



# APRENDIZADO POR REFORÇO

## Aula 3: Métodos Tabulares

Lucas Pereira Cotrim

Marcos Menon José

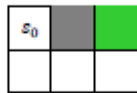
[lucas.cotrim@maua.br](mailto:lucas.cotrim@maua.br)

[marcos.jose@maua.br](mailto:marcos.jose@maua.br)

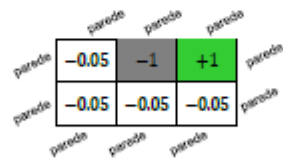
# ESCLARECIMENTOS E1

## Exercício E1 – Processos de Decisão de Markov (MDPs)

- 1) Dado um ambiente do tipo *Grid World* composto por 6 casas em duas fileiras, conforme a figura abaixo:

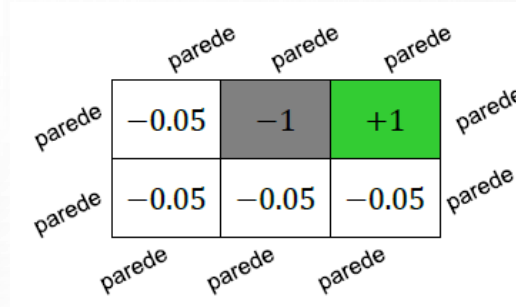


Considere um robô móvel na posição  $s_0$  do ambiente com capacidade de se movimentar nas 4 direções:  $\mathcal{A} = \{N=\uparrow, E=\rightarrow, S=\downarrow, W=\leftarrow\}$ . A posição de destino do robô é representada pela casa verde, enquanto a casa cinza representa um obstáculo (ambos são estados terminais, qualquer ação tomada mantém o agente no próprio estado). Uma Função de Recompensa  $\mathcal{R}$  que representa a tarefa de posicionamento do robô é ilustrada abaixo, indicando a recompensa obtida por um agente ao tomar qualquer ação na casa correspondente ( $\mathcal{R}(s, \cdot)$ ):



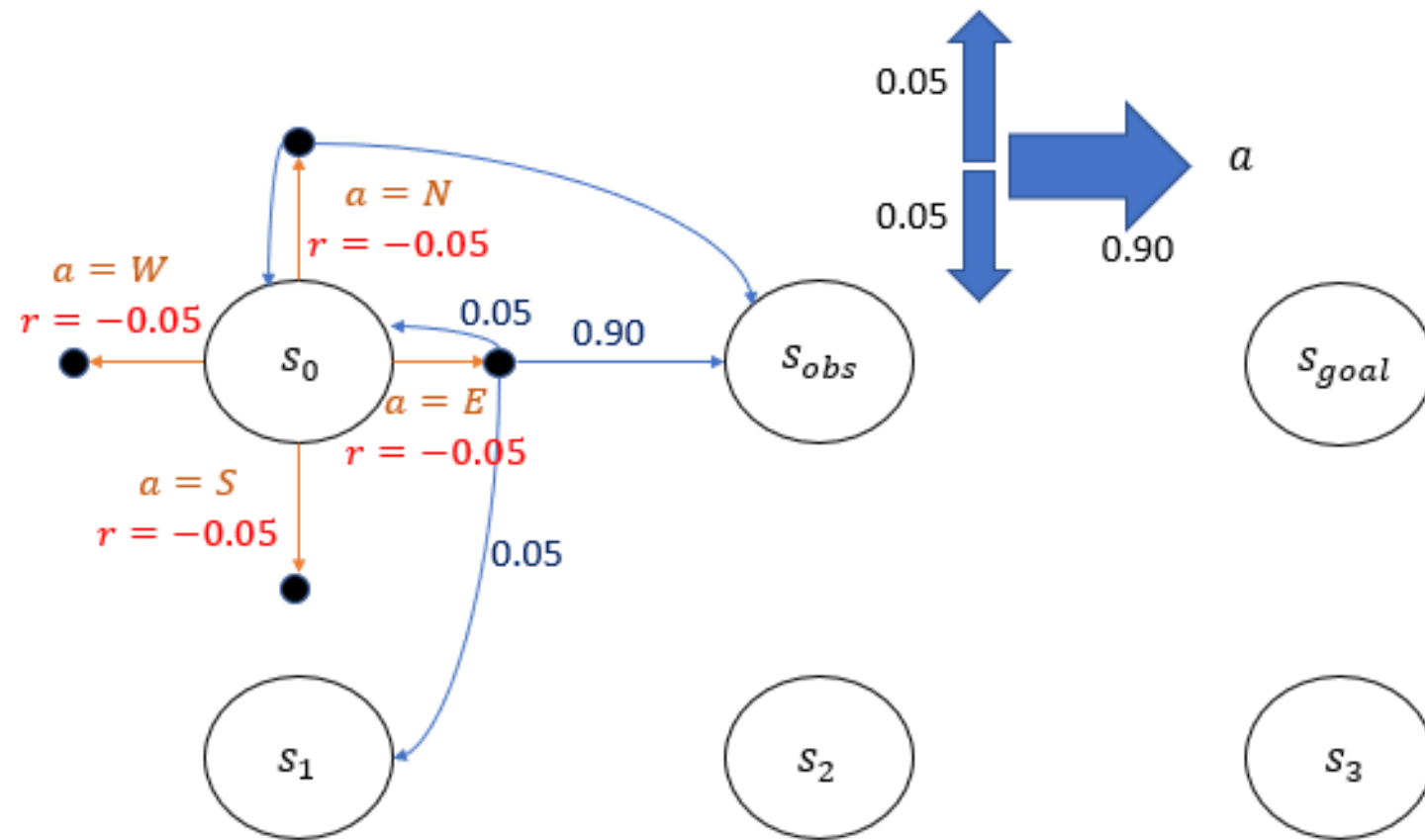
Quando o robô executa uma ação, ele se move para a casa vizinha na direção escolhida em 90% das vezes, com 5% de chance de ocorrer um escorregamento em cada direção perpendicular à ação tomada. Se o robô fosse colidir com uma parede externa do ambiente ele em vez disso permanece na mesma posição.

Todas as ações podem ser tomadas em qualquer estado, somente após o agente escolher a ação tomada que a função de probabilidades de transição do ambiente ( $\mathcal{P}_{ss'}^a$ ) determina o estado seguinte de acordo com o escorregamento.



- Todas as 4 ações  $\{\uparrow, \downarrow, \rightarrow, \leftarrow\}$  são possíveis em todos os 6 estados.
- O problema segue o seguinte ordem:
  - 1) Agente escolhe qual ação  $a \in \mathcal{A}$  tomar.
  - 2) Agente recebe recompensa  $r(s, a)$ .
  - 3) Dinâmica de transição de estados  $\mathcal{P}_{ss'}^a$ , é aplicada.
  - 4) Possível colisão com parede é verificada.

# ESCLARECIMENTOS E1



$$V_{s_0} = 0.25[-0.05 + \gamma(0.90V_{obs} + 0.05V_{s_1} + 0.05V_{s_0})] \\ + 0.25[ \dots ] \\ + 0.25[ \dots ] \\ + 0.25[ \dots ]$$

$$V_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_{\pi}(s') \right)$$

# RELEMBRANDO ÚLTIMA AULA

Na última aula vimos:

- A representação de um ambiente de Aprendizado por Reforço como um MDP.

## PROCESSO DE DECISÃO DE MARKOV (MDP)

31

Um **Processo de Decisão de Markov** (ou MDP) é uma tupla  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ , onde:

- $\mathcal{S}$  é um conjunto finito de estados.
- $\mathcal{A}$  é um conjunto finito de ações.
- $\mathcal{P}$  é uma função  $\mathcal{P}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1] \subset \mathbb{R}$  de probabilidades de transições de estados.  
$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$
- $\mathcal{R}$  é uma função de recompensa  $\mathcal{R}: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  tal que  $\mathcal{R}(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- $\gamma \in [0,1] \subset \mathbb{R}$  é um fator de desconto.

Um MDP é a extensão de um MRP onde estuda-se o processo de tomada de decisões de um agente que interage com o ambiente de modo a maximizar as recompensas obtidas.

# RELEMBRANDO ÚLTIMA AULA

Na última aula vimos:

- Como relacionar os valores de estados com os valores de estados sucessores a partir da Equação de Bellman.

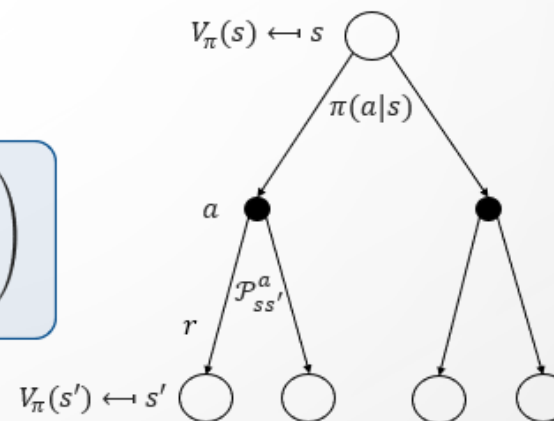
## EQUAÇÃO DE BELLMAN PARA MDP ( $V_\pi$ )

42

Substituindo (4) em (3), podemos escrever  $V_\pi(s)$   
em função de  $V_\pi(s')$ :

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_\pi(s') \right)$$

Bellman Expectation Equation for  $V_\pi$



# RELEMBRANDO ÚLTIMA AULA

Na última aula vimos:

- Como avaliar uma política qualquer  $\pi$  para obter sua função Valor  $V_\pi(s)$  por meio da solução analítica da Equação de Bellman.

## EQUAÇÃO DE BELLMAN PARA MDP EM FORMA MATRICIAL ( $V_\pi$ ) <sup>45</sup>

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_\pi(s') \right)$$

Utilizando o MRP induzido pela política  $\pi$  podemos escrever a Equação de Bellman em forma matricial:

$$v_\pi = \mathcal{R}^{(\pi)} + \gamma \mathcal{P}^{(\pi)} v, \quad \text{onde} \quad \begin{cases} \mathcal{P}_{ss'}^{(\pi)} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}_{ss'}^a \\ \mathcal{R}_s^{(\pi)} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}(s, a) \end{cases}$$

com solução:

$$v_\pi = (I - \gamma \mathcal{P}^{(\pi)})^{-1} \mathcal{R}^{(\pi)}$$



Em problemas práticos o MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  associado ao problema pode ser muito complexo ou desconhecido.

Na aula de hoje veremos:

- Como resolver os problemas de previsão (obter  $V_\pi, Q_\pi$ ) e controle (obter  $V^*, Q^*, \pi^*$ ) de modo iterativo com Programação Dinâmica, quando conhecemos o modelo do MDP (Model-Based Prediction/Control).
- Como resolver o problema de previsão (obter  $V_\pi, Q_\pi$ ) somente por experiência, quando o modelo do MDP é desconhecido (Model-Free Prediction).

# TÓPICOS DA AULA

- Programação Dinâmica (DP)
  - Avaliação de Política (Policy Evaluation)
  - Iteração sobre Função Política (Policy Iteration)
  - Iteração sobre Função Valor (Value Iteration)
- Model-Free Prediction
  - Aprendizado por Métodos de Monte-Carlo
  - Aprendizado por Diferenças Temporais (Temporal-Difference Learning)



# PROGRAMAÇÃO DINÂMICA (DP)

Programação Dinâmica (DP)

# PROGRAMAÇÃO DINÂMICA: DEFINIÇÃO

Programação Dinâmica é um método geral para solução de problemas com as seguintes propriedades:

- Subestrutura Ótima
  - Princípio da Otimalidade
  - Solução Ótima pode ser decomposta em subproblemas
- Superposição de Subproblemas
  - Recorrência de subproblemas
  - Soluções podem ser armazenadas e reutilizadas

Processos de Decisão de Markov (MDPs) apresentam ambas propriedades devido à decomposição recursiva da **Equação de Bellman** e ao armazenamento de soluções em uma **Função Valor**

# PLANEJAMENTO POR PROGRAMAÇÃO DINÂMICA

Programação Dinâmica constitui um conjunto de algoritmos que assumem conhecimento completo do MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  e podem ser utilizados em tarefas de planejamento:

- Previsão
  - **Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  e política  $\pi$  (ou MRP  $M' = \langle \mathcal{S}, \mathcal{P}_{ss'}^{(\pi)}, \mathcal{R}_s^{(\pi)}, \gamma \rangle$ )
  - **Output:** Funções Valor  $V_\pi$  e  $Q_\pi$
- Controle
  - **Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - **Output:** Funções Valor e Política ótimas  $V^*$ ,  $Q^*$  e  $\pi^*$ .

Programação Dinâmica (DP)

Avaliação de Política (Policy Evaluation)

# DP: POLICY EVALUATION

A Avaliação Iterativa de uma Função Política de Ações  $\pi$  consiste em obter funções valores sucessivas que se aproximam da Função Valor  $V_\pi$ :

$$V_0 \rightarrow V_1 \rightarrow \dots \rightarrow V_\pi$$

Isso pode ser feito com Atualizações Síncronas (Synchronous Backups):

- Inicializa-se  $V_0$  aleatoriamente.
- Repetimos para  $k = 0, 1, \dots$ :
  - Para todos estados  $s \in \mathcal{S}$ :
  - Obtemos  $V_{k+1}(s)$  a partir de  $V_k(s')$ , onde  $s'$  é um estado sucessor de  $s$ .

Como atualizar  $V_k \rightarrow V_{k+1}$ ? Equação de Bellman

# DP: POLICY EVALUATION

Atualização de  $V_k$  para obter  $V_{k+1}$ :

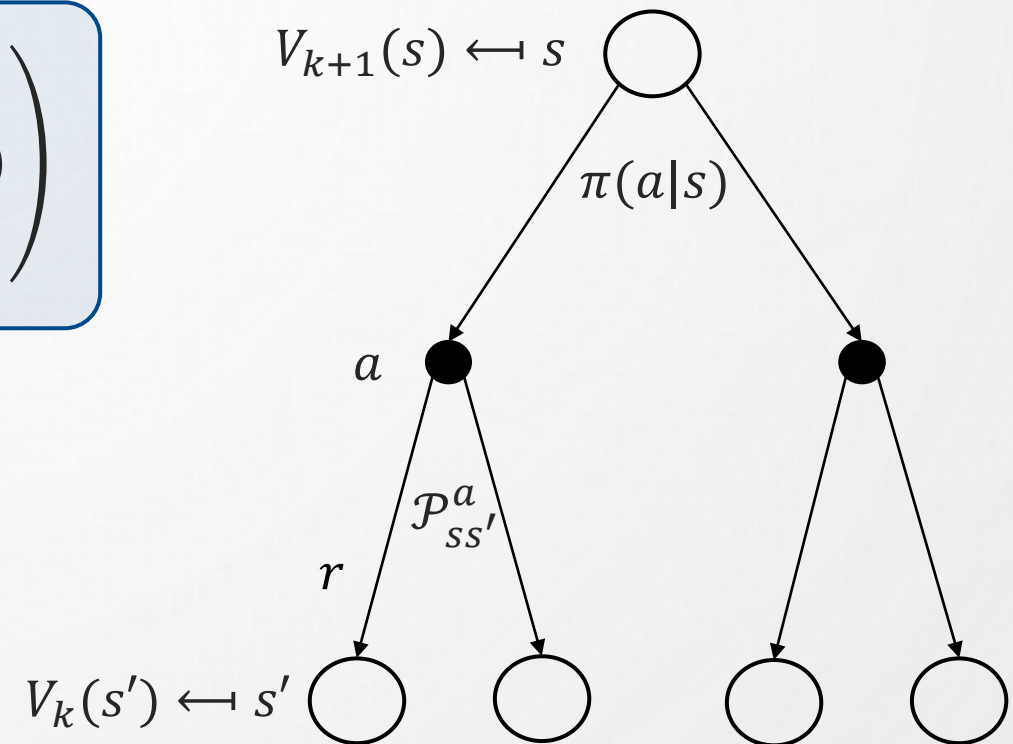
$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

Em forma matricial:  $\mathbf{v}_{k+1} = \mathbf{R}^{(\pi)} + \gamma \mathbf{P}^{(\pi)} \mathbf{v}_k$

Percebe-se que, após convergência, a equação de Bellman de  $V_\pi$  é satisfeita:

$$V_{k+1}(s) = V_k(s), \forall s \in \mathcal{S} \Rightarrow V_{k+1}(s) = V_\pi(s)$$

onde  $V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_\pi(s') \right)$





# DP: POLICY EVALUATION

**Algoritmo:** Avaliação Iterativa de Política  $\pi$  para obtenção de  $V_\pi$  (Atualizações Síncronas)

**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  e política  $\pi$  a ser avaliada.

**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

**Inicializar**  $V_0(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

$k = 0$

Repetir:

$\Delta = 0$

Repetir para cada  $s \in \mathcal{S}$ :

$$V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

$$\Delta \leftarrow \max\{\Delta, |V_{k+1}(s) - V_k(s)|\}$$

$k \leftarrow k + 1$

Até que  $\Delta < \theta$

**Retorna:**  $V_{k+1}$

# DP: POLICY EVALUATION - EXEMPLO

Dado o seguinte *gridworld*, vamos definir o MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ :

- $\mathcal{S} = \{[0,0], [0,1], [0,2], [0,3], [1,0], [1,1], [1,2], [1,3], [2,0], [2,1], [2,2], [2,3], [3,0], [3,1], [3,2], [3,3]\}$
- $\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$
- $\mathcal{P}$  é determinístico ( $\mathcal{P}_{ss'}^a$  são matrizes binárias com  $\sum_{s'} \mathcal{P}_{ss'}^a = 1, \forall s, a$ ) e o estado sucessor é o indicado pela ação. No caso de ações que levariam ao exterior do *gridworld* ou ações tomadas em estados terminais o agente permanece no mesmo estado.
- $\mathcal{R}(s, a) = -1$  para qualquer ação tomada em um estado em branco e  $\mathcal{R}(s, a) = 0$  para qualquer ação tomada em um estado terminal (sombreado).
- $\gamma = 1$ : MDP episódico não descontado.

