

08MIAR - Aprendizaje por refuerzo

Sesión 1 – Introducción

The logo consists of a solid orange circle containing the lowercase letters "viu" in a white, sans-serif font.

viu

Universidad
Internacional
de Valencia

De:

 Planeta Formación y Universidades

Sobre mí

Gabriel Muñoz

Lead Data Scientist en Intelygenz (<https://intelygenz.com/>)

Co-fundador de *MAD_RL*, comunidad de aprendizaje por refuerzo con base en Madrid

gabrielenrique.munoz@campusviu.es
<https://es.linkedin.com/in/gabrielmunozrios>

Sobre la asignatura

El contenido de la asignatura se organizará en tres bloques:

- En el primer bloque introduciremos el contexto en el que se desarrollan las soluciones de aprendizaje por refuerzo, así como los conceptos y términos más importantes.
- El segundo bloque estará compuesto del estudio de los algoritmos que forman el estado del arte actual. Principalmente, cuáles son las estrategias de aprendizaje y cómo se produce el *sampling* de datos durante el mismo.
- El tercer bloque cubrirá una serie de sesiones prácticas donde implementaremos soluciones de aprendizaje por refuerzo usando diferentes *frameworks* y librerías.

Para más información sobre las sesiones: **Anexo de organización de sesiones en la sección de guía didáctica de la asignatura.**

Sobre la asignatura

Respecto a los requisitos de la asignatura:

- Los conceptos teóricos que veremos se basarán en conceptos matemáticos y estadísticos. Hablaremos de distribuciones de probabilidad, de búsqueda óptima, de cadenas de *Markov*, etc. El conocer estos conceptos no es una restricción para el curso, pero si que es recomendable.
- En la parte práctica trabajaremos con Python y algunas librerías típicas de entornos para trabajar con datos, como por ejemplo *Keras*, *Tensorflow* y *Pytorch*. Habrá disponible una sesión práctica grabada enfocada en la puesta a punto del entorno y la presentación de los frameworks que utilizaremos.
- En ambas partes será imprescindible conocer y tener cierta experiencia con modelos *Deep Learning* y, en concreto, con redes convolucionales.

Sobre la asignatura

El método de evaluación de la asignatura estará compuesto de:

- Participación en foros de debate (10%)
- Actividad sobre artículo científico (10%)
- Actividad sobre una serie de vídeos teóricos (10%)
- Proyecto práctico (30%)
- Examen final (40%)

Índice

¿Qué es aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

Retos actuales y de futuro

Vista general

Conclusiones

Bibliografía recomendada

¿Qué es aprendizaje por refuerzo?

“Aprender a partir de la interacción es una idea fundamental y común a todas las teorías del aprendizaje y de la inteligencia”

“Aprendizaje por refuerzo [...] es a la vez un problema, un conjunto de soluciones que funcionan bien sobre un conjunto de problemas y el campo que estudia esos problemas y los métodos que pueden solucionarlo.”

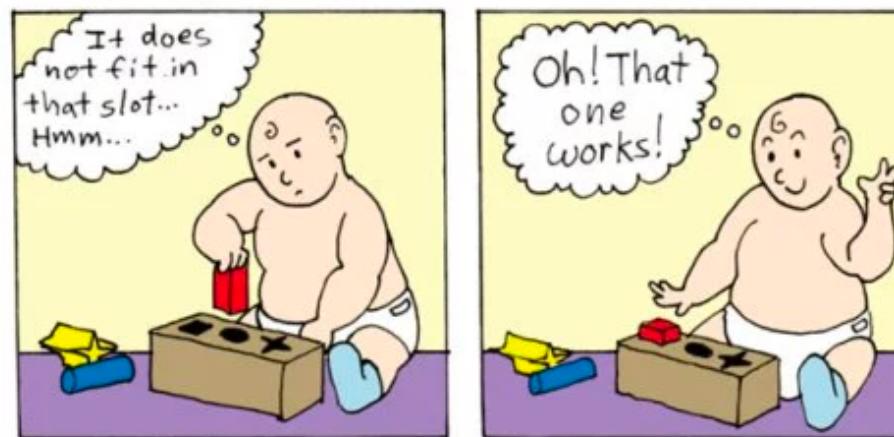
Reinforcement learning: An Introduction, R. Sutton & A. Barto

“El objetivo del aprendizaje por refuerzo es entrenar un agente inteligente que es capaz de interactuar con un entorno de manera inteligente.”

Deep Q Network vs Policy gradients, Felix Yu

¿Qué es aprendizaje por refuerzo?

Podemos ver el aprendizaje por refuerzo como el estudio y diseño de **agentes** que aprenden por medio de **prueba-y-error**.



<https://rochemamabolo.files.wordpress.com/2018/07/trial02.jpg?w=490>

Sesión 1 - Introducción

¿Qué es aprendizaje por refuerzo?

Si nos paramos y analizamos otras ramas dentro del aprendizaje basado en datos, encontramos dos grandes conjuntos: métodos supervisados y métodos no supervisados. Podríamos añadir aprendizaje por refuerzo como un tercer conjunto dentro de esta clasificación.

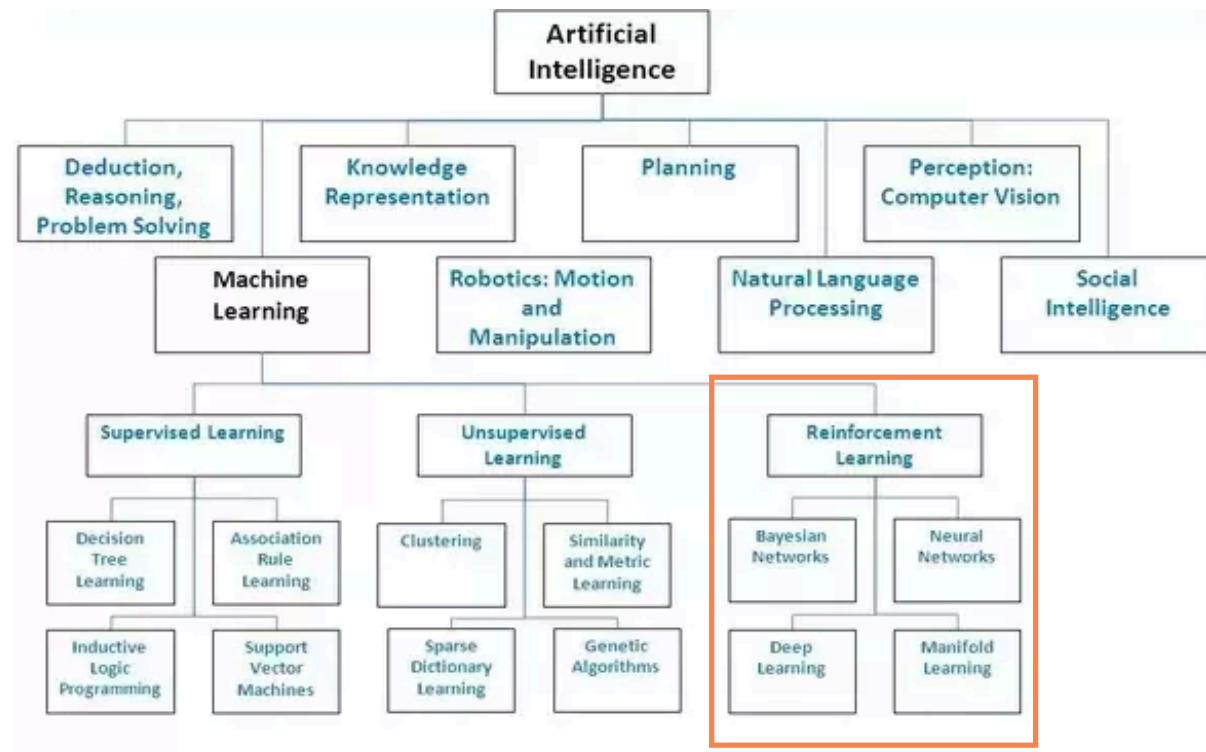
Desde un punto de vista de datos, podemos definir la relación de cada conjunto con el conocimiento que se puede obtener a partir de los datos:

Análisis descriptivo → Métodos no supervisados

Análisis predictivo → Métodos supervisados

Análisis prescriptivo → Métodos aprendizaje por refuerzo

¿Qué es aprendizaje por refuerzo?



<https://qph.fs.quoracdn.net/main-qimg-541146e2d9d611a42ebb5074aa72fef4.webp>

Sesión 1 - Introducción

¿Qué es aprendizaje por refuerzo?

En mi opinión, el aprendizaje por refuerzo desarrolla de una manera natural e intuitiva lo que conocemos (o interpretamos) como aprendizaje.

Al producirse el aprendizaje de una manera similar a la de los humanos, se podría empezar a hablar y discutir en términos de adaptación y/o transferencia de conocimiento entre distintos problemas.

Precisamente la transferencia de conocimiento entre distintos problemas (***transfer learning***) es una rama de investigación con mucha fuerza hoy en día dentro de la inteligencia artificial y, en concreto, del ***Deep Learning***.

Otro punto muy interesante es el concepto de bias o sesgo, que tanto impacto tiene en los datos con los que trabajamos. Dentro de una solución guiada por aprendizaje por refuerzo podríamos mitigar mucho este problema, ya que el aprendizaje se produce *from scratch*.

Estado del arte

Como en otras ramas de la inteligencia artificial, los primeros algoritmos y soluciones basadas en aprendizaje por refuerzo datan de hace algunas décadas. En nuestro caso, fue en los años 80/ principios de los 90 cuando hubo una tendencia positiva en el estudio y uso de algoritmos basados en aprendizaje por refuerzo.

Aunque los resultados fueron prometedores, debido a las capacidades computacionales y al desarrollo de otros algoritmos con enfoques similares en aquel momento (**algoritmos evolutivos**), las soluciones basadas en aprendizaje por refuerzo disminuyeron hasta hace unos años.

Como apunte, la principal diferencia entre algoritmos de aprendizaje por refuerzo y algoritmos evolutivos es que los agentes de algoritmos evolutivos no aprenden de la interacción con su entorno. Las reglas de su interacción son inamovibles, por lo que no pueden aprovechar todo la información de la que disponen para aprender y adaptarse al problema que intentan resolver.

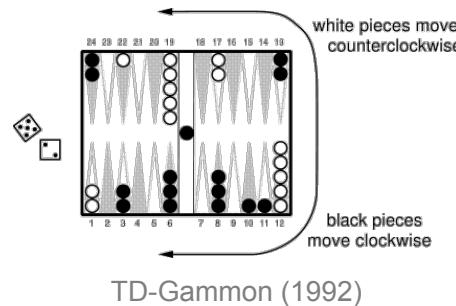
Estado del arte

Como decíamos, hasta hace algunos años no hubo una explosión en este tipo de algoritmos y del potencial que tienen. Una relación directa ha sido el magnífico desarrollo de las técnicas de *Deep Learning* así como la capacidad computacional que ahora tenemos a nuestra disposición.

Igualmente, si tuviéramos que elegir tres hitos que sentaron las bases para hablar de aprendizaje por refuerzo hoy serían:



Minsky PHD thesis (1954)

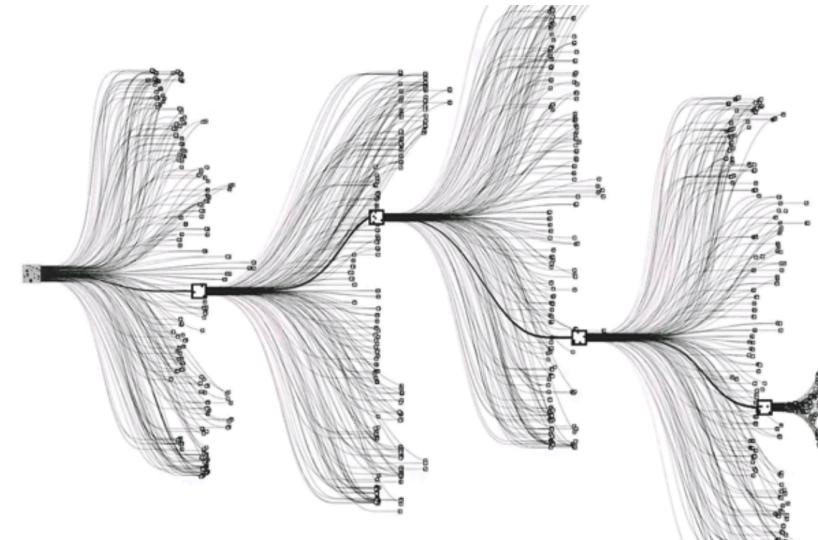


TD-Gammon (1992)



DeepMind (2015)

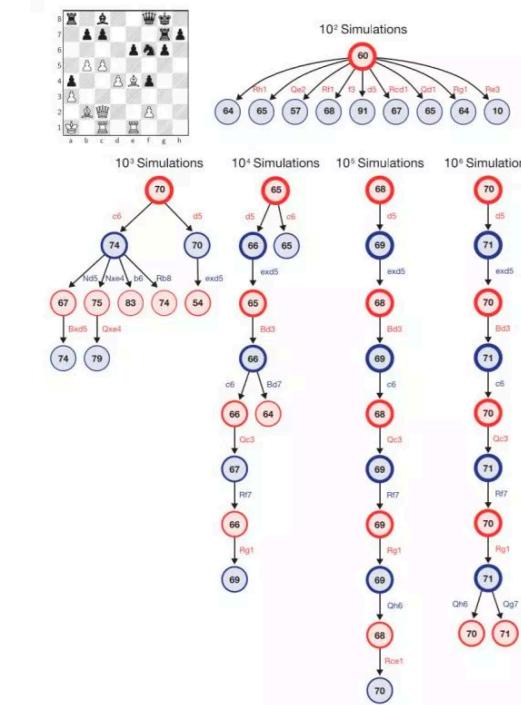
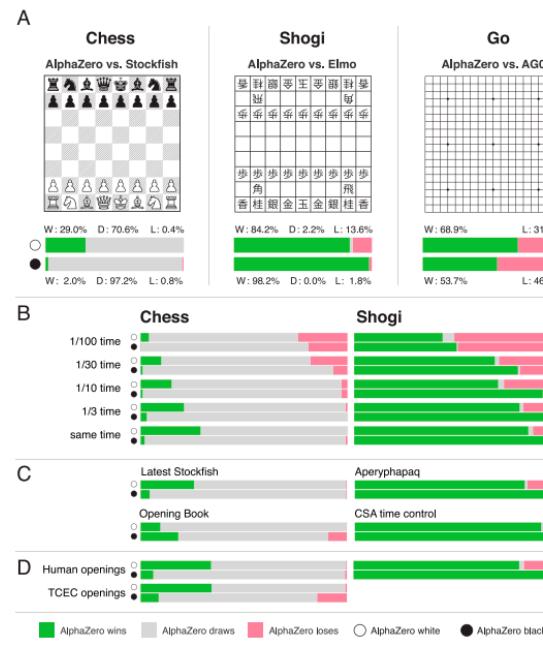
Estado del arte



Hitos de DeepMind: **AlphaGo**

https://obs-educom.cdnstatics.com/sites/default/files/styles/blog_post/public/post/blog_obs_tendencias_e_innovacion_roman-febrero_1_foto3.jpg?itok=caNOUFa-
https://cdn-images-1.medium.com/max/2600/1*qpzAxoUR9POLY1_zJhU5g.png

Estado del arte



Hitos de DeepMind: **AlphaZero**

<https://cdn.chess24.com/f1DREmrdrFOR-HKtB3JRcA/original/chess-performance.png>
<https://vonneumannmachine.files.wordpress.com/2018/12/captura.jpg?w=614&h=893>

Estado del arte

Otra compañía que ha sido un pulmón en el desarrollo del aprendizaje por refuerzo estos últimos años ha sido OpenAI.

La misión de OpenAI es la creación de una inteligencia artificial de carácter general con una fuerte base ética y de principios.

OpenAI han sido los responsables de algunos de los algoritmos que ahora mismo son estado del arte, así como de ser la primera compañía en ganar con inteligencia artificial en entornos de colaboración (y muy avanzados en cuanto a complejidad). Un ejemplo es el videojuego *DOTA2*.



<https://venturebeat.com/wp-content/uploads/2019/03/openai-1.png?fit=2400%2C1000&strip=all>

Sesión 1 - Introducción

Estado del arte

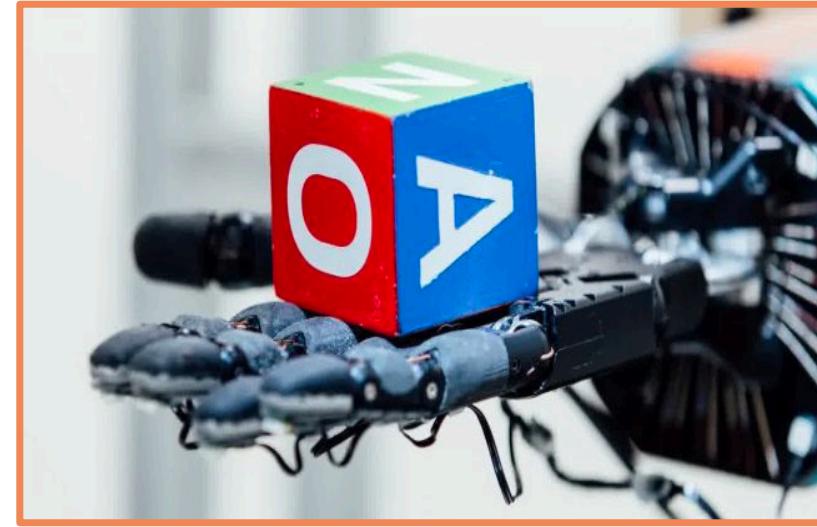
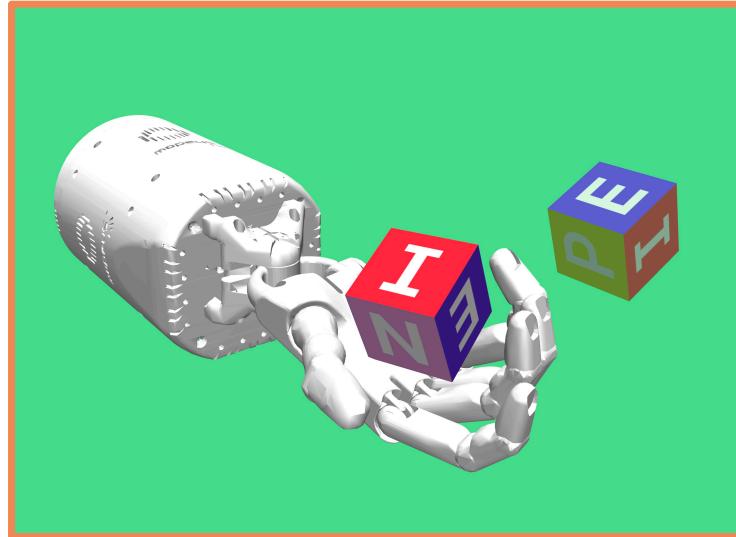


Hitos de OpenAI: Dota2

<https://cdn.estnn.com/wp-content/uploads/2019/04/22152826/Dota-2-Open-AI-2.png.jpg>

Sesión 1 - Introducción

Estado del arte



Hitos de OpenAI: **Robótica**

<https://spectrum.ieee.org/image/MzAyMzczMg.jpeg>
<https://www.extremetech.com/wp-content/uploads/2018/07/openai-640x353.jpg>

Sesión 1 - Introducción

Índice

¿Qué es aprendizaje por refuerzo?

Estado del arte

Distintos enfoques de aprendizaje por refuerzo

Retos actuales y de futuro

Vista general

Conclusiones

Bibliografía recomendada

Distintos enfoques del aprendizaje por refuerzo

Una posible clasificación que podemos hacer cuando hablamos de aprendizaje por refuerzo está relacionado con el entorno donde se ejecutará la solución desarrollada.

En este sentido tenemos dos grupos principales:

Entorno puramente digital - Ámbito de investigación

Entorno Digital-a-Real - Ámbito de investigación/aplicación

Vamos a entrar en detalle con cada enfoque.

Distintos enfoques del aprendizaje por refuerzo



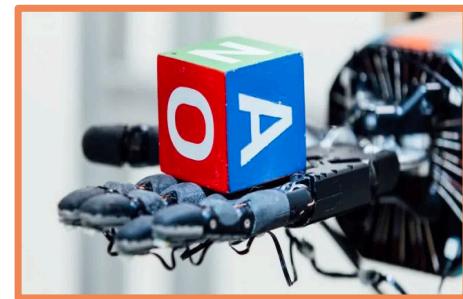
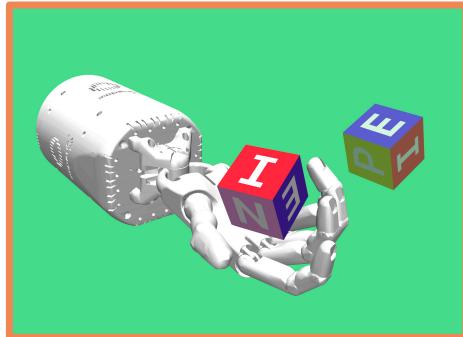
Entorno puramente digital - Ámbito de investigación

Este es el entorno típico que nos encontraremos en la mayoría de ejemplos a día de hoy.

Trabajaremos en un entorno totalmente simulado y controlado para poder probar nuevos algoritmos, nuevos modelos, etc.

Este enfoque es fundamental para contrastar hipótesis teóricas y comprobar su posible aplicación en el mundo real.

Distintos enfoques del aprendizaje por refuerzo



Entorno Digital-a-Real - Ámbito de investigación

En este caso tenemos una situación parecida al enfoque anterior aunque ahora usaremos el conocimiento extraído en un entorno real.

