Algoritmos de Diferencia Temporal

Felipe Buitrago Carmona - Facultad de Inteligencia Artificial e Ingenierías - Universidad de Caldas

Recordemos

PROGRAMACIÓN DINÁMICA (predicción y control)

basados en modelos

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

MONTE CARLO (predicción y control)

libres de modelos

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

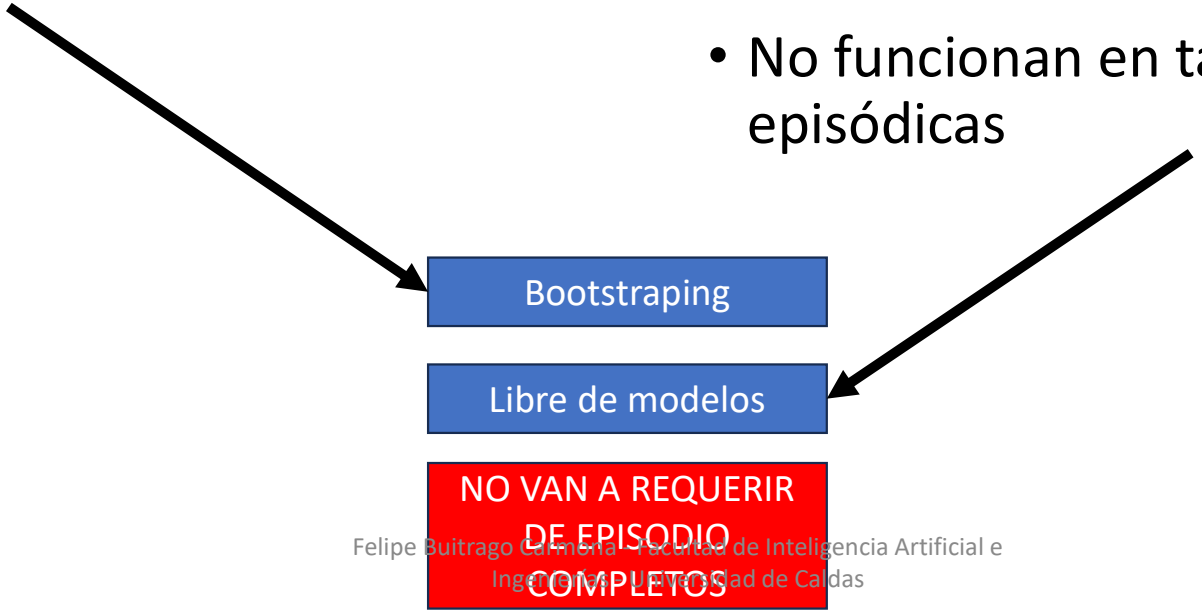
Enfoque algoritmos de diferencia temporal

Programación Dinámica

- Requieren un modelo del entorno

Montecarlo

- Requieren de episodios completos
- No funcionan en tareas no-episódicas



PROGRAMACIÓN DINÁMICA
(predicción y control)

basados en modelos

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

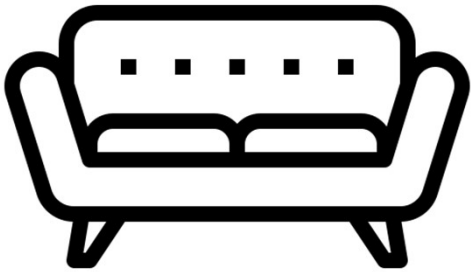
MONTE CARLO
(predicción y control)

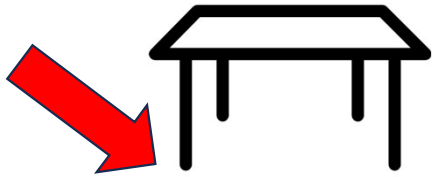
libres de modelos

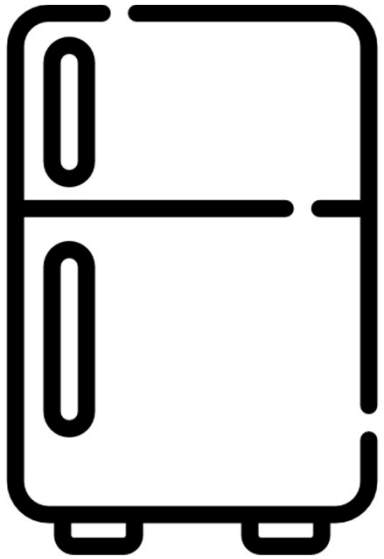
$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \lambda^k R_{t+k+1} | S_t = s \right]$$

