

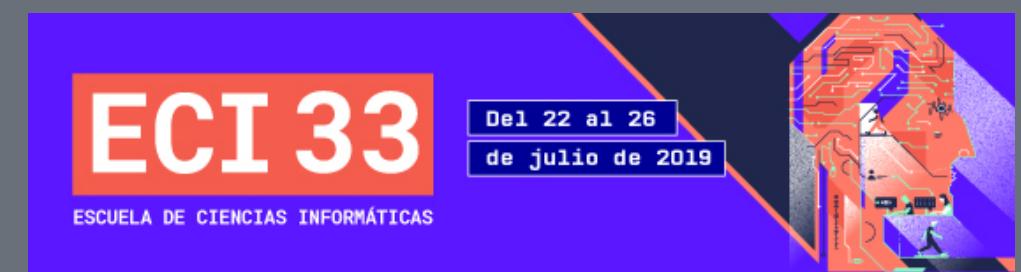
Aprendizaje Profundo por Refuerzo

03. Aprendizaje por refuerzo

Dr. Juan Gómez Romero

Investigador Senior

Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada



Aprendizaje por refuerzo

Bibliografía



UNIVERSIDAD
DE GRANADA

R.S. Sutton, A.G. Barto (2018) Reinforcement Learning. MIT Press.

<https://mitpress.mit.edu/books/reinforcement-learning-second-edition>

A. Zai, B. Brown (2018) Deep Reinforcement Learning in Action. Manning.

M. Morales (2018) Grokking Deep Reinforcement Learning. Manning.

Índice

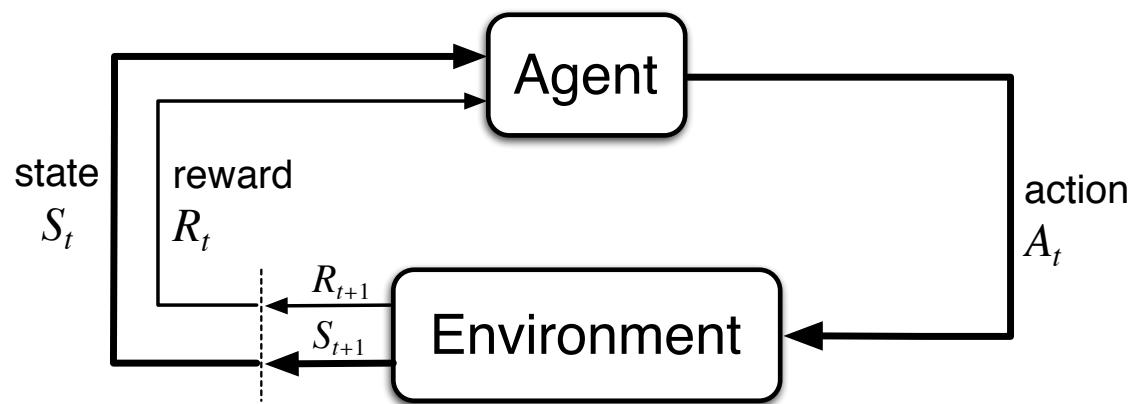
1. Procesos de Decisión de Markov (MDP)
2. Método de Montecarlo
3. Q-Learning

1 MDP

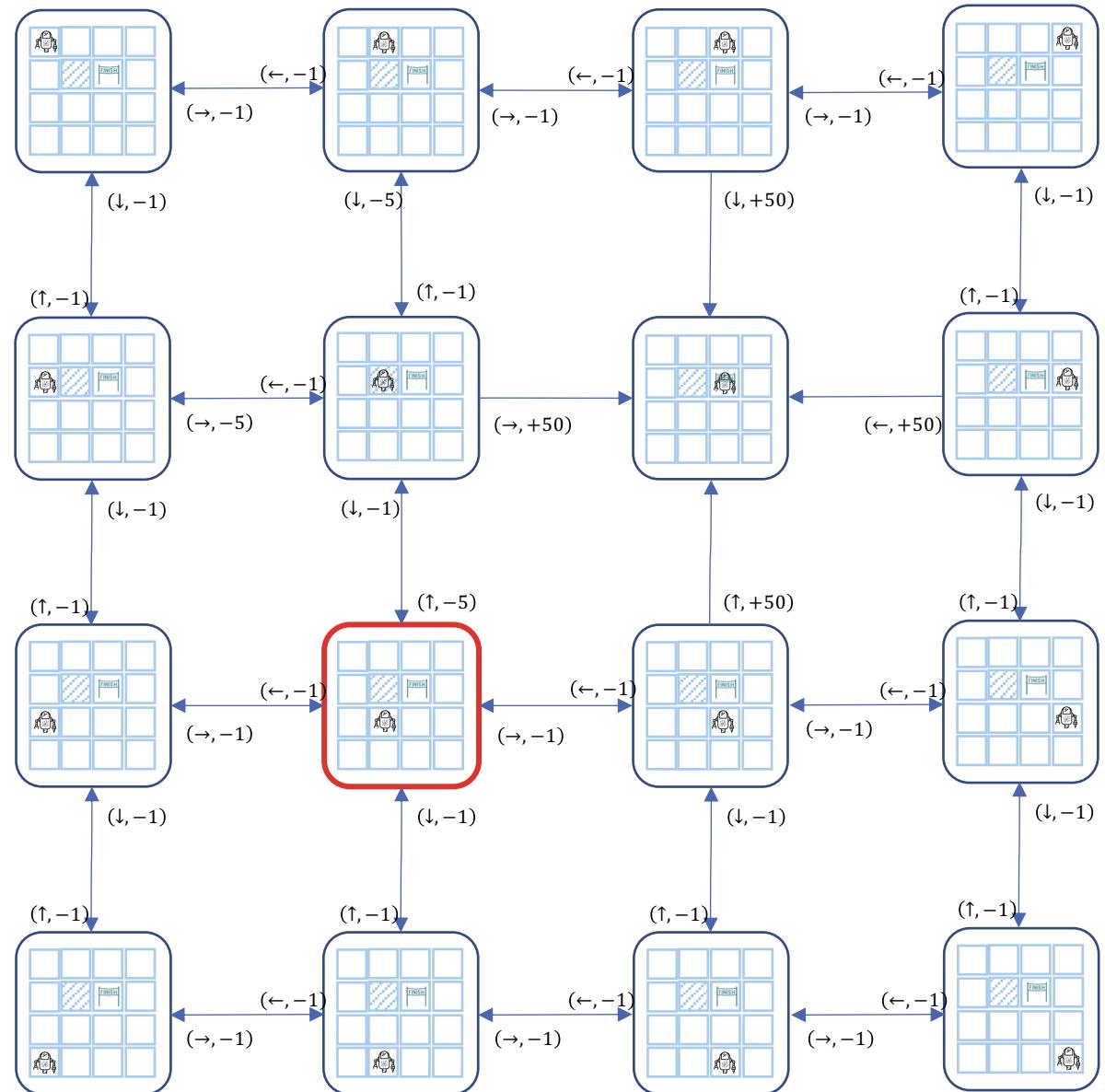
PROCESOS DE DECISIÓN DE MARKOV

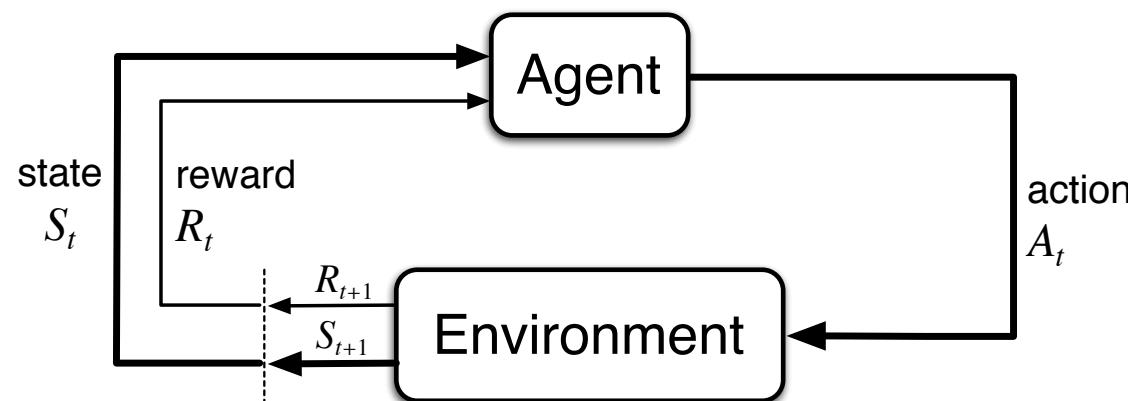
Aprendizaje por refuerzo

Problema de búsqueda en el grafo de estados y transiciones.
Optimizar la función de recompensa acumulada (*max*)



R.S. Sutton, A.G. Barto (2018) **Reinforcement Learning**. MIT Press.