Un caso común es la robótica, en el que se diseña toda la simulación en un entorno digital para usar la solución obtenida en brazos robóticos reales.

Retos actuales y de futuro

El principal reto en el que el aprendizaje por refuerzo se centra actualmente es encontrar **casos de uso** en los que se pueda aplicar de una manera beneficiosa.

Es verdad que en los últimos años se han ido desarrollando soluciones con aplicaciones reales en ámbitos como la robótica, control automático, etc., pero todavía no tienen un impacto de negocio como para tomar a estas soluciones más en “serio”.

Para la mayoría de estos casos hay soluciones tradicionales que funcionan muy bien y no necesitan de la complejidad que necesita el aprendizaje por refuerzo.

Retos actuales y de futuro

Una de las complejidades que presentan las soluciones basadas en aprendizaje por refuerzo, y que veremos en las siguientes sesiones, es **la cantidad de hiperparámetros** que se necesitan configurar para que una solución funcione.

Y no sólo eso, ¿qué significa que una *solución funcione*?

Veremos en siguientes sesiones conceptos para poder discutir sobre esta pregunta, pero lo que está claro es que el concepto de que algo sea útil o no sigue estando muy sesgado a los ojos de quien lo mire (como es normal por otra parte).

Retos actuales y de futuro

Otro reto es que aún teniendo mucho dominio de la materia y conociendo muy bien todos los conceptos, el control sobre el proceso de aprendizaje es vago.

En otras disciplinas de la inteligencia artificial, como por ejemplo en la aplicación de técnicas de Deep Learning, tenemos una metodología que, aunque sea empírica, podemos controlar y medir cómo el sistema evoluciona y si lo está haciendo de forma favorable o no.

En el caso del aprendizaje por refuerzo no se dan las mismas circunstancias. Dependiendo del algoritmo, del reto y de la información disponible, la forma de medir la bondad de una solución difiere y por tanto le añade una complejidad extra a su diseño.

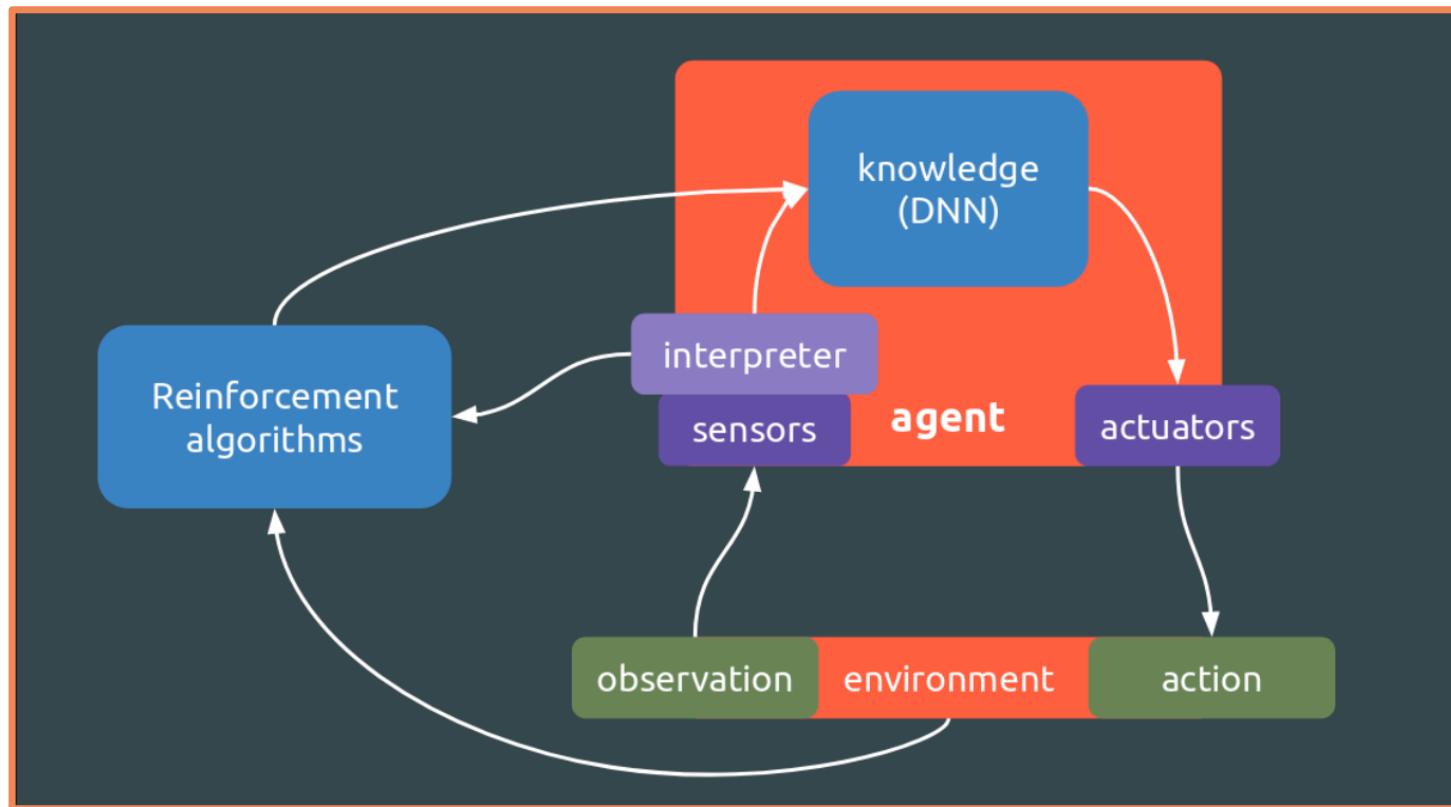
Retos actuales y de futuro

Aún así, todos los grupos de investigación comparten la idea de que el **potencial que tiene este enfoque de la inteligencia artificial** es *infinito*.

La mayoría de problemas que ahora mismo podemos solucionar son problemas más cercanos a la automatización que al razonamiento y adaptación que una inteligencia real permitiría.

Es aquí donde también se presenta un gran reto: ser capaz de encontrar la madurez suficiente para afrontar verdaderos problemas de Inteligencia Artificial siguiendo esta metodología.

Vista general



Conclusiones

- El aprendizaje por refuerzo es una de las ramas más prometedoras dentro de la inteligencia artificial.
- El proceso de aprendizaje se asemeja a la idea de aprendizaje que tenemos los seres humanos.
- Aunque es una rama muy prometedora, todavía están en sus primeros pasos. No está totalmente asentada y sus soluciones no se pueden aplicar en todos los campos y/o dominios.
- La unión de modelos de Deep Learning con algoritmos de aprendizaje por refuerzo ha sido la mezcla perfecta para abrir todo el conjunto de posibilidades con tanto potencial que tenemos a nuestro alcance actualmente.

Bibliografía recomendada

“Reinforcement Learning: An introduction”, Sutton y Barto:
<http://incompleteideas.net/book/bookdraft2017nov5.pdf>

Contenidos y documentación de OpenAI SpinningupRL:
<https://spinningup.openai.com/en/latest/>

08MIAR - Aprendizaje por refuerzo

Sesión 2 – Conceptos y terminología

Curso 21/22

The logo consists of the lowercase letters "viu" in white, sans-serif font, centered within a solid orange rounded rectangle.

viu

Universidad
Internacional
de Valencia

De:

 Planeta Formación y Universidades

Índice

Vista general

Conceptos básicos

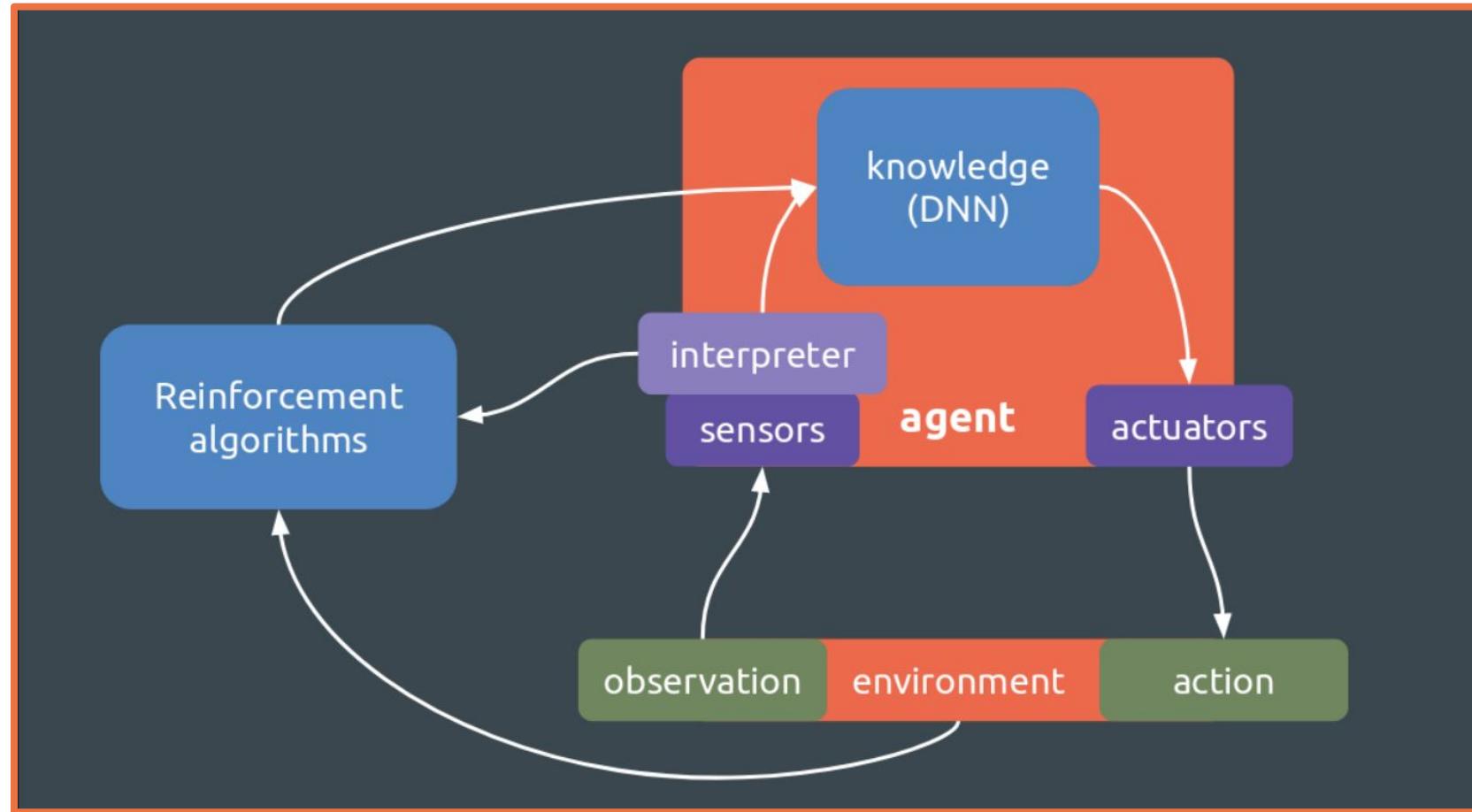
Conceptos avanzados

Clasificación de problemas de aprendizaje por refuerzo

Jerarquía de algoritmos

Conclusiones

Vista general



Conceptos básicos

Para entender bien los algoritmos que veremos y cómo podríamos aplicarlos a distintos problemas, necesitamos conocer previamente los conceptos y la terminología que usaremos durante el desarrollo.

En el caso del aprendizaje por refuerzo, la terminología es muy característica y sus conceptos muy cercanos a las matemáticas y estadística que subyace en este tipo de problemas.

En esta sesión entenderemos los conceptos de una manera intuitiva y a alto nivel, así como la representación de cada uno para explotar su uso más adelante.

En las sesiones sobre los algoritmos será donde entraremos más en detalle, desde un punto de vista formal y analítico, sobre los conceptos que veremos a continuación.

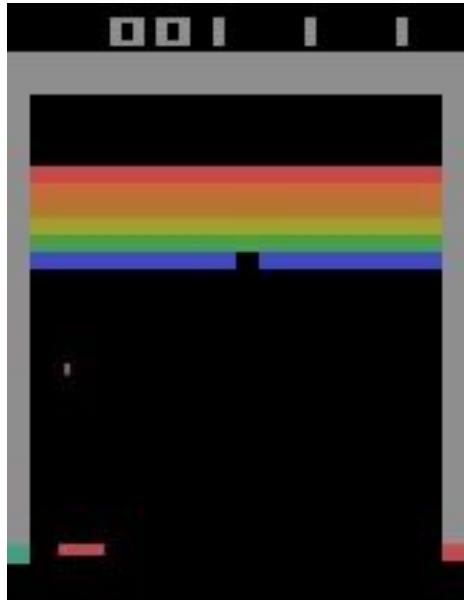
Conceptos básicos

En nuestro caso, los retos en los que trabajaremos estarán basados en videojuegos ya que casan muy bien con el poder aplicar y probar nuestros algoritmos sobre ellos. Además, los videojuegos están definidos por reglas que rigen cómo se pueden comportar nuestros agentes en los mismos.

Por otro lado, hay que tener en cuenta que para poder demostrar empíricamente que nuestras soluciones *funcionan*, necesitamos controlar en todo momento la simulación, de ahí que los videojuegos sean una buena elección.

Para entender más fácilmente cada uno de estos conceptos vamos a usar como base un entorno digital común para todos, el videojuego de ***breakout de Atari***.

Entorno



El entorno es el mundo donde el agente vive e interactúa con los elementos de su alrededor.

El entorno no sólo se compone de los elementos visuales de la simulación, también de la lógica que subyace y que define cómo se puntúa, tiempos disponibles, etc.

El entorno es uno de los actores principales en nuestros problemas ya que es la simulación sobre la que nuestro agente aprenderá.

Entorno

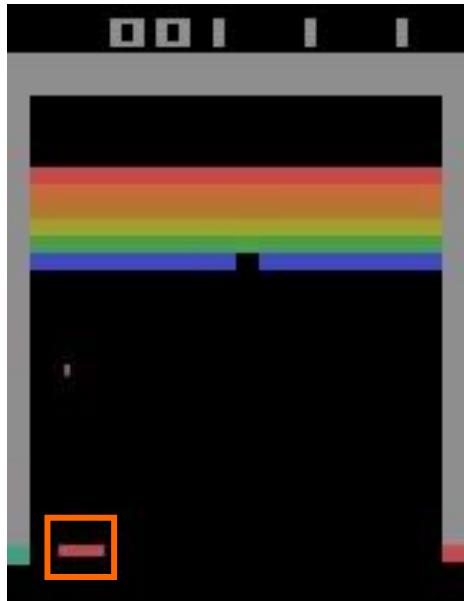
Todos los conceptos que abordaremos a alto nivel tienen una relación directa con una parte que también nos interesa, la analítica/tecnológica.

Iremos viendo cómo se representa cada uno de estos conceptos desde un punto de vista más cercano a (*pseudo*)código para poder trabajar con ellos.

En el caso del entorno, normalmente encontraremos un *wrapper* de la simulación. Al ejecutarlo, o al interactuar con el agente, nos irá devolviendo toda la información disponible.

```
# Python code  
  
env = wrapper.open("Atari.rom")  
  
env.action_space  
  
env.step(action)  
  
env.internal_variables
```

Agente



El agente es la otra pieza fundamental de nuestra solución.

Un agente es una entidad que interactúa con elementos del entorno (que estén disponibles en el momento de la interacción) para llegar a un objetivo.

Generalmente, este objetivo es maximizar una recompensa (como el en caso de la mayoría de juegos).

El agente será la entidad inteligente que irá aprendiendo durante las ejecuciones del experimento.

Agente

El agente será la pieza de nuestras soluciones en la que recaerá más responsabilidad por nuestra parte.

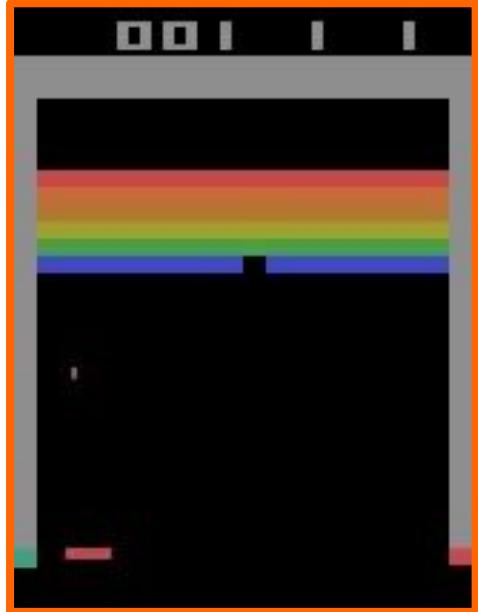
Comúnmente nos encontraremos una clase que sea la que englobe todas las funciones y comportamientos que el agente puede realizar durante su aprendizaje.

Será en el agente donde se relaciona la información que proviene del entorno con el módulo de aprendizaje que hayamos definido (en este curso, una red neuronal).

```
# Python code

class OurAgent:
    def __init__(...):
        ...
    def select_action(...):
        ...
    def preprocess_obs(...):
        ...
    def update_states(...):
        ...
```

Observación



Una vez tenemos el *feedback-loop* entre nuestro entorno y nuestro agente, ¿cómo se realiza el aprendizaje?

En primer lugar tendremos observaciones. Cada vez que el agente toma una decisión, esa decisión afecta al entorno. Cada *fotografía* del entorno tras una decisión del agente es lo que podemos entender como observación.

Cada observación contiene toda la información disponible en el entorno en ese momento.

Observación

Normalmente, sobre todo en el ámbito de los videojuegos, las observaciones van a tener la forma de un *array* o lista de elementos.

En nuestro reto, cada uno de estos elementos será un pixel de la pantalla.

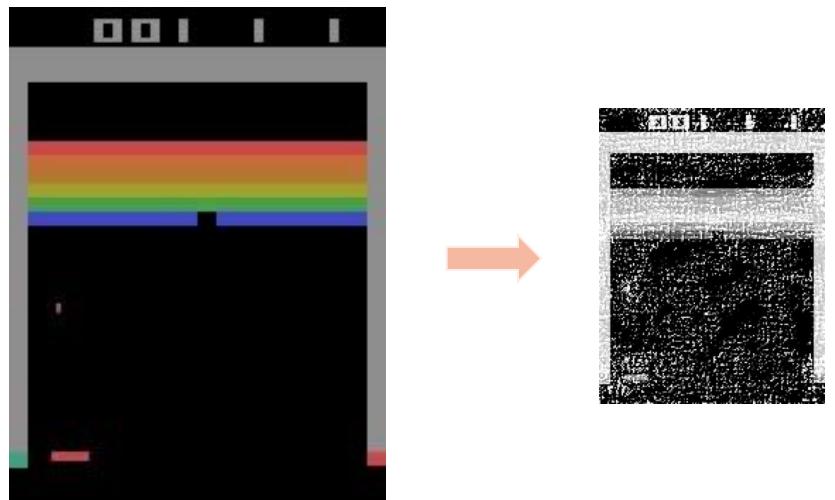
Dependiendo de la información disponible, podemos encontrar información extra en estructuras de datos o en otros campos que el entorno nos devuelve.

```
# Python code

obs, info, reward, done = env.step(...)

# obs
[[[255, 255, 255, 255],
  [120, 123, 122, 123],
  ...
  [110, 110, 115, 115],
  [255, 255, 255, 255]]]
```

Estado



Cuando el entorno nos devuelve la observación en un momento determinado de la ejecución, la información que el agente obtiene es *raw*, es decir, es tal y como el entorno la genera.

Dependiendo de cómo se defina el módulo de aprendizaje del agente, y de la disponibilidad computacional que tengamos, esta observación necesita un tratamiento.

Esto es lo que se conoce como estado. El estado es la observación lista para ser consumida por nuestro agente y poder tomar decisiones.

Otra posible interpretación del concepto de estado es la relación directa con la definición formal mediante cadenas de Markov.

Estado

En el caso del estado, partimos de la información que tenemos a partir de la observación.

A esta observación le aplicamos todo lo necesario en cuanto a pre-procesamiento, redimensionado de los datos, etc.

Una vez aplicado este pre-procesamiento, la nueva estructura de datos estaría lista para que el agente la pudiera usar adecuadamente.

```
# Python code
state = resize(obs)
state = preprocess_and_rgb(State)

# state
[[[1.0, 1.0, 1.0, 1.0],
  [0.5, 0.5, 0.5, 0.5],
  ...
  [0.3, 0.4, 0.3, 0.3],
  [1.0, 1.0, 1.0, 1.0]]]
```

Acciones



https://images.lukiegames.com/t_300e2/assets/images/atari/at_atari_2600_joystick.jpg

Sesión 2 – Conceptos y terminología

Toda decisión que hemos ido comentando que el agente va a realizar en el entorno es lo que se conoce como acción.

El agente siempre tiene a su disposición una lista de acciones disponibles, que puede ir cambiando conforme la simulación vaya evolucionando.

Tanto los estados como las acciones son conceptos que abren sus propias vías de investigación debido a problemas subyacentes como búsqueda óptima, combinatoria, etc.

Acciones

Para nosotros, el listado de acciones será una estructura de datos como un array de la forma *one-hot-encoding*.

Tendremos una lista con todas las acciones disponibles en cada posición y pondremos a 1 la acción elegida en ese momento de la ejecución, dejando a 0 el resto.