Bootstraping: Tomar lo anterior para mejorar lo actual

Ejemplo

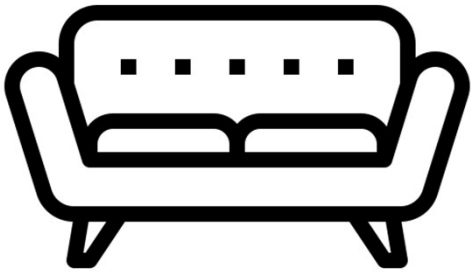


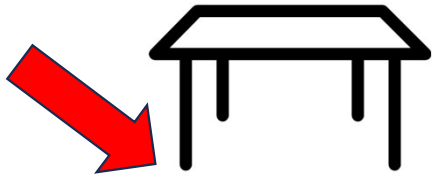


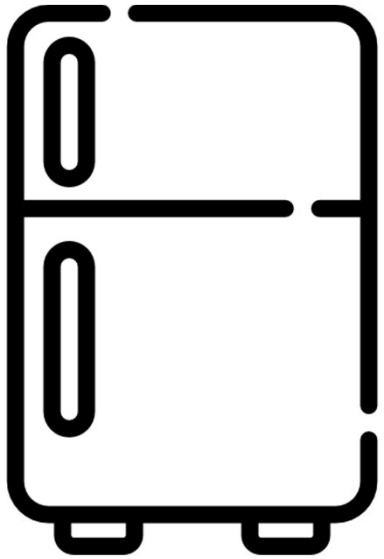


Montecarlo requiere hacer todo el episodio completo para calcular la política

Ejemplo







En diferencia temporal si el agente obtiene una recompensa negativa, será suficiente para determinar que la política no es adecuada

Predicción con diferencia temporal

Felipe Buitrago Carmona - Facultad de Inteligencia Artificial e Ingenierías - Universidad de Caldas

PREDICCIÓN CON MONTE CARLO

PREDICCIÓN CON DIFERENCIA TEMPORAL

estimar la función estado-valor

estimar la función estado-valor

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

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T$$

$$\gamma[R_{t+2} + \gamma R_{t+3} + \cdots + \gamma^{T-2} R_T]$$

$$G_t = R_{t+1} + \gamma G_{t+1}$$

De esta manera no será necesario calcular toda la ecuación para obtener un retorno sino que bastará con uno solo

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \quad \rightarrow \quad \mu_{act.} \leftarrow \mu_{ant.} + \frac{1}{N}(x_N - \mu_{ant.})$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}[R_{t+1} + \gamma G_{t+1} - V(S_t)]$$

- Entrada: $\pi \leftarrow$ la política a evaluar
- Parámetro: α (tasa de aprendizaje), entre 0 y 1
- Inicializar $V(s) \leftarrow$ valores arbitrarios para cada estado "s" (que se actualizará con el algoritmo). Los estados terminales siempre tendrán un $V(s) = 0$
- Repetir K veces y en cada iteración:
  - (a) Generar un episodio usando π Generamos un episodio a partir de la política
  - (b) Ir al primer estado del episodio (S)
  - (c) Por cada instante de tiempo "t" en el episodio:
    - (i) A $\leftarrow$ acción determinada por la política
    - (ii) Interacción: tomando la acción A, observar el retorno (R) y el nuevo estado (S') obtenidos
    - (iii) $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$: actualizar el valor del estado de forma incremental y tras sólo una interacción
    - (iv) $S \leftarrow S'$
    - (v) Detenerse al llegar a un estado terminal

Tome una acción desde e es estado en el que está

Con esa única acción calcular función estado valor

Control con diferencia temporal

Felipe Buitrago Carmona - Facultad de Inteligencia Artificial e Ingenierías - Universidad de Caldas

