Reinforcement Learning

IA 2020/2021

Conținut

Învățare cu întărire (Reinforcement learning)

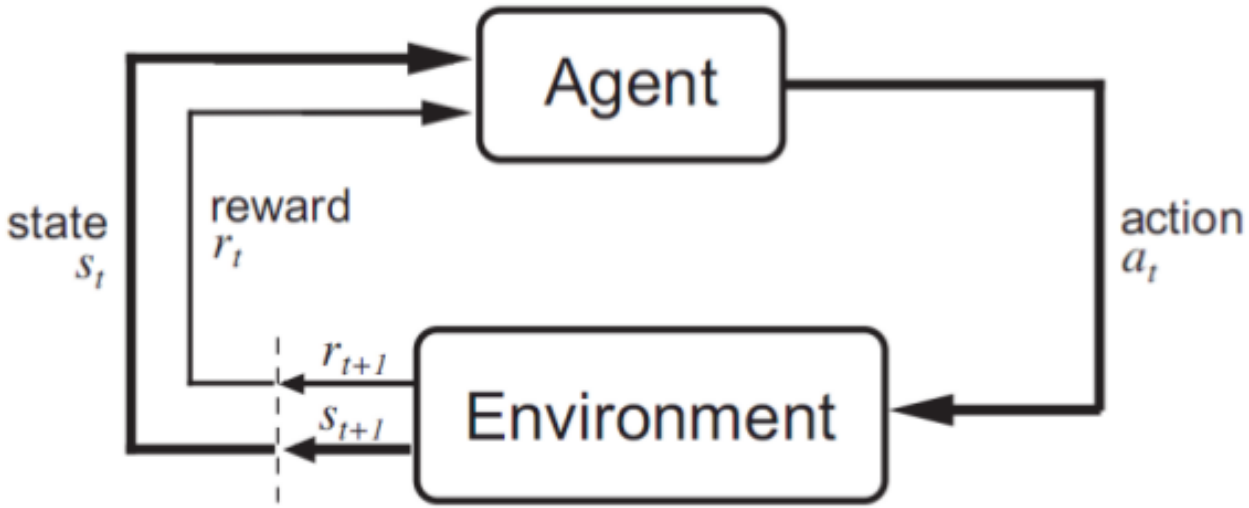
Agentul trebuie să învețe un comportament, fără a avea un instructor

- Agentul are o sarcină de îndeplinit
- Efectuează o serie de acțiuni
- Primește feedback (reacția mediului): cât de bine a acționat pentru a-și îndeplini sarcina
- Agentul urmărește îndeplinirea sarcinii în mod repetat

Această modalitatea de învățare se numește învățare cu întărire: agentul primește o recompensă pozitivă dacă îndeplinește bine sarcina, respectiv o recompensă negativă dacă nu îndeplinește bine sarcina

Modelul de interacțiune

- Agentul efectuează acțiuni
- Mediul îi prezintă agentului situații numite stări și acordă recompense

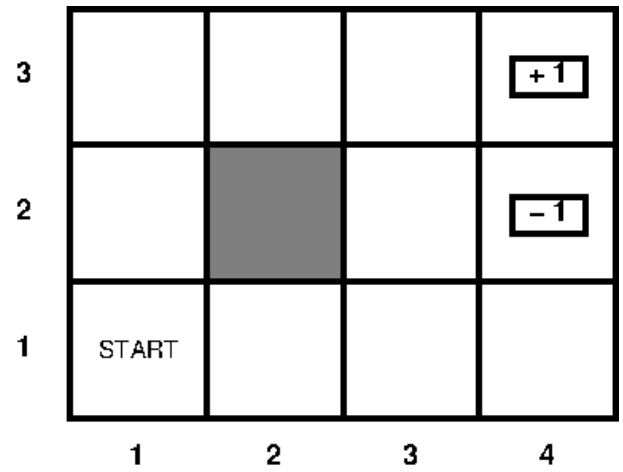


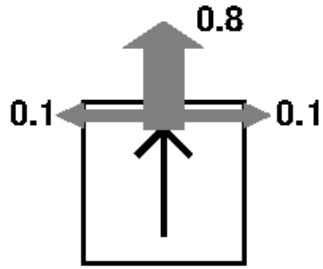
Învățare cu întărire

- Scopul: de a determina agentul să acționeze a.î. să-și maximizeze recompensele
- Agentul trebuie să identifice secvența de acțiuni ce conduce la îndeplinirea sarcinii
  - Date de antrenare: (S, A, R) Stare, Acțiune, Recompensă

