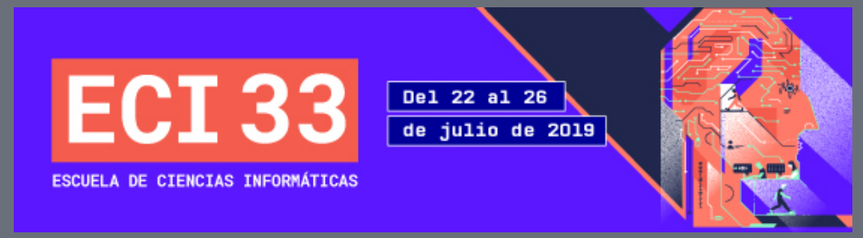
Aprendizaje Profundo por Refuerzo
03. Aprendizaje por refuerzo

Dr. Juan Gómez Romero
Investigador Senior
Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada

UBA
Universidad de Buenos Aires
Argentina virtus robur et studium

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 MDP

PROCESOS DE DECISIÓN DE MARKOV

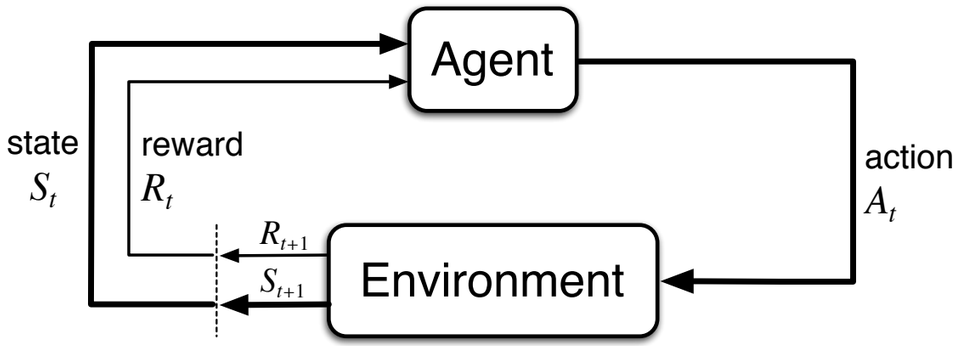
MDP

Introducción

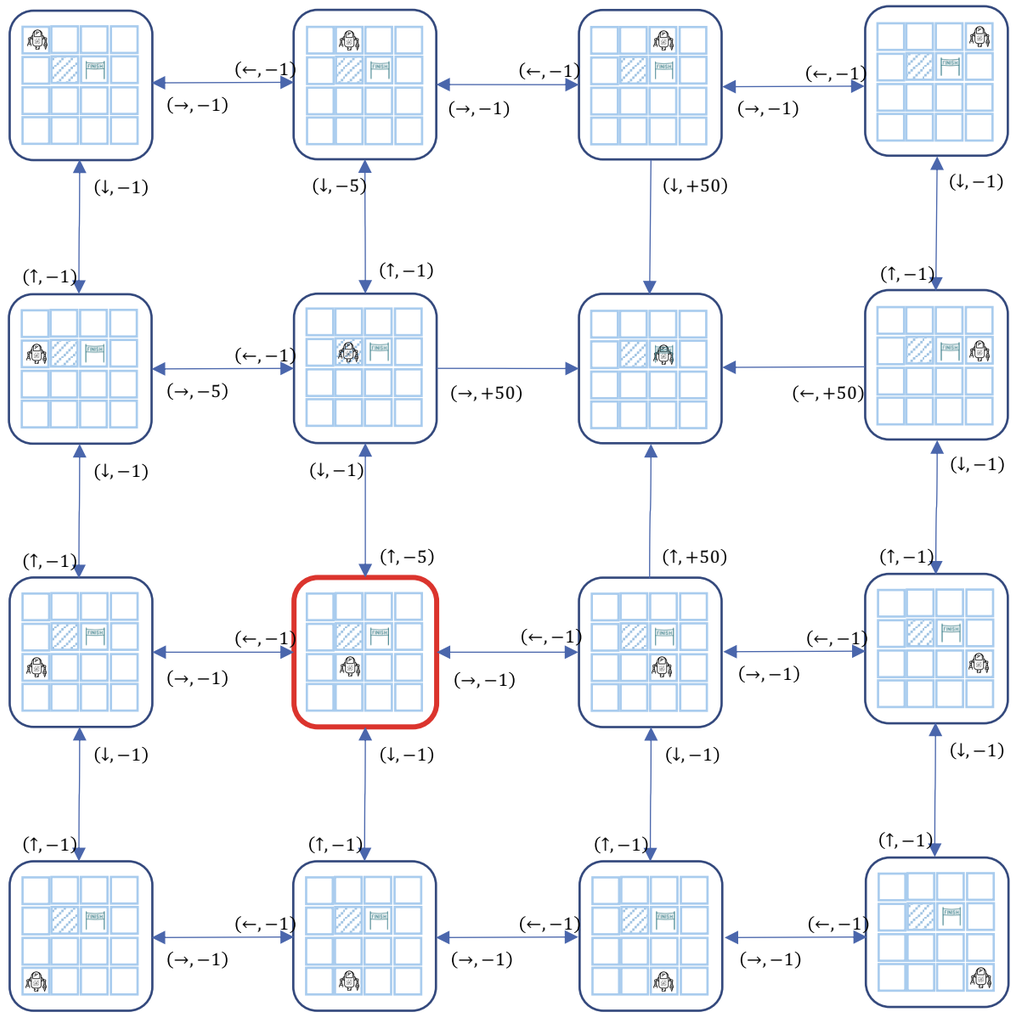
Aprendizaje por refuerzo

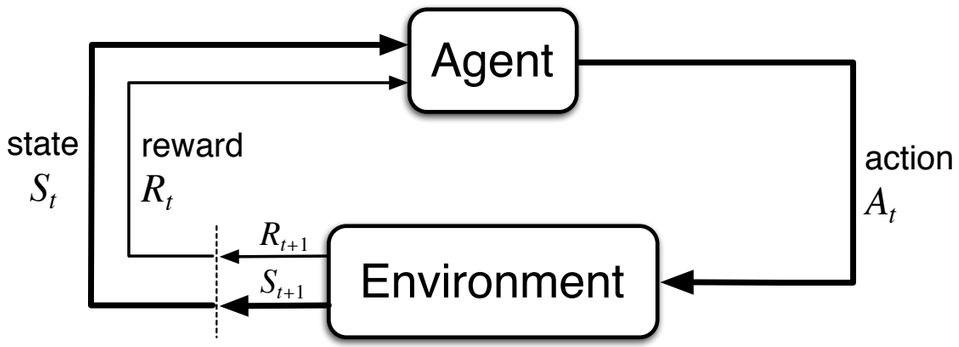
Problema de búsqueda en el grafo de estados y transiciones.

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: $0, 1, 2, \ldots$