Control con diferencia temporal

CONTROL CON MONTE CARLO

obtener una **política óptima**

CONTROL CON DIFERENCIA TEMPORAL

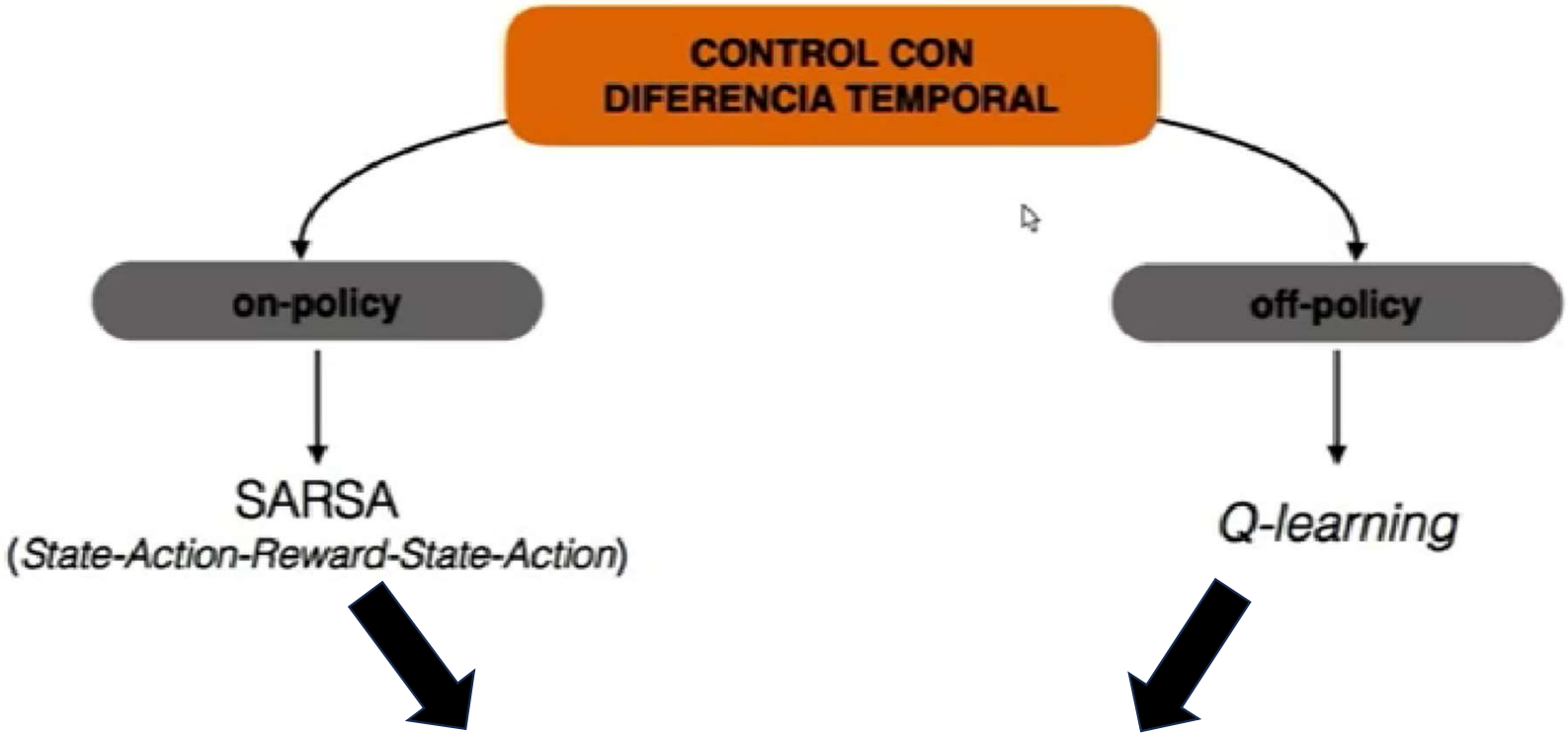
obtener una **política óptima**

Control con diferencia temporal

Utiliza la misma política para que de forma iterativa optimizarla

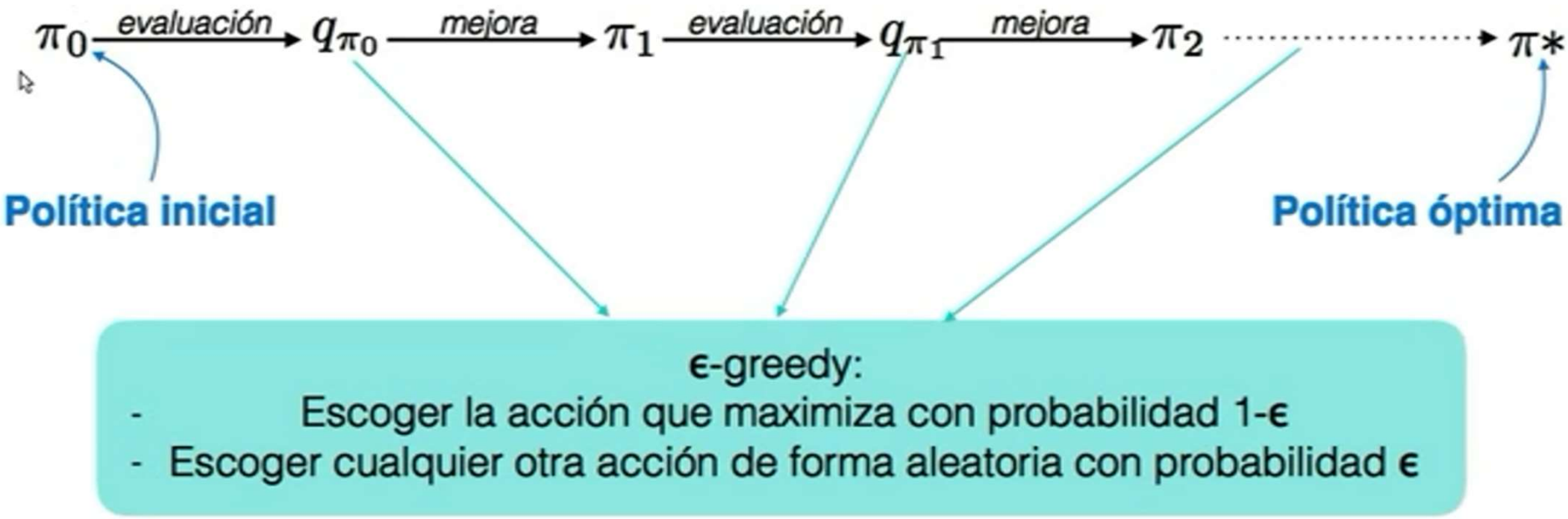
No utilizan la política para encontrar la política óptima sino que utilizan una política "auxiliar"

Control con diferencia temporal



Los 2 parten del principio de **exploración-explotación (ϵ-Greedy)**. Recordemos que este concepto trata sobre que no siempre se tomará la acción óptima sino que se da paso a la aleatoriedad para diversificar las soluciones

Control “on-policy” con diferencia temporal



La diferencia con respecto al anterior es que las función Q, se calculará de manera incremental teniendo en cuenta que solo bastará una sola interacción (Acción) para ir realizando el cálculo

SARSA

Control “Off-Policy” con diferencia temporal

Felipe Buitrago Carmona - Facultad de Inteligencia Artificial e Ingenierías - Universidad de Caldas

SARSA

SARSA (State-Action-Reward-State-Action) es un **algoritmo de aprendizaje por refuerzo basado en diferencias temporales (TD)** que aprende una política de control. A diferencia de métodos como Q-learning, SARSA es un enfoque **"on-policy"**, lo que significa que aprende la política que está siguiendo durante el entrenamiento.

SARSA

SARSA actualiza los valores Q estimados en función de la política actual del agente. Su nombre proviene de la secuencia de pasos clave utilizados para la actualización:

- **S**: Estado actual.
- **A**: Acción tomada en ese estado.
- **R**: Recompensa recibida después de tomar la acción.
- **S'**: Nuevo estado resultante de la acción.
- **A'**: Próxima acción elegida en el nuevo estado, siguiendo la política actual.

SARSA
Control “on-policy” con diferencia temporal

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

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

Actual — Anterior — Tasa Aprendizaje — Target — Anterior

- $Q(S, A)$: Valor Q actual para el par estado-acción.
- α: Tasa de aprendizaje.
- γ: Factor de descuento.
- R: Recompensa inmediata.
- $Q(S', A')$: Valor Q estimado para el próximo par estado-acción.