Conținut

Decizii secvențiale





- Mediu determinist:
  - (sus, sus, dreapta, dreapta, dreapta)
- Mediu stochastic
  - Model de tranziții $P(s'|s,a)$: probabilitatea de a ajunge din starea s în starea s' efectuând acțiunea a
  - Acțiunea obține efectul dorit cu probabilitatea 0.8
  - Agentul primește o recompensă: -0.04 pentru stările nonterminale; +/-1 pentru stările terminale

Presupunerea Markov

- Starea curentă s_t depinde de un istoric finit al stărilor anterioare
- Proces Markov de ordin întâi: starea curentă s_t depinde doar de starea anterioară s_{t-1}

$$P(s_t|s_{t-1}, \ldots, s_0) = P(s_t|s_{t-1})$$

Proces de decizie Markov (Markov Decision Process)

Proces de decizie Markov: o problemă de decizie secvențială pentru un mediu stochastic cu un model de tranziție Markov și recompense aditive

- Stări $s \in S$ (starea inițială s_0), acțiuni $a \in A$
- Modelul de tranziții $P(s'|s, a)$
- Funcția de recompensă $R(s)$

Cum arată o soluție? Trebuie să specifice ce trebuie să facă agentul în fiecare stare (politică π)

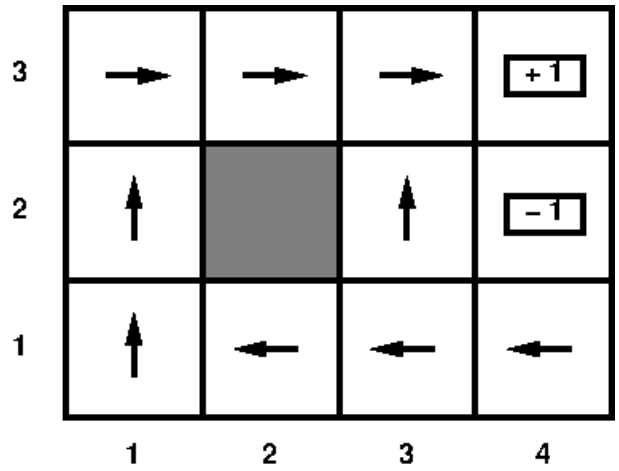
Markov Decision Process (MDP)

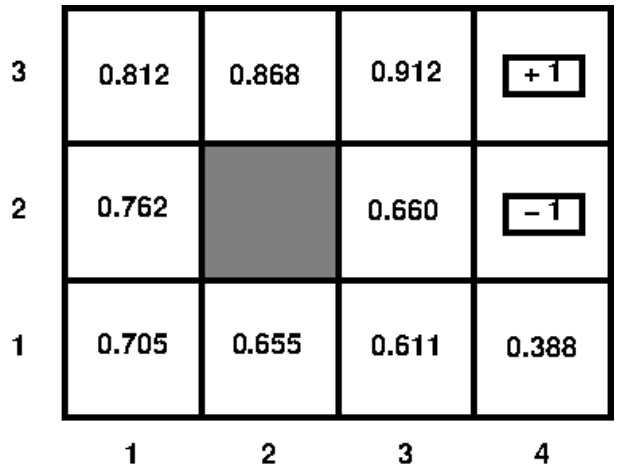
- În problemele de căutare, scopul este de a identifica o *secvență* optimă
- În MDP, scopul este de a identifica o *politică* optimă π^* (tactică/strategie)
  - $\pi : S \to A$; $\pi(s)$ este acțiunea recomandată în starea s

