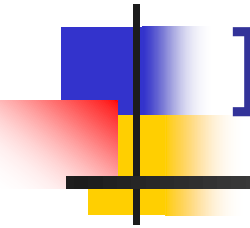


# Inteligență artificială

## 13. Învățarea cu întărire

**Florin Leon**

Universitatea Tehnică „Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare

http://florinleon.byethost24.com/curs_ia.html

# Învățarea cu întărire

1. Introducere
2. Procese de decizie Markov
3. Învățarea pasivă
4. Învățarea activă
5. Optimizări
6. Concluzii

# Învățarea cu întărire

1. Introducere
2. Procese de decizie Markov
3. Învățarea pasivă
4. Învățarea activă
5. Optimizări
6. Concluzii

# Tipuri de învățare

- Învățarea supervizată
  - Date de antrenare: (**A**, *c*) – atribute, clasă
  - Scopul este predicția lui *c* și minimizarea erorii
  - Clasificare, regresie
- Învățarea nesupervizată
  - Date de antrenare: **X** – numai atributele
  - Scopul este determinarea unor grupuri de puncte similare în spațiul multidimensional cu |**X**| dimensiuni
  - Grupare (clusterizare, partiționare)

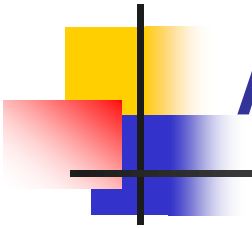
# Învățarea cu întărire

- engl. "reinforcement learning"
- Agentul trebuie să învețe un comportament fără a avea un instructor
  - Agentul are o sarcină de îndeplinit
  - Efectuează niște acțiuni în mediul de execuție
  - Ulterior, primește un *feedback* (reacția mediului) privind cât de bine a acționat în vederea îndeplinirii sarcinii
  - Agentul urmărește îndeplinirea aceleiași sarcini în mod repetat
- Un agent este o entitate autonomă (software sau hardware) care acționează fără intervenție umană și este parte integrantă din mediul său de execuție

# Învățarea cu întărire

- Agentul primește o recompensă (întărire pozitivă) dacă îndeplinește bine sarcina
- Agentul primește o pedeapsă (întărire negativă) dacă îndeplinește rău sarcina

- *Experiența este un profesor dur pentru că întâi îți dă testul și abia apoi îți predă lecția.* (Vernon Law)



# Aplicații

- Medii complexe pentru care nu se pot stabili probabilitățile și recompensele
  - Jocuri: spațiu uriaș de stări
    - Se poate spune doar la sfârșit dacă recompensa este pozitivă (s-a câștigat) sau negativă (s-a pierdut)
  - Mediul fizic: roboți, elicoptere
    - Recompense negative doar pentru deviere de la curs, clătinare, prăbușire

# Aplicații

- Dame – Samuel (1959, 1967)
- TD-Gammon (joc de table) – Tesauro (1992)
  - Funcția de evaluare: rețea neuronală cu 1 strat ascuns cu 40 de neuroni
  - După 300 000 de jocuri cu el însuși, a atins performanțe de campion mondial din top 3
- Alte jocuri: solitaire, șah

# Aplicații

- Controlul pendulului inversat
  - Algoritmul Boxes – Michie & Chambers (1968)
  - Pendulul inversat *triplu*
- În general, probleme pentru care este dificilă determinarea apriori a unor reguli de rezolvare
  - Controlul roboților sau al altor echipamente
  - Stabilirea prețurilor, livrarea produselor
  - Listă de aplicații de succes: http://umichrl.pbworks.com/w/page/7597597/Successes-of-Reinforcement-Learning

# Aplicații

- Algoritmul Pegasus (Ng & Jordan, 2000): zbor autonom al unui elicopter


# Învățare cu întărire profundă

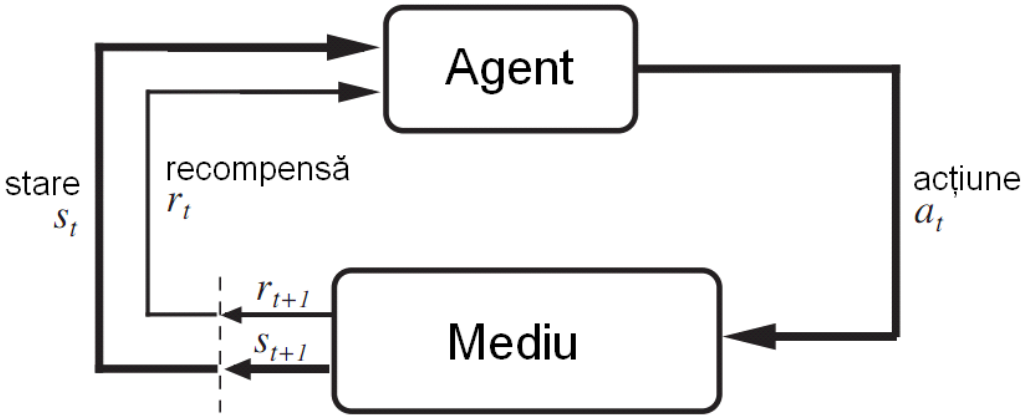
- engl. "Deep Reinforcement Learning"
- Google DeepMind (2015)
- Programul a învățat să joace jocurile Atari 2600 urmărind direct doar afișajul și scorul

# AlphaGo (Google DeepMind)

- 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 Lee 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

# Modelul de interacțiune din învățarea cu întărire

- Agentul
  - Efectuează acțiuni
- Mediul
  - Acordă recompense
  - Îi prezintă agentului situații numite stări



# Învățarea cu întărire

- Scopul este de a determina agentul să acționeze astfel încât să-și maximizeze recompensele totale
- Agentul trebuie să-și dea seama ce secvență de acțiuni conduce la îndeplinirea sarcinii
- Datele de antrenare sunt triplete de tipul (*S*, *A*, *R*): *Stare, Acțiune, Recompensă*
- Acțiunile afectează de obicei și recompensele ulterioare, nu numai pe cele imediate: au un efect întârziat

# Gruk

- *Der findes en visdommens vej – det er dén, som bør være let at erindre: dum dig og dum dig og dum dig igen, men mindre og mindre og mindre.*

- *Există un drum al înțelepciunii – este unul ce ar trebui să fie ușor de ținut minte: greșește și greșește și greșește din nou, dar mai puțin și mai puțin și mai puțin.*

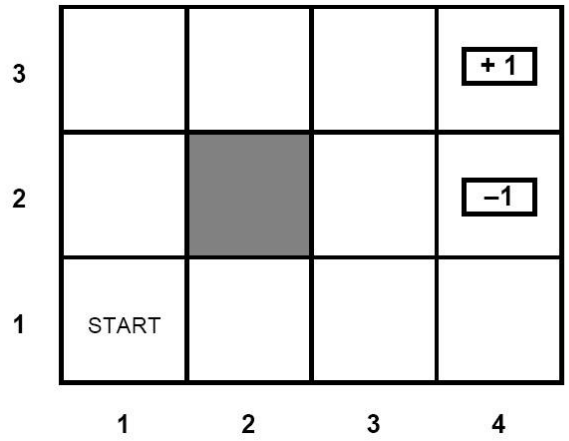
- *The road to wisdom? – Well, it's plain and simple to express: err and err and err again but less and less and less.*

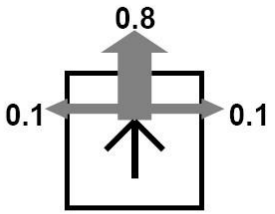
Piet Hein (1905-1996), inventator și poet danez

# Învățarea cu întărire

1. Introducere
2. Procese de decizie Markov
   - 2.1. Iterarea valorilor
   - 2.2. Iterarea politicilor
3. Învățarea pasivă
4. Învățarea activă
5. Optimizări
6. Concluzii

