

# Reinforcement learning (I)

IA 2025/2026

# Conținut

## Introducere

Procese de decizie Markov

Value iteration

Policy iteration

# Învățare cu întărire (*Reinforcement learning*)

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

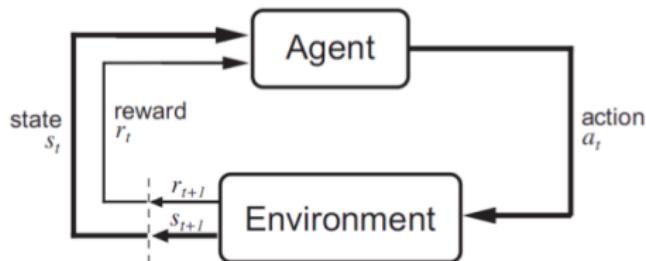
- ▶ Agentul are o sarcină de îndeplinit
- ▶ Efectuează o serie de acțiuni
- ▶ Primește *feedback* din partea mediului: cât de bine a acționat pentru a-și îndeplini sarcina.

Agentul primește o recompensă pozitivă dacă îndeplinește bine sarcina, altfel o recompensă negativă.

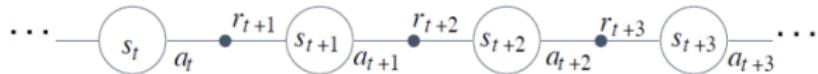
Această modalitatea de învățare se numește **învățare cu întărire**.

# Modelul de interacțiune

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



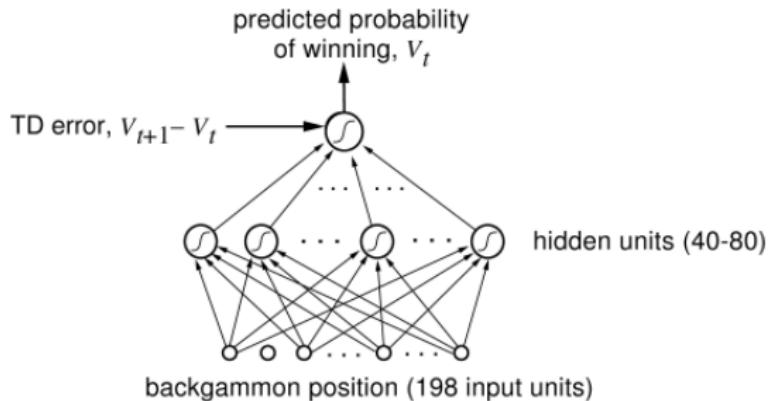
Traекторie / episod  $(s_0, a_0, r_1, s_1, a_1, r_2, \dots)$



# 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
  - ▶ Obținerea de date și învățare supervizată pe aceste date  
Date de antrenare:  $(S, A, R)$  Stare, Acțiune, Recompensă

## Aplicații: TD-Gammon



O rețea neuronală antrenată cu *TD-learning*

- ▶ intrarea: configurație
- ▶ ieșirea: o estimare a valorii pentru acea configurație

Invață din simulări (joacă jocuri împotriva lui însuși)

# Aplicații: robotică

AIBO: învață să meargă '04  
(utilizând *Policy gradient*)



Fig. 5. The training environment for our experiments. Each Aibo times itself as it moves back and forth between a pair of landmarks (A and A', B and B', or C and C').

[https://www.cs.utexas.edu/users/  
AustinVilla/?p=research/learned\\_walk](https://www.cs.utexas.edu/users/AustinVilla/?p=research/learned_walk)

Minitaur quadrupedal robot: învață să meargă (*actor-critic deep RL*)

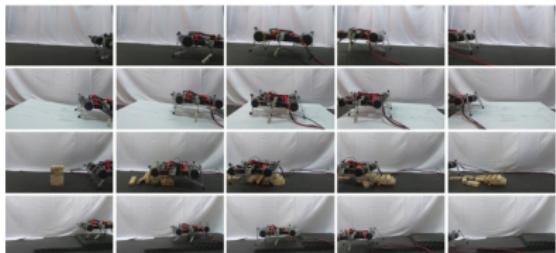


Fig. 9: We trained the Minitaur robot to walk on flat terrain (first row) in about two hours. At test time, we introduced obstacles, including a slope, wooden blocks, and steps, which were not present at training time, and the learned policy was able to generalize to the unseen situations without difficulty (other rows).