- Utilitate: suma recompenselor pentru o secvență de stări
  - Recompensa este câștigul imediat, pe termen scurt
  - Utilitatea este câștigul total, pe termen lung

- Mediu stochastic: putem avea o secvență diferită de stări când executăm aceeași politică din starea inițială
  Calitatea unei politici: utilitatea așteptată a secvențelor posibile de stări;
  Politica optimă π^* maximizează utilitatea așteptată;

Exemplu: politica optimă și valorile stărilor:





Utilități

- Orizont finit
  - $U_h([s_0, s_1, \ldots, s_{N+k}]) = U_h([s_0, s_1, \ldots, s_N]), \forall k > 0$
  - După momentul N, nu mai contează nimic
  - Politica optimă nu este staționară: acțiunea optimală pentru o anumită stare se poate schimba în timp

Utilități

- Orizont infinit
  - Nu există un termen limită fix
  - Politica optimă este staționară
  - a. Recompense aditive

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$$

  - b. Recompense actualizate (*discounted*)

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$$

$\gamma \in [0, 1]$ factorul de actualizare (*discount factor*) indică faptul că recompensele viitoare conteaza mai puțin decât cele imediate

Orizont infinit - evaluare

Trebuie să ne asigurăm că utilitatea unei secvențe posibil infinite este finită.

- **Abordarea 1**. Dacă recompensele sunt mărginite și $\gamma < 1$ atunci:

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = R_{max}/(1-\gamma)$$

- **Abordarea 2**. Dacă mediul conține stări terminale și se garantează faptul că agentul va atinge una din ele (avem o politică adecvată, *proper policy*), putem utiliza $\gamma = 1$
- **Abordarea 3**. Compararea recompenselor medii (pentru fiecare pas)

Utilitatea unei stări

- Fiecare politică generează secvențe multiple de stări, datorită incertitudinii tranzițiilor $P(s'|s,a)$
- Utilitatea așteptată obținută prin execuția politicii π din starea s:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

S_t o variabilă aleatoare: starea în care ajunge agentul la timpul t executând politica π, $S_0 = s$.
(= valoarea așteptată a sumei tuturor recompenselor actualizate obținute pentru toate secvențele posibile de stări)

Evaluarea unei politici

- Politica optimă

$$\pi_s^* = argmax_\pi U^\pi(s)$$

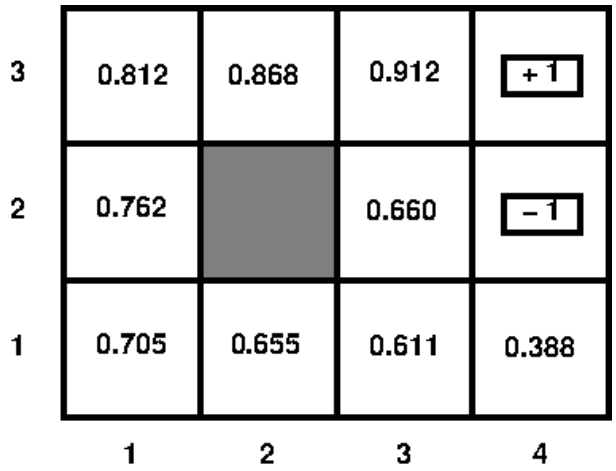
- $U^{\pi^*}(s)$ utilitatea adevărată a unei stări: valoarea așteptată a sumei recompenselor actualizate dacă agentul execută o politică optimă; $U(s)$

- Principiul Maximum Expected Utility: alege acțiunea care maximizează utilitatea așteptată a stării ulterioare

$$\pi^*(s) = argmax_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

Exemplu

Fie $\gamma = 1$ și $R(s) = -0.04$.



Aproape de starea finală utilitățile sunt mai mari pentru că este nevoie de mai puțini pași cu recompensă negativă pentru atingerea stării respective.

Ecuația Bellman

$$U(s) = R(s) + \gamma max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

Ecuația Bellman (1957): utilitatea unei stări este recompensa imediată pentru acea stare plus utilitatea așteptată maximă a stării următoare.

