

# Aprendizaje por refuerzo

Métodos basados en muestreo (3)

---

Antonio Manjavacas Lucas

manjavacas@ugr.es

1. Actualización incremental
2. *TD-learning*
3. SARSA
4. *Q-learning*
5. *Expected SARSA*
6. Control *n-step*

# Actualización incremental

---

# Actualización incremental

Cuando estudiamos los problemas tipo *bandits*, vimos en qué consistía una **regla de actualización incremental**:

$$\text{valorEstimado}' \leftarrow \text{valorEstimado} + \underbrace{\alpha}_{\text{step size}} \cdot \underbrace{[\text{objetivo} - \text{valorEstimado}]}_{\text{error de estimación}}$$

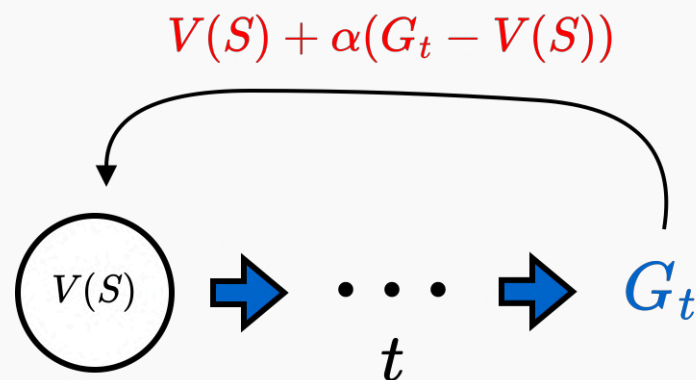
- El **error de estimación** se reduce a medida que las **estimaciones** se acercan al **objetivo**.
- Determina **cuánto** nos hemos equivocado en nuestra estimación más reciente.

# Actualización incremental

En los métodos **Monte Carlo**, podemos aplicar esta regla de actualización en la estimación de  $v_\pi, q_\pi$ .

Por ejemplo, la estimación  $V \approx v_\pi$  se aproximaría tal que:

$$V(S_t) \leftarrow V(S_t) + \underbrace{\alpha}_{\text{step size}} \cdot \underbrace{\left[ \underbrace{G_t}_{\text{objetivo}} - \underbrace{V(S_t)}_{\text{estimación actual}} \right]}_{\text{error de estimación}}$$



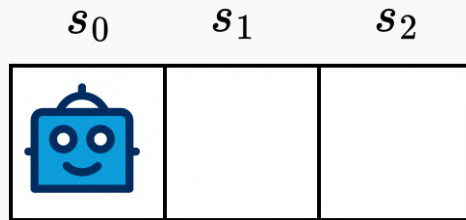
El **objetivo**, en este caso, es  $G_t$ .

# Actualización incremental con Monte Carlo

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

- $G_t$  es el **retorno** obtenido a partir del *time step*  $t$ .
- $\alpha \in (0, 1]$  es un parámetro denominado **step size**, que determina el “peso” de la actualización.
- $G_t - V(S_t)$  es la **diferencia** entre el nuevo valor obtenido y el valor estimado actual. Representa la dirección y magnitud de la actualización de  $V(S_t)$ .

# Ejemplo

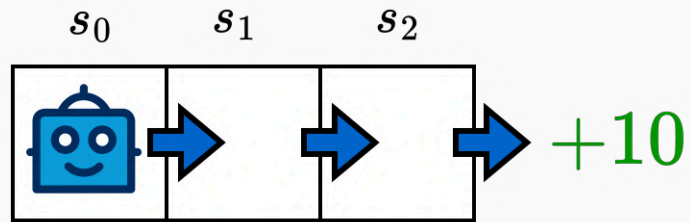


$$\alpha = 0.9$$

$$v(s_0) = \mathbf{0}$$

# Ejemplo

*Episodio 1*



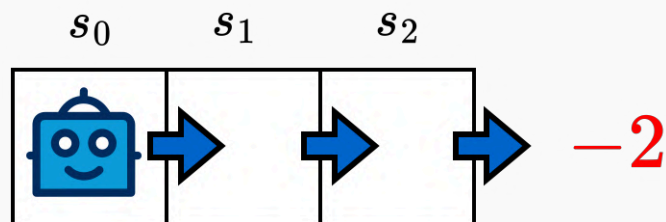
$$G_t = +10$$

$$v(s_0) \leftarrow v(s_0) + \alpha \cdot [G_t - v(s_0)] = 0 + 0.9 \cdot [10 - 0] = \mathbf{9}$$



# Ejemplo

*Episodio 2*

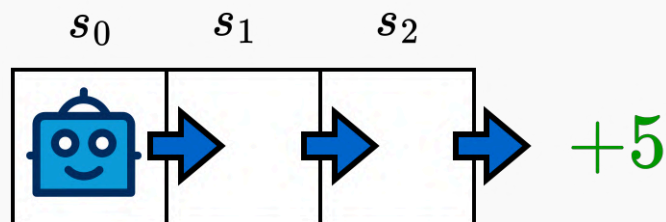


$$G_t = -2$$

$$v(s_0) \leftarrow v(s_0) + \alpha \cdot [G_t - v(s_0)] = 9 + 0.9 \cdot [-2 - 9] = -\mathbf{0.9}$$

# Ejemplo

*Episodio 3*

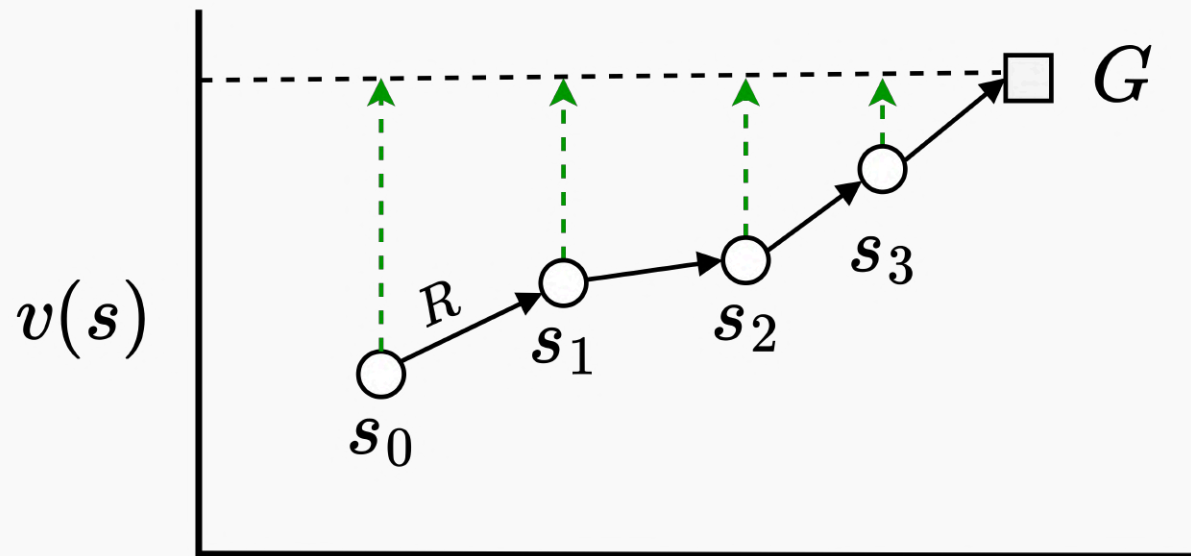


$$G_t = +5$$

$$v(s_0) \leftarrow v(s_0) + \alpha \cdot [G_t - v(s_0)] = -0.9 + 0.9 \cdot [5 - (-0.9)] = \mathbf{4.41}$$

# Limitaciones de Monte Carlo

Como los métodos **Monte Carlo** emplean el valor de retorno  $G$  para aproximar  $v_\pi, q_\pi$ , **deben esperar al final del episodio** para actualizar los valores estimados.



# Limitaciones de Monte Carlo

## LIMITACIONES

- En problemas con **episodios muy largos**, el aprendizaje basado en MC puede ser demasiado lento.
- Además, las actualizaciones suelen ser muy extremas (**alta variabilidad**).
  - *Se acumulan múltiples eventos aleatorios a lo largo de una misma trayectoria.*
- ¿Y qué ocurre si abordamos **problemas continuados**, donde no hay un estado final?