Veremos que dependiendo del algoritmo, la evaluación del agente para elegir una acción difiere: selección por estimación de bondad de la acción, selección acorde a una distribución de probabilidad, etc.

```
# Python code
action = agent.select_action(state)

# action
[0, 0, 0, 1] # Move left

env.step(action)
```

Espacio de estados y acciones

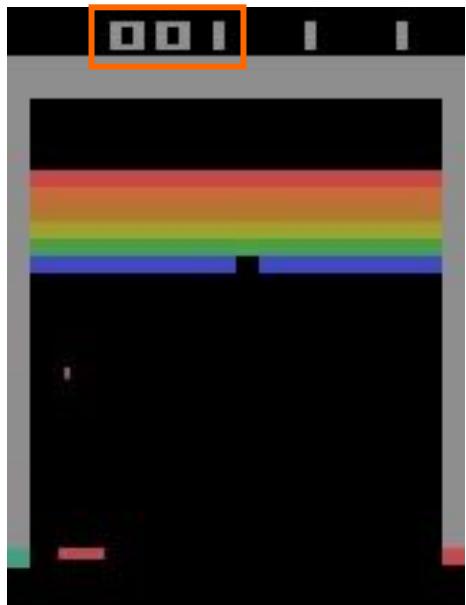
Tanto las acciones como los estados tienen características matemáticas muy interesantes. Al conjunto de todos los estados y acciones disponibles se le conoce como espacio.

Son estos espacios los que producen que el aprendizaje por refuerzo tenga un fuerte componente de optimización y que, dependiendo del problema, una búsqueda óptima en estos espacios es la parte crucial de nuestras soluciones.

Matemáticamente, el espacio de estados se puede ver como una cadena de *Markov* (MDP), en el que cada estado va cambiando en el tiempo.

En el caso de las acciones, la búsqueda óptima está más relacionada con técnicas meta-heurísticas, *back-tracking* o búsqueda en árbol. Estas decisiones también son muy dependientes del problema que estemos modelando.

Recompensa



Una vez que el agente toma una acción, ¿cómo sabe si la acción que ha tomado es buena o mala?

Para guiar el aprendizaje y analizar qué acciones son buenas o malas usamos la recompensa. La recompensa es el *feedback* que nos devuelve el entorno para evaluar cómo lo está haciendo el agente.

Por ejemplo, si jugamos a algún videojuego, la recompensa sería la puntuación del mismo. Pero también tenemos otras alternativas como: medir la barra de vida, cuántos *continues* nos quedan, velocidad a la que jugamos, etc.

Recompensa

La recompensa es un valor que nos devuelve el entorno. Dependiendo del entorno en el que estemos trabajando, este valor puede variar aunque siempre intenta ser fiel al comportamiento básico de la simulación.

El punto de complejidad respecto a la recompensa se produce cuando definimos nuestras propias funciones. Por ejemplo, dependiendo del experimento nos puede interesar potenciar ciertas acciones o incluso usar también un castigo para puntuar situaciones adversas.

```
# Python code

obs, info, reward, done =
env.step(...)

# reward
0.8

# Define a reward taking into
account
# the game counter
def my_reward(...):
    ...
```

Iteración (Step)

Cuando decíamos que la observación era una *fotografía* del entorno en un momento determinado, esa definición de momento es lo que se conoce como iteración o *step*.

La iteración se produce cuando congelamos todos los componentes de nuestra ejecución en un instante de tiempo.

Este componente también es muy útil para estimar la ejecución y bondad de ciertos algoritmos ya que será el valor con el que mediremos tiempos y velocidad de convergencia.

Episodio

Por último, tenemos el concepto de episodio. El episodio se desarrolla desde el comienzo de una ejecución hasta que llegamos a *game over*.

Dependiendo del entorno podemos encontrarnos con que el episodio se corresponda con una fase o reto, o que se desarrolle en el tiempo hasta que termine de alguna manera.

El episodio está compuesto de un conjunto de iteraciones.

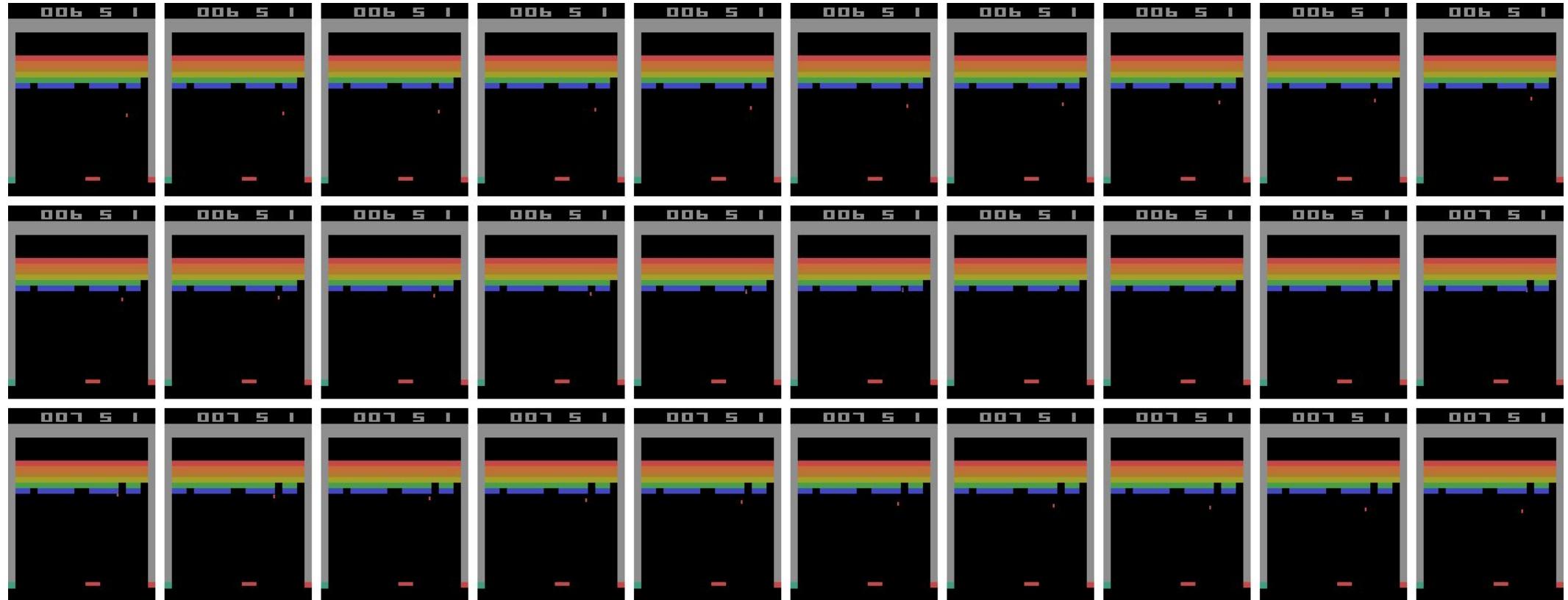
Episodio e iteración

Si queremos interpretar las iteraciones en nuestro código, nos encontramos con una variable para controlar el número de ejecuciones de nuestra simulación.

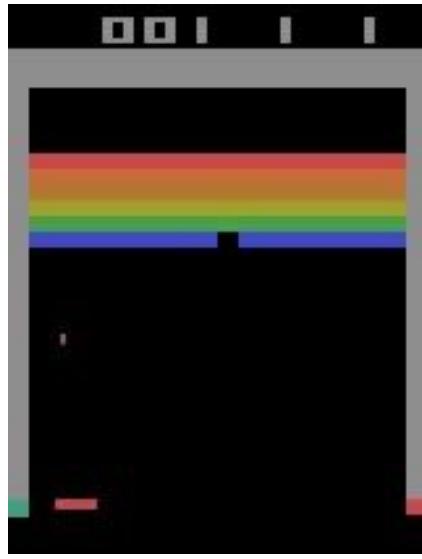
Los episodios están un nivel por encima para definir el número de pasadas globales (o partidas) que nuestro agente va a realizar.

Ambos conceptos los podemos ver como dos bucles anidados para controlar desde lo más atómico hasta el nivel superior. Normalmente, el bucle que controlamos es el de los episodios, siendo el bucle de las iteraciones interno en la propia simulación.

Episodio e iteración



Estrategia (Policy)



← → ?

<https://gym.openai.com/videos/2019-04-06--My9IiAbqha/Breakout-v0/poster.jpg>

Sesión 2 – Conceptos y terminología

Cuando hablamos de que el agente aprende sobre un entorno, en realidad nos referimos a que el agente encuentra una estrategia para maximizar un objetivo.

Esta estrategia será la que nos diga qué acción tomar en un estado. Nuestra meta es que nuestro agente aprenda la estrategia óptima.

La búsqueda de la mejor estrategia posible es la base del aprendizaje por refuerzo.

En inglés se usa el término *policy* por lo que iremos intercambiando ambos términos durante nuestro curso.

Estrategia (Policy)

Desde un punto de vista programático, la estrategia es el concepto ligado a nuestro modelo de Deep Learning.

Va a ser nuestro modelo el que se encargue de modelar las acciones que se toman dependiendo de los estados en los que el agente se encuentra.

Dependiendo del algoritmo podemos tener distintos comportamientos en la estrategia.

```
# Python code
action = agent_model(state)

# Q models
action_selected = np.argmax(action)

# PG models
action_selected =
    distribution(1, prob=action)
```

Transiciones

En cada paso de un estado a otro se produce lo que llamamos transición. Las transiciones están compuestas de una tupla de elementos:

[Estado, Siguiente estado, recompensa, acción]

Con esta tupla mínima podemos modelar el paso de un estado a otro con información suficiente como para que el agente vaya aprendiendo. Dependiendo del algoritmo usado, esta tupla puede contener otros elementos que añadan información útil al agente.

Transiciones

Lo más importante para las transiciones es seleccionar una estructura de datos adecuada.

Podemos seleccionar desde la más simple, una lista de elementos, a usar colas de mensajería que se encarguen de procesar y gestionar todas las transiciones.

Lo más importante es mantener la misma estructura durante todas las ejecuciones, para que no afecte desde un punto de vista computacional.

```
# Python code
transition = (state, next_state,
               reward, action)

# Lists
transitions.append(transition)

# Queue
transitions = Queue()
transitions.push(transition)
```

Experiencia

Dependiendo del algoritmo que usemos nos interesa extraer un tipo de información u otra.

Esto es lo que se conoce como experiencia, porque va directamente relacionado con la información pasada que el agente tiene a su disposición para ir aprendiendo.

En su forma más básica se almacena la información de las transiciones, pero dependiendo del algoritmo esa información puede variar.

Experiencia

Al igual que con las transiciones, la experiencia también necesita de una estructura de datos adecuada.

En este caso, es común usar colas de un tamaño fijo ya que, como veremos en los algoritmos, la experiencia está relacionada con un conjunto fijo de número de transiciones para ir entrenando a nuestro agente.

```
# Python code

class Experience():
    def __init__():
        # Queue with 120 elements
        self.data = Queue(120)

    def add(elem):
        self.data.push(elem)

    def get():
        self.data.pop()

transition = (state, next_state,
              reward, action)
experience = Experience()
experience.data.add(transition)
```

Exploración



EXPLORATION

Playing the other machines to see if any pay out more.

Proceso por el que el agente va adquiriendo experiencia a partir de prueba-y-error en el entorno.

Normalmente se ejecuta al comienzo de una simulación, para empezar el almacenamiento de las transiciones y así comenzar el aprendizaje.

En su forma más básica, la exploración está controlada por una variable aleatoria que va disminuyendo en el tiempo. Dependiendo del valor de esta variable vamos decidiendo si la acción que toma el agente es aleatoria o no.

Exploración

Como hemos indicado, el proceso de exploración está controlado por una variable externa para tomar una acción de una manera aleatoria o siguiendo la estrategia que se está aprendiendo.

Los valores típicos para esta variable van desde 0.99 (acción siempre aleatoria) hasta 0.05 (acción la mayoría de veces usando la *policy* aprendida).

Justo durante el intervalo de tiempo entre estos dos valores será nuestro tiempo de exploración.

```
# Python code

epsilon = 0.99

# During the simulations
(...)

random_number = np.random(...)

if random_number > epsilon:
    # policy action
else:
    # random action

epsilon -= 0.01
```

Explotación



EXPLORATION

Playing the machine that (currently) pays out the most.

<http://www.cs.us.es/~fsancho/images/2018-01/exploitation-exploration-1024x370.png>

Una vez que el agente ha explorado un tiempo suficiente (esto es totalmente dependiente del problema), comienza el proceso de explotación.

En esta parte, el agente ha aprendido cómo tomar decisiones a partir del proceso de aprendizaje, sin necesidad de explorar situaciones nuevas.

Es común dejar siempre un grado de aleatoriedad para la decisión de qué acción tomar, aunque esa probabilidad es muy pequeña.

Explotación

Como hemos visto en la diapositiva anterior, en explotación consideramos que el agente ha aprendido lo suficiente, por lo que las acciones que se seleccionan las obtenemos directamente de nuestra *policy*.

Para ello obtenemos las acciones directamente usando nuestro modelo, que es el encargado de aproximar la mejor *policy* aprendida.

Este estado es la situación típica cuando desplegamos nuestra solución.

Clasificación de problemas de aprendizaje por refuerzo

En la primera sesión vimos distintos enfoques a la hora de clasificar los problemas y retos que nos podemos encontrar cuando queremos aplicar técnicas de aprendizaje por refuerzo. La mayoría de estos enfoques estaban definidos a alto nivel.

Si bajamos a detalle, podemos analizar los problemas de aprendizaje por refuerzo desde el punto de vista del comportamiento del agente, de la información que tenemos a nuestra disposición, etc. Este análisis nos permite saber qué algoritmo es el más adecuado acorde al reto que queramos solucionar.

En las siguientes diapositivas nos centraremos en dos clasificaciones básicas:
Basada en modelo
Basada en estrategia

Hay más, pero para nuestro objetivo estas dos son las más importantes.

Basados en Modelo

Cuando nos referimos a “algoritmos de aprendizaje por refuerzo basados en modelo” podemos encontrar dos posibilidades: *model free* y *model based*.

En *model free* la simulación no conoce un modelo para “predecir” siguientes estados en el entorno. Es el caso típico que nos encontramos en la mayoría de ejemplos hoy en día, como es el caso en casi todos los videojuegos. El comportamiento de los entornos en este tipo de simulaciones no son deterministas, de ahí que no se pueda conocer un modelo.

Al no poder anticipar siguientes estados, la estrategia que busca el agente es dependiente sólo del estado actual en el que se encuentra.

Basados en Modelo



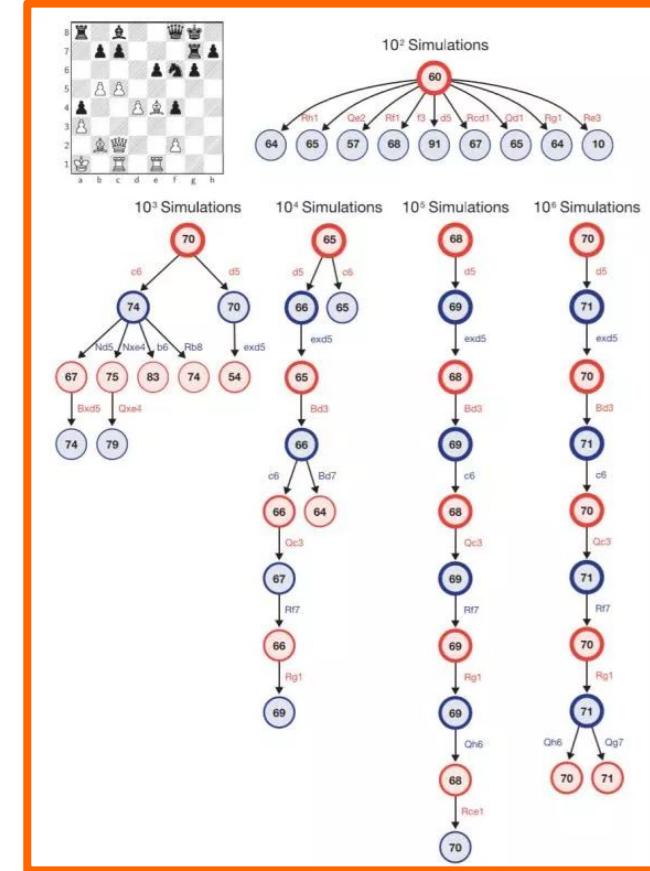
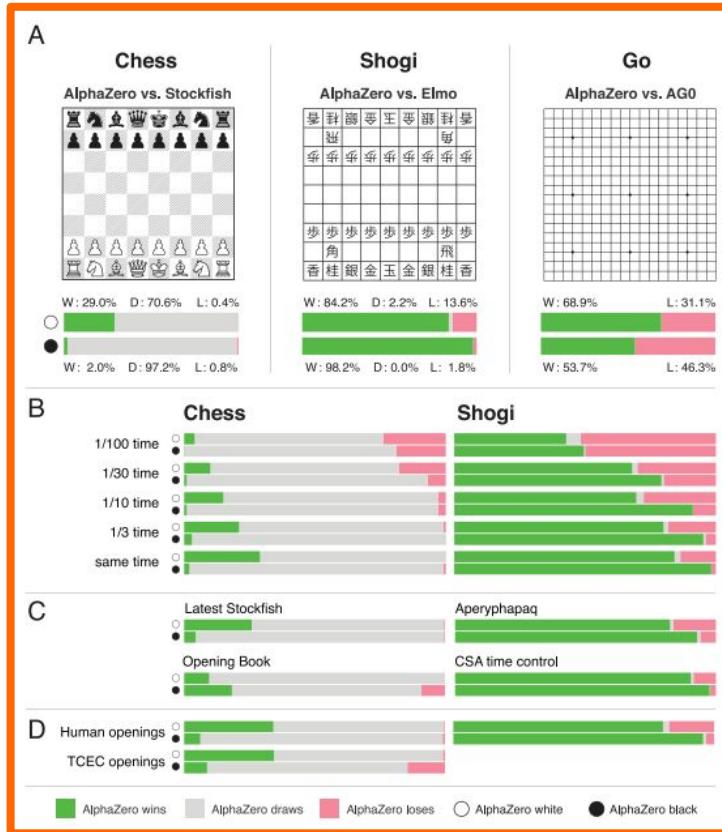
Basados en Modelo

En el caso de *model based* sí que conocemos un modelo de cómo se comporta el entorno en relación con las posibles acciones que el agente tome durante la ejecución.

Al tener el conocimiento de este modelo, es típico combinar los algoritmos de aprendizaje con refuerzo con otras soluciones de inteligencia artificial como técnicas de planificación y búsquedas óptimas con objetivo de mejorar el proceso de aprendizaje.

Por ello, en este tipo de enfoque el agente no sólo se encarga de aprender la estrategia óptima sino que también esta estrategia va acompañada de algoritmos de optimización para evitar los costes computacionales de hacer búsquedas *por fuerza bruta*.

Basados en Modelo



Basados en Estrategia

Una vez vista la primera clasificación, ahora es el turno de la otra opción: por estrategia. Evidentemente, cuando hablamos de estrategia nos referimos a nuestra definición de estrategia o *policy*. Al igual que con la clasificación basada en modelos, tendremos dos posibles conjuntos para identificar a los algoritmos de aprendizaje por refuerzo.

El primero de ellos es el que se conoce como *on policy*. Durante el proceso de aprendizaje, la estrategia que nuestros agentes siguen puede ir cambiando en el tiempo. Con *on policy* nos referimos a que el agente sólo puede usar la experiencia pasada de una estrategia específica.

Normalmente este tipo de aprendizaje se caracteriza por más tiempo de entrenamiento para encontrar una solución óptima así como de más varianza en los datos a la hora de entrenar a nuestro agente. En este tipo de aprendizaje encontramos muchos de los algoritmos de la familia de *Policy gradients*.

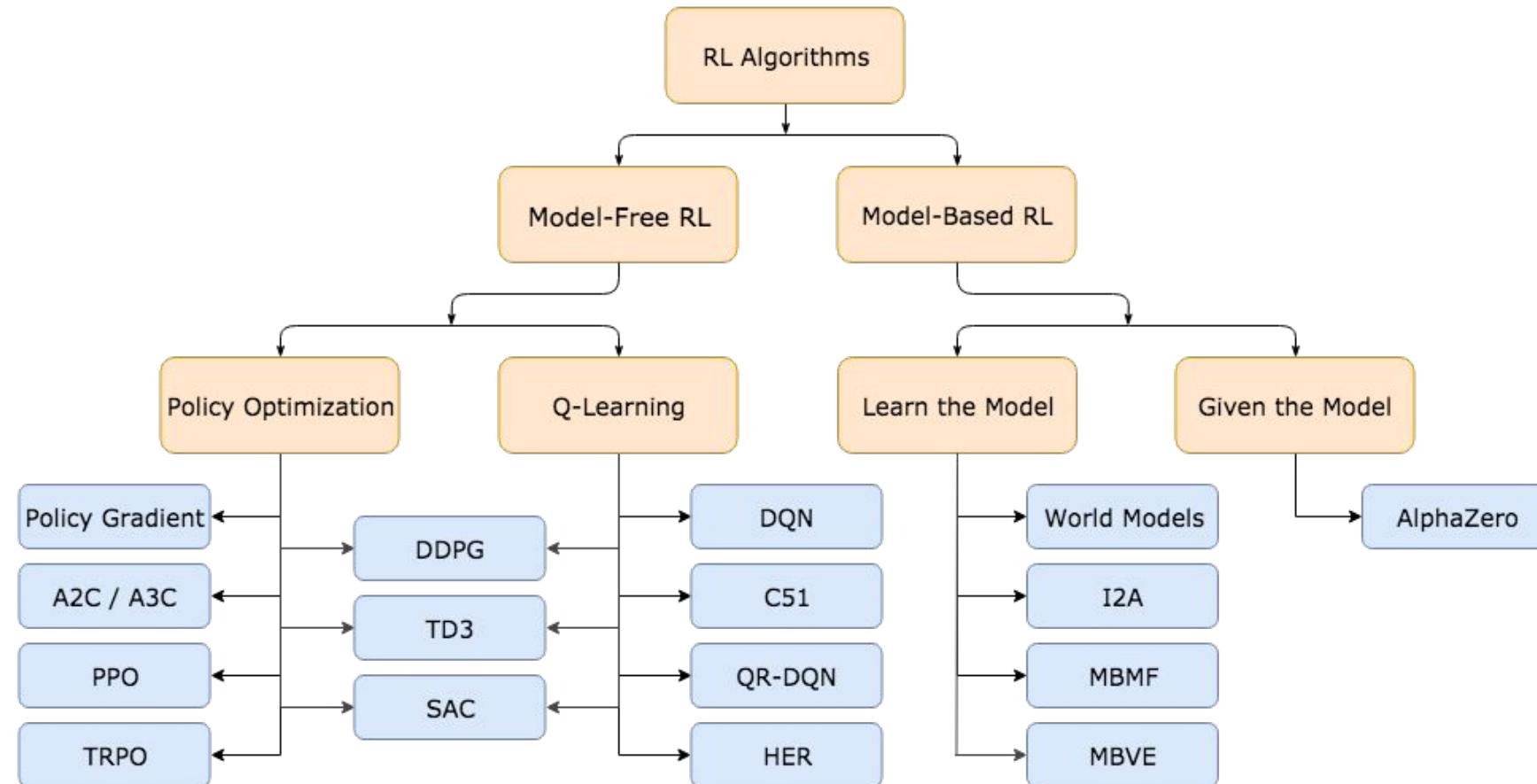
Basados en Estrategia

Por otro lado encontramos la otra opción para clasificar nuestros algoritmos, una estrategia *off policy*. Este es el caso contrario, podemos usar experiencia adquirida con estrategias distintas cada vez que queramos hacer una actualización del aprendizaje de nuestro agente.

