

APRENDIZAJE POR REFUERZO

Planificación

Antonio Manjavacas

manjavacas@ugr.es

CONTENIDOS

- 1. Modelos y planificación**
- 2. Dyna**
- 3. Dyna-Q**
- 4. Dyna-Q+**
- 5. Trabajo propuesto**

Hasta el momento hemos visto dos tipos de **algoritmos de RL**...

Model-based

- Requieren un **modelo** del entorno.
- Se basan en la **planificación** a partir de ese modelo.
- Ej. programación dinámica, búsqueda heurística.

Model-free

- No necesitan un **modelo** del entorno.
- Se basan en la capacidad de **aprendizaje**.
- Ej. Monte Carlo, TD.

A pesar de sus diferencias, tanto los métodos *model-based* como los *model-free* persiguen **un mismo objetivo**: el cálculo de las **funciones de valor óptimas** (v^* , q^*) y, por tanto, de una **política óptima** (π^*).

También comparten el mismo *modus operandi*:

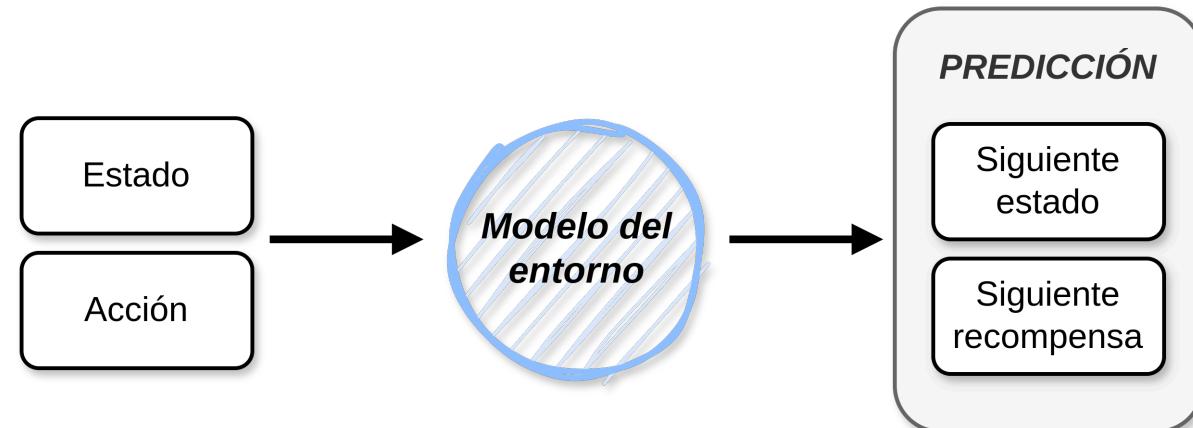
1. **Mirar hacia delante**, anticipando las consecuencias de posibles eventos futuros.
2. Calcular un «**valor de reserva**» (*backed-up*).
3. Utilizar ese valor como herramienta (*target*, **objetivo**) para aproximar v o q .

MODELOS Y PLANIFICACIÓN

¿Qué es un *modelo*?

Modelo de entorno

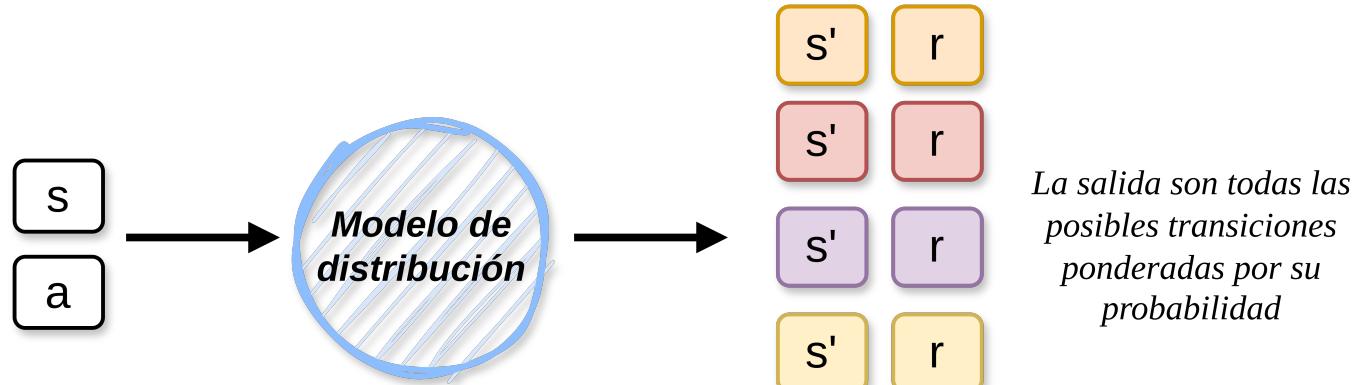
Representación formal/abstracta del entorno que permite al agente **predecir** las consecuencias de sus acciones. Esto es equivalente a **estimar transiciones**.



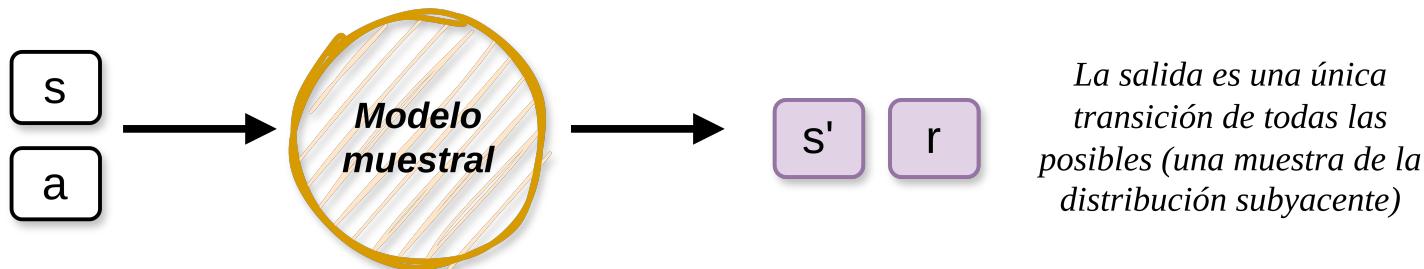
Si el entorno es **estocástico**, son posibles múltiples estados/recompensas siguientes, cada uno con diferente probabilidad.

Algunos modelos ofrecen una descripción de todas las probabilidades de transición. Estos se denominan **modelos de distribución** (*distribution models*).

Otros modelos solamente ofrecen como salida una de todas las posibles transiciones. Como esta transición se *muestra* en base a su probabilidad con respecto a todas las transiciones posibles, se denominan **modelos muestrales** (*sample models*).



La salida son todas las posibles transiciones ponderadas por su probabilidad



La salida es una única transición de todas las posibles (una muestra de la distribución subyacente)

En **programación dinámica**, por ejemplo, se trabaja con **modelos de distribución**, ya que conocemos las dinámicas del MDP:

$$p(s', r|s, a)$$

Los modelos de distribución son más robustos que los muestrales, ya que pueden utilizarse igualmente para producir muestras.

No obstante, en la práctica es mucho más fácil obtener un modelo muestral.

