Einheiten. Du bist nun im Zustand 15. Du hast 2 mögliche Aktionen

Agent: Ich nehme Aktion 1.

Umgebung: Du erhältst ein reinforcement von -4 Einheiten. Du bist nun im Zustand 65. Du hast 4 mögliche Aktionen

Agent: Ich nehme Aktion 2.

Umgebung: Du erhältst ein reinforcement von 5 Einheiten. Du bist nun im Zustand 44. Du hast 5 mögliche Aktionen

...

Aufgabe: finde Taktik $\pi : S \to A$, die einige langlebige Maße des reinforcement maximiert.
I.A. nehmen wir an:

- die Umgebung ist nicht deterministisch, d.h. gleiche Aktionen in gleichen Zuständen führen zu unterschiedlichen Zuständen und reinforcement-Werten.

- die Umgebung ist fest. d.h. die Wahrscheinlichkeiten Zustandsveränderungen zu machen oder reinforcement-Werte zu empfangen ändert sich nicht.

Modelle des optimalen Verhaltens

finite-horizon Modell:
zu einem gegebenen Zeitpunkt soll der Agent seine Belohnungen für die nächsten k-Schritte optimieren:

$$E(\sum_{t=0}^{k} r_t)$$

Auf zwei Arten nutzbar:

- nicht-statische Taktik:
  1. Schritt: wähle k-Schritt optimale Aktion
  2. Schritt: wähle k-1-Schritt optimale Aktion

  ⋮

  letzter Schritt: wähle 1-Schritt optimale Aktion und ende
- *receding horizon controll*
  - wählt immer die k-Schritt optimale Aktion
  - benutzt immer die gleiche Taktik
  - aber der Wert von k begrenzt die Sichtweite des Agenten

  finite-horizon Modell kann mehr Rechenzeit benötigen als andere Modelle; gewöhnlich ist nicht genau bekannt, wieviel Zeit der Agent hat.

average-case Modell:
Agent wählt Aktionen, die die durchschnittliche Belohnung optimieren:

$$lim_{k\to\infty}E(\frac{1}{k}\sum_{t=0}^{k}r_t)$$

Problem: keine Möglichkeit zwischen zwei Strategien zu unterscheiden.

1. größere Belohnungen in der Anfangsphase

2. tue ersteres nicht

Belohnung in der Anfangsphase wird duch die durchschnittliche Leistung überlagert.

Lösung?

infinite-horizon discounted Modell:

- verallgemeinerung der beiden Modelle

- zieht die Belohnung über lange Laufzeit des Agenten in Betracht

- aber zukünftige Belohnungen werden exponentiell nachlassend entsprechen dem discount-Faktor γ $[0,1]$ berücksichtigt.

$$E(\sum_{t=1}^{\infty}\gamma^t r_t)$$

Bemessen der Lernleistung

Ein Maß für das Optimum gibt uns ein Kriterium, um die Qualität eines Lernalgorithmus zu beurteilen
Leider gibt es unterschiedliche Maße:

- letztendliche Konvergenz zum Optimum

  - viele Algorithmen haben eine beweisbare Garantie für eine asymptotische Konvergenz zum Optimum.

  - nicht-vollständig für die Untersuchung der online-performance
    Beispiel: schnell und preiswert lernender Agent, der nur 99 % des Optimums erreicht, ist oft besser als ein langsam Lernender, der garantiert 100 % erreicht.

- Konvergenzgeschwindigkeit zum Optimum

  - Optimum ist gewöhnlich ein asymptotische Ergebniss; dehalb ist die Konvergenzgeschwindigkeit ein schlechtes Maß.

  - Praktikabler: Geschwindigkeit der Konvergenz zum fast-optimalen (*speed convergence to near-optimality*). Dieses Maß geht aber an der Definition des, wie Nah am Optimum ausreichend ist, vorbei.

  - verwandtes Maß ist die Höhe der Leistung nach einer gegebener Zeit. Erfordert aber die Definition der gegebenen Zeit.

- Regret
  - Maß der zu erwartenden Abnahme der Belohnung aufgrund der Ausführung des Lernalgorithmus, anstatt sich von Anfang an optimal zu verhalten.
  - Bezieht sich auf die Idee der *mistake bounds*.
  - Bestraft Fehler, wo immer sie auch während der Laufzeit auftreten.

Exploitation versus Exploration

Verwendung des Gelernten
versus
Lernen während der Verwendung

Beispiel: the two-armed bandit
Agent befindet sich in einem Raum mit zwei Spielautomaten (einarmiger Bandit).
Es ist kein Guthaben zum Spielen nötig.
Wenn Hebel i betätigt, zahlt Automat i 1 oder 0 aus, bezüglich Wahrscheinlichkeitsparameter p_i.
Wie soll die Strategie aussehen?
D.h. welcher Hebel soll in Schritt t betätigt werden, bei einer gegebenen Liste der zuvor gewählten Hebel und ihrer Auszahlungen?

Unterschied zwischen Verwendung des Gelernten und Lernen während der Verwendung!