[https://sites.google.com/view/  
minitaur-locomotion/](https://sites.google.com/view/minitaur-locomotion/)

# Deep Reinforcement Learning

**AlphaGo** (Google DeepMind, '15): programul a învățat să joace jocurile Atari 2600 urmărind doar afișajul și scorul

- ▶ 2016: a câștigat împotriva lui Lee Sedol, jucător de GO profesionist cu 9 dan, premiu: 1 000 000\$
- ▶ 2017: a câștigat împotriva lui Ke Jie, cel mai bun jucător de GO

**AlphaGo Zero**: a învățat să joace fără informații din jocuri ale persoanelor umane și a învins AlphaGo cu 100-0 (oct. '17)

**AlphaZero** a învins AlphaGo Zero cu 60-40 și a ajuns după doar 9 ore de antrenare la un nivel superior tuturor programelor de șah și GO existente (dec. '17)

# Conținut

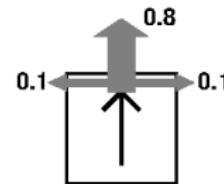
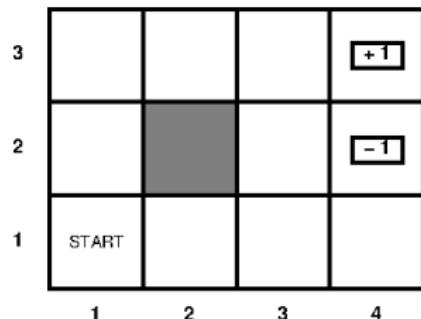
Introducere

Procese de decizie Markov

Value iteration

Policy iteration

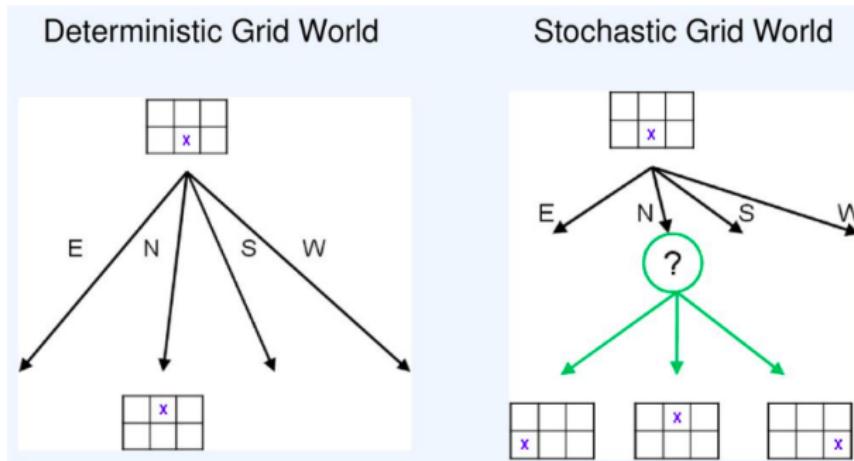
# Decizii secentuale



- ▶ Mediu determinist
  - ▶ (sus, sus, dreapta, dreapta, dreapta)
- ▶ Mediu stochastic
  - ▶ Model de tranzitii  $P(s'|s, a)$ : 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

# Deterministic vs. stochastic

Stochastic: pentru sevență (sus, sus, dreapta, dreapta, dreapta), ajungem în starea finală cu probabilitatea  $0.8^5 = 0.33$



# Presupunerea Markov

Proces Markov: un model matematic pentru a descrie o secvență de evenimente.

- ▶ 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  $s_{t-1}$

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

# Proces de decizie Markov (*Markov Decision Process*)

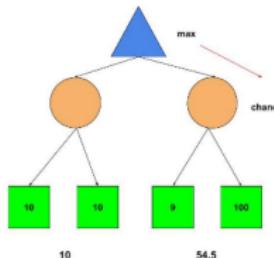
Proces de decizie Markov: o problemă de decizie sevențială pentru un mediu stochastic cu un model de tranziție Markov și recompense aditive

- ▶ 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 specifică ce acțiune să execute agentul în fiecare stare (**politică  $\pi$** ).