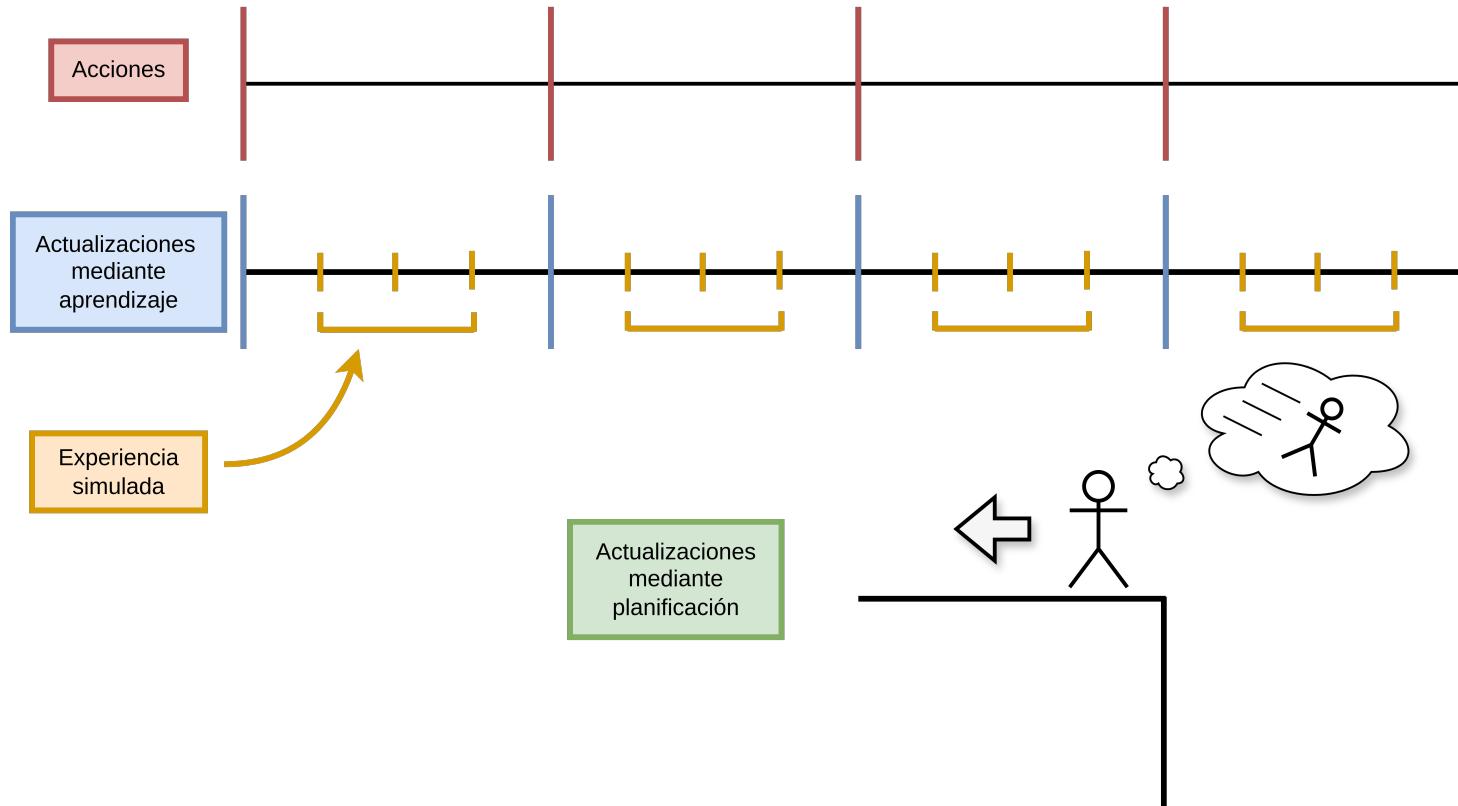
Un **modelo muestral** puede emplearse para simular un episodio completo (de entre todos los posibles).

Un **modelo de distribución** permite obtener todos los posibles episodios y sus probabilidades.

- Facilitan la *evaluación de riesgos* asociados a una determinada transición.

Los modelos permiten imitar/generar **experiencia**. Es decir, permiten **simular** las dinámicas del entorno modelado y producir **experiencia simulada**.

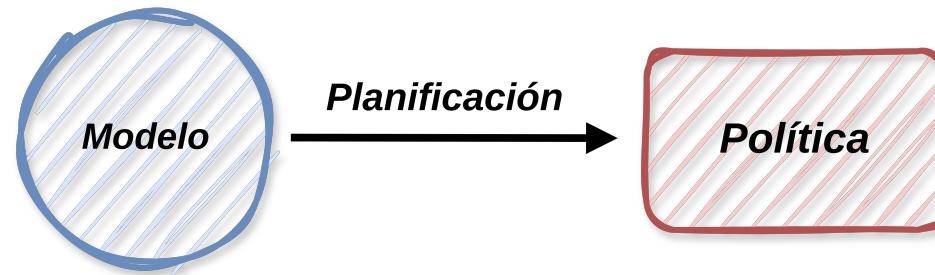
- Permiten la toma de decisiones informada sin necesidad de interactuar con el entorno real.
- 💡 La experiencia muestreada del modelo puede interpretarse como **imaginar** posibles escenarios que pueden ocurrir en el entorno real, y que dependen del entendimiento que se tiene del entorno.



Planificación

Planificación

Cualquier proceso computacional que toma un **modelo** como entrada y produce o mejora una **política** que interactúa con el entorno modelado.



- *Definir una «estrategia» para actuar sobre el entorno modelado.*

Planificación sobre el espacio de estados

Búsqueda de una política óptima o camino óptimo hacia un objetivo a lo largo del **espacio de estados**.

- Las acciones producen transiciones entre **estados** y las **funciones de valor** se calculan sobre dichos estados.
- Es el tipo de planificación comúnmente empleada en RL.

- *Es el tipo de planificación que vamos a considerar.*

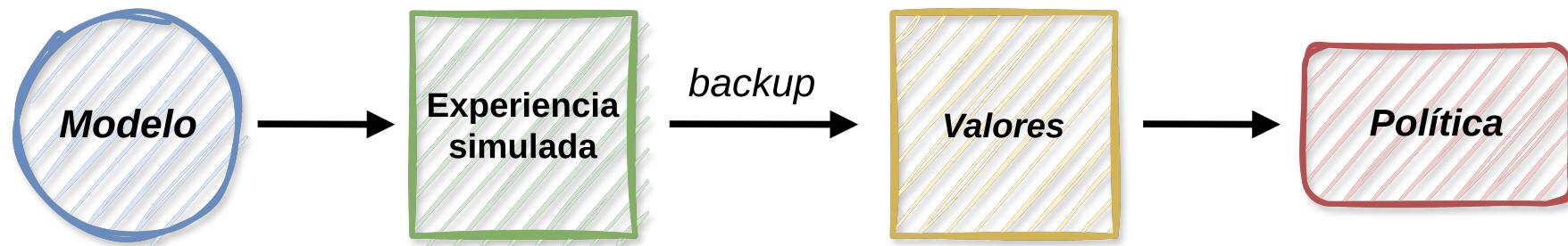
Planificación sobre el espacio de planes

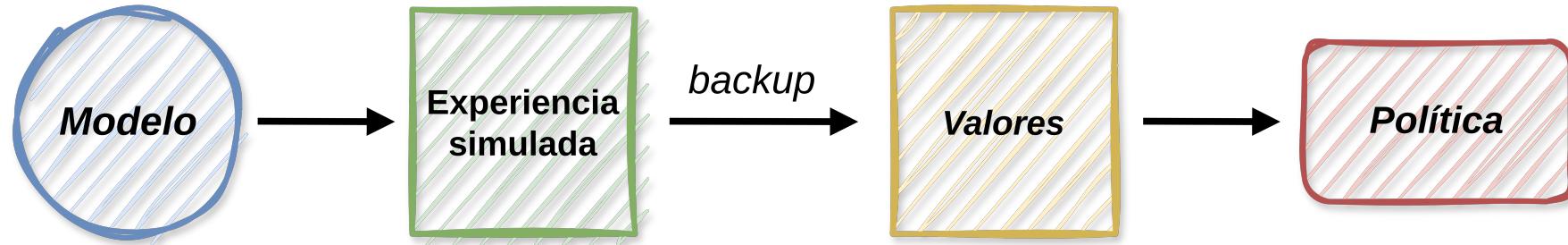
Búsqueda a lo largo del espacio de planes/estrategias.