R.S. Sutton, A.G. Barto (2018) **Reinforcement Learning**. MIT Press.

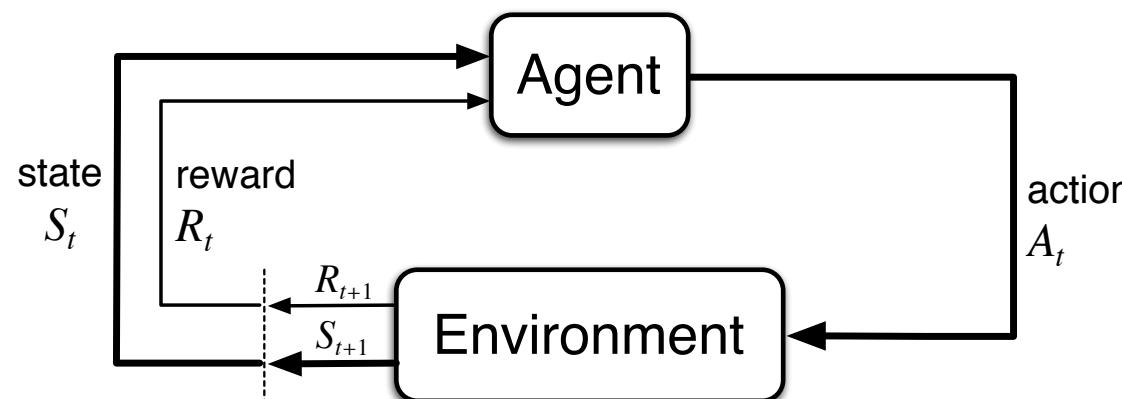
Markov Decision Process - MDP

tiempo: $0, 1, 2, \dots$
estado: $S_t \in \mathcal{S}$
acción: $A_t \in \mathcal{A}(S_t)$
recompensa: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$
estado siguiente: $S_{t+1} \in \mathcal{S}$

secuencia:
 $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$

dinámica del MDP:

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$



Markov Decision Process - MDP

Recompensa acumulada:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T$$

Recompensa con descuento:

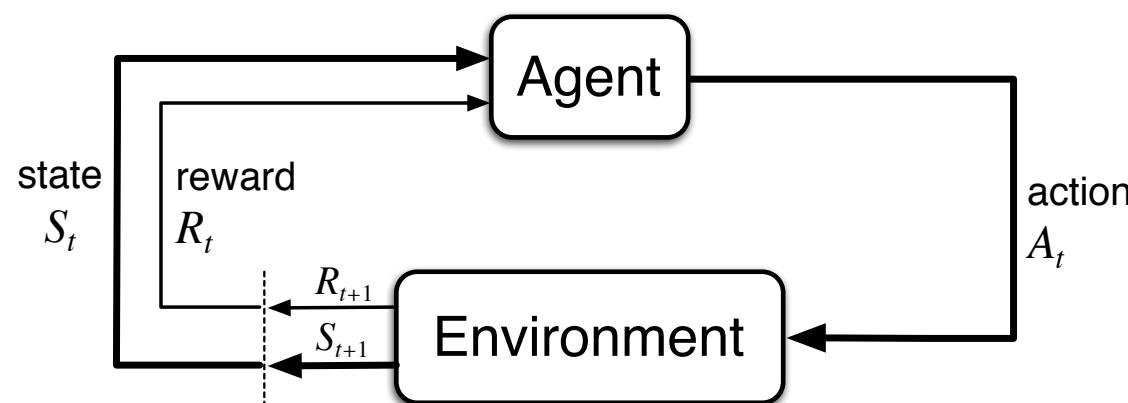
$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^T \gamma^k R_{t+k+1}$$

Política de actuación: π

Acción según π : $\pi(a|s)$

Política de actuación óptima: π_*

R.S. Sutton, A.G. Barto (2018) *Reinforcement Learning*. MIT Press.



R.S. Sutton, A.G. Barto (2018) *Reinforcement Learning*. MIT Press.

Markov Decision Process - MDP

Función de estado-valor [*state value*] (para todo s):

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

Función de acción-valor [*action value*] (para todo (s, a)):

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

Comparación de políticas de actuación y política óptima

$$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v'_{\pi'}(s)$$

$$\pi_* \geq \pi' \quad \forall \pi' \quad (\text{garantizada})$$

MDP

Formulación

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

Política π_1 ($\gamma = 1$) :

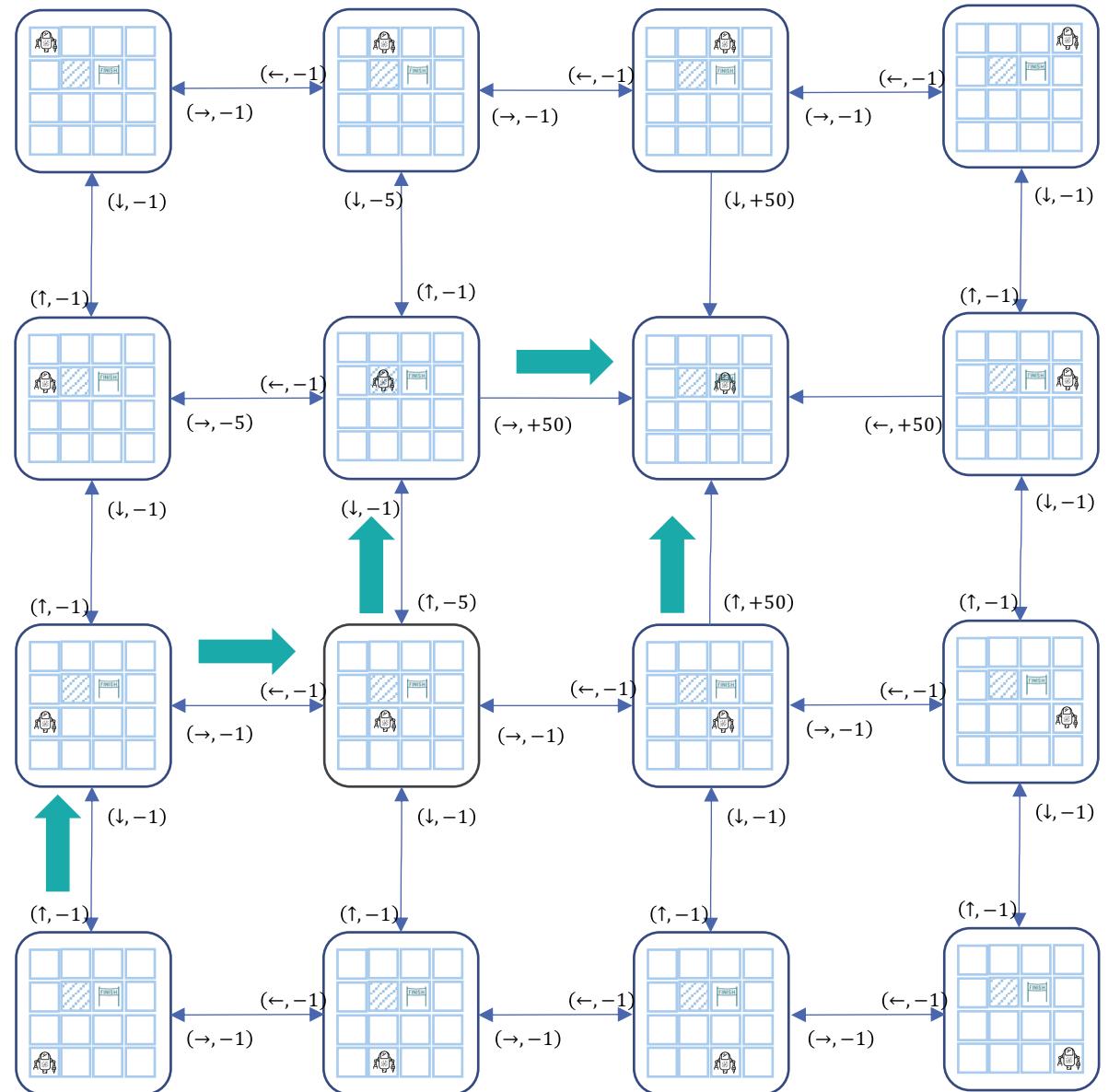
Estado	Acción
(3, 0)	\uparrow
(2, 0)	\rightarrow
(2, 1)	\uparrow
(1, 1)	\rightarrow
(2, 2)	\uparrow
...	