estado: $S_t \in \mathcal{S}$

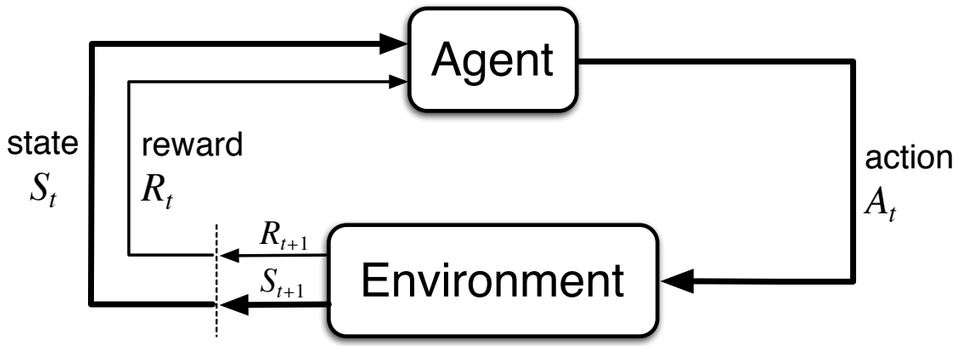
acción: $A_t \in \mathcal{A}(S_t)$

recompensa: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

estado siguiente: $S_{t+1} \in \mathcal{S}$

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

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



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

Markov Decision Process - MDP

Recompensa acumulada:

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

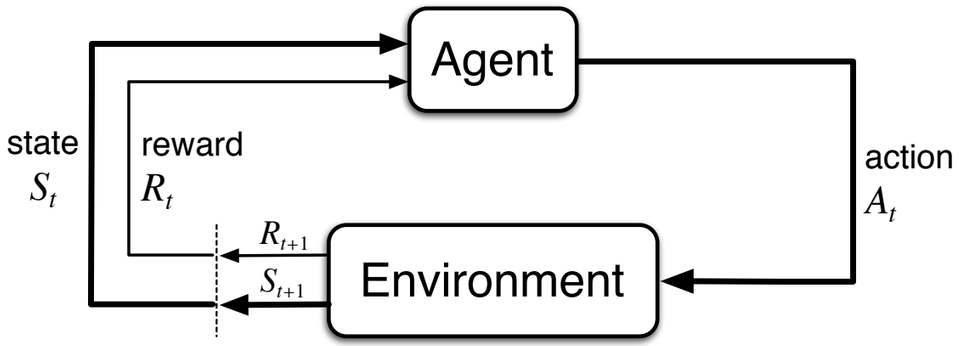
Recompensa con descuento:

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

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.

