

# Optimización mediante algoritmos por refuerzo

## Robótica



Alberto Díaz y Raúl Lara

Curso 2022/2023

Departamento de Sistemas Informáticos

License CC BY-NC-SA 4.0

# Paradigmas de aprendizaje en *Machine Learning*

---

**Supervisado** : Se aprende de ejemplos con sus correspondientes respuestas.

- Problemas de regresión y clasificación.

**No supervisado** : Búsqueda de patrones en datos no etiquetados.

- Problemas de *clustering*, reducción de la dimensionalidad, recodificación, ...

**Por refuerzo** : Se aprende a través de la experiencia a base de recompensas.

- Problemas de aprendizaje de políticas de decisión.

"De varias respuestas dadas al mismo hecho, las seguidas de satisfacción para el animal estarán, en igualdad de condiciones, más firmemente conectadas con este, de modo que tenderán a repetirse; las seguidas de incomodidad para el animal tendrán, en igualdad de condiciones, sus conexiones con el hecho debilitadas, de modo que tenderán a ocurrir menos. Cuanto mayor sea la satisfacción o el malestar, mayor será el refuerzo o el deterioro del vínculo."

**- Edward Thorndike - Law of Effect -**

# Aprendizaje por refuerzo (RL)

---

Área del *machine learning* donde **los agentes aprenden interactuando** :

- **Imita** de manera fundamental el **aprendizaje** de muchos **seres vivos** .
- Esa interacción produce tanto resultados deseados como no deseados.
- Se entrena con la **recompensa o castigo** determinados para dicho resultado.
- El agente tratará de maximizar la recompensa a largo plazo.

Se utiliza principalmente en dos áreas hoy en día:

- **Juegos** : Los agentes aprenden las reglas y las jugadas jugando.
- **Control** : Los agentes aprenden en entornos de simulación las mejores políticas de control para un problema determinado.

---

Un ejemplo curioso es el publicado en <https://www.nature.com/articles/nature14236>, donde describen cómo un agente aprende a jugar a 49 juegos de Atari 2600 llegando a un nivel de destreza comparable al humano.