$$v_{\pi_1}(\langle 2, 1 \rangle) = -5 + 50 = 45$$

$$v_{\pi_1}(\langle 2, 2 \rangle) = 50$$

$$v_{\pi_1}(\langle 3, 0 \rangle) = -1 - 1 - 5 + 50 = 43$$

$$v_{\pi_1}(\langle 2, 0 \rangle) = -1 - 5 + 50 = 42$$



Formulación

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

Política π_2 ($\gamma = 1$) :

Estado	Acción
(3, 0)	\uparrow
(2, 0)	\rightarrow
(2, 1)	\rightarrow
(1, 1)	\rightarrow
(2, 2)	\uparrow
...	

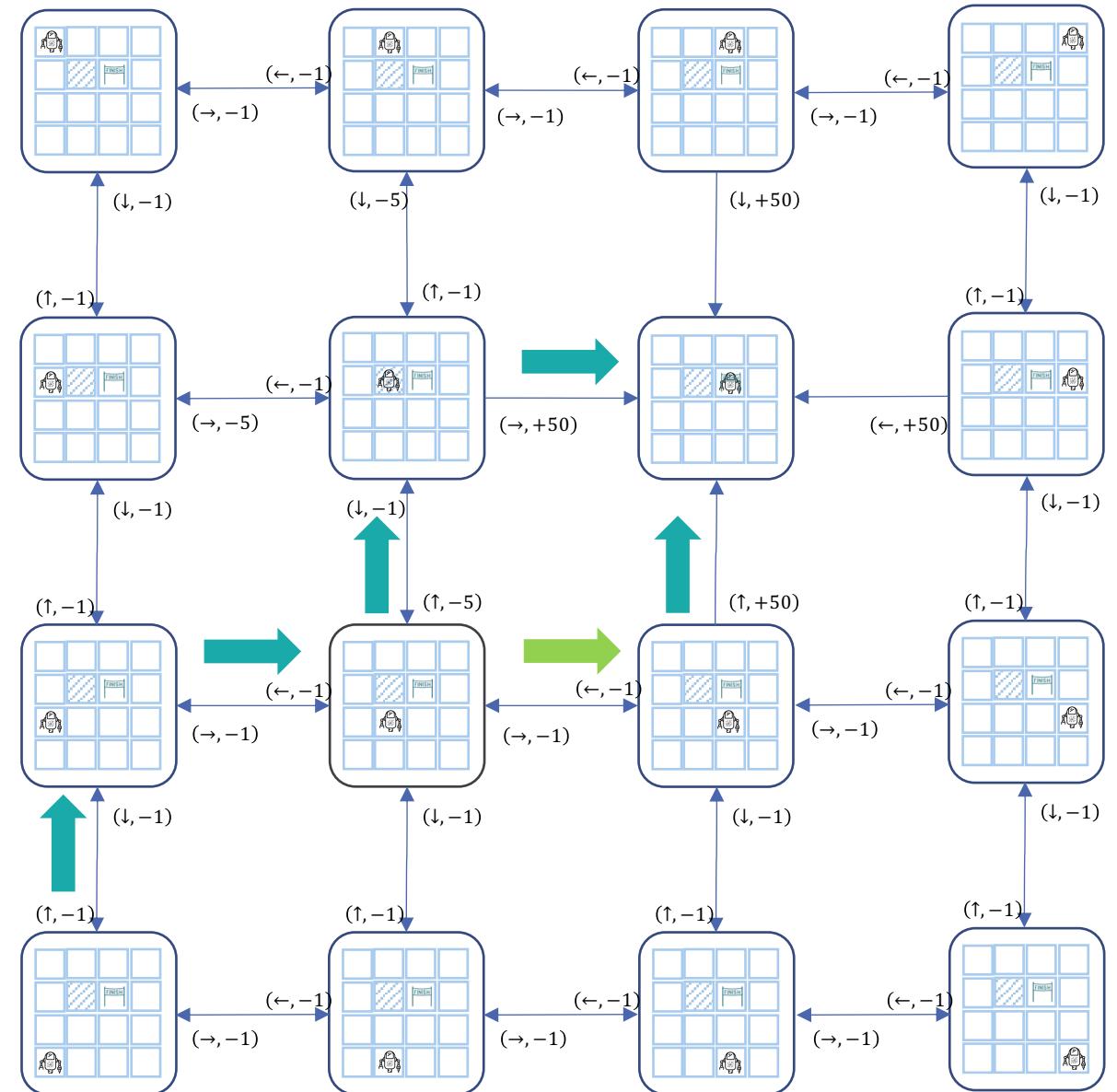


$$v_{\pi_2}(\langle 2, 1 \rangle) = -1 + 50 = 49 \geq 45$$

$$v_{\pi_2}(\langle 2, 2 \rangle) = 50 \geq 50$$

$$v_{\pi_2}(\langle 3, 0 \rangle) = -1 - 1 - 1 + 50 = 47 \geq 43$$

$$v_{\pi_2}(\langle 2, 0 \rangle) = -1 - 1 + 50 = 48 \geq 42$$



Formulación

$$q_{\pi}(s, a) \doteq \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$

Política π_2 ($\gamma = 1$) :

Estado	Acción
(3, 0)	\uparrow
(2, 0)	\rightarrow
(2, 1)	\rightarrow
(1, 1)	\rightarrow
(2, 2)	\uparrow
(3, 1)	\leftarrow
...	