- Las operaciones llevadas a cabo consisten en transformar un plan/estrategia en otro/a diferente.
- Las funciones de valor (si las hay) se definen sobre el **espacio de planes**.
- Estos métodos de planificación no son apropiados para problemas de decisión secuenciales y estocásticos como los que se abordan en RL, pero son comunes en otros ámbitos de la IA.

Todo método de **planificación sobre el espacio de estados** guarda una misma estructura:

- 1. Calcular las funciones de valor** como paso intermedio y fundamental de cara a mejorar la **política** existente.
- Dicho cálculo se realiza mediante **actualizaciones** u **operaciones de backup** aplicadas a experiencia simulada.

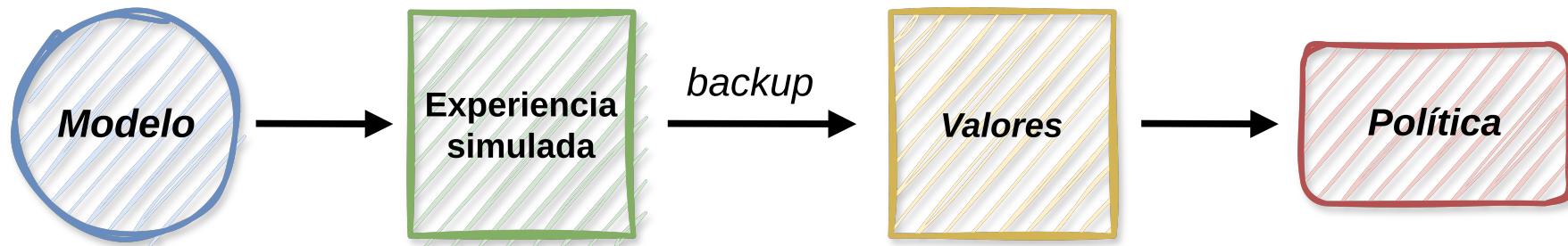




1. Se realizan pasadas a lo largo del espacio de estados.
2. Se genera, para cada estado, la distribución de todas las posibles transiciones.
3. Cada distribución se emplea para calcular un valor *backup*. Este se utilizará como objetivo en las actualizaciones de los valores (el *update target*).
4. Se actualiza el valor estimado del estado.

Otros métodos de planificación sobre el espacio de estados **se ajustan también a esta estructura**, variando únicamente:

1. La **forma de actualizar** los valores.
2. El **orden** en que se realizan estas actualizaciones.
3. Cuánto **tiempo** se mantiene la información retenida (*backed-up*).



La diferencia entre planificación y aprendizaje es que...

- la **planificación** emplea experiencia simulada, generada por un **modelo**,
- ...mientras que el **aprendizaje** se basa en experiencia real generada por el **entorno**.

Tanto **planificación** como **aprendizaje** buscan estimar las funciones de valor mediante operaciones *backup*.

- Esto permite que múltiples ideas y algoritmos puedan transferirse de unos métodos a otros.
- Por ejemplo, métodos de aprendizaje que emplean experiencia simulada en vez de experiencia real.

Random-sample one-step tabular Q-planning

Loop forever:

1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(S)$, at random
2. Send S, A to a sample model, and obtain
a sample next reward, R , and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S' :

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

La diferencia entre ***Q-learning*** y ***Q-planning*** es de dónde proviene la información empleada para mejorar la política (entorno real vs. modelo del entorno).

DYNA

La **experiencia** de un agente puede emplearse para:

1. Mejorar su modelo del entorno (hacerlo más preciso).

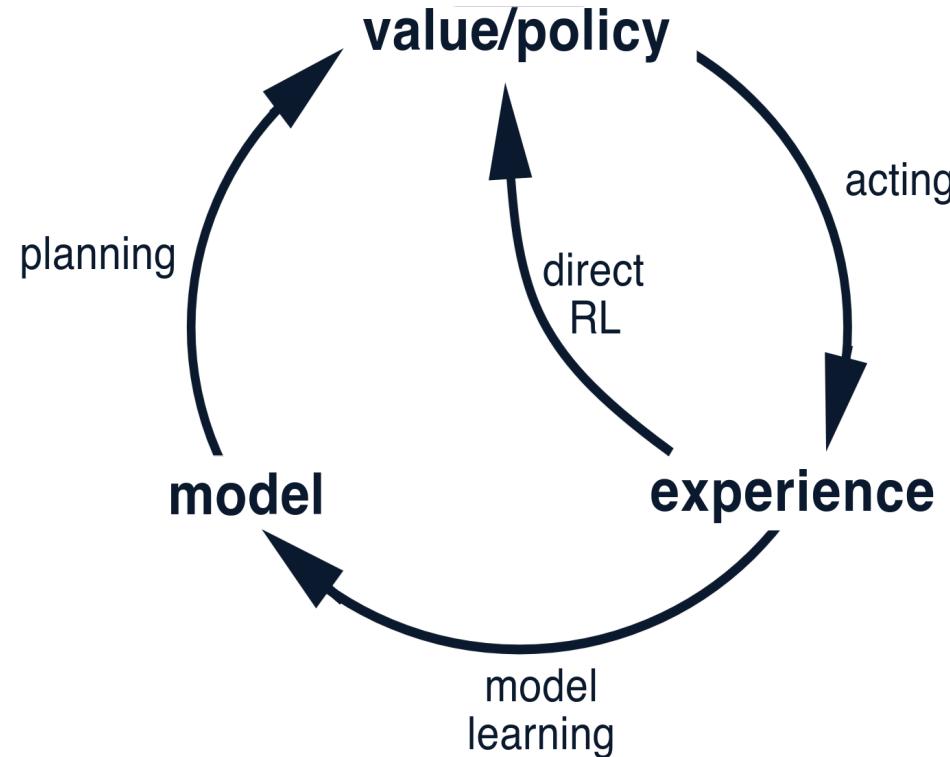
● Aprendizaje del modelo (***Model learning***)

2. Mejorar la estimación de la función de valor y la política

● RL directo (***DIRECT reinforcement learning***)

