Transparanten bij het vak Inleiding Adaptieve Systemen: Reinforcement Leren. M. Wiering

---

Inhouds opgave

- Markov Decision Problems
- Dynamisch Programmeren: herhaling
- Reinforcement Leren: principes
- Temporal difference leren
- Q-leren
- Model gebaseerd leren

Leerdoelen:

1. De theorie begrijpen en de RL algoritmen kunnen opschrijven/gebruiken.
2. Begrijpen waarom exploratie/generalisatie van belang is en manieren kunnen vertellen hoe we dat kunnen aanpakken.
3. Applicaties kunnen bedenken voor RL toepassingen.

Markov besluits problemen

Een Markov decision process (MDP) bestaat uit:

- S: Eindige verzameling toestanden $\{S_1, S_2, \ldots, S_n\}$.
- A: Eindige verzameling acties.
- $P(i, a, j)$: kans om een stapje naar toestand j te maken als actie a wordt geselecteerd in toestand i.
- $R(i, a, j)$ beloning voor het maken van een transitie van toestand i naar toestand j door het executeren van actie a
- γ: discount parameter voor toekomstige beloningen: $(0 \leq \gamma \leq 1)$

Stap = overgang (transitie) van de ene toestand naar de volgende ($\sum_j P(i, a, j) = 1$)

Toestanden kunnen **terminaal** zijn: ketens van stappen die hier terecht komen worden niet verder voortgezet

---

Markov eigenschap

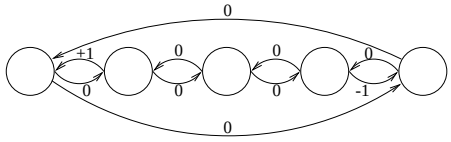
De huidige toestand en actie geven alle mogelijke informatie voor het voorspellen naar welke volgende toestand een stap gemaakt zal worden:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, \ldots, s_1, a_1)$$

Dus, voor het voorspellen van de toekomst doet het er niet toe hoe je in de huidige toestand gekomen bent.

Vergelijk processen in de natuurkunde: waar zou het verleden gerepresenteerd moeten zijn?

Voorbeeld MDP:



Deterministic MDP
Number of states = 5
Number of actions = 2

Passief leren — leert uitkomst van proces zonder besluiten te kunnen nemen welke uitkomst van proces beinvloeden → predictie.

Voorbeeld: in het bovenstaande MDP worden alle acties met 50% gekozen. Wat is de verwachte som der beloningen in de toekomst?

Actief leren — leert **policy** welke acties selecteert zodat uitkomst van proces voor de agent zo goed mogelijk is → controle.

Voorbeeld: in bovenstaande MDP: wat is de optimale actie in elke toestand? Wat is dan de verwachte som der beloningen in de toekomst?

Actie selectie policy

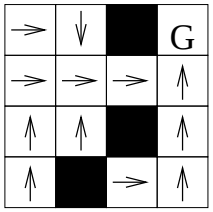
Policy Π selecteert een actie als een functie van de huidige toestand

$$a_t = \Pi(s_t)$$

Doel: Leer de policy Π^* welke de toekomstige verwachte beloningen maximaliseert:

$$\Pi^* = \arg\max_{\Pi} E(\sum_{t=0}^{\infty} \gamma^t R(s_t, \Pi(s_t), s_{t+1}) | s_0 = s)$$

Voorbeeld policy:



Er zijn $|A|^{|S|}$ policies, hoe weten we welke policy het beste is?

Waarde-functie (utiliteiten functie): De waarde van een toestand schat de verwachte toekomstige beloningen:

$$V(s) = E(\sum_{t=0}^{\infty} \gamma^t R(s_t, \Pi(s_t), s_{t+1}) | s_0 = s)$$

De Q-functie schat de waarde voor het selecteren van een actie in een gegeven toestand:

$$Q(s_t, a_t) = \sum_{s_{t+1}} P(s_t, a_t, s_{t+1})(R(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1}))$$

Voorbeeld waarde functie (in deterministische wereld):

5	6		10
6	7	8	9
5	6		8
4		6	7

De V-functie en de Q-functie

We maken gebruik van 2 waarde functies: de V-functie voor het evalueren van toestanden en de Q-functie voor het evalueren van toestand/actie paren.