Algoritmo SARSA

- Entradas: la política a optimizar (¡puede ser totalmente arbitraria!)
- Parámetros: α y ϵ (ambas entre 0 y 1)
- Inicializar Q(S,A) para cada uno de los pares estado-acción. Los estados terminales siempre serán cero
- Repetir K veces y en cada iteración:
    (a) Generar un episodio usando la política
    (b) Ir al primer estado del episodio (S)
    (c) A $\leftarrow$ elegir una acción determinada por la política pero usando un enfoque ϵ-greedy
    (d) Por cada instante de tiempo "t" en el episodio:
        (i) Tomar la acción A y observar la recompensa obtenida (R) y el estado resultante (S')
        (ii)Escoger la nueva acción A' a partir del estado resultante (S') usando un enfoque ϵ-greedy
        (iii)$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$: actualizar la función acción-valor de forma incremental y considerando sólo un instante de tiempo en la interacción
        (iv)$S \leftarrow S'$; $A \leftarrow A'$: actualizar el estado y la acción
        (v)Detenerse al llegar a un estado terminal

Ejemplos – Robot Laberinto

Un robot aprende a navegar por un laberinto para encontrar un objetivo. El espacio de estados son las posiciones posibles del robot en el laberinto, y las acciones son movimientos (arriba, abajo, izquierda, derecha). El agente recibe una recompensa positiva al llegar al objetivo y una recompensa negativa al chocar con paredes.

Con SARSA:

1. El robot comienza en una posición (S) y elige una acción (A) usando la política actual (p. ej., ε-greedy).
2. El robot se mueve al nuevo estado (S'), recibe una recompensa (R), y elige la próxima acción (A').
3. SARSA actualiza $Q(S, A)$ usando la fórmula y repite el proceso.

Ejemplos – Control de un semáforo inteligente

Un semáforo inteligente ajusta el tiempo de luz verde en función del tráfico. El estado (S) puede

CONTROL CON SARSA

obtener una **política óptima**

CONTROL CON Q-LEARNING

obtener una **política óptima**

2. Observa el efecto (S') y recibe una recompensa (R).
3. Evalúa la próxima acción (A') según la política actual y ajusta $Q(S, A)$.

Ventajas

1. **Aprendizaje basado en la política actual ("on-policy"):**
   - SARSA actualiza el valor $Q(S, A)$ siguiendo la política real del agente. Esto hace que las decisiones estén alineadas con la forma en que el agente realmente interactúa con el entorno.
2. **Adecuado para entornos con riesgo o ruido:**
   - En situaciones donde ciertas acciones pueden ser peligrosas (como robots en entornos reales o sistemas financieros), SARSA evita estrategias agresivas, ya que aprende basado en lo que el agente realmente hace, no en lo óptimo teórico.
3. **Exploración controlada:**
   - Si el agente utiliza una política de exploración (como ε-greedy), SARSA incorpora esa exploración en su aprendizaje. Esto significa que evalúa los resultados de la política exploratoria, no de una política óptima hipotética.
4. **Estabilidad:**
   - Tiende a ser más estable que Q-Learning en entornos no deterministas, ya que no depende del valor máximo $\max_{a'} Q(S', a')$, que podría ser una estimación sesgada o imprecisa en entornos ruidosos.
5. **Simplicidad:**
   - La implementación de SARSA es sencilla y directa, lo que lo hace una buena opción para introducirse en el aprendizaje por refuerzo.

Desventajas

1. **Convergencia más lenta:**
   - SARSA puede tardar más en encontrar la política óptima porque aprende según la política actual (que puede ser subóptima), mientras que otros algoritmos, como Q-Learning, apuntan directamente a la política óptima.
2. **Dependencia de la política de exploración:**
   - El rendimiento del aprendizaje puede estar limitado por la calidad de la política exploratoria utilizada. Una política de exploración pobre puede llevar a resultados subóptimos.
3. **Rendimiento subóptimo:**
   - Si se permite suficiente exploración, Q-Learning eventualmente superará a SARSA en términos de encontrar la política óptima, ya que SARSA puede quedarse atrapado en políticas seguras y no explorar suficientemente.
4. **Menor adaptabilidad a cambios en el entorno:**
   - Al estar tan alineado con la política actual, SARSA puede ser más lento para adaptarse a cambios en un entorno dinámico.
5. **Exploración conservadora:**
   - En entornos donde se necesita tomar riesgos para obtener mejores recompensas, SARSA puede no ser ideal debido a su enfoque cauteloso.