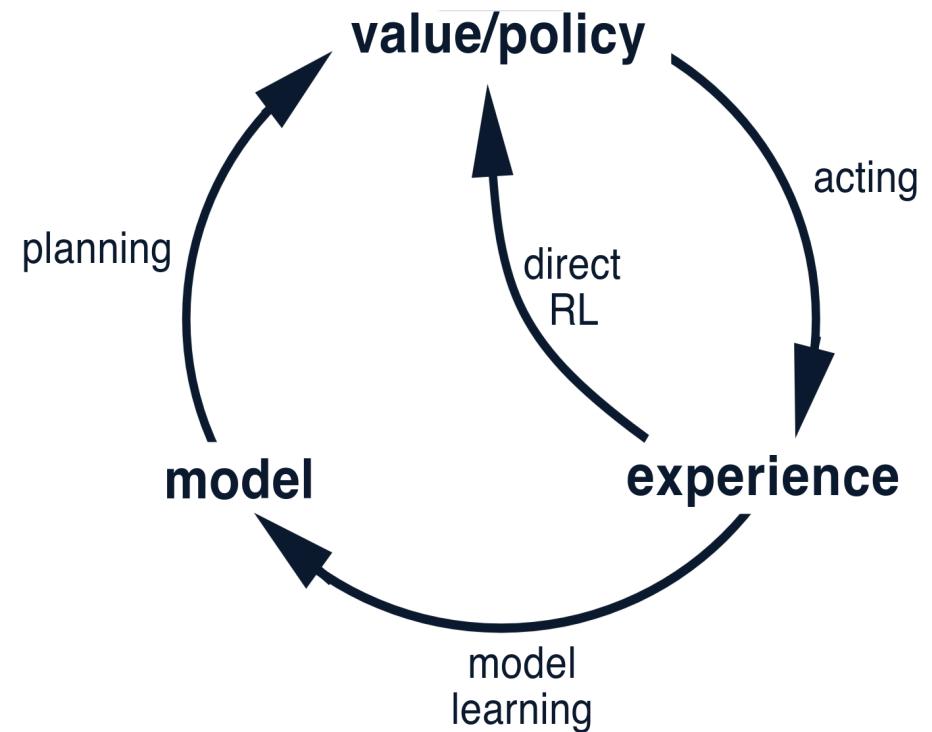
● RL indirecto (***INDIRECT reinforcement learning***)

La relación entre **experiencia, modelo, función de valor y política** es la siguiente:



La **experiencia** sirve para mejorar las funciones de valor y las políticas, ya sea **directa** o **indirectamente** a partir del modelo.

- **RL indirecto (INDIRECT RL)**: permite obtener una buena política con pocas interacciones con el entorno real.
 - Se basa en la **planificación** a partir del modelo.
- **RL directo (DIRECT RL)**: es una solución más simple que no se ve afectada por sesgos en el diseño del modelo.
 - **Aprendizaje** basado en la interacción directa con el entorno real.



¿En qué se diferencia **planificación de aprendizaje**?

¿En qué se diferencia **planificación de aprendizaje**?

 La **planificación** decide una secuencia de acciones antes de que se tomen, basándose en un modelo del entorno predefinido.

- Se emplea cuando contamos con un modelo fiable, o cuando la interacción con el entorno es costosa/arriesgada.
- Ej. programación dinámica.

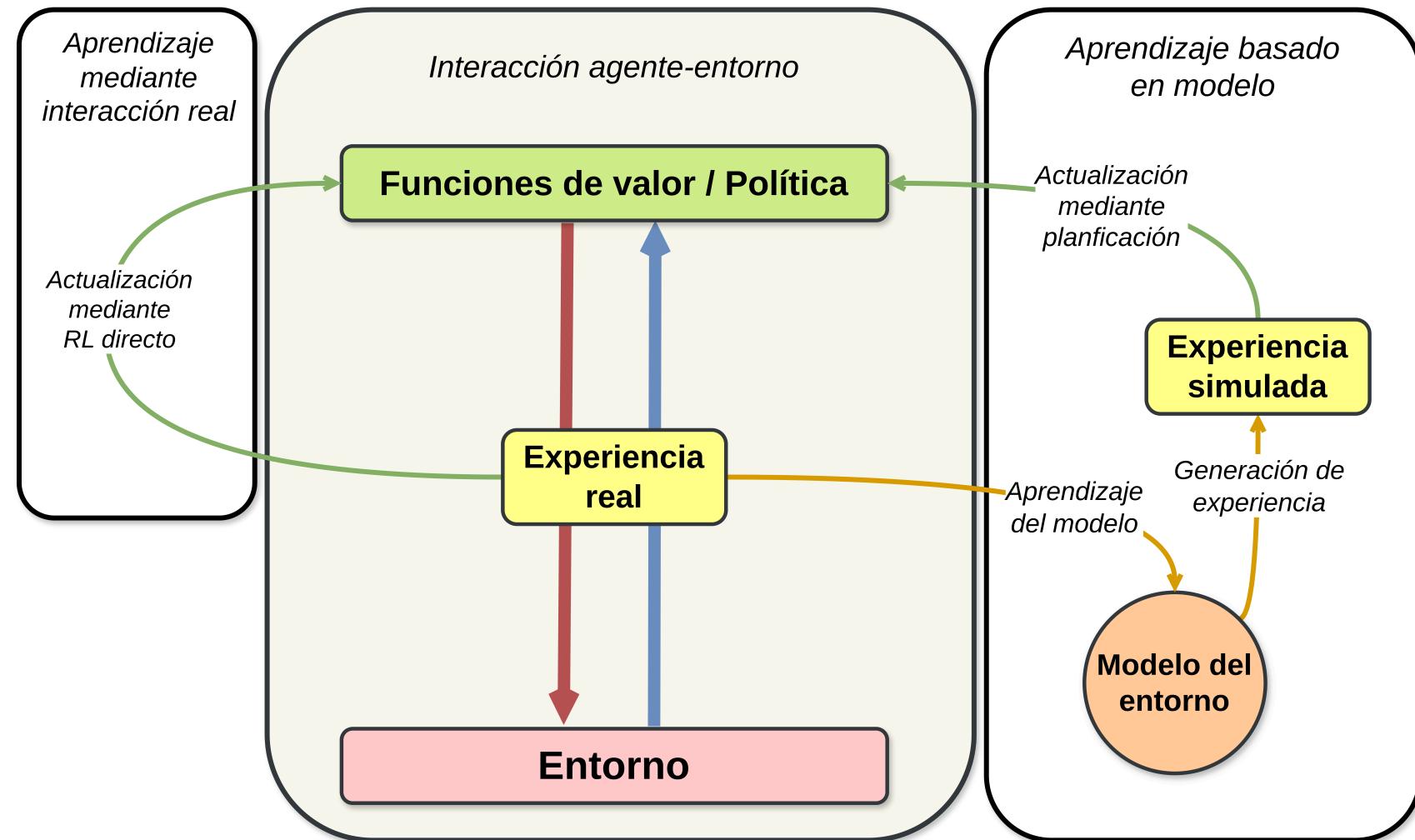
 El **aprendizaje** sí **requiere de la experiencia con el entorno**, ya sea para aprender las funciones de valor/política directamente (*direct RL*), o indirectamente (*indirect RL*) a través de un modelo ajustado con la experiencia obtenida.