# Decizii secvențiale




- Mediu determinist:
  - (sus, sus, dreapta, dreapta, dreapta)
- Mediu stohastic:
  - Model de tranziții $T(s, a, s')$: probabilitatea de a ajunge din starea $s$ în starea $s'$ efectuând acțiunea $a$

# Presupunerea Markov

- Starea curentă $s_t$ depinde doar de un istoric finit al stărilor anterioare
- Proces Markov de ordin întâi: starea curentă $s_t$ depinde doar de starea anterioară $s_{t-1}$
  - $P(s_t \mid s_{t-1}, \ldots, s_0) = P(s_t \mid s_{t-1})$

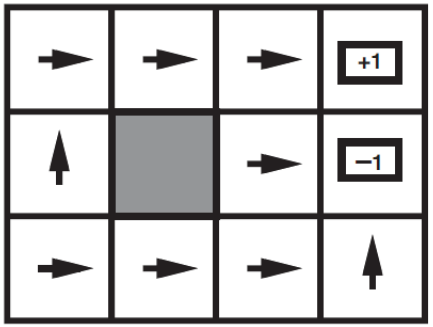
# Proces de decizie Markov

- Starea inițială $s_0$
- Modelul de tranziții $T(s, a, s')$
- Funcția de recompensă $R(s)$

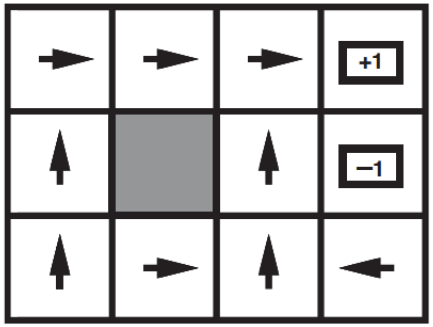
# Definiții

- Soluția unei astfel de probleme se numește politică (*policy*) și se notează $\pi$
  - Traduceri alternative pentru *policy*: tactică sau strategie
- $\pi(s)$ este acțiunea recomandată în starea $s$
  - Politica este descrierea unui reflex
  - Politica este definită pentru toate stările: ce acțiune trebuie să facă agentul în orice stare s-ar afla
- Se numește utilitate suma recompenselor pentru o secvență de stări
  - Recompensa este câștigul imediat, pe termen scurt
  - Utilitatea este câștigul total, pe termen lung
- Într-un mediu stohastic, se calculează utilitatea așteptată
- Politica optimă $\pi^*$ maximizează utilitatea așteptată

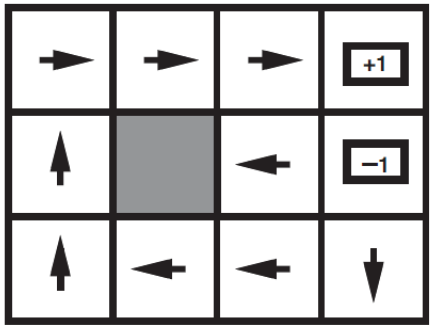
# Exemple de politici



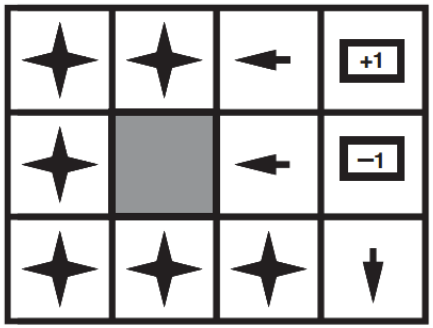

$R(s) < -1.6284$




$-0.4278 < R(s) < -0.0850$




$-0.0221 < R(s) < 0$



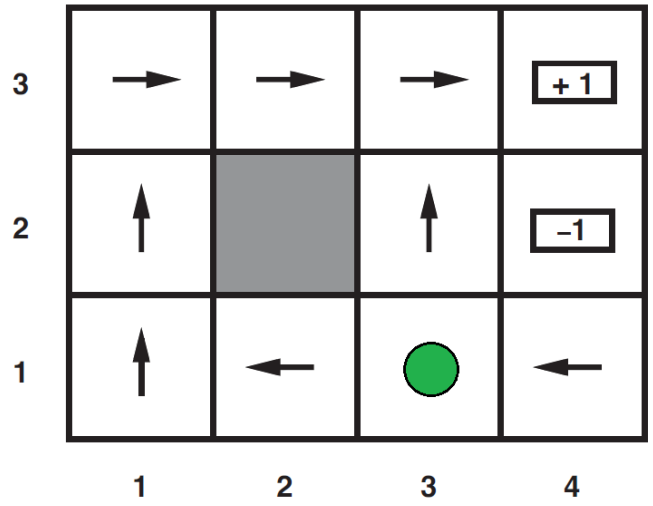

$R(s) > 0$

*R*(*s*) este recompensa în fiecare stare neterminală

# Staționaritate

- Orizont finit
  - $U_h([s_0, s_1, ..., s_{N+k}]) = U_h([s_0, s_1, ..., s_N])$
  - După momentul $N$, nu mai contează nimic

*history*




- $N = 3 \Rightarrow$ trebuie să riște (sus)
- $N = 100 \Rightarrow$ poate alege soluția mai sigură (stânga)
- Politica optimă este nestaționară

# Staționaritate

- Orizont infinit
  - Politica optimă este staționară
  - Recompense aditive
    - $U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$
  - Recompense actualizate (*discounted*)
    - $U([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$
    - $\gamma \in [0, 1]$ este factorul de actualizare (*discount factor*), care indică faptul că recompensele viitoare contează mai puțin decât recompensele imediate

# Evaluarea unui orizont infinit

- 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)

# Evaluarea unei politici

- Fiecare politică generează secvențe multiple de stări, datorită incertitudinii tranzițiilor $T(s, a, s')$
- Utilitatea (sau valoarea) unei politici $\pi$ este valoarea așteptată a sumei tuturor recompenselor actualizate observate după toate secvențele posibile de stări

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right]$$

- Valoarea așteptată: $E[A] = \Sigma_i(P(A_i) \cdot A_i)$

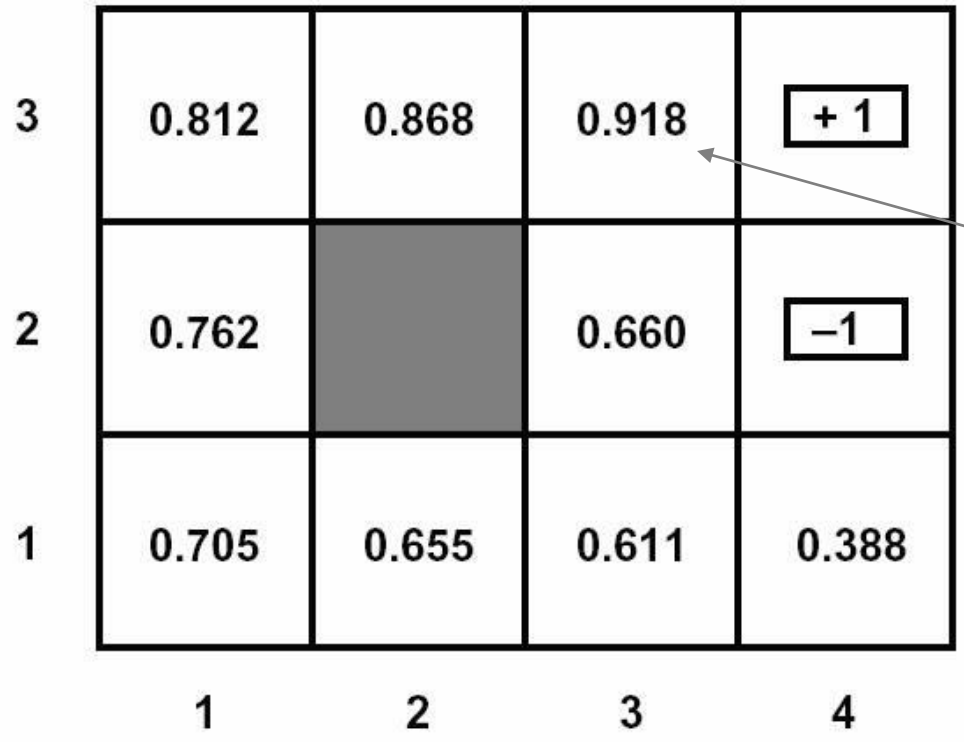
# Utilitățile stărilor

- Utilitatea unei stări $s$ este utilitatea așteptată a secvenței de stări următoare
- Secvența este determinată de $\pi(s)$