Exemplu

Utilitatea stării (1,1):

$$\begin{aligned} U(1,1) = -0.04 + \gamma max[&0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), \quad (Up) \\ &0.9U(1,1) + 0.1U(1,2), \quad (Left) \\ &0.9U(1,1) + 0.1U(2,1), \quad (Down) \\ &0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)] \quad (Right) \end{aligned}$$

Cea mai bună acțiune: *Up*.

Rezolvarea unui proces de decizie Markov

- n stări posibile
- n ecuații Bellman, una pentru fiecare stare
- n ecuații cu n necunoscute: $U(s)$
- Nu se poate rezolva ca sistem de ecuații liniare din cauza funcției *max*

I. Iterarea valorilor (*Value iteration*)

Calculează utilitatea fiecărei stări și identifică acțiunea optimă în fiecare stare

Algoritm pentru calcularea politicii optime:

- Inițializează utilitățile cu valori arbitrare
- Actualizează utilitatea fiecărei stări din utilitățile vecinilor

$$U_{i+1}(s) = R(s) + \gamma max_a \sum_{s'} P(s'|s,a) U_i(s')$$

- Repetă pentru fiecare s simultan, până la atingerea unui echilibru

Iterarea valorilor: pseudocod

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

II. Iterarea politicilor

- Dacă o acțiune este în mod evident mai bună decât toate celelalte, nu avem nevoie de valorile exacte ale utilităților
- Algoritmul alternează următorii pași:
  - 1. Evaluarea politicii: dată o politică π_i, calculează $U_i = U^{\pi_i}$ utilitățile stărilor pe baza politicii π_i
  - 2. Îmbunătățirea politicii: calculează o nouă politică π_{i+1}, pe baza utilităților U_i

1. Evaluarea politicii

Acțiunea în fiecare stare e fixată de politică; la iterația i, politica π_i specifică acțiunea $\pi_i(s)$ în starea s.

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

(Ecuații Bellman simplificate)

- Sistem de n ecuații liniare cu n necunoscute
- Se poate rezolva *exact* în $O(n^3)$ sau în mod *aproximativ*
- aplicam *Value iteration*

$$U_{i+1}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) U_i(s')$$

Exemplu: $\pi_i(1,1) = Up$, $\pi_i(1,2) = Up$
Ecuațiile Bellman simplificate:
$U_i(1,1) = -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1)$
$U_i(1,2) = -0.04 + 0.8U_i(1,3) + 0.2U_i(1,2)$

2. Îmbunătățirea politicii

- Valorile $U(s)$ se cunosc
- Calculează pentru fiecare s, acțiunea optimă

$$a_i^*(s) = max_a \sum_{s'} P(s'|s,a)U(s')$$

- Dacă $a_i^*(s) \neq \pi_i(s)$, actualizează politica: $\pi_{i+1}(s) \leftarrow a_i^*(s)$
- Se pot actualiza doar părțile "promițătoare" ale spațiului de căutare

Iterarea politicilor: pseudocod

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Conținut

Reinforcement learning

- Proces de decizie Markov
  - Mulțimea de stări S, mulțimea de acțiuni A
  - Modelul de tranziții $P(s'|s, a)$ este cunoscut
  - Funcția de recompensă $R(s)$ este cunoscută
  - Calculează o politică optimă
- Învățare cu întărire
  - Se bazează pe procese de decizie Markov, dar:
  - Modelul de tranziții este necunoscut
  - Funcția de recompensă este necunoscută
  - Învață o politică optimă