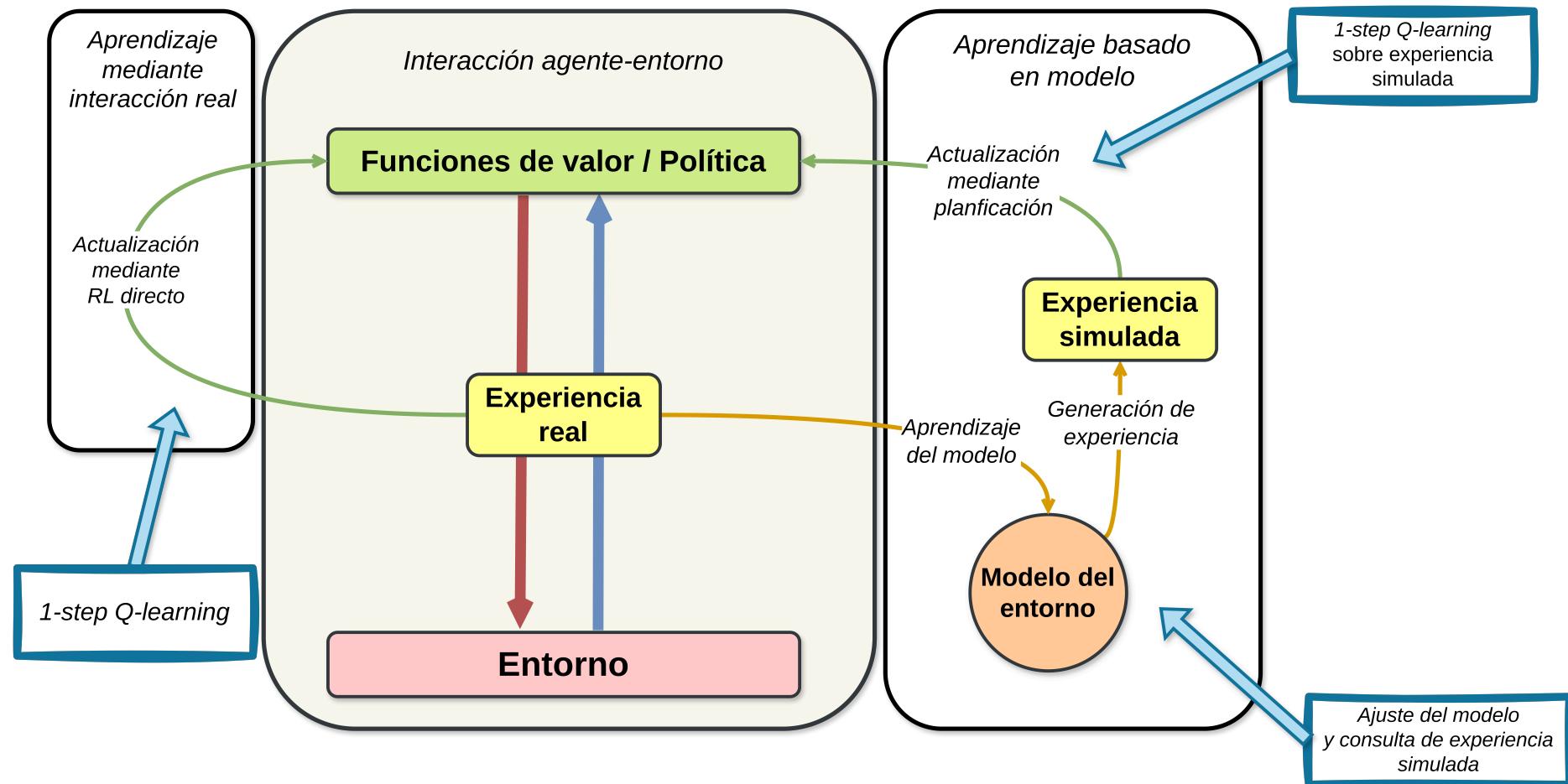
- Se emplea cuando no hay un modelo del entorno, o es incompleto.
- Ej. métodos basados en TD.

DYNA-Q

Dyna-Q

Arquitectura de agente que integra **planificación, actuación y aprendizaje** de forma ***online***.



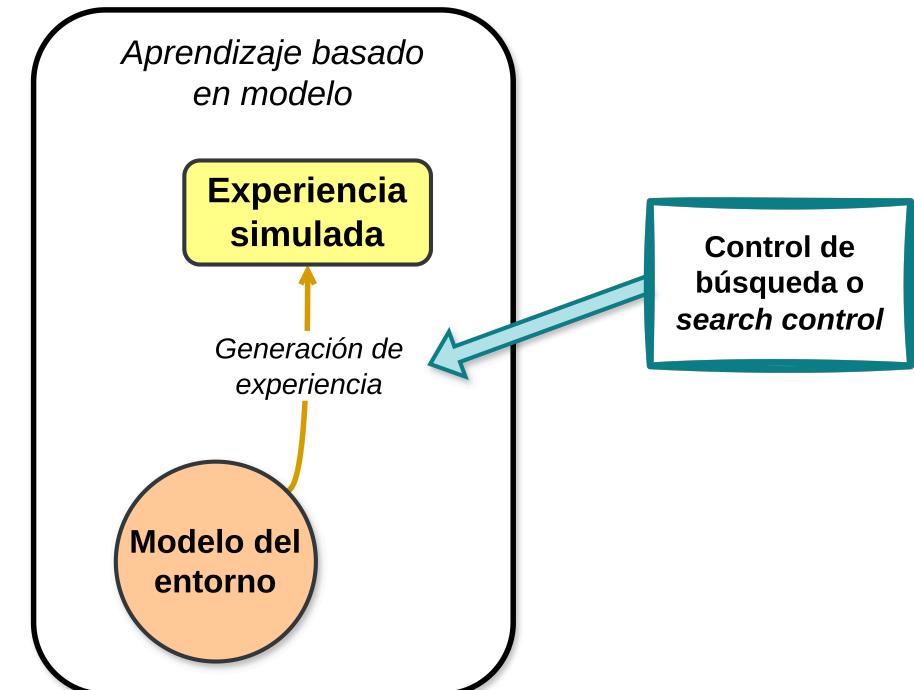


Al **modelo** se le consulta un par **estado–acción** previamente experimentado, y devuelve los siguientes **estado–recompensa** como predicción.

$$m(S_t, A_t) \rightarrow S_{t+1}, R_{t+1}$$

Contar con un buen modelo supone necesitar de menos experiencia **real** para alcanzar una misma política → se reducen las interacciones con el entorno.

Se denomina **control de búsqueda** (*search control*) al proceso de selección del estado–acción inicial en la **experiencia simulada** obtenida del **modelo** y empleada para aprender mediante **planificación**.



Generalmente, Dyna-Q emplea el **mismo método** para el **aprendizaje** a partir de experiencia **real** y **simulada** (ej. 1-step *Q-learning*).

Aprendizaje y planificación solamente varían en la **procedencia** de la **experiencia** empleada (entorno real o modelo del entorno).

Conceptualmente, la **planificación, actuación, aprendizaje del modelo y aprendizaje directo mediante RL** ocurren **simultáneamente**.

No obstante, la implementación de Dyna-Q de forma secuencial requiere dividir estos procesos dependiendo de sus **requisitos computacionales**:

- **Actuación**
- **Aprendizaje del modelo**
- **RL directo**

Ocurren rápidamente y prácticamente en paralelo.

- **Planificación**

Implica operaciones más costosas que requieren un mayor tiempo de ejecución.

Tabular Dyna-Q

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

Loop forever:

- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon\text{-greedy}(S, Q)$
- (c) Take action A ; observe resultant reward, R , and state, S'
- (d) $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
- (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Loop repeat n times:
 - $S \leftarrow$ random previously observed state
 - $A \leftarrow$ random action previously taken in S
 - $R, S' \leftarrow Model(S, A)$
 - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

Tabular Dyna-Q

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

Loop forever:

- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon\text{-greedy}(S, Q)$
- (c) Take action A ; observe resultant reward, R , and state, S'
- (d) $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
- (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Loop repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S, A)$

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

(a) Percibir estado

(b) Seleccionar acción

(c) **Actuar** y observar transición

(d) **RL directo**

(e) **Aprendizaje del modelo**

(f) **Planificación**

Si eliminamos **(e)** y **(f)**, tenemos el *1-step tabular Q-learning* convencional.

Los dos primeros pasos en **(f)** se corresponden con *search control*.

Tabular Dyna-Q

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

Loop forever:

- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon\text{-greedy}(S, Q)$
- (c) Take action A ; observe resultant reward, R , and state, S'
- (d) $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
- (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Loop repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S, A)$

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

Se realizan tantas **iteraciones de planificación** (n) como se especifiquen en (f).

Si $n = 0$, tenemos un agente de RL convencional (*direct RL*) que no planifica.