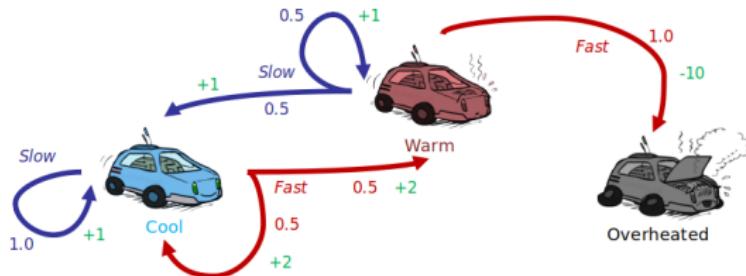
Sunt probleme de căutare nedeterministe.

- ▶ Alg. de căutare. *Expectimax*: când oponentul nu joacă optim.

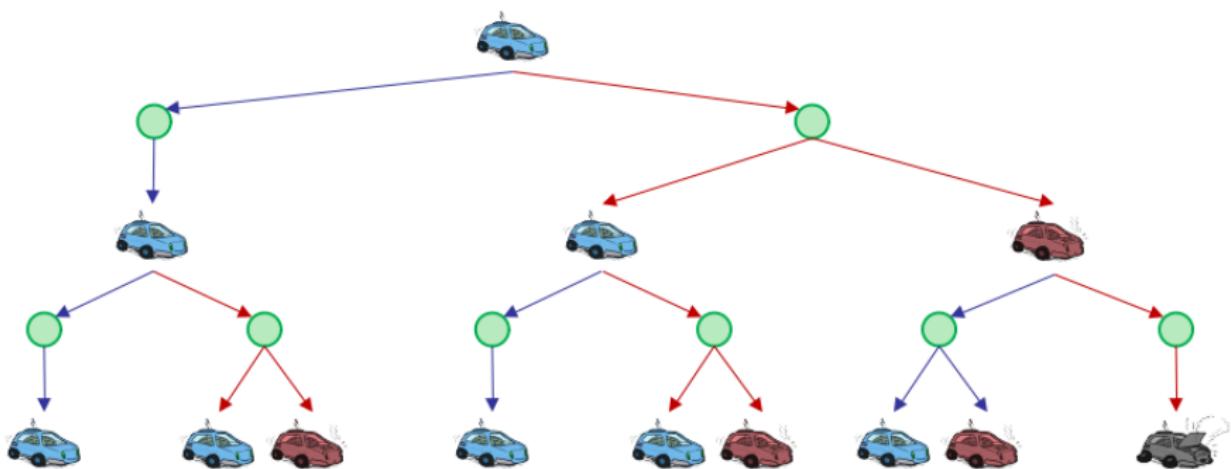


## Exemplu

- ▶ Un robot (mașină) vrea să călătorească departe, repede.
- ▶ Stări: *Cool*, *Warm*, *Overheated*. Acțiuni: *Slow*, *Fast*. Dacă merge mai repede, primește o recompensă dublă.

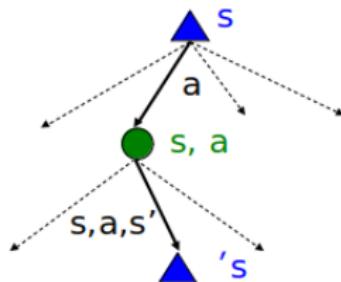


## Exemplu: arborele de căutare



## Arborei de căutare pentru MDP

- ▶ Fiecare stare proiectează un arbore de căutare.



$s$  stare,  $(s, a, s')$  tranziție,  $P(s'|s, a)$ ,  $R(s, a, s')$

- ▶ În problemele de căutare, scopul este de a identifica o secvență optimă.

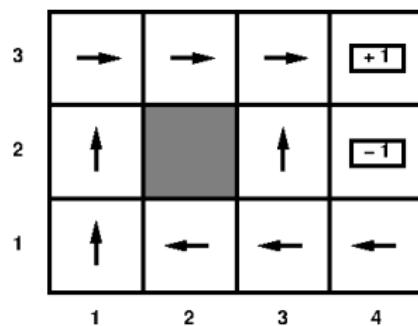
În MDP, scopul este de a identifica o politică optimă  $\pi^*$  (strategie)

- ▶  $\pi : S \rightarrow A$

$\pi(s)$  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.
- ▶ **Calitatea unei politici:** utilitatea așteptată a secvențelor posibile de stări.  
Mediu stochastic: putem avea o secvență diferită de stări când executăm aceeași politică din starea inițială.
- ▶ **Politica optimă**  $\pi^*$  maximizează utilitatea așteptată.