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s\right]$$

# Exemplu

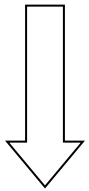
- Fie $\gamma = 1$ și $R(s) = -0.04$




lângă scop utilitățile sunt mai mari pentru că este nevoie de mai puțini pași cu recompensă negativă pentru atingerea stării respective

# Ecuația Bellman

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s\right]$$



$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$



Ecuația Bellman

# Politica optimă

- Utilitatea unei stări este recompensa imediată pentru acea stare plus utilitatea așteptată maximă a stării următoare

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')U(s')$$

- Politica optimă alege acțiunea care conduce în starea cu cea mai mare utilitate așteptată

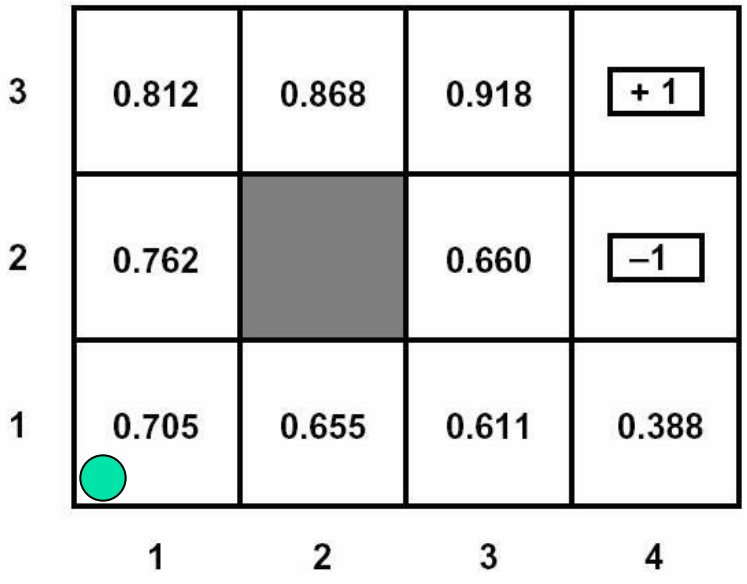
$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} T(s, a, s')U(s')$$

# A2.1. Iterarea valorilor

- engl. “value iteration”
- Este un algoritm pentru calcularea politicii optime
- Se inițializează utilitatea fiecărei stări
- În mod iterativ, se atribuie valorile corecte pentru utilități
- Se folosesc utilitățile stărilor pentru selectarea unei acțiuni optime în fiecare stare
- Se alege starea cu utilitatea cea mai mare

# Exemplu

- Se consideră rezultatele tuturor acțiunilor posibile pentru a selecta acțiunea cea mai bună și a atribui utilitatea sa așteptată următoarei stări din ecuația Bellman
- Exemplu: utilitatea stării (1,1)




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

# 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
- Se rezolvă iterativ:

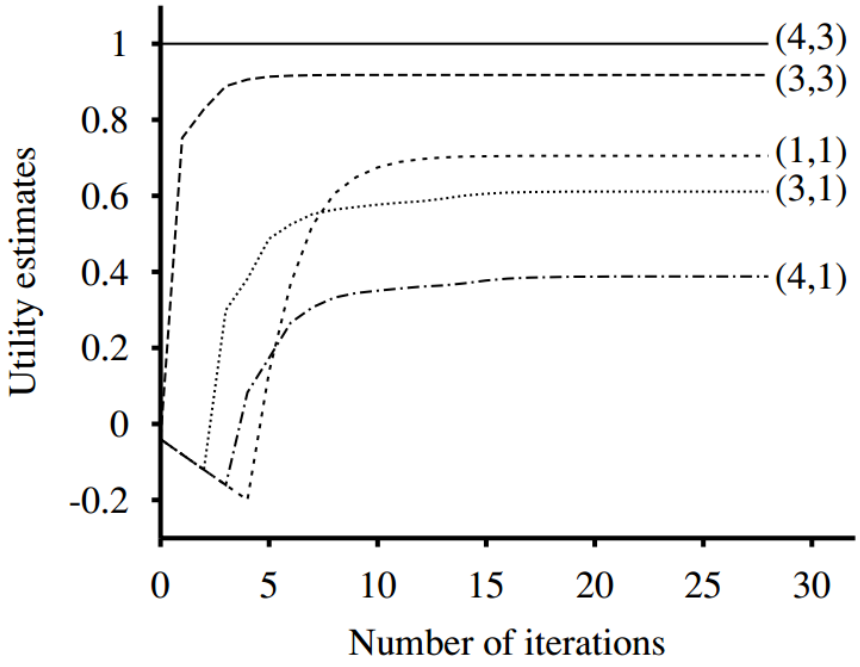
$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U_i(s')$$