## LIMITACIONES

- En problemas con **episodios muy largos**, el aprendizaje basado en MC puede ser demasiado lento.
- Además, las actualizaciones suelen ser muy extremas (**alta variabilidad**).
  - *Se acumulan múltiples eventos aleatorios a lo largo de una misma trayectoria.*
- ¿Y qué ocurre si abordamos **problemas continuados**, donde no hay un estado final?

**¿Alguna alternativa?**

*TD-learning*

---

## TD-learning

El **aprendizaje por diferencia temporal** (*Temporal Difference Learning* o, abreviado, **TD-learning**) es un conjunto de métodos *model-free* utilizados para estimar funciones de valor mediante actualizaciones iterativas basadas en la diferencia temporal entre predicciones sucesivas.

Veamos en detalle en qué se diferencia de MC...

Tanto **TD** como **MC** se basan en la **experiencia** para resolver el problema de **predicción** de valores.

- Dado un conjunto de experiencias bajo la política  $\pi$ , actualizan su estimación  $V$  de  $v_\pi$  para todo estado no terminal  $S_t$  que tiene lugar durante esa experiencia.

Como hemos visto, **MC** requiere esperar al **final de episodio** para obtener  $G_t$ , que es empleado como **objetivo** en la actualización de  $V(S_t)$ :

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

Decimos que es una **actualización *no sesgada*** del valor estimado.



Las trayectorias del agente son **aleatorias**, por lo que se necesitan **muchos datos** (experiencia, trayectorias, **retornos...**) para poder estimar correctamente las funciones de valor.

Las trayectorias del agente son **aleatorias**, por lo que se necesitan **muchos datos** (experiencia, trayectorias, **retornos...**) para poder estimar correctamente las funciones de valor.

? ¿Cómo evitamos esperar al **final** de un episodio para obtener  $G_t$ ?

Las trayectorias del agente son **aleatorias**, por lo que se necesitan **muchos datos** (experiencia, trayectorias, **retornos...**) para poder estimar correctamente las funciones de valor.

? ¿Cómo evitamos esperar al **final** de un episodio para obtener  $G_t$ ?

☒ Utilizando una **estimación** de  $G_t$ .

Las trayectorias del agente son **aleatorias**, por lo que se necesitan **muchos datos** (experiencia, trayectorias, **retornos...**) para poder estimar correctamente las funciones de valor.

? ¿Cómo evitamos esperar al **final** de un episodio para obtener  $G_t$ ?

☑ Utilizando una **estimación** de  $G_t$ .

? ¿Y qué es, por definición, “una estimación de  $G_t$ ”?

Las trayectorias del agente son **aleatorias**, por lo que se necesitan **muchos datos** (experiencia, trayectorias, **retornos...**) para poder estimar correctamente las funciones de valor.

? ¿Cómo evitamos esperar al **final** de un episodio para obtener  $G_t$ ?

☑ Utilizando una **estimación** de  $G_t$ .

? ¿Y qué es, por definición, “una estimación de  $G_t$ ”?

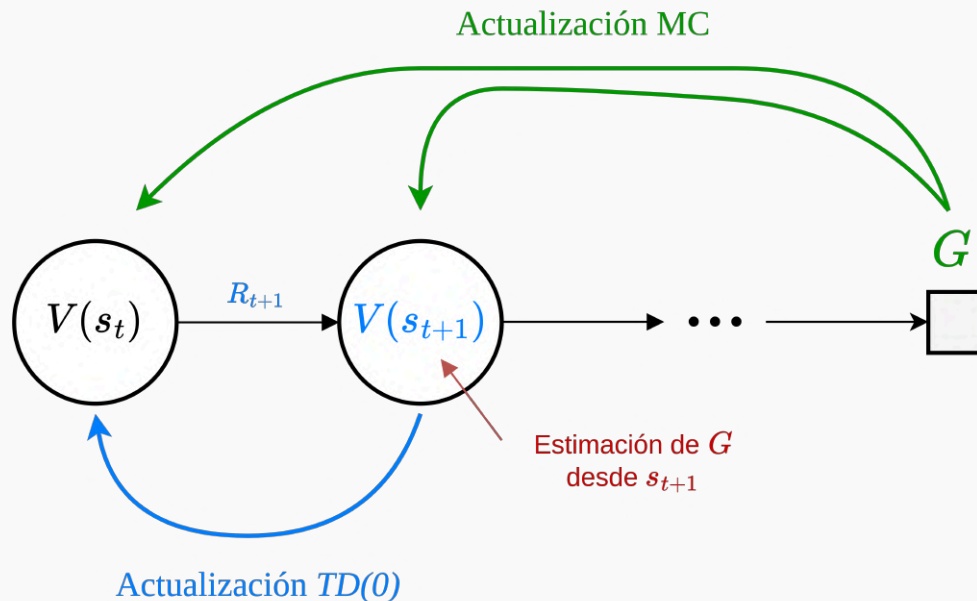
🧠 ¡La **función de valor**  $V \approx v_\pi$  que estamos aproximando!

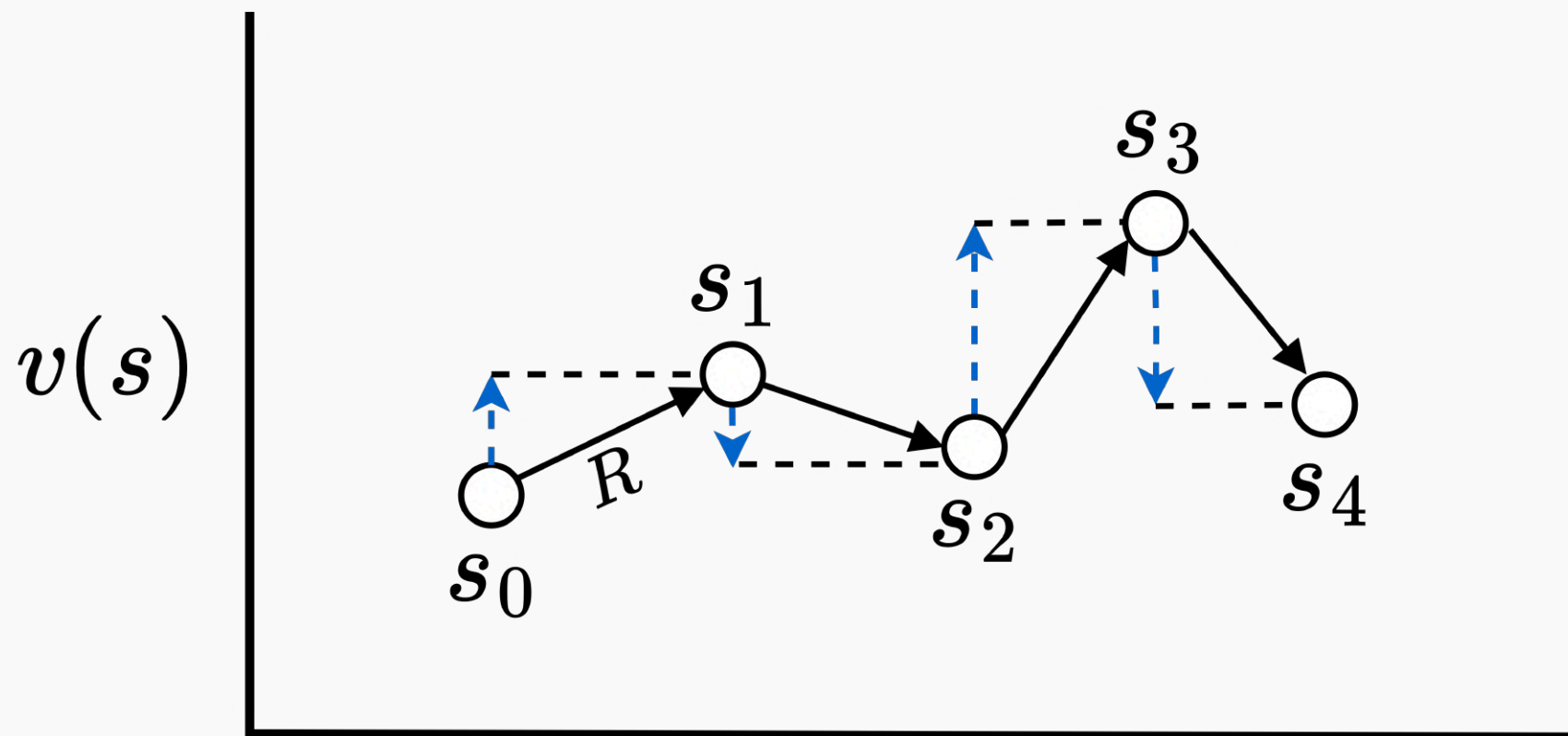
✚ Recordemos que  $v_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$ .