En la práctica, invertir en planificación puede suponer un **aprendizaje más rápido...**

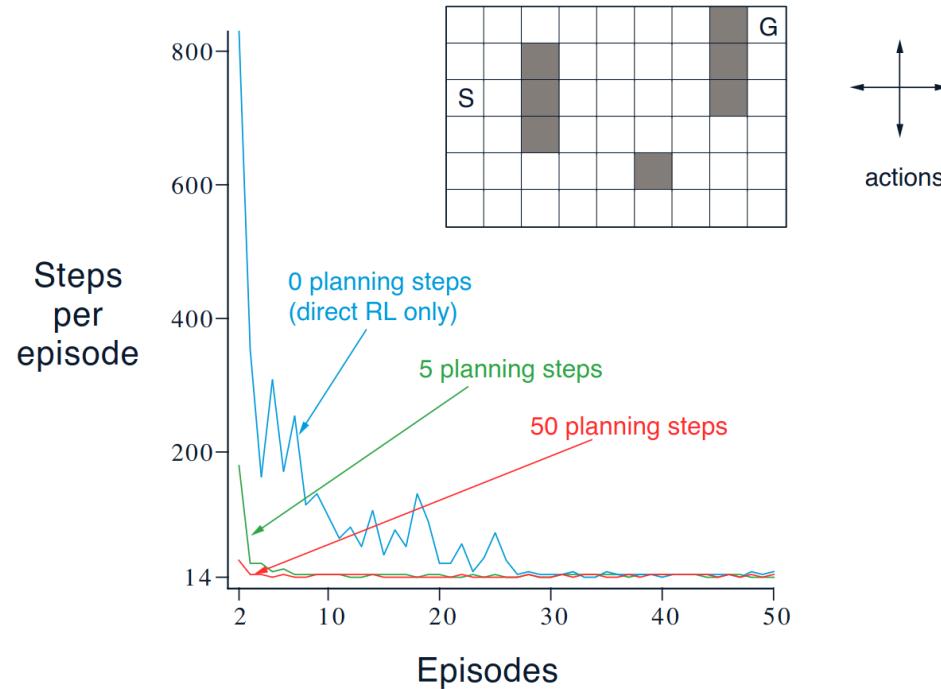


Figure 8.2: A simple maze (inset) and the average learning curves for Dyna-Q agents varying in their number of planning steps (n) per real step. The task is to travel from S to G as quickly as possible.

En la práctica, invertir en planificación puede suponer un **aprendizaje más rápido...**

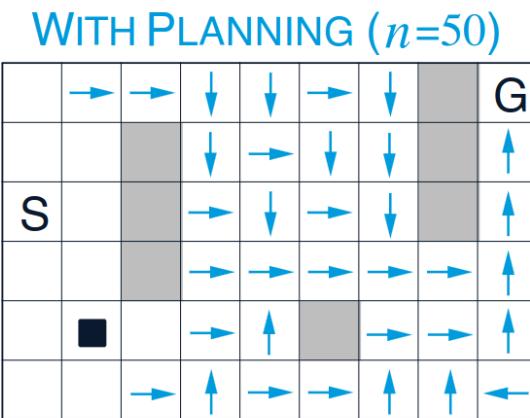
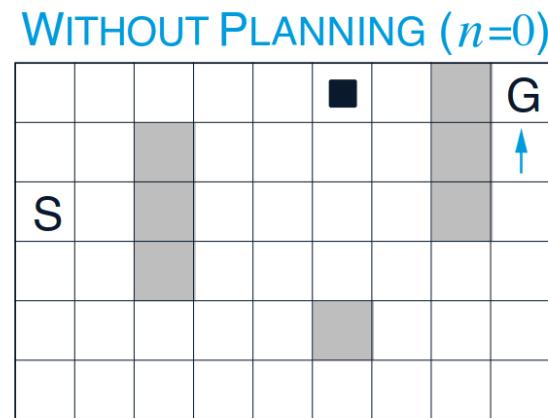


Figure 8.3: Policies found by planning and nonplanning Dyna-Q agents halfway through the second episode. The arrows indicate the greedy action in each state; if no arrow is shown for a state, then all of its action values were equal. The black square indicates the location of the agent. ■



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

In **Dyna-Q**, **learning** and **planning** are accomplished by exactly the same algorithm, operating on **real experience** for **learning** and on **simulated experience** for **planning**. Because planning proceeds incrementally, it is trivial to **intermix** planning and acting. Both proceed as fast as they can.

The agent is always reactive and always deliberative, responding instantly to the latest sensory information and yet always planning in the background. Also ongoing in the background is the **model-learning process**. As new information is gained, the model is **updated** to better **match reality**. As the model changes, the ongoing **planning** process will gradually compute a different way of behaving to **match** the new model.

Hablando en código...

```
def dynaq(self, env, n_episodes, plan_steps, alpha, gamma, epsilon):

    q_table, model = {}, {}

    for _ in range(n_episodes):
        s = env.start_pos
        while True:

            if s not in self.q_table:
                q_table[s] = {action: 0 for action in env.actions}

            a = e_greedy(s, actions, epsilon)

            s_next, r, end = env.step(s, a)

            if s_next not in self.q_table:
                self.q_table[s_next] = {
                    action: 0 for action in env.actions}

            ...
```

```
...  
  
# Q-learning update  
td_target = r + gamma * max(q_table[s_next].values())  
td_error = td_target - q_table[s][a]  
q_table[s][a] += alpha * td_error  
  
# Update Model  
if s not in model:  
    model[s] = {}  
model[s][a] = (s_next, r)  
  
...
```

```
...
```

```
# Planning
for _ in range(plan_steps):
    sim_s = random.choice(list(self.model.keys()))
    sim_a = random.choice(list(self.model[sim_s].keys()))

    sim_s_next, sim_r = self.model[sim_s][sim_a]

    sim_td_target = sim_r + self.gamma * \
        max(self.q_table[sim_s_next].values())
    sim_td_error = sim_td_target - self.q_table[sim_s][sim_a]
    self.q_table[sim_s][sim_a] += self.alpha * sim_td_error

    s = s_next

if end:
    break
```

¿Qué ocurre si el modelo
NO es preciso?

Si un **modelo** del entorno **no es preciso**, las transiciones predichas no son correctas.

- Es decir, la función de transición $p(s, a|s', r)$ modelada **no se ajusta a la realidad**.