Markov Decision Process - MDP

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

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

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

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

Comparación de políticas de actuación y política óptima

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

$\pi_* \geq \pi' \; \forall \pi'$ *(garantizada)*

MDP

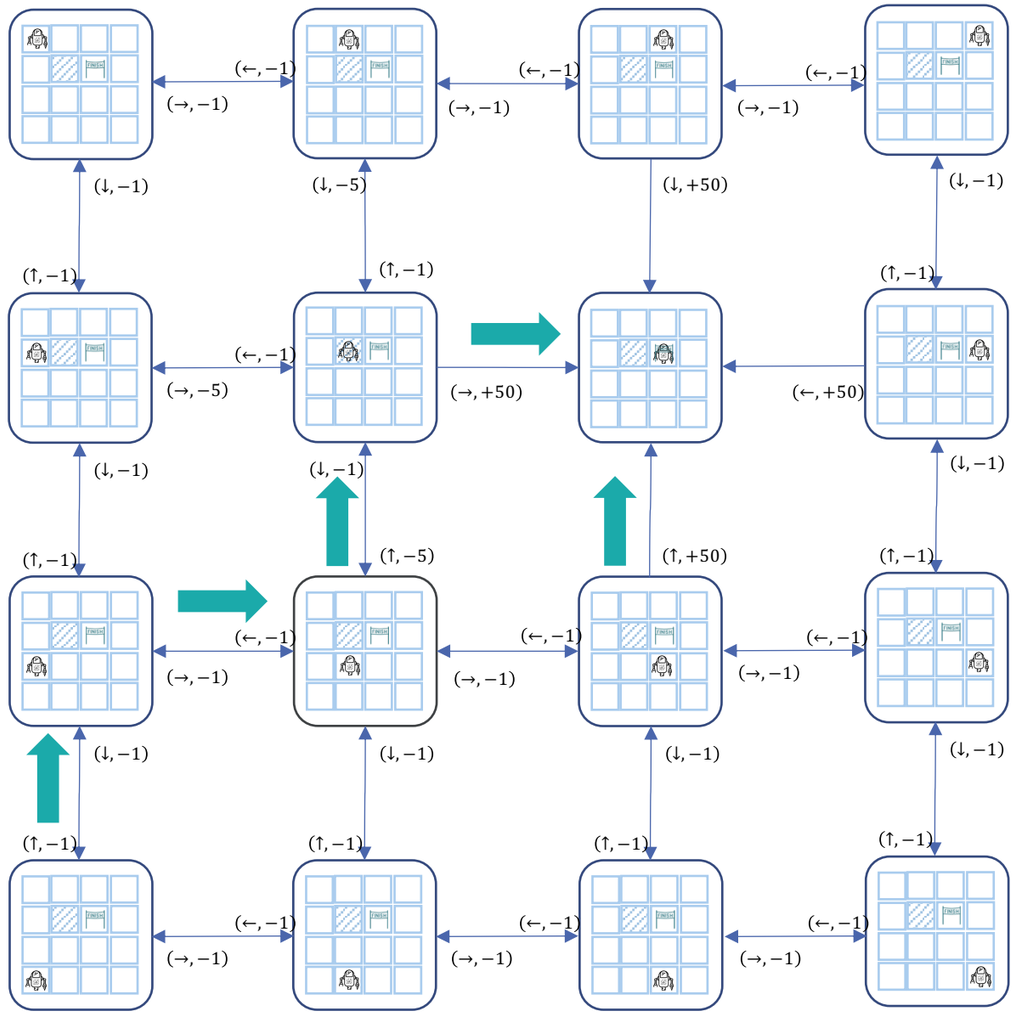
Formulación

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

Política π_1 $(\gamma = 1)$:

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

$$v_{\pi_1}(\langle 2,1\rangle) = -5 + 50 = 45$$
$$v_{\pi_1}(\langle 2,2\rangle) = 50$$
$$v_{\pi_1}(\langle 3,0\rangle) = -1 - 1 - 5 + 50 = 43$$
$$v_{\pi_1}(\langle 2,0\rangle) = -1 - 5 + 50 = 42$$