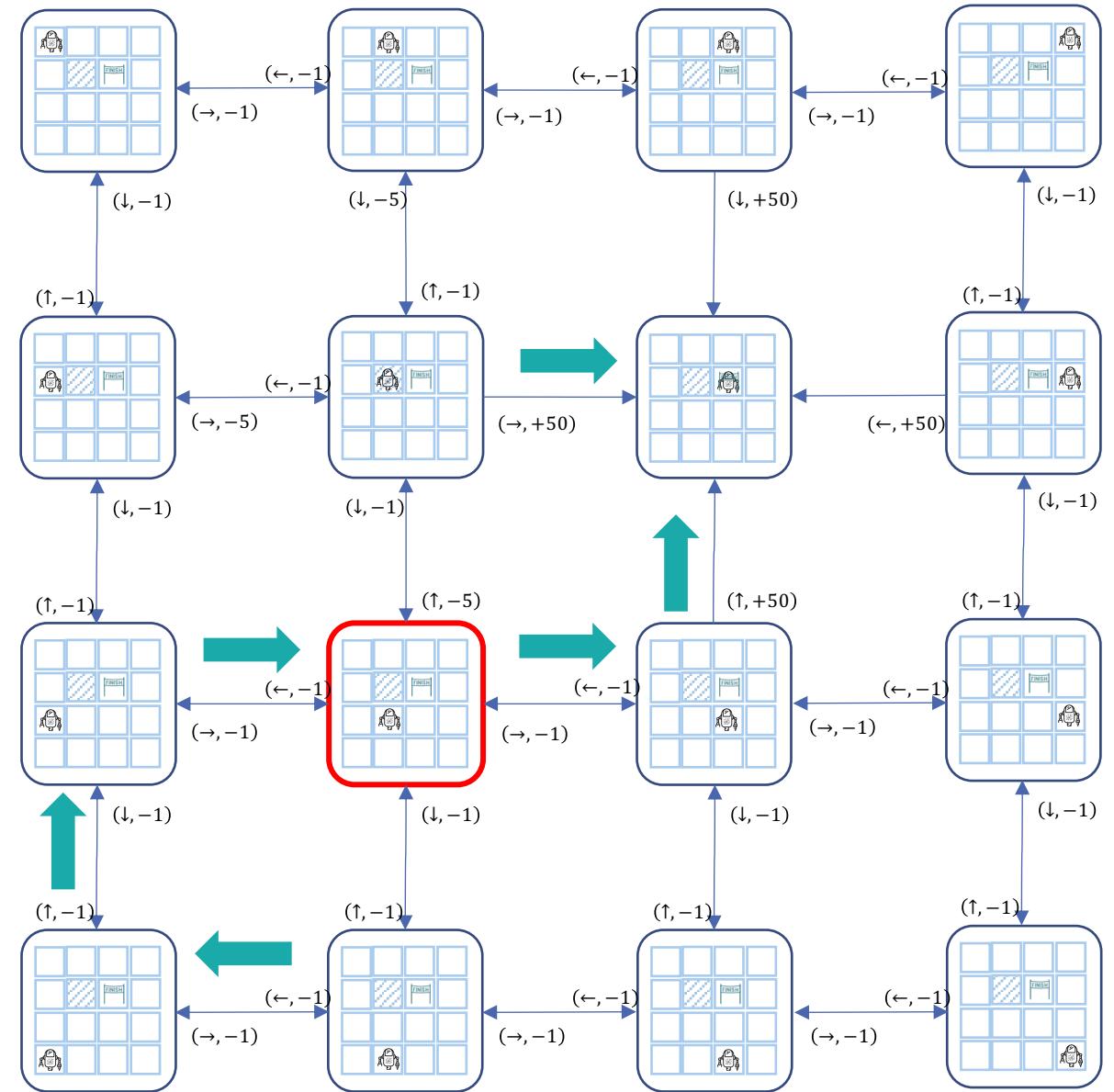
$$v_{\pi}(s) = q_{\pi}(s, \pi(a|s))$$

$$q_{\pi_2}(\langle 2, 1 \rangle, \uparrow) = -5 + 50 = 45$$

$$q_{\pi_2}(\langle 2, 1 \rangle, \rightarrow) = -1 + 50 = 49$$

$$q_{\pi_2}(\langle 2, 1 \rangle, \leftarrow) = -1 - 1 - 1 + 50 = 47$$

$$q_{\pi_2}(\langle 2, 1 \rangle, \downarrow) = -1 - 1 - 1 - 1 - 1 + 50 = 45$$



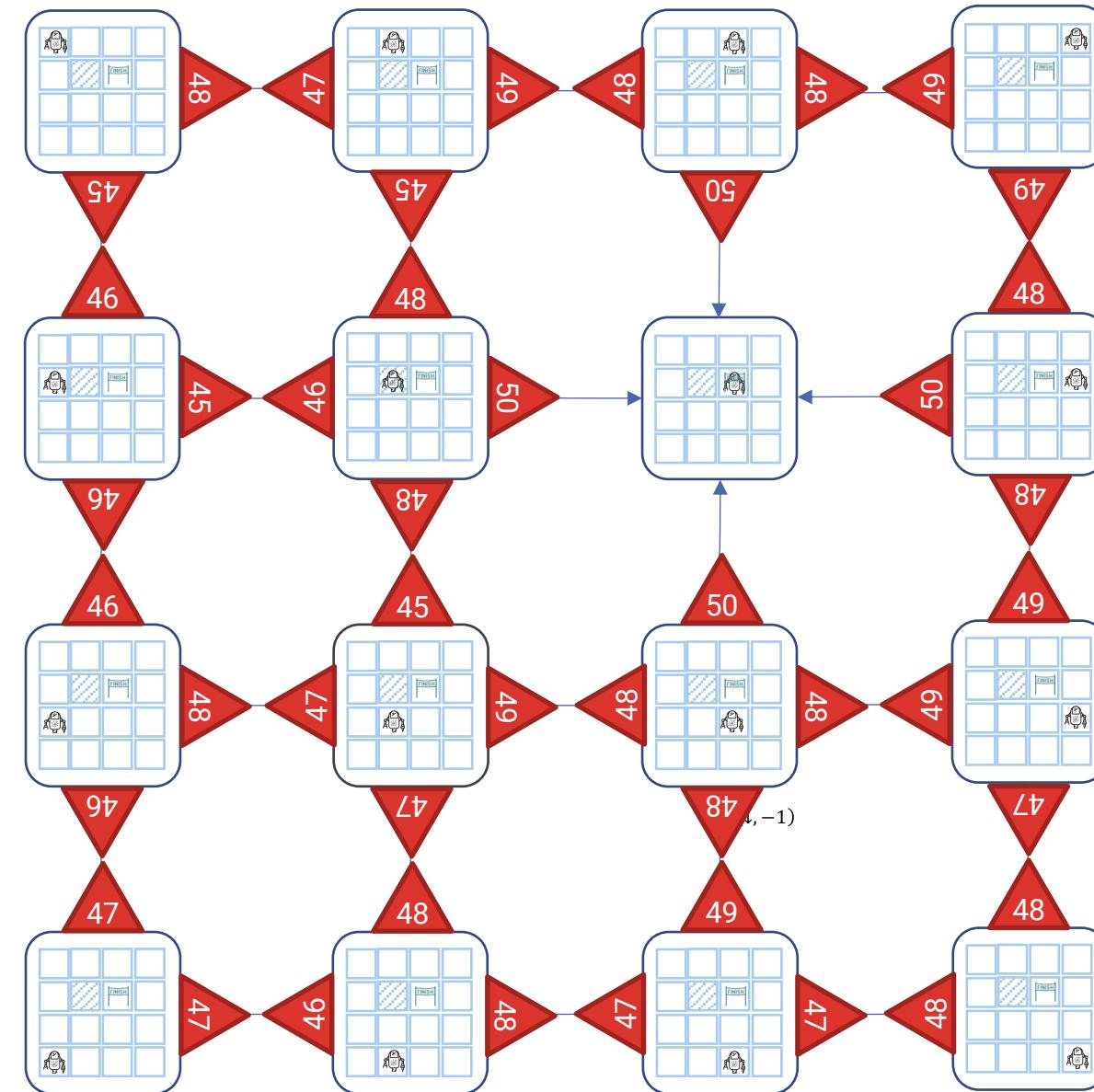
Ecuación de Bellman (recursividad de v_π)

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[\underbrace{R_{t+1}}_{\substack{\text{recompensa} \\ \text{al realizar acción}}} + \underbrace{\gamma v_\pi(S_{t+1})}_{\substack{\text{valor del} \\ \text{siguiente estado}}} \mid S_t = s] \end{aligned}$$