Als V, de toestand waarde-functie bekend is, dan kunnen we in een toestand alle acties uitproberen, de nieuwe toestand bepalen (met behulp van het model) en die actie selecteren welke leidt tot de grootst verwachte som van toekomstige beloningen.

Als de Q-functie bekend is dan kunnen we in elke toestand direct de actie selecteren met de hoogste Q-waarde (hiervoor is dan ook geen model meer nodig).

Dynamisch programmeren (Bellman 1957)

De optimale Q-functie voldoet aan de Bellman vergelijking:

$$Q^*(i, a) = \sum_j P(i, a, j)(R(i, a, j) + \gamma V^*(j))$$

Hier is $V^*(j) = \max_a Q^*(j, a)$

De optimale policy verkrijgen we dan door:

$$\Pi^*(i) = \arg\max_a Q^*(i, a)$$

Opmerkingen:

- V^* is uniek bepaald
- Π^* is niet altijd uniek bepaald (soms zijn er meerdere optimale policies)

Value Iteration

We kunnen de optimale policy en de Q-functie berekenen door gebruik te maken van een dynamisch programmeer algoritme:

Value iteration:

1. Initialiseer de Q-waarde en V-waarden (b.v. op 0)
2. Maak een "update" voor de Q-waarden:

   $$Q(i, a) := \sum_j P(i, a, j)(R(i, a, j) + \gamma V(j))$$

   Voor terminale toestanden geldt:
   $P(i, a, i) = 1$ en $R(i, a, i) = 0$ voor elke actie.(Of $P(i, a, j) = 0$ voor alle j)

3. Bereken dan de nieuwe waarde functie:

$$V(i) := \max_a Q(i,a) \qquad (1)$$

4. Pas de policy aan zodat in elke toestand de actie met maximale huidige waarde wordt geselecteerd.

$$\Pi(i) := argmax_a Q(i,a)$$

5. Ga naar (2) totdat V niet meer verandert

Evalueren van een policy

Als we de policy vastleggen, kunnen we berekenen wat de exacte waarde van een bepaalde toestand is. Dit correspondeert met passief leren (waarbij de vastgelegde policy de overgangskansen bepaalt).

Omdat we nu een vaste policy Π hebben kunnen we de acties uit de transitie en belonings functies elimineren:

$$P(i,j) = P(i,\Pi(i),j) \text{ en: } R(i,j) = R(i,\Pi(i),j)$$

Nu is $V^\Pi(i)$ voor elke toestand i vastgelegd:

- voor terminale toestanden i:

$$V^\Pi(i) = 0$$

- voor niet-terminale toestanden i:

$$V^\Pi(i) = \sum_j P(i,j)(R(i,j) + \gamma V^\Pi(j))$$

Stelsel van n lineaire vergelijkingen met n onbekenden $V(i)$
$\Longrightarrow$ precies één oplossing voor de V-functie.

Hoe bepaal je de n onbekenden $V(i)$?

Twee methoden:

- **Gauss-eliminatie** (= 'vegen')

$$\begin{aligned} V(1) &= \sum_j P(1,j)(R(1,j) + \gamma V(j)) \\ V(2) &= \sum_j P(2,j)(R(2,j) + \gamma V(j)) \\ .. &= \\ V(n) &= \sum_j P(n,j)(R(n,j) + \gamma V(j)) \end{aligned}$$

En los de onbekenden op.

- **Policy evaluation**:

Start met $V(i) = 0$ voor alle toestanden i

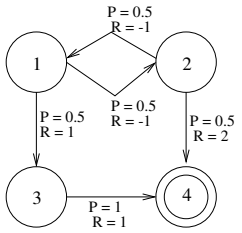
en herhaal

$$V(i) := \sum_j P(i,j)(R(i,j) + \gamma V(j))$$

een groot aantal keer voor alle niet-terminale toestanden i

Opgave:

Gegeven de toestanden 1 t/m 4 waarvan 4 terminaal is:



Bereken de waarden voor alle toestanden.