# TD-learning

Es decir, para actualizar el valor de cada estado **no es necesario esperar al final del episodio** y emplear  $G_t$  como objetivo.

Podemos simplemente utilizar **el valor del siguiente estado**, que es por definición, una expectativa del retorno que podemos obtener.





Veámoslo de otra forma...

El **valor**  $v_\pi(s)$  es el retorno  $G_t$  que esperamos obtener siguiendo  $\pi$  desde el estado  $s$ :

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

La definición de **retorno** es:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-1} R_T \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots + \gamma^{T-2} R_T) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Tenemos una **definición recursiva del retorno**.



Vamos a introducir esta **definición recursiva** de retorno en la definición de  $v_\pi$ .

Si las combinamos tenemos:

$$\begin{aligned}v_\pi(s) &= \mathbb{E}[G_t \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]\end{aligned}$$

Es decir, el **retorno esperado**,  $G_t$ , es igual a la suma de la **recompensa inmediata**,  $R_{t+1}$ , y el **retorno descontado futuro**,  $\gamma G_{t+1}$ .

Por definición, este es igual a **la función de valor descontada del siguiente estado**,  $\gamma v_\pi(S_{t+1})$ .

Esto significa que podemos estimar la función de valor  $v_{\pi}(s)$  en cada *time step*.

Dado una experiencia  $S_t, A_t, R_{t+1}, S_{t+1} \sim \pi$ , la estimación  $V(S_t)$  se actualiza de la siguiente forma:

$$V_{t+1}(S_t) = V_t(S_t) + \alpha[R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)]$$

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

Este método se denomina **TD(0)** o *one-step TD*, porque únicamente tiene en cuenta el *time step* inmediatamente posterior.

$$V(S_t) \leftarrow \underbrace{V(S_t)}_{\substack{\text{Valor} \\ \text{estimado} \\ \text{actual}}} + \alpha \left[ \underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD-target}} - \underbrace{V(S_t)}_{\text{TD-error}} \right]$$

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

El *TD-error* mide la diferencia entre dos predicciones: la actual y la recién obtenida.

⇒ Es decir, mide la **diferencia temporal**, o *temporal difference*.

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

El *TD-error* mide la diferencia entre dos predicciones: la actual y la recién obtenida.

➡ Es decir, mide la **diferencia temporal**, o *temporal difference*.

Pregunta...

? ¿Cómo afecta el valor de  $\alpha$  a la convergencia de TD?

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

El *TD-error* mide la diferencia entre dos predicciones: la actual y la recién obtenida.

➡ Es decir, mide la **diferencia temporal**, o *temporal difference*.

## Pregunta...

? ¿Cómo afecta el valor de  $\alpha$  a la convergencia de TD?

- Si  $\alpha$  es más pequeño, TD converge más despacio, pero disminuye el error.

Decimos que TD está basado en **bootstrapping** porque las actualizaciones de la función de valor estimada  $V(S_t)$  se basan en otro valor estimado  $V(S_{t+1})$ .

... y no valores reales como  $G_t$ .

- Se dice que el *TD-target* es, por tanto, **una estimación sesgada** de la función de valor real  $v_\pi$ .

Una ventaja frente a MC es que **la varianza es mucho menor**, ya que el *TD-target* solamente depende de una acción/transición/recompensa, por lo que no se acumula tanta aleatoriedad.

- En la práctica, esto se traduce en un **aprendizaje más rápido**. ⚡ ⚡ ⚡

*Scalable model-free learning*

*Prediction learning*

*It is the unsupervised supervised learning*

*Temporal difference is a method for learning to predict*

*Learning a guess from a guess*

*TD-error emulates dopamine in mammals*

*Richard Sutton*



 Morales, M. (2020). *Grokking deep reinforcement learning*. Manning Publications.


TD methods estimate  $v_{\pi}(s)$  using an estimate of  $v_{\pi}(s)$ . It **bootstraps** and makes a guess from a guess; it uses an **estimated return** instead of the actual return. More concretely, it uses  $R_{t+1} + \gamma V_t(S_{t+1})$  to calculate and estimate  $V_{t+1}(S_t)$ .


Because it also uses **one step of the actual return**  $R_{t+1}$ , things work out fine. That reward signal  $R_{t+1}$  progressively **“injects reality”** into the estimates.

 TD,  MC y  DP

# TD-learning vs. MC

Una posible analogía...

 **TD** es similar a corregir tu comportamiento tan pronto como puedas. No esperas a observar las repercusiones finales de tus acciones, sino que te basas en el *feedback* inmediato + tus expectativas futuras.

 **MC** equivale a realizar una acción y esperar a observar sus resultados a largo plazo para ver si fue buena o mala. En función de esto, modificas tu comportamiento.

 Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

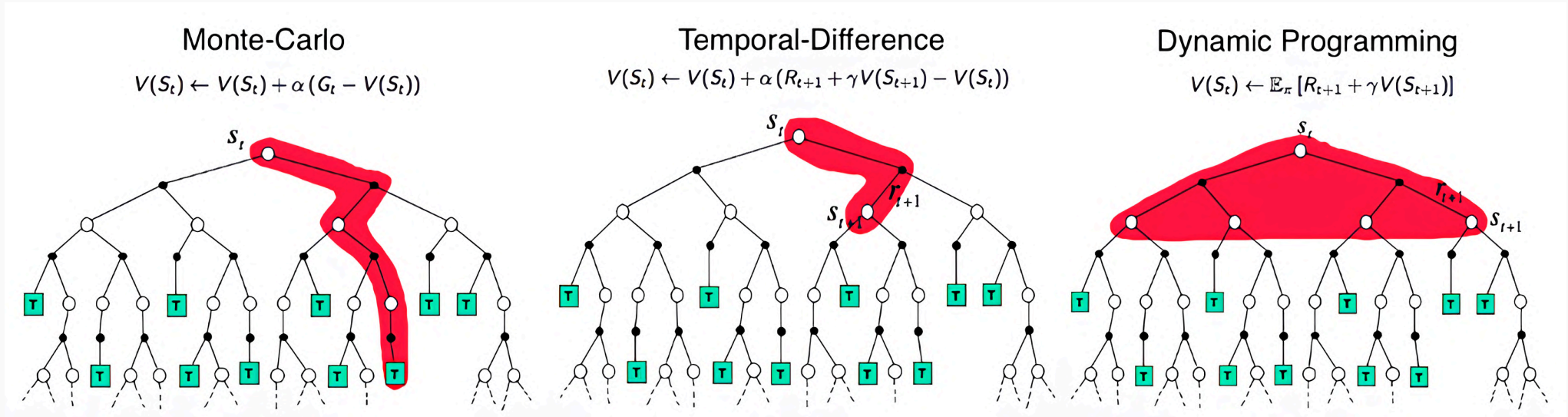
*If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be **temporal-difference (TD) learning**.*

*TD learning is a **combination of Monte Carlo ideas and dynamic programming (DP) ideas**. Like Monte Carlo methods, TD methods can learn directly from raw **experience without a model** of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned **estimates**, without waiting for a final outcome (they **bootstrap**).*

# TD-learning vs. MC y DP

🕒 TD combina:

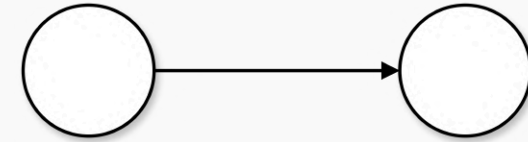
- El **muestreo** de 🎲 MC.
- El **bootstrapping** de 🔄 DP.