MDP

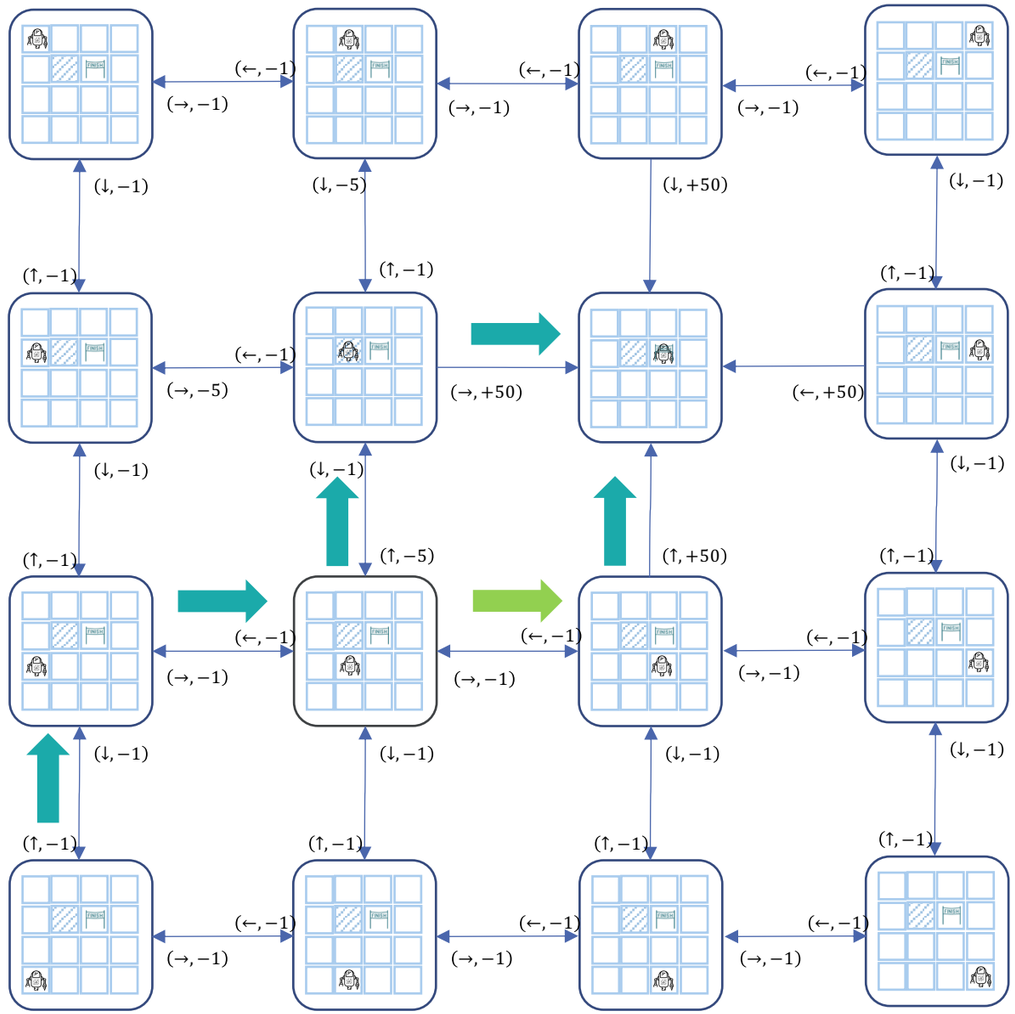
Formulación

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

Política π_2 $(\gamma = 1)$:

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

$$v_{\pi_2}(\langle 2,1\rangle) = -1 + 50 = 49 \geq 45$$
$$v_{\pi_2}(\langle 2,2\rangle) = 50 \geq 50$$
$$v_{\pi_2}(\langle 3,0\rangle) = -1 - 1 - 1 + 50 = 47 \geq 43$$
$$v_{\pi_2}(\langle 2,0\rangle) = -1 - 1 + 50 = 48 \geq 42$$



MDP

Formulación

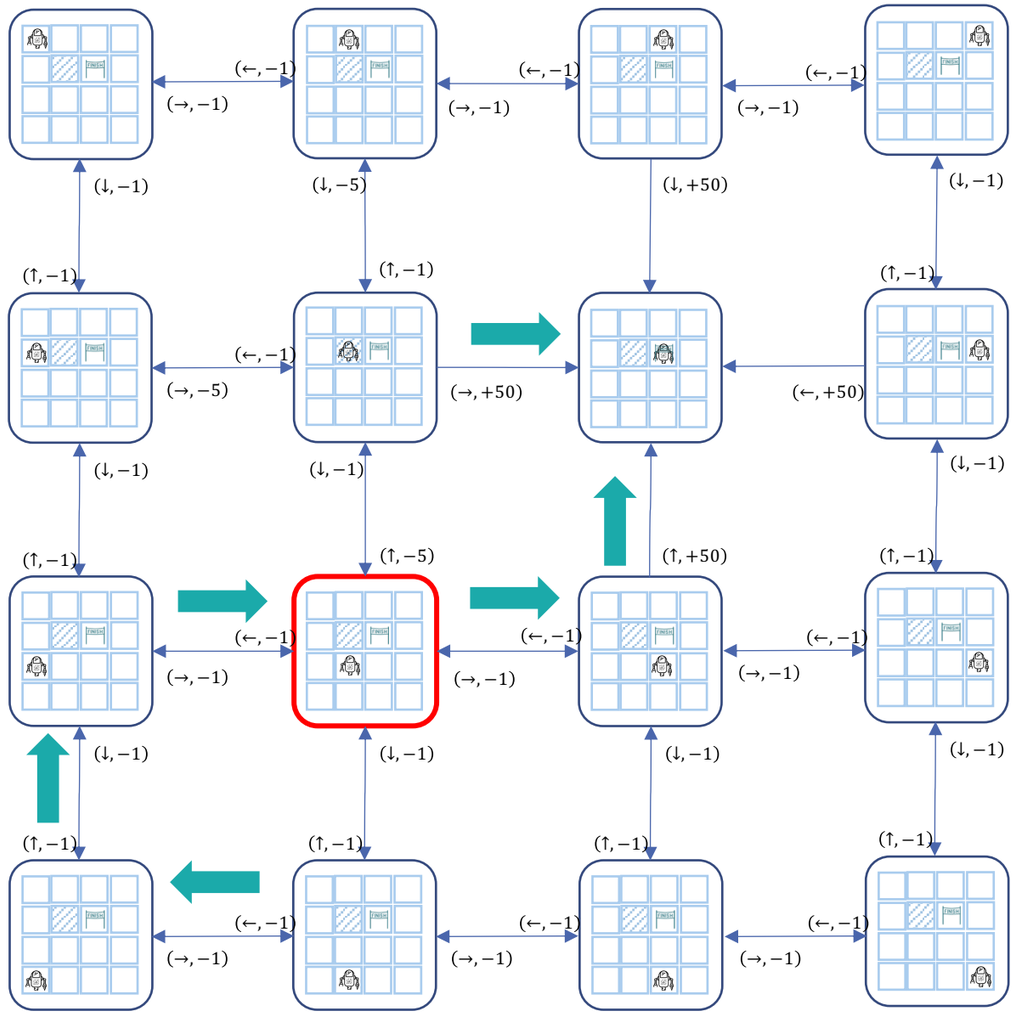
$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \;\middle|\; S_t = s, A_t = a\right]$$

Política π_2 $(\gamma = 1)$:

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

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

$q_{\pi_2}(\langle 2,1\rangle, \uparrow) = -5 + 50 = 45$
$q_{\pi_2}(\langle 2,1\rangle, \rightarrow) = -1 + 50 = \mathbf{49}$
$q_{\pi_2}(\langle 2,1\rangle, \leftarrow) = -1 - 1 - 1 + 50 = 47$
$q_{\pi_2}(\langle 2,1\rangle, \downarrow) = -1 - 1 - 1 - 1 - 1 + 50 = 45$