Esta situación está muy relacionada con conceptos vistos durante la sesión, por ejemplo la experiencia. Si ejecutamos nuestra simulación y vamos almacenando la experiencia que el agente va adquiriendo a la vez que el agente va aprendiendo, es normal que encontremos transiciones que pertenecen a estrategias distintas.

Este enfoque, como todos, tiene sus pros y sus contras. El punto a resaltar es que aunque podamos usar más información pasada para el aprendizaje, puede conllevar a más tiempo de convergencia dependiendo de la información almacenada. Un ejemplo de familia de algoritmos que siguen este tipo de comportamiento son las *Q-networks*.

Jerarquía de algoritmos



Conclusiones

- La terminología y los conceptos vistos son muy específicos de aprendizaje por refuerzo. Hay que saber diferenciarlos de otras definiciones dentro de la inteligencia artificial o el mundo tecnológico.
- Todos los conceptos están muy relacionados entre sí. En este sentido destacan los conceptos de Observación y Estado, así como Experiencia y Transición.
- La aplicación de los conceptos vistos es muy flexible dependiendo del algoritmo usado y del reto que se quiera resolver.
- Las clasificaciones presentadas son fundamentales para entender qué nos ofrece cada enfoque dentro de todas las posibilidades de las soluciones de aprendizaje por refuerzo.

Bibliografía recomendada

“Reinforcement Learning: An introduction”, Sutton y Barto:

<http://incompleteideas.net/book/bookdraft2017nov5.pdf>

(Capítulo 2, *Multi Armed Bandits*, donde se trata el dilema de exploración-explotación)

Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems

Levine S. et al, <https://arxiv.org/pdf/2005.01643.pdf>

Aprendizaje por refuerzo

Sesión 3 – Algoritmos base: Deep Q-network

Curso 21/22



viu

**Universidad
Internacional
de Valencia**

De:

 Planeta Formación y Universidades

Índice

Definición *Q-learning*

Conceptos importantes

Ejemplo *Q-learning*: *gridworld*

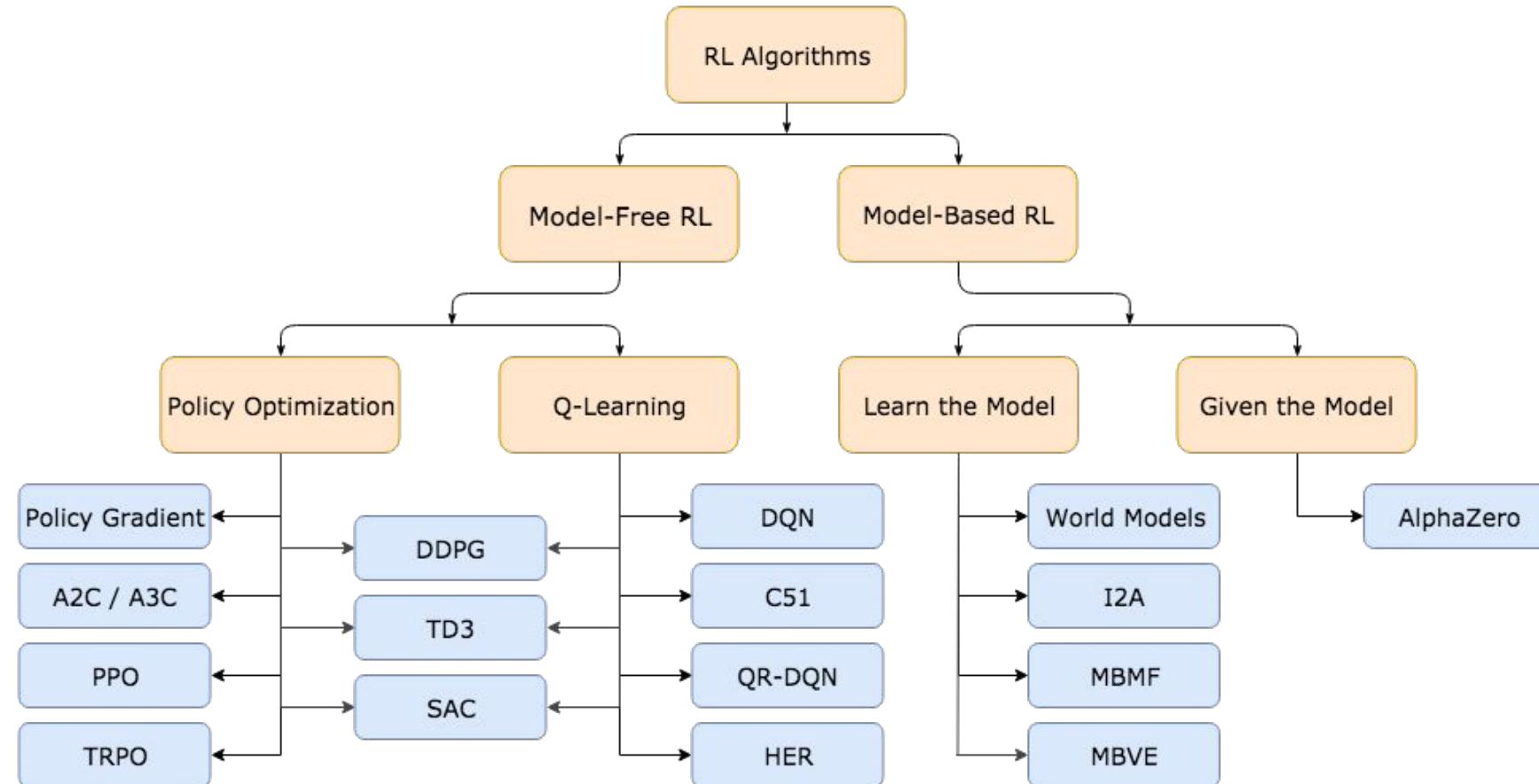
Deep Q-network

Proceso de aprendizaje

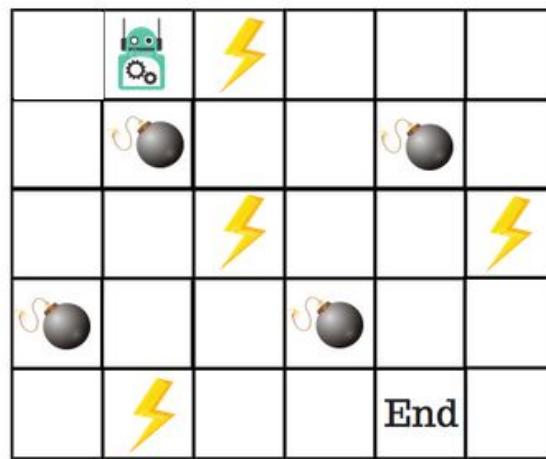
Algoritmo *DQN*

Conclusiones

Definición Q-learning



Definición Q-learning



Actions : ↑ → ↓ ←

	↑	→	↓	←
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

Cuando hablamos de DQN, tenemos que comenzar hablando de Q-learning.

Q-learning es un algoritmo de aprendizaje por refuerzo que se basa en el aprendizaje a partir de diferencias temporales. Matemáticamente, su enfoque es como una cadena de Markov, en el que la transición entre estados sólo depende de la información que tenemos en el estado actual.

Por ello, la teoría nos dice que existe una función capaz de modelar qué acción es la mejor en cada estado.

Definición Q-learning

Usando esta función Q podríamos encontrar la estrategia óptima para nuestro agente. Los parámetros de la función Q van a ser pares estado-acción. Es común usar una tabla que relacione estados con acciones.

Esta función nos devolverá, para cada par estado-acción, "el valor esperado de la suma de las recompensas futuras". De esta forma podemos ir midiendo en cada estado cuál es la acción que más nos conviene tomar para maximizar la recompensa futura.

Conceptos importantes

$$Q^\pi(s, a) = E[R_t]$$

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

Para encontrar la estrategia óptima, este algoritmo se basa en el proceso de exploración-explotación que vimos en la sesión anterior.

La estrategia irá tomando la acción que maximiza el valor de la recompensa esperada.

Respecto a la recompensa esperada, es común usar un ***discount factor*** para estimar la importancia que tendrán los valores futuros.

Conceptos importantes

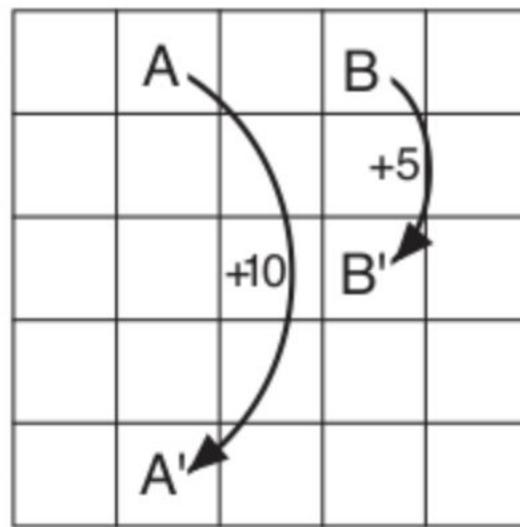
Pero, ¿dónde queda la idea de diseñar como cadena de Markov?

Para ello usaremos la ecuación de Bellman. Como trabajaremos con la recompensa esperada a futuro, necesitamos alguna forma de modelar este comportamiento.

“La ecuación de Bellman proporciona una definición recursiva con la que podremos encontrar la función óptima Q. La fórmula $Q^(s, a)$ es igual a la suma de la recompensa inmediata, después de ejecutar la función A en el estado S, y la recompensa futura esperada después de la transición al siguiente estado S”.*

$$Q^*(s, a) = r + \gamma \max_{a'} Q(s', a')$$

Ejemplo Q-learning: *gridworld*



- Ejemplo *gridworld* de *Sutton & Barto, Capítulo 3*
- Las celdas corresponden con los estados del entorno.
- Cuatro acciones disponibles. Todas tienen la misma probabilidad de ejecutarse (1/4)
- La recompensa para cada acción-estado es:
 - +10 para la transición de A a A' (que es la única acción que se puede ejecutar en A)
 - +5 para la transición de B a B' (que es la única acción que se puede ejecutar en B')
 - -1 si el movimiento se sale del *grid*
 - 0 en cualquier otro caso

Ejemplo Q-learning: *gridworld*

```
array([
    [0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0.],
    [0., 0., 0., 0., 0.]])
```

Comenzamos en la posición $[0, 0]$. Para cada acción posible:

- Si nos movemos arriba:
 $0.25 * (-1 + 0.9 * 0) = -0.25$
- Si nos movemos abajo:
 $0.25 * (0 + 0.9 * 0) = 0$
- Si nos movemos a la derecha:
 $0.25 * (0 + 0.9 * 0) = 0$
- Si nos movemos a la izquierda:
 $0.25 * (-1 + 0.9 * 0) = -0.25$

El valor esperado en este estado es ~ -0.5

Ejemplo Q-learning: *gridworld*

```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

[0.]

Una vez hemos calculado el valor en $[0, 0]$, si nos movemos a $[1, 0]$:

- Si nos movemos arriba:

$$0.25 * (0 + 0.9 * -0.5) = -0.1125$$

- Si nos movemos abajo:

$$0.25 * (0 + 0.9 * 0) = 0$$

- Si nos movemos a la derecha:

$$0.25 * (0 + 0.9 * 0) = 0$$

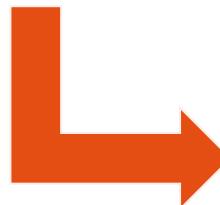
- Si nos movemos a la izquierda:

$$0.25 * (-1 + 0.9 * 0) = -0.25$$

El valor esperado en este estado es ~ -0.4

Ejemplo Q-learning: *gridworld*

```
array([[-0.5, 10., 2., 5., 0.6],
       [-0.4, 2.2, 0.9, 1.3, 0.2],
       [-0.3, 0.4, 0.3, 0.4, -0.1],
       [-0.3, 0., 0.1, 0.1, -0.3],
       [-0.6, -0.4, -0.3, -0.3, -0.6]])
```



```
array([[ 1.4,  9.7,  3.7,  5.3,  1. ],
       [ 0.4,  2.6,  1.8,  1.7,  0.4],
       [-0.2,  0.6,  0.6,  0.5, -0.1],
       [-0.5, -0.,  0.1,  0., -0.5],
       [-1., -0.6, -0.5, -0.6, -1. ]])
```



Q-values finales para conocer la estrategia óptima.