Tipuri de învățare cu întărire

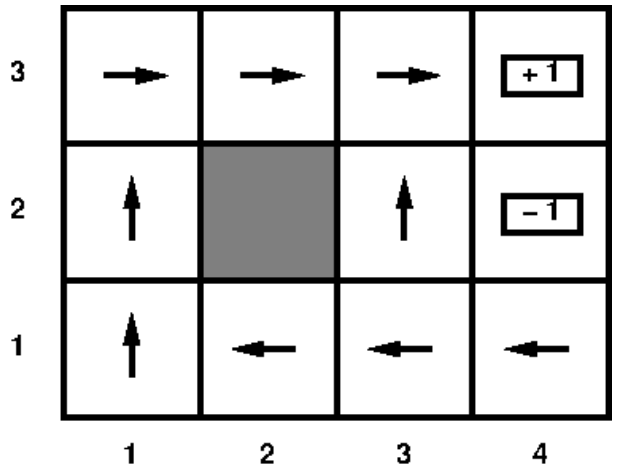
- Pasivă/activă
  - Pasivă: agentul execută o politică fixă și o evaluează
  - Activă: agentul își actualizează politica pe măsură ce învață
- Bazată pe model/fără model
  - Bazată pe model: învață modelul de tranziții și recompense și îl folosește pentru a descoperi politica optimă
  - Fără model: descoperă politica optimă fără a învăța modelul

Conținut

Învățarea pasivă

- Învățarea pasivă este o modalitate de explorare a mediului
- Politica este fixă (în starea s execută întotdeauna acțiunea $\pi(s)$)
- Scopul: învață cât de bună este politica π - învață utilitatea $U^{\pi}(s)$ a fiecărei stări
- Abordare similară cu evaluarea politicii din cadrul algoritmului *Iterarea politicilor*; diferența: nu cunoaște modelul de tranziție $P(s'|s, a)$ și nici $R(s)$

Învățarea pasivă



- Agentul execută o serie de încercări (*trials*)
  $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow$
  $(2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$
  $(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow$
  $(3,2)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$
  $(1,1)_{-.04} \rightsquigarrow (2,1)_{-.04} \rightsquigarrow (3,1)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (4,2)_{-1}$
- Politica este aceeași, dar mediul este nedeterminist
- Scopul este să învețe utilitatea așteptată $U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$

1. Estimarea directă a utilității

- Utilitatea unei stări este recompensa totală așteptată de la acea stare înainte (**reward-to-go**)
- De exemplu, prima încercare produce:
  - în starea (1,1) recompensa totală 0.72 (1 - .04 x 7)
  - în starea (1,2) două recompense totale 0.76 și 0.84
  - în starea (1,3) două recompense totale 0.80 și 0.88
- Utilitatea estimată: media valorilor eșantionate
  - U(1,1) = 0.72, U(1,2) = 0.80, U(1,3) = 0.84 etc.

Estimarea directă a utilității

Nu ține cont de faptul că utilitatea unei stări depinde de utilitățile stărilor succesoare (constrângerile date de ecuațiile Bellman)

- Căutarea într-un spațiu mult mai mare decât cel necesar
- Convergența este foarte lentă

2. Programarea dinamică adaptivă

- Utilizăm programare dinamică pentru învățarea modelului de tranziții și rezolvarea procesului de decizie Markov
- Se folosesc ecuațiile Bellman

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s')$$

- Trebuie estimate $P(s'|s, \pi(s))$ și $R(s)$ din încercări
- Probabilitățile și recompensele învățate se introduc în ecuațiile Bellman
- Se rezolvă sistemul de ecuații liniare cu necunoscutele $U^{\pi}(s)$

Programarea dinamică adaptivă

- Procesul de învățare a modelului: utilizăm un tabel de probabilități (cât de des apare rezultatul unei acțiuni și estimăm probabilitatea de tranziție)
- Exemplu: acțiunea *Right* este executată de 3 ori în starea (1,3) și în 2 cazuri starea rezultantă este (2,3)
  $\implies P((2,3)|(1,3), Right) = 2/3$

Programarea dinamică adaptivă

- PDA este ineficientă dacă spațiul stărilor este mare
  - Sistem de ecuații liniare de ordin n
  - Jocul de table: 10^{50} ecuații cu 10^{50} necunoscute

3. Învățarea diferențelor temporale (*Temporal Differences*)

- Combină avantajele celor două abordări anterioare (Estimarea directă a utilității și Programarea dinamică adaptivă)
  - Actualizează doar stările direct afectate
  - Satisface aproximativ ecuațiile Bellman