MDP

Formulación

UNIVERSIDAD DE GRANADA

Ecuación de Bellman (recursividad de v_π)

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

recompensa al realizar acción — *valor del siguiente estado*

Ecuación de Bellman (optimalidad de v_*)

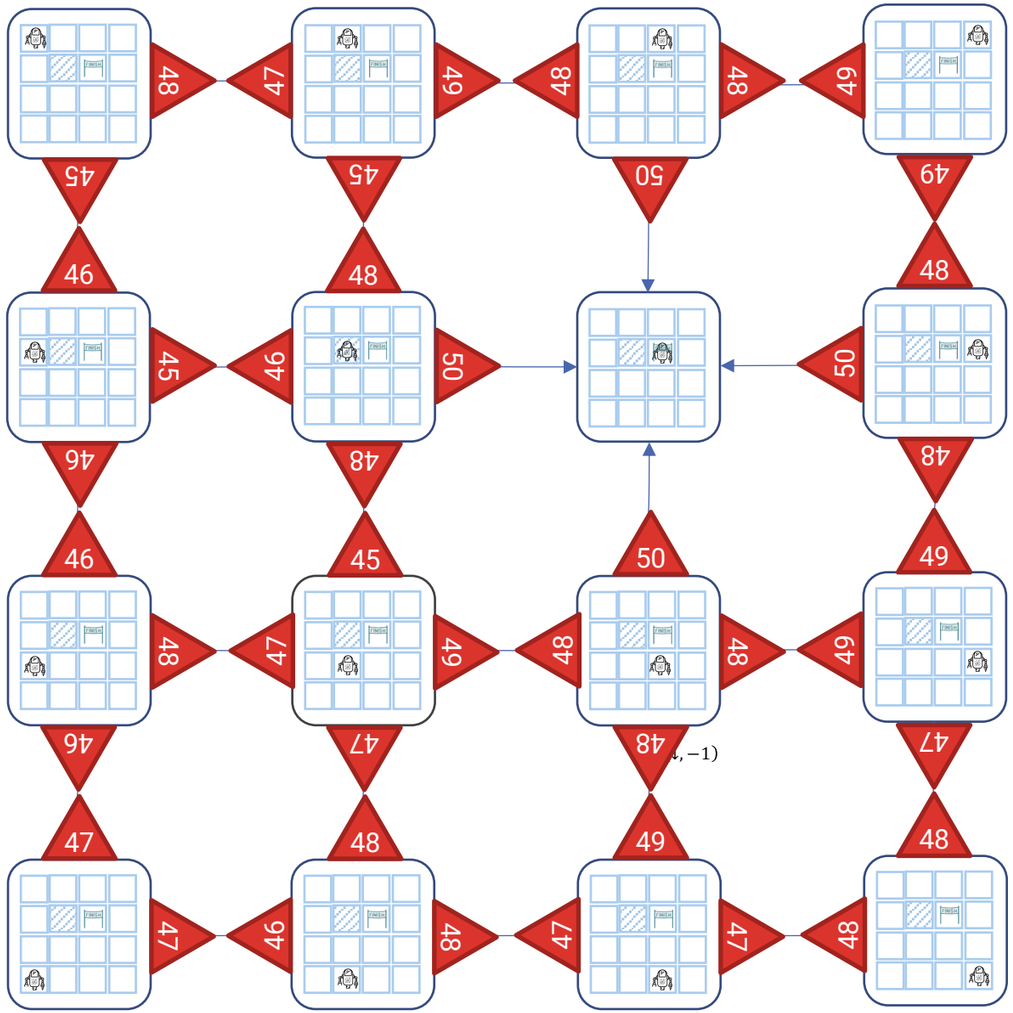
$$v_{\pi_*}(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a)$$

valoración de un estado (política óptima) — *valoración de la mejor acción desde ese estado (política óptima)*