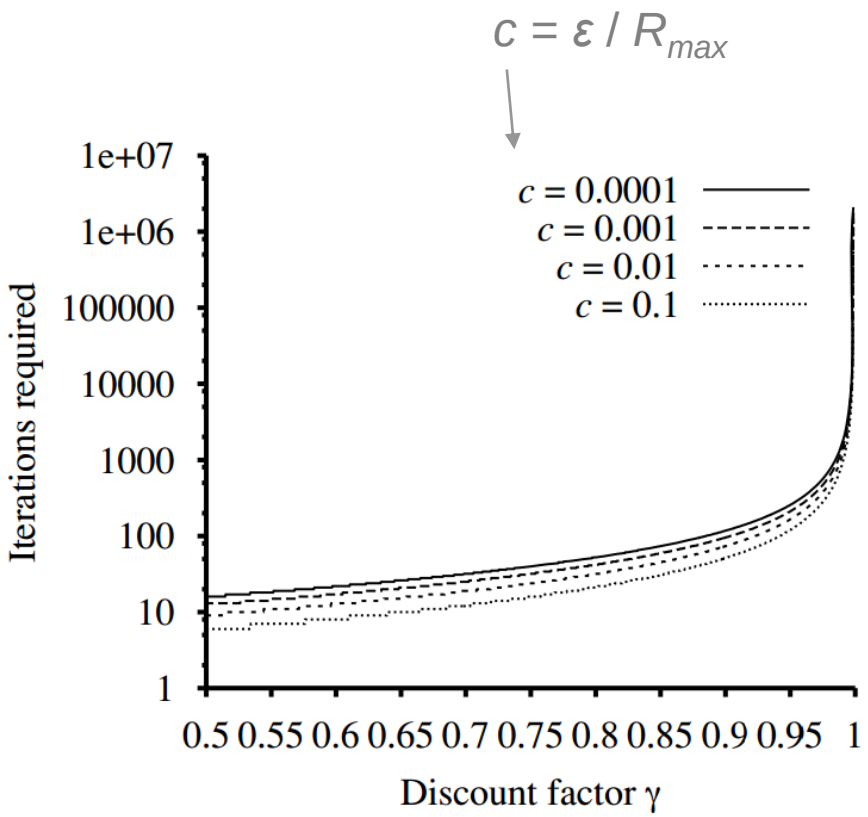
# Pseudocod: iterarea valorilor

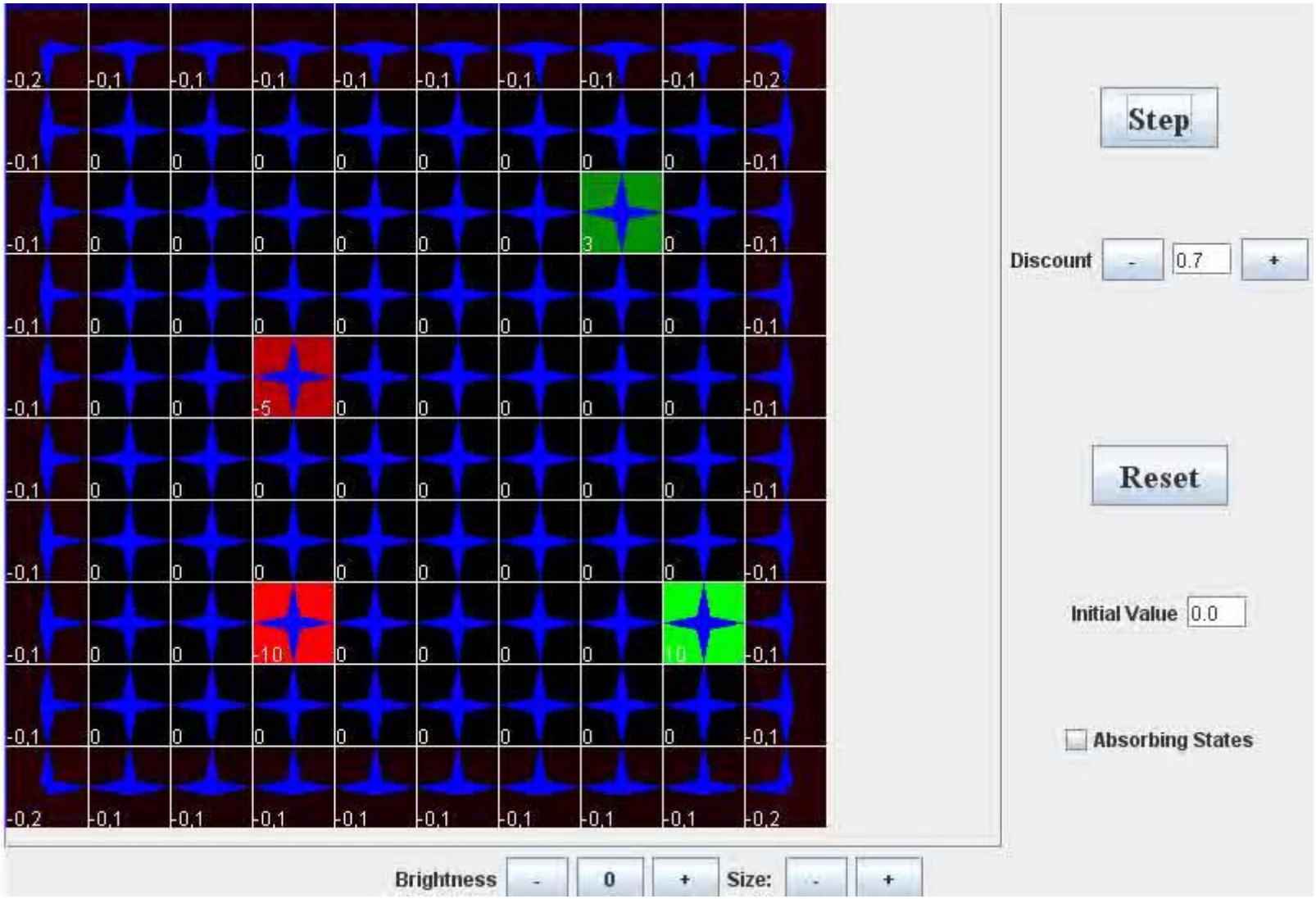
**function** VALUE-ITERATION($mdp, \epsilon$) **returns** a utility function
  **inputs**: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount $\gamma$
    $\epsilon$, the maximum error allowed in the utility of any state
  **local variables**: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

# Convergența






http://people.cs.ubc.ca/~poole/demos/mdp/vi.html

# A2.2. Iterarea politicilor

- engl. “policy iteration”
- Dacă o acțiune este în mod evident mai bună decât toate celelalte, nu avem nevoie de valorile exacte ale utilităților
- Algoritmul alternează doi pași:
  - Evaluarea politicii: calcularea utilităților tuturor stărilor pe baza politicii $\pi_i$
  - Îmbunătățirea politicii: calcularea unei noi politici $\pi_{i+1}$ pe baza utilităților $U_i$

# Evaluarea politicii

$$U_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i(s')$$

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

# Îmbunătățirea politicii

- Se cunosc toate $U(s)$
- Se calculează pentru fiecare $s$:

$$\max_a \sum_{s'} T(s,a,s')\ U[s']$$

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

# Pseudocod: iterarea politicilor

**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
    $\pi$, a policy vector indexed by state, initially random

  **repeat**
    $U \leftarrow$ POLICY-EVALUATION($\pi$, $U$, $mdp$)
    $unchanged? \leftarrow$ 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** $\pi$

# Învățarea cu întărire

1. Introducere
2. Procese de decizie Markov
3. Învățarea pasivă
   - 3.1. Estimarea directă a utilității
   - 3.2. Programarea dinamică adaptivă
   - 3.3. Învățarea diferențelor temporale
4. Învățarea activă
5. Optimizări
6. Concluzii

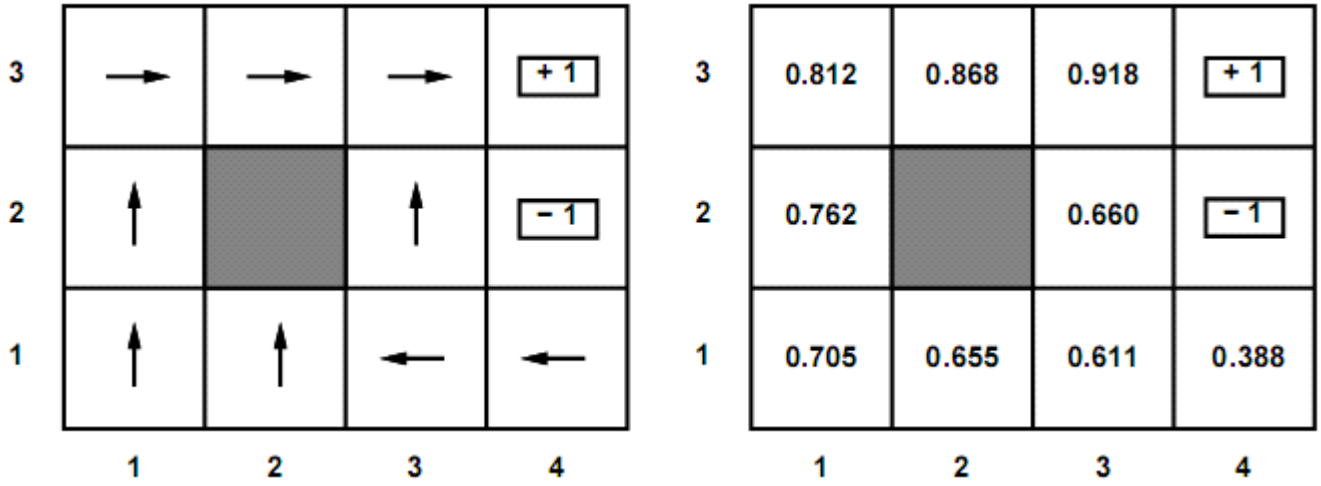
# Învățarea cu întărire

- Proces de decizie Markov
  - Mulțimea de stări *S*, mulțimea de acțiuni *A*
  - Modelul de tranziții *T*(*s*, *a*, *s'*) 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