[0,0]	[0,1]	[0,2]	[0,3]
[1,0]	[1,1]	[1,2]	[1,3]
[2,0]	[2,1]	[2,2]	[2,3]
[3,0]	[3,1]	[3,2]	[3,3]

Agente Aleatório:

$$\pi(\uparrow | s) = \pi(\downarrow | s) = \pi(\rightarrow | s) = \pi(\leftarrow | s) = 0.25, \forall s \in \mathcal{S}$$

# DP: POLICY EVALUATION - EXEMPLO

$V_0$

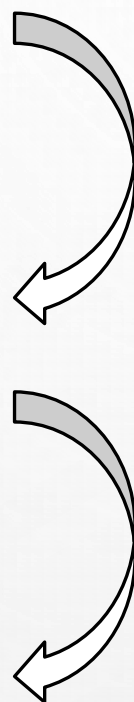
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$V_1$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$V_2$

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0



**Algoritmo:** Avaliação Iterativa de Política  $\pi$  para obtenção de  $V_\pi$  (Atualizações Síncronas)

**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  e política  $\pi$  a ser avaliada.

**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

Inicializar  $V_0(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

$k = 0$

Repetir:

$\Delta = 0$

Repetir para cada  $s \in \mathcal{S}$ :

$$V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

$$\Delta \leftarrow \max\{\Delta; |V_{k+1}(s) - V_k(s)|\}$$

$k \leftarrow k + 1$

Até que  $\Delta < \theta$

**Retorna:**  $V_{k+1}$

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

$$-1.75 = 0.25[-1 + 1(-1)] + 0.25[-1 + 1(-1)] + 0.25[-1 + 1(-1)] + 0.25[-1 + 1(0)]$$

# DP: POLICY EVALUATION - EXEMPLO

$$V_3$$

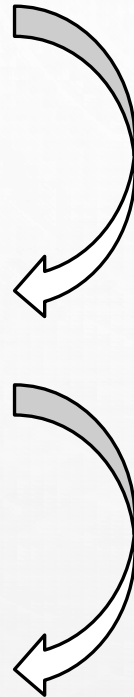
0	-2.44	-2.94	-3
-2.44	-2.87	-3	-2.94
-2.94	-3	-2.87	-2.44
-3	-2.94	-2.44	0

$$V_{10}$$

0	-6.14	-8.35	-8.97
-6.14	-7.74	-8.43	-8.35
-8.35	-8.43	-7.74	-6.14
-8.97	-8.35	-6.14	0

$$V_\infty$$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0



**Algoritmo:** Avaliação Iterativa de Política  $\pi$  para obtenção de  $V_\pi$  (Atualizações Síncronas)

**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  e política  $\pi$  a ser avaliada.

**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

Inicializar  $V_0(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

$k = 0$

Repetir:

$\Delta = 0$

Repetir para cada  $s \in \mathcal{S}$ :

$$V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

$$\Delta \leftarrow \max\{\Delta; |V_{k+1}(s) - V_k(s)|\}$$

$k \leftarrow k + 1$

Até que  $\Delta < \theta$

**Retorna:**  $V_{k+1}$

$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

# DP: POLICY EVALUATION - EXEMPLO

## DP\_Methods.ipynb

DP\_Methods.ipynb ☆

File Edit View Insert Runtime Tools Help

Comment Share Settings

RAM Disk

Editing

Table of contents

Dynamic Programmig: Policy Evaluation, Policy Iteration, Value Iteration

Imports

Funções Auxiliares

Environment Class

Create Environment

Função get\_state\_index(s)

Random Policy

DP: Iterative Policy Evaluation

Plot Value Function

DP: Policy Iteration

greedy\_improvement(env,V)

check\_policy\_stable(old\_policy, new\_policy):

Policy Iteration(env, theta)

DP: Value Iteration

+ Section

+ Code + Text

Dynamic Programmig: Policy Evaluation, Policy Iteration, Value Iteration

Vamos implementar o algoritmo de Programação Dinâmica para Avaliação Iterativa de uma Política com Atualizações Síncronas. O problema a ser resolvido é o de posicionamento de um agente em um ambiente Grid World conforme visto em aula:

DP: POLICY EVALUATION - EXEMPLO

Dado o seguinte *gridworld*, vamos definir o MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  :

- $\mathcal{S} = \{[0,0], [0,1], [0,2], [0,3], [1,0], [1,1], [1,2], [1,3], [2,0], [2,1], [2,2], [2,3], [3,0], [3,1], [3,2], [3,3]\}$
- $\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$
- $\mathcal{P}$  é determinístico ( $\mathcal{P}_{ss'}^a$  são matrizes binárias com  $\sum_{s'} \mathcal{P}_{ss'}^a = 1, \forall s, a$ ) e o estado sucessor é o indicado pela ação. No caso de ações que levariam ao exterior do *gridworld* ou ações tomadas em estados terminais o agente permanece no mesmo estado.
- $\mathcal{R}(s, a) = -1$  para qualquer ação tomada em um estado em branco e  $\mathcal{R}(s, a) = 0$  para qualquer ação tomada em um estado terminal (sombreado).
- $\gamma = 1$ : MDP episódico não descontado.

[0,0]	[0,1]	[0,2]	[0,3]
[1,0]	[1,1]	[1,2]	[1,3]
[2,0]	[2,1]	[2,2]	[2,3]
[3,0]	[3,1]	[3,2]	[3,3]

Agente Aleatório:

$$\pi(\uparrow | s) = \pi(\downarrow | s) = \pi(\rightarrow | s) = \pi(\leftarrow | s) = 0.25, \forall s \in \mathcal{S}$$

Programação Dinâmica (DP)

Iteração sobre Função Política (Policy Iteration)



# DP: POLICY ITERATION

Vimos como utilizar Programação Dinâmica para avaliar uma política (obter  $V_{\pi}(s)$ ).

Como descobrir a política ótima  $V^*$ ?

- Ideia: Encontrar uma política  $\pi'$  melhor que  $\pi$  a partir de  $V_{\pi}$ .
- Vamos partir de uma política  $\pi$  qualquer e melhorá-la iterativamente até encontrar  $\pi^*$ :

$$\pi_0, V_{\pi_0}, \pi_1, V_{\pi_1}, \pi_2, V_{\pi_2} \dots, \pi^*, V^*$$

- Métodos de DP que otimizam a política de ações desta forma são denominados *Policy Iteration*.

# DP: POLICY ITERATION

Como melhorar uma Política de Ações?

Métodos de Iteração sobre Função Política (Policy Iteration) consistem em repetidamente avaliar a função valor  $V_\pi$  da política atual e melhorar esta política a partir de  $V_\pi$ :

- Avaliação de Política  $\pi$  (Policy Evaluation)

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

- Melhorar  $\pi$  por meio de comportamento *greedy* com relação a  $V_\pi$  (Policy Improvement)

$$\pi' = \text{greedy}(V_\pi)$$

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

Esse processo iterativo sempre converge para a política ótima  $\pi^*$ .

# COMPORTAMENTO GREEDY

O comportamento *greedy* (guloso) com relação a uma Função Valor  $V_\pi$  consiste em sempre tomar ações que levam a estados de maior valor:

$$\pi'(a|s) = \begin{cases} 1, & \text{se } a = \operatorname{argmax}_{a \in \mathcal{A}} \left[ \overbrace{\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_\pi(s')}^{Q_\pi(s, a)} \right] \\ 0, & \text{caso contrário} \end{cases}$$

Ou seja, o agente toma sempre as ações que acredita serem as melhores dado seu conhecimento atual  $V_\pi$  (Exploitation).

# COMO O COMPORTAMENTO GREEDY MELHORA A POLÍTICA $\pi$

- Considere a política determinística  $\pi(s) = a$
- Seja  $\pi'$  a política obtida pelo comportamento greedy sobre  $V_\pi$ :

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q_\pi(s, a)$$

- Temos que:

$$Q_\pi(s, a = \pi'(s)) = \max_{a \in \mathcal{A}} Q_\pi(s, a) \geq Q_\pi(s, a = \pi(s)) = V_\pi(s)$$

- Assim, a Função Valor necessariamente melhora  $V_{\pi'}(s) \geq V_\pi(s)$ :

$$\begin{aligned} V_\pi(s) &\leq Q_\pi(s, \pi'(s)) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma Q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 Q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s] \\ &= V_{\pi'}(s) \end{aligned}$$

# CONVERGÊNCIA DA ITERAÇÃO SOBRE POLÍTICA

Quando há convergência o sinal de desigualdade da equação anterior é substituído por um sinal de igualdade:

$$Q_{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} Q_{\pi}(s, a) = Q_{\pi}(s, \pi(s)) = V_{\pi}(s)$$

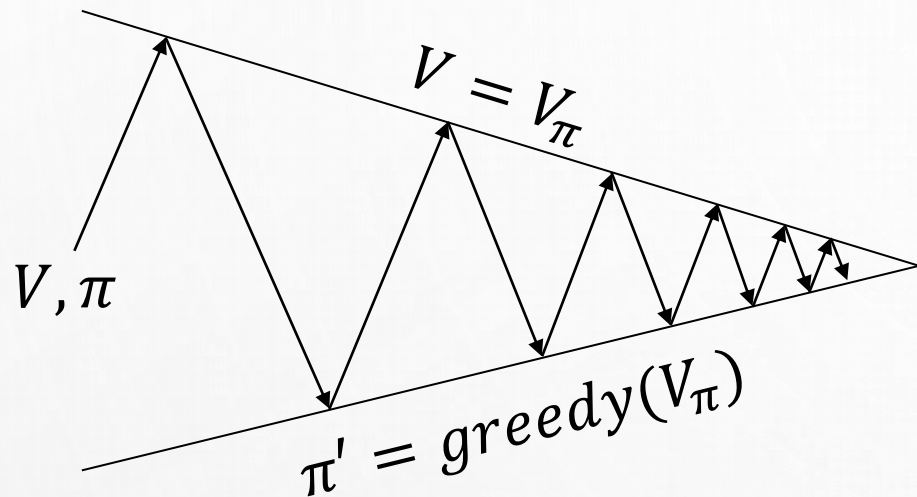
Então a Equação de Bellman da Otimalidade é satisfeita:

$$V_{\pi}(s) = \max_{a \in \mathcal{A}} Q_{\pi}(s, a)$$

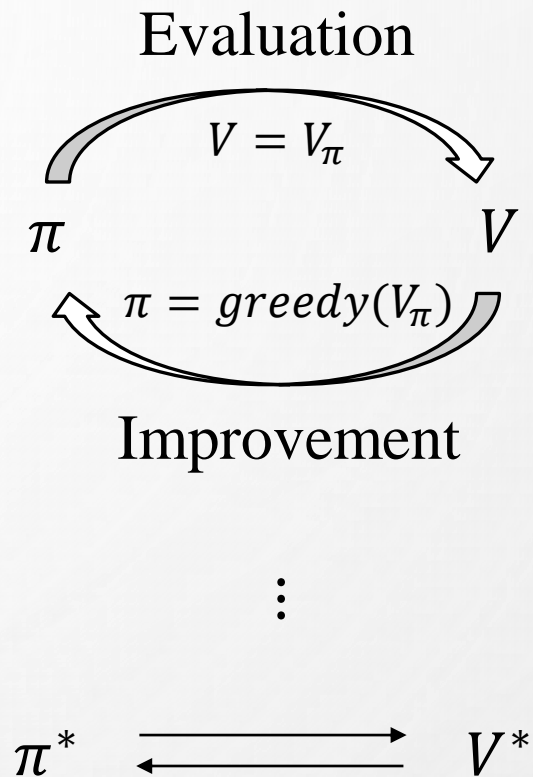
Portanto,  $V_{\pi}(s) = V^*(s), \forall s \in \mathcal{S}$  e a política  $\pi$  é uma política ótima:

$$\pi = \pi^*$$

# DP: POLICY ITERATION



- **Policy Evaluation:** Estimar  $V_\pi$  utilizando o algoritmo de Avaliação Iterativa da Política  $\pi$ .
- **Policy Improvement:** Obter  $\pi' \geq \pi$  por meio de comportamento *greedy* sobre  $V_\pi$ .





# DP: POLICY ITERATION

## Algoritmo: Iteração sobre Política $\pi$ para obter $\pi^*$ , $V^*$ (Atualizações Síncronas)

**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ .

**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

**Inicializar:**  $V_0(s)$  e  $\pi(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

### 1) Policy Evaluation

$k = 0$

Repetir:

$\Delta = 0$

Repetir para cada  $s \in \mathcal{S}$ :

$$V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

$$\Delta \leftarrow \max\{\Delta, |V_{k+1}(s) - V_k(s)|\}$$

$k \leftarrow k + 1$

Até que  $\Delta < \theta$

### 2) Policy Improvement

$policy\_stable \leftarrow True$

Repetir para cada  $s \in \mathcal{S}$ :

$a_{old} \leftarrow \pi(s)$

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} [\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_{k+1}(s')]$$

Se  $a_{old} \neq \pi(s)$ , então  $policy\_stable \leftarrow False$

Se  $policy\_stable$ , **Retorna**  $\pi \approx \pi^*$ ,  $V_{k+1} \approx V^*$ , caso contrário volta para 1)

# DP: POLICY ITERATION - EXEMPLO

## 1) Policy Evaluation

$V_\pi$  para política aleatória

$V_\infty = V_\pi$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

## 2) Policy Improvement

Política greedy com relação a  $V_\pi$

$\pi' = greedy(V_\pi)$

	←	←	↖
↑	↖	↖	↓
↑	↗	↘	↓
↗	→	→	

Aplicado ao exemplo do *gridworld* o algoritmo de Iteração sobre Política convergiu para a política ótima em apenas uma iteração:

$$\pi' = greedy(V_{\pi_{rand}}) = \pi^*$$

No caso geral podem ser necessárias iterações adicionais.

# DP: POLICY ITERATION - EXEMPLO

## DP\_Methods.ipynb

The screenshot displays a Jupyter Notebook interface with the file 'DP\_Methods.ipynb' open. The left sidebar shows a 'Table of contents' with the following items:

- Dynamic Programming: Policy Evaluation, Policy Iteration, Value Iteration
- Imports
- Funções Auxiliares
- Environment Class
  - Create Environment
  - Função get\_state\_index(s)
- Random Policy
- DP: Iterative Policy Evaluation
- Plot Value Function
- DP: Policy Iteration** (highlighted)
  - greedy\_improvement(env,V)
  - check\_policy\_stable(old\_policy, new\_policy):
  - PolicyIteration(env, theta)
- DP: Value Iteration
- Section

The main content area shows the 'DP: Policy Iteration' section, which includes the following text:

Vamos implementar uma função que implementa o algoritmo de Iteração sobre a Função Política, retornando a função Valor ótima  $V^*(s)$  e política ótima  $\pi^*$ .

Para isso, vamos criar uma função auxiliar de melhoria de política com base em comportamento greedy sobre  $V_\pi$  e chamar iterativamente as funções de avaliação e melhoria.