MDP

Formulación

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



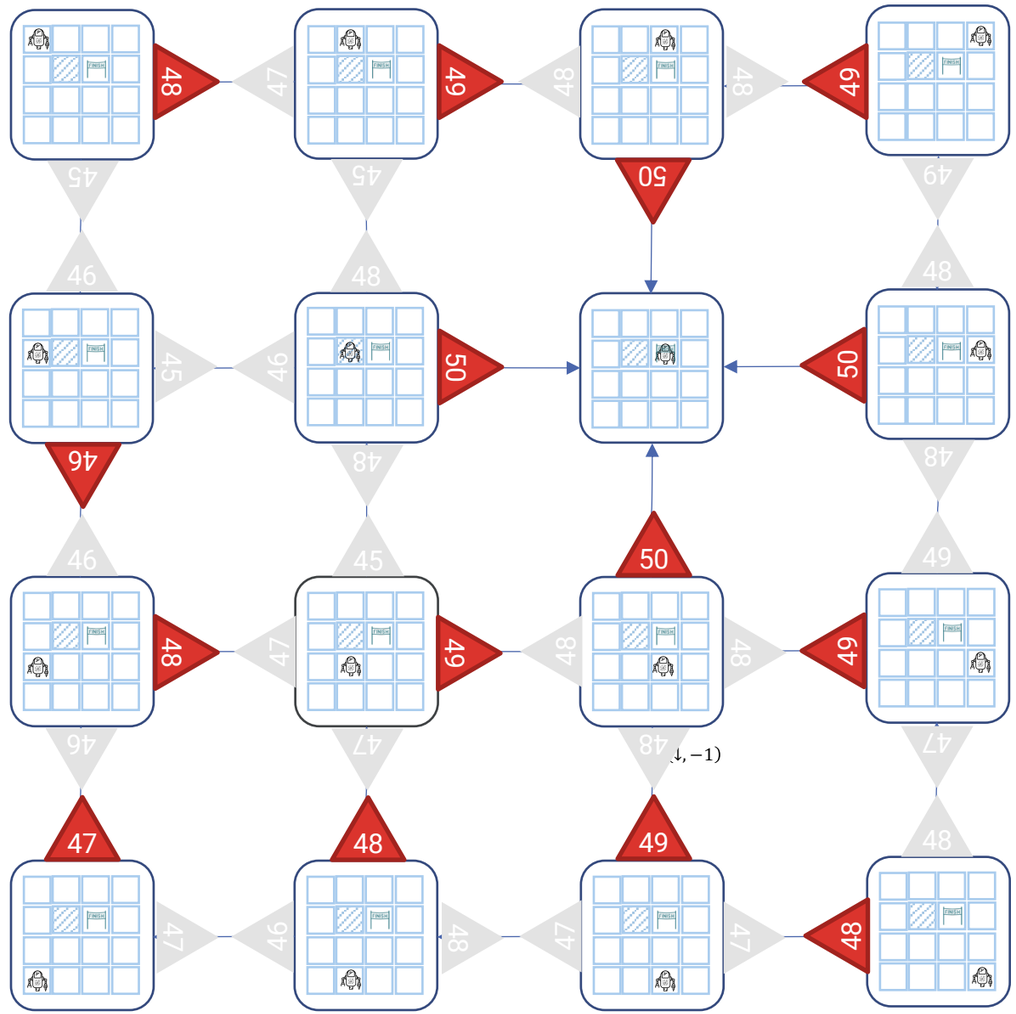
MDP

Formulación

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

Resaltar valor máximo de cada estado:

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



MDP

Formulación

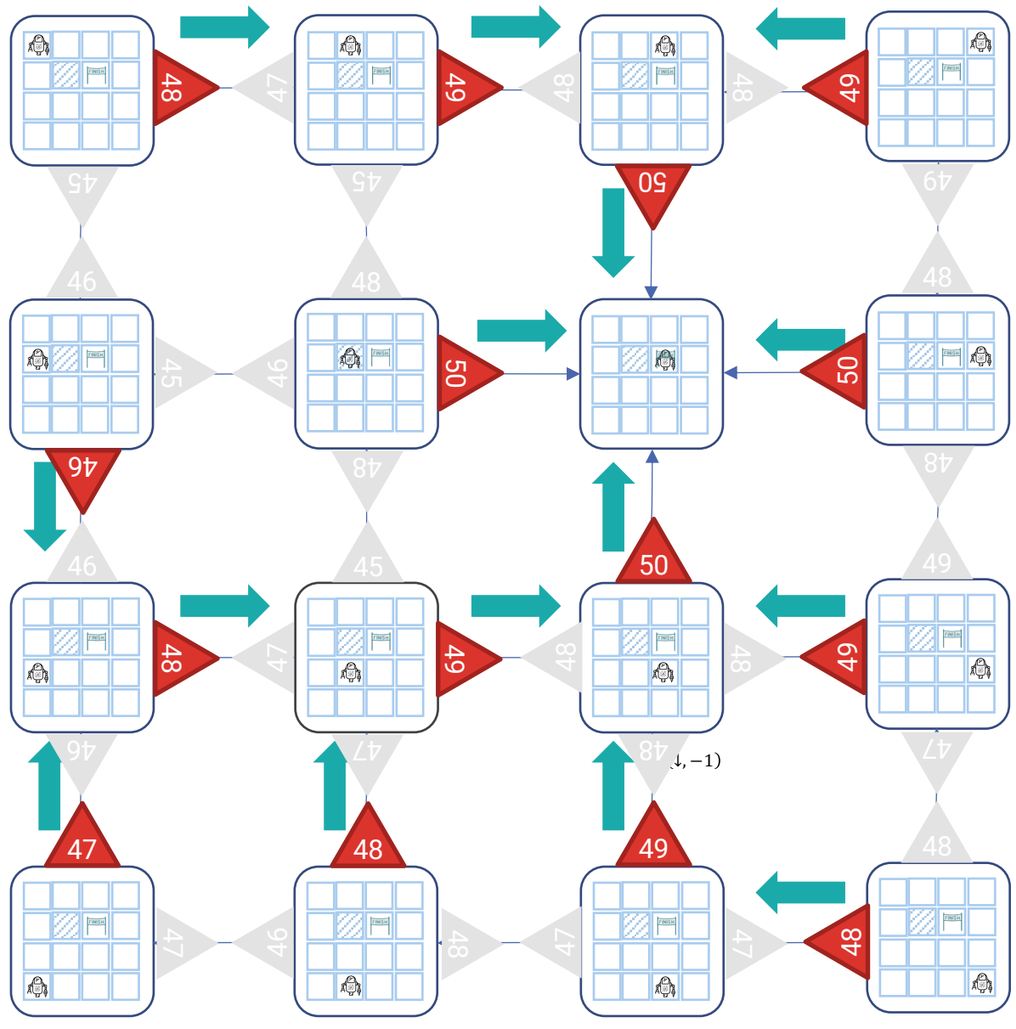
$$q_{\pi_*}(s,a)$$

Resaltar valor máximo de cada estado:

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

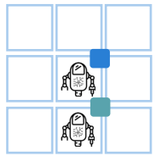
Reconstruir π_*

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

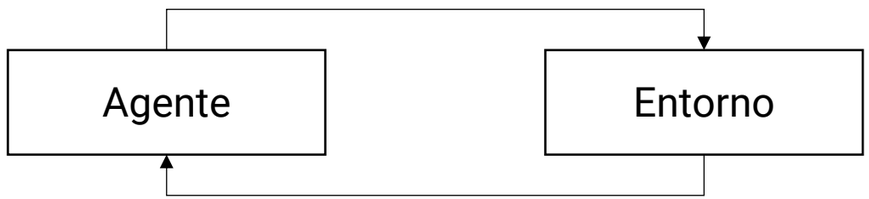


MDP

Aprendizaje por refuerzo



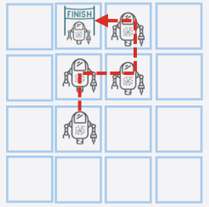
acción $\sim \pi(a|s)$



recompensa r**, nuevo estado** s'

Episodios de entrenamiento

Episodio #1

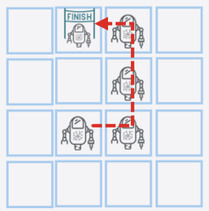


$\pi(a|s)$

(s_0, a_0, r_{a_0}, s_1)
(s_1, a_1, r_{a_1}, s_2)
(s_2, a_2, r_{a_2}, s_3)
...

$R = \sum r_{a_i}$

Episodio #2

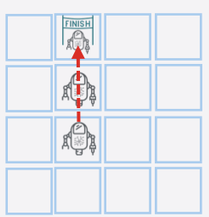


$\pi(a|s)$

(s_0, a_0, r_{a_0}, s_1)
(s_1, a_1, r_{a_1}, s_2)
(s_2, a_2, r_{a_2}, s_3)
...

$R = \sum r_{a_i}$

Episodio #n



$\pi(a|s)$

(s_0, a_0, r_{a_0}, s_1)
(s_1, a_1, r_{a_1}, s_2)
(s_2, a_2, r_{a_2}, s_3)
...

$R = \sum r_{a_i}$

training

Estado	Acción	q
$(3, 0)$	$\uparrow$	10
$(2, 0)$	$\rightarrow$	12
...		...

training

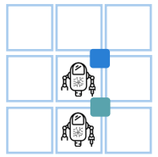
$$\pi' = \arg\max_{\pi} \sum_{t=0} \gamma^t * r_{a_t}(s_t, s_{t+1})$$

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

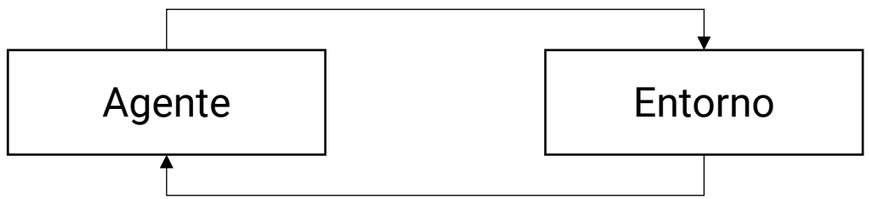
$\gamma \in [0, 1]$: tasa de descuento

MDP

Aprendizaje por refuerzo



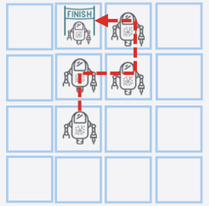
acción $\sim \pi(a|s;\ \Theta)$



recompensa r**, nuevo estado** s'

Episodios de entrenamiento

Episodio #1

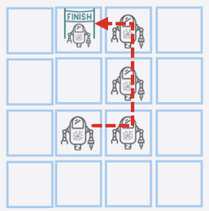


(s_0, a_0, r_{a_0}, s_1)
(s_1, a_1, r_{a_1}, s_2)
(s_2, a_2, r_{a_2}, s_3)
...

$\pi(a|s)$

$R = \sum r_{a_i}$

Episodio #2

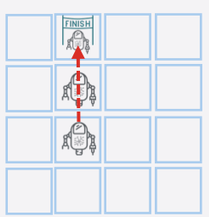


(s_0, a_0, r_{a_0}, s_1)
(s_1, a_1, r_{a_1}, s_2)
(s_2, a_2, r_{a_2}, s_3)
...

$\pi(a|s)$

$R = \sum r_{a_i}$

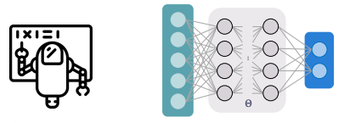
Episodio #n



(s_0, a_0, r_{a_0}, s_1)
(s_1, a_1, r_{a_1}, s_2)
(s_2, a_2, r_{a_2}, s_3)
...