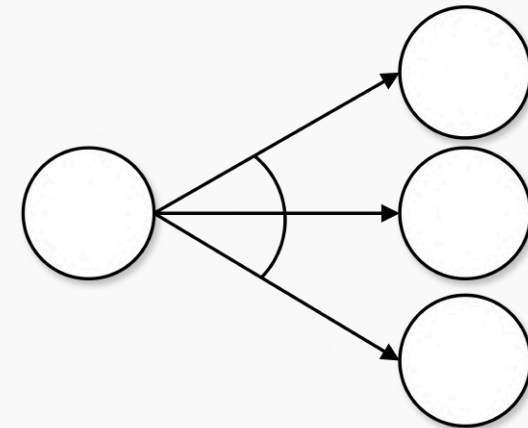
# Sample updates vs. Expected updates

## Sample updates vs. Expected updates



**Sample updates** (📅 MC, 🕒 TD): se basan en un único estado sucesor aleatorio (o par acción-estado).




**Expected updates** (🔄 DP): se basan en una distribución completa de todos los posibles estados sucesores (o pares acción-estado).



# Sample updates

 MC y  TD actualizan valores a partir de muestras aleatorias (actualizaciones muestreadas o *sample updates*).

Cada actualización supone:

1. *Mirar hacia delante*  hacia un posible estado sucesor aleatorio (o par acción-estado).
2. Usar el *valor del sucesor y la recompensa obtenida* para calcular un valor estimado.
3. *Actualizar* el valor estimado original.

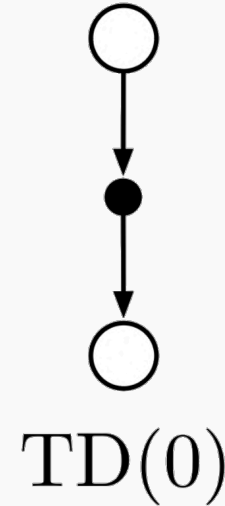


Diagrama *backup* de TD(0)

# TD error vs. MC error

El **TD error** mide la diferencia entre el valor estimado de  $S_t$  y la estimación actualizada  $R_{t+1} + \gamma V(S_{t+1})$ .

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

El **MC error** equivale a la suma de *TD-errors* consecutivos (**MC error =  $\sum$  TD errors**).  
Esto es:

$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k$$



# TD error vs. MC error

El **TD error** ( $\delta$ ) mide la diferencia entre el valor estimado de  $S_t$  y la estimación actualizada  $R_{t+1} + \gamma V(S_{t+1})$ .

$$\delta_t = \underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{Nuevo valor estimado}} - \underbrace{V(S_t)}_{\text{Estimación actual}}$$

- El *TD-error* es proporcional a los cambios a lo largo del tiempo del valor estimado.

El **MC error** es equivalente a la suma de *TD-errors* consecutivos:

$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k$$

## Ventajas de 🕒 TD vs. 🧩 DP

- Al basarse en la **experiencia**, no necesitamos un modelo del entorno.
  - Recompensas para cada par acción-estado
  - Probabilidades de transición
  - Etc.
- Permite abordar **problemas con grandes espacios de estados/acciones** de forma más eficiente.

## Ventajas de 🕒 TD vs. 📅 MC

- Permite abordar problemas con **episodios largos** de forma más eficiente.
- Alternativa a MC en **problemas continuados**.
- TD aprende transición-a-transición, por lo que no depende de acciones tomadas en el futuro.
  - MC requiere esperar al final del episodio.
  - TD **sólo necesita el siguiente timestep**.

*n*-step TD

Hemos visto que TD(0) solamente considera el estado inmediatamente posterior (*1-step TD*).

$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$$

Pero TD permite tantas estimaciones hacia delante como queramos ➡ *TD(n)*.

Por ejemplo, el *target* de *TD(1)* sería:

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

En general:

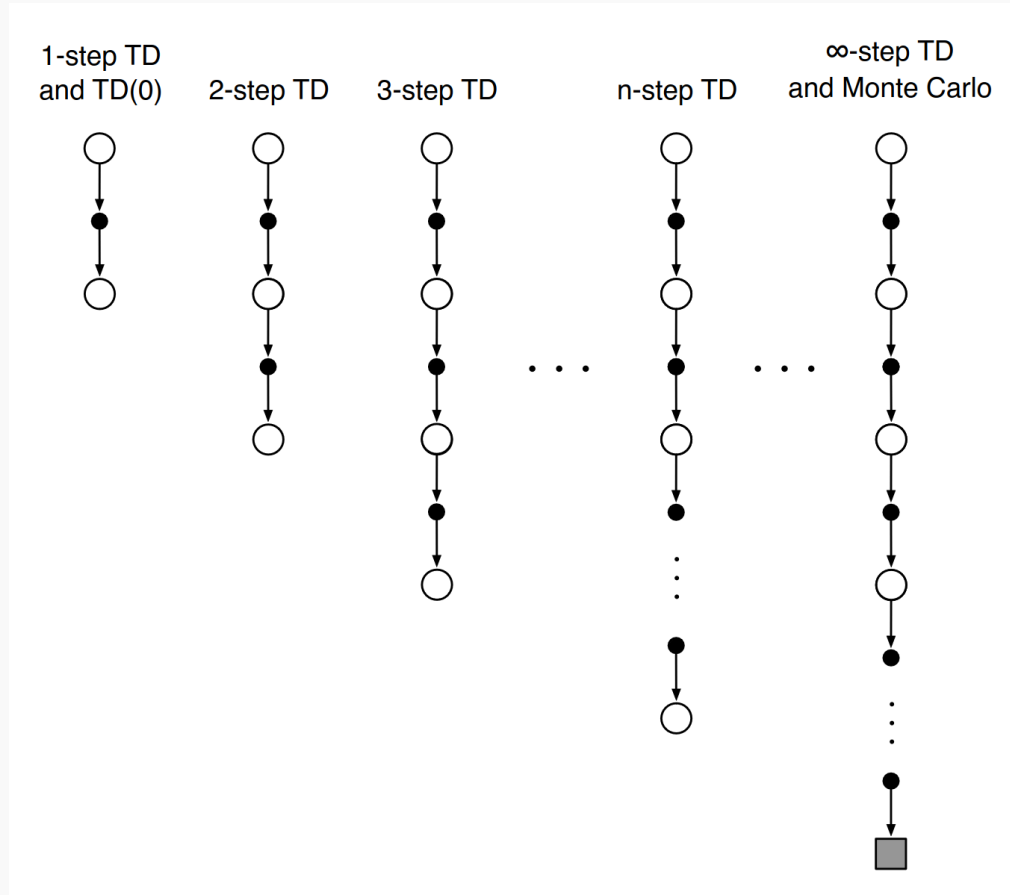
$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

Lo que resulta en la siguiente actualización de los *state-values*:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha [G_{t:t+n} - V_{t+n-1}(S_t)], \quad 0 \leq t < T$$

Es lo que denominamos *n-step TD*.