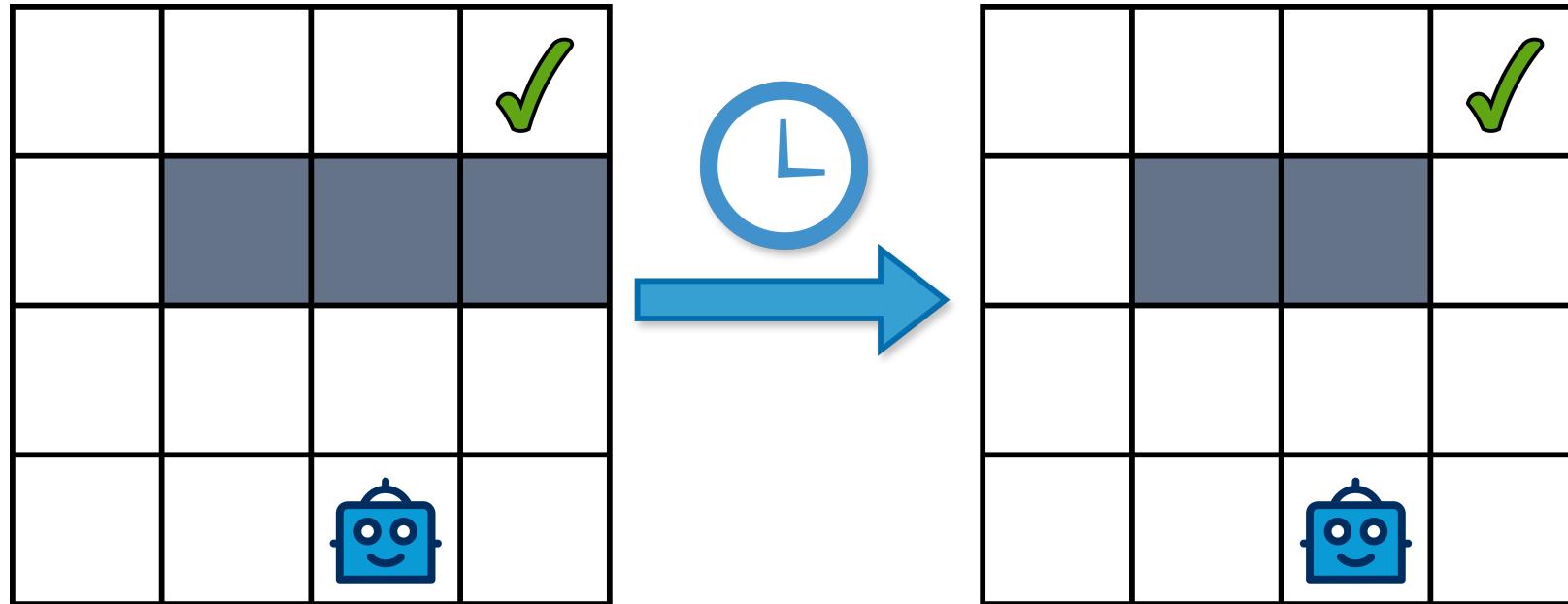
Los errores en los modelos se deben a la **estocasticidad** de los entornos en combinación con la **falta de experiencia**.

Si el entorno cambia con el tiempo, un modelo aprendido puede dejar de ser útil.

Otra posibilidad es que el modelo empleado generalice de forma imperfecta, en caso de que sea un modelo obtenido mediante aproximación de funciones.

Por tanto...

1. Modelos que no contienen todas las **transiciones** → **MODELOS INCOMPLETOS**
2. Modelos imprecisos porque el **entorno** ha variado → **MODELOS DESACTUALIZADOS**



¿Qué ocurre si **planificamos** con modelos imprecisos?

- Si el modelo es **incompleto**, necesitamos tiempo para **almacenar transiciones y obtener experiencia**.
- Si el modelo **difiere de la realidad** porque el **entorno** real ha cambiado, la política/función de valor puede actualizarse en una **dirección errónea**.

Cuando el modelo es incorrecto, la planificación da lugar a **políticas subóptimas**.

En algunos casos, la política subóptima obtenida mediante planificación puede llevar al **descubrimiento y corrección** de los **errores** del modelo.

- Esto ocurre principalmente cuando el modelo es **optimista** y se predicen mejores recompensas o transiciones de las que son realmente posibles.
- La política subóptima trata de explotar estas oportunidades y de esta forma descubre que **sus expectativas son falsas**.

Para el agente, es necesario mantener la certeza de que su modelo se encuentra correctamente actualizado.

La **exploración** le permite asegurar la precisión del modelo, al mismo tiempo que se dedica a la **explotación** de la política óptima (asumiendo que es correcta).

Una ventaja de **Dyna-Q** es que permite planificar aun contando con un modelo incompleto, ya que sólo muestrea estados y acciones que hayan sido **previamente visitados**. Véanse los primeros dos pasos de **(f)**:

Tabular Dyna-Q

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

Loop forever:

- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon\text{-greedy}(S, Q)$
- (c) Take action A ; observe resultant reward, R , and state, S'
- (d) $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
- (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Loop repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S, A)$

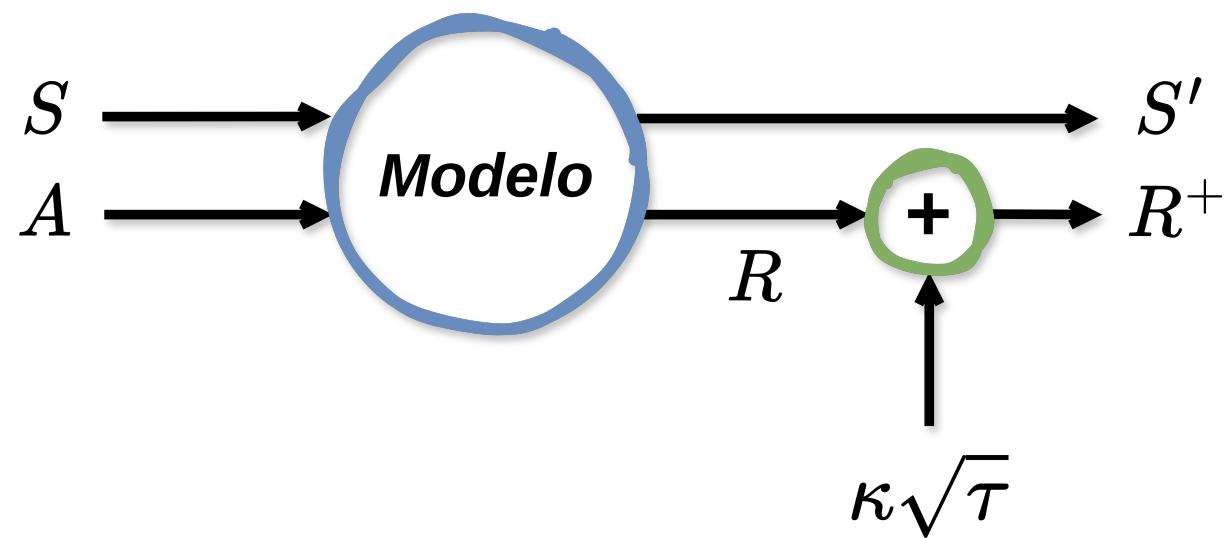
$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

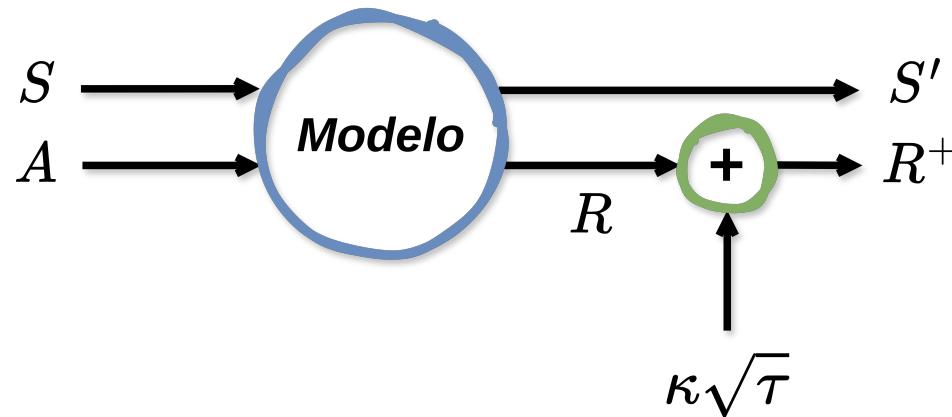
DYNA-Q+

Es más probable que un **modelo** esté **desactualizado** en **estados/acciones** que lleven más **tiempo sin visitarse**.

Para asegurar que el agente revisita estados periódicamente, podemos añadir un «**bonus**» a la recompensa empleada en la planificación.