```
array([[ 3.4,  8.9,  4.5,  5.4,  1.6],
       [ 1.6,  3.1,  2.4,  2.,  0.6],
       [ 0.2,  0.9,  0.8,  0.5, -0.3],
       [-0.8, -0.3, -0.2, -0.5, -1.1],
       [-1.7, -1.2, -1.1, -1.3, -1.9]])
```

Deep Q-network

Al ver Q-learning, hemos ido aproximando los *q_values* durante las iteraciones de nuestro proceso de aprendizaje. Nuestro agente ha ido aprendiendo una estrategia a partir de la tabla que se va creando.

Esto ha sido posible porque el problema no era demasiado complejo en términos de dimensionalidad (datos y atributos). En la realidad, la complejidad hace que esta forma de aprendizaje sea intratable.

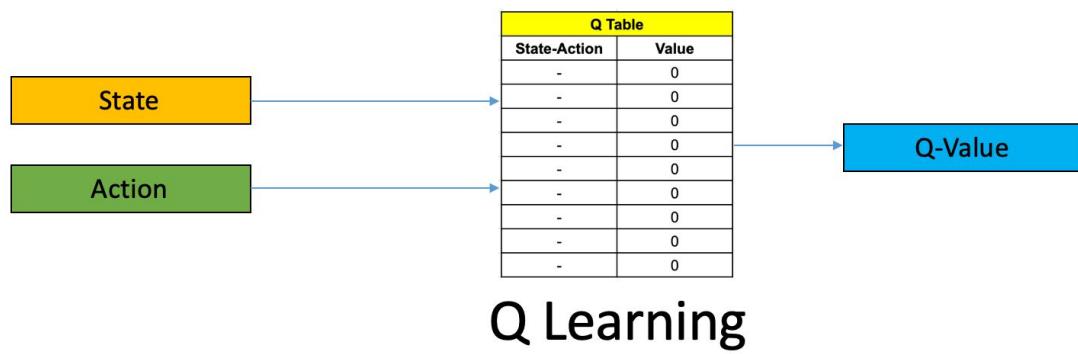
Por ello, en vez de una tabla de *q_values* usaremos una red neuronal como función aproximadora . En vez de aprender los *q_values*, aprenderemos los parámetros de nuestro modelo. Normalmente, y debido a que la mayoría de simulaciones trabajan con la pantalla directamente, el tipo de red neuronal que usaremos serán redes convolucionales.

Deep Q-network

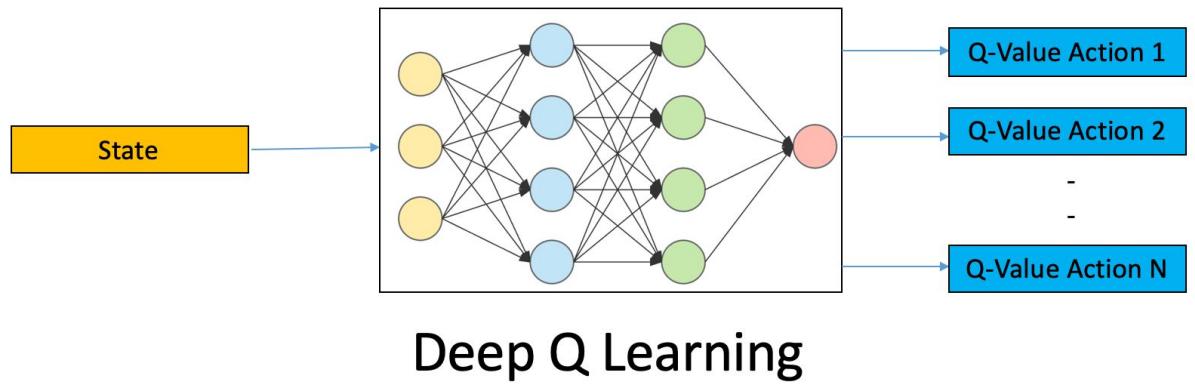
El uso de redes neuronales como función aproximadora conlleva más decisiones por nuestra parte a la hora del desarrollo de nuestras soluciones.

Ya comentamos en sesiones pasadas que a todos los hiperparámetros necesarios de los algoritmos de aprendizaje por refuerzo hay que sumarle los hiperparámetros de los modelos basados en Deep Learning. Esto hace que el proceso de aprendizaje sea empírico, en el que la prueba y error de nuestras hipótesis es fundamental.

Deep Q-network



Q Learning



Deep Q Learning

Proceso de aprendizaje

Como queremos aproximar la función Q con un modelo de Deep Learning, necesitamos una función de coste que mida cómo lo estamos haciendo.

Podemos usar la ecuación de Bellman para, iterativamente, aproximar la función Q usando aprendizaje basado en diferencia temporal.

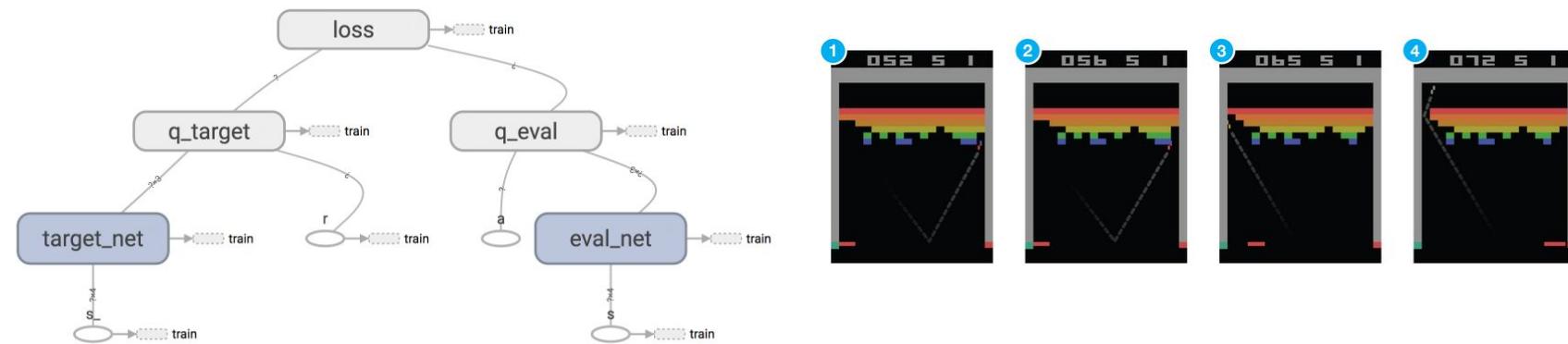
Concretamente, en cada instante de tiempo buscaremos minimizar el error que se produce entre $Q(s, a)$ (término que se predice) y la ecuación de Bellman (término objetivo).

$$loss = \left(\underbrace{r + \gamma \max_{a'} \hat{Q}(s, a')}_{\text{Target}} - \underbrace{Q(s, a)}_{\text{Prediction}} \right)^2$$

Reward Decay Rate

Proceso de aprendizaje

Respecto al proceso de aprendizaje y a las DQN, hay otros dos elementos que son fundamentales para asegurar la convergencia de nuestro modelo. Estos elementos son la *target network* y el uso de una *secuencia de frames*.



Algoritmo DQN

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{D} to capacity N

Initialize action-value function Q with random weights

for episode = 1, M **do**

 Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

Conclusiones

- Hemos estudiado el primero de los algoritmos base que veremos en la asignatura, DQN. Este algoritmo pertenece a la familia de métodos *off-policy*.
- Está basado en Q-learning, donde el objetivo es estimar la recompensa esperada para cada par estado/acción.
- DQN combina Q-learning con arquitecturas de Deep Learning, para poder abarcar problemas con espacios de dimensiones muy grandes
- Algunas variaciones, necesarias para la convergencia de la solución, en la versión final del algoritmo de DQN son el uso de una target network y de secuencias de frames como datos de entrada del modelo del agente.

Bibliografía recomendada

- *Human level control through Deep Reinforcement learning, Google Deepmind*
<https://deepmind.com/research/publications/2019/human-level-control-through-deep-reinforcement-learning>
- *An Introduction to Q-learning: Reinforcement Learning, Sayak Paul, Floydhub*
<https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/>

08MIAR - Aprendizaje por refuerzo

Sesión 4 – Algoritmos base: Policy Gradient

Curso 21/22



**Universidad
Internacional
de Valencia**

De:

Planeta Formación y Universidades

Índice

Definición *Policy Gradient*

Conceptos importantes

Deep Policy Gradient

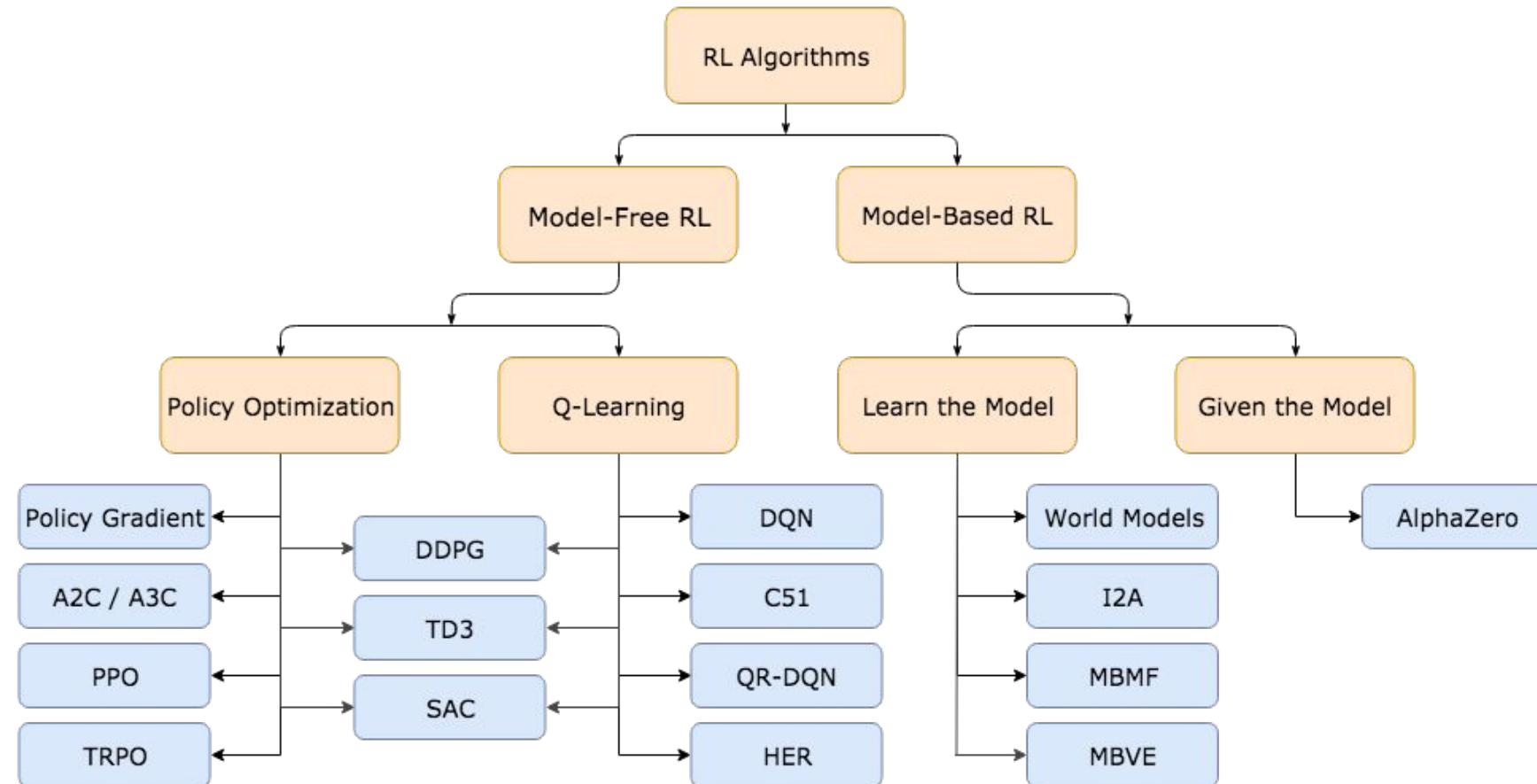
Proceso de aprendizaje

Algoritmo: REINFORCE

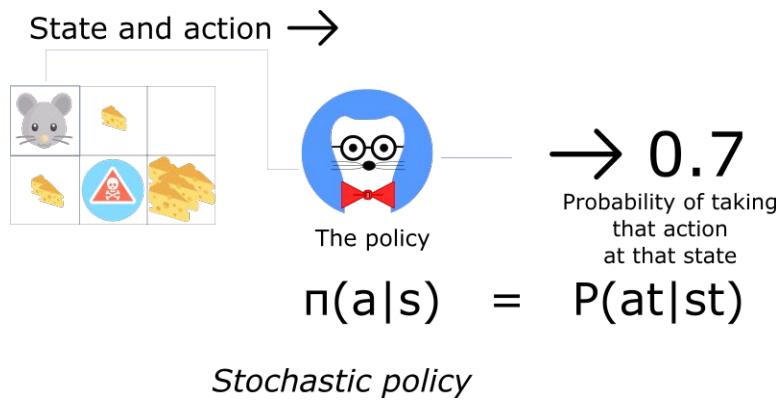
Algoritmo: Vanilla Policy Gradient

Conclusiones

Definición Policy Gradient



Definición Policy Gradient



En el algoritmo anterior, nuestro objetivo era aproximar una función, la función Q. Con esta función podríamos encontrar la estrategia óptima usando los valores de recompensa esperada a partir de la relación entre estado y acción.

Ahora, en vez de centrarnos en esos valores, trabajaremos directamente sobre la estrategia, es decir, sobre la distribución de probabilidades de las acciones siguiendo la estrategia que se está aprendiendo.

Definición Q-learning

Optimizar directamente la estrategia es una técnica ampliamente reconocida dentro de los algoritmos de aprendizaje por refuerzo. Algunos de los métodos que actualmente son el estado del arte en muchos problemas tienen como su base el algoritmo de *Policy Gradient*.

El nombre *Policy Gradient* viene justo de la idea de atacar directamente a la *policy* para maximizar la recompensa esperada, de ahí que se busca el gradiente de la *policy* para maximizar ese valor.

¿Cuál será la forma de maximizar la recompensa esperada? Intuitivamente, cuando estemos en un estado tendremos a nuestra disposición un conjunto de acciones. De entre estas acciones, potenciaremos las que nos devuelvan una recompensa positiva mientras que evitaremos (o ponderaremos negativamente) las que nos devuelvan una recompensa negativa.

Conceptos importantes

Al ser nuestro objetivo modelar la selección de acciones, en cada *step* obtendremos una lista de probabilidades, cada una relacionada con las acciones disponibles para el agente en un estado.

Las probabilidades de cada acción para ese estado se irán actualizando, acorde al factor con el que potenciamos la acción. Este proceso está guiado por la maximización de la recompensa esperada.

$$R_t = \sum_{i=t}^T \gamma^{i-t} r_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + r^{T-t} r_T$$

$$\begin{aligned} J(\theta) &= \mathbb{E}\left[\sum_{t=0}^{T-1} r_{t+1} | \pi_\theta\right] \\ &= \sum_{t=i}^{T-1} P(s_t, a_t | \tau) r_{t+1} \end{aligned}$$

Conceptos importantes

Si en las DQN usábamos la ecuación de Bellman para encontrar nuestra estrategia óptima, en *Policy Gradients* usaremos una función similar a la usada en el ámbito de Deep Learning, *Cross-Entropy function*.

$$\mathcal{L}(y - \hat{y}) = - \sum_{i=1}^n y_i \log \hat{y}_i$$

Siguiendo con la idea de potenciar las acciones “buenas”, en su definición más básica usaremos como ponderación la propia recompensa de tomar una acción determinada. Esta idea está bien como base para definir nuestros algoritmos, pero veremos cómo se producen algunas situaciones no deseadas con este enfoque que tendremos que evitar.

$$\mathbb{E}[f(x)] = \sum_x P(x)f(x)$$

Conceptos importantes

Algunas de las ventajas de usar Policy Gradients son:

- 1) Los métodos basados en Policy Gradient tienen mejores propiedades desde el punto de vista de la convergencia, ya que se optimizan los parámetros de la policy directamente en vez de utilizar los valores de las acciones.
- 2) Este tipo de métodos están más preparados para abordar retos que contengan un espacio de acciones *muy* grande o que trabajen en espacio de acciones continuos.
- 3) Algoritmos basados en Policy Gradient pueden aprender estrategias estocásticas, lo que implica que no se necesite utilizar el enfoque de exploración/explotación que vimos en DQN (más sobre esto en la siguiente sección).

Deep Policy Gradient

En el caso de Policy Gradient, la combinación con técnicas de Deep Learning es similar al ejemplo visto en DQN.

Como se ha comentado en las secciones anteriores, el detalle más importante es el hecho de que ahora estamos modelando la probabilidad de las acciones y no el valor de recompensa esperado.

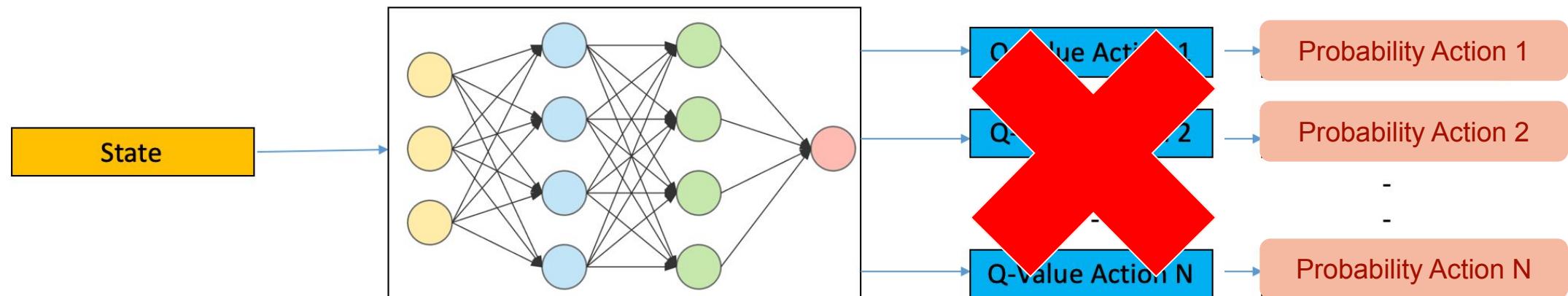
Por ello, normalmente encontraremos como función de activación en la salida del modelo de Deep Learning una función *Softmax*, para poder modelar nuestra capa de salida como una distribución de probabilidad. Si el conjunto de acciones es reducido, podemos encontrar otras opciones como *Sigmoid*.

Deep Policy Gradient

Otro detalle importante es cómo se realiza el proceso de exploración. A diferencia de DQN, en Policy Gradient (y toda su familia de algoritmos) la exploración va intrínseca en el modo en el que se seleccionan las acciones ya que se escogen siguiendo una **distribución aleatoria de probabilidad**.

En cada step de la ejecución, se calcularán las probabilidades de las acciones para el estado en el que se encuentra el agente y se utilizará esa distribución de probabilidades como los pesos de la selección de la acción de manera aleatoria.

Deep Policy Gradient



Deep Q Learning

Proceso de aprendizaje

Como indicábamos hace unas diapositivas, en *Policy Gradients* usaremos un factor que nos dirá cómo de bueno es tomar una acción o no.

Ese factor de escala será R_t y decidirá como la probabilidad $P(a)$ debe cambiar para maximizar la recompensa futura esperada.

“Si una acción es buena (por ejemplo, R_t con valor muy grande), $P(a)$ será multiplicado por un peso grande. Por otro lado, si la acción es mala, la probabilidad $P(a)$ se descartará. Eventualmente, acciones buenas incrementarán su probabilidad para ser seleccionadas en iteraciones futuras.”

$$\nabla_{\theta} E[R_t] = E[\nabla_{\theta} \log P(a) R_t]$$

Proceso de aprendizaje

La función de coste que se utiliza en Policy Gradient surge de la aplicación de lo que se conoce como *Log Derivative Trick*. Básicamente, nuestra función de coste calcula el *ratio* entre la actualización de la policy y la probabilidad de la acción tomada, para de esa manera no impactar negativamente al proceso de optimización cuando una acción tiene probabilidad alta:

$$\nabla \ln f(x) = \frac{\nabla f(x)}{f(x)} \quad \longrightarrow \quad \frac{\nabla \pi_\theta(a|s)}{\pi_\theta(a|s)} = \nabla_\theta \log \pi_\theta(s|a)$$

Ese *ratio* lo podemos transformar en el logaritmo que utilizaremos en la función de coste, manteniendo las mismas propiedades desde un punto de vista matemático para nuestro proceso de optimización.

Proceso de aprendizaje

Durante el proceso de aprendizaje debemos tener algunos conceptos y situaciones presentes, para entender qué está ocurriendo:

- Una de las primeras decisiones que debemos tomar es “¿Cuántos *steps* vamos a usar para ir modificando la estrategia?”. El proceso de aprendizaje se puede ver más o menos impactado dependiendo del número de iteraciones que realicemos para ir almacenando nuestra experiencia.
- Además, al trabajar con la trayectoria, las recompensas obtenidas se procesarán siguiendo un enfoque conocido como *discounted rewards*. Al tener una trayectoria finita, utilizaremos las recompensas en sentido inverso para ir estimando la recompensa esperada en los siguientes estados y de esta forma poder ponderar las acciones de manera adecuada.
- Por otro lado, usar la recompensa como factor de las probabilidades de las acciones produce una varianza en los datos muy grande. Tened en cuenta que con esta definición la probabilidad de una acción en un estado puede cambiar dependiendo de si la recompensa cambia también. Esto dificulta el aprendizaje ya que no se encuentra una correlación entre estado y probabilidad de acción fácilmente. Esta situación se puede dar en muchos escenarios, sobre todo en las simulaciones basadas en videojuegos.

Proceso de aprendizaje

Para suavizar el problema con el uso de la recompensa como factor, hay otras definiciones como:

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_{\theta} \mathbb{E} [\sum_{t=0}^{\infty} r_t]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E} \left[\sum_{t=0}^{\infty} \Psi_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right], \quad (1)$$

where Ψ_t may be one of the following:

- | | |
|--|---|
| 1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory. | 4. $Q^{\pi}(s_t, a_t)$: state-action value function. |
| 2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t . | 5. $A^{\pi}(s_t, a_t)$: advantage function. |
| 3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula. | 6. $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: TD residual. |

The latter formulas use the definitions

$$V^{\pi}(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad Q^{\pi}(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t+1:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad (2)$$

$$A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t), \quad (\text{Advantage function}). \quad (3)$$

Algoritmo: Vanilla Policy Gradient

Algorithm 1 “Vanilla” policy gradient algorithm

Initialize policy parameter θ , baseline b

for iteration=1, 2, . . . **do**

 Collect a set of trajectories by executing the current policy

 At each timestep in each trajectory, compute

 the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and

 the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.

 Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
 summed over all trajectories and timesteps.

 Update the policy, using a policy gradient estimate \hat{g} ,
 which is a sum of terms $\nabla_\theta \log \pi(a_t | s_t, \theta) \hat{A}_t$

end for

Conclusiones

- Policy Gradient es uno de los algoritmos base dentro del aprendizaje por refuerzo. Este algoritmo es el origen de algunos de los algoritmos más potentes actualmente. Es un algoritmo de tipo on-policy.
- En comparación con DQN, algunas de las características a destacar son una mayor capacidad de convergencia, apto para trabajar con espacios de acciones grandes (y continuos) y posibilidad de aprender policies estocásticas.
- La función de coste en Policy Gradient se centra en ir optimizando la propia policy aplicando *gradient ascent* sobre la probabilidad de la acción seleccionada y su recompensa obtenida.
- Al ser un algoritmo base, veremos en las siguientes sesiones de la asignatura las evoluciones que se han ido produciendo para estabilizar y mejorar el proceso de aprendizaje del agente.

Bibliografía recomendada

- “Reinforcement Learning: An Introduction”, Sutton y Barto:
<http://incompleteideas.net/book/bookdraft2017nov5.pdf>
(Capítulo 13, *Policy Gradient Methods*)
- *An Intuitive explanation on Policy Gradients*, Adrien Lucas, Towards data science / Medium
<https://towardsdatascience.com/an-intuitive-explanation-of-policy-gradient-part-1-reinforce-aa4392cbfd3c>

08MIAR - Aprendizaje por refuerzo

Sesión 5 – Evolución del algoritmo de Policy Gradient: Actor-Critic y A2C/A3C

Curso 21/22



**Universidad
Internacional
de Valencia**

De:

 Planeta Formación y Universidades

Índice

Introducción

Actor-Critic: Definición & Conceptos importantes

Algoritmo: *Actor-Critic*

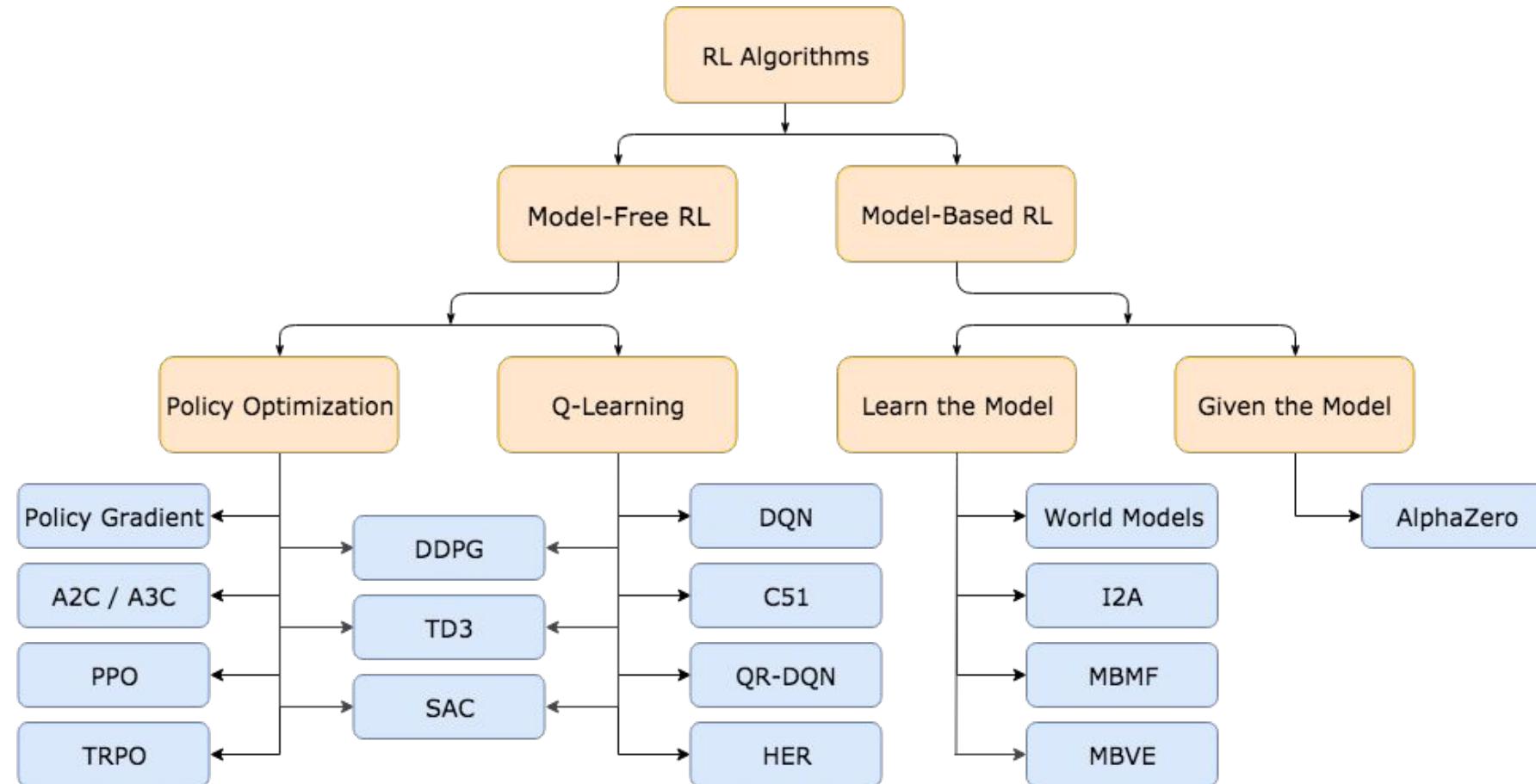
Evolución de *Actor-Critic*: A2C/A3C

Algoritmo: A2C

Algoritmo: A3C

Conclusiones

Introducción



https://spinningup.openai.com/en/latest/_images/rl_algorithms_9_15.svg

Sesión 5 – Algoritmos Actor-Critic y A2C/A3C

Introducción

Antes de continuar con los algoritmos, recordemos justo uno de los puntos más relevantes del algoritmo de Policy Gradient, el uso de la recompensa como factor de relevancia. Este hecho provocaba que en algunos casos las soluciones tuvieran problemas de convergencia y de demasiada varianza a la hora de entrenar el modelo.

Para solventar esta situación y como punto de mejora introducida por estos algoritmos, vamos a presentar el concepto conocido como *Value*. De manera informal, podemos definir el *Value* como:

- El valor medio de recompensa esperada en el estado actual
- Cómo de bueno es estar en el estado actual de la ejecución
- Cómo de bien estoy ejecutando la *policy* actual

El *value* es un concepto fundamental en aprendizaje por refuerzo para poder derivar funciones más complejas a la hora de ponderar las acciones seleccionadas por el agente.

Introducción

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_{\theta} \mathbb{E} [\sum_{t=0}^{\infty} r_t]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E} \left[\sum_{t=0}^{\infty} \Psi_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right], \quad (1)$$

where Ψ_t may be one of the following:

- | | |
|--|---|
| 1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory. | 4. $Q^{\pi}(s_t, a_t)$: state-action value function. |
| 2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t . | 5. $A^{\pi}(s_t, a_t)$: advantage function. |
| 3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula. | 6. $r_t + V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$: TD residual. |

The latter formulas use the definitions

$$V^{\pi}(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad Q^{\pi}(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t+1:\infty}}} \left[\sum_{l=0}^{\infty} r_{t+l} \right] \quad (2)$$

$$A^{\pi}(s_t, a_t) := Q^{\pi}(s_t, a_t) - V^{\pi}(s_t), \quad (\text{Advantage function}). \quad (3)$$

Introducción

Es común confundir o utilizar en el mismo contexto los conceptos de Value y Q_value. La diferencia principal es el parámetro de entrada en cada caso, ya que el Value se calcula a partir del estado donde se encuentra el agente mientras que el Q_value se obtiene relacionando el estado y la acción tomada.

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

$$Q^{\pi}(s_t, a_t) = \underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]$$

Actor-Critic: Definición & Conceptos importantes

El primero de los algoritmos que estudiaremos en la sesión de hoy es el conocido como algoritmo de *Actor-Critic*. La principal característica de este enfoque es que el modelo del agente se divide en dos componentes, el *Actor* y el *Critic*:

- El *Actor* será el responsable de devolver la distribución de probabilidades asociada a las acciones que el agente puede tomar, similar a Policy Gradient.
- El *Critic* se encargará de estimar el *Value* en el estado en el que se encuentre el agente en la ejecución.

El resto de la lógica de este algoritmo es similar a la implementación estudiada de Policy Gradient.

Actor-Critic: Definición & Conceptos importantes

Como este algoritmo sigue comportándose siguiendo una estrategia *on-policy*, el *sampling* de los datos es igual que en el caso de Policy Gradient.

Definimos un número finito de *steps*, que van a definir la trayectoria que vamos a ejecutar, y vamos almacenando toda las transiciones que se van obteniendo a partir de la interacción agente-entorno.

Como novedad, tenemos que tener en cuenta que necesitamos estimar el *Value* en cada estado que hemos almacenado.

$$R_t = \sum_{i=t}^T \gamma^{i-t} r_i = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^{T-t} r_T \quad v_\pi(s) = \mathbb{E}_\pi [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

Actor-Critic: Definición & Conceptos importantes

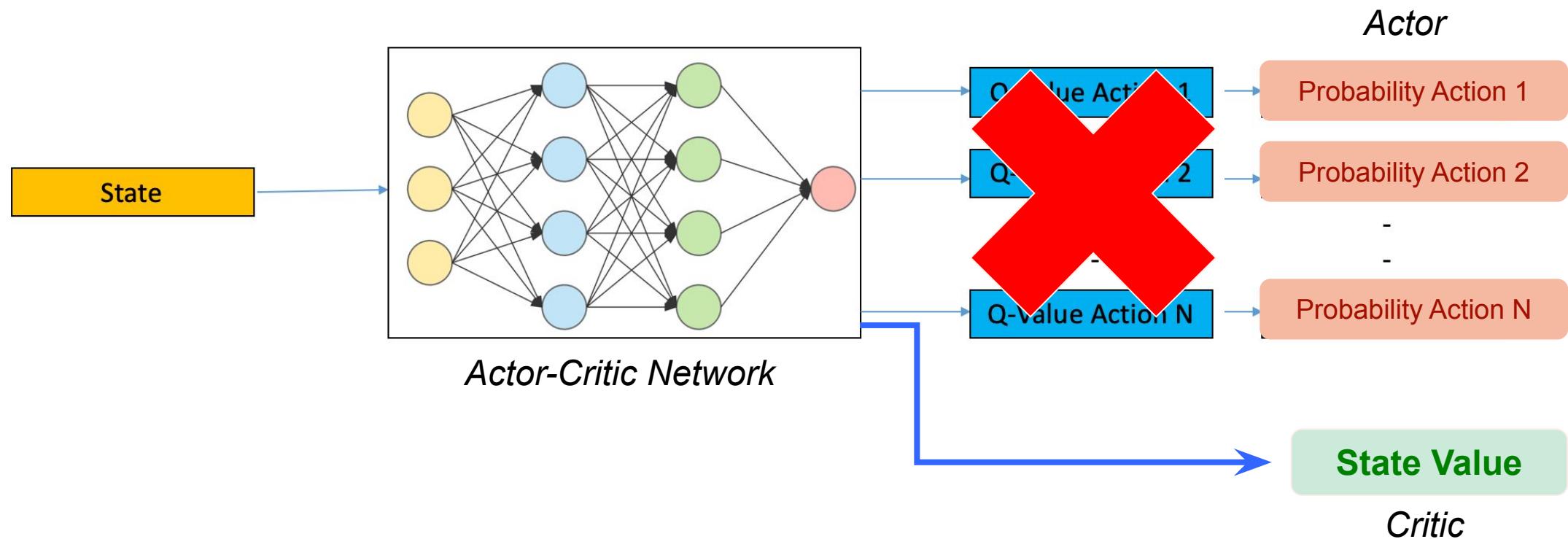
Respecto a la función de coste, al dividir el objetivo en dos estimaciones diferentes, ésta estará compuesta de dos componentes, una parte enfocada en el *Actor* y otra en el *Critic*.

Para el *Actor*, la fórmula que empleamos es la misma que definimos en Policy Gradient, con la variación de que en vez de la recompensa utilizamos el *Value* estimado. El *Critic* estimará el error cometido entre el *Value* recolectado y el *Value* estimado.

El hecho de utilizar el *Value* en la función de coste durante el proceso de aprendizaje va a dotar al entrenamiento de más robustez a la hora de relacionar recompensas obtenidas con las acciones ejecutadas, ya que el *Value* es un estimador más “general” en comparación con la recompensa directa de la acción.

$$\nabla_{\theta} \log \pi_{\theta}(s, a) (R - V_{\varphi}(s)) \quad (R - V_{\varphi}(s))^2$$

Actor-Critic: Definición & Conceptos importantes



Algoritmo Actor-Critic

Algorithm 1 Monte Carlo on policy actor-critic.

Require: Initialize policy π with parameters θ_π and value critic v_π with parameters θ_v

- 1: **for** each episode **do**
 - 2: Get initial state s
 - 3: Initialize storage buffer S, A, R, S'
 - 4: **for** $i = 1, 2, 3 \dots N$ steps **do**
 - 5: Sample action with policy: $a \sim \pi_\theta(s)$
 - 6: Run action through environment, obtain reward and post state: $r, s' \leftarrow ENV(s, a)$
 - 7: Collect and store: $S, A, R, S' \leftarrow s, a, r, s'$
 - 8: $s \leftarrow s'$
 - 9: **end for**
 - 10: Compute discount returns: $\hat{V} = \sum_{l=0}^{N-1} \gamma^l r_{t+l}$
 - 11: Update θ_v to minimize $\sum_{n=1}^N \|v_\pi(s_n) - \hat{V}_n\|^2$
 - 12: With learning rate α , update policy: $\theta_\pi \leftarrow \theta_\pi + \alpha \nabla_\theta \log \pi(A|S) v_\pi(S)$
 - 13: **end for**
-

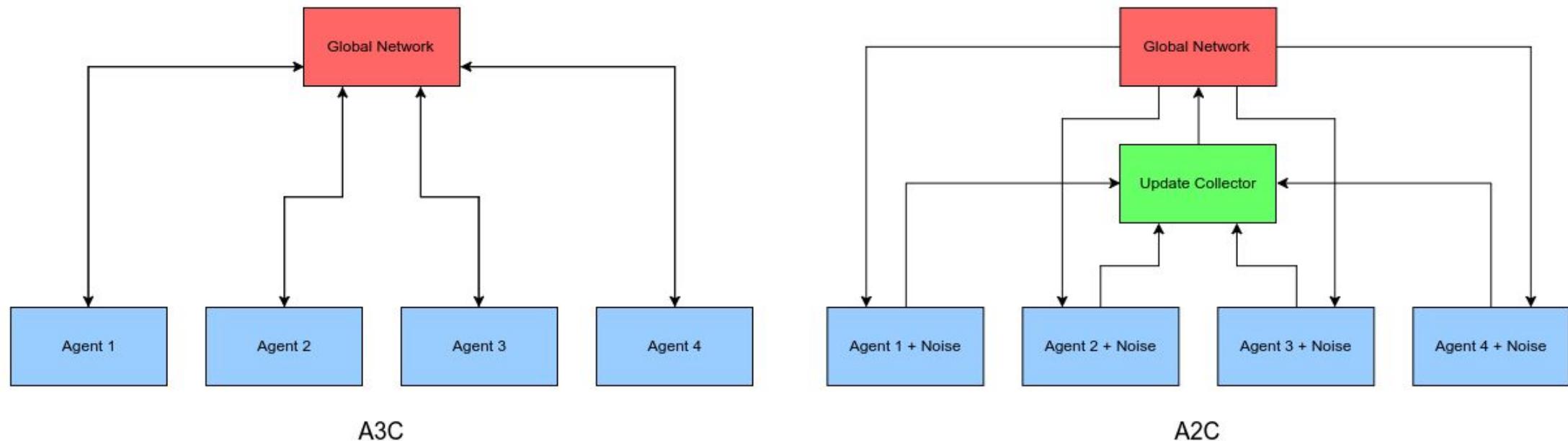
Evolución de Actor-Critic: A2C/A3C

El siguiente de los algoritmos que veremos es el conocido como (*Asynchronous*) *Advantage Actor Critic*, del que podemos encontrar dos versiones: una síncrona y otra asíncrona.

El componente de la sincronicidad está relacionado con el hecho de que esta nueva versión es un algoritmo multi-proceso, en el que una batería de agentes se ejecutan en paralelo durante el entrenamiento en un entorno determinado. En esa ejecución, dependiendo de la comunicación que haya entre ellos, hablaremos de A2C (sincronía) o A3C (asincronía).

Por comunicación nos referimos a cómo se actualiza el modelo global. Hay que tener en cuenta que aunque cada agente tenga su propia versión del modelo para la toma de acciones, todos comparten ese conocimiento global por medio de un modelo general. Cómo se actualice este modelo determina la versión del algoritmo con la que trabajamos.

Evolución de Actor-Critic: A2C/A3C



<https://external-preview.redd.it/uwqehzHvsm6y82P2d1jReEkS1NvhqUv6EPITNZI-aSk.png?auto=webp&s=a68dcc5f54e690d3a2acb2b7510848bc8726ae8>

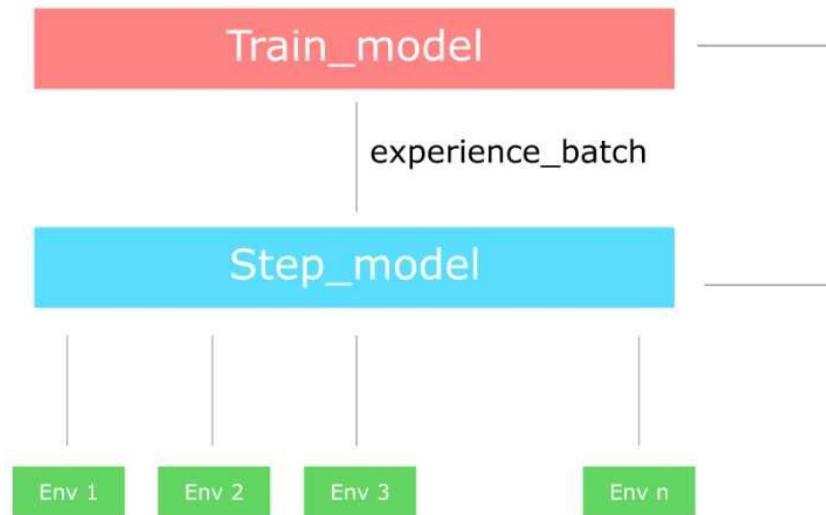
Características principales

Algunas de las características que podemos destacar para A2C/A3C son:

- Posibilidad de modelar espacios de acciones continuos y entornos estocásticos
- Tener una mayor diversidad y volumen de datos disponibles para el proceso de entrenamiento.
- Cada proceso se ejecuta en CPU, por lo que permite trabajar con problemas de visión por computador sin tener que utilizar GPU
- Al utilizar un entrenamiento multi-proceso, los tiempos de entrenamiento se reducen de manera notoria.

Características principales

Una alternativa de entrenamiento en el caso de A2C sería:



Evolución de Actor-Critic: A2C/A3C

El otro elemento que es nuevo en este enfoque es el reemplazo del Value como factor de las acciones tomadas por el agente. En su lugar, usaremos la conocida como función de ventaja:

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad (\text{Advantage function}).$$

Esta función es una medida más precisa para decidir si el agente va por el buen camino o no siguiendo la estrategia actual. Es una medida que compara el *Value* del estado con la recompensa esperada obtenida con la acción tomada en el estado actual, de tal forma que define un umbral de bondad de la acción seleccionada con respecto al resto de opciones.

Evolución de Actor-Critic: A2C/A3C

Actualmente, las funciones de relevancia que más se utilizan en Aprendizaje por refuerzo son funciones derivadas de esta función de Ventaja. Por ejemplo, otra opción que encontraremos ampliamente en la literatura es la conocida como *Generalized Advantage estimation*:

$$\text{GAE}(\gamma, \lambda) : \quad \hat{A}_t := \delta_t + \gamma V(s_{t+1}) - V(s_t)$$

Algoritmo: A2C

Algorithm 1 Parallel advantage actor-critic