**DP: POLICY ITERATION**

**Algoritmo:** Iteração sobre Política  $\pi$  para obter  $\pi^*$ ,  $V^*$  (Atualizações Síncronas)

**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ .

**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

**Inicializar:**  $V_0(s)$  e  $\pi(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

**1) Policy Evaluation**

$k = 0$

Repetir:

$\Delta = 0$

Repetir para cada  $s \in \mathcal{S}$ :

$$V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right)$$

$$\Delta \leftarrow \max(\Delta, |V_{k+1}(s) - V_k(s)|)$$

$k \leftarrow k + 1$

Até que  $\Delta < \theta$

**2) Policy Improvement**

$policy\_stable \leftarrow True$

Repetir para cada  $s \in \mathcal{S}$ :

$$a_{old} \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \text{argmax}_a \left[ \mathcal{P}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_{k+1}(s') \right]$$

# DP: POLICY ITERATION - EXEMPLO

Vamos analisar a evolução da política *greedy* a cada iteração do algoritmo Policy Evaluation:

$V_k$  para política aleatória

$$V_0$$

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Política greedy com relação a  $V_k$

$\pi' = greedy(V_0)$

	↕	↕	↕
↕	↕	↕	↕
↕	↕	↕	↕
↕	↕	↕	

$$V_1$$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$\pi' = greedy(V_1)$

	←	↕	↕
↑	↕	↕	↕
↕	↕	↕	↓
↕	↕	→	

# DP: POLICY ITERATION - EXEMPLO

$V_k$  para política aleatória

$V_2$

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0

$V_3$

0	-2.44	-2.94	-3
-2.44	-2.87	-3	-2.94
-2.94	-3	-2.87	-2.44
-3	-2.94	-2.44	0

$V_{10}$

0	-6.14	-8.35	-8.97
-6.14	-7.74	-8.43	-8.35
-8.35	-8.43	-7.74	-6.14
-8.97	-8.35	-6.14	0

Política greedy com relação a  $V_k$

$\pi' = greedy(V_2)$

	←	←	↕
↑	↖	↕	↓
↑	↕	↘	↓
↕	→	→	

$\pi' = greedy(V_3)$

	←	←	↖
↑	↖	↖	↓
↑	↗	↘	↓
↖	→	→	

$\pi' = greedy(V_{10})$

	←	←	↖
↑	↖	↖	↓
↑	↗	↘	↓
↖	→	→	

Política Ótima é  
obtida a partir de  
 $k = 3$

↓  $\pi^*$

# DP: EXTENSÕES DE POLICY ITERATION

Precisamos executar o algoritmo de Avaliação da Função Política até a convergência para  $V_\pi$  no passo de Policy Evaluation?

Em vez de calcular  $V_0 \rightarrow V_1 \rightarrow \dots \rightarrow V_\infty$  para somente depois melhorar a política  $\pi$ , podemos definir um critério de parada:

- $\epsilon$ -convergência da Função Valor:  $|V_{k+1}(s) - V_k(s)| < \epsilon, \forall s \in \mathcal{S}$
- Parada após  $k$  iterações do algoritmo de Avaliação Iterativa da Política.

Se atualizarmos a política com  $\pi' = \text{greedy}(V_\pi)$  a cada iteração (parar após  $k = 1$ ), temos o Algoritmo de Iteração sobre Função Valor.



# DP: VALUE ITERATION

Programação Dinâmica (DP)

Iteração sobre Função Valor (Value Iteration)

# PRINCÍPIO DA OTIMALIDADE DE BELLMAN

O Princípio da Otimalidade de Bellman afirma que uma política  $\pi(a|s)$  atinge o valor ótimo a partir do estado  $s$ ,  $V_\pi(s) = V^*(s)$ , se e somente se:

- Para cada estado  $s'$  alcançável de  $s$ :
- $\pi$  atinge o valor ótimo a partir de  $s'$ , ou seja,  $V_\pi(s') = V^*(s')$

Assim, uma política ótima pode ser dividida em dois componentes:

- Uma ação inicial ótima  $A^*$
- Seguida de uma política ótima a partir do estado sucessor  $S'$

# DP: VALUE ITERATION

Devido ao Princípio da Otimalidade de Bellman, se conhecemos a solução dos subproblemas  $V^*(s')$ , podemos determinar a solução  $V^*(s)$  a partir de *one-step look ahead*:

$$V^*(s) \leftarrow \max_{a \in \mathcal{A}} \left[ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V^*(s') \right]$$

- A ideia por trás do algoritmo de Iteração sobre a Função Valor é aplicar a atualização acima recursivamente, de modo que as estimativas da função valor tendem à Função Valor Ótima:  $V_0 \rightarrow V_1 \rightarrow V_2 \rightarrow \dots \rightarrow V^*$

# DP: VALUE ITERATION

A Iteração sobre Função Valor (Value Iteration) consiste em obter funções valores sucessivas que se aproximam da Função Valor Ótima  $V^*$ :

$$V_0 \rightarrow V_1 \rightarrow \dots \rightarrow V^*$$

Isso pode ser feito com Atualizações Síncronas (Synchronous Backups) a partir da aplicação recursiva da Equação de Bellman da Otimalidade :

- Inicializa-se  $V_0$  aleatoriamente.
- Repetimos para  $k = 0, 1, \dots$  :
  - Para todos estados  $s \in \mathcal{S}$ :
  - Obtemos  $V_{k+1}(s)$  a partir de  $V_k(s')$ , onde  $s'$  é um estado sucessor de  $s$ .

Não há uma política explícita, a cada iteração a Função Valor  $V_k$  pode não corresponder à nenhuma política  $\pi$ .

# DP: VALUE ITERATION

Atualização de  $V_k$  para obter  $V_{k+1}$ :

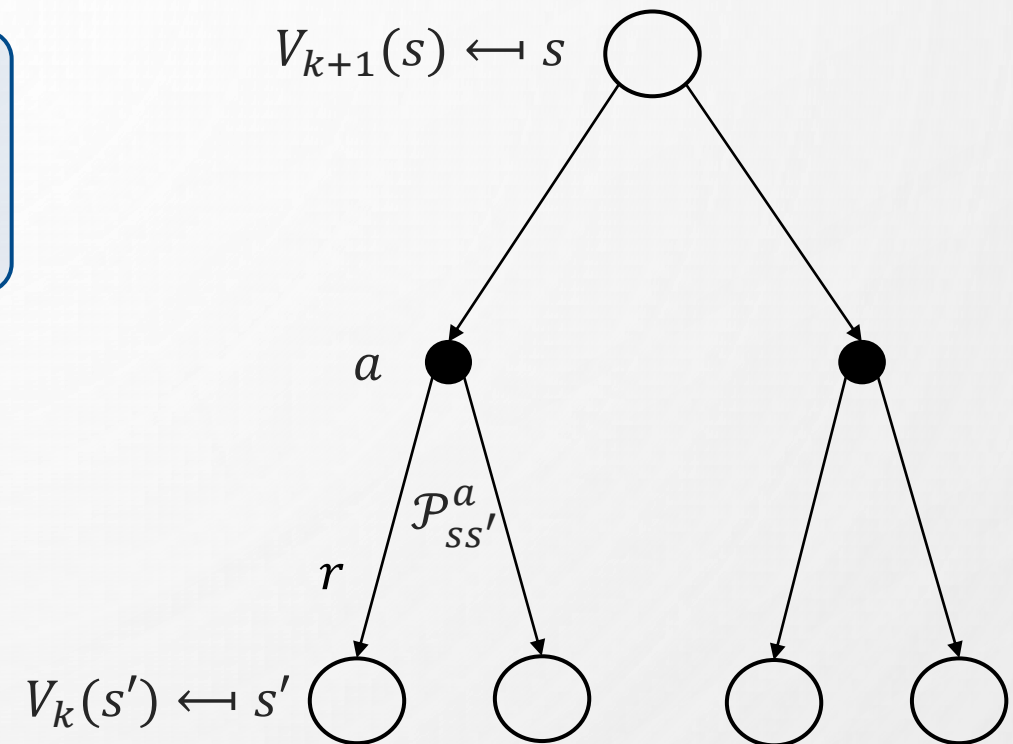
$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left[ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s') \right]$$

Em forma matricial:  $\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} [\mathbf{R}^a + \gamma \mathbf{P}^a \mathbf{v}_k]$

Percebe-se que, após convergência, a equação de Bellman da Otimalidade é satisfeita:

$$V_{k+1}(s) = V_k(s), \forall s \in \mathcal{S} \Rightarrow V_{k+1}(s) = V^*(s)$$

onde  $V^*(s) = \max_{a \in \mathcal{A}} [\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V^*(s')]$



# DP: VALUE ITERATION

**Algoritmo:** Iteração sobre Função Valor para obtenção de  $\pi^*, V^*$  (Atualizações Síncronas)

**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ .

**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

**Inicializar**  $V_0(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

$k = 0$

Repetir:

$\Delta = 0$

Repetir para cada  $s \in \mathcal{S}$ :

$$V_{k+1} \leftarrow (s) \max_{a \in \mathcal{A}} [\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s')]$$

$$\Delta \leftarrow \max\{\Delta; |V_{k+1}(s) - V_k(s)|\}$$

$k \leftarrow k + 1$

Até que  $\Delta < \theta$

**Retorna:**  $\pi = \operatorname{argmax}_{a \in \mathcal{A}} [\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_{k+1}(s')] \approx \pi^*$  e  $V_{k+1} \approx V^*$



# DP: VALUE ITERATION - EXEMPLO

## DP\_Methods.ipynb

DP\_Methods.ipynb ☆

File Edit View Insert Runtime Tools Help [All changes saved](#)

Comment Share

RAM  Disk

Editing

Table of contents

- Dynamic Programmig: Policy Evaluation, Policy Iteration, Value Iteration
- Imports
- Funções Auxiliares
- Environment Class
  - Create Environment
  - Função get\_state\_index(s)
- Random Policy
- DP: Iterative Policy Evaluation
- Plot Value Function
- DP: Policy Iteration
  - greedy\_improvement(env,V)
  - check\_policy\_stable(old\_policy, new\_policy):
  - PolicyIteration(env, theta)
- DP: Value Iteration**

+ Code + Text

DP: Value Iteration

Vamos agora obter a função valor ótima  $V^*$  pelo algoritmo de iteração sobre a função valor.

DP: VALUE ITERATION
35

**Algoritmo:** Iteração sobre Função Valor para obtenção de  $\pi^*, V^*$  (Atualizações Síncronas)

**Input:** MDP  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ .

**Parâmetro do Algoritmo:**  $\theta > 0$  para determinar precisão da estimativa.

**Inicializar**  $V_0(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

$k = 0$

Repetir:

$\Delta = 0$

Repetir para cada  $s \in \mathcal{S}$ :

$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} [\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_k(s')]$

$\Delta \leftarrow \max\{\Delta, |V_{k+1}(s) - V_k(s)|\}$

$k \leftarrow k + 1$

Até que  $\Delta < \theta$

**Retorna:**  $\pi = \operatorname{argmax}_{a \in \mathcal{A}} [\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_{k+1}(s')] \approx \pi^*$  e  $V_{k+1} \approx V^*$

# DYNAMIC PROGRAMMING: VISÃO GERAL DOS ALGORITMOS

Os Algoritmos de Programação Dinâmica estudados são caracterizados pelo problema que resolvem (Previsão ou Controle) e pela Equação de Bellman aplicada para atualizações sucessivas:

Problema	Equação de Bellman	Algoritmo
Previsão (determinar $V_\pi$ )	Bellman Expectation Equation	Policy Evaluation
Controle (determinar $\pi^*, V^*$ )	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Controle (determinar $\pi^*, V^*$ )	Bellman Optimality Equation	Value Iteration

- Algoritmos são baseados na Função Valor dos Estados ( $V_\pi$  ou  $V^*$ ).
- Complexidade  $\mathcal{O}(mn^2)$ , onde  $|\mathcal{S}| = n$  e  $|\mathcal{A}| = m$ .

# DP: ATUALIZAÇÕES ASSÍNCRONAS

Os algoritmos vistos até agora implementam **Atualizações Síncronas**:

- A cada iteração, armazena-se a Função Valor antiga  $V_k$  e a nova  $V_{k+1}$ , todos os estados são percorridos e atualiza-se  $V_{k+1}(s)$  a partir de  $V_k(s')$ .
- Estados são atualizados em paralelo.

Programação Dinâmica com **Atualizações Assíncronas** possui as seguintes características:

- Estados são atualizados individualmente e em série, em qualquer ordem.
- A cada iteração, armazena-se apenas uma Função Valor  $V$  e atualiza-se o valor  $V(s)$  de um estado apenas.
- Tempo computacional consideravelmente reduzido.

# DP: ATUALIZAÇÕES ASSÍNCRONAS

Iteração sobre Função Valor com  
Atualizações Síncronas

Para todo  $s \in \mathcal{S}$ :

$$V_{new}(s) \leftarrow \max_{a \in \mathcal{A}} \left[ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V_{old}(s') \right]$$

$$V_{old} \leftarrow V_{new}$$

Iteração sobre Função Valor com  
Atualizações Assíncronas

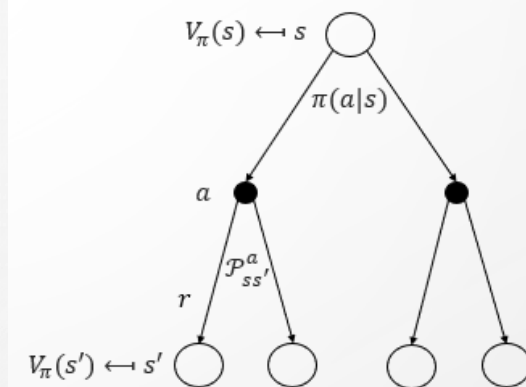
Para todo  $s \in \mathcal{S}$ :

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \left[ \mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a V(s') \right]$$

# DP: FULL-WIDTH BACKUPS

Os Algoritmos de Programação Dinâmica implementam Atualizações de Largura Completa (**Full-Width Backups**):

- Para cada atualização (síncrona ou assíncrona), todo estado sucessor e todas ações são consideradas.
- Isso é possível devido ao conhecimento do modelo  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  (Model-Based RL).



Programação Dinâmica é eficaz para problemas de tamanho médio (~Milhões de Estados).

Para problemas maiores, DP torna-se inviável (**Curse of Dimensionality**).

Outra limitação é a necessidade de conhecimento do modelo, a seguir vamos estudar algoritmos de aprendizado que se baseiam em **experiência amostrada (Model-Free)**.

# SAMPLE BACKUPS

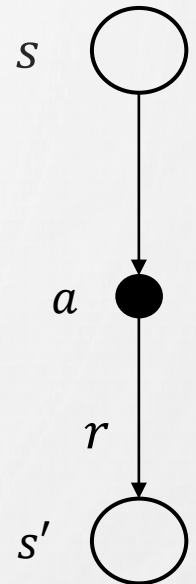
Em seguida, vamos estudar métodos que utilizam Atualizações Amostradas (Sample Backups) e não dependem do conhecimento de um modelo (Model-Free).

- Esses métodos utilizam transições e recompensas  $(S, A, R, S')$  amostradas a partir de interação com o ambiente, sem o conhecimento explícito do modelo

$$M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$$

- Vantagens:
  - Model-Free: Nenhum conhecimento do MDP é necessário.
  - Viável para espaços de estados maiores.

Torna-se necessário garantir exploração adequada do espaço de estados:  $\epsilon$ -greedy





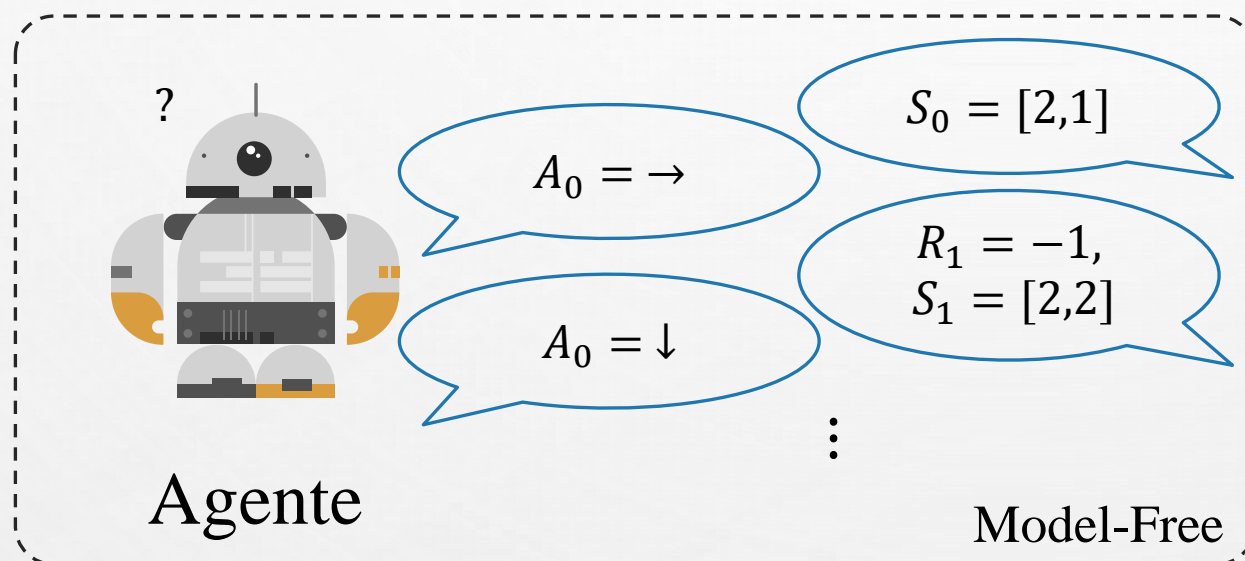
# MODEL-FREE PREDICTION

Model-Free Prediction

# MODEL-FREE PREDICTION: INTRODUÇÃO

Métodos de Programação Dinâmica possuem interesse teórico, mas em aplicações práticas o modelo do ambiente é desconhecido ou complexo demais.

- Model-Free Prediction: Como avaliar uma política de ações  $\pi$  exclusivamente a partir de interações com o ambiente?