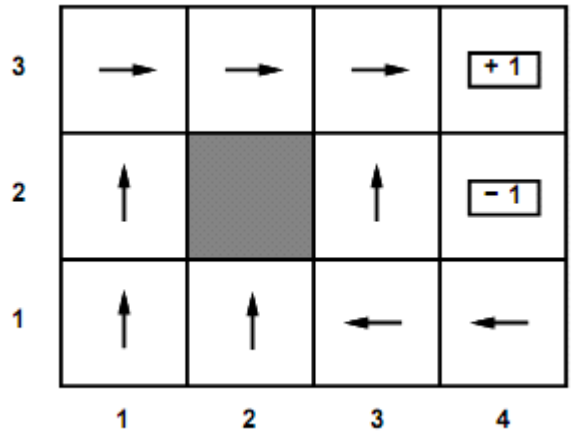
- Pasivă sau activă
  - Pasivă: agentul execută o politică fixă și o evaluează
  - Activă: agentul își actualizează politica pe măsură ce învață
- Bazată pe model sau fără model
  - Bazată pe model: învață modelul de tranziții și recompense și îl folosește pentru a descoperi politica optimă
  - Fără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

# Învățarea pasivă




- Învățarea pasivă este o modalitate de explorare a mediului
  - De exemplu, un robot cu scopul de a trece prin toate stările mediului
- Evaluează cât de bună este o politică $\pi$
- Învață utilitatea $U^{\pi}(s)$ a fiecărei stări
- Idee similară cu evaluarea politicii pentru PDM

# Învățarea pasivă




Agentul execută o serie de încercări (*trials*)

Politica este aceeași, dar mediul este nedeterminist

$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
$$(1,1)_{\textbf{-.04}} \rightsquigarrow (2,1)_{\textbf{-.04}} \rightsquigarrow (3,1)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (4,2)_{\textbf{-1}}$$

Scopul este să învețe utilitatea așteptată $U^{\pi}(s)$

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right]$$

# A3.1. Estimarea directă a utilității

- De exemplu, prima încercare produce:
  - în starea (1,1) recompensa totală 0.72

$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$

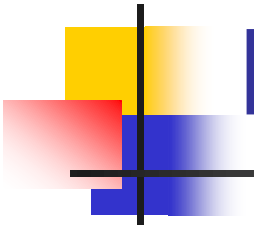
$-0.04 \cdot 7 + 1 = 0.72$ (cu $\gamma = 1$)

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right]$$

# Estimarea directă a utilității

- De exemplu, prima încercare produce:
  - în starea (1,1) recompensa totală 0.72
  - 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 unei stări este suma așteptată a recompenselor de la acea stare înainte
- Utilitatea estimată poate fi media valorilor eșantionate
  - $U(1,1) = 0.72$, $U(1,2) = 0.80$, $U(1,3) = 0.84$ etc.

# Estimarea directă a utilității



- Utilitatea unei stări depinde de utilitățile stărilor succesoare (constrângerile date de ecuațiile Bellman)
- Estimarea directă a utilității ignoră această informație
- Căutarea se face într-un spațiu mult mai mare decât cel necesar de fapt
- Convergența este foarte lentă

# A3.2. Programarea dinamică adaptivă

- Se folosesc ecuațiile Bellman

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^{\pi}(s')$$

- Trebuie estimate $T(s, \pi(s), s')$ și $R(s)$ din încercări
  - Frecvențele tranzițiilor și mediile recompenselor
  - Probabilitățile și recompensele învățate se introduc în ecuațiile Bellman
  - Se rezolvă sistemul de ecuații liniare cu necunoscutele $U^{\pi}(s)$

# Pseudocod: învățare cu întărire pasivă prin programare dinamică adaptivă

**function** PASSIVE-ADP-AGENT($percept$) **returns** an action
  **inputs**: $percept$, a percept indicating the current state $s'$ and reward signal $r'$
  **persistent**: $\pi$, a fixed policy
    $mdp$, an MDP with model $P$, rewards $R$, discount $\gamma$
    $U$, a table of utilities, initially empty
    $N_{sa}$, a table of frequencies for state–action pairs, initially zero
    $N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
    $s$, $a$, the previous state and action, initially null

  **if** $s'$ is new **then** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
  **if** $s$ is not null **then**
    increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$
    **for each** $t$ such that $N_{s'|sa}[t, s, a]$ is nonzero **do**
      $P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \,/\, N_{sa}[s, a]$
  $U \leftarrow$ POLICY-EVALUATION($\pi$, $U$, $mdp$)
  **if** $s'$.TERMINAL? **then** $s, a \leftarrow$ null **else** $s, a \leftarrow s', \pi[s']$
  **return** $a$

*t* este o stare

având aproximările pentru *T*, *R* și π, se calculează utilitățile ca la iterarea politicilor PDM, prin rezolvarea unui sistem de ecuații liniare