```

1: Initialize timestep counter  $N = 0$  and network weights  $\theta, \theta_v$ 
2: Instantiate set  $e$  of  $n_e$  environments
3: repeat
4:   for  $t = 1$  to  $t_{max}$  do
5:     Sample  $a_t$  from  $\pi(a_t | s_t; \theta)$ 
6:     Calculate  $v_t$  from  $V(s_t; \theta_v)$ 
7:     parallel for  $i = 1$  to  $n_e$  do
8:       Perform action  $a_{t,i}$  in environment  $e_i$ 
9:       Observe new state  $s_{t+1,i}$  and reward  $r_{t+1,i}$ 
10:      end parallel for
11:    end for
12:     $R_{t_{max}+1} = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_{t_{max}+1}; \theta) & \text{for non-terminal } s_t \end{cases}$ 
13:    for  $t = t_{max}$  down to 1 do
14:       $R_t = r_t + \gamma R_{t+1}$ 
15:    end for
16:     $d\theta = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} (R_{t,i} - v_{t,i}) \nabla_\theta \log \pi(a_{t,i} | s_{t,i}; \theta) + \beta \nabla_\theta H(\pi(s_{e,t}; \theta))$ 
17:     $d\theta_v = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} \nabla_{\theta_v} (R_{t,i} - V(s_{t,i}; \theta_v))^2$ 
18:    Update  $\theta$  using  $d\theta$  and  $\theta_v$  using  $d\theta_v$ .
19:     $N \leftarrow N + n_e \cdot t_{max}$ 
20: until  $N \geq N_{max}$ 
  
```

Algoritmo: A3C

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```

// Assume global shared parameter vectors  $\theta$  and  $\theta_v$  and global shared counter  $T = 0$ 
// Assume thread-specific parameter vectors  $\theta'$  and  $\theta'_v$ 
Initialize thread step counter  $t \leftarrow 1$ 
repeat
  Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ .
  Synchronize thread-specific parameters  $\theta' = \theta$  and  $\theta'_v = \theta_v$ 
   $t_{start} = t$ 
  Get state  $s_t$ 
  repeat
    Perform  $a_t$  according to policy  $\pi(a_t|s_t; \theta')$ 
    Receive reward  $r_t$  and new state  $s_{t+1}$ 
     $t \leftarrow t + 1$ 
     $T \leftarrow T + 1$ 
  until terminal  $s_t$  or  $t - t_{start} == t_{max}$ 
   $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{Bootstrap from last state} \end{cases}$ 
  for  $i \in \{t - 1, \dots, t_{start}\}$  do
     $R \leftarrow r_i + \gamma R$ 
    Accumulate gradients wrt  $\theta'$ :  $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$ 
    Accumulate gradients wrt  $\theta'_v$ :  $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$ 
  end for
  Perform asynchronous update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ .
until  $T > T_{max}$ 

```

Conclusiones

- A partir el algoritmo de Policy Gradient, la primera variante que hemos analizado es el algoritmo de Actor-Critic, donde el modelo del agente predice las probabilidades de las acciones (Actor) y el Value del estado (Critic)
- El uso del Value como factor de relevancia de las acciones tomadas proporciona más estabilidad durante el aprendizaje. Además, hemos introducido una versión mejorada a partir del Value y de las recompensas estimadas para una acción específica, como es la función de Ventaja.
- Al incluir multiproceso y el uso de la función de Ventaja en el algoritmo de Actor-Critic, hemos desarrollado los algoritmos de A2C/A3C para disminuir los tiempos de entrenamiento y facilitar la convergencia de la solución
- La principal diferencia entre A2C y A3C es que el primero hace actualizaciones del modelo de manera síncrona mientras que el segundo las realiza asíncronamente.

Bibliografía recomendada

- *Asynchronous methods for deep reinforcement learning*, Mnih V. et al, Arxiv

<https://arxiv.org/abs/1602.01783>

- *OpenAI Baselines: ACKTR & A2C*, OpenAI

<https://openai.com/blog/baselines-acktr-a2c/>

08MIAR - Aprendizaje por refuerzo

Sesión 6 – Espacios de acciones continuos: DDPG & PPO

Curso 21/22



viu

**Universidad
Internacional
de Valencia**

De:

 Planeta Formación y Universidades

Índice

Introducción

Deep Deterministic Policy Gradient (DDPG)

Algoritmo: *DDPG*

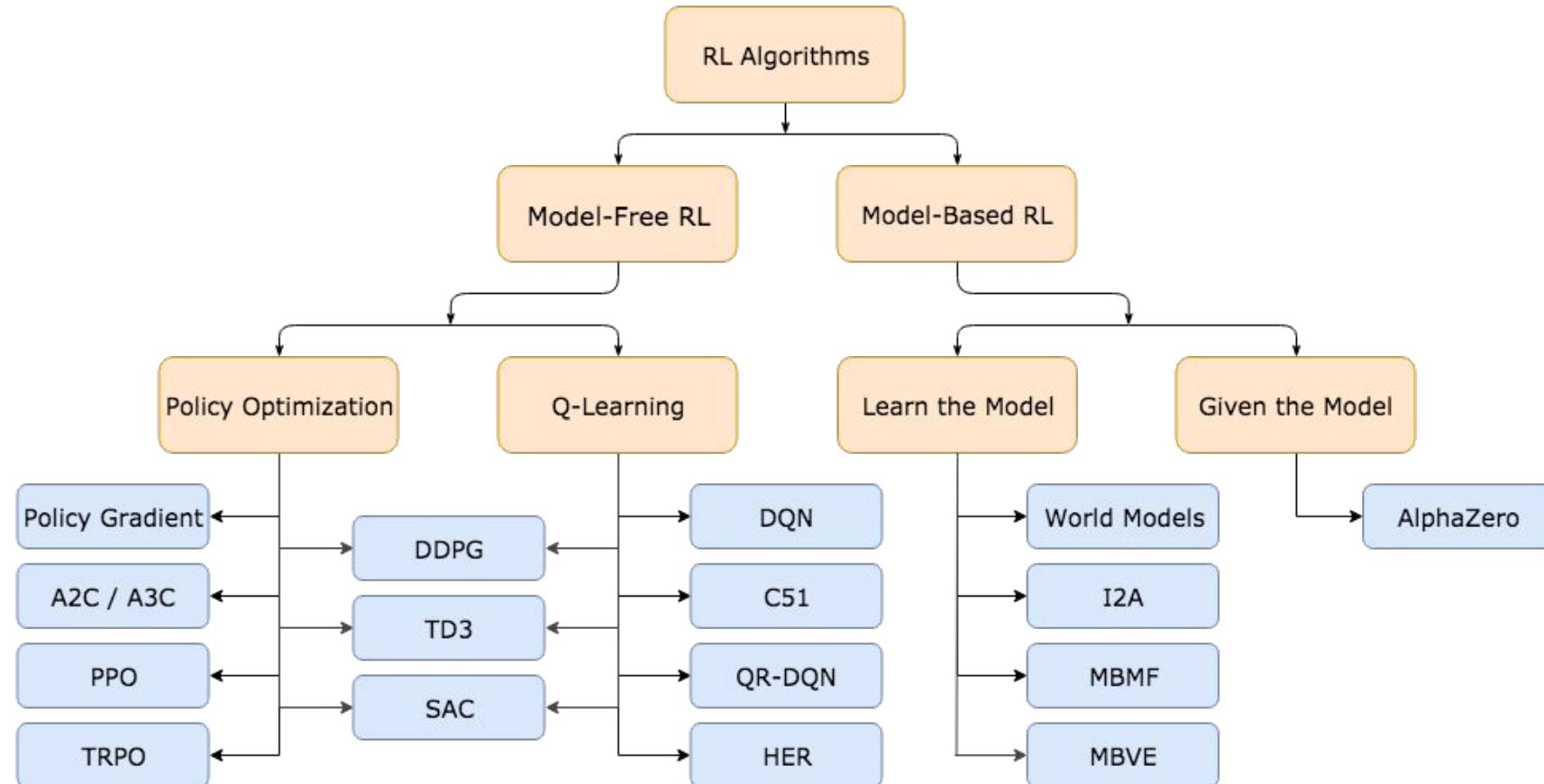
Proximal Policy Optimization (PPO)

Algoritmo: *PPO*

Alternativas a *DDPG* y *PPO*

Conclusiones

Introducción



https://spinningup.openai.com/en/latest/_images/rl_algorithms_9_15.svg

Sesión 6 – Espacios de acciones continuos: DDPG & PPO

Introducción

Una vez hemos estudiado las diferentes versiones de algoritmos de la familia de Policy gradients vamos a entrar en detalle en dos soluciones muy extendidas dentro del mundo del aprendizaje por refuerzo, DDPG y PPO.