El caso extremo es **Monte Carlo**:



## *n*-step TD for estimating $V \approx v_\pi$

Input: a policy  $\pi$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , a positive integer  $n$

Initialize  $V(s)$  arbitrarily, for all  $s \in \mathcal{S}$

All store and access operations (for  $S_t$  and  $R_t$ ) can take their index mod  $n + 1$

Loop for each episode:

    Initialize and store  $S_0 \neq$  terminal

$T \leftarrow \infty$

    Loop for  $t = 0, 1, 2, \dots$  :

        If  $t < T$ , then:

            Take an action according to  $\pi(\cdot | S_t)$

            Observe and store the next reward as  $R_{t+1}$  and the next state as  $S_{t+1}$

            If  $S_{t+1}$  is terminal, then  $T \leftarrow t + 1$

$\tau \leftarrow t - n + 1$  ( $\tau$  is the time whose state's estimate is being updated)

        If  $\tau \geq 0$ :

$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$

            If  $\tau + n < T$ , then:  $G \leftarrow G + \gamma^n V(S_{\tau+n})$  ( $G_{\tau:\tau+n}$ )

$V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$

    Until  $\tau = T - 1$

# Garantías de TD



*¿Pero TD es un método sólido/robusto?*  
*¿Podemos garantizar la **convergencia** en  $v_\pi$ ,  $q_\pi$ ?*

*¿Pero TD es un método sólido/robusto?  
¿Podemos garantizar la **convergencia** en  $v_\pi, q_\pi$ ?*

☑ **¡SÍ!** Para cualquier política  $\pi$ , está demostrado que TD converge en la función de valor real  $v_\pi, q_\pi$ .

*¿Pero TD es un método sólido/robusto?  
¿Podemos garantizar la **convergencia** en  $v_\pi$ ,  $q_\pi$ ?*

☑ **¡SÍ!** Para cualquier política  $\pi$ , está demostrado que TD converge en la función de valor real  $v_\pi$ ,  $q_\pi$ .

*Si MC y TD convergen asintóticamente en los valores correctos...  
¿Qué método lo hace más **rápido**? ¿Cuál hace un uso más **eficiente** de los datos?*

➡ Si bien no existe una demostración formal concreta, empíricamente se observa que, en problemas estocásticos, 🕒 **TD tiende a converger más rápido que un 📅 MC con  $\alpha$  constante.**

# SARSA

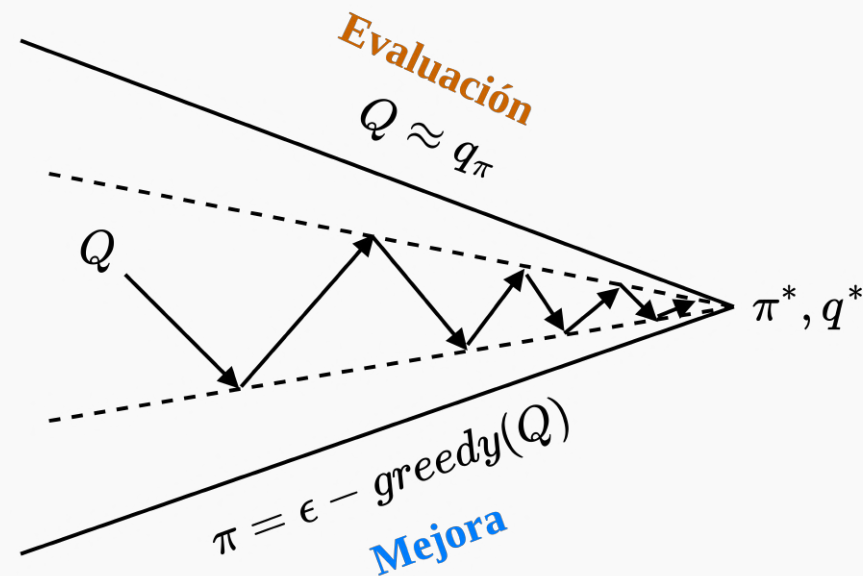


# Control TD

Una vez vista la aplicación de TD en **predicción**, pasamos a abordar el problema de **control**.

De nuevo, seguiremos el patrón de **GPI** (evaluación + mejora), pero empleando TD en la evaluación/predicción.

- También volvemos a encontrarnos con el dilema **exploración-explotación**, que abordaremos desde dos aproximaciones: métodos **on-policy** y **off-policy**, como en MC.

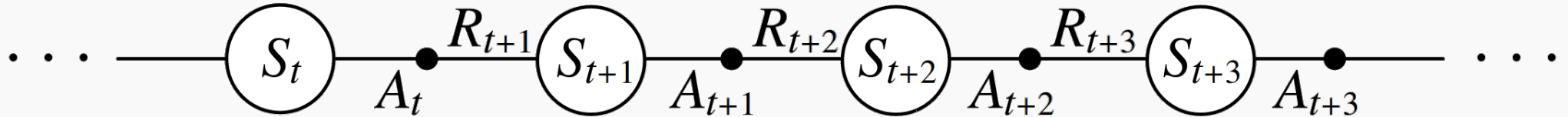


# SARSA: control TD *on-policy*

Buscamos aprender la **función acción-valor**  $q_{\pi}(s, a)$  y obtener así la **política óptima**.

Podemos hacerlo de forma similar al caso de  $v_{\pi}$ ...

- Antes considerábamos transiciones de **estado** a **estado** y aprendíamos el valor de los estados.
- Ahora, consideramos transiciones  $(s, a)$  a  $(s, a)$  y aprendemos los valores de los pares **acción-estado**.



# SARSA: control TD *on-policy*

La **regla de actualización** de los *action values* es la siguiente:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- Esta regla se aplica en cada transición desde estados  $S_t$  no terminales.
- Si  $S_{t+1}$  es terminal,  $Q(S_{t+1}, A_{t+1}) = 0$ , o lo que es lo mismo:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} - Q(S_t, A_t)]$$

- La convergencia de  $Q$  en  $q_\pi$  empleando TD(0) está demostrada.

## SARSA: control TD *on-policy*

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Esta regla de actualización emplea todos los elementos de la quintupla

$$\langle S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} \rangle$$



## SARSA: control TD *on-policy*

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Esta regla de actualización emplea todos los elementos de la quintupla

$$\underbrace{\langle S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1} \rangle}_{\text{SARSA}}$$

# SARSA: control TD *on-policy*

Es relativamente sencillo diseñar un algoritmo de **control *on-policy*** basado en el método de predicción de SARSA:

1. Como en cualquier método *on-policy*, estimamos continuamente  $q_\pi$ , siendo  $\pi$  nuestra política de comportamiento.
2. Al mismo tiempo,  $\pi$  se actualiza de forma ***greedy*** con respecto al  $q_\pi$  estimado.

Se emplea una política  ***$\epsilon$ -greedy*** o, en general  *$\epsilon$ -soft*.

Diagrama *backup* de SARSA



Sarsa

# SARSA: control TD *on-policy*

## Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

    Initialize  $S$

    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

    Loop for each step of episode:

        Take action  $A$ , observe  $R, S'$

        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

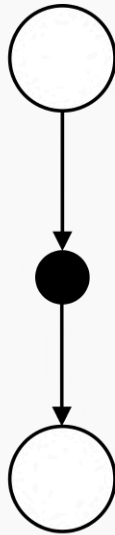
    until  $S$  is terminal

# SARSA: control TD *on-policy*

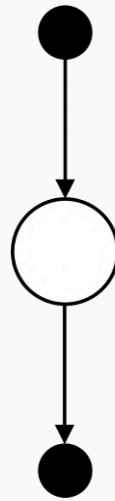
SARSA es un método basado en experiencia/muestreo (*sample-based*) que resuelve la **ecuación de Bellman** para *action-values*  $q(s, a)$ .

- Al ser ***on-policy***, cuenta con una única política (la de comportamiento).
- SARSA **aprende rápido** durante cada episodio. En caso de que la política actual no esté dando buenos resultados, se actualiza rápidamente sin esperar al final del episodio.

# Diagramas *backup*

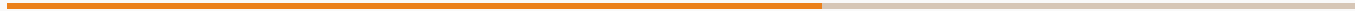


*TD(0)*



*SARSA*

# *Q-learning*



## *Q-learning: control TD off-policy*

Uno de los mayores hitos en el campo del aprendizaje por refuerzo fue el desarrollo de un algoritmo de **control off-policy basado en TD**, conocido como **Q-learning** (Watkins, 1989).

La regla de actualización en la que se basa es la siguiente:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

En este caso, la función acción-valor  $Q$  se aproxima directamente a  $q_*$  (la función acción-valor óptima).

# Q-learning vs. SARSA

*Q-learning* no itera entre *policy evaluation* y *policy improvement*.

- Aprende los valores óptimos directamente.

Tiene un *TD-target* más estable que SARSA, porque sólo cambia si cambia el valor máximo de  $Q(S_{t+1}, A_{t+1})$ .