# Terminología

---

**Agente inteligente** : Entidad que interactúa con el **entorno** .

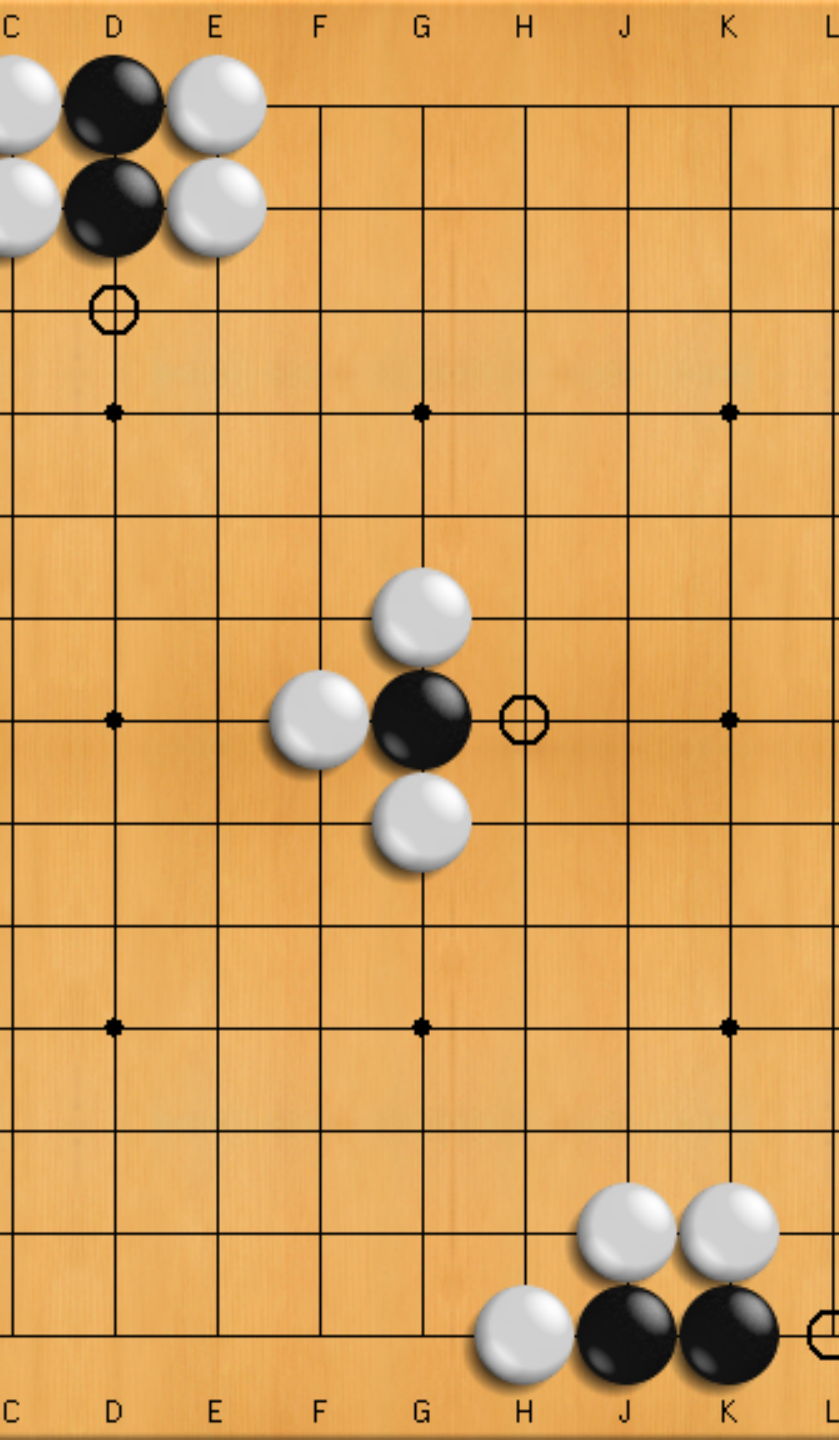
- Usaremos indistintamente los conceptos de agente, agente inteligente y robot.
- **Entorno** : El mundo físico o virtual con el que interactúa el agente.

**Estado** y **observación** : Información que el agente obtiene del entorno:

- Estado  $S_i$ : Descripción **completa** del estado del entorno (e.g. juego del Go).
- Observación  $O_i$ : Descripción **parcial** del estado del entorno (e.g. Warcraft II).

**Espacio de acciones** : Conjunto de acciones que puede realizar el agente:

- **Discreto** : El conjunto es finito (e.g. juego del Go).
- **Continuo** : El conjunto es infinito (e.g. vehículo autónomo).



## Ejemplo #1: Juego del Go

- Agente: Robot que juega al Go.
- Entorno/mundo: El tablero en el que se juega.
- Estado: Colocacion concreta de las piedras.
- Observación: Estado (sin información oculta).
- Espacio de acciones (finito): Poner piedra en una casilla vacía.



## Ejemplo #2: Warcraft II

---

- Agente: Robot que juega al Warcraft II.
- Entorno/mundo: Pantalla en la que se juega.
- Estado: Situación de la pantalla en un momento determinado.
- Observación: Lo que el agente ve en un instante determinado (sin la niebla de guerra).
- Espacio de acciones (finito): Mover unidades, construir edificios, ...





## Ejemplo #3: Coche autónomo

---

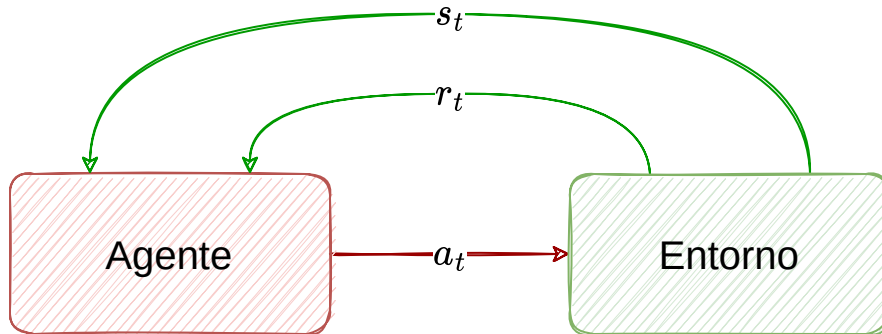
- Agente: Robot que conduce el vehículo.
- Entorno/mundo: El continente en el que se encuentra el vehículo.
- Estado: Estado del continente en un momento determinado.
- Observación: Lo que el agente ve por sus sensores en un instante determinado.
- Espacio de acciones (infinito): Girar el volante un determinado ángulo, aumentar y disminuir aceleración, ...