De hecho, PPO se puede interpretar como un siguiente nivel dentro de la evolución natural de la familia de Policy Gradient.

Por su parte, DDPG es un algoritmo que combina características de DQN y de Policy Gradient, sacando partido de ambos enfoques.

Es importante resaltar cómo estos dos algoritmos (al igual que la familia de Policy Gradients) pueden trabajar en espacios de acciones continuos.

Introducción

Además de desarrollarse en espacios de acciones continuos, los algoritmos de DDPG y PPO incluyen novedades también desde el punto de vista del aprendizaje, del uso que pueden hacer de una ejecución multiproceso o de la combinación de diferentes arquitecturas de modelos que pertenecen a diferentes definiciones.

Estas características hacen que estos dos algoritmos sean muy utilizados en entornos que, además de la necesidad de trabajar en espacios continuos y complejos, necesiten de una alta carga computacional en la simulación del entorno, como pueden ser entornos de robótica o de navegación autónoma.

Deep Deterministic Policy Gradient

Deterministic Policy Gradient Algorithms

David Silver
DeepMind Technologies, London, UK

Guy Lever
University College London, UK

Nicolas Heess, Thomas Degrif, Daan Wierstra, Martin Riedmiller
DeepMind Technologies, London, UK

DAVID@DEEPMIND.COM

GUY.LEVER@UCL.AC.UK

*@DEEPMIND.COM

Abstract

In this paper we consider *deterministic* policy gradient algorithms for reinforcement learning with continuous actions. The deterministic policy gradient has a particularly appealing form: it is the expected gradient of the action-value function. This simple form means that the deterministic policy gradient can be estimated much more efficiently than the usual stochastic policy gradient. To ensure adequate exploration, we introduce an off-policy actor-critic algorithm that learns a deterministic target policy from an exploratory behaviour policy. We demonstrate that deterministic policy gradient algorithms can significantly outperform their stochastic counterparts in high-dimensional action spaces.

case, as policy variance tends to zero, of the stochastic policy gradient.

From a practical viewpoint, there is a crucial difference between the stochastic and deterministic policy gradients. In the stochastic case, the policy gradient integrates over both state and action spaces, whereas in the deterministic case it only integrates over the state space. As a result, computing the stochastic policy gradient may require more samples, especially if the action space has many dimensions.

In order to explore the full state and action space, a stochastic policy is often necessary. To ensure that our deterministic policy gradient algorithms continue to explore satisfactorily, we introduce an off-policy learning algorithm. The basic idea is to choose actions according to a stochastic behaviour policy (to ensure adequate exploration), but to learn about a deterministic target policy (exploiting the ef-

Deep Deterministic Policy Gradient (DDPG) es un algoritmo que aprende de manera simultánea una función Q y una policy.

Usa una estrategia off-policy y la ecuación de Bellman para modelar la función Q. Para la policy, aprovecha la función Q modelada como factor de relevancia en la misma.

Paper: Silver et al, 2014

(Continuous Control with Deep Reinforcement Learning, Lillicrap et al. 2016)

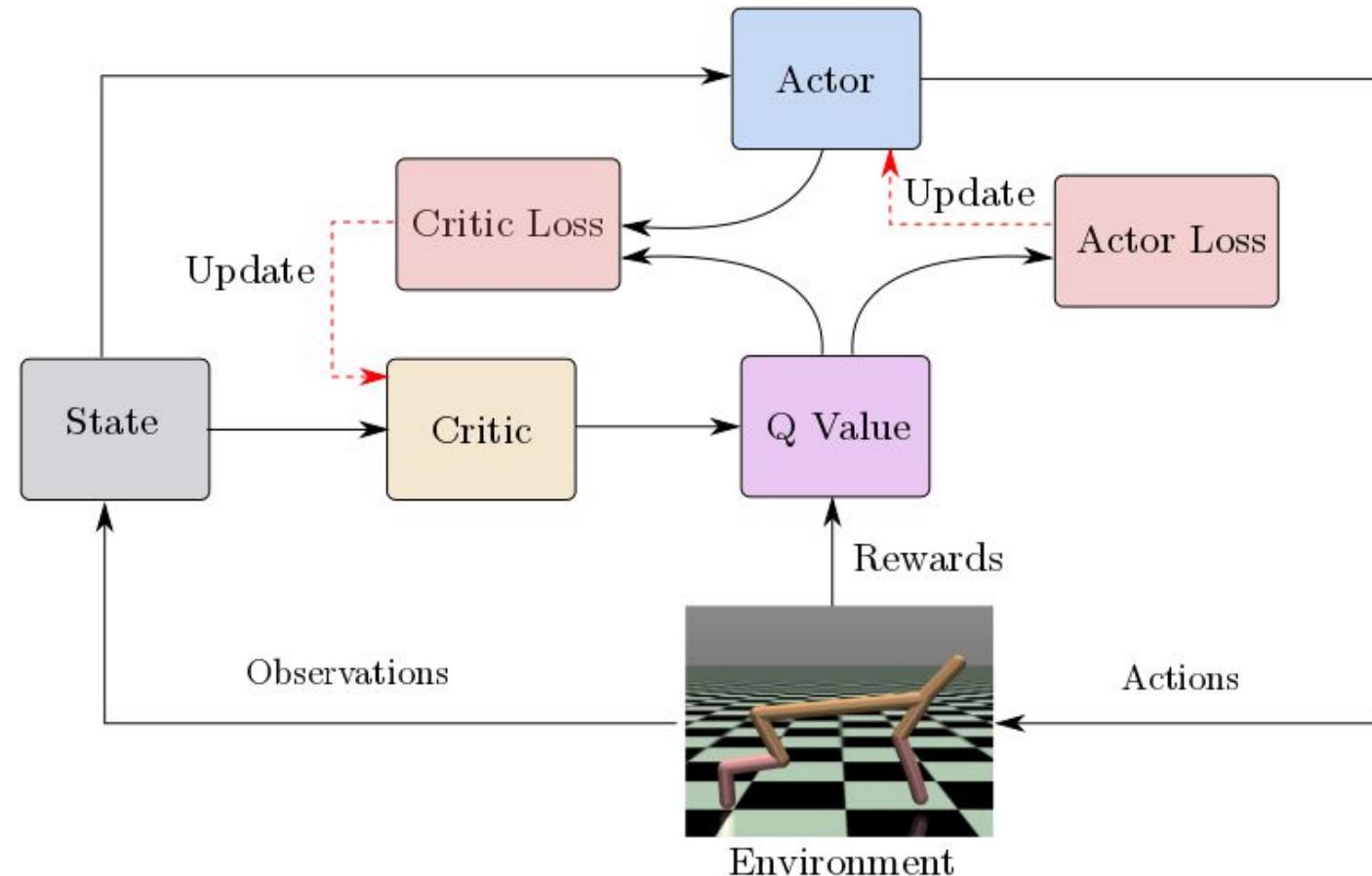
- DDPG es un algoritmo off-policy
- DDPG sólo se puede utilizar en entornos con espacios de acciones continuos
- DDPG se puede ver como la “forma de aplicar Q-learning en espacios de acciones continuos”.

Características principales

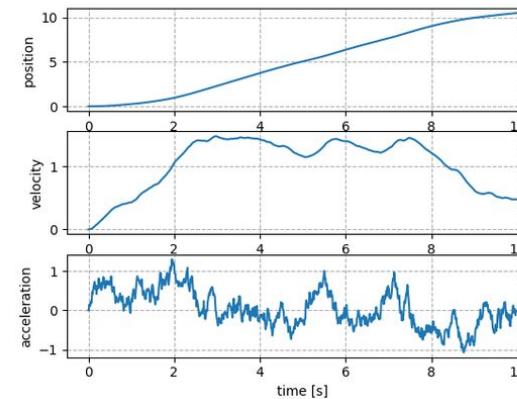
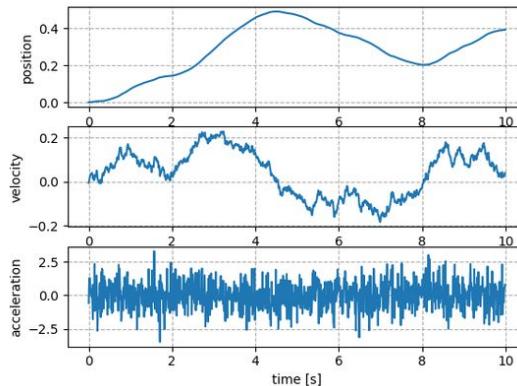
Algunas de las características (y decisiones) que se nos presentan con DDPG son:

- El espacio de acciones es continuo. Cómo podemos aplicar una estrategia seleccionando la acción que maximice la recompensa esperada en un espacio continuo?
- Al ser off-policy, se utilizará un replay buffer para almacenar la experiencia que se va acumulando.
- Se definirán cuatro redes neuronales diferentes: Dos redes para predicción (actor y critic networks) y dos redes target (actor y critic target networks).
- El proceso de exploración es diferente a los estudiados hasta ahora. Se llevará a cabo incluyendo una función de ruido en el momento de la selección de la acción.

Características principales



Características principales



El proceso de exploración tipo que encontraremos en el algoritmo será llevado a cabo utilizando el proceso de Ornstein-Uhlenbeck.

La idea de este proceso es utilizar un proceso que mantenga una correlación temporal para que la exploración sea más suave que con otras posibles distribuciones.

https://en.wikipedia.org/wiki/Ornstein%E2%80%93Uhlenbeck_process

<https://www.quora.com/Why-do-we-use-the-Ornstein-Uhlenbeck-Process-in-the-exploration-of-DDPG>

Algoritmo DDPG

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .

Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer R

for episode = 1, M **do**

 Initialize a random process \mathcal{N} for action exploration

 Receive initial observation state s_1

for t = 1, T **do**

 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise

 Execute action a_t and observe reward r_t and observe new state s_{t+1}

 Store transition (s_t, a_t, r_t, s_{t+1}) in R

 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

 Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

end for

end for

Proximal Policy Optimization (PPO)

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI

{joschu, filip, prafulla, alec, oleg}@openai.com

Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a “surrogate” objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

<https://spinningup.openai.com/en/latest/algorithms/ppo.html>

Sesión 6 – Espacios de acciones continuos: DDPG & PPO

La idea clave en la que se centra el algoritmo de PPO es:

Cómo podemos mejorar lo máximo posible la policy actual, usando la trayectoria recolectada, controlando que el aprendizaje no colapse?

Paper: Schulman et al. 2017

- Algoritmo basado en otro conocido como TRPO
- PPO es un algoritmo on-policy. Pertenece a la familia de Policy Gradient.
- PPO puede ser utilizado para entornos con espacios de acciones tanto discretos como continuos.
- Normalmente, las implementaciones que encontraremos de PPO utilizan una ejecución multiproceso para tareas de recolección de datos.

Características principales

Algunas de las características (y decisiones) que se nos presentan con PPO son:

- El espacio de acciones puede ser discreto o continuo
- Al igual que los otros algoritmos de su familia, sigue una filosofía on-policy. Nuevamente, tenemos que decidir el tamaño de la trayectoria para almacenar los datos que se utilicen para el entrenamiento. En este caso, tenemos que tener en cuenta que la ejecución multiproceso aumenta el volumen recolectado.
- La arquitectura de modelo utilizada es similar a Actor-Critic
- La exploración se realiza de la misma manera que los algoritmos Actor-Critic, utilizando una distribución de probabilidad aleatoria ponderada.

Características principales

La mayor diferencia aparece en la función de coste, PPO ofrece dos versiones diferentes:

- Una función de coste basada en **Kullback-Leibler divergence**
- Una función de coste basada en **Clipped surrogate objective**

Será la segunda opción la que trataremos en la asignatura.

Características principales

La primera diferencia que introduce esta función es la sustitución del logaritmo de la probabilidad de la acción por el ratio de la probabilidad de la acción con la policy actual en comparación con la versión de la policy anterior.

Este ratio será mayor que 1 cuando la acción es más probable con la nueva policy y estará entre 0 y 1 cuando la acción sea menos probable.

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t | s_t) \hat{A}_t \right] \quad L^{TRPO}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t | s_t)}{\pi_{\theta old}(a_t | s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

Características principales

Para estabilizar el proceso de aprendizaje (con probabilidades que sean muy muy grandes), PPO introduce dos límites para los casos en los que este ratio se dispare y, por tanto, los cambios en la policy sean muy drásticos.

Básicamente, junto al ratio calculado se define un rango (a partir de un nuevo hiperparámetro, con valor por defecto 0.1/0.2) para que la actualización de los pesos del modelo se lleve a cabo con el valor mínimo de este rango. Así aseguramos que la actualización se va haciendo de forma “suave” y en una buena dirección.

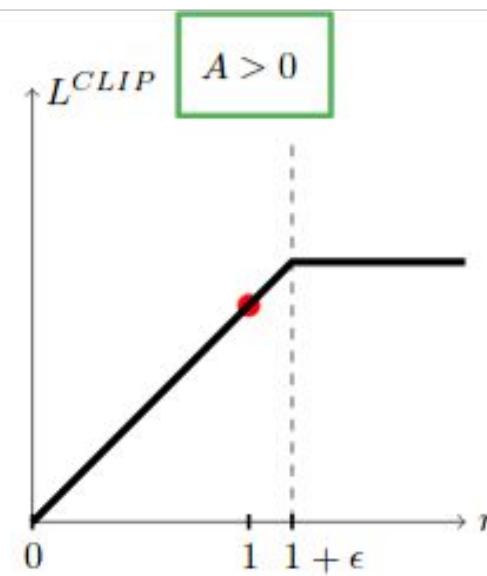
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(\overline{r_t(\theta) \hat{A}_t}, \overline{\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t} \right) \right]$$

min of the same objective from before and the clipped one

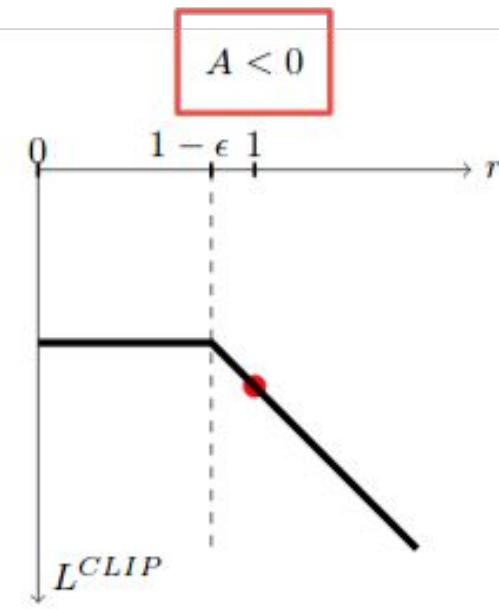
same objective same, but $r(\theta)$ is clipped
 from before between $(1 - \epsilon, 1 + \epsilon)$

Características principales

If the action was **good**....



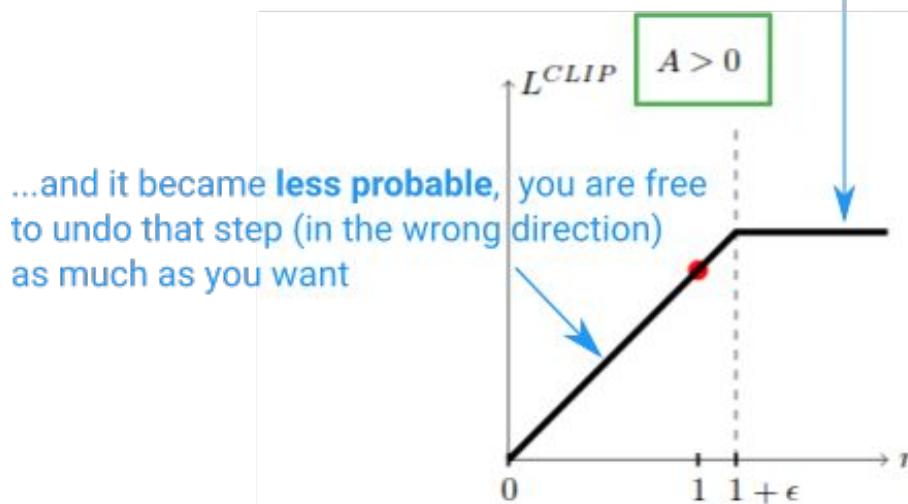
If the action was **bad**....



Características principales

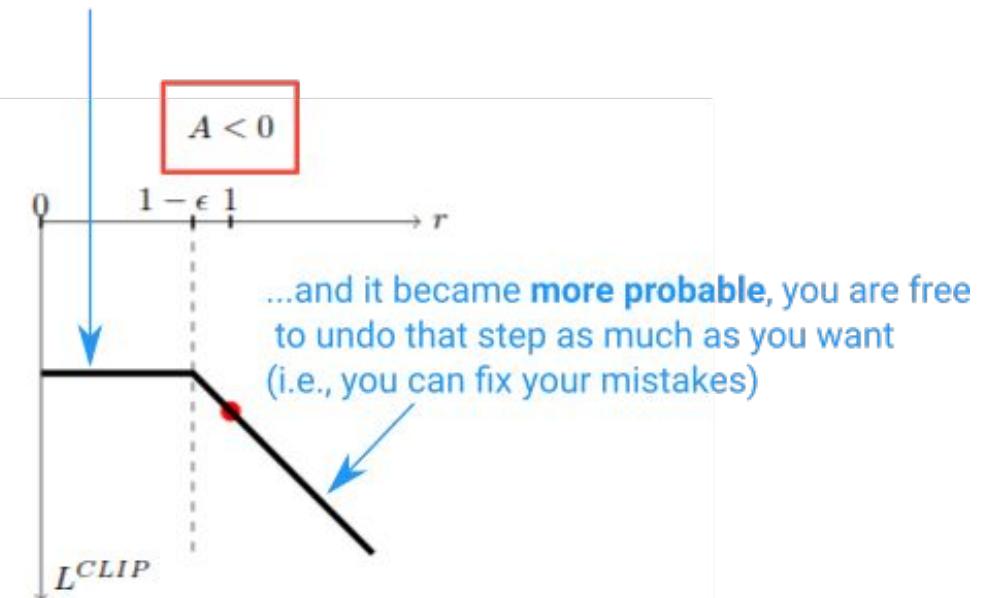
If the action was **good**....

...and it became **more probable** the last time you took a gradient step, don't keep updating it too far or else the policy might get worse



If the action was **bad**....

...and it became **less probable**, don't keep making it too much less probable or else the policy might get worse (i.e., don't step too far)



Algoritmo PPO

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_k}(a_t | s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

Alternativas a DDPG y PPO

Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja¹ Aurick Zhou¹ Pieter Abbeel¹ Sergey Levine¹

Abstract

Model-free deep reinforcement learning (RL) algorithms have been demonstrated on a range of challenging decision making and control tasks. However, these methods typically suffer from two major challenges: very high sample complexity and brittle convergence properties, which necessitate meticulous hyperparameter tuning. Both of these challenges severely limit the applicability of such methods to complex, real-world domains. In this paper, we propose soft actor-critic, an off-policy actor-critic deep RL algorithm based on the

of these methods in real-world domains has been hampered by two major challenges. First, model-free deep RL methods are notoriously expensive in terms of their sample complexity. Even relatively simple tasks can require millions of steps of data collection, and complex behaviors with high-dimensional observations might need substantially more. Second, these methods are often brittle with respect to their hyperparameters: learning rates, exploration constants, and other settings must be set carefully for different problem settings to achieve good results. Both of these challenges severely limit the applicability of model-free deep RL to real-world tasks.

Hindsight Experience Replay

Marcin Andrychowicz*, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong,
Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel†, Wojciech Zaremba†
OpenAI

Abstract

Dealing with sparse rewards is one of the biggest challenges in Reinforcement Learning (RL). We present a novel technique called *Hindsight Experience Replay* which allows sample-efficient learning from rewards which are sparse and binary and therefore avoid the need for complicated reward engineering. It can be combined with an arbitrary off-policy RL algorithm and may be seen as a form of implicit curriculum.

We demonstrate our approach on the task of manipulating objects with a robotic arm. In particular, we run experiments on three different tasks: pushing, sliding, and pick-and-place, in each case using only binary rewards indicating whether or not the task is completed. Our ablation studies show that Hindsight Experience Replay is a crucial ingredient which makes training possible in these challenging environments. We show that our policies trained on a physics simulation can be deployed on a physical robot and successfully complete the task. The video presenting our experiments is available at <https://goo.gl/SMrQnI>.