Ecuación de Bellman (optimalidad de v_*)

$$\begin{aligned} v_{\pi_*}(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\ &= \max_{\substack{\text{valoración} \\ \text{de un estado} \\ (\text{política óptima})}} q_{\pi_*}(s, a) \end{aligned}$$

$$q_{\pi_*}(s, a)$$



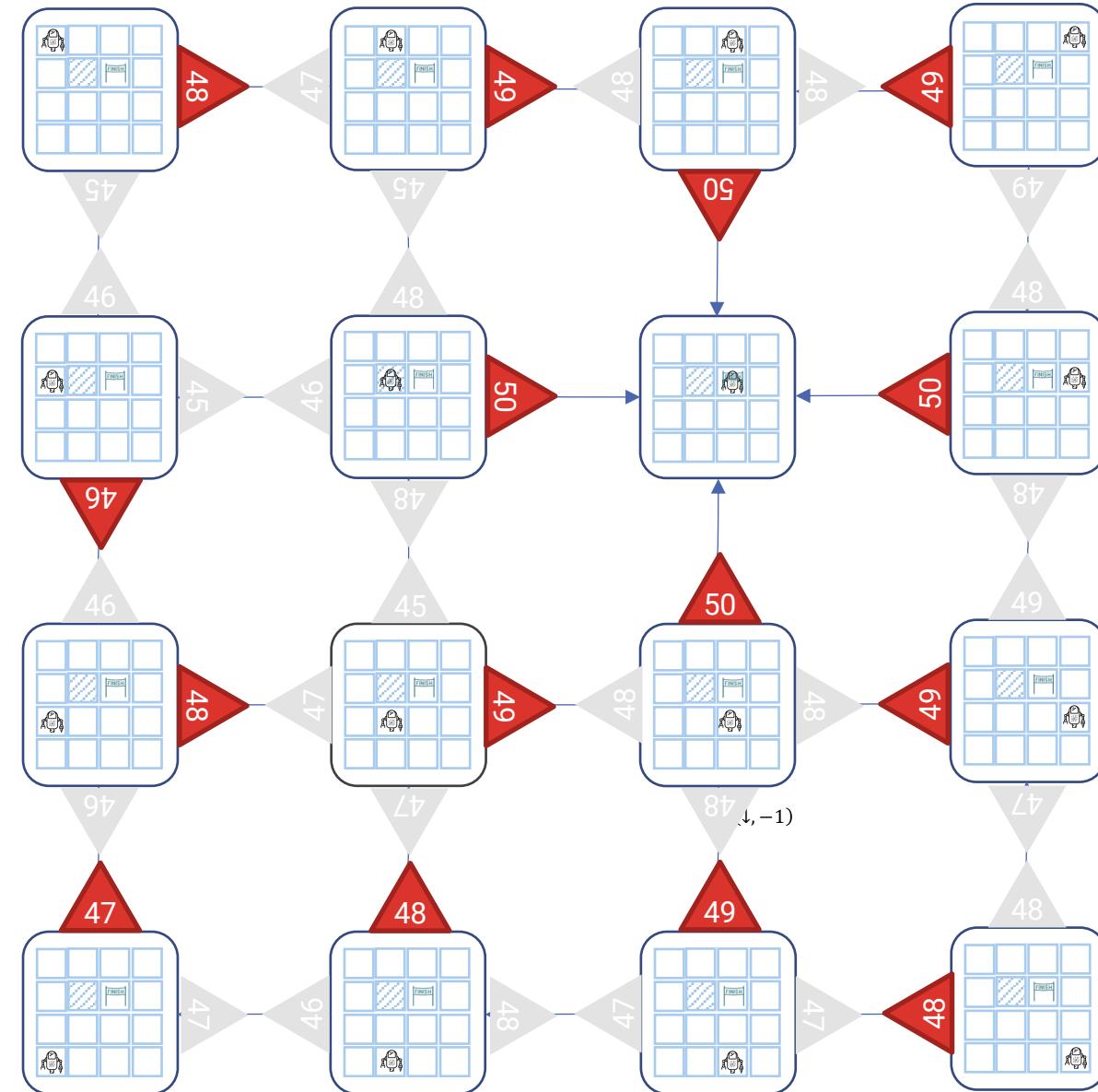
MDP

Formulación

$$q_{\pi_*}(s, a)$$

Resaltar valor máximo de cada estado:

$$\nu_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a)$$



MDP

Formulación

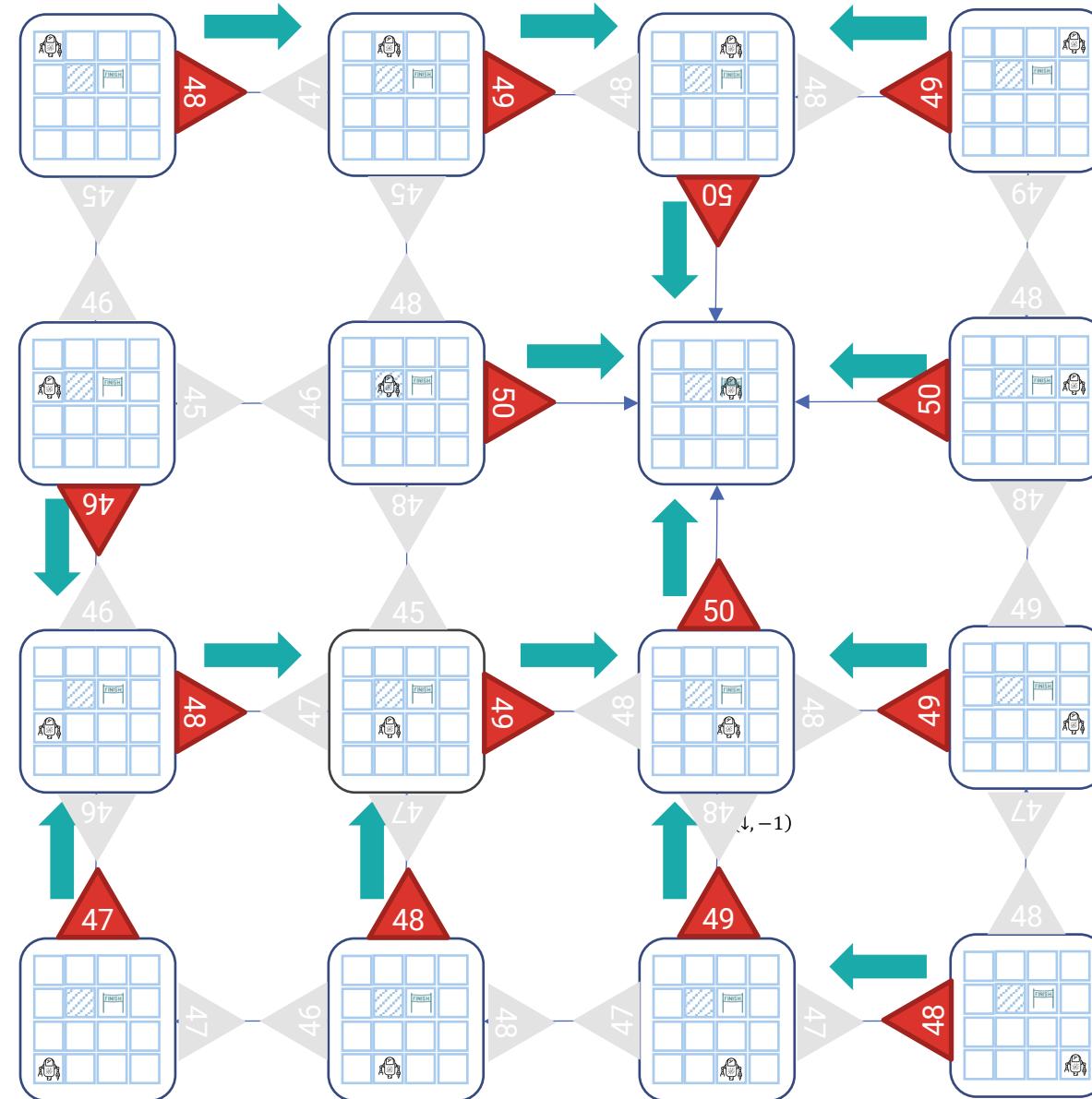
$$q_{\pi_*}(s, a)$$

Resaltar valor máximo de cada estado:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a)$$

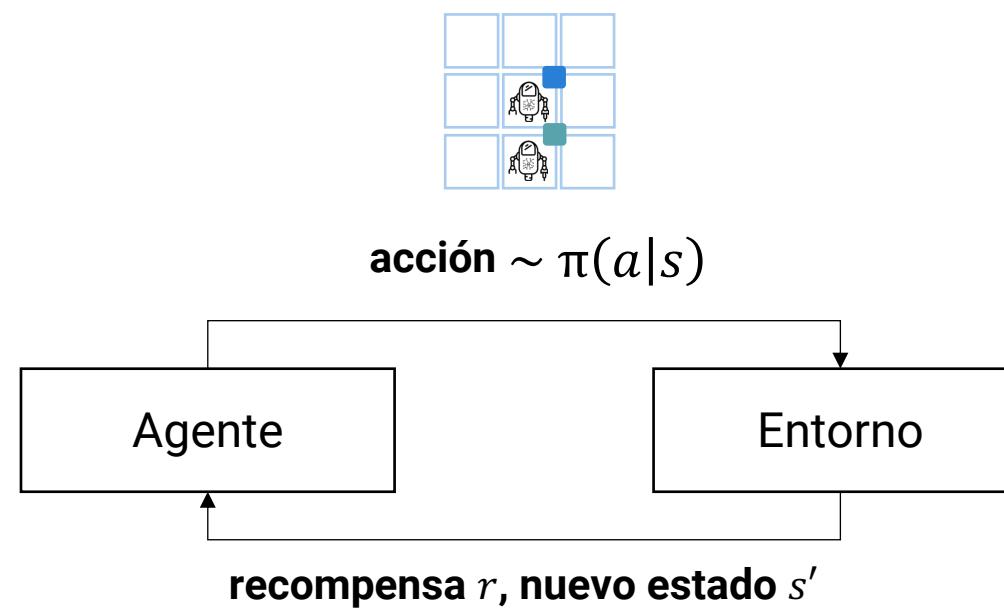
Reconstruir π_*

Estado	Acción
(3, 0)	→
(2, 0)	↑
...	