# Modelo de interacción agente-entorno

---

El proceso de aprendizaje por refuerzo es el siguiente:



1. El agente lee un estado  $S_0$  del entorno.
2. De acuerdo a  $S_0$ , realiza la acción  $A_0$ .
3. El entorno pasa al nuevo estado  $S_1$ .
4. El agente recibe una recompensa  $R_1$ .

---

Este bucle produce una secuencia de estados, acciones y recompensas:

$$S_0, A_0, R_1, S_1, A_1, \dots$$

# Hipótesis de la recompensa

---

El agente quiere **maximizar la recompensa acumulada** (rendimiento esperado).

- La recompensa es el *feedback* que el entorno da al agente.
- Le permite saber al agente si la acción es buena o no.

La recompensa acumulada es la suma de todas las recompensas de la secuencia.

$$R(\tau) = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

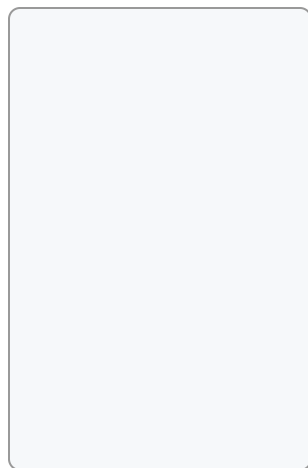
Sin embargo, las recompensas no tienen por qué tener todo su valor siempre.

- De ahí el factor de ajuste en la recompensa, o  $\gamma \in [0, 1]$ .

# Ajuste en la recompensa

---

Supongamos el siguiente ejemplo no tan raro de la realidad que nos espera:



- El agente se mueve una celda cada  $t$ .
- Los zombies también.
- Objetivo: Conseguir el mayor número de puntos de supervivencia...
- Si nos arañan perdemos.

$\gamma$  indica si interesan más recompensas a **corto** ( $\gamma \approx 0$ ) o a **largo** ( $\gamma \approx 1$ ) **plazo**.

Las recompensas que llegan antes tienen más probabilidades de suceder.

Esto es porque son más predecibles que las recompensas a largo plazo

# Tareas y problemas

---

Se entiende por tarea a una instancia de un problema. Existen dos tipos diferenciados:

Tenemos dos tipos bien diferenciados de tareas:

- **Episódicas** : Poseen estado inicial y terminal o final (e.g. Sonic the Hedgehog).
- **Continuas** : Tarea que no posee estado terminal (e.g. vehículo autónomo).

Por cierto, aprovechamos para definir:

- **Probabilidad de transición**:  $P(S_{t+1} | (S_t, A_t), (S_{t-1}, A_{t-1}), \dots, (S_1, A_1))$
- **Probabilidad de recompensa**:  $P(R_{t+1} | (S_t, A_t), (S_{t-1}, A_{t-1}), \dots, (S_1, A_1))$

# Resolución de problemas

---

Política  $\pi$  (de *policy*): Función que asigna una acción  $A$  a un estado determinado  $S$ .

$$\pi(S) = A$$

Objetivo: Encontrar la función  $\pi$  que **maximiza el rendimiento esperado cuando el agente actúa de acuerdo con ella.**

Existen dos métodos para aprender esta función  $\pi$ :

- **Directo** : Qué acción debe realizar en el estado actual.
- **Indirecto** : Qué estados son mejores para tomar la acción que lleva a esos estados.