<https://arxiv.org/abs/1801.01290>

<https://arxiv.org/abs/1801.01290>

Bibliografía recomendada

- *Learning Dexterity*, OpenAI

<https://openai.com/blog/learning-dexterity/>

- *What is the way to understand Proximal Policy Optimization Algorithm in RL?*, Cyberman, A. Stackoverflow

<https://stackoverflow.com/questions/46422845/what-is-the-way-to-understand-proximal-policy-optimization-algorithm-in-rl>

08MIAR - Aprendizaje por refuerzo

Sesión 7 – Algoritmos basados en modelo

Curso 21/22



**Universidad
Internacional
de Valencia**

De:

 Planeta Formación y Universidades

Índice

Introducción

Model based: Modelo conocido vs Model desconocido

Model learning: Aspectos principales

Planning & Learning

Ejemplos de soluciones *Model Based*

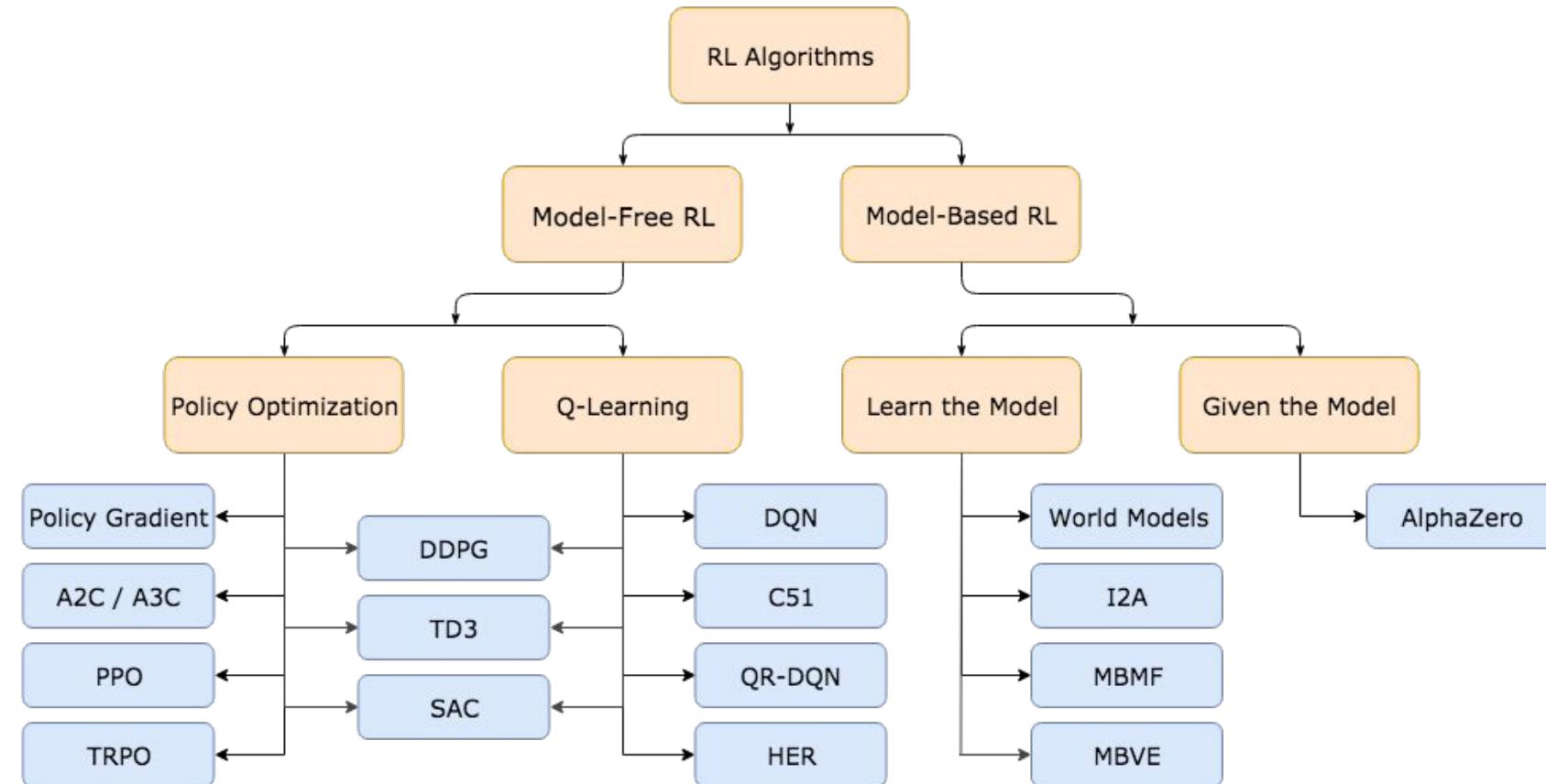
Conclusiones

Introducción

The next big step forward in AI will be systems that actually understand their worlds. The world is only accessed through the lens of experience, so to understand the world means to be able to predict and control your experience, your sense data, with some accuracy and flexibility. In other words, understanding means forming a predictive model of the world and using it to get what you want. This is model-based reinforcement learning.

Richard Sutton

Introducción



https://spinningup.openai.com/en/latest/_images/rl_algorithms_9_15.svg

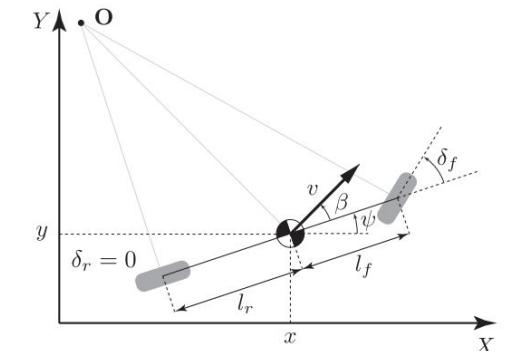
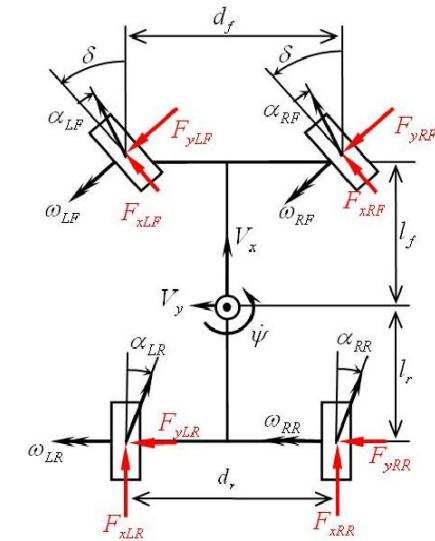
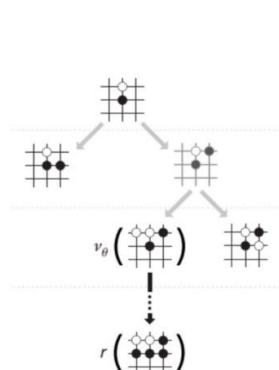
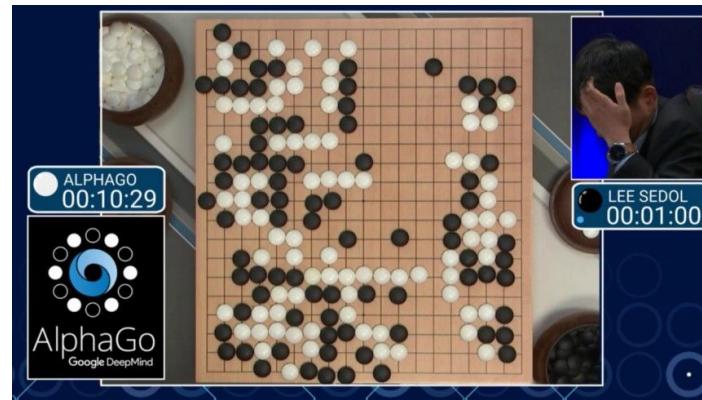
Introducción

La principal diferencia entre soluciones *model free* y *model based* es el conocimiento a priori de las dinámicas del entorno.

Si se conocen las dinámicas del entorno, se pueden estimar las transiciones que el agente puede ejecutar desde el estado actual para valorar cuál es la mejor acción a tomar.

Actualmente, uno de los retos principales aparece con entornos en los que el espacio de acciones es muy elevado y, por tanto, esta estimación es muy demandante en cuanto a recursos computacionales necesarios.

Introducción



Introducción

En esta asignatura nos centraremos en un tipo específico de implementación: acompañar el aprendizaje del modelo del agente con *metaheurísticas* u otras técnicas de optimización, que sirvan de apoyo al proceso de aprendizaje.

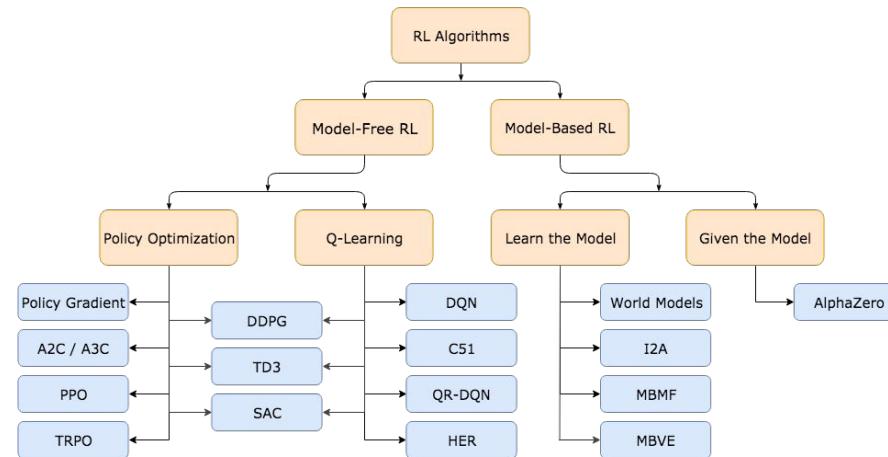
Dentro de las opciones que podemos encontrar, las más utilizadas son la búsqueda en árbol o algoritmos basados en población (genéticos). Todos ellos basados a su vez en simulaciones de Montecarlo para poder analizar los distintos caminos disponibles.

Introducción

Además, otro punto de vista de *Model based* es su conexión directa con problemas de control por computador. En este sentido, es común ver el modelo como un controlador y que el problema se transforme en “minimizar el error cometido” en vez de “maximizar la recompensa esperada”.

Por último, hay un aspecto muy importante a tener en cuenta en relación con la recompensa. Hasta ahora, la recompensa nos ha venido dada como valores que el entorno nos devuelve de manera arbitraria. En soluciones basadas en modelo es común conocer la estructura o función de recompensa, lo que complementa al uso de técnicas de optimización a la hora de estimar las acciones.

Model based: Modelo conocido vs Modelo desconocido



Cuando hablamos de soluciones basadas en modelo podemos encontrarnos en dos situaciones diferentes: que el modelo sea conocido o no.

Si el modelo es conocido, podremos aplicar directamente las técnicas de metaheurísticas vistas anteriormente.

Si el modelo es desconocido, tendremos que llevar a cabo un paso previo en la ejecución para aprender o aproximar de la mejor manera posible el modelo del entorno.

Model based: Modelo conocido vs Modelo desconocido

Como hemos comentado, si el modelo es conocido entonces podemos utilizar ese conocimiento para enriquecer el proceso de entrenamiento. Esta es la situación típica con metaheurísticas, planificación y otras técnicas de optimización. Este es también el tipo de ejecución en la que se basan soluciones como Alphago o Alphazero.

El elemento principal a tener en cuenta en este tipo de soluciones es la capacidad computacional necesaria para la implementación. Los ejemplos de Alphago y Alphazero muestran esta problemática, ya que aunque podamos estimar las transiciones del entorno de una manera precisa, es imposible poder llevar todas las estimaciones posibles en cada momento de la ejecución.

Model based: Modelo conocido vs Modelo desconocido

Si el modelo es desconocido, tendremos que llevar a cabo una fase previa de aprendizaje o aproximación. Lo común es utilizar un enfoque de aprendizaje supervisado/no supervisado para crear el modelo del entorno utilizando datos recolectados. Una vez tengamos esta aproximación, podemos utilizarla en el algoritmo de aprendizaje por refuerzo que deseemos.

A diferencia del uso de metaheurísticas, este tipo de solución Model-based tiene su base en algoritmos/procesos basados en gradientes, debido a la necesidad de optimizar la representación que se va obteniendo del entorno.

Model learning: Aspectos principales

Model-based Reinforcement Learning: A Survey.

Thomas M. Moerland^{1,2}, Joost Broekens², and Catholijn M. Jonker^{1,2}

¹ Interactive Intelligence, TU Delft, The Netherlands

² LIACS, Leiden University, The Netherlands

Model learning: Aspectos principales

El primer aspecto a analizar es el tipo de modelo que se va a estimar a partir del entorno. Podemos encontrar tres tipos principales:

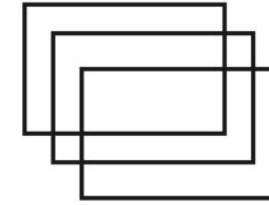
- 1) Modelo *forward* $(s_t, a_t) \rightarrow s_{t+1}$
- 2) Modelo *backward* $s_{t+1} \rightarrow (s_t, a_t)$
- 3) Modelo inverso $(s_t, s_{t+1}) \rightarrow a_t$

Model learning: Aspectos principales

Las observaciones parciales se refieren a cuando la información disponible en una observación no es completa. Es una situación muy típica que puede ocurrir por la propia naturaleza del problema. Algunas formas de suavizar esta situación son:



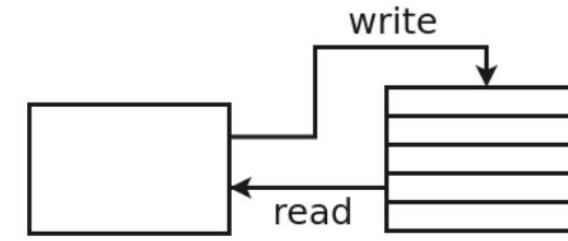
Single state



Window



Recurrency



Neural Turing Machine

Model learning: Aspectos principales

Uno de los puntos más críticos cuando se aproxima el modelo de un entorno es la abstracción de los estados. Debido a la alta carga de información de este tipo de problemas, el modelo del entorno siempre va a ser una representación del mismo.

Normalmente, esta representación va asociada a algún método o proceso de abstracción, desde una reducción de dimensionalidades hasta el uso de alguna arquitectura de Deep Learning.

Model learning: Aspectos principales

Otros aspectos analizados en el artículo de referencia son la incertidumbre, la aleatoriedad o la estacionalidad de los datos.

Incertidumbre y aleatoriedad son conceptos que están muy relacionados entre sí, ya que la incertidumbre trata la falta de información en los datos para aproximar un modelo preciso, mientras que la aleatoriedad trata la propia naturaleza estocástica del proceso que se está modelando. La aleatoriedad siempre va a estar presente mientras que la incertidumbre se podría disminuir teniendo más datos disponibles.

La estacionalidad se refiere a si cambian las funciones de transición o recompensa a lo largo del tiempo. Estos cambios, si no son percibidos por el agente, pueden conllevar a que la solución se desgaste con el paso del tiempo.

Model learning: Aspectos principales

Model-based Reinforcement Learning: A Survey.

Thomas M. Moerland^{1,2}, Joost Broekens², and Catholijn M. Jonker^{1,2}

¹ Interactive Intelligence, TU Delft, The Netherlands

² LIACS, Leiden University, The Netherlands

Planning & Learning

A la hora de aplicar planificación junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

- 1) En qué estado comenzamos la planificación?
- 2) Al ejecutar simulaciones, cuánta performance reservamos para la planificación?
- 3) Cómo llevamos a cabo la planificación?
- 4) Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?

Planning & Learning

A la hora de aplicar planificación junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

- 1) En qué estado comenzamos la planificación?
- 2) Al ejecutar simulaciones, cuánta performance reservamos para la planificación?
- 3) Cómo llevamos a cabo la planificación?
- 4) Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?

- Estado aleatorio
- Estado visitado
- Estado prioritario
- Estado actual

Planning & Learning

A la hora de aplicar planificación junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

- 1) En qué estado comenzamos la planificación?
- 2) Al ejecutar simulaciones, cuánta performance reservamos para la planificación?
- 3) Cómo llevamos a cabo la planificación?
- 4) Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?

- Después de cuántos steps comenzamos?
- Cuánto esfuerzo ejecutamos por iteración?
(AlphaGo Zero, 1MCTS, 1600 trazas de profundidad 200)

Planning & Learning

A la hora de aplicar planificación junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

- 1) En qué estado comenzamos la planificación?
- 2) Al ejecutar simulaciones, cuánta performance reservamos para la planificación?
- 3) Cómo llevamos a cabo la planificación?**
- 4) Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?

- De tipo discreto (árbol, tabla) o diferencial (modelos de transición y recompensa diferenciables)
- La dirección puede ser *forward* o *backward*
- Tenemos que definir los niveles de profundidad y anchura en la simulación

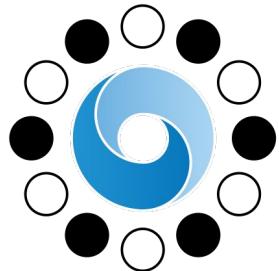
Planning & Learning

A la hora de aplicar planificación junto a nuestra solución de Aprendizaje por refuerzo se nos presentan una serie de preguntas:

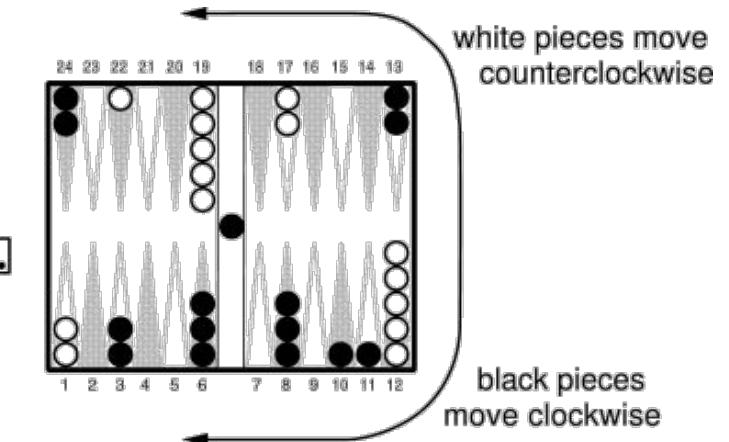
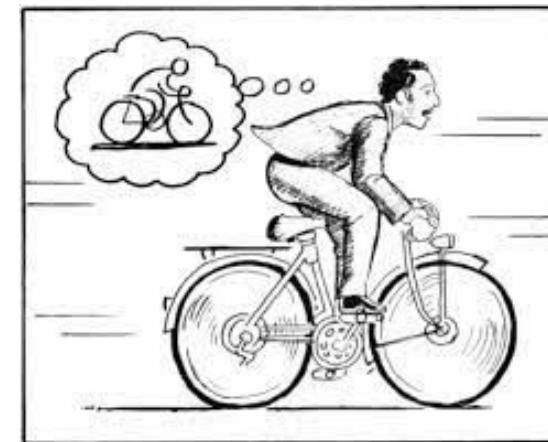
- 1) En qué estado comenzamos la planificación?
- 2) Al ejecutar simulaciones, cuánta performance reservamos para la planificación?
- 3) Cómo llevamos a cabo la planificación?
- 4) Cómo relacionamos la planificación con el proceso de aprendizaje y la toma de acciones?

- Usando la policy y el value para llevar a cabo la planificación
- Usando el resultado de la planificación para actualizar la policy y el value
- Usando directamente la planificación para seleccionar una acción

Ejemplos de soluciones Model Based

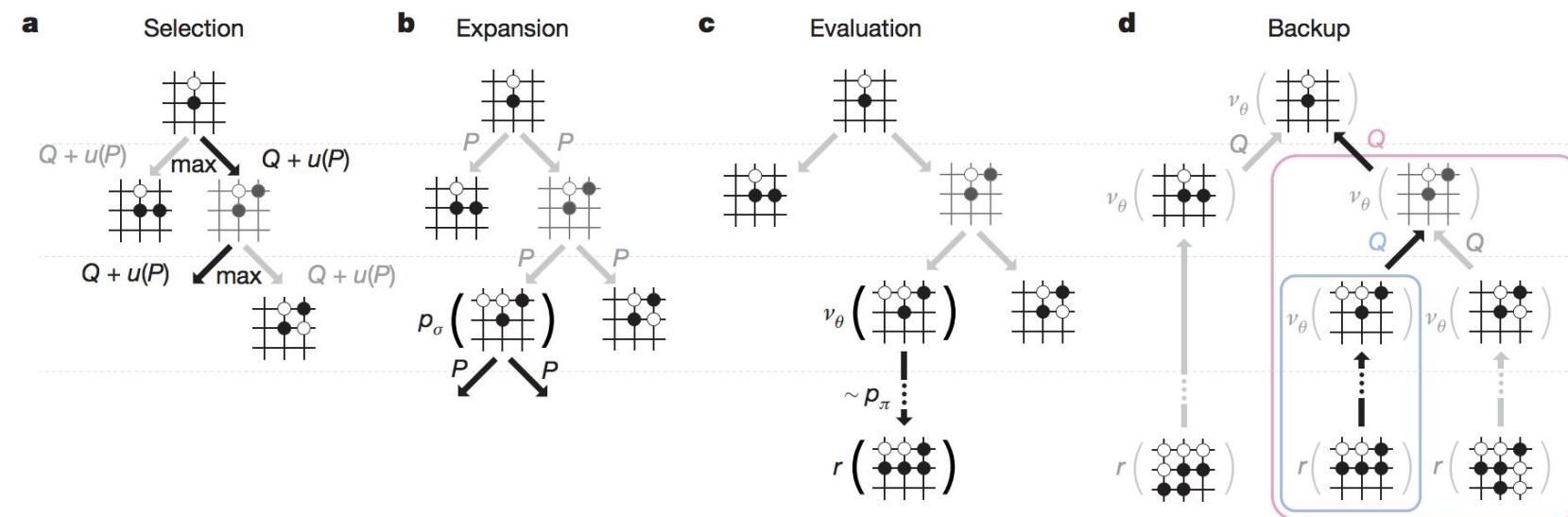


AlphaGo