[0,0]	[0,1]	[0,2]	[0,3]
[1,0]	[1,1]	[1,2]	[1,3]
[2,0]	[2,1]	[2,2]	[2,3]
[3,0]	[3,1]	[3,2]	[3,3]

Ambiente

# MODEL-FREE PREDICTION: MÉTODOS DE MONTE-CARLO

**Métodos de Monte-Carlo** aprendem diretamente a partir de **episódios completos** de experiência do agente.

Ideia: O valor de um estado é dado pelo valor esperado dos retornos daquele estado.

$$V_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t | S_t = s] = \mathbb{E}_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]$$

- Aproximar o valor esperado por uma média empírica dos retornos obtidos naquele estado em todos episódios.
- Quanto maior o número de episódios, mais próximo esse valor é do valor real  $V_{\pi}(s)$ .

Desvantagem: Aplicável somente a problemas episódicos (horizonte finito T).

# FIRST-VISIT MONTE-CARLO POLICY EVALUATION

Seja  $\pi$  uma política a ser avaliada,  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  um MDP desconhecido e

$\{(S_0^{ep}, A_0^{ep}, R_1^{ep}, S_1^{ep}, A_1^{ep}, R_2^{ep}, \dots)\}_{ep=1}^{N_{eps}} \sim \pi$  um conjunto de episódios de experiência obtidos ao seguir a política  $\pi$  em  $M$ .

- Para avaliar  $V_\pi(s)$ :
- **No primeiro timestep  $t$**  em que  $s$  é visitado em cada episódio:
- Incrementar contador  $N(s) \leftarrow N(s) + 1$
- Incrementar Retorno Total  $S_G(s) \leftarrow S_G(s) + G_t$
- Estimar valor do estado  $s$  como:  $V_\pi(s) \approx \frac{S_G(s)}{N(s)} \rightarrow V_\pi(s), N(s) \rightarrow \infty$

# FIRST-VISIT MONTE-CARLO POLICY EVALUATION

**Algoritmo:** **First-Visit** Monte-Carlo Policy Evaluation for estimating  $V_\pi$

**Input:** Política  $\pi$  e conjunto de episódios  $\{(S_0^{ep}, A_0^{ep}, R_1^{ep}, \dots, S_{T-1}, A_{T-1}, R_T)\}_{ep=1}^{N_{eps}} \sim \pi$ .

**Inicializar**  $N(s) = 0, S_G(s) = 0$  para todo  $s \in \mathcal{S}$ .

Repetir para  $ep = 1, \dots, N_{eps}$ :

$G = 0$

Repetir para cada timestep  $t = T - 1, T - 2, \dots, 0$  do episódio  $ep$ :

$G \leftarrow \gamma G + R_{t+1}^{ep}$

Se  $S_t^{ep}$  não aparece em  $(S_0^{ep}, \dots, S_{t-1}^{ep})$ :

$N(S_t^{ep}) \leftarrow N(S_t^{ep}) + 1$

$S_G(S_t^{ep}) \leftarrow S_G(S_t^{ep}) + G$

**Retorna:**  $V(s) = \frac{S_G(s)}{N(s)} \approx V_\pi(s), \forall s$

# FIRST-VISIT MONTE-CARLO POLICY EVALUATION: EXEMPLO

Seja o seguinte ambiente do tipo Gridworld 3x3, com 2 estados terminais ( $s_1$  e  $s_9$ ) e 7 estados não terminais ( $s_2, \dots, s_8$ ).

$s_1$	$s_2$	$s_3$
$s_4$	$s_5$	$s_6$
$s_7$	$s_8$	$s_9$

$$\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$$

Supondo que o objetivo de um agente neste ambiente é chegar a qualquer estado terminal o mais rápido possível. Podemos definir a seguinte função recompensa:

0	-1	-1
-1	-1	-1
-1	-1	0

$$\mathcal{R}(s, a) = \begin{cases} -1, & \forall a, \text{ se } s \in \{s_2, \dots, s_8\} \\ 0, & \forall a, \text{ se } s \in \{s_1, s_9\} \end{cases}$$



# FIRST-VISIT MONTE-CARLO POLICY EVALUATION: EXEMPLO

Vamos amostrar episódios neste ambiente de acordo com a política aleatória  $\pi$  e estimar a Função Valor dos Estados  $V_\pi$ .

$Ep_1: (s_4, \rightarrow, -1, s_5, \uparrow, -1, s_2, \rightarrow, -1, s_3, \downarrow, -1, s_6, \downarrow, -1, s_9)$

Aplicando Monte-Carlo por primeira visita para calcular  $S_G(s_i)$  para cada estado:

$$\begin{aligned} S_G(s_1) &= 0, & S_G(s_2) &= -2.71, & S_G(s_3) &= -1.9 \\ S_G(s_4) &= -4.68559, & S_G(s_5) &= -3.439, & S_G(s_6) &= -1, \\ S_G(s_7) &= 0, & S_G(s_8) &= 0, & S_G(s_9) &= 0 \end{aligned}$$

$s_1$	$s_2$	$s_3$
$s_4$	$s_5$	$s_6$
$s_7$	$s_8$	$s_9$

# FIRST-VISIT MONTE-CARLO POLICY EVALUATION: EXEMPLO

Seja  $\pi$  a política aleatória e o ambiente dado pelo gridworld 3x3, onde  $r(s, a) = -1$  para todo estado em branco e  $r(s, a) = 0$  para estados terminais, com  $\gamma = 0.9$ . Seja o seguinte conjunto de episódios:

- $Ep_1: (s_4, \rightarrow, -1, s_5, \uparrow, -1, s_2, \rightarrow, -1, s_3, \downarrow, -1, s_6, \downarrow, -1, s_9)$
- $Ep_2: (s_7, \rightarrow, -1, s_8, \rightarrow, -1, s_9)$

$s_1$	$s_2$	$s_3$
$s_4$	$s_5$	$s_6$
$s_7$	$s_8$	$s_9$

Aplicando Monte-Carlo por primeira visita para calcular  $S_G(s_i)$  para cada estado:

$$\begin{aligned}
 S_G(s_1) &= 0, & S_G(s_2) &= -2.71, & S_G(s_3) &= -1.9 \\
 S_G(s_4) &= -4.68559, & S_G(s_5) &= -3.439, & S_G(s_6) &= -1, \\
 S_G(s_7) &= 0 + (-1.9), & S_G(s_8) &= 0 + (-1), & S_G(s_9) &= 0 + 0
 \end{aligned}$$

# FIRST-VISIT MONTE-CARLO POLICY EVALUATION: EXEMPLO

Seja  $\pi$  a política aleatória e o ambiente dado pelo gridworld 3x3, onde  $r(s, a) = -1$  para todo estado em branco e  $r(s, a) = 0$  para estados terminais, com  $\gamma = 0.9$ . Seja o seguinte conjunto de episódios:

- $Ep_1: (s_4, \rightarrow, -1, s_5, \uparrow, -1, s_2, \rightarrow, -1, s_3, \downarrow, -1, s_6, \downarrow, -1, s_9)$
- $Ep_2: (s_7, \rightarrow, -1, s_8, \rightarrow, -1, s_9)$
- $Ep_3: (s_2, \downarrow, -1, s_5, \leftarrow, -1, s_4, \leftarrow, -1, s_4, \uparrow, -1, s_1)$

$s_1$	$s_2$	$s_3$
$s_4$	$s_5$	$s_6$
$s_7$	$s_8$	$s_9$

Aplicando Monte-Carlo por primeira visita para calcular  $S_G(s_i)$  para cada estado:

$$\begin{array}{lll}
 S_G(s_1) = 0 + 0, & S_G(s_2) = -2.71 + (-3.439), & S_G(s_3) = -1.9 \\
 S_G(s_4) = -4.68559 + (-1.9), & S_G(s_5) = -3.439 + (-2.71), & S_G(s_6) = -1, \\
 S_G(s_7) = 0 + (-1.9), & S_G(s_8) = 0 + (-1), & S_G(s_9) = 0 + 0
 \end{array}$$

# FIRST-VISIT MONTE-CARLO POLICY EVALUATION: EXEMPLO

Seja  $\pi$  a política aleatória e o ambiente dado pelo gridworld 3x3, onde  $r(s, a) = -1$  para todo estado em branco e  $r(s, a) = 0$  para estados terminais, com  $\gamma = 0.9$ . Seja o seguinte conjunto de episódios:

- $Ep_1: (s_4, \rightarrow, -1, s_5, \uparrow, -1, s_2, \rightarrow, -1, s_3, \downarrow, -1, s_6, \downarrow, -1, s_9)$
- $Ep_2: (s_7, \rightarrow, -1, s_8, \rightarrow, -1, s_9)$
- $Ep_3: (s_2, \downarrow, -1, s_5, \leftarrow, -1, s_4, \leftarrow, -1, s_4, \uparrow, -1, s_1)$
- $Ep_4: (s_7, \uparrow, -1, s_4, \rightarrow, -1, s_5, \rightarrow, -1, s_6, \uparrow, -1, s_3, \leftarrow, -1, s_2, \leftarrow, -1, s_1)$

$s_1$	$s_2$	$s_3$
$s_4$	$s_5$	$s_6$
$s_7$	$s_8$	$s_9$

Aplicando Monte-Carlo por primeira visita para estimar  $V_\pi(s)$  temos:

$$S_G(s_1) = [0], S_G(s_2) = [(-2.71) + (-3.439) + (-1.9)], S_G(s_3) = [(-1.9) + (-1.9)]$$

$$S_G(s_4) = [(-4.68559) + (-1.9) + (-4.0951)], S_G(s_5) = [(-3.439) + (-2.71) + (-3.439)], S_G(s_6) = [(-1) + (-2.71)],$$

$$S_G(s_7) = [(-1.9) + (-4.68559)], S_G(s_8) = [(-1)], S_G(s_9) = [0 + 0]$$

# FIRST-VISIT MONTE-CARLO POLICY EVALUATION: EXEMPLO

Assim, dividindo os retornos totais  $S_G$  pelo números de primeiras visitas  $N$  em cada estado, temos:

$$V(s_1) = \frac{S_G(s_1)}{N(s_1)} = \frac{0}{1} = 0, V(s_2) = \frac{S_G(s_2)}{N(s_2)} = \frac{-7.149}{3}, V(s_3) = \frac{S_G(s_3)}{N(s_3)} = \frac{-3.8}{2}$$

$$V(s_4) = \frac{S_G(s_4)}{N(s_4)} = \frac{-10.68069}{3} = 0, V(s_5) = \frac{S_G(s_5)}{N(s_5)} = \frac{-9.588}{3}, V(s_6) = \frac{S_G(s_6)}{N(s_6)} = \frac{-3.71}{2}$$

$$V(s_7) = \frac{S_G(s_7)}{N(s_7)} = \frac{-6.58559}{2} = 0, V(s_8) = \frac{S_G(s_8)}{N(s_8)} = \frac{-1}{1}, V(s_9) = \frac{S_G(s_9)}{N(s_9)} = \frac{0}{2}$$

$s_1$	$s_2$	$s_3$
$s_4$	$s_5$	$s_6$
$s_7$	$s_8$	$s_9$

Os valores de cada estado são aproximadamente:

$$V \approx V_\pi$$

(para estimativas mais precisas é necessária uma quantidade maior de episódios)

0	-2.38	-1.9
-3.56	-3.20	-1.85
-3.29	-1	0

MonteCarlo\_PolicyEvaluation.ipynb
Comment Share

File Edit View Insert Runtime Tools Help Last saved at 9:03 PM

+ Code + Text
Connect Editing

## First-Visit Monte Carlo Policy Evaluation (Exemplo Gridworld)

Vamos implementar o algoritmo de avaliação iterativa de política para o exemplo gridworld 3x3 e comparar a função valor  $V_{\pi}(s)$  de política aleatória com a obtida nos slides a partir de apenas 4 episódios.

**FIRST-VISIT MONTE-CARLO POLICY EVALUATION: EXEMPLO**
40

Seja o seguinte ambiente do tipo Gridworld 3x3, com 2 estados terminais ( $s_1$  e  $s_9$ ) e 7 estados não terminais ( $s_2, \dots, s_8$ ).

$s_1$	$s_2$	$s_3$
$s_4$	$s_5$	$s_6$
$s_7$	$s_8$	$s_9$

Supondo que o objetivo de um agente neste ambiente é chegar a qualquer estado terminal o mais rápido possível. Podemos definir a seguinte função recompensa:

0	-1	-1
-1	-1	-1
-1	-1	0

$$\mathcal{R}(s, a) = \begin{cases} -1, & \forall a, se \ s \in \{s_2, \dots, s_8\} \\ 0, & \forall a, se \ s \in \{s_1, s_9\} \end{cases}$$

$s_1$	$s_2$	$s_3$
$s_4$	$s_5$	$s_6$
$s_7$	$s_8$	$s_9$

0	-1	-1
-1	-1	-1
-1	-1	0

$$\mathcal{R}(s, a) = \begin{cases} -1, & \forall a, s \in \{s_2, \dots, s_8\} \\ 0, & \forall a, s \in \{s_1, s_9\} \end{cases}$$



# FIRST-VISIT MONTE-CARLO POLICY EVALUATION: EXEMPLO

## Monte-Carlo (4 episódios)

$$V \approx V_{\pi}$$

0	-2.38	-1.9
-3.56	-3.20	-1.85
-3.29	-1	0

$$V(s_1) = \frac{S_G(s_1)}{N(s_1)} = \frac{0}{1} = 0, V(s_2) = \frac{S_G(s_2)}{N(s_2)} = \frac{-7.149}{3}, V(s_3) = \frac{S_G(s_3)}{N(s_3)} = \frac{-3.8}{2}$$

$$V(s_4) = \frac{S_G(s_4)}{N(s_4)} = \frac{-10.68069}{3} = 0, V(s_5) = \frac{S_G(s_5)}{N(s_5)} = \frac{-9.588}{3}, V(s_6) = \frac{S_G(s_6)}{N(s_6)} = \frac{-3.71}{2}$$

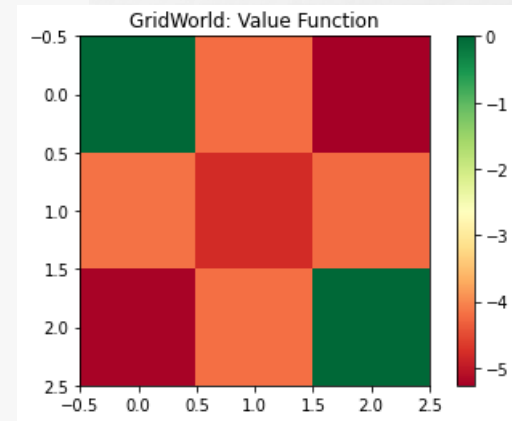
$$V(s_7) = \frac{S_G(s_7)}{N(s_7)} = \frac{-6.58559}{2} = 0, V(s_8) = \frac{S_G(s_8)}{N(s_8)} = \frac{-1}{1}, V(s_9) = \frac{S_G(s_9)}{N(s_9)} = \frac{0}{2}$$

## Monte-Carlo (10k episódios)

$$V \approx V_{\pi}$$

0	-4.19	-5.26
-4.16	-4.77	-4.22
-5.21	-4.18	0

```
def first_visit_monte_carlo_policy_evaluation(env, agent, N_eps=1000):
    n = len(env.state_space)
    # Initialize Sum of Returns S and State Visit Count N
    S = np.zeros(n)
    N = np.zeros(n)
    # Loop over episodes
    for i in range(N_eps):
        episode = generate_episode(env, random_agent)
        G = 0
        T = len(episode)
        # Loop over timesteps on episode
        for t in range(T-1, -1, -1):
            transition = episode[t]
            G = env.gamma*G + transition.r
            visited_states = [e.s for e in episode[0:t]]
            if (first_visit(visited_states, transition.s)):
                s_index = state_index(transition.s)
                N[s_index] += 1
                S[s_index] += G
        # Value Function Estimate V ~ S/N
        V = S/N
    return V
```



# EVERY-VISIT MONTE-CARLO POLICY EVALUATION

Seja  $\pi$  uma política a ser avaliada,  $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  um MDP desconhecido e

$\{(S_0^{ep}, A_0^{ep}, R_1^{ep}, S_1^{ep}, A_1^{ep}, R_2^{ep}, \dots)\}_{ep=1}^{N_{eps}} \sim \pi$  um conjunto de episódios de experiência obtidos ao seguir a política  $\pi$  em  $M$ .

- Para avaliar  $V_\pi(s)$ :
- **Em cada timestep  $t$**  em que  $s$  é visitado em cada episódio:
- Incrementar contador  $N(s) \leftarrow N(s) + 1$
- Incrementar Retorno Total  $S_G(s) \leftarrow S_G(s) + G_t$
- Estimar valor do estado  $s$  como:  $V_\pi(s) \approx \frac{S_G(s)}{N(s)} \rightarrow V_\pi(s), N(s) \rightarrow \infty$

# EVERY-VISIT MONTE-CARLO POLICY EVALUATION

**Algoritmo:** **Every-Visit** Monte-Carlo Policy Evaluation for estimating  $V_\pi$

**Input:** Política  $\pi$  e conjunto de episódios  $\{(S_0^{ep}, A_0^{ep}, R_1^{ep}, \dots, S_{T-1}, A_{T-1}, R_T)\}_{ep=1}^{N_{eps}} \sim \pi$ .