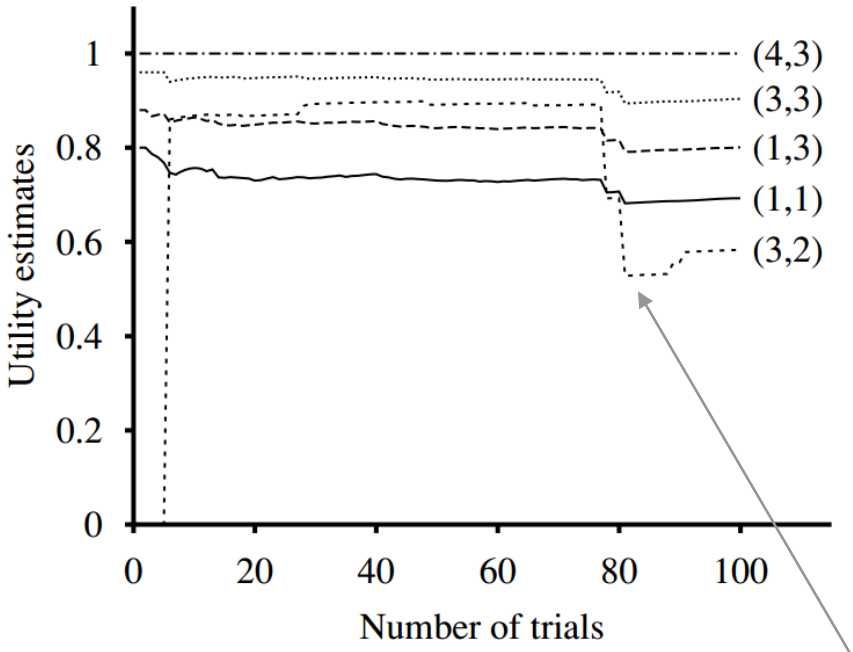
# Exemplu

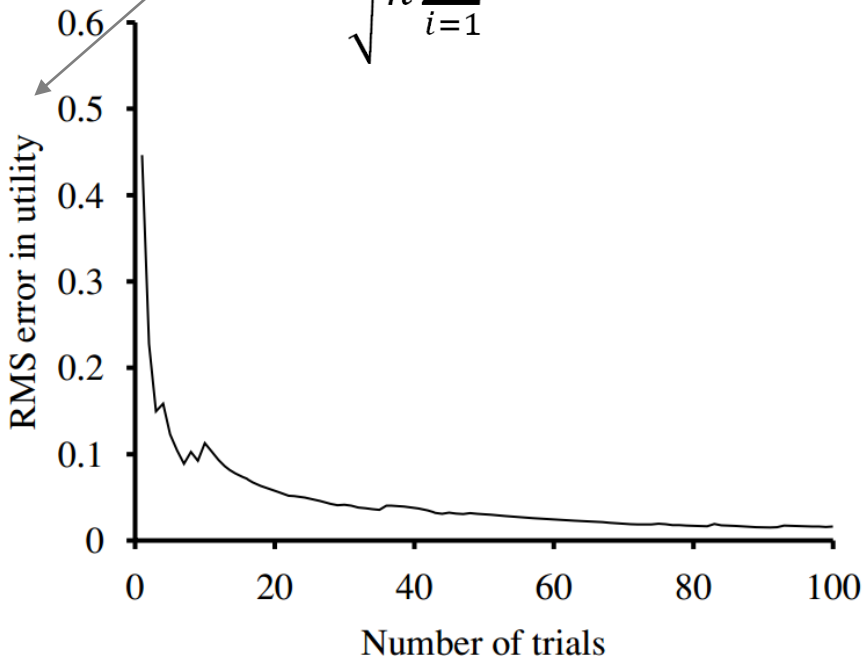
$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow \boxed{(1,3)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}}} \rightsquigarrow \boxed{(1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
$$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow \boxed{(1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
$$(1,1)_{\textbf{-.04}} \rightsquigarrow (2,1)_{\textbf{-.04}} \rightsquigarrow (3,1)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (4,2)_{\textbf{-1}}$$

- Acțiunea *Right* este executată de 3 ori în starea (1,3) și în 2 cazuri starea rezultantă este (2,3)
  - $\Rightarrow T((1,3), \textit{Right}, (2,3)) = 2/3$

# Convergența

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2}$$




aici agentul întâlnește prima dată starea cu $R = -1$

# Programarea dinamică adaptivă

- PDA este un standard de comparare pentru alți algoritmi de învățare cu întărire
- PDA este ineficient dacă spațiul stărilor este mare
  - Sistem de ecuații liniare de ordin *n*
  - Jocul de table: $10^{50}$ ecuații cu $10^{50}$ necunoscute

# Euristici pentru PDA

- Mai ales la începutul învățării, modelul oricum nu este cel corect, deci calculul exact al tuturor utilităților nu se justifică
- Se pot folosi euristici, de exemplu, baleierea prioritară (*prioritized sweeping*): se ajustează cu precădere stările ai căror succesori probabili tocmai au suferit o modificare mare a utilităților estimate
- Prin folosirea euristicilor, calculele sunt efectuate cu câteva ordine de magnitudine mai eficient

# A3.3. Învățarea diferențelor temporale

- engl. "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
- Exemplu: $(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$
  $(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$
  $(1,1)_{\textbf{-.04}} \rightsquigarrow (2,1)_{\textbf{-.04}} \rightsquigarrow (3,1)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (4,2)_{\textbf{-1}}$
  - După prima încercare: $U(1,3) = 0.84$, $U(2,3) = 0.92$
  - Fie tranziția $(1,3) \rightarrow (2,3)$ în a doua încercare
  - Între cele două stări, constrângerea dată de ecuația Bellman impune ca $U(1,3) = -0.04 + U(2,3) = 0.88$ (cu $\gamma = 1$)
  - Estimarea $U(1,3) = 0.84$ este mai mică și trebuie mărită puțin

# Învățarea diferențelor temporale

- Ecuația diferențelor temporale propune un compromis:

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

- Metoda aplică o serie de corecții pentru a converge la soluția ecuațiilor Bellman
- Corecțiile au loc proporțional cu probabilitățile: actualizările pentru *s'* vor avea loc în $T(s, \pi(s), s')$ din cazuri
- Rata de învățare $\alpha$ determină viteza de convergență la utilitatea reală
- Metoda diferențelor temporale nu are nevoie de model pentru a realiza actualizările

# Rata de învățare

- Dacă $\alpha$ este fix, atunci valoarea medie a lui $U^{\pi}(s)$ converge la valoarea corectă
- Dacă $\alpha$ descrește pe măsură ce numărul de vizitări $n$ ale unei stări crește, atunci chiar $U(s)$ converge la valoarea corectă
- Convergența este garantată dacă:

$$\sum_{m=1}^{\infty} \alpha(m) = \infty \qquad \sum_{m=1}^{\infty} \alpha^2(m) < \infty$$

- Funcția $\alpha(n) = 1/n$ satisface această condiție
- De multe ori, se folosește funcția $\alpha(n) = 1 / (n+1) \in (0, 1]$

# Pseudocod: învățare cu întărire pasivă prin diferențe temporale

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

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

**persistent**: $\pi$, 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**

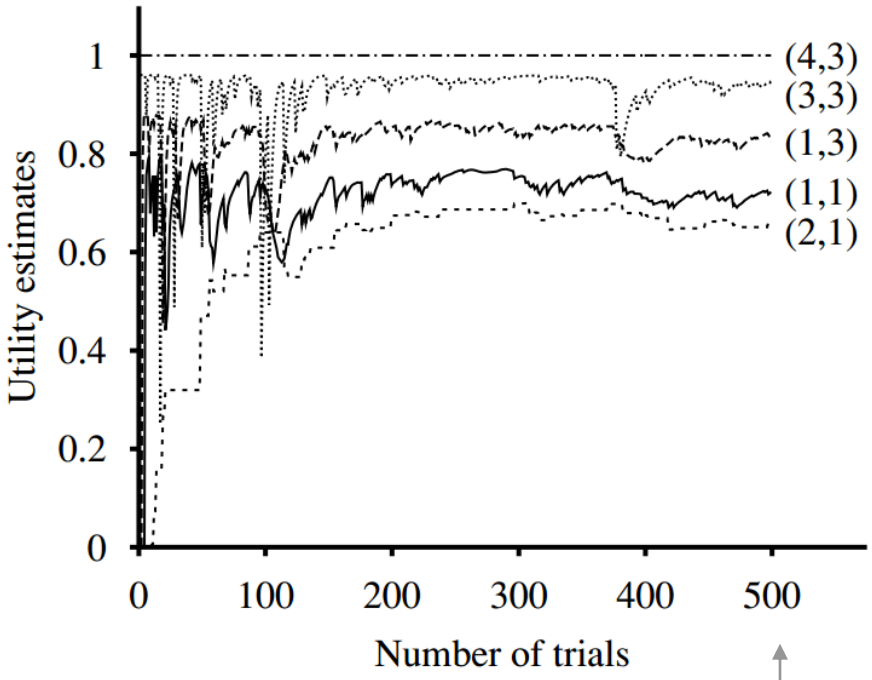
increment $N_s[s]$

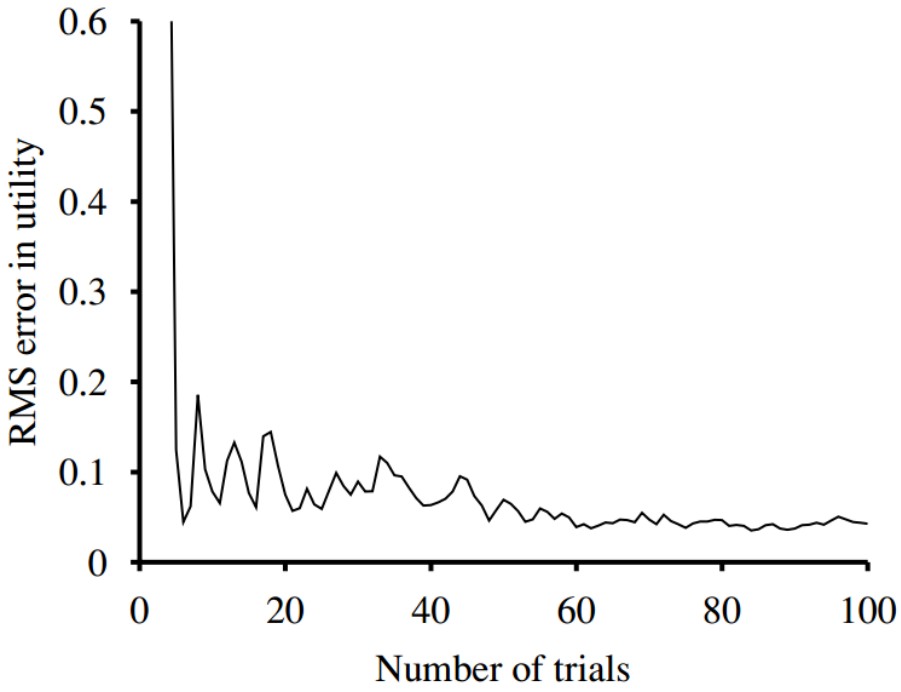
$U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma\, U[s'] - U[s])$