Q-Learning

Control “On-Policy” con diferencia temporal

Felipe Buitrago Carmona - Facultad de Inteligencia Artificial e Ingenierías - Universidad de Caldas

Q-Learning

- Q-Learning es un **algoritmo de aprendizaje por refuerzo basado en diferencias temporales (TD)** utilizado para encontrar la **política óptima** en un entorno. Es un método **"off-policy"**, lo que significa que aprende la política óptima independientemente de las acciones que el agente toma durante el entrenamiento.

Q-Learning

En Q-Learning, el agente aprende una función $Q(S, A)$, que estima el valor esperado de tomar una acción A en un estado S y seguir la política óptima a partir de allí. La actualización se realiza mediante la siguiente fórmula:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right]$$

Donde:

- $Q(S, A)$: Valor Q actual para el par estado-acción.
- α: Tasa de aprendizaje (qué tanto se ajustan los valores actuales).
- γ: Factor de descuento (cuánto se valoran las recompensas futuras).
- R: Recompensa inmediata obtenida al tomar la acción A en el estado S.
- $\max_{a'} Q(S', a')$: Máximo valor Q estimado para el siguiente estado S', considerando todas las acciones posibles.

El uso de $\max_{a'} Q(S', a')$ hace que Q-Learning sea "**off-policy**", porque actualiza los valores asumiendo que el agente sigue la política óptima, incluso si en la práctica sigue una política diferente (como ε-greedy durante la exploración).

Q-Learning
Control “off-policy” con diferencia temporal

obtener una **política óptima**

Requiere una política

obtener una **política óptima**

NO Requiere una política

Q-Learning
Control “off-policy” con diferencia temporal

CONTROL CON SARSA

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

CONTROL CON Q-LEARNING

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

Algoritmo Q-Learning

- Entradas: ¡ninguna! (no es necesaria una política)
- Parámetros: α y ϵ (ambas entre 0 y 1)
- Inicializar Q(S,A) para cada uno de los pares estado-acción. Los estados terminales siempre serán cero
- Repetir K veces y en cada iteración:
  - (a) Generar un episodio usando la política derivada de la función Q(S,A) actual
  - (b) Ir al primer estado del episodio (S)
  - (c) A ← elegir una acción a partir de Q(S,A) y usando un enfoque ϵ-greedy
  - (d) Por cada instante de tiempo "t" en el episodio:
    - (i) Tomar la acción A y observar la recompensa obtenida (R) y el estado resultante (S')
      ~~Escoger la nueva acción A' a partir del estado resultante (S') usando un enfoque ϵ-greedy~~
    - (ii) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$: actualizar la función acción-valor de forma incremental, escogiendo la mejor acción y considerando sólo un instante de tiempo en la interacción
    - (iii) S ← S': actualizar sólo el estado (pues la acción se actualizó en el paso (ii))
    - (iv) Detenerse al llegar a un estado terminal

Ejemplos - Robot en un laberinto

Un robot aprende a navegar por un laberinto para alcanzar un objetivo.

- **Estados** (S): Posiciones posibles en el laberinto.
- **Acciones** (A): Moverse (arriba, abajo, izquierda, derecha).
- **Recompensa** (R): +10 por llegar al objetivo, -1 por cada paso tomado, y -10 al chocar con paredes.

Con Q-Learning:

1. El robot comienza en un estado inicial (S) y selecciona una acción (A) basada en una política de exploración (como ε-greedy).
2. Se mueve al nuevo estado (S'), recibe una recompensa (R).
3. Actualiza $Q(S, A)$ usando la recompensa inmediata y el máximo valor Q estimado para el estado siguiente ($\max_{a'} Q(S', a')$).
4. Continúa explorando hasta que converge a la política óptima.

A diferencia de SARSA, Q-Learning no depende de la acción seleccionada en S', sino del mejor valor Q estimado.