**Inicializar**  $N(s) = 0, S_G(s) = 0$  para todo  $s \in \mathcal{S}$ .

Repetir para  $ep = 1, \dots, N_{eps}$ :

$G = 0$

Repetir para cada timestep  $t = T - 1, T - 2, \dots, 0$  do episódio  $ep$ :

$G \leftarrow \gamma G + R_{t+1}^{ep}$

$N(S_t^{ep}) \leftarrow N(S_t^{ep}) + 1$

$S_G(S_t^{ep}) \leftarrow S_G(S_t^{ep}) + G$

**Retorna:**  $V(s) = \frac{S_G(s)}{N(s)} \approx V_\pi(s), \forall s$

# ATUALIZAÇÕES INCREMENTAIS DE MONTE-CARLO

Não é preciso realizar todos os episódios antes de estimar  $V_\pi$ , é possível realizar atualizações incrementais sobre a estimativa

$V \approx V_\pi$  a cada episódio  $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$ :

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} [G_t - V(S_t)]$$

Esse tipo de atualização permite a utilização de critérios de parada baseados na  $\epsilon$ -convergência da Função Valor:

$$|V_{k+1}(s) - V_k(s)| < \epsilon$$

A média de uma sequência pode ser calculada de forma incremental:

$$\begin{aligned} \mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1}) \end{aligned}$$

# MODEL-FREE PREDICTION: TEMPORAL-DIFFERENCE LEARNING $TD(0)$

Model-Free Prediction

Temporal-Difference Learning  $TD(0)$

# TEMPORAL-DIFFERENCE LEARNING

Assim como métodos de Monte-Carlo, métodos de aprendizado por diferenças temporais (Temporal-Difference Learning) utilizam exclusivamente interações com o ambiente, sem conhecimento do MDP (**Model-Free**).

- No entanto, métodos de Diferenças Temporais aprendem através de **episódios incompletos**, por meio de *Bootstrapping*.
- Em Temporal-Difference Learning, não é necessário esperar o fim do episódio para calcular os retornos  $G_t$ . Em vez disso, **estima-se os retornos** a partir do conhecimento atual da Função Valor  $V$ .



# COMPARAÇÃO: MONTE-CARLO E TEMPORAL-DIFFERENCE LEARNING

## Incremental Every-Visit Monte-Carlo Policy Evaluation

$V(S_t)$  é atualizado na direção do **retorno observado**  $G_t$ :

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Métodos de Monte-Carlo são menos eficientes, pois precisam esperar até o final do episódio para atualizar  $V(s)$ .

Apresentam convergência menos oscilatória.

## Temporal-Difference Learning TD(0)

$V(S_t)$  é atualizado na direção do **retorno estimado**  $R_{t+1} + \gamma V(S_{t+1})$ :

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

TD Target:  $R_{t+1} + \gamma V(S_{t+1})$

TD Error:  $\delta_t = [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

Métodos de aprendizado por diferenças temporais podem ser aplicados em problemas de horizonte infinito:  $t = 1, 2, 3, \dots \rightarrow \infty$

# BIAS VARIANCE TRADE-OFF

- O Retorno  $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$  é uma estimativa **sem *bias*** de  $V_\pi(S_t)$ .
- O TD target verdadeiro  $R_{t+1} + \gamma V_\pi(S_{t+1})$  é uma estimativa **sem *bias*** de  $V_\pi(S_t)$ .
- Como não conhecemos  $V_\pi$ , TD target  $R_{t+1} + \gamma V(S_{t+1})$  é uma estimativa **com *bias*** de  $V_\pi(S_t)$ .
- No entanto, o TD target  $R_{t+1} + \gamma V(S_{t+1})$  apresenta menor variância:
  - O retorno  $G_t$  depende de múltiplas ações, transições e recompensas aleatórias.
  - O TD target depende de apenas uma ação, transição e recompensa.

Monte-Carlo:  $\uparrow$  variância,  $\downarrow$  *bias*

TD-Learning:  $\downarrow$  variância,  $\uparrow$  *bias*

# TEMPORAL-DIFFERENCE LEARNING: TD(0) FOR POLICY EVALUATION

## **Algoritmo:** TD(0) Policy Evaluation for estimating $V_\pi$

**Input:** Política  $\pi$  a ser avaliada

**Parâmetro do Algoritmo:** Taxa de aprendizado  $\alpha \in (0,1]$ .

**Inicializar**  $V_0(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V_0(s_{term}) = 0$  para estados terminais.

Repetir para cada episódio

    Inicializar estado inicial  $S_0 \sim \mathbb{P}(S_0 = s), s \in \mathcal{S}$

    Repetir para cada timestep  $t = 0, 1, 2, \dots$ :

        Amostrar ação da política  $A_t \sim \pi(a|S_t)$

        Executar ação  $A_t$  e observar  $R_{t+1}, S_{t+1}$

$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

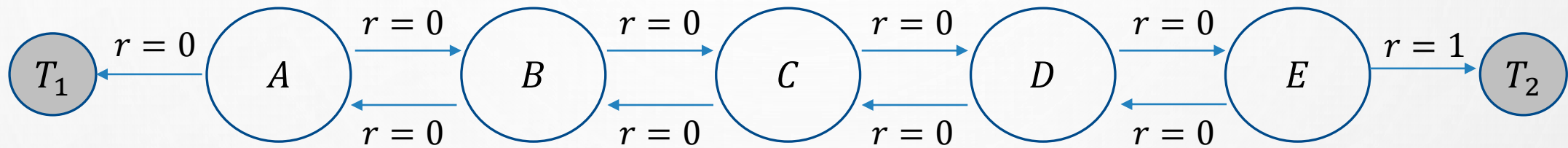
$S_t = S_{t+1}$

    Até que  $S_t$  seja um estado terminal

**Retorna:**  $V(s) \approx V_\pi(s), \forall s \in \mathcal{S}$

# RANDOM WALK EXAMPLE: MC, TD(0)

Seja o seguinte MDP, em que o agente começa no estado C e recebe recompensas  $\mathcal{R}(s, a) = 0, \forall s \in \{A, B, C, D, E\}, \forall a \in \{\leftarrow, \rightarrow\}$  com exceção de  $\mathcal{R}(E, \rightarrow) = 1$ .

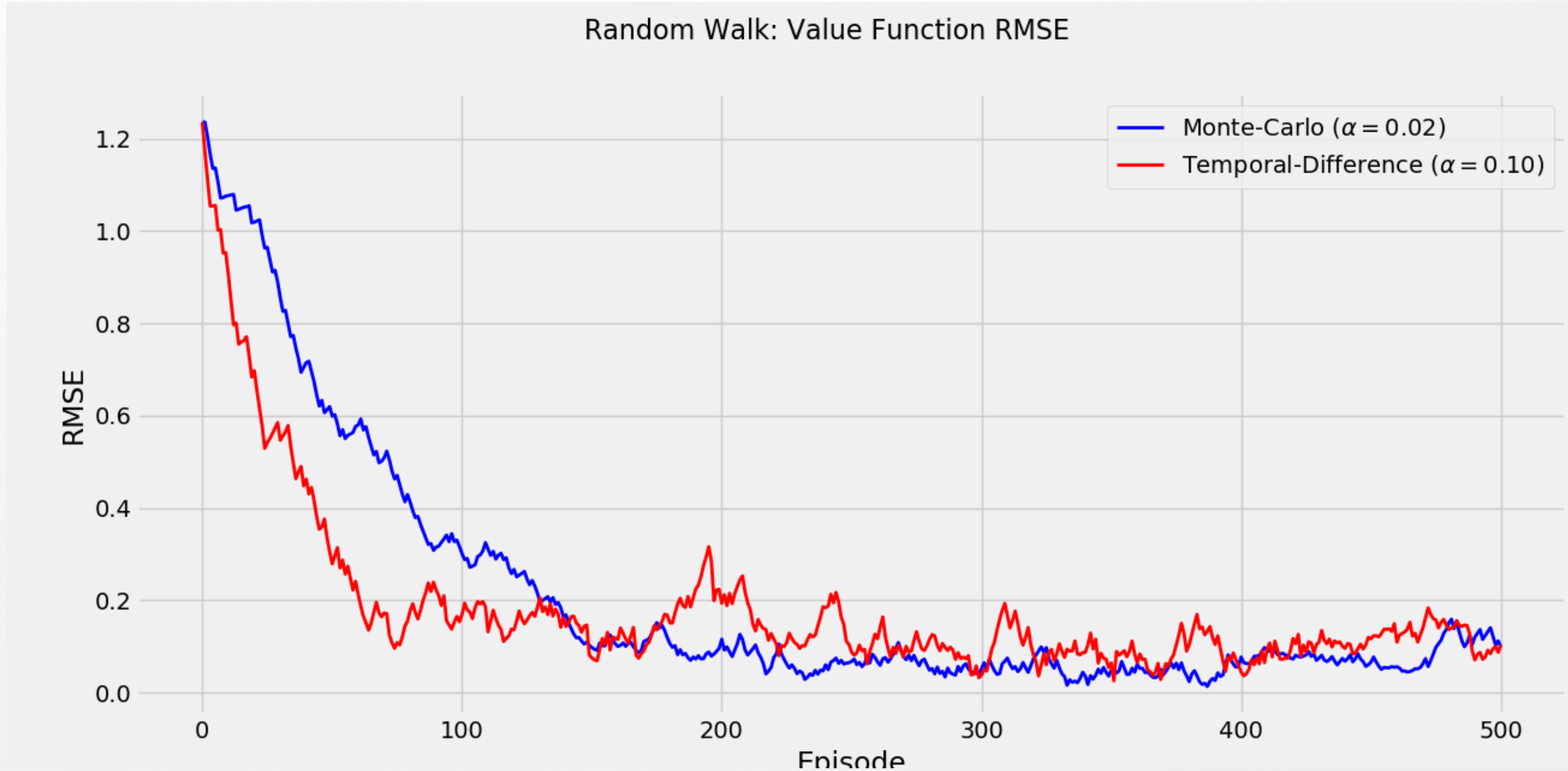


Como a única recompensa  $r = 1$  corresponde a chegar no estado  $T_2$ , temos que  $V_\pi(s)$  é a probabilidade de terminar no estado  $T_2$  dado que  $S_0 = s$

Qual é a função Valor dos Estados para uma política aleatória uniforme (para  $\gamma = 1$ )?

$$V_\pi(A) = 1/6, V_\pi(B) = 2/6, V_\pi(C) = 3/6, V_\pi(D) = 4/6, V_\pi(E) = 5/6$$

# RANDOM WALK EXAMPLE: MC, TD(0)



## MC\_TD0\_Comparison\_RandomWalk.ipynb

MonteCarlo\_TDO\_Comparison.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Table of contents

Imports

Environment Class

Create Environment

Random Policy

Transition Class

Generate Episode

Plot Episode

First-Visit Monte-Carlo Policy Evaluation

TD(0) Policy Evaluation

MC vs TD(0) Comparison

Section

RAM Disk

Editing

Monte-Carlo vs TD(0) Policy Evaluation (Random Walk Environment)

Tanto Monte-Carlo quanto o Método de Diferenças Temporais TD(0) podem ser utilizados na tarefa de Model-Free Prediction, que consiste em avaliar o comportamento de um agente em determinado MDP sem conhecimento do modelo do ambiente, somente através da experiência obtida pelo agente.

RANDOM WALK EXAMPLE: MC, TD(0)

Seja o seguinte MDP, em que o agente começa no estado C e recebe recompensas  $\mathcal{R}(s, a) = 0, \forall s \in \{A, B, C, D, E\}, \forall a \in \{\leftarrow, \rightarrow\}$  com exceção de  $\mathcal{R}(E, \rightarrow) = 1$ .

```
graph LR; T1((T1)) -- "r=0" --> A((A)); A -- "r=0" --> B((B)); B -- "r=0" --> C((C)); C -- "r=0" --> D((D)); D -- "r=0" --> E((E)); E -- "r=1" --> T2((T2)); A -- "r=0" --> B; B -- "r=0" --> A; C -- "r=0" --> D; D -- "r=0" --> C; E -- "r=0" --> D;
```

Como a única recompensa  $r = 1$  corresponde a chegar no estado  $T_2$ , temos que  $V_\pi(s)$  é a probabilidade de terminar no estado  $T_2$  dado que  $S_0 = s$

Qual é a função Valor dos Estados para uma política aleatória uniforme (para  $\gamma = 1$ )?

$V_\pi(A) = 1/6, V_\pi(B) = 2/6, V_\pi(C) = 3/6, V_\pi(D) = 4/6, V_\pi(E) = 5/6$

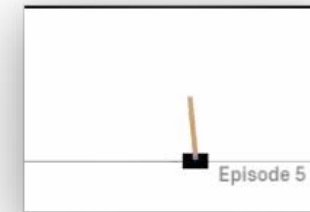


# OPEN AI GYM LIBRARY

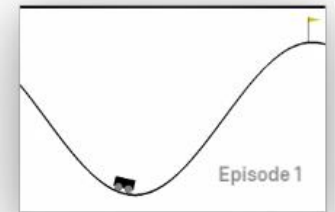
- gym: Biblioteca desenvolvida pela OpenAI para benchmarking de algoritmos de Aprendizado por Reforço em centenas de ambientes diferentes.



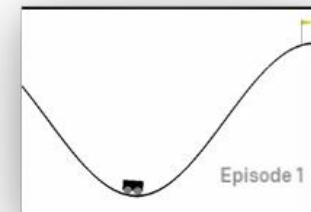
Acrobot-v1  
Swing up a two-link robot.



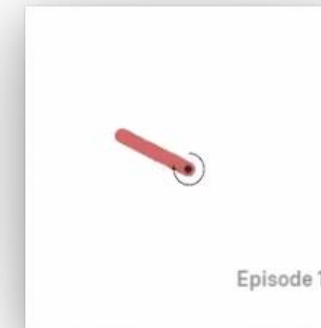
CartPole-v1  
Balance a pole on a cart.



MountainCar-v0  
Drive up a big hill.



MountainCarContinuous-v0  
Drive up a big hill with continuous control.



Pendulum-v0

fonte: <https://openai.com/blog/gym-retro/>

# ROULETTE EXAMPLE: MC

`gym_Roulette-v0.ipynb`

gym\_Roulette-v0.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 9:30 AM

Comment Share Settings L


RAM Disk Editing

Table of contents

- OpenAI gym
  - Imports
  - Create Environment
  - Test Environment
  - Agent Class
    - Create Random Agent
  - Transition Class
  - Generate Episode
  - Plot Episode
  - Monte-Carlo Policy Evaluation for  $Q(s,a)$
  - plot\_Q\_values(Q)
- Section

Code Text

Create Environment

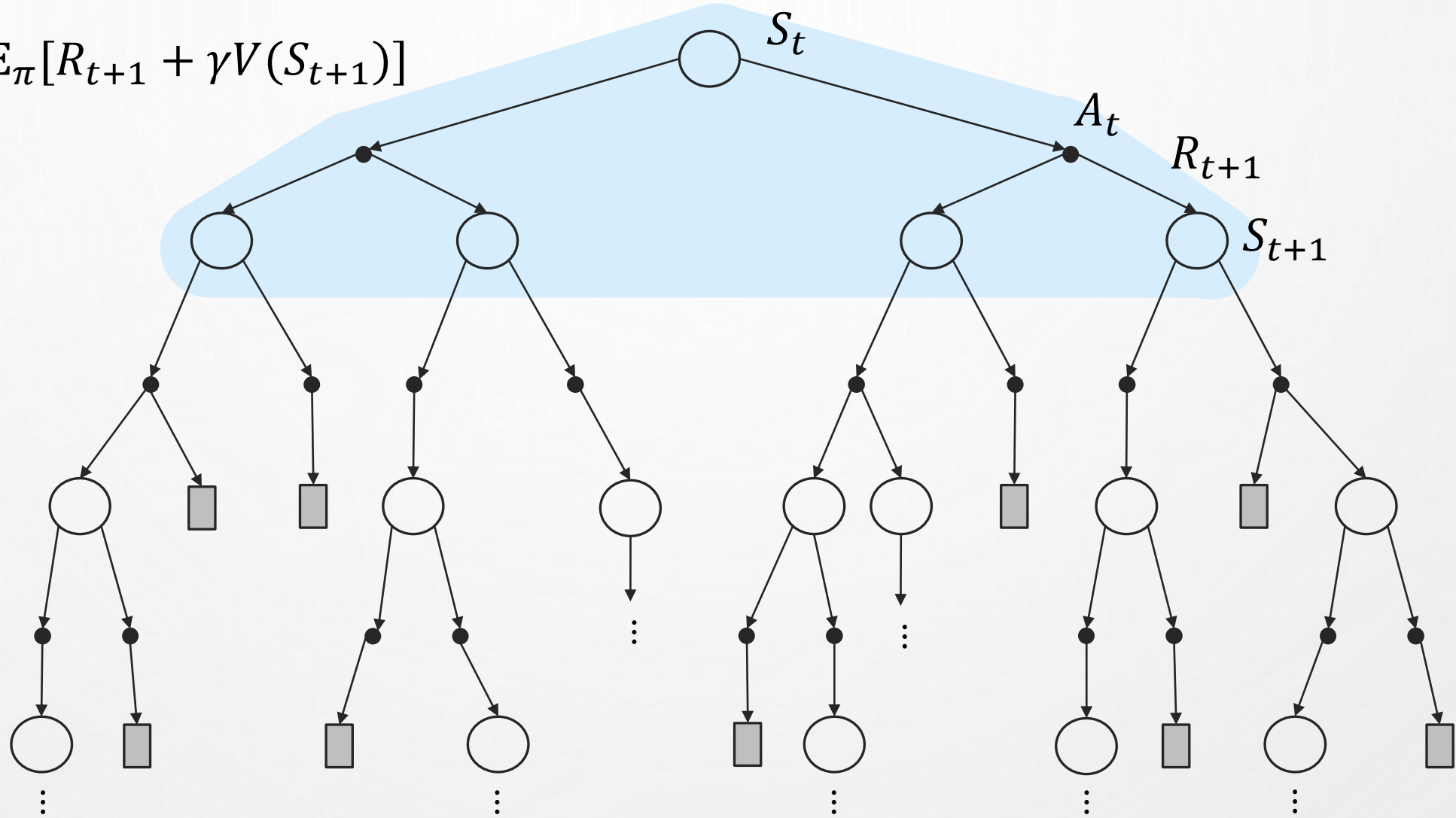


```
[74] env = gym.make('Roulette-v0')

print('Observation Space:')
print(env.observation_space)
print('Action Space:')
```