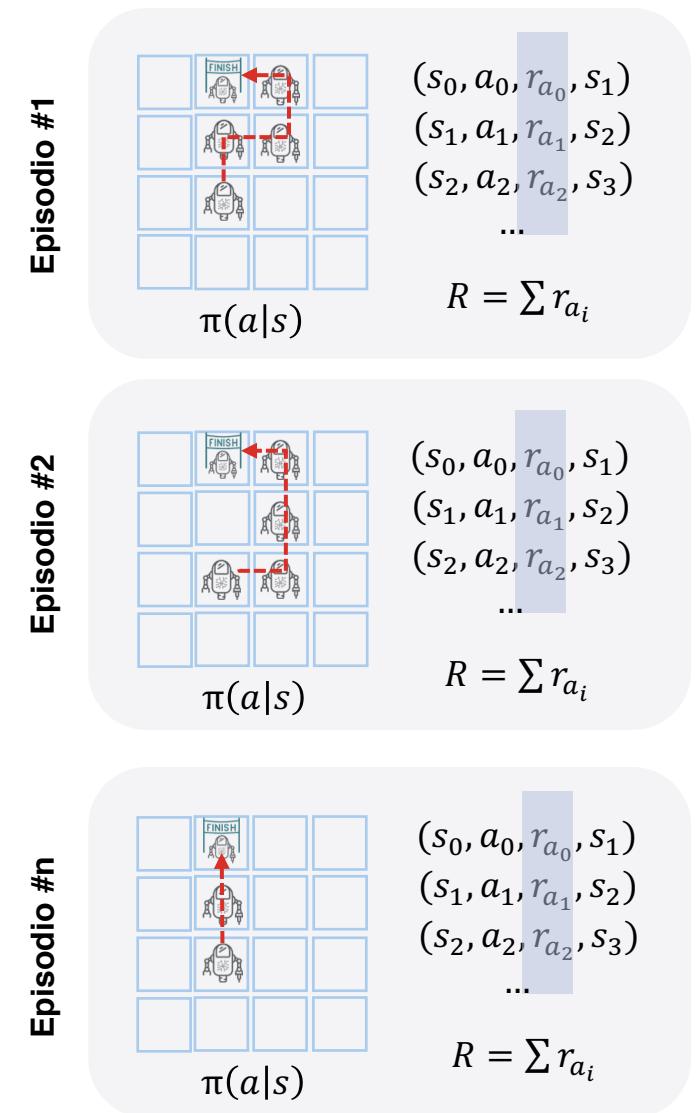


MDP

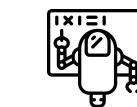
Aprendizaje por refuerzo



Episodios de entrenamiento



Estado	Acción	q
(3, 0)	↑	10
(2, 0)	→	12
...		...

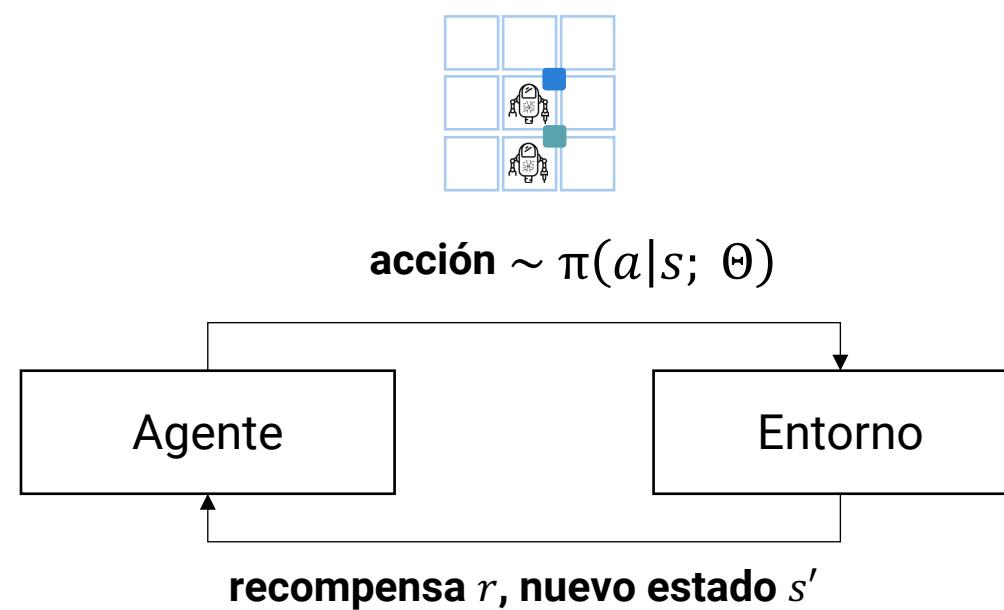


training

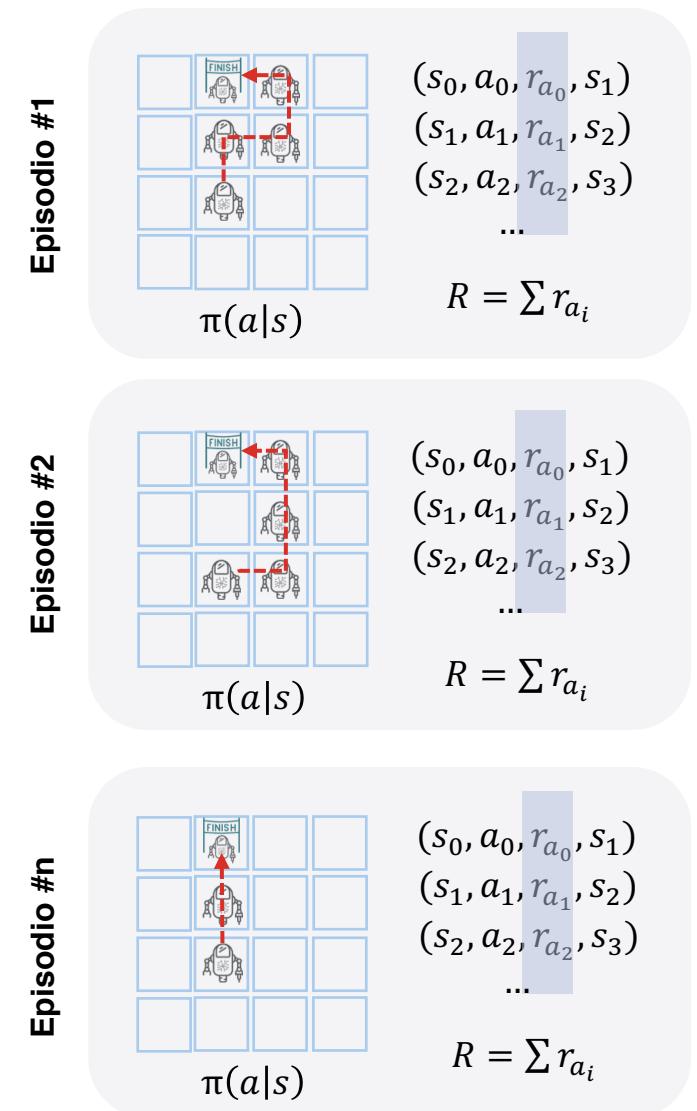
$$\pi' = \arg \max_{\pi} \sum_{t=0}^{\infty} \gamma^t * r_{a_t}(s_t, s_{t+1})$$