Ejemplos –
Agente de juegos: Entrenando a un Pac-Man

Pac-Man aprende a maximizar su puntuación mientras evita fantasmas.

- **Estados** (S): Posición de Pac-Man y de los fantasmas.
- **Acciones** (A): Moverse (arriba, abajo, izquierda, derecha).
- **Recompensa** (R): +10 por comer puntos, +50 por comer frutas, -50 si lo atrapan los fantasmas.

Con Q-Learning:

1. Pac-Man selecciona acciones basadas en una política exploratoria.
2. Tras cada acción, observa el nuevo estado (S') y recibe la recompensa correspondiente.
3. Actualiza $Q(S, A)$ usando el valor máximo para el nuevo estado S', independientemente de la acción realmente tomada.
4. Repite el proceso para aprender a maximizar la recompensa total.

Q-Learning VS SARSA

Felipe Buitrago Carmona - Facultad de Inteligencia Artificial e Ingenierías - Universidad de Caldas

Q-Learning vs SARSA

Aspecto	Q-Learning	SARSA
Naturaleza	Off-policy	On-policy
Política	Aprende la política óptima	Aprende según la política actual
Fórmula de actualización	Usa $\max_{a'} Q(S', a')$	Usa $Q(S', A')$ de la política actual
Uso	Más agresivo, puede explorar estrategias arriesgadas.	Más seguro, adecuado para entornos con ruido.

SARSA

Felipe Buitrago Carmona - Facultad de Inteligencia Artificial e Ingenierías - Universidad de Caldas

Ventajas

1. **Convergencia a la política óptima:**
   - Q-Learning garantiza la convergencia a la política óptima (π^*) si se exploran todas las acciones suficientes veces y los parámetros (como la tasa de aprendizaje) están bien configurados.
2. **Independencia de la política de exploración:**
   - Como es un algoritmo **off-policy**, Q-Learning aprende la política óptima sin importar la estrategia que el agente utilice para explorar el entorno (por ejemplo, ε-greedy, softmax, etc.).
3. **Mayor eficacia en exploración agresiva:**
   - Q-Learning puede aprovechar estrategias exploratorias agresivas para descubrir mejores recompensas a largo plazo, ya que no está restringido a la política actual.
4. **Flexibilidad:**
   - Es aplicable tanto en entornos deterministas como estocásticos, y se adapta bien a problemas discretos con un espacio de estados y acciones manejables.
5. **Generalización más rápida en entornos simples:**
   - En escenarios con menos ruido o donde las recompensas están claramente definidas, Q-Learning puede aprender más rápido que SARSA, ya que apunta directamente al máximo valor $\max_{a'} Q(S', a')$.
6. **Versatilidad:**
   - Se puede integrar con técnicas avanzadas, como redes neuronales (Deep Q-Networks, DQN), para manejar espacios de estados y acciones más grandes y continuos.

Desventajas

1. **Susceptible al ruido:**
   - Al depender del valor máximo $\max_{a'} Q(S', a')$ estimado, puede sobreestimar los valores Q en entornos ruidosos, lo que lleva a decisiones subóptimas.
2. **Riesgos en entornos críticos:**
   - Su enfoque agresivo hacia las recompensas óptimas puede llevar a exploraciones peligrosas o no seguras en sistemas sensibles (como robots físicos o aplicaciones médicas).
3. **Requiere exploración suficiente:**
   - Para aprender la política óptima, el agente necesita explorar exhaustivamente todos los estados y acciones, lo que puede ser ineficiente en entornos grandes o con muchas combinaciones posibles.
4. **Inestabilidad en problemas complejos:**
   - En problemas con espacios de estados grandes o continuos, la estimación precisa de $\max_{a'} Q(S', a')$ puede ser difícil, especialmente cuando se utilizan aproximadores de funciones, como redes neuronales.
5. **Convergencia lenta en casos específicos:**
   - En entornos donde las recompensas óptimas están ocultas detrás de secuencias largas de acciones, puede requerir más tiempo para converger, especialmente sin estrategias exploratorias adecuadas.
6. **Alto costo computacional en grandes espacios de estado:**
   - Q-Learning utiliza una tabla Q, lo que lo hace ineficiente para problemas de gran escala. Para estos casos, se necesita integrar métodos como DQN, pero eso aumenta la complejidad.