Exemplu: politica optimă și valorile stăriilor

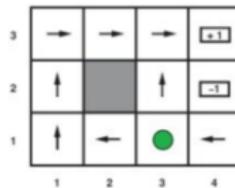


|   |       |       |       |       |
|---|-------|-------|-------|-------|
| 3 | 0.812 | 0.868 | 0.912 | +1    |
| 2 | 0.762 |       | 0.660 | -1    |
| 1 | 0.705 | 0.655 | 0.611 | 0.388 |
|   | 1     | 2     | 3     | 4     |

## Orizont finit

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

**Exemplu:**



- ▶  $N = 3 \rightarrow$  trebuie să rîște (sus)
- ▶  $N = 100 \rightarrow$  poate alege soluția mai sigură (stânga)

## Orizont infinit

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

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

- ▶ b. Recompense **actualizate** (*discounted*)

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

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

## Recompense actualizate

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

- ▶ Valorile recompenselor scad exponential

Exemplu 1:  $\gamma = 0.5$

$$R(s_0) = 1, R(s_1) = 2, \dots$$

$$U([1, 2, 3]) = 1 * 1 + 0.5 * 2 + 0.25 * 3$$

$$U([1, 2, 3]) < U([3, 2, 1])$$

Exemplu 2: Quiz

## 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:

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

- ▶ **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  $\gamma = 1$

## Utilitatea așteptată

- ▶ Fiecare politică generează secvențe multiple de stări, datorită incertitudinii tranzitiei  $P(s'|s, a)$
- ▶ Fie  $S_t$  o variabilă aleatoare: starea în care ajunge agentul la momentul  $t$  executând politica  $\pi$ ;  $S_0 = s$ .  
Utilitatea așteptată obținută prin execuția politicii  $\pi$  din starea  $s$ :

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

( = valoarea așteptată a sumei recompenselor actualizate, obținute pentru toate secvențele posibile de stări)

# Evaluarea unei politici

- ▶ Politica optimă

$$\pi_s^* = \operatorname{argmax}_{\pi} U^\pi(s)$$

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

Exemplu: fie  $\gamma = 1$  și  $R(s) = -0.04$ .

|   |       |       |       |       |
|---|-------|-------|-------|-------|
| 3 | 0.812 | 0.868 | 0.912 | +1    |
| 2 | 0.762 |       | 0.660 | -1    |
| 1 | 0.705 | 0.655 | 0.611 | 0.388 |

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

# Maximum Expected Utility

Principiul **Maximum Expected Utility**: alege acțiunea care maximizează utilitatea așteptată a stării următoare

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

## 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,  $R(s)$ , plus utilitatea așteptată maximă a stării următoare.

## Exemplu

Utilitatea stării (1,1):

$$U(1,1) = -0.04 + \gamma \max[0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), \quad (Up)$$
$$\qquad\qquad\qquad 0.9U(1,1) + 0.1U(1,2), \quad (Left)$$
$$\qquad\qquad\qquad 0.9U(1,1) + 0.1U(2,1), \quad (Down)$$
$$\qquad\qquad\qquad 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)] \quad (Right)$$

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*

# Conținut

Introducere

Procese de decizie Markov

Value iteration

Policy iteration

## 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( $mdp, \epsilon$ ) **returns** a utility function

**inputs:**  $mdp$ , an MDP with states  $S$ , actions  $A(s)$ , transition model  $P(s' | s, a)$ , 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**

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

**if**  $|U'[s] - U[s]| > \delta$  **then**  $\delta \leftarrow |U'[s] - U[s]|$

**until**  $\delta < \epsilon(1 - \gamma)/\gamma$

**return**  $U$

# Iterarea valorilor

Value Iteration, for estimating  $\pi \approx \pi_*$

Algorithm parameter: a small threshold  $\theta > 0$  determining accuracy of estimation  
Initialize  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop:

```
|   Δ ← 0
|   Loop for each  $s \in \mathcal{S}$ :
|      $v \leftarrow V(s)$ 
|      $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$ 
|     Δ ← max(Δ, |v - V(s)|)
until Δ < θ
```