con $a_t \sim \pi(a|s)$

$\gamma \in [0, 1]$: tasa de descuento



Episodios de entrenamiento



training

$$\Theta' = \arg \max_{\Theta} \sum_{t=0}^{\infty} \gamma^t * r_{a_t}(s_t, s_{t+1})$$

con $a_t \sim \pi(s; \Theta)$

$\gamma \in [0, 1]$: tasa de descuento

2 Método de Montecarlo

FORMULACIÓN, IMPLEMENTACIÓN

Método de Montecarlo

Formulación



UNIVERSIDAD
DE GRANADA

Método de Montecarlo (con mejora incremental de política)

params: tabla Q inicial, política inicial π_0 , n_episodios

for i = {0, ..., n_episodios}

 generar episodio según π_i

 actualizar Q : $Q'(s, a) \approx Q(s, a) + \hat{q}_{\pi_i}(s, a)$

 obtener política mejorada π_{i+1} a partir de Q

Estado	Acción	Q
(3, 0)	↑	10
(3, 0)	→	12
(3, 1)	→	9
...		

Consideraciones

Si un par $\langle \text{estado}, \text{acción} \rangle$ se visita más de una vez en un episodio, se puede:

- Tomar la media de los valores G obtenidos (predicción *every visit*)
- Quedarse con el valor G de la primera visita (predicción *first visit*)

Para explorar más espacio, en lugar de π se utiliza ϵ -greedy(Q):

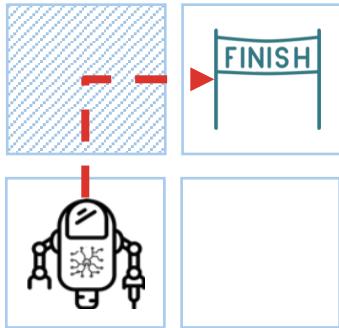
- con probabilidad ϵ se toma una acción aleatoria
- con probabilidad $1 - \epsilon$ se toma la acción de π

Método de Montecarlo

Ejemplo



UNIVERSIDAD
DE GRANADA



S_t	A_t	R_{t+1}
$<1, 0>$	\uparrow	-5
$<0, 0>$	\rightarrow	+50

```
for i_episode in range(1, num_episodes+1):
    # generar episodio
    episode = generate_episode(env, Q, epsilon)

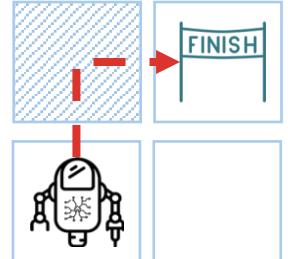
    # actualizar tabla Q
    # - obtener estados, acciones y recompensas del episodio
    states, actions, rewards = zip(*episode)
    # - obtener gamma para aplicar descuentos
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])
    # - actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion)
    for i, state in enumerate(states):
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])
        N[state][actions[i]] += 1.0
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
```

Método de Montecarlo

Ejemplo



UNIVERSIDAD
DE GRANADA



S_t	A_t	R_{t+1}
$<1, 0>$	\uparrow	-5
$<0, 0>$	\rightarrow	+50

```
for i_episode in range(1, num_episodes+1):

    # generar episodio
    episode = generate_episode(env, Q, epsilon)

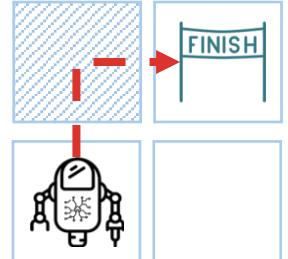
    # actualizar tabla Q
    # - obtener estados, acciones y recompensas del episodio
    states, actions, rewards = zip(*episode)
    # - obtener gamma para aplicar descuentos
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])
    # - actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion)
    for i, state in enumerate(states):
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])
        N[state][actions[i]] += 1.0
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
```

Método de Montecarlo

Ejemplo



UNIVERSIDAD
DE GRANADA



$$\gamma = 0.9$$

discounts

γ^0	γ^1
1	0.9

```
for i_episode in range(1, num_episodes+1):  
  
    # generar episodio  
    episode = generate_episode(env, Q, epsilon)  
  
    # actualizar tabla Q  
    # - obtener estados, acciones y recompensas del episodio  
    states, actions, rewards = zip(*episode)  
    # - obtener gamma para aplicar descuentos  
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])  
    # - actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion)  
    for i, state in enumerate(states):  
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])  
        N[state][actions[i]] += 1.0  
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
```

Método de Montecarlo

Ejemplo



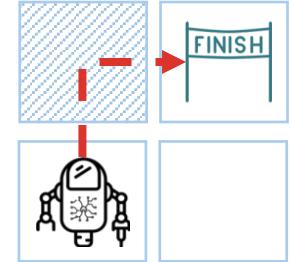
UNIVERSIDAD
DE GRANADA

S_t	A_t	R_{t+1}
<1, 0>	↑	-5
<0, 0>	→	+50

discounts

γ^0
 γ^1

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$



S_t	A_t			
	↑	→	↓	←
<0, 0>				
<0, 1>				
<1, 0>				
<1, 1>				

```

for i_episode in range(1, num_episodes+1):

    # generar episodio
    episode = generate_episode(env, Q, epsilon)

    # actualizar tabla Q
    # - obtener estados, acciones y recompensas del episodio
    states, actions, rewards = zip(*episode)
    # - obtener gamma para aplicar descuentos
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])
    # - actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion)
    for i, state in enumerate(states):
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])
        N[state][actions[i]] += 1.0
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
    
```

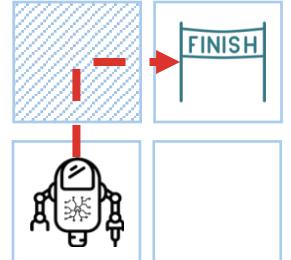
sum([-5, +50] • [1, 0.9]) = 40

Método de Montecarlo

Ejemplo



UNIVERSIDAD
DE GRANADA



S_t	Q	A_t			
		↑	→	↓	←
	<0, 0>				
	<0, 1>				
	<1, 0>	40			
	<1, 1>				

```
for i_episode in range(1, num_episodes+1):  
  
    # generar episodio  
    episode = generate_episode(env, Q, epsilon)  
  
    # actualizar tabla Q  
    # - obtener estados, acciones y recompensas del episodio  
    states, actions, rewards = zip(*episode)  
    # - obtener gamma para aplicar descuentos  
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])  
    # - actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion)  
    for i, state in enumerate(states):  
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])  
        N[state][actions[i]] += 1.0  
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
```

Método de Montecarlo

Ejemplo

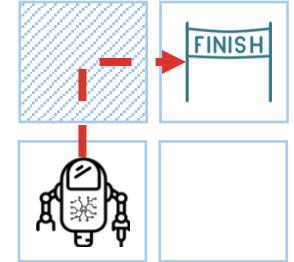


UNIVERSIDAD
DE GRANADA

S_t	A_t	R_{t+1}
$<1, 0>$	\uparrow	-5
$<0, 0>$	\rightarrow	+50

discounts
 γ^0

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$



S_t	A_t			
	\uparrow	\rightarrow	\downarrow	\leftarrow
$<0, 0>$				
$<0, 1>$				
$<1, 0>$	40			
$<1, 1>$				

```

for i_episode in range(1, num_episodes+1):

    # generar episodio
    episode = generate_episode(env, Q, epsilon)

    # actualizar tabla Q
    # - obtener estados, acciones y recompensas del episodio
    states, actions, rewards = zip(*episode)
    # - obtener gamma para aplicar descuentos
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])
    # - actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion)
    for i, state in enumerate(states):
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])
        N[state][actions[i]] += 1.0
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
    
```