- Utilizează tranzițiile observate pentru a ajusta utilitățile
  Exemplu:
  - După prima încercare, estimările $U^{\pi}(1,3) = 0.84$, $U^{\pi}(2,3) = 0.92$
  - Fie tranziția $(1,3) \rightarrow (2,3)$ în a doua încercare
  - Între cele două stări, constrângerea dată de ecuația Bellman impune ca $U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3) = 0.88$ (cu $\gamma = 1$)
  - Estimarea $U^{\pi}(1,3) = 0.84$ este mai mică și trebuie mărită

Învățarea diferențelor temporale

- Ecuația diferențelor temporale utilizează diferența utilităților între stări succesive:

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

  α rata de învățare
- Metoda aplică o serie de corecții pentru a converge
- Rata de învățare α determină viteza de convergență la utilitatea reală
- Metoda nu are nevoie de un model de tranziții pentru a realiza actualizările

Învățarea diferențelor temporale

- Actualizarea implică doar succesorul s', pe când condițiile de echilibru implică toate stările posibile următoare
- Cum tranzițiile rare apar rar, valoarea medie a lui $U^{\pi}(s)$ va converge la valoarea corectă
- Dacă α este o funcție care scade pe măsură ce numărul de vizitări ale unei stări crește, atunci $U^{\pi}(s)$ converge la valoarea corectă
  - funcția $\alpha(n) = 1/n$ sau $\alpha(n) = 1/(1+n) \in (0, 1]$

Diferențe temporale: pseudocod

function PASSIVE-TD-AGENT(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r'

persistent: π, a fixed policy

U, a table of utilities, initially empty

N_s, a table of frequencies for states, initially zero

s, a, r, the previous state, action, and reward, initially null

if s' is new **then** $U[s'] \leftarrow r'$

if s is not null **then**

increment $N_s[s]$

$U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma\, U[s'] - U[s])$

if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$

return a

Programare dinamică adaptivă vs. Diferențe temporale

- DT nu are nevoie de model, PDA este bazată pe model
- DT actualizează doar succesorul observat și nu toți succesorii
- Diferențele scad pe măsură ce numărul de încercări crește
- DT converge mai lent, dar execută calcule mai simple
- DT poate fi văzut ca o aproximare a PDA

Conținut

Invățarea activă

- Agentul pasiv are o politică fixă/agentul activ trebuie sa decidă acțiunile
- Agentul pasiv învață utilitățile stărilor și probabilitățile tranzițiilor și alege acțiunile optime în mod greedy
- Agentul activ își actualizează politica pe măsură ce învață
  - Scopul este să învețe politica optimă pentru maximizarea utilității
  - Însă funcția utilitate nu este cunoscută decât aproximativ
- Dilema exploatare-explorare a agentului
  - să își maximizeze utilitatea pe baza cunoștințelor curente, sau
  - să încerce să își îmbunătățească aceste cunoștințe

Exploatarea și explorarea

Este necesar un compromis între

- Exploatare
  - Agentul oprește învățarea și execută acțiunile date de politică
  - Are ca efect maximizarea recompenselor folosind estimările curente
- Explorare
  - Agentul învață încercând acțiuni noi
  - Poate conduce la maximizarea recompenselor pe termen lung

Algoritmul Q-Learning

- Algoritmul Q-Learning (Watkins, 1989) învață o funcție acțiune-valoare $Q(s, a)$ (Q *quality*)
  - utilitățile $U(s) = max_a Q(s, a)$
  - un agent TD care învață o funcție Q nu are nevoie de un model de forma $P(s'|s, a)$ pentru învățare sau selecția acțiunii (**model-free**)
- Ecuațiile adevărate la echilibru când valorile Q sunt corecte

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) max_{a'} Q(s', a')$$

Acestea pot fi utilizate într-un proces iterativ care calculează valorile Q exacte.

