

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

"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 -

Tipos de aprendizaje en máquinas

Existen tres tipos principales 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.

Aprendizaje por refuerzo (RL)

Subcampo 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:

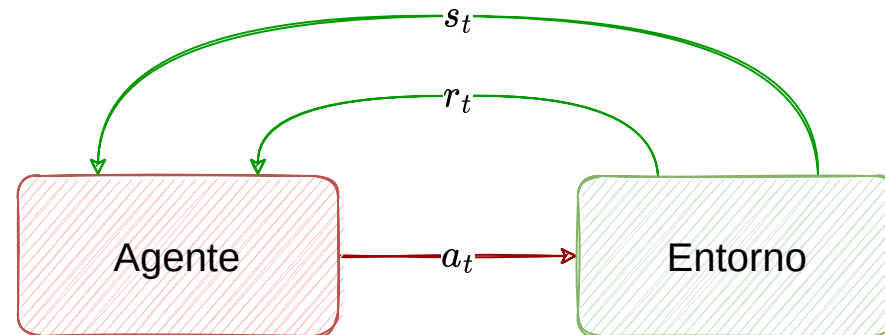
- **Juegos**: Los agentes aprenden las reglas y las jugadas jugando.
- **Control**: Los agentes aprenden en entornos de simulación las mejores políticas de control para un problema determinado.

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.

Modelo de interacción agente-entorno

Comportamiento : Sucesión de estados, acciones y recompensas asociadas:

$$(s_1, a_1, r_2), (s_2, a_2, r_3), \dots, (s_i, a_i, r_{i+1}), \dots$$



Definimos:

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

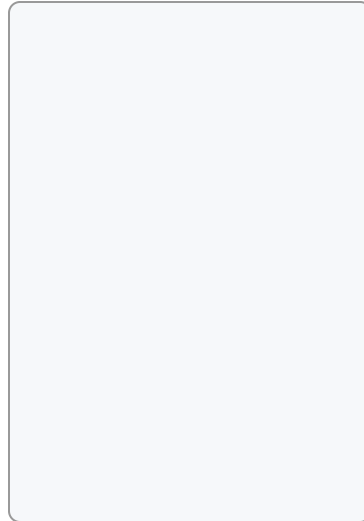
Proceso de aprendizaje por refuerzo

El proceso implica los siguientes pasos:

1. Observar el entorno en el que se encuentra el agente.
2. Decidir qué acción tomar usando alguna estrategia de toma de decisiones.
3. Ejecutar dicha acción.
4. Recibir una recompensa o castigo en función de la acción tomada.
5. Aprender de la experiencia y perfeccionar la estrategia de toma de decisiones.
6. Iterar todo el proceso hasta encontrar una estrategia óptima.

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).

Los conceptos cadena de Márkov y proceso Markov se usan indistintamente cuando el espacio de estados del proceso este es discreto.

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

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. γ^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:

- α : 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)

Estrategia epsilon-greedy

Estrategia muy sencilla para evitar mínimos locales:

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

Por lo general, se empieza con ϵ alto (mucha exploración).

- Según el superviviente aprende más sobre la ciudad, ϵ disminuye.
- Ha explorado mucho, así que puede centrarse en explotar lo conocido.

Solución #2: *Policy learning*

Trata de determinar una función π 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 profundas².

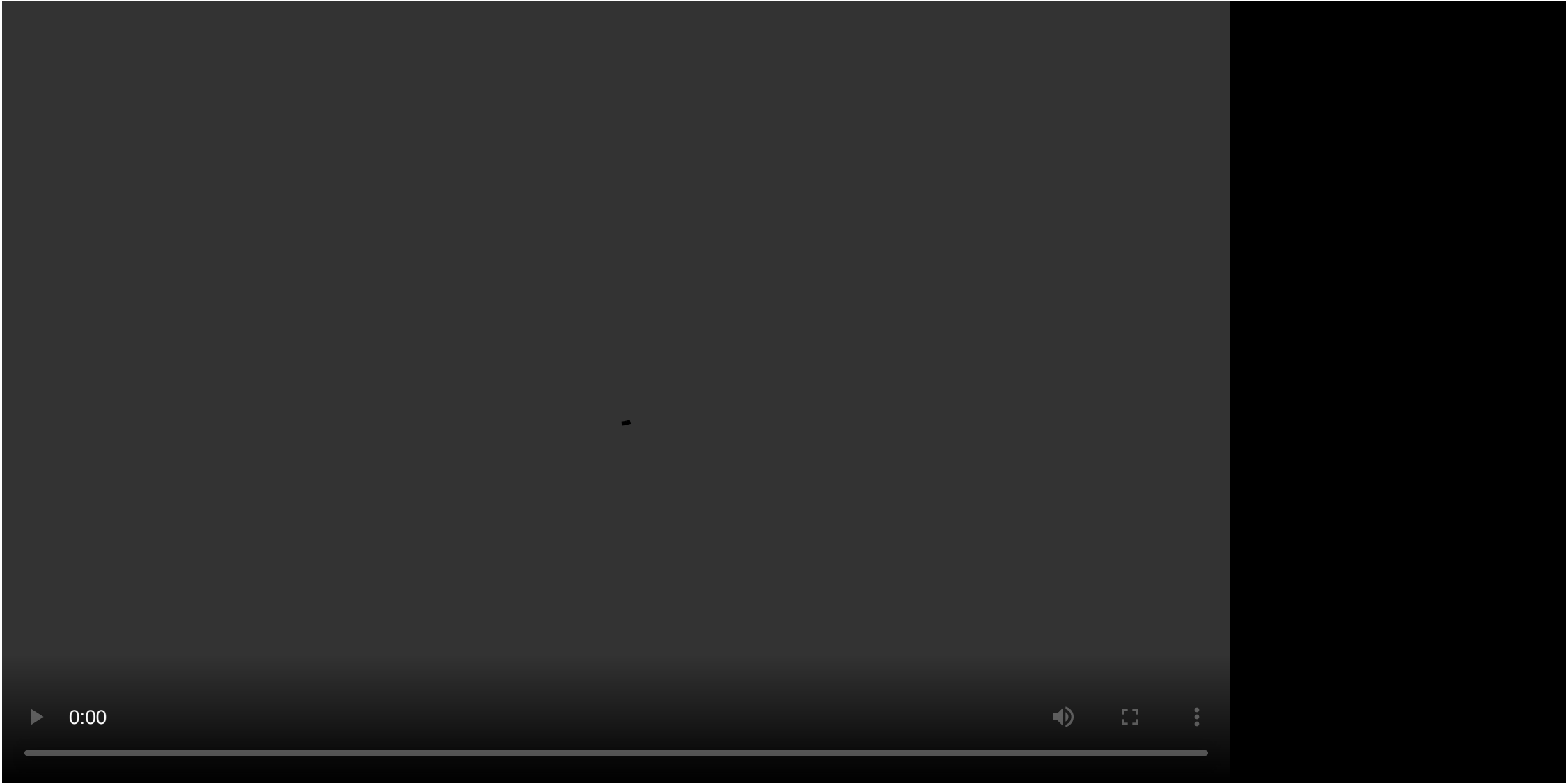
Asynchronous Advantage Actor-Critic (A3C)

Es una combinación de las dos técnicas anteriores³, 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.

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

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



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**¹.

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.

¹ 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

- 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?

¡GRACIAS!