Dynamisch programmeren als planning tool

Planning: bereken acties om doel te verwezenlijken. Voorbeeld: A* planning. Problemen met niet deterministische omgevingen.

DP: gegeven een toestand: selecteer een actie en volg vervolgens de (optimale) policy

DP voordeel: Tijdens het runnen kunnen acties direct geselecteerd worden (dus zonder kostbare plan operaties)

DP nadeel: de waarde functie moet nauwkeurig zijn.

Problemen voor het gebruik van dynamisch programmeren:

- Een a-priori model van het Markov decision process is nodig (de omgeving moet bekend zijn)

- Als er veel variabelen zijn wordt de toestandsruimte zeer groot (bv. n binaire variabelen $\rightarrow 2^n$ toestanden. DP wordt computationeel dan heel duur.
- Wat als de toestandsruimte continu is?
- Wat als acties/tijd continu zijn?
- Wat als omgeving niet-Markov?

De beste actie leren met RL

Stel je speelt de twee-armige bandiet: er zijn twee acties (L en R), beide kosten een euro.

De linkerarm heeft kans 10% op uitbetalen van 6 euro.

De rechterarm heeft kans 1% op uitbetalen van 100 euro.

Helaas weet je de kansen en opbrengsten niet.

Door herhaaldelijk beide armen uit te proberen, kun je de kans op winst en het winstbedrag leren (gewoon door het gemiddelde te bepalen).

Als de kansen en de bedragen nauwkeurig bekend zijn is het simpel om optimaal te spelen.

2-armige bandiet en exploratie

Stel je speelt het spel en krijgt de volgende resultaten:

(1, L, -1)
(2, R, -1)
(3, L, +5)

De verwachtings waarden kunnen we opschrijven als een quadruple:

(Actie A, kans P, winstbedrag R, gem. waarde V)

Voor bovenstaande ervaringen worden deze:
(L, 0.5, 5, 2.0) en (R, 0, ?, -1).

Als we nu verder spelen, moeten we dan direct L kiezen, of toch R blijven uitproberen?

Dit wordt het exploratie/exploitatie dilemma genoemd

Vergelijk: kiezen we voor informatie voor meer toekomstige beloning of voor directe beloning?

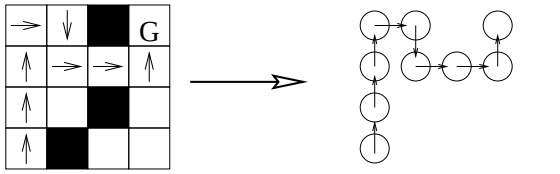
Reinforcement leren

Geen a-priori gegeven model (transitie kansen, beloningen) is nodig.

Reinforcement leren leert een subjectieve "view" op de wereld door de interactie met de wereld.

Een policy wordt getest hetgeen ervaringen oplevert waarvan geleerd kan worden om een nieuwe policy te berekenen.

Exploratie van de toestands ruimte is nodig.



Epoch = Sequentie
Ervaringen (stapjes)

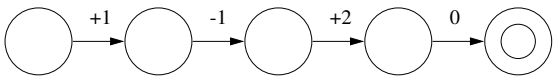
Subjectieve kijk van
de agent op de wereld

Principes van reinforcement leren (RL)

Om de Q-functie te leren, herhalen RL algoritmen voortdurend het volgende:

1. Selecteer actie a_t gegeven de toestand s_t
2. Vergaar de beloning r_t en observeer de opvolgende toestand s_{t+1}
3. Maak een "update" van de Q-functie door gebruik te maken van de laatse ervaring: (s_t, a_t, r_t, s_{t+1})

Epoch = keten opeenvolgende toestanden eindigend in terminale toestand (of na vast aantal stapjes).



Uit de epochs willen we de waarde functie en de optimale strategie leren

---

Vier RL methoden:

- **Monte Carlo sampling** (Naïef updaten)

- **Temporal difference (TD) leren**
- **Q-leren**
- **Model-gebaseerd dynamisch programmeren**

De eerste drie methodes gebruiken geen transitie model en worden daarom ook vaak **direct RL** of **model-free** RL genoemd.