- SARSA:  $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$
- Q-learning:  $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$



# Q-learning vs. SARSA

**SARSA**  $\Rightarrow$   $a$  se obtiene de la política de comportamiento.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma Q\left(S_{t+1}, \underbrace{A_{t+1}}_{a \sim b}\right) - Q(S_t, A_t) \right]$$

**Q-learning**  $\Rightarrow$   $a$  se obtiene de la política objetivo.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q\left(S_{t+1}, \underbrace{a}_{a \sim \pi}\right) - Q(S_t, A_t) \right]$$

# Q-learning: control TD off-policy

## Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

    Initialize  $S$

    Loop for each step of episode:

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

        Take action  $A$ , observe  $R, S'$

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

    until  $S$  is terminal

# Q-learning: control TD off-policy

## Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$

Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

Initialize  $S$

Loop for each step of episode:

Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

Take action  $A$ , observe  $R$ ,  $S'$

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

until  $S$  is terminal

**Política de  
comportamiento**  
(generar experiencia)

**Política objetivo**  
(comportamiento óptimo)

## Q-learning: control TD off-policy

La **política de comportamiento** determina qué pares acción-estado se visitan y actualizan.

- El único requisito para que el algoritmo converja es que se visiten y actualicen periódicamente todos los pares acción-estado.

Es común representar  $Q(s, a)$  de forma **tabular**.

- El algoritmo de *Q-learning* actualiza esta tabla (*Q-table*) de forma iterativa en base a la experiencia generada.

La **política objetivo** es aquella que actúa de forma **greedy** con respecto a los *Q-values* aproximados.

# Q-learning: control TD off-policy

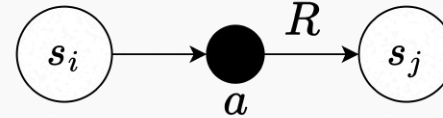
	←	↑	↓	→
$s_0$	0	0	0	0
$s_1$	0	0	0	0
$s_2$	0	0	0	0
$s_3$	0	0	0	0
$s_4$	0	0	0	0

Tabla  $Q$

...	$a$	...
$s_i$	$Q(s_i, a)$	
...		

$s_i$

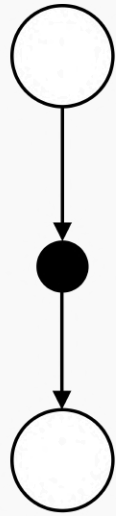
Elegir acción siguiendo la **política de comportamiento** (ej.  $\epsilon$ -greedy)



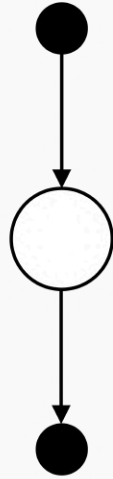
Actualizar  $Q(s_i, a)$  utilizando la recompensa  $R$  obtenida y el máximo  $Q$ -value obtenible según la **política objetivo / tabla  $Q$** .

$$Q(s_i, a) \leftarrow Q(s_i, a) + \alpha[R + \gamma \max Q(s_j, a) - Q(s_i, a)]$$

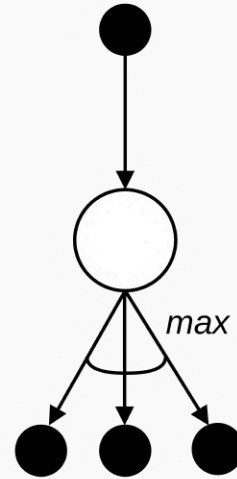
# Diagramas *backup*



*TD(0)*

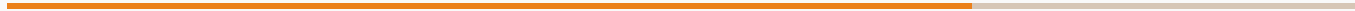


*SARSA*



*Q-learning*

*Expected SARSA*



# Expected SARSA

Recapitulemos...

- Regla de actualización de **SARSA** (control TD *on-policy*):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- Regla de actualización de **Q-learning** (control TD *off-policy*):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$



# Expected SARSA

**Expected SARSA** es una alternativa que utiliza la siguiente **regla de actualización**:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \mathbb{E}_{\pi} [Q(S_{t+1}, A_{t+1} | S_{t+1})] - Q(S_t, A_t)] \\ &\leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \sum_a \pi(a | S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right] \end{aligned}$$

Sigue un esquema similar al de *Q-learning* pero, en vez utilizar el valor máximo para  $Q(S_{t+1}, A_{t+1})$ , utiliza el **valor esperado**.

*Expected SARSA* emplea la suma de todos los posibles valores  $Q(S_{t+1}, A_{t+1})$  ponderados por la probabilidad de elección de  $A_{t+1}$ .

*TD targets* de los diferentes algoritmos vistos hasta el momento

- **SARSA:**  $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$
- ***Q-learning*:**  $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$
- ***Expected SARSA*:**  $R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a)$

**SARSA:**  $R_{t+1} + \gamma Q(s_{t+1}, A_{t+1})$

- Emplea como estimación el valor de un par  $(s_{t+1}, A_{t+1})$  elegido aleatoriamente por la política de comportamiento.

**Q-learning:**  $R_{t+1} + \gamma \max_a Q(s_{t+1}, a)$

- Emplea como estimación el valor del par  $(s_{t+1}, A_{t+1})$  con valor máximo, que sería el elegido por la política objetivo.

**Expected SARSA:**  $R_{t+1} + \gamma \sum_a \pi(a|s_{t+1}) Q(s_{t+1}, a)$

- Emplea como estimación la suma de los valores de todos los pares  $(s_{t+1}, A_{t+1})$  alcanzables, ponderados por sus probabilidades.

# Expected SARSA

*Expected SARSA* es computacionalmente más complejo que SARSA, pero elimina la **alta varianza** asociada a la selección aleatoria de  $A_{t+1}$ .

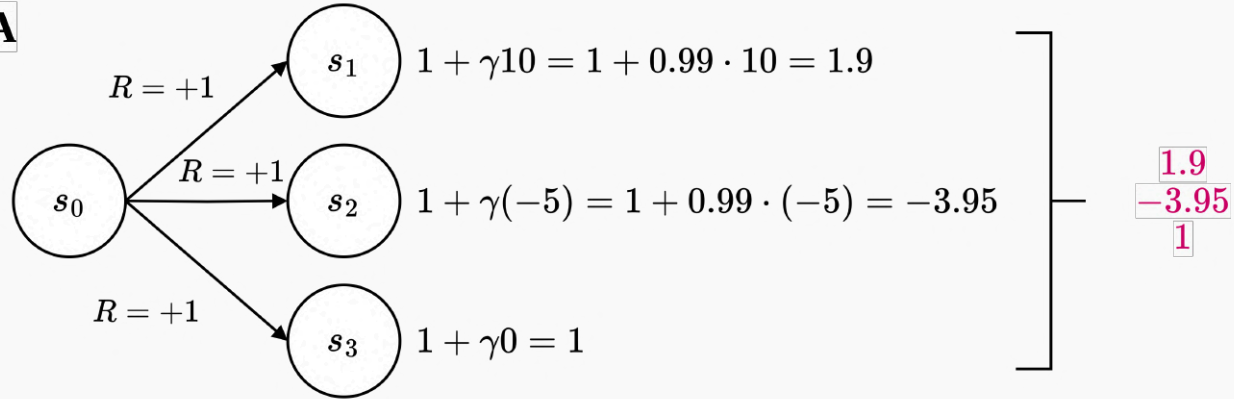
- De esta forma, mejora el rendimiento vs. SARSA (en la mayoría de los casos).

Puede utilizarse de forma **on-policy** u **off-policy**, empleando una política de comportamiento diferente a  $\pi$ .