- Ecuația de actualizare pentru TD Q-Learning:

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

(de fiecare dată când executând acțiunea a în starea s rezultă s')

Algoritmul Q-Learning: pseudocod

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return a

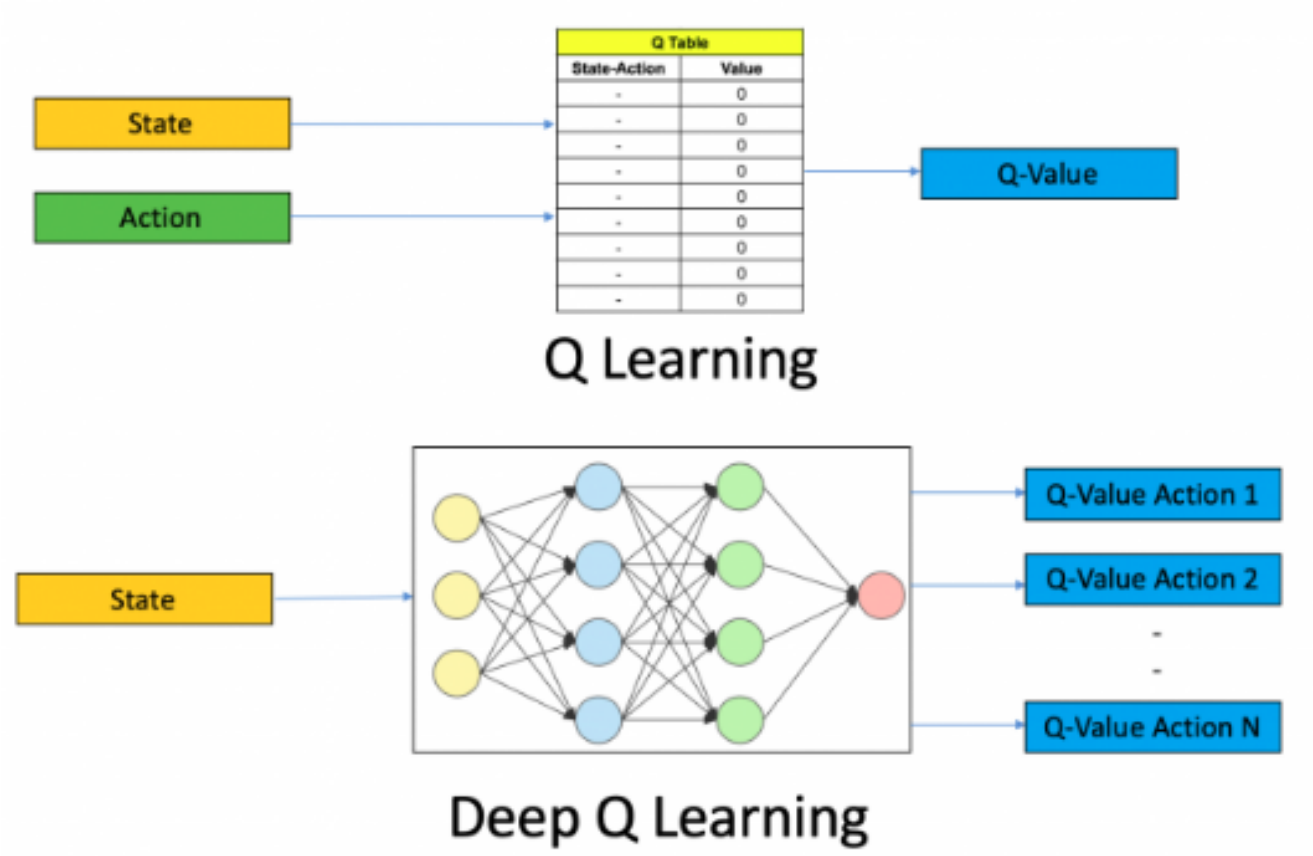
Algoritmul Q-Learning

- Coeficientul de învățare α determină viteza de actualizare a estimărilor
  - de obicei, $\alpha \in (0, 1)$
- Q-Learning este mai lent decât PDA

Deep Q-learning

Utilizează o rețea neurală profundă pentru a aproxima valorile Q

- starea inițială este introdusă în rețeaua neuronală și valoarea Q a tuturor acțiunilor posibile este generată ca ieșire