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

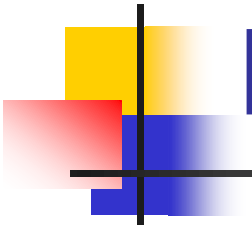
**return** $a$

# Convergența




la PDA era aproximativ 100



# Discuție

- DT nu au 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 converg mai lent, dar execută calcule mai simple
- DT pot fi văzute ca o aproximare a PDA

# Învățarea cu întărire

1. Introducere
2. Procese de decizie Markov
3. Învățarea pasivă
4. Învățarea activă
   - 4.1. Explorare și exploatare
   - 4.2. Algoritmul Q-Learning
5. Optimizări
6. Concluzii

# Învățarea activă

- Agentul pasiv învață utilitățile stărilor 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, la un moment dat, funcția de 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
- Exploatarea
  - Agentul oprește învățarea și execută acțiunile date de politică
  - Are ca efect maximizarea recompenselor folosind estimările curente
- Explorarea
  - Agentul învață încercând acțiuni noi, în mod aleatoriu
  - Poate conduce la maximizarea recompenselor pe termen lung

# Explorare sau exploatare?

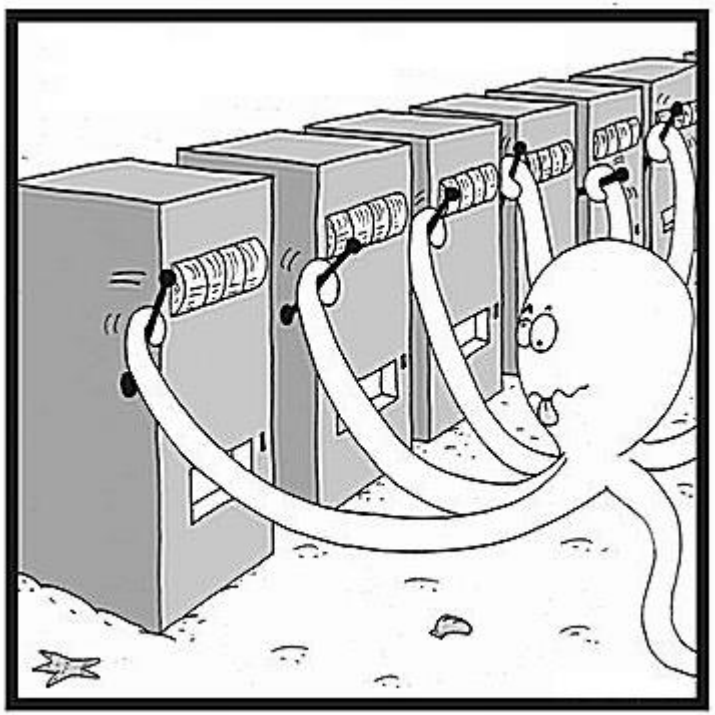
- Agentul trebuie să favorizeze:
  - Explorarea stărilor și acțiunilor necunoscute sau
  - Exploatarea stărilor și acțiunilor despre care știe deja că îi aduc recompense mari
- Soluții pentru dilemă
  - Metoda *ε-greedy*
  - Alte metode GLIE (*greedy in the limit of infinite exploration*)
    - O metodă GLIE trebuie să încerce fiecare acțiune în fiecare stare de un număr teoretic nelimitat de ori
    - În final, trebuie să devină *greedy*

# A4.1. Metoda $\varepsilon$-*greedy*

- Fie $\varepsilon \in [0, 1]$
- Acțiunea următoare selectată va fi:
  - O acțiune aleatorie cu probabilitatea $\varepsilon$
  - Acțiunea optimă cunoscută cu probabilitatea $1 - \varepsilon$
- Implementare:
  - Inițial $\varepsilon = 1$ (explorare pură)
  - Când se termină un episod de învățare, $\varepsilon$ scade, de exemplu, cu 0.05 (crește progresiv rata de exploatare)
  - $\varepsilon$ nu scade niciodată sub un prag, de exemplu, 0.1. Astfel, agentul are mereu o șansă de explorare, pentru a evita optimele locale

# Exploatarea și explorarea

- Exploatare pură
  - De obicei generează politici suboptime
- Explorare pură
  - Învață modele din ce în ce mai bune, dar recompensele sunt mici
- *n-armed bandit* ⟶



# Metodă GLIE alternativă

- Se acordă ponderi mai mari acțiunilor neîncercate frecvent
- Se acordă ponderi mai mici acțiunilor cu utilitate mică
- Se modifică ecuațiile Bellman folosind utilități optimiste $U^+(s)$

$$U^+(s) = R(s) + \gamma \max_a f\left(\sum_{s'} T(s,a,s')U^+(s'), N(a,s)\right)$$

numărul de aplicări ale acțiunii *a* în starea *s*

- Funcția de explorare $f(u,n)$
  - Crește cu utilitatea așteptată *u*
  - Scade cu numărul de încercări *n*
- Exemplu:

parametru fix

$$f(u,n) = \begin{cases} R^+, & \text{if} n < N \\ u, & \text{otherwise} \end{cases}$$

estimare optimistă a celei mai mari recompense care poate fi obținută în orice stare

- Se încurajează acțiunile către regiuni neexplorate
- Oferă convergență mai bună și politici aproape optime

# A4.2. Algoritmul *Q-Learning*

- Algoritmul *Q-Learning* învață o funcție $Q(a,s)$
  - $Q$ vine de la *quality* (calitatea unei combinații stare-acțiune)
  - Utilitățile $U(s) = \max_a Q(a,s)$
- Ecuațiile de constrângere la echilibru

$$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s')$$

- *Q-Learning* este o metodă DT fără model

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

# Algoritmul *Q-Learning*

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

- Actualizările se fac de fiecare dată când acțiunea *a* aplicată în *s* duce în *s'*
- Rata de învățare $\alpha$ determină viteza de actualizare a estimărilor
  - De obicei, $\alpha \in (0,1)$
- *Q-Learning* este mai lent decât PDA