# Métodos directos (basados en políticas)

---

En estos métodos **aprendemos directamente la función  $\pi$**  . Existen dos tipos:

## Determinista

Devuelve **siempre la misma acción** para un estado determinado.

$$\pi(S) = A$$

---

Por ejemplo:

$$\pi(S_i) = \{\blacktriangleright\}$$

## No determinista

Devuelve una **distribución de probabilidad** sobre las acciones.

$$\pi(S) = P[A|S]$$

---

Por ejemplo:

$$\pi(S_i) = \{(\blacktriangleleft, 0.3), (\blacktriangleright, 0.5), (\blacktriangledown, 0.1), (\blacktriangle, 0.1)\}$$



# Métodos indirectos (basados en valores)

---

Aprendemos una función  $V$  que **relaciona un estado con su valor estimado**.

- Valor: Recompensa acumulada si empieza en ese estado y se mueve al mejor estado.
- El agente selecciona la acción de mayor valor

## Valor estado

$$V_{\pi}(S) = \mathbb{E}_{\pi}[R_t | S_t = S]$$

## Valor par estado-acción

$$V_{\pi}(S, A) = \mathbb{E}_{\pi}[R_t | S_t = S, A_t = A]$$

---

Independientemente de la función elegida, el resultado será la recompensa esperada.

- Ojo: **Para** calcular **cada valor de un estado** (o par estado-acción), hay que **sumar todas las recompensas** que puede obtener un agente si empieza en ese estado.

# Ecuación de Bellman

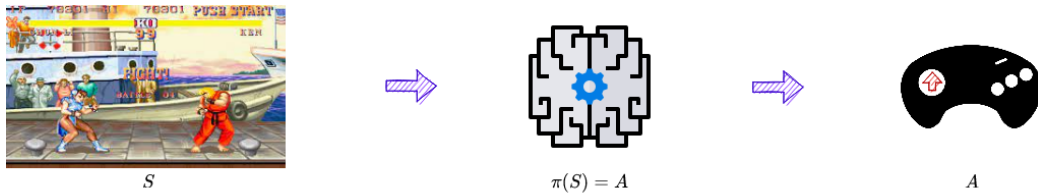
---

Simplifica el cálculo del **valor** del estado o del par estado-acción.

**ME HE QUEDADO AQUÍ**

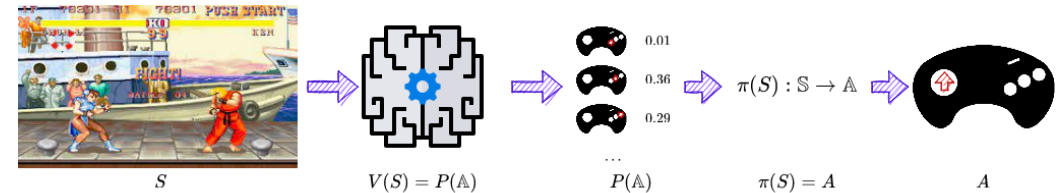
# Comparativa entre métodos directos e indirectos

## Métodos directos



La **política óptima** se encuentra **entrenando** la política **directamente**.

## Métodos indirectos



Encontrar una **función de valor óptima** lleva a tener una **política óptima**.

Por lo tanto Independientemente del método, tendremos una política.

- Pero en el caso de los métodos basados en valores no la entrenamos.
- Será una "simple" función que usará los valores dados por la función  $V$ .

# Estrategia epsilon-greedy

---

Política sencilla para elegir acción que mantiene el equilibrio exploración/explotación.

- El agente elige una acción de forma aleatoria con probabilidad  $\epsilon$
- La mejor acción conocida con probabilidad  $1 - \epsilon$ .

Por lo general, se empieza con un  $\epsilon$  alto (muchacha exploración).

- Según el agente aprende más sobre el entorno,  $\epsilon$  disminuye.
- Ha explorado mucho, así que puede centrarse en explotar lo conocido.