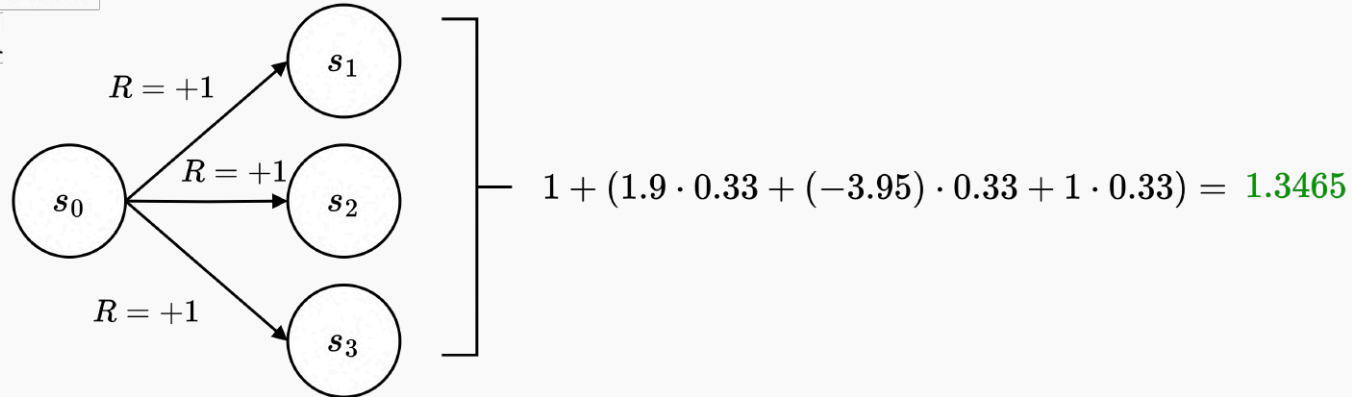
- *Expected SARSA* engloba a *Q-learning* y mejora a SARSA, a cambio de cierto aumento del coste computacional (a medida que el número de acciones posibles crece).
- Por lo general, *Expected SARSA* **domina** al resto de algoritmos basados en TD.

# Expected SARSA vs. SARSA

## SARSA



## EXPECTED SARSA



La actualización del target en Expected SARSA presenta una menor varianza.

## Expected SARSA vs. Q-learning

La principal característica de *Expected SARSA* es su forma de estimar *action-values* en base a los valores esperados.

Con respecto a su implementación, este puede ser tanto *on-policy* como *off-policy*.

En el caso *off-policy*...

¿Qué ocurre si la **política objetivo** ( $\pi$ ) de *Expected SARSA* es *greedy* con respecto a los *action-values* estimados?

$Q(s_{t+1}, a')$	-1.7	0	4.6	2.4
$\pi(a'   s_{t+1})$	0	0	1	0

$$\sum_a \pi(a | s_{t+1}) Q(s_{t+1}, a) = \max_a Q(s_{t+1}, a)$$

## Expected SARSA vs. Q-learning

La principal característica de *Expected SARSA* es su forma de estimar *action-values* en base a los valores esperados.

Con respecto a su implementación, este puede ser tanto *on-policy* como *off-policy*.

En el caso *off-policy*...

¿Qué ocurre si la **política objetivo** ( $\pi$ ) de *Expected SARSA* es *greedy* con respecto a los *action-values* estimados?

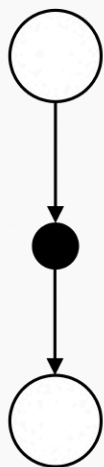
$Q(S_{t+1}, a')$	-1.7	0	4.6	2.4
$\pi(a'   S_{t+1})$	0	0	1	0

$$\sum_a \pi(a | S_{t+1}) Q(S_{t+1}, a) = \underbrace{\max_a Q(S_{t+1}, a)}_{\text{Q-learning}}$$

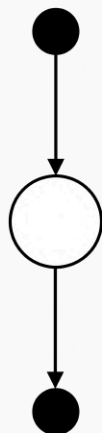


Vemos que *Q-learning* es un caso especial de *Expected SARSA*.

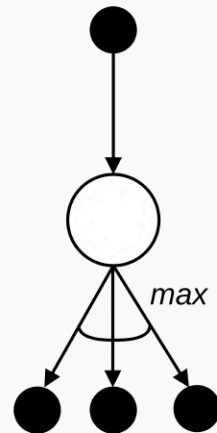
# Comparativa de diagramas *backup*



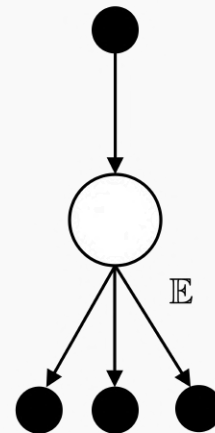
*TD(0)*



*SARSA*



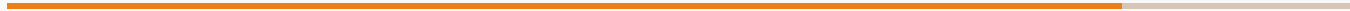
*Q-learning*



*Expected  
SARSA*



Control  $n$ -step



# *n*-step SARSA

Como vimos en el caso de TD, es posible extender los métodos de control vistos a escenarios *n*-step.

En el caso de *n*-step SARSA, el retorno *n*-step sería:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

Lo que da lugar a la siguiente **actualización** de los *action-values*:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

# *n*-step SARSA

1-step Sarsa  
aka Sarsa(0)



2-step Sarsa



3-step Sarsa



...

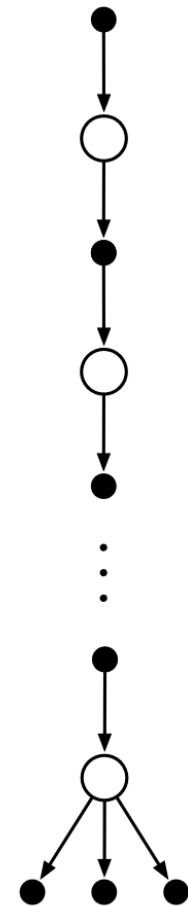
*n*-step Sarsa



$\infty$ -step Sarsa  
aka Monte Carlo



*n*-step  
Expected Sarsa



# *n*-step SARSA

## *n*-step Sarsa for estimating $Q \approx q_*$ or $q_\pi$

Initialize  $Q(s, a)$  arbitrarily, for all  $s \in \mathcal{S}, a \in \mathcal{A}$

Initialize  $\pi$  to be  $\varepsilon$ -greedy with respect to  $Q$ , or to a fixed given policy

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$ , a positive integer  $n$

All store and access operations (for  $S_t$ ,  $A_t$ , and  $R_t$ ) can take their index mod  $n + 1$

Loop for each episode:

    Initialize and store  $S_0 \neq \text{terminal}$

    Select and store an action  $A_0 \sim \pi(\cdot | S_0)$

$T \leftarrow \infty$

    Loop for  $t = 0, 1, 2, \dots$ :

        If  $t < T$ , then:

            Take action  $A_t$

            Observe and store the next reward as  $R_{t+1}$  and the next state as  $S_{t+1}$

            If  $S_{t+1}$  is terminal, then:

$T \leftarrow t + 1$

            else:

                Select and store an action  $A_{t+1} \sim \pi(\cdot | S_{t+1})$

$\tau \leftarrow t - n + 1$  ( $\tau$  is the time whose estimate is being updated)

        If  $\tau \geq 0$ :

$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$

            If  $\tau + n < T$ , then  $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$  ( $G_{\tau:\tau+n}$ )