De vierde schat eerst een transitie model en berekent de waarde functie aan de hand van dynamisch programmeer achtige methoden. Daarom wordt deze methode ook wel **indirect RL** of **model-based** RL genoemd.

Monte Carlo Sampling

- Bepaal voor elke toestand s in een epoch k de
  reward-to-go: a_k = de som van alle beloningen in die epoch vanaf het eerste moment dat die toestand bezocht is tot de epoch afgelopen is
- Schatting voor utiliteit van een toestand:

  neem het gemiddelde van alle rewards-to-go van alle keren dat die toestand in een epoch voorkomt

$$V(s) = \frac{\sum_{i=1}^{k} a_i(s)|\text{ s bezocht in epoch i}}{\text{aantal epochs dat s bezocht werd}}$$

Bezwaar: deze methode convergeert zeer langzaam (update variantie is heel groot)

Temporal difference leren:

In plaats van direct de hele toestand keten te gebruiken voor een update, kunnen we ook alleen de opvolgende toestand gebruiken.

Doe voor elke stap van i naar j in een epoch:

- als j terminaal: $V(i) := V(i) + \alpha(R(i,j) - V(i))$
- als j niet terminaal:

$$V(i) := V(i) + \alpha(R(i,j) + \gamma V(j) - V(i))$$

Hier is α een klein positief getal, de **learning rate**

Idee: geef elke keer $V(i)$ een duwtje in de gewenste richting

Bij vaste α komt dit snel in de buurt van de echte utiliteit, maar convergeert daarna niet verder

Als α steeds kleiner wordt naarmate toestand i vaker bezocht is, convergeert het wel

---

Voorbeeld:

Als $P(i,j) = \frac{1}{3}$ en $P(i,k) = \frac{2}{3}$, en de overgang $i \to j$ komt 10 keer voor en de overgang $i \to k$ komt 20 keer voor, dan:

$$10\times : V(i) := V(i) + \alpha(R(i,j) + \gamma V(j) - V(i))$$

$$20\times : V(i) := V(i) + \alpha(R(i,k) + \gamma V(k) - V(i))$$

$$\approx\ V(i) := V(i) + \alpha(10R(i,j) + 10\gamma V(j) +$$

$$20R(i,k) + 20\gamma V(k) - 30V(i)),$$

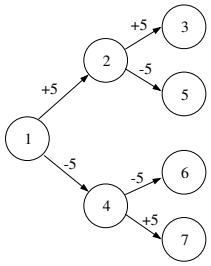
$$\Leftrightarrow$$

$$30\alpha V(i) = \alpha(10R(i,j) + 10\gamma V(j) + 20R(i,k) + 20\gamma V(k))$$

precies een stap in de gewenste richting $\Longrightarrow$

$$V(i) := \frac{1}{3}(R(i,j) + \gamma V(j)) + \frac{2}{3}(R(i,k) + \gamma V(k))$$

Opgave:



Stel elke overgang 50% kans.

Stel vervolgens dat de agent de volgende epochs (sequenties van toestanden) meemaakt:
$\{1,2,3\}$
$\{1,4,7\}$
$\{1,2,5\}$

$\{1, 2, 3\}$
Welke updates van de V-functie zal de agent maken met Monte Carlo sampling?
Welke met TD-leren?

Q-leren

Q-learning (Watkins, 1989) verandert een enkele Q-waarde gegeven de ervaring (s_t, a_t, r_t, s_{t+1}):

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$$

Hierbij is $V(s) = \max_a Q(s, a)$.

Als Q-leren gebruikt wordt, convergeert de Q-functie naar de optimale Q-functie als alle toestand/actie paren oneindig vaak bezocht worden (en de leersnelheid afneemt).

Q-leren is meest gebruikte RL methode.