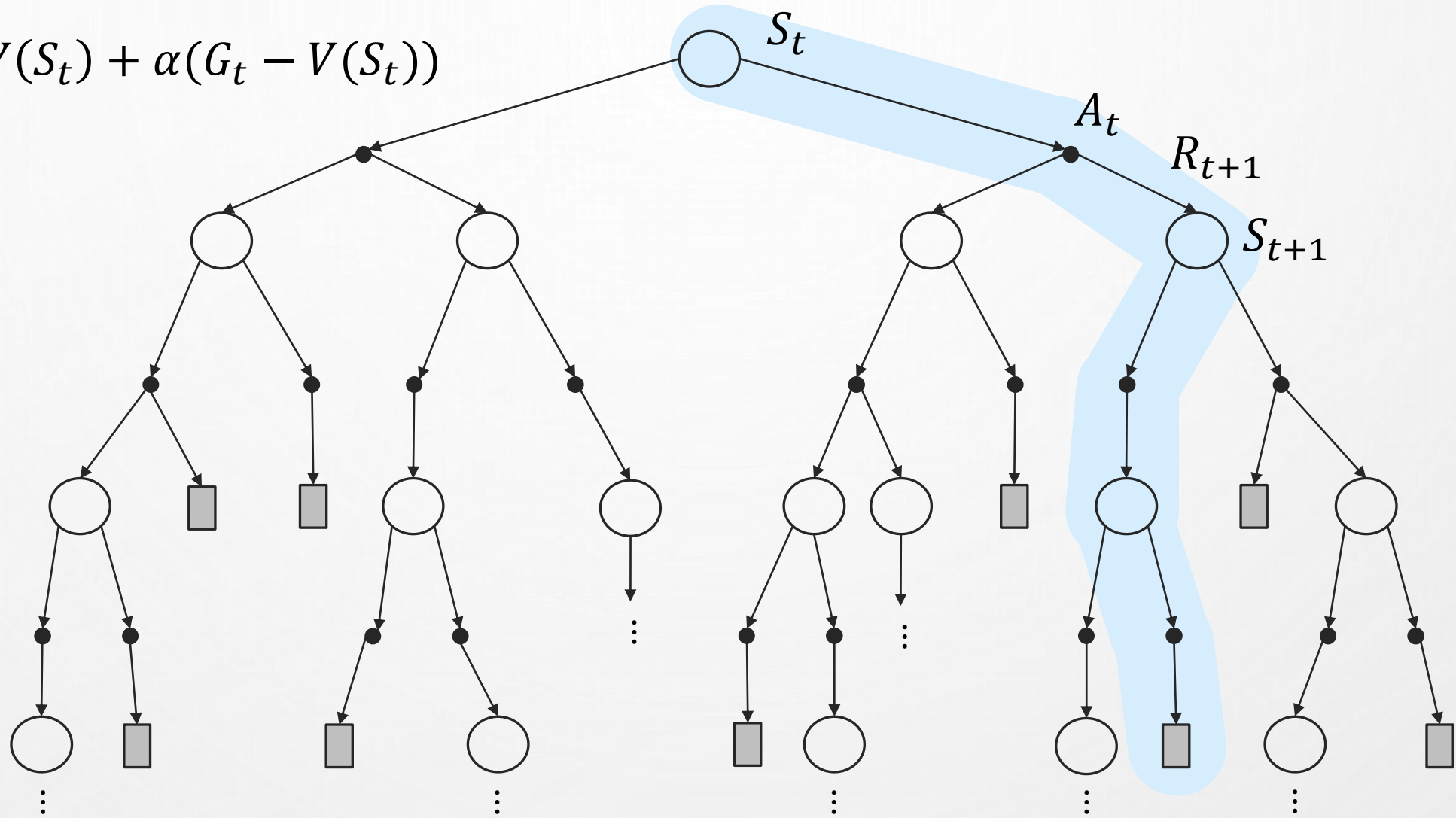
# ATUALIZAÇÕES DE PROGRAMAÇÃO DINÂMICA (DP)

$$V(S_t) \leftarrow \mathbb{E}_{\pi}[R_{t+1} + \gamma V(S_{t+1})]$$

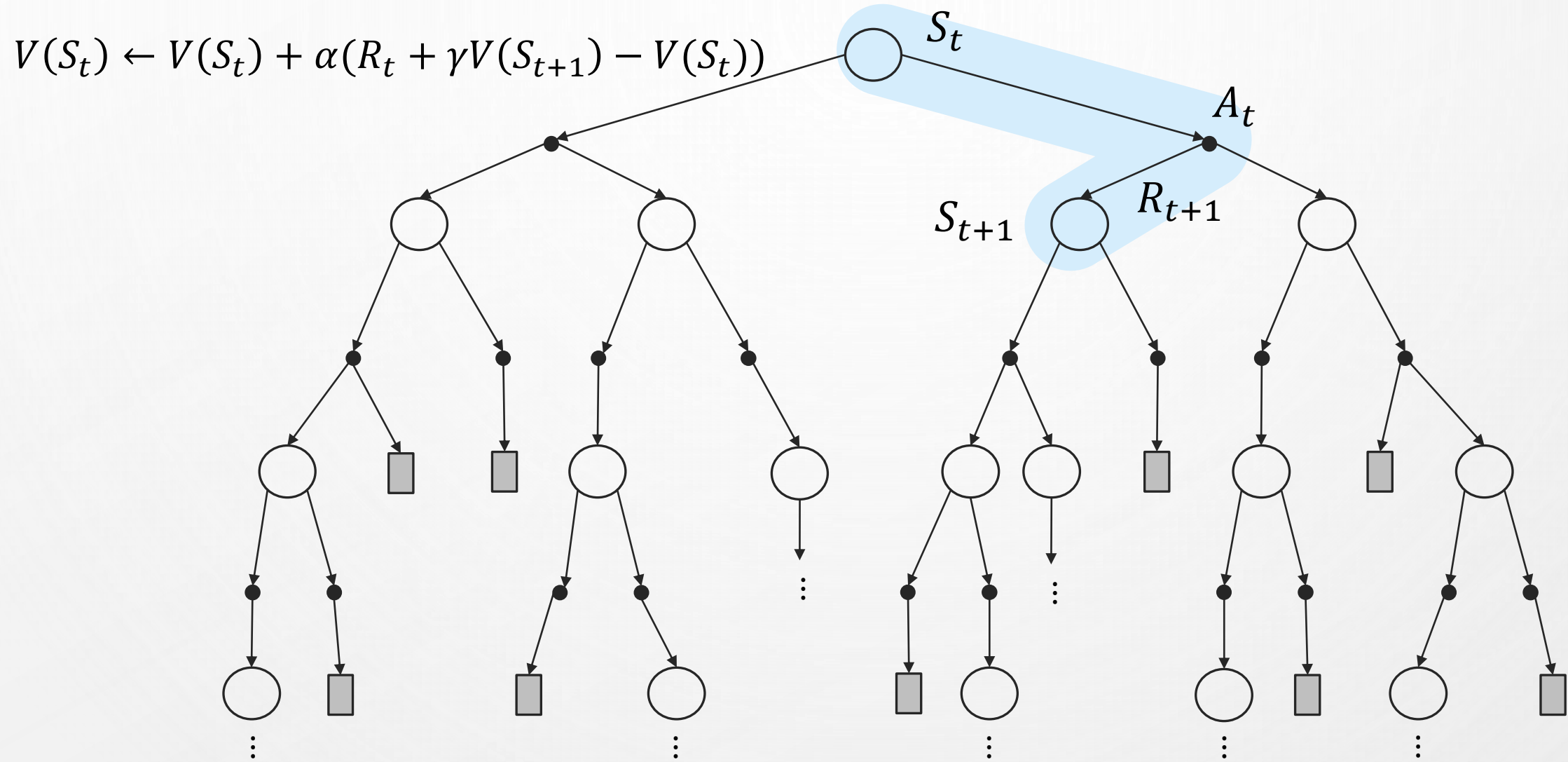


# ATUALIZAÇÕES DE MONTE-CARLO

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



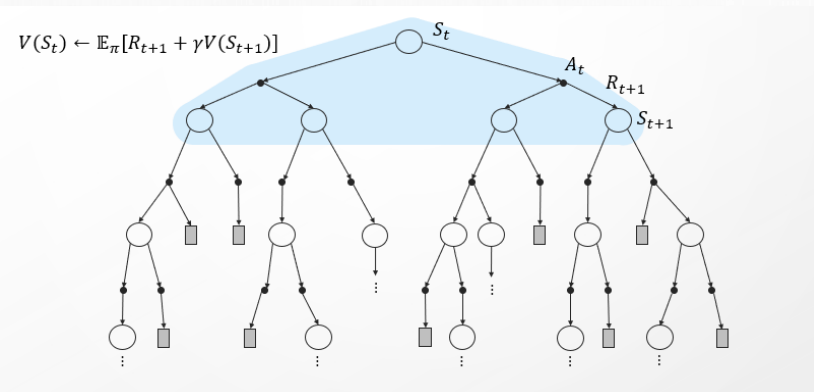
# ATUALIZAÇÕES DE DIFERENÇAS TEMPORAIS TD(0)





# COMPARAÇÃO: DP, MC, TD

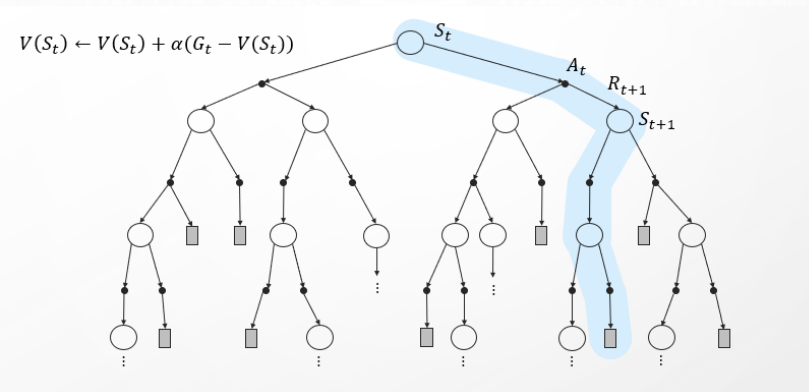
## Programação Dinâmica (DP)



Model-Based, Calcula valor esperado de estimativas de retornos.

- Bootstrapping: ✓
- Sampling: ✗

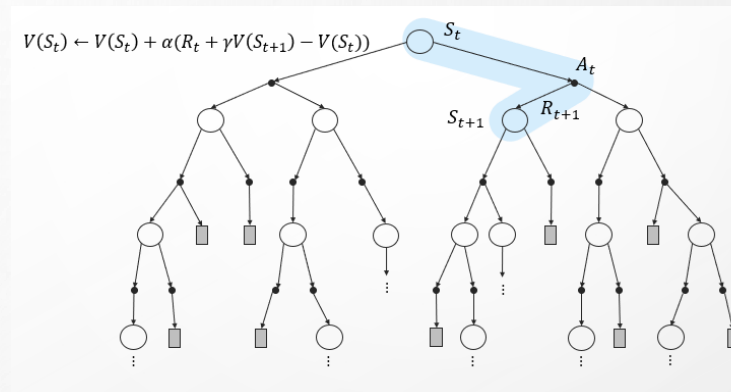
## Monte-Carlo (MC)



Model-Free, Amostra retornos reais e calcula sua média.

- Bootstrapping: ✗
- Sampling: ✓

## Diferenças Temporais (TD)

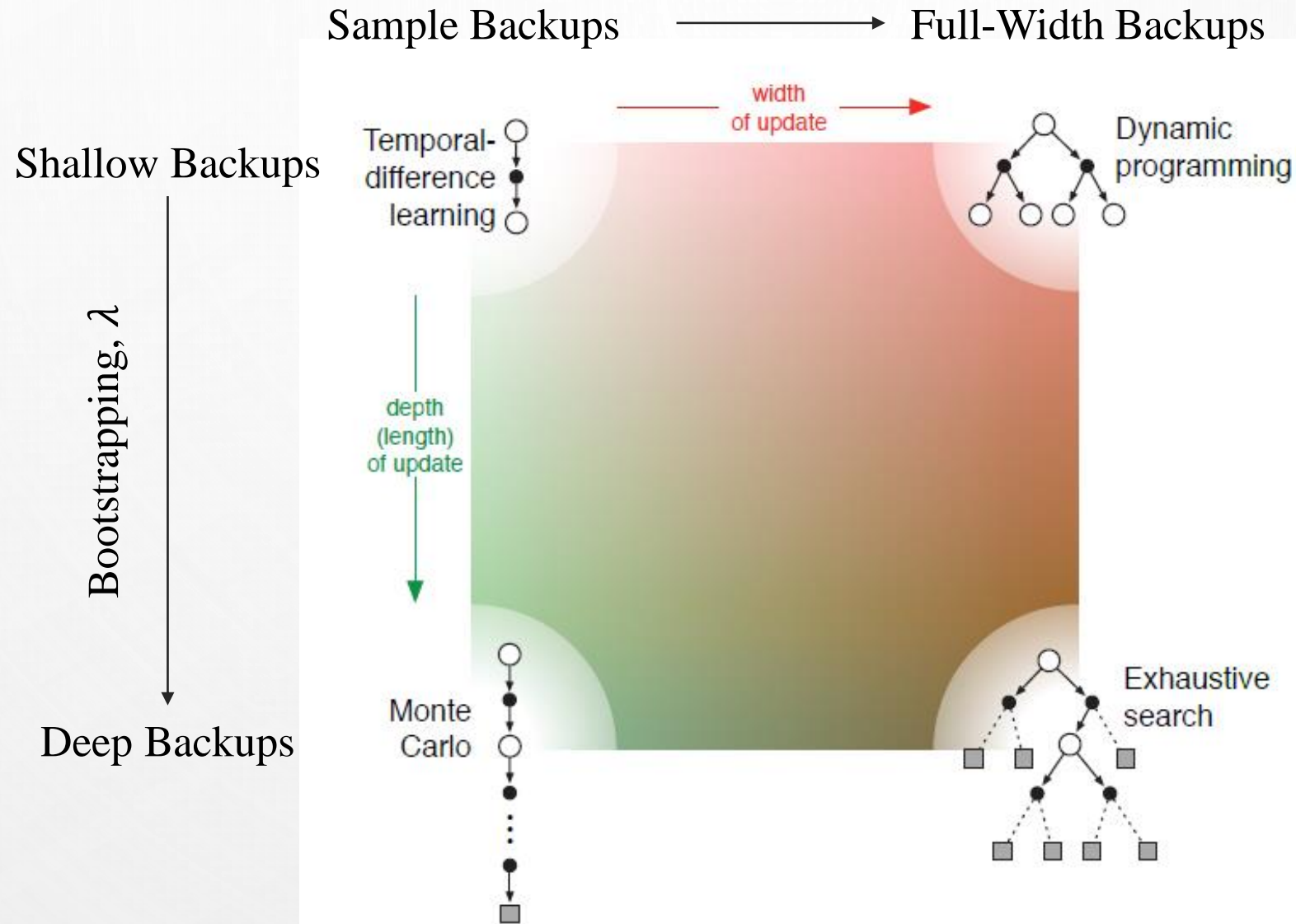


Model-Free, Amostra estimativas de retornos e calcula sua média.