# ***Q-Learning***

---

Es un método indirecto (método basado en valores)

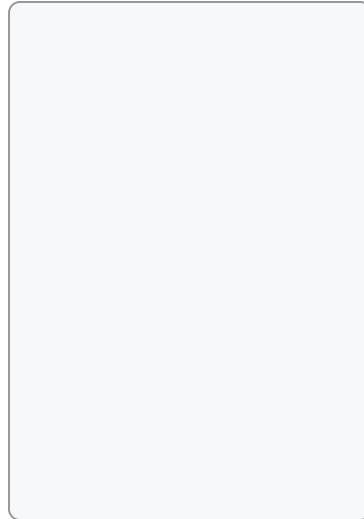


# Ejemplo: Hambre y zombies

---

Objetivo: Utilizar técnicas de RL para que el superviviente llegue a su destino.

---



---

Hay que comenzar considerando los estados, las acciones y las recompensas.



# Estados

---

El agente se encuentra en un estado y toma una acción de acuerdo a este.

Espacio de estados: Todas las situaciones posibles en las que se puede encontrar el agente.

- Debe contener información suficiente para tomar una decisión correcta.

En el ejemplo, son todas las posiciones que podría ocupar el agente (35).

- Podríamos complicarlo más, por ejemplo, obligando a llevar comida.
- Esto implicaría los 35 estados con y sin comida encima ( $35 + 35 = 70$ ).
- Pero nos quedaremos con el ejemplo simple.

# Acciones

---

El agente se encuentra con uno de los 35 estados y realiza una acción.

- 5 acciones posibles: arriba, abajo, izquierda, derecha y coger comida.

**Espacio de acciones** : Conjunto de todas las acciones posibles para un estado.





# Recompensas

---

El superviviente está motivado por la recompensa, así que aprenderá a:

- Encontrar la comida y el objetivo.
- Evitar las zonas infestadas de zombies.

Algunos puntos a tener en cuenta para el agente:

- Alta recompensa por llegar a las montañas  (+1000); es el objetivo.
- Ligera recompensa por encontrar comida   (+10) porque está bien.
- Penalización si llega a un zombie  (-50) porque no interesa en absoluto.

Es importante tener en cuenta que la recompensa no siempre es inmediata:

- Puede haber tramos sin nada hasta llegar a un estado muy bueno.

*OpenAI Gym*

[CREAR EL ENTORNO DE OPENAI GYM PARA EL EJEMPLO:

<https://towardsdatascience.com/creating-a-custom-openai-gym-environment-for-stock-trading-be532be3910e>]

# El entorno *OpenAI Gym*

---

OpenAI Gym es una biblioteca de entornos de aprendizaje por refuerzo.

- Proporciona entornos de juego para probar nuestros agentes desarrollados.

Se encarga de proporcionar toda la información que el agente necesitaría:

- Entorno, posibles acciones y recompensas, estado actual, ...
- Sólo tenemos que preocuparnos de la lógica del agente.

El ejemplo anterior se ha implementado como entorno para practicar con él.

# Instalación

---

La biblioteca está disponible a través de Pypi:

```
pip install gym
```

Una vez instalada, podemos cargar el entorno del juego y mostrar su aspecto:

```
import gym

env = gym.make("Starvation and Zombies").env
env.render()
```



# Propiedad de Márkov

---

El estado futuro del proceso depende del estado actual, y no de los anteriores.

- Es un estado que cumplen ciertos procesos estocásticos.
- Definida por Andréi Markov en 1906 en su Teoría de Cadenas de Márkov.

Al proceso que satisface esta propiedad se denomina **Proceso de Márkov**.

- Concretamente se denominan Procesos de Márkov de **primer orden**.
- La definición se puede extender a  $n$  estados anteriores (proceso de orden  $n$ ).
- Si el espacio de estados es finito, equivale a una **cadena de Márkov**.

Si hay que quedarse con algo, nos dice que nuestro agente sólo necesita el estado actual para decidir qué acción tomar.