Output a deterministic policy,  $\pi \approx \pi_*$ , such that

$$\pi(s) = \arg \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$$

Din Sutton&Barto. *Reinforcement Learning: an introduction*

Funcția utilitate  $U \leftrightarrow V$  funcție valoare

## Iterarea valorilor

- ▶ Exemplu: Cursa
- ▶ Demo: <https://courses.grainger.illinois.edu/cs440/fa2018/lectures/mdp-value-demo.pdf>

## Probleme ale algoritmului Iterarea valorilor

- ▶  $O(S^2A)$  per iterare
- ▶ Valoarea “max” pentru fiecare stare se modifică rar
- ▶ Politica converge adesea cu mult înaintea valorilor

# Conținut

Introducere

Procese de decizie Markov

Value iteration

Policy iteration

## II. Iterarea politicilor

- ▶ Dacă **fixăm politica**, avem o singură acțiune per stare
- ▶ Algoritmul alternează următorii pași:
  - ▶ 1. **Evaluarea politicii**: dată o politică  $\pi_i$ , calculează  $U_i = U^{\pi_i}$  utilitățile stărilor pe baza politicii  $\pi_i$
  - ▶ 2. **Îmbunătățirea politicii**: calculează o nouă politică  $\pi_{i+1}$ , pe baza utilităților  $U_i$

Repetă acești pași până când politica converge

# 1. Evaluarea politicii

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

## Ecuații Bellman simplificate

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

- ▶ Pentru  $n$  stări, sistem de  $n$  ecuații liniare cu  $n$  necunoscute
- ▶ Se poate rezolva exact în  $O(n^3)$  sau în mod *aproximativ*
- ▶ Aplicăm *Value iteration*

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

# Evaluarea politicii

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  $P(s' | s, a)$ 
    local variables:  $U$ , a vector of utilities for states in  $S$ , initially zero
                     $\pi$ , a policy vector indexed by state, initially random

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

Demo: [https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld\\_dp.html](https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html)

# Metode de tip Monte-Carlo

Sunt utilizate atunci când mediul este necunoscut.

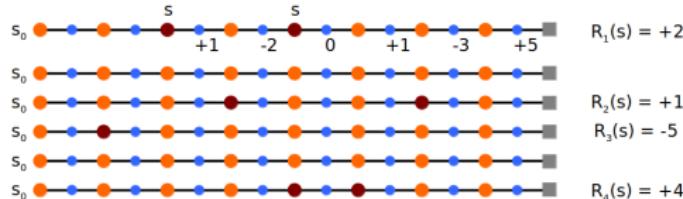
Idee: **eșantionare** pentru a estima funcțiile utilitate  $U_\pi(s), \forall s$ .

Generăm traiectorii pentru fiecare episod.

Recompensa (*reward-to-go*) de la momentul  $t$ :  $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots$   
 $U_\pi(s) = E[G_t | S_t = s]$ .

**Estimăm utilitatea unei stări:**

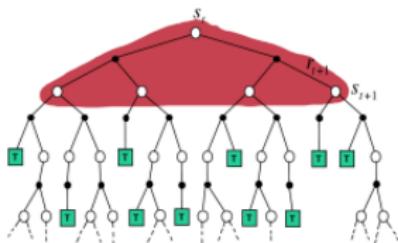
- ▶ *First-visit MC*: media sumei recompenselor obținute în urma primelor vizitări ale lui  $s$



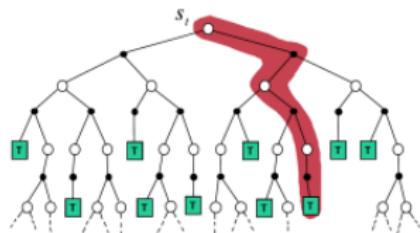
$$V(s) \approx (2 + 1 - 5 + 4)/4 = 0.5$$

- ▶ *Every-visit MC*: media sumei recompenselor obținute pentru toate vizitările lui  $s$

# Programare dinamică vs. Monte Carlo



Dynamic programming  
bootstrapping



Monte Carlo  
sampling

Monte Carlo: aşteptăm până la terminarea episodului pentru a calcula recompensele  $G_t$ . O variație mai mare decât la DP.

# Bibliografie

- ▶ Russel&Norvig. Artificial Intelligence: a modern approach. Ch 17. Making complex decisions
  
- ▶ Sutton&Barto. Reinforcement Learning: an introduction. Ch 3. Finite Markov Decision Processes, Ch 4. Dynamic Programming  
<http://incompleteideas.net/book/RLbook2020.pdf>