- Bootstrapping: ✓
- Sampling: ✓



# VISÃO GERAL DE ALGORITMOS DE APRENDIZADO POR REFORÇO



# MODEL-FREE PREDICTION: TEMPORAL-DIFFERENCE LEARNING $TD(\lambda)$

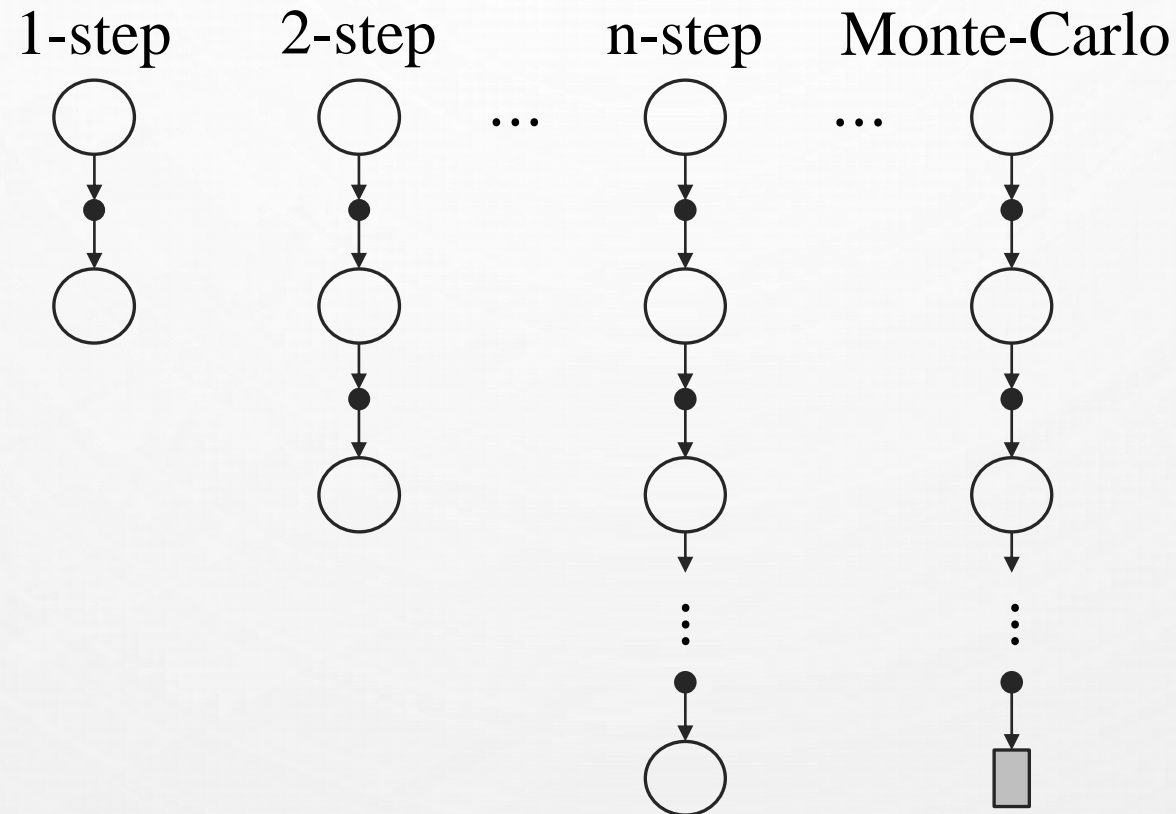
Model-Free Prediction

Temporal-Difference Learning  $TD(\lambda)$

# TEMPORAL-DIFFERENCE LEARNING: N-STEP LOOK-AHEAD

No algoritmo  $TD(0)$  o retorno era estimado por meio de 1-step look-ahead.

Podemos olhar  $n$  passos no futuro para estimar o retorno:



# TEMPORAL-DIFFERENCE LEARNING: N-STEP LOOK-AHEAD

Podemos definir as seguintes estimativas de retornos com n-step look-ahead:

$$n = 1 \text{ (TD(0))} \quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$$

$$n = 2 \quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$$

$$\vdots$$

$$n \quad G_t^{(n)} = \overbrace{R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n}}^{\text{observado}} + \overbrace{\gamma^n V(S_{t+n})}^{\text{estimado}}$$

$$n \rightarrow \infty \text{ (MC)} \quad G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$$

Assim, os algoritmos de n-step Temporal-Difference Learning aplicam a estimativa do retorno com n-step look-ahead:

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t^{(n)} - V(S_t)]$$

# TEMPORAL-DIFFERENCE LEARNING: N-STEP LOOK-AHEAD

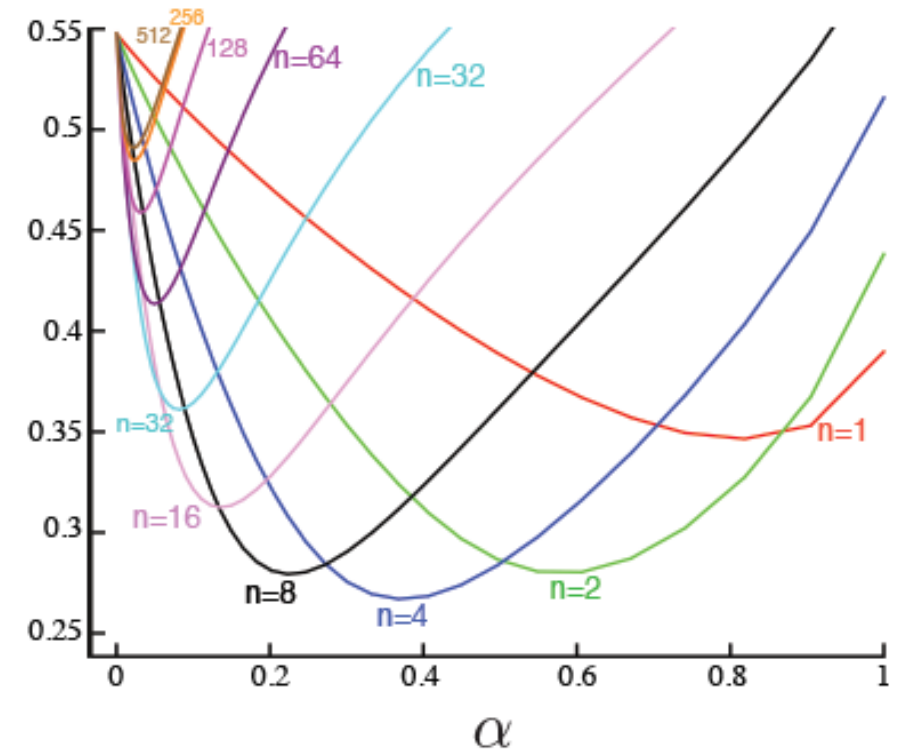
Diferentes problemas apresentam diferentes valores ótimos do número de passos  $n$  par estimativa dos retornos.

Como escolher o melhor valor de  $n$ ?

Ideia: Estimar o retorno como uma média

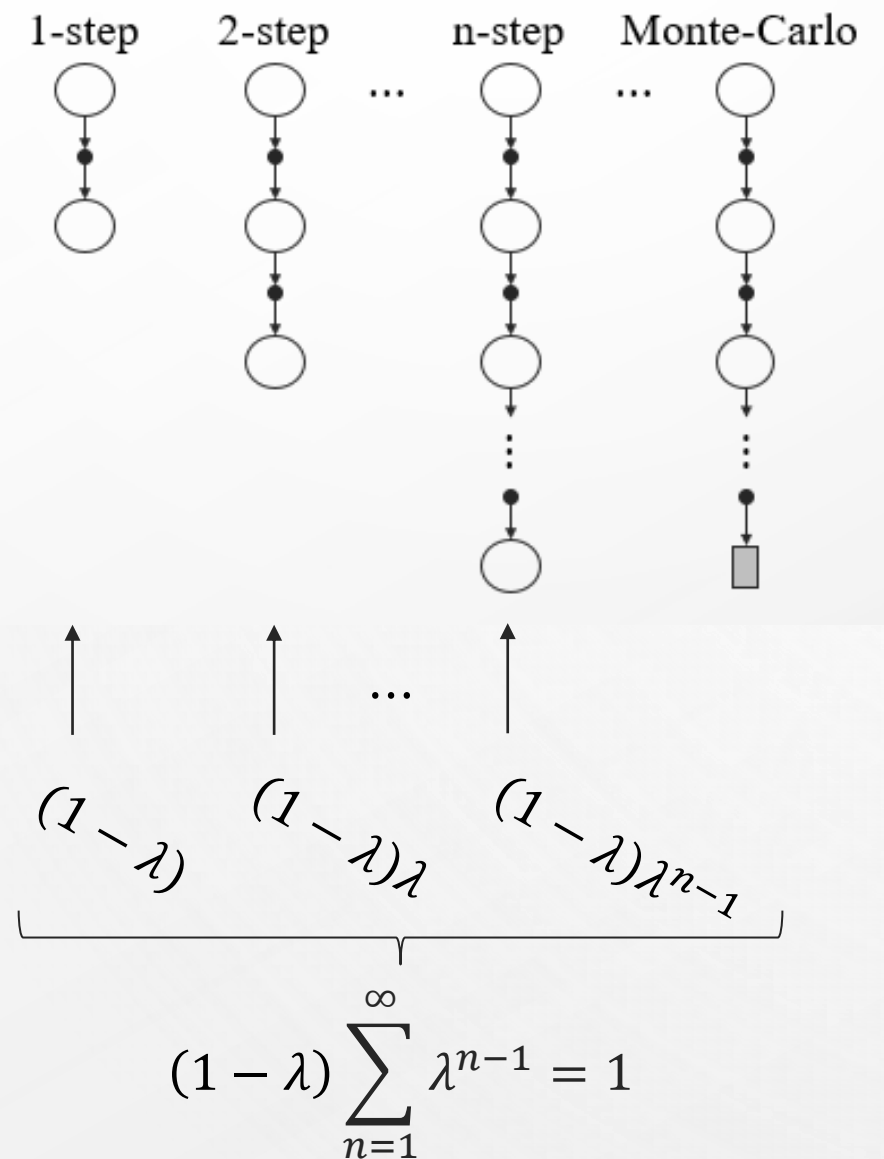
ponderada de  $G_t^{(1)}, G_t^{(2)}, \dots, G_t^{(n)}, \dots$

Average  
RMS error  
over 19 states  
and first 10  
episodes



Erro RMS médio sobre 19 estados para problema de Random Walk nos primeiros 10 episódios para diferentes valores de  $n$  e  $\alpha$ .

# TEMPORAL-DIFFERENCE LEARNING: RETORNO $G_t^\lambda$



- O Retorno  $G_t^\lambda$  combina todos retornos  $G_t^{(n)}$  com uma média ponderada da seguinte forma:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

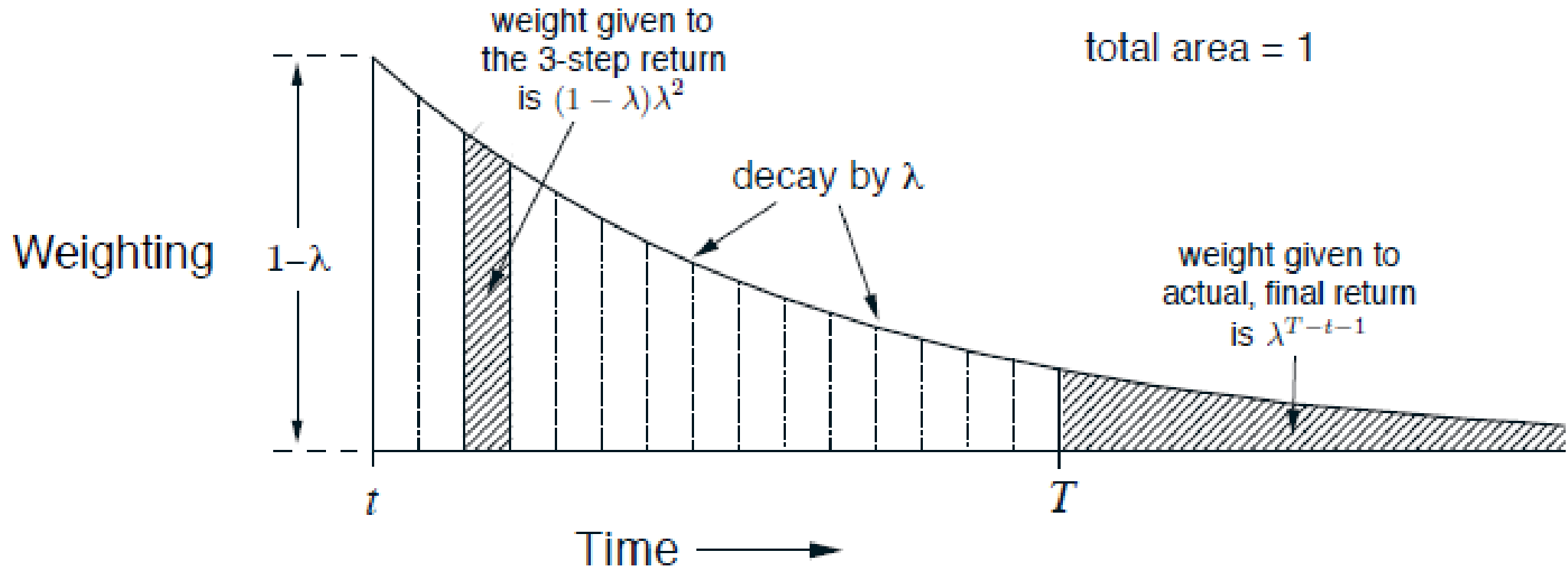
onde  $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$

- O algoritmo  $TD(\lambda)$  possui a seguinte lei de atualização:

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t^\lambda - V(S_t)]$$



# TEMPORAL-DIFFERENCE LEARNING: RETORNO $G_t^\lambda$

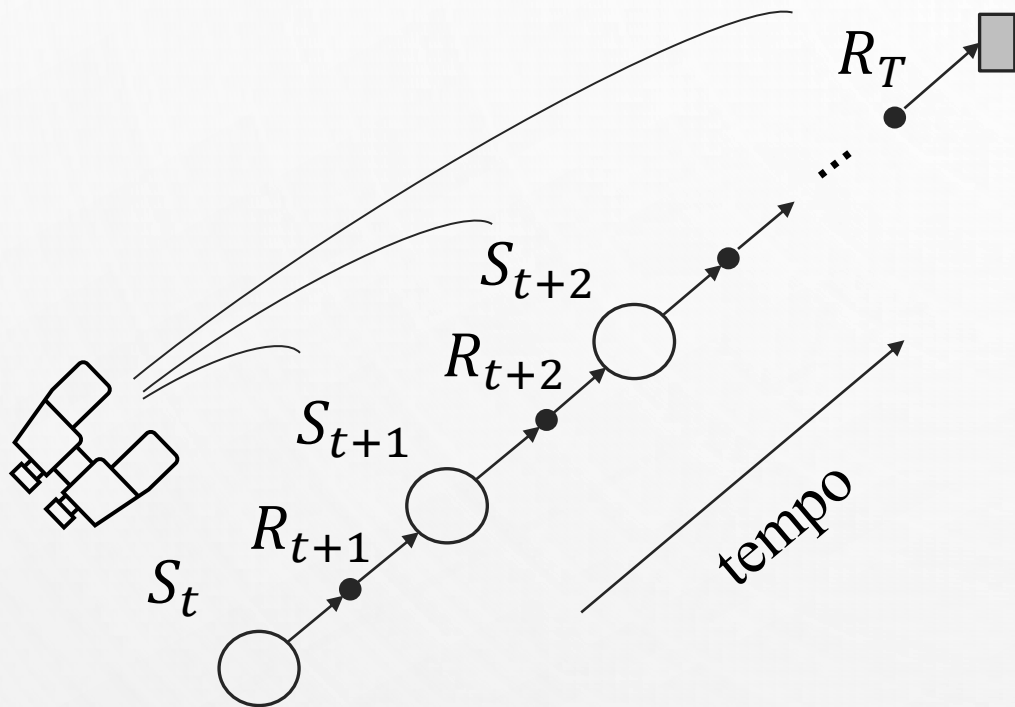


$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

Ponderação dos retornos  $G_t^{(n)}$  para cálculo de  $G_t^\lambda$ .