# Pseudocod: *Q-Learning*

**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** $Q[s, None] \leftarrow r'$

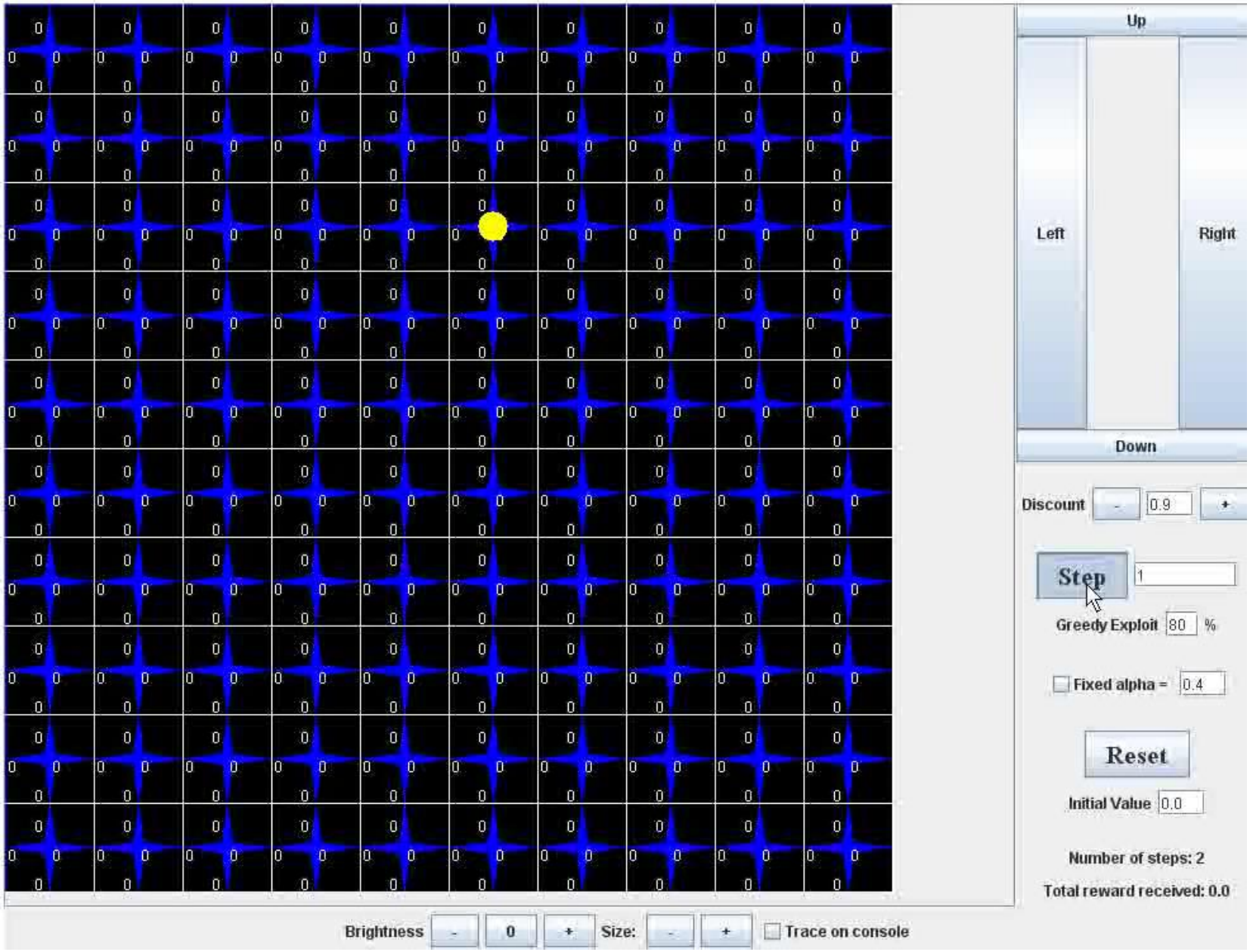
**if** $s$ is not null **then**

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** $a$




http://people.cs.ubc.ca/~poole/demos/rl/q.html

# Învățarea cu întărire

1. Introducere
2. Procese de decizie Markov
3. Învățarea pasivă
4. Învățarea activă
5. Optimizări
6. Concluzii

# Aproximarea utilităților

- Problemele reale au spații foarte mari
  - Jocul de table are aproximativ $10^{50}$ stări
- Modelul este greu de memorat
  - PDA memorează $T(s,a,s')$, $N(a,s)$
  - DT, QL memorează $U(s)$, respectiv $Q(a,s)$
- Utilitățile se pot aproxima cu funcții parametrice

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \cdots + \theta_n f_n(s)$$

  - $f_i$ sunt funcții de bază
  - Rezultă o compresie semnificativă (pentru table $\approx 1{:}10^{44}$)
- Învățarea modelului aproximativ ajută generalizarea
  - Se pot aproxima utilitățile stărilor vizitate
  - Se pot prezice utilitățile stărilor nevizitate

# Exemplu

- Pentru problema prezentată cu 4 x 3 stări

$$\hat{U}_\theta(x,y) = \theta_0 + \theta_1 x + \theta_2 y$$

$$(\theta_0, \theta_1, \theta_2) = (0.5, 0.2, 0.1) \Rightarrow \hat{U}_\theta(1,1) = 0.8$$

$$E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$$

eroarea — utilitatea aproximată — utilitatea observată

# Actualizarea parametrilor

$$E_j(s) = (\hat{U}_\theta(s) - u_j(s))^2/2$$

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j(s)}{\partial \theta_i} = \theta_i + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

$$\begin{aligned}
\theta_0 &\leftarrow \theta_0 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) \\
\theta_1 &\leftarrow \theta_1 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) x \\
\theta_2 &\leftarrow \theta_2 + \alpha \left(u_j(s) - \hat{U}_\theta(s)\right) y
\end{aligned}$$

parametrii se actualizează la fiecare pas / observație

schimbarea parametrilor la un moment dat modifică *toate* utilitățile

# Aproximarea utilităților

- În prezent, pentru mediile complexe, utilitățile se aproximează folosind rețele neuronale profunde

# Învățarea aproximărilor

- Actualizările pentru diferențe temporale

$$\theta_i \leftarrow \theta_i + \alpha\,[R(s) + \gamma\,\hat{U}_\theta(s') - \hat{U}_\theta(s)]\frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

- Actualizările pentru *Q-Learning*

$$\theta_i \leftarrow \theta_i + \alpha\,[R(s) + \gamma\,\max_{a'}\hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)]\frac{\partial \hat{Q}_\theta(s, a)}{\partial \theta_i}$$

# Căutarea politicii

- Reprezintă politica drept o funcție parametrizată
- Actualizează parametrii până când politica se îmbunătățește
- Reprezentare simplă: $\pi(s) = \max_a \hat{Q}_\theta(a, s)$
- Se ajustează $\theta$ pentru a se îmbunătăți politica
  - $\pi$ este o funcție neliniară de $\theta$, discontinuă pentru acțiuni discrete
  - Actualizările pe baza gradienților nu sunt posibile
- Politica se poate reprezenta stohastic (softmax):

$$\pi_\theta(s, a) = \frac{\exp(\hat{Q}_\theta(s, a))}{\sum_{a'} \exp(\hat{Q}_\theta(s, a))}$$

  - Dă probabilități de acțiune
  - Apropiată de varianta deterministă (max) dacă o acțiune este mult mai bună decât alte acțiuni
  - Politica devine o funcție diferențiabilă de $\theta$
- Politica diferențiabilă poate fi optimizată cu metode bazate pe gradient

# Funcția softmax: exemplu

- $Q_1 = 2$, $Q_2 = 0$, $Q_3 = 5$, $Q_4 = 4$
- Abordarea deterministă:
  - max $\Rightarrow Q_3$
- Abordarea stohastică:

$$\pi_\theta(s,a) = \frac{\exp(\hat{Q}_\theta(s,a))}{\sum_{a'} \exp(\hat{Q}_\theta(s,a))}$$

  - $\exp(Q_1) = 7.4$, $\exp(Q_2) = 1$, $\exp(Q_3) = 148.4$, $\exp(Q_4) = 54.6$
  - suma = 211.4
  - $\Rightarrow P(Q_1) = 7.4 / 211.4 = 3.5\%$
  - Analog, $P(Q_2) = 0.5\%$, $P(Q_3) = 70.2\%$, $P(Q_4) = 25.8\%$
- Dacă valoarea maximă este mult mai mare decât celelalte, probabilitatea sa se apropie de 1

# Concluzii

- Învățarea cu întărire este necesară pentru agenții care evoluează în medii necunoscute
- În lumea reală, există numeroase aplicații bazate pe învățarea cu întărire
- Învățarea pasivă presupune evaluarea unei politici date
- Învățarea activă presupune învățarea unei politici optime
- Aproximarea funcțiilor este necesară pentru probleme cu spații mari de stări