Agent kann glauben, daß ein Hebel ziemlich hohe Auszahlungswahrscheinlichkeit hat;
⇒ diesen die ganze Zeit wählen oder einen anderen, über welchen wenig Information existiert, der aber schlecht zu sein scheint?
Je länger das Spiel dauert, um so schlechter sind die Konsequenzen eines frühzeitigen Konvergierenz zu einem suboptimalen Hebel!
⇒ Um so mehr sollte der Agent lernen!

Delayed reinforcement

Markov Decision Processes

Probleme mit delayed reinforcement sind gut modelliert als *Markov decision processes*(MDP). Ein MDP besteht aus:

- einer endlichen Menge von Zuständen *S*
- einer endlichen Menge von Aktionen *A*
- einer Belohnungsfunktion $R : S \times A \rightarrow \Re$
- einer Zustandsüberführungsfunktion $T : S \times A \rightarrow \Pi(S)$, mit $\Pi(S)$ ist eine Menge von Wahrscheinlichkeitsverteilungen über die Menge *S*.

Finden einer Taktik bei gegebenem Modell

Definition: Der optimale Wert eine Zustandes ist die erwartete unendlich abnehmende Summe von Belohnungen, welch der Agent erhält, wenn er in diesem Zustand starten und die optimale Taktik ausführt. Sie wird geschrieben

$$V^*(s) = max_\pi E(\sum_{t=0}^{\infty} \gamma^t r_t).$$

Diese Optimuswertfunktion ist eindeutig und kann rekursiv definiert werden als

$$V^*(s) = max_a(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')),$$

die optimale Taktik ist dann

$$\pi^*(s) = arg\ max_a(R(s,a)+\gamma \sum_{s' \in S} T(s,a,s')V^*(s')).$$

Werte Iteration:
Algorithmus *Werte Iteration*:

initialisiere $V(S)$ willkürlich
loop
  loop für $s \in S$
    $V(s) := max_a(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')$
  end loop
end loop

Eine approximale Optimumsfunktion bringt eine approximal gute Taktik hervor:

$$|V^* - \widehat{V}|_{sup} \leq \epsilon \Rightarrow |V^* - V_{\widehat{\pi}}|_{sup} \leq 2\epsilon \frac{\gamma}{1-\gamma}$$

- Nicht eindeutig, wann der Werte Iterations Algorithmus beendet werden kann.
- *Bellman residual* Theorem: Differenz zweier aufeinanderfolgender Wertefunktionen $> \epsilon \Rightarrow$ maximale Differenz einer solchen und der optimale Wertefunktion $> 2\epsilon\gamma/(1-\gamma)$.
- Die Kosten sind polynomiell in der Anzahl der Zustände.

Taktik Iteration:
Algorithmus *Taktik Iteration*:

wähle eine beliebige Taktik π
loop
  berechne den Wert der Taktik-Funktion π:
  $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s,a,s')V_\pi(s'))$
  verbessere die Taktik in jedem Zustand:
  $\pi'(s) := arg\ max_a$
      $(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V_\pi(s'))$
  $\pi := \pi'$
bis keine Verbesserungen mehr möglich sind

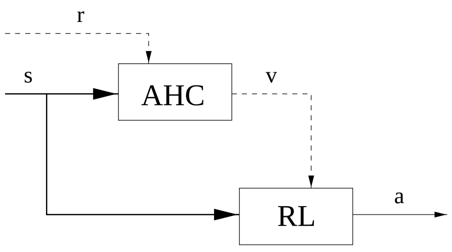
Lernen einer Taktik ohne Anfangsmodell

- bisherige Methoden betrachteten wir unter der Annahme eines bereits existierenden Modells
- Modell besteht aus Wissen über:
  1. die Wahscheinlichkeitsfunktion der Zustandsüberführung $T(s,a,s')$
  2. die reinforcement Funktion $R(s,a)$
- ein solches Modell ist aber im vorraus nicht bekannt!
- Stattdessen soll der Agent mit der realen Welt interagieren, um Informationen zu erhalten, die mit den Möglichkeiten eines geigneten Algorithmus verarbeitet werden können, um eine optimale Taktik zu erzeugen.
- Es gibt nun zwei Wege um fortzufahren:
  1. Model-based: Lerne ein Modell, und benutze es, um einen Controller abzuleiten.
  2. Model-free: Lerne einen Controller, ohne ein Modell zu lernen.

Model-free learning:

- Das größte Problem, das uns begegnet ist das *temporal credit assignment*.
- Ist die Aktion, die wir gerade nehmen, eine gute, wenn sie weitreichende Auswirkungen haben kann?
- In langlebigen Anwendungen ist es schwierig zu wissen, was das "Ende" ist.
- Stattdessen Einblicke von der Wert-Iteration. Verändere den Wert eines Zustandes aufgrund seiner unmittelbaren Belohnung und dem Wert des Folgezustandes (*temporal-difference metods*).

AHC und TD:
Der *adaptive heuristic critic* Algorithmus ist eine lernende Analogie der Taktik Iteration.