Fonte: Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction*, The MIT Press (2020)

# TEMPORAL-DIFFERENCE LEARNING: FORWARD VIEW $TD(\lambda)$

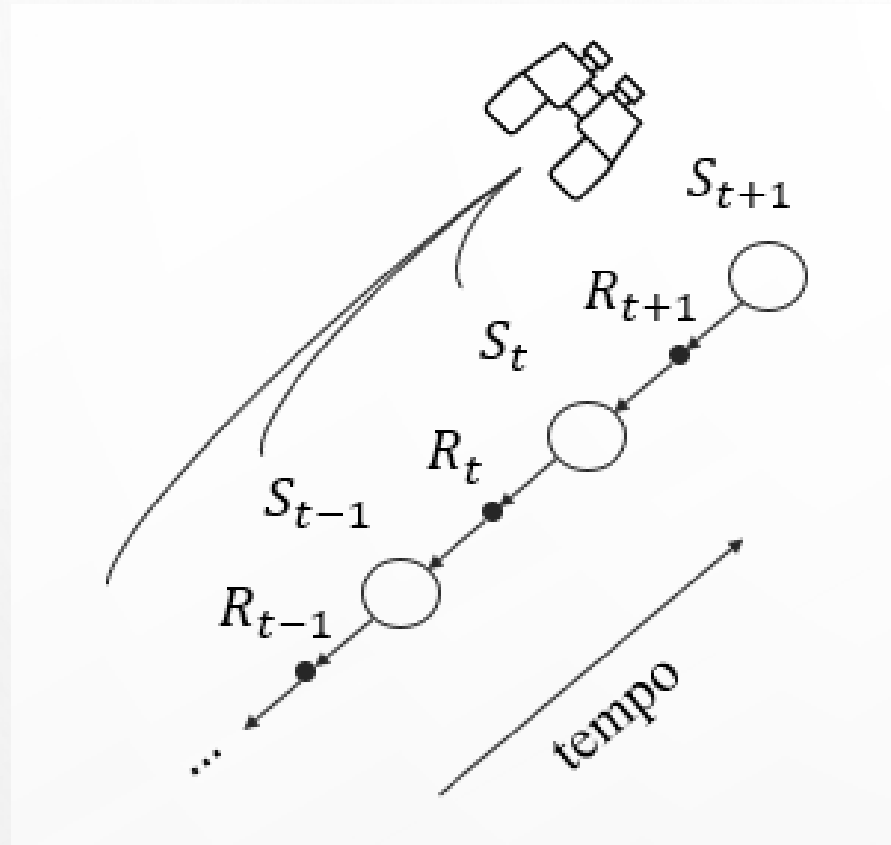


$$V(S_t) \leftarrow V(S_t) + \alpha[G_t^\lambda - V(S_t)]$$

- Atualiza a Função Valor  $V(s)$  na direção do Retorno  $G_t^\lambda$ .
- Olha para o futuro para determinar  $G_t^\lambda$ .
- Como em métodos de Monte-Carlo, só pode ser implementado em problemas de horizonte finito, a partir de episódios completos.

Como implementar  $TD(\lambda)$  online, sem a necessidade de episódios completos?

# TEMPORAL-DIFFERENCE LEARNING: BACKWARD VIEW $TD(\lambda)$

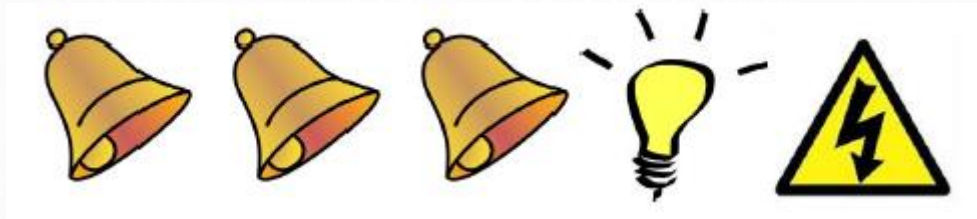


- $TD(\lambda)$ , assim como MC, requer a simulação de episódios completos.
- Seria interessante atualizar a função valor a cada passo, não somente no final do episódio. De modo a aprender online.

Como implementar  $TD(\lambda)$  online, sem a necessidade de episódios completos?

→ Eligibility Traces

# TEMPORAL-DIFFERENCE LEARNING: ELIGIBILITY TRACES



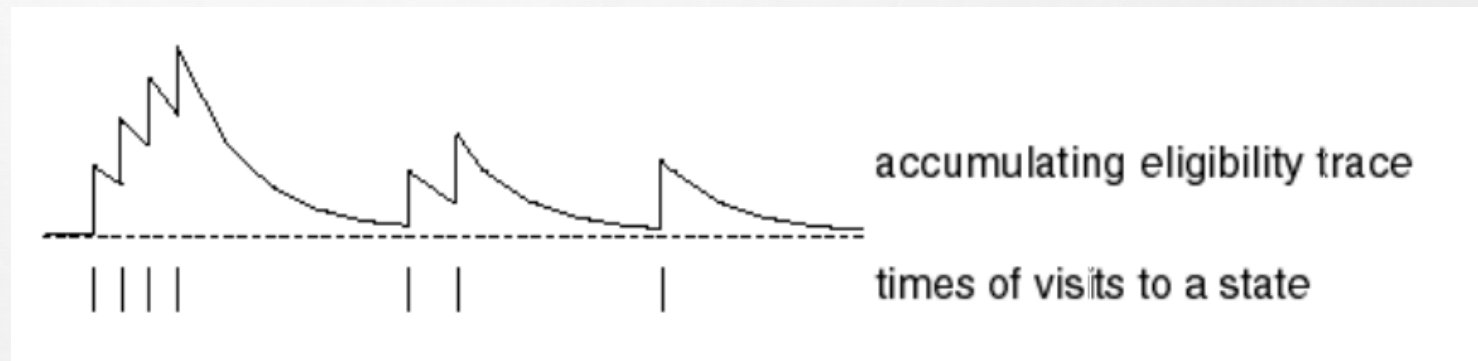
Problema de Atribuição de Crédito: O sino ou a lâmpada causou o choque?

- Heurística Frequentista: Atribuir crédito a estados mais frequentes (sino)
- Heurística de Recência: Atribuir crédito a estados mais recentes (lâmpada)

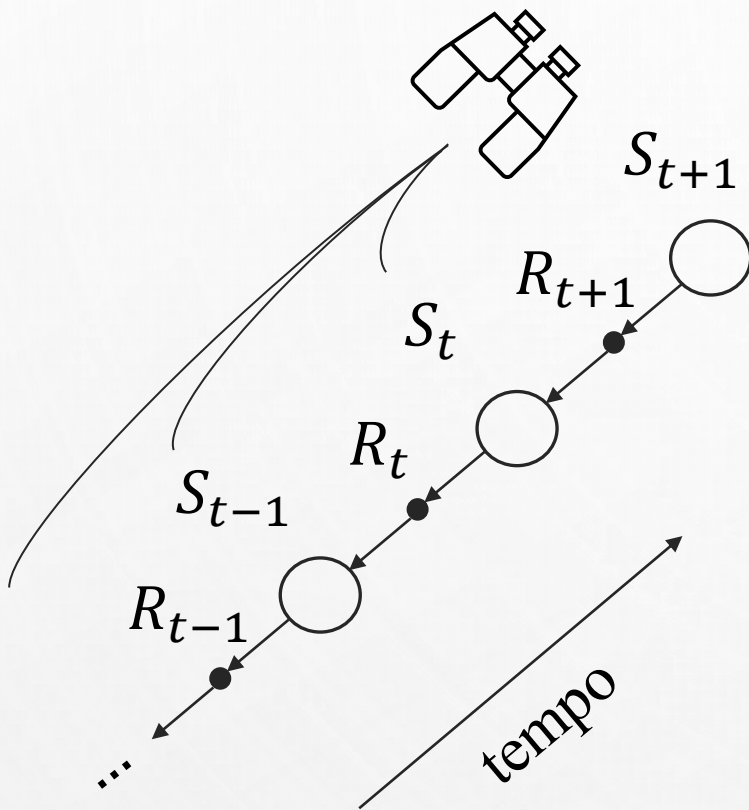
**Eligibility Traces:** Combinam ambas abordagens:

$$E_0(s) = 0$$

$$E_t(s) = \gamma\lambda E_{t-1}(s) + 1(S_t = s)$$



# TEMPORAL-DIFFERENCE LEARNING: BACKWARD VIEW $TD(\lambda)$



$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

- Mantém registro de Eligibility Traces para cada estado  $s$
- Atualiza Função Valor  $V(s)$  de acordo com TD-error  $\delta_t$  e Eligibility Trace  $E_t(s)$ :

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

- Quando  $\lambda = 0$ , somente o estado atual é atualizado, o que equivale a  $TD(0)$ :

$$E_t(s) = 1 \Rightarrow V(S_t) \leftarrow V(S_t) + \alpha \delta_t$$

# TEMPORAL-DIFFERENCE LEARNING: BACKWARD-VIEW $TD(\lambda)$ FOR POLICY EVALUATION

**Algoritmo:** Backward View  $TD(\lambda)$  Policy Evaluation for estimating  $V_\pi$

**Input:** Política  $\pi$  a ser avaliada

**Parâmetro do Algoritmo:** Taxa de aprendizado  $\alpha \in (0,1]$  e taxa de decaimento  $\lambda \in [0,1]$ .

**Inicializar**  $V(s)$  arbitrariamente para todo  $s \in \mathcal{S}$ , com  $V(s_{term}) = 0$  para estados terminais.

Repetir para cada episódio:

    Inicializar estado inicial  $S_0 \sim \mathbb{P}(S_0 = s), s \in \mathcal{S}$

    Inicializar Eligibility Traces  $E(s) = 0, \forall s \in \mathcal{S}$

    Repetir para cada timestep  $t = 0, 1, 2, \dots$ :

        Amostrar ação da política  $A_t \sim \pi(a|S_t)$

        Executar ação  $A_t$  e observar  $R_{t+1}, S_{t+1}$

        Atualizar Eligibility Traces:  $E(s) \leftarrow \gamma\lambda E(s) + 1(S_t = s), \forall s \in \mathcal{S}$

        Calcular TD-error:  $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

        Para todo estado  $s$ :

$V(s) \leftarrow V(s) + \alpha \delta_t E(s)$

$S_t = S_{t+1}$


    Até que  $S_t$  seja um estado terminal

**Retorna:**  $V(s) \approx V_\pi(s), \forall s \in \mathcal{S}$



## Exercício $E_2$

# EXERCÍCIO $E_2$

 E2.ipynb ☆  
File Edit View Insert Runtime Tools Help [All changes saved](#)

Comment Share ⚙️ L

Table of contents

Exercício 2 - Programação Dinâmica: Model-Based Prediction and Control

Ambiente GridWorld

Processo de Decisão de Markov (MDP)

DP: Policy Evaluation

DP: Policy Iteration

Tarefas

Código

Imports

Global Variables

Funções Auxiliares

fig2img(fig):

get\_states\_list(env):

random\_policy(states\_list):

plot\_value\_function(env,V):

greedy\_improvement(env,V):

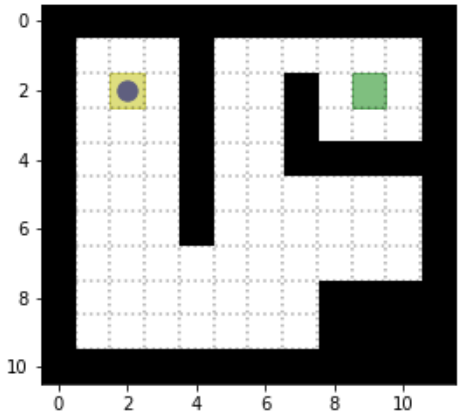
+ Code + Text

Connect Editing

↑ ↓ ↻ ↺ ↻ ↻ ↻ ↻

## Exercício 2 - Programação Dinâmica: Model-Based Prediction and Control

Gridworld



A 10x10 grid world environment. The grid is bounded by black walls. The start cell is a yellow square with a blue dot at (2, 2). The goal cell is a green square at (9, 2). Black walls are located at (4, 0) to (4, 6), (7, 2) to (7, 4), (8, 2) to (8, 4), (8, 5) to (8, 8), and (9, 0) to (9, 8). The axes are labeled from 0 to 10.

### Ambiente GridWorld

Considere o ambiente ilustrado na figura acima, em que células em preto representam paredes, células em branco indicam estados nos quais um agente (azul) pode se encontrar. O problema consiste em levar o agente de uma posição inicial (amarelo) até uma posição de

# EXERCÍCIO $E_2$

- Tarefa a)

Implementar função *env.step(s, a)*

# EXERCÍCIO $E_2$

- Tarefa a)

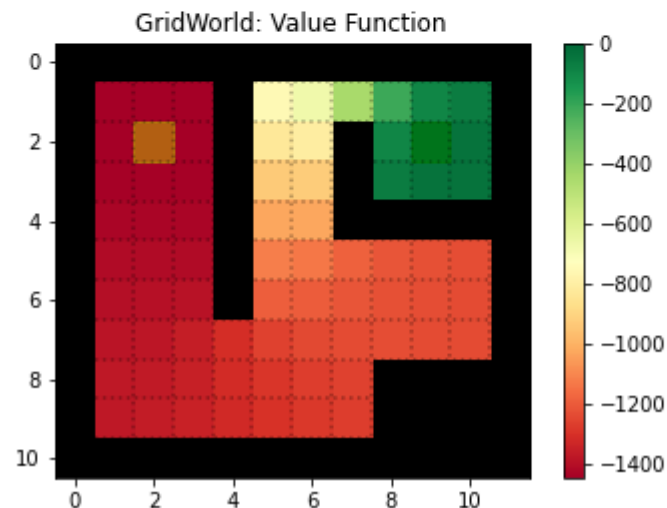
Implementar função `env.step(s, a)`

- Tarefa b)

`policy_evaluation(env, policy, theta)`

Value Function  $V(s)$ :

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 -1446 -1446 -1446  0 -750 -667 -441 -210 -93 -66  0]
 [ 0 -1443 -1443 -1442  0 -829 -807  0 -93  0 -35  0]
 [ 0 -1436 -1435 -1435  0 -926 -922  0 -66 -35 -37  0]
 [ 0 -1425 -1425 -1424  0 -1025 -1027  0  0  0  0  0]
 [ 0 -1412 -1410 -1408  0 -1117 -1132 -1186 -1212 -1225 -1230  0]
 [ 0 -1397 -1393 -1386  0 -1190 -1194 -1210 -1221 -1229 -1232  0]
 [ 0 -1382 -1374 -1355 -1310 -1257 -1239 -1235 -1231 -1232 -1234  0]
 [ 0 -1372 -1363 -1345 -1315 -1284 -1266 -1258  0  0  0  0]
 [ 0 -1367 -1359 -1342 -1318 -1295 -1278 -1270  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0]]
```



# EXERCÍCIO $E_2$

- Tarefa a)

Implementar função `env.step(s, a)`

- Tarefa b)

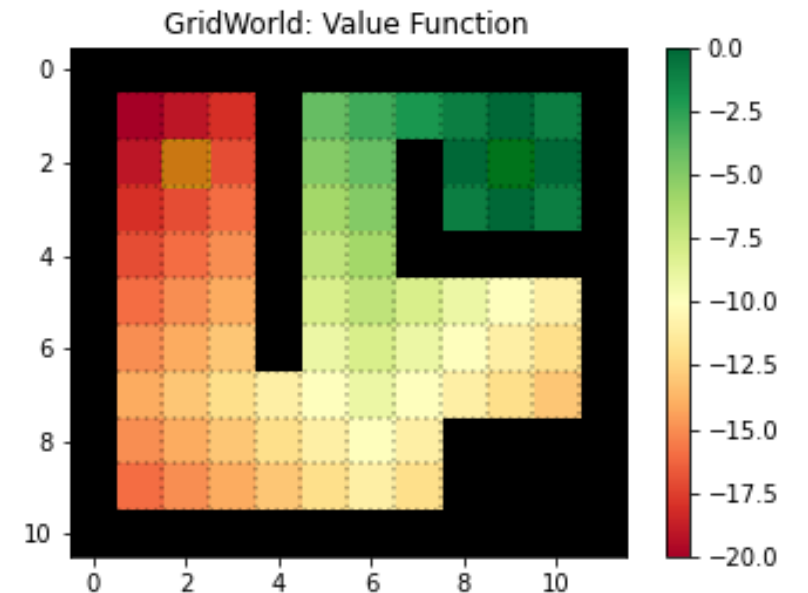
`policy_evaluation(env, policy, theta)`

- Tarefa c)

`policy_iteration(env, theta)`

Value Function  $V(s)$ :

```
[[ 0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 -20 -19 -18  0 -4 -3 -2 -1  0 -1  0]
 [ 0 -19 -18 -17  0 -5 -4  0  0  0  0  0]
 [ 0 -18 -17 -16  0 -6 -5  0 -1  0 -1  0]
 [ 0 -17 -16 -15  0 -7 -6  0  0  0  0  0]
 [ 0 -16 -15 -14  0 -8 -7 -8 -9 -10 -11  0]
 [ 0 -15 -14 -13  0 -9 -8 -9 -10 -11 -12  0]
 [ 0 -14 -13 -12 -11 -10 -9 -10 -11 -12 -13  0]
 [ 0 -15 -14 -13 -12 -11 -10 -11  0  0  0  0]
 [ 0 -16 -15 -14 -13 -12 -11 -12  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0]]
```



# REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction*, The MIT Press (2020). [Cp. 4,5,6]
- [2] <https://medium.com/@edu.pignatelli/iterative-policy-evaluation-33056f3f21a4>
- [3] <https://becomesentient.com/mdp-dynamic-programming/>
- [4] <https://zsalloum.medium.com/monte-carlo-in-reinforcement-learning-the-easy-way-564c53010511>



Muito obrigado a todos!

Dúvidas