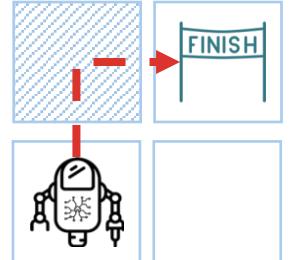
sum([+50] • [1]) = 50

Método de Montecarlo

Ejemplo



UNIVERSIDAD
DE GRANADA



S_t	A_t	Q			
		↑	→	↓	←
	<0, 0>		50		
	<0, 1>				
	<1, 0>	40			
	<1, 1>				

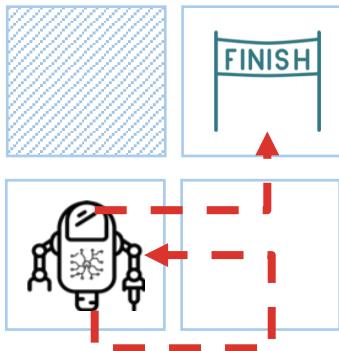
```
for i_episode in range(1, num_episodes+1):  
  
    # generar episodio  
    episode = generate_episode(env, Q, epsilon)  
  
    # actualizar tabla Q  
    # - obtener estados, acciones y recompensas del episodio  
    states, actions, rewards = zip(*episode)  
    # - obtener gamma para aplicar descuentos  
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])  
    # - actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion)  
    for i, state in enumerate(states):  
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])  
        N[state][actions[i]] += 1.0  
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
```

Método de Montecarlo

Ejemplo



UNIVERSIDAD
DE GRANADA



S_t	A_t	R_{t+1}
$<1, 0>$	\rightarrow	-1
$<1, 1>$	\leftarrow	-1
$<1, 0>$	\rightarrow	-1
$<1, 1>$	\uparrow	+50

```
for i_episode in range(1, num_episodes+1):
    # generar episodio
    episode = generate_episode(env, Q, epsilon)

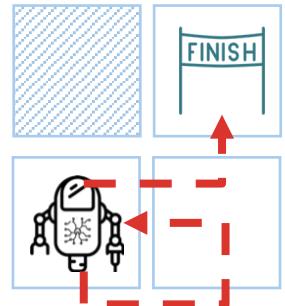
    # actualizar tabla Q
    # - obtener estados, acciones y recompensas del episodio
    states, actions, rewards = zip(*episode)
    # - obtener gamma para aplicar descuentos
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])
    # - actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion)
    for i, state in enumerate(states):
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])
        N[state][actions[i]] += 1.0
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
```

Método de Montecarlo

Ejemplo



UNIVERSIDAD
DE GRANADA



S_t	Q	A_t			
		↑	→	↓	←
	<0, 0>		50		
	<0, 1>				
	<1, 0>	40	38.9		
	<1, 1>	50			38.6

```
for i_episode in range(1, num_episodes+1):
    # generar episodio
    episode = generate_episode(env, Q, epsilon)

    # actualizar tabla Q
    # - obtener estados, acciones y recompensas del episodio
    states, actions, rewards = zip(*episode)
    # - obtener gamma para aplicar descuentos
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])
    # - actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion)
    for i, state in enumerate(states):
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])
        N[state][actions[i]] += 1.0
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
```



3 Q-Learning

FORMULACIÓN, IMPLEMENTACIÓN

Q-Learning

Limitaciones del método de Montecarlo



UNIVERSIDAD
DE GRANADA

En el método de Montecarlo, Q se actualiza en cada episodio:

- Una acción queda *diluida* entre todas las de su episodio

Recordemos que: $q_{\pi}(s, a) \doteq \mathbb{E}_{\pi} \left[\sum_{k=0}^T \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$

El valor estimado para una acción depende mucho de cómo continúe el episodio

- Una acción queda *diluida* entre las realizadas en todos los episodios

Recordemos que la actualización sobre Q considera valores de todos los episodios anteriores

```
for i_episode in range(1, num_episodes+1):
    episode = generate_episode(env, Q, epsilon, render)
    states, actions, rewards = zip(*episode)
    discounts = np.array([gamma**i for i in range(len(rewards)+1)])
    
    # actualizar suma de recompensa, numero de visitas y Q para cada (estado, accion) del episodio
    for i, state in enumerate(states):
        returns_sum[state][actions[i]] += sum(rewards[i:]*discounts[:-1+i])
        N[state][actions[i]] += 1.0
        Q[state][actions[i]] = returns_sum[state][actions[i]] / N[state][actions[i]]
```

Montecarlo

El algoritmo puede tardar mucho en converger

Q-Learning

Formulación

Métodos de diferencia temporal

1. Consideran un paso dentro del episodio
2. Actualizan Q después de cada paso con:
recompensa inmediata (1 paso) [dada por el entorno] +
estimación recompensa futura ($t-1$ pasos) [dada por Q ; i.e. *bootstrapping*]
3. Ponderan la importancia de la recompensa inmediata respecto a la
recompensa futura

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

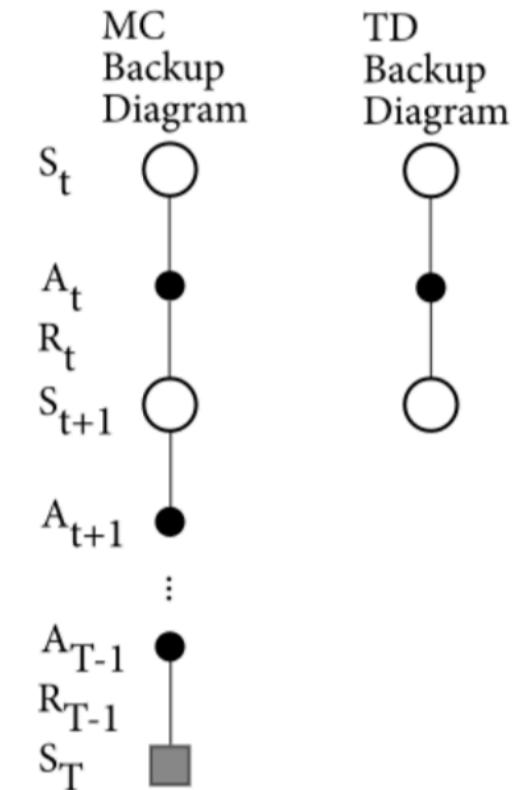
$S_0 A_0 R_1 S_1$ | $A_1 R_2 S_2$ |

$$\rightarrow Q(S_0, A_0) \leftarrow Q(S_0, A_0) + \alpha \left(R_1 + \gamma \max_{a \in \mathcal{A}} Q(S_1, a) - Q(S_0, A_0) \right)$$

$\pi \leftarrow \epsilon_0\text{-greedy}(Q)$

$$Q(S_1, A_1) \leftarrow Q(S_1, A_1) + \alpha \left(R_2 + \gamma \max_{a \in \mathcal{A}} Q(S_2, a) - Q(S_1, A_1) \right)$$

$\pi \leftarrow \epsilon_0\text{-greedy}(Q)$



MC update equation:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

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

TD update equation:

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

M. Morales (2019) **Grokking Reinforcement Learning**. Manning



Q-Learning

Formulación



UNIVERSIDAD
DE GRANADA

Q-Learning (SARSA-max)

params: Q inicial, n_episodios, α

for i = {0, ..., n_episodios}

 while *episodio no terminado*

 elegir acción A_t con política ϵ -greedy(Q)

 aplicar A_t y obtener R_{t+1}, S_{t+1}

 Actualizar Q

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

Índice

1. Procesos de Decisión de Markov (MDP)
2. Método de Montecarlo
3. Q-Learning

Aprendizaje por refuerzo

Temas avanzados

Otras variantes de diferencia temporal

SARSA, Expected SARSA, Double Q-Learning

Demostraciones de *optimalidad* y convergencia de los métodos



C. Watkins, P. Dayan (1992) Q-Learning. Machine Learning 8, 279 – 292 [[link](#)]

Discretización en problemas con espacio de estados continuo

Discretización en problemas con espacio de acciones continuo

Recompensas dispersas (*sparse rewards*)

Planificación y aprendizaje

Random-sample one-step tabular Q-planning

Montecarlo Tree Search (MCTS)



UNIVERSIDAD
DE GRANADA