$\pi(a|s)$

$R = \sum r_{a_i}$



training

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

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
$(3,0)$	$\uparrow$	10
$(3,0)$	$\rightarrow$	12
$(3,1)$	$\rightarrow$	9
…		

Consideraciones

Si un par *<estado, acción>* se visita más de una vez en un episodio, se puede:

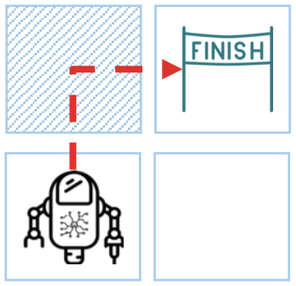
- Tomar la media de los valores G obtenidos (predicción *every visit*)
- Quedarse con el valor G de la primera visita (predicción *first visit*)

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



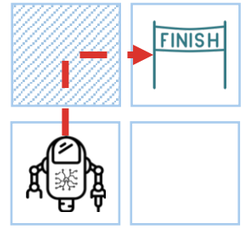
S_t	A_t	R_{t+1}
<1, 0>	↑	−5
<0, 0>	→	+50

```
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



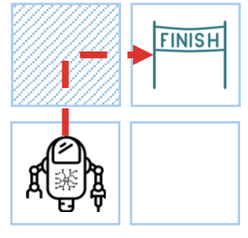
S_t	A_t	R_{t+1}
<1, 0>	↑	−5
<0, 0>	→	+50

```python
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

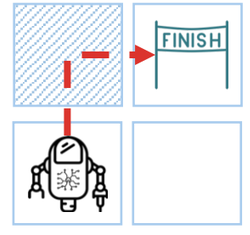
S_t	A_t	R_{t+1}
<1, 0>	↑	−5
<0, 0>	→	+50

discounts

γ^0

γ^1

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



returns_sum

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

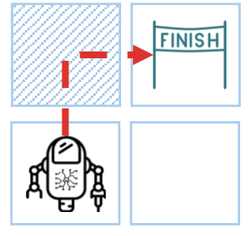
sum([−5, +50] • [1, 0.9]) = **40**

```
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



Q		A_t			
		↑	→	↓	←
S_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

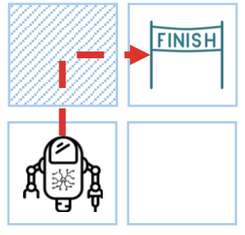
Ejemplo

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

discounts

γ^0

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



returns_sum

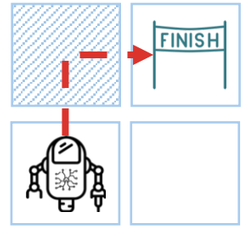
S_t	A_t ↑	→	↓	←
<0, 0>				
<0, 1>				
<1, 0>	**40**			
<1, 1>				

sum([+50] • [1]) = **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

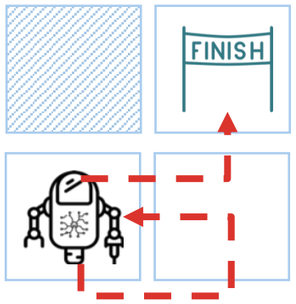


Q		A_t			
		↑	→	↓	←
S_t	<0, 0>		**50**		
	<0, 1>				
	<1, 0>	**40**			
	<1, 1>				

```python
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



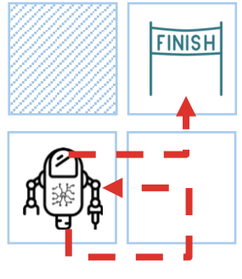
S_t	A_t	R_{t+1}
<1, 0>	→	−1
<1, 1>	←	−1
<1, 0>	→	−1
<1, 1>	↑	+50

```
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



Q		A_t			
		↑	→	↓	←
S_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

Montecarlo

```
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]]
```

El algoritmo puede tardar mucho en converger

Q-Learning

Formulación

UNIVERSIDAD DE GRANADA

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

$\pi \leftarrow \epsilon$–greedy(Q)

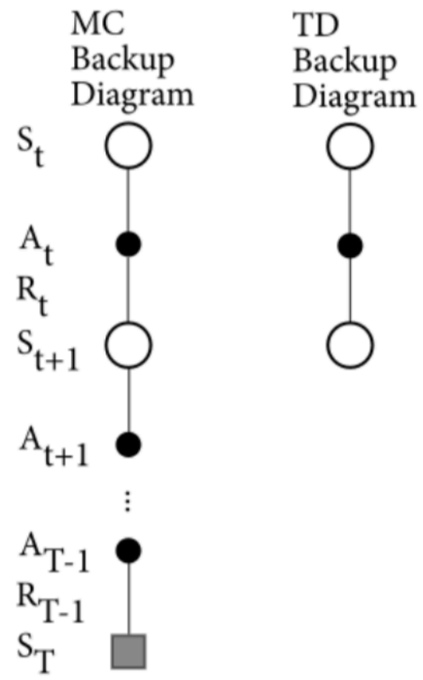
$S_0\ A_0\ R_1\ S_1 \quad | \quad A_1\ R_2\ S_2 \quad |$

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

$\pi \leftarrow \epsilon_0$–greedy$(Q)$

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

$\pi \leftarrow \epsilon_0$–greedy$(Q)$



MC update equation:
$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$
$V(S_t) = V(S_t) + \alpha(G_t - V(S_t))$

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

Aprendizaje por refuerzo

Temas avanzados

UNIVERSIDAD DE GRANADA

Otras variantes de diferencia temporal

SARSA, Expected SARSA, Double Q-Learning

Demostraciones de *optimalidad* y convergencia de los métodos

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)