Se asume que el proceso de decisión de un agente es un MDP:

- $P(s_{t+1} | (s_t, a_t), (s_{t-1}, a_{t-1}), \dots, (s_1, a_1)) = P(s_{t+1} | (s_t, a_t))$
- $P(r_{t+1} | (s_t, a_t), (s_{t-1}, a_{t-1}), \dots, (s_1, a_1)) = P(r_{t+1} | (s_t, a_t))$

Los procesos de decisión de un

Sistemas de toma de decisiones basados en procesos de Márkov. Incluyen:

- $S$ : Conjunto finito de estados.
- $A$ : Conjunto finito de acciones.
- $P(s_i | (s_j, a))$ : Probabilidad de transición de  $s_i$  a  $s_j$  con la acción  $a$ .
- $\pi : S \rightarrow A$ : Función que define las políticas de decisión.
- 
- **Transiciones** entre estados.
- **Recompensas** por transición. Pueden ser positivas o negativas.
- Factor de descuento  $\gamma \in [0, 1]$ : Importancia entre recompensas inmediatas o futuras (generalmente  $\gamma^t$ )

# MDP en nuestro ejemplo

---

El objetivo del superviviente es intentar maximizar la suma de las recompensas futuras tomando la mejor acción para cada estado:

$$\sum_{t=0}^{\infty} r_{e_t, a_t} \cdot \gamma^t$$

Explicado:

1. Estamos sumando para cada paso de tiempo  $t$ , de ahí el sumatorio.
2. Cada paso de tiempo tiene una recompensa  $r_{e_t, a_t}$  asociada la acción tomada.
3.  $\gamma^t$  es el factor de descuento en 1 por ahora y olvidémonos de ello.

Una vez formalizado el problema, vamos a explorar algunas soluciones.

# Solución #1: Q-learning

---

Se apoya en una función denominada acción-valor (*action-value*) o función  $Q$ :

- Entrada: Estado y acción a realizar.
- Salida: Recompensa esperada de esa acción (y de todas las posteriores).

La función  $Q$  se actualiza de forma iterativa:

1. Antes de explorar el entorno,  $Q$  da el mismo valor fijo (arbitrario).
2. Según se explora, aproxima mejor el valor de la acción  $a$  en un estado  $s$ .
3. Según se avanza, la función  $Q$  se actualiza.

Representa suma de las recompensas de elegir la acción  $Q$  y todas las acciones óptimas posteriores.

$$Q(e_t, a_t) = Q(e_t, a_t) + \alpha \cdot (r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Realizar  $a_t$  en el estado  $e_t$  actualiza su valor con un término que contiene:

- $\alpha$ : Lo "agresivo" que estamos haciendo el entrenamiento.
- $r_t$ : Estimación que obtuvimos al actuar en el estado  $e_t$  anteriormente.
- $\max_a Q(s_{t+1}, a)$ : Recompensa futura estimada.
- Se resta además el valor antiguo para incrementar o disminuir la diferencia en la estimación.

Ahora tenemos una estimación de valor para cada par estado-acción.

- Con el podemos elegir la acción que nos interesa (e.g. usando epsilon-greedy)

## Solución #2: *Policy learning*

---

Trata de determinar una función  $\pi$  asigna la mejor acción a un estado dado:

$$a = \pi(e)$$

*"Cuando observo el estado  $e$ , lo mejor que puedo hacer es tomar la acción  $a$ "*

Esta función es una función compleja que tratamos de aproximar.

- Y lo más "sencillo" y rápido es usar redes neuronales para ello.

# Otras soluciones

---

## *Deep Q-networks* (DQN)

Son aproximaciones de funciones  $Q$  utilizando redes neuronales profundas<sup>2</sup>.

## **Asynchronous Advantage Actor-Critic (A3C)**

Es una combinación de las dos técnicas anteriores<sup>3</sup>, combinando:

- Un actor: Red de políticas de actuación que deciden qué acción tomar.
- Un crítico: DQN que decide el valor de cada acción a tomar.

---

<sup>2</sup> <https://www.nature.com/articles/nature14236>

<sup>3</sup> <https://proceedings.mlr.press/v48/mniha16.html>



▶ 0:00 / 1:02



# Relevancia del aprendizaje por refuerzo

"El Go es un juego estudiado por los humanos durante más de 2500 años. AlphaZero, en un tiempo insignificante (3 días), pasó de conocer sólo las reglas del juego a vencer a los mejores jugadores del mundo, superando todo nuestro conocimiento acumulado durante milenios. Ningún campo del aprendizaje automático ha permitido avanzar tanto en este tipo de problemas como el aprendizaje por refuerzo."