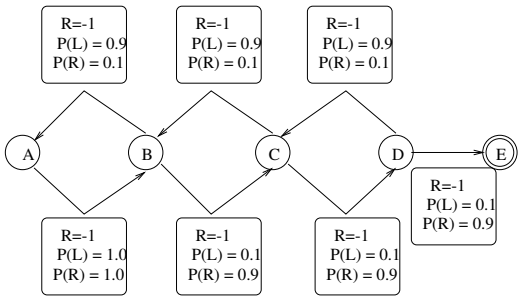
Voordeel van Q-leren: simpel te implementeren.

Nadeel van Q-leren: kan lang duren voordat beloning aan eind van keten terug gepropageerd is naar een toestand.

Voorbeeld Q-leren

We hebben de volgende toestandsgraaf met overgangen voor de acties Links (L) en rechts (R).

Er zijn 5 toestanden: A,B,C,D,E. E is een terminale toestand.



Stel de volgende overgangen worden gemaakt:
(A, L, B); (B, R, C); (C, R, D); (D,R,E)
(C,L,D); (D, L, C); (C,R,D); (D,R,E)
(B,L,A); (A,R,B); (B,L,C); (C,R,D); (D,L,E)

Vraag: Wat zijn de resulterende Q-waarden als Q-leren gebruikt wordt ($\alpha = 0.5$)?

Model-gebaseerd RL

Model-gebaseerd RL schat eerst de transitie en de belonings functies:

Maak schatting van $P(i, a, j)$:

$$\hat{P}(i, a, j) := \frac{\#\text{ overgangen van } i \to j \text{ waarbij } a \text{ gekozen}}{\#\text{ keren actie } a \text{ in toestand } i \text{ gekozen}}$$

Doe hetzelfde voor de beloning:

$$\hat{R}(i, a, j) := \frac{\text{som beloningen op overgang van } i \to j \text{ na kiezen } a}{\#\text{ transities van } i \text{ naar } j \text{ waarbij } a \text{ gekozen werd}}$$

Herhaal de update

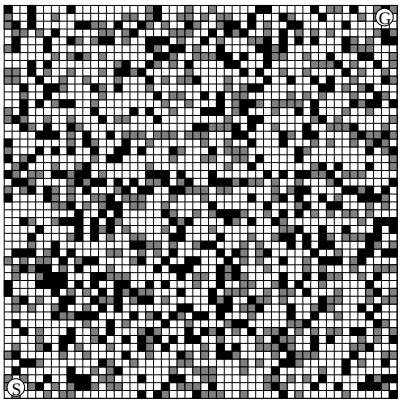
$$Q(i, a) := \sum_j \hat{P}(i, a, j)(\hat{R}(i, a, j) + \gamma V(j))$$

een aantal keer voor alle niet-terminale toestanden

Vaak is het onnodig om alle Q-waarden te updaten:

Slechts een subset van de Q-waarden zal significant veranderen door de laatste ervaring. Snellere update-methoden houden hier rekening mee
(bv. **Prioritized sweeping**)

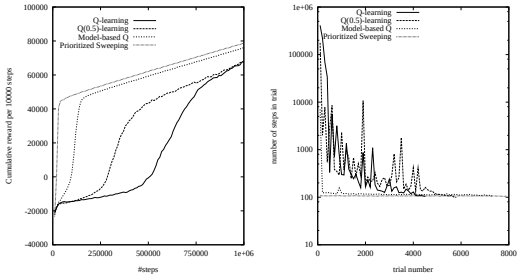
Experimentele vergelijking



50×50 maze
Reward goal = + 100 ; Reward blocked = -2 ;
Reward penalty = -10 ; otherwise -1;
10% noise in action execution
Max-random exploration (30% $\to$ 0% noise)

Indirect vs. Direct RL

Voordelen direct RL:



- Minder geheugen ruimte nodig (transitie functie kan groot zijn)
- Werkt ook met niet discrete representaties (bv. neurale netwerken)
- Kan beter werken als Markov eigenschap niet geldt

Nadelen direct RL:

- Veel informatie wordt weggegooid
- Agent heeft geen mogelijkheid tot introspectie: bv. welke actie heb ik nog weinig uitgeprobeerd (voor exploratie)
- Leren kan veel langer duren
- Geleerde waarde functie meestal veel minder nauwkeurig