Deep Q-learning

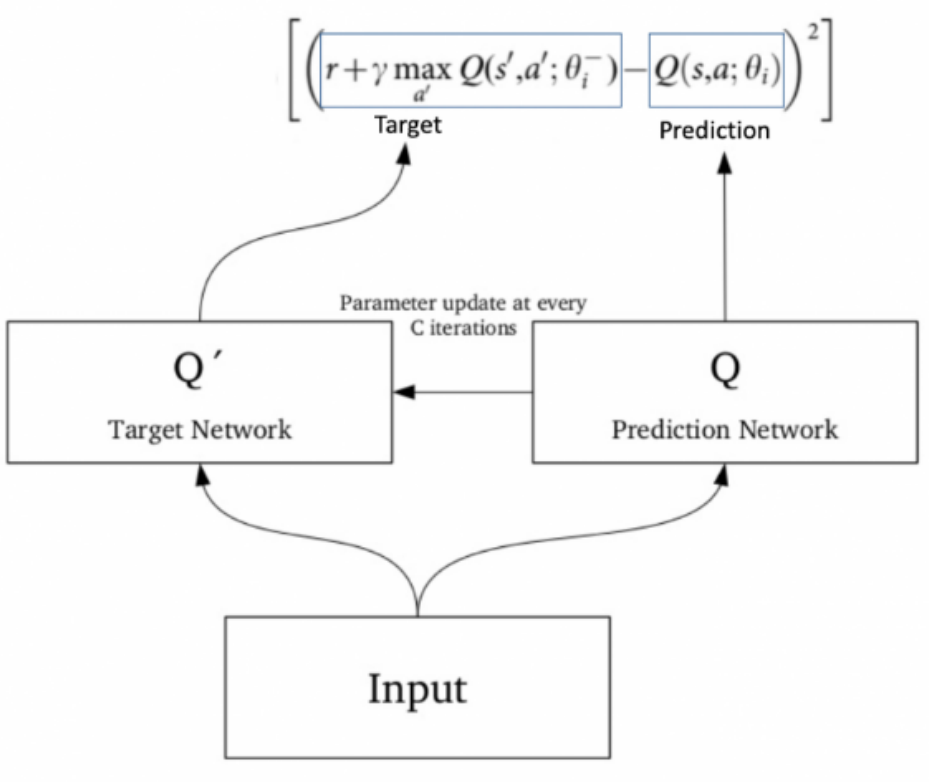
- Funcția loss: eroarea medie patratică (MSE) a valorii Q prezise și a valorii țintă Q^*; nu cunoaștem valoarea țintă
- Ecuația de actualizare a valorii Q derivată din ecuația Bellman:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Valoarea țintă: $R_{t+1} + \gamma max_a Q(S_{t+1}, a)$

Deep Q-learning

- valoarea țintă se modifică la fiecare iterație
- soluție: o rețea separată pentru a estima valoarea țintă
- la fiecare C iterații, parametrii din rețeaua de predicție sunt copiați în rețeaua țintă



Deep Reinforcement Learning

- AlphaGo - Google DeepMind (2015)
- Programul a învățat să joace jocurile Atari 2600 urmărind direct doar afișajul și scorul

- Martie 2016: a câștigat cu 4-1 împotriva lui Lee Sedol, jucător de go profesionist cu 9 dan, premiu: 1 000 000 $
- Mai 2017: a câștigat împotriva lui Ke Jie, cel mai bun jucător de go din lume
- Octombrie 2017: AlphaGo Zero, a învățat să joace fără informații din jocuri ale oamenilor (doar pe baza regulilor jocului) și a învins AlphaGo cu 100-0
- Decembrie 2017: AlphaZero a învins AlphaGo Zero cu 60-40 și a ajuns după doar 8 ore de antrenare la un nivel superior tuturor programelor de go și șah existente

Conținut

Concluzii

- Învățarea cu întărire este necesară pentru agenții care evoluează în medii necunoscute
- Învățarea pasivă presupune evaluarea unei politici date
- Învățarea activă presupune învățarea unei politici optime