# Relevancia del aprendizaje por refuerzo hoy en día

---

Podemos decir que es prácticamente el único paradigma de aprendizaje:

- Capaz de aprender comportamientos complejos en entornos complejos.
- Que ha podido hacerlo prácticamente sin supervisión humana.

Ofrece a la robótica forma abordar cómo diseñar comportamientos difíciles.

- Que por otro lado, son prácticamente todos.
- Las cosas fáciles para un humano suelen ser las más complejas de diseñar.

Permite a robots descubrir de forma autónoma comportamientos óptimos:

- No se detalla la solución al problema, sino que se interacciona con el entorno.
- La retroalimentación de el efecto sobre el entorno permite aprender.

# La utilidad de los modelos aproximados

---

Los datos del mundo real pueden usarse para aprender modelos aproximados.

- Mejor, porque el proceso de aprendizaje por ensayo y error es muy lento.
- Sobre todo en un sistema que tiene que hacerlo en un entorno físico.
- Las simulaciones suelen ser mucho más rápidas que el tiempo real.
- Y también también mucho más seguras para el robot y el entorno
- ***Mental rehearsal***: Describe el proceso de aprendizaje en simulación.

Suele ocurrir que un modelo aprende en simulación pero falla en la realidad:

- Esto se conoce como **sesgo de simulación**.
- Es análogo al sobreajuste en el aprendizaje supervisado.
- Se ha demostrado que puede abordarse introduciendo modelos estocásticos.

# Impacto del uso de conocimiento o información previa

---

El conocimiento previo puede ayudar a guiar el proceso de aprendizaje:

- Este enfoque reduce significativamente el espacio de búsqueda.
- Esto produce una **aceleración** dramática **en el proceso de aprendizaje** .
- También **reduce la posibilidad de encontrar mejores óptimos**<sup>1</sup>.

Existen dos técnicas principales para introducir conocimiento previo:

- A través de la **demostración** : Se da una política inicial semi-exitosa.
- A través de la **estructuración de la tarea** : Se da la tarea dividida.

---

<sup>1</sup> Alpha Go fue entrenado con un conocimiento previo de Go, pero Alpha Go Zero no sabía nada del juego. El resultado fue que Alpha Go Zero jugó y ganó a Alpha Go en 100 partidas.

# Desafíos del aprendizaje por refuerzo

---

**La maldición de la dimensionalidad** : El espacio de búsqueda crece exponencialmente con el número de estados.

**La maldición del mundo real** : El mundo real es muy complejo y no se puede simular.

- Desgaste, estocasticidad, cambios de dinámica, intensidad de la luz, ...

**La maldición de la incertidumbre del modelo** : El modelo no es perfecto y no se puede simular.

- Cada pequeño error se acumula, haciendo que conseguir un modelo suficientemente preciso del robot y su entorno sea un reto

La pregunta central de la filosofía moral es: ¿qué debemos hacer?

- ¿Cómo debemos vivir? ¿Qué acciones son correctas o incorrectas?
- Una posible respuesta es que, claramente, depende de los valores de cada uno.

A medida que vayamos creando una IA cada vez más avanzada, ésta empezará a salir de los problemas donde la recompensa se define mediante un número de puntos ganados en el juego, y requerirá recompensas más complejas.

- Los vehículos autónomos, por ejemplo, son agentes que tienen que tomar decisiones con una definición de recompensa algo más compleja
- Al principio, la recompensa podría estar ligada a algo como "llegar a salvo al destino".
- Pero ¿y si se ve obligado a elegir entre mantener el rumbo y atropellar a cinco peatones o desviarse y atropellar a uno? ¿debe desviarse o incluso dañar al conductor con una maniobra peligrosa? ¿Y si el único peatón es un niño, o un anciano, o el próximo Einstein o Hitler? ¿Cambia eso la decisión? ¿por qué? ¿Y si al dar un volantazo también se destruimos una escultura extremadamente valiosa e irremplazable?
- De repente tenemos un problema mucho más complejo cuando intentamos definir la función objetivo, y las respuestas no son tan sencillas.



**¡GRACIAS!**