Esta es la propuesta del algoritmo **Dyna-Q+**.

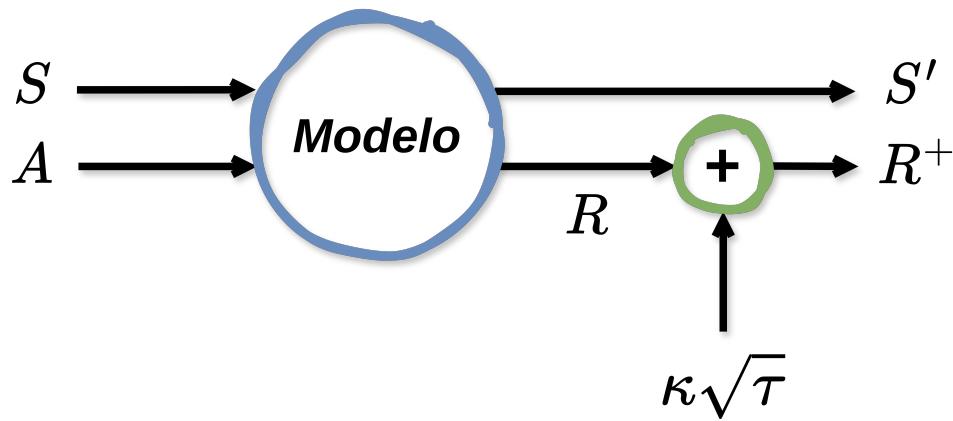




Dyna-Q+ mantiene un registro de los **time steps transcurridos desde la última ocurrencia** de cada par acción-estado en el entorno real.

Cuanto más tiempo haya transcurrido, se asume una mayor probabilidad de que las dinámicas asociadas a un par acción-estado hayan cambiado y que, por tanto, el modelo sea **incorrecto**.

- Para favorecer que «se vuelva intentar», se añade un **bonus** a la recompensa de las experiencias simuladas que impliquen al par poco visitado.



Si la recompensa de una transición es r y la transición no se ha probado desde hace τ time steps, la recompensa de esa transición pasa a ser $r + \kappa\sqrt{\tau}$, siendo κ un valor pequeño.

$$r' = r + \kappa\sqrt{\tau}$$

Se promueve que el agente revisite estados cada X tiempo para confirmar que el modelo esté actualizado.

Se actualiza la **política** para dirigir al agente a **estados/acciones no visitados** desde hace tiempo.

De esta forma, mantenemos el **modelo actualizado**.

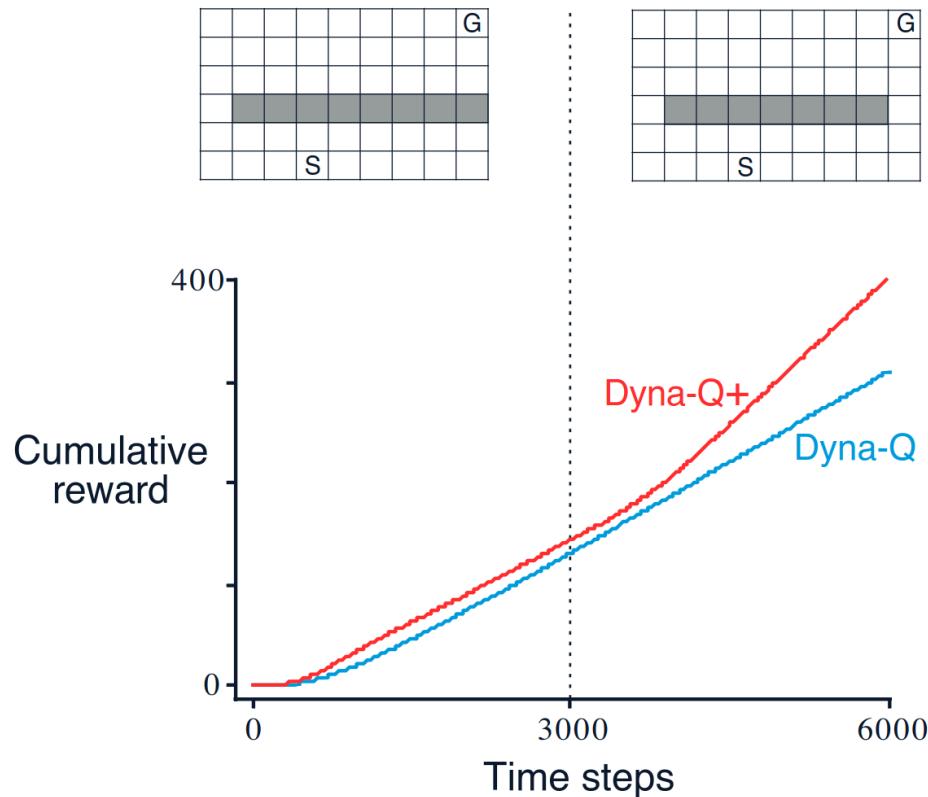


Figure 8.5: Average performance of Dyna agents on a shortcut task. The left environment was used for the first 3000 steps, the right environment for the rest.

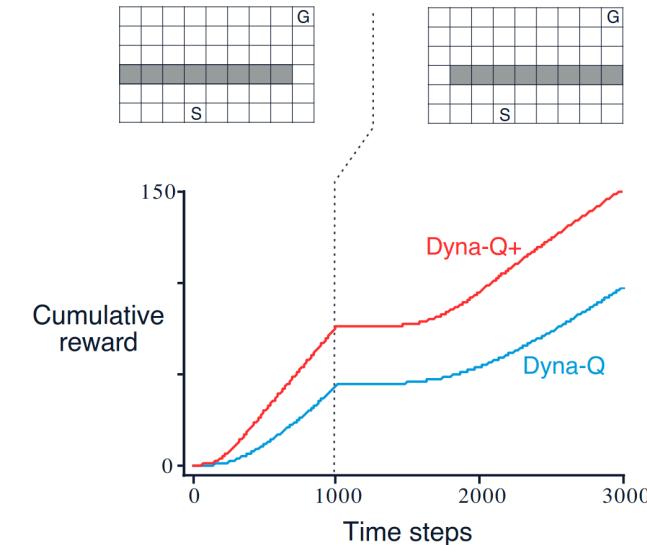


Figure 8.4: Average performance of Dyna agents on a blocking task. The left environment was used for the first 1000 steps, the right environment for the rest. Dyna-Q+ is Dyna-Q with an exploration bonus that encourages exploration. ■

TRABAJO PROPUESTO

- Analiza esta **implementación de DynaQ** y trata de hacer la tuya:
 - <https://github.com/manjavacas/rl-temario/tree/main/ejemplos/dynaq>
 - Pruébala en un entorno de **Gymnasium** (ej. *Frozen Lake*).
- Leer capítulos **8.6. Trajectory Sampling** y **8.11. Monte Carlo Tree Search** de Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction.
 - ¿Conoces alguna aplicación conocida donde se haya utilizado **Monte Carlo Tree Search**?

Bibliografía y recursos

- **Capítulo 8** de Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction.
- <https://youtu.be/lMutbeOHtc?si=XFPMLOxD6MZNQgUy>
- https://youtu.be/jlC0TFTUefs?si=O_L4fL6ddW8X1N_U
- <https://youtu.be/aUjuBvqJ8UM?si=lU11f24dn27LnISD>
- <https://www.davidsilver.uk/wp-content/uploads/2020/03/dyna.pdf>

APRENDIZAJE POR REFUERZO

Planificación

Antonio Manjavacas

manjavacas@ugr.es