Adaptive heuristic critic architecture

- zwei Komponenten: eine critic (AHC) und eine reinforcement learning (RL) Komponente

- RL kann eine Instanz von jeder der k-armed-Bandit Algorithmen mit einer Modifikation für multiple Zustände sein.

- statt Maximierung der Belohnungen, Maximierung der heuristischen Werte *v*, die von AHC berechnet werden.

- AHC benutzt die realen externen reinforcement-Werte, um zu lernen, Situationen mit ihren Werten unter Aufführung der aktuell in RL instanzierten Taktik, aufzuzeichnen.

- Die Analogie zur Taktik Iteration kann man sehen, wenn wir uns vorstellen, daß die Komponenten altenierend arbeiten (in den meisten Implementierungen arbeiten die Komponenten gleichzeitig).

- Taktik π, durch RL erfüllt, ist fest und AHC lernt die Wertefunktion V_π für diese Taktik.

- Danach AHC fest und RL lernt neuen Taktik π', welche die neue Wertefunktion maximiert.

Bleibt zu klären, wie AHC die Werte einer Taktik lernen kann.

Suttons TD Algorithmus (Sutton 1988) mit der gewichteten Erneuerungsregel

$$V(s) := (1-\alpha)V(s) + \alpha(r + \gamma V(s')).$$

Immer wenn ein Zustand *s* besucht wird, wird sein Wert erneuert, um dichter an $r + \gamma V(s')$ zu sein, mit

- *r*, die instanzierte erhaltene Belohnung

- $V(s')$, der Wert des aktuellen Folgezustandes

- α, der Lernrate

Die motivierende Idee ist, daß $r + \gamma V(s')$ eine Stichprobe für den Wert von $V(s)$ ist. Wenn die Lernrate α richtig eingestellt ist (sie muß langsam abnehmen), garantiert TD ein konvergieren zur optimalen Wertefunktion.

Die TD-Regel, wie gezeigt, ist eine Instanz für eine allgemein Klasse von Algorithmen, genannt $TD(\lambda)$, mit $\lambda = 0$. TD(0) ist nur in der Lage bei Schätzungen von Wertveränderungen einen Schritt weit zu blicken. Die TD(0)-Regel kann algebraisch umgeschrieben werden zu

$$V(s) := V(s) + \alpha(r + \gamma V(s') - V(s));$$

die allgemein $TD(\lambda)$-Regel ist ähnlich,

$$V(s) := V(s) + \alpha(r + \gamma V(s') - V(s))e(s),$$

wird aber eher auf jeden Zustand entsprechend seiner Berechtigung e(s), als nur auf den unmittelbar vorherigen Zustand angewendet. Der Berechtigungspfad ist

$$e(s) = \sum_{k=1}^{t} (\lambda\gamma)^{t-k} I(s = s_k),$$

mit *I* ist eine Indikatorfunktion. Bezeichnet den Grad der Besuche des Zustands *s* in der jüngsten Vergangenheit. Wird benutzt, um alle Zustände, die kürzlich besucht wurden, entsprechend ihrer Berechtigung zu erneuern.

$\lambda = 0$ ist gleichbedeutend mit TD(0).
$\lambda = 1$ ist grob gesagt gleichwertig zum Belohnen aller Zustände in Bezug auf ihren Wert am Ende der Laufzeit.

$TD(\lambda)$ ist rechnerisch aufwendiger, aber konvergiert erheblich schneller für große λ.

Q-learning:

Die Arbeit der zwei Komponenten des AHC in einer einheitlichen Weise kann durch Watkins' Q-learning (Watkins 1989, Watkins & Dayon 1992) geleistet werden. Um Q-learning zu verstehen, brauchen wir einige zusätzliche Notationen:

Sei $Q^*(s,a)$ das erwartete abnehmende reinforcement, mit Aktion *a* in Situation *s*, dann fahre fort mit der der Wahl von Aktionen, um Q^* zu maximieren. Rekursiv geschrieben:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') max_{a'} Q^*(s',a');$$

Watkins hat gezeigt, daß $V^*(s) = max_a Q^*(s,a)$ und somit $\pi^*(s) = arg\ max_a Q^*(s,a)$ die optimale Taktik ist.

Wir können die Q-Werte on-line mit einer Methode wie bei TD schätzen, die aber auch verwendet wird, um die Taktik zu definieren. Eine Aktion kann gewählt werden, indem man die mit maximalem Q-Wert für die aktuelle Situation nimmt.
Die Q-Lernregel ist

$$Q(s,a) := (1-\alpha)Q(s,a) + \alpha(r + \gamma max_{a'} Q(s',a'))$$

Jede Aktion in jedem Zustand unendlich oft in unendlicher Laufzeit ausgeführt und α passend verringert
$\Rightarrow$ Q-Werte konvergieren mit Wahrscheinlichkeit 1 gegen Q^*.

Q-learning kann verallgemeinert werden, um Zustände zu erneuern, die wie in $TD(\lambda)$ früher als einen Schritt vorher auftraten.