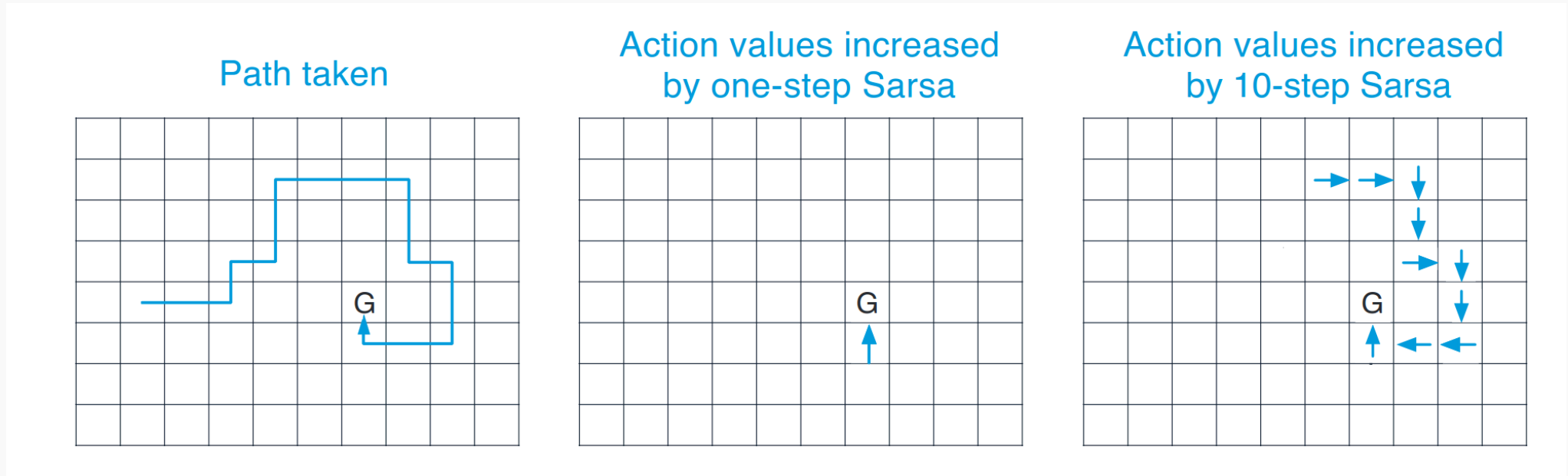
$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha [G - Q(S_\tau, A_\tau)]$

            If  $\pi$  is being learned, then ensure that  $\pi(\cdot | S_\tau)$  is  $\varepsilon$ -greedy wrt  $Q$

    Until  $\tau = T - 1$

# Ejemplo

En este ejemplo comparamos el número de *action-values* que se habrán reforzado al final de un episodio empleando **1-step SARSA** y **10-step SARSA**.



El método *1-step* refuerza sólo la última acción de la secuencia de acciones que condujeron a *G*, mientras que el método *n-step* ( $n = 10$ ) refuerza las últimas  $n$  acciones de la secuencia, por lo que se aprende mucho más en un solo episodio.

# Métodos *n*-step off-policy

En el caso de los métodos *off-policy*, tenemos:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \rho_{t+1:t+n} [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

Donde el *importance sampling ratio* es:

$$\rho_{t:h} \prod_{k=t}^{\min(h, T-1)} \frac{\pi(A_k | S_k)}{b(A_k | S_k)}$$

# Métdos $n$ -step off-policy

Off-policy  $n$ -step Sarsa for estimating  $Q \approx q_*$  or  $q_\pi$

Input: an arbitrary behavior policy  $b$  such that  $b(a|s) > 0$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}$

Initialize  $Q(s, a)$  arbitrarily, for all  $s \in \mathcal{S}, a \in \mathcal{A}$

Initialize  $\pi$  to be greedy with respect to  $Q$ , or as a fixed given policy

Algorithm parameters: step size  $\alpha \in (0, 1]$ , a positive integer  $n$

All store and access operations (for  $S_t$ ,  $A_t$ , and  $R_t$ ) can take their index mod  $n + 1$

Loop for each episode:

    Initialize and store  $S_0 \neq$  terminal

    Select and store an action  $A_0 \sim b(\cdot|S_0)$

$T \leftarrow \infty$

    Loop for  $t = 0, 1, 2, \dots$ :

        If  $t < T$ , then:

            Take action  $A_t$

            Observe and store the next reward as  $R_{t+1}$  and the next state as  $S_{t+1}$

            If  $S_{t+1}$  is terminal, then:

$T \leftarrow t + 1$

            else:

                Select and store an action  $A_{t+1} \sim b(\cdot|S_{t+1})$

$\tau \leftarrow t - n + 1$  ( $\tau$  is the time whose estimate is being updated)

        If  $\tau \geq 0$ :

$\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$  ( $\rho_{\tau+1:t+n-1}$ )

$G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$

            If  $\tau + n < T$ , then:  $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$  ( $G_{\tau:\tau+n}$ )

$Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha \rho [G - Q(S_\tau, A_\tau)]$

            If  $\pi$  is being learned, then ensure that  $\pi(\cdot|S_\tau)$  is greedy wrt  $Q$

    Until  $\tau = T - 1$

Show me the code!



# SARSA

```
class SARSA:

    def __init__(self, env, alpha=.1, gamma=1, epsilon=.1):

        self.env = env

        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon

        self.actions = ['up', 'down', 'left', 'right']

        self.q_table = {
            (x, y): {action: 0 for action in self.actions}
            for x in range(self.env.height) for y in range(self.env.width)
        }

        self.total_rewards = []
```

# SARSA

```
def get_action(self, state):
    '''Returns a sampled action according to E-greedy policy'''

    if np.random.uniform(0, 1) < self.epsilon:
        return np.random.choice(self.actions)
    else:
        x, y = state
        state_actions = self.q_table[(x, y)]
        max_value = max(state_actions.values())
        max_actions = [action for action,
                        value in state_actions.items() if value == max_value]
        return np.random.choice(max_actions)

def update_q_value(self, s, a, r, s_next, a_next):
    '''Applies the SARSA update rule for state-action values'''

    td_target = r + self.gamma * self.q_table[s_next][a_next]
    td_error = td_target - self.q_table[s][a]
    self.q_table[s][a] += self.alpha * td_error
```

# SARSA

```
def learn(self, n_episodes):
    '''Learn Q values by interaction'''

    for _ in range(n_episodes):
        total_reward = 0
        s = (0, 0)
        a = self.get_action(s)

        while not self.env.is_terminal(s):
            s_next, reward = self.env.get_transition(s, a)
            a_next = self.get_action(s_next)

            self.update_q_value(s, a, reward, s_next, a_next)

            s = s_next
            a = a_next
            total_reward += reward

        self.total_rewards.append(total_reward)

    print('Best SARSA episode reward = ', max(self.total_rewards))
```

```
def update_q_value(self, s, a, r, s_next):  
    '''  
    Applies the Q-learning update rule for state-action values  
    '''  
    max_q_next = max(self.q_table[s_next].values())  
  
    td_target = r + self.gamma * max_q_next  
    td_error = td_target - self.q_table[s][a]  
    self.q_table[s][a] += self.alpha * td_error
```

# Expected SARSA

```
def update_q_value(self, s, a, r, s_next):  
    '''  
    Applies the Expected SARSA update rule for state-action values  
    '''  
    expected_q_next = np.mean([self.q_table[s_next][action]  
                               for action in self.actions])  
  
    td_target = r + self.gamma * expected_q_next  
    td_error = td_target - self.q_table[s][a]  
    self.q_table[s][a] += self.alpha * td_error
```

# Trabajo propuesto

- Probar los **algoritmos** vistos (*¡trata de hacer tu propia implementación!*):

[https://github.com/manjavacas/rl-temario/tree/main/codigo/cliff\\_walking](https://github.com/manjavacas/rl-temario/tree/main/codigo/cliff_walking)

<https://github.com/manjavacas/rl-temario/tree/main/codigo/montecarlo>

- Comparativa de **Monte Carlo**, **SARSA**, ***Q-learning*** y ***Expected SARSA*** para un mismo problema, variando los diferentes parámetros (ej.  $\alpha$ ,  $\gamma$ ).
  - Pruébalos en un entorno de Gymnasium.

## Bibliografía y webs recomendadas:

- **Capítulos 6 y 7** de Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction.
- **Capítulo 6** de Morales, M. (2020). Grokking deep reinforcement learning. Manning Publications.
- <https://www.youtube.com/watch?v=bpUzPiWM7o>
- <https://www.youtube.com/watch?v=AjIG3ykOxmY>
- [https://www.youtube.com/watch?v=0g4j2k\\_Ggc4](https://www.youtube.com/watch?v=0g4j2k_Ggc4)
- <https://www.youtube.com/watch?v=0iqz4tcKN58>
- <https://www.youtube.com/watch?v=C3p2wl4RAi8>
- <https://www.statlect.com/asymptotic-theory/importance-sampling>

# Aprendizaje por refuerzo

Métodos basados en muestreo (3)

---

Antonio Manjavacas Lucas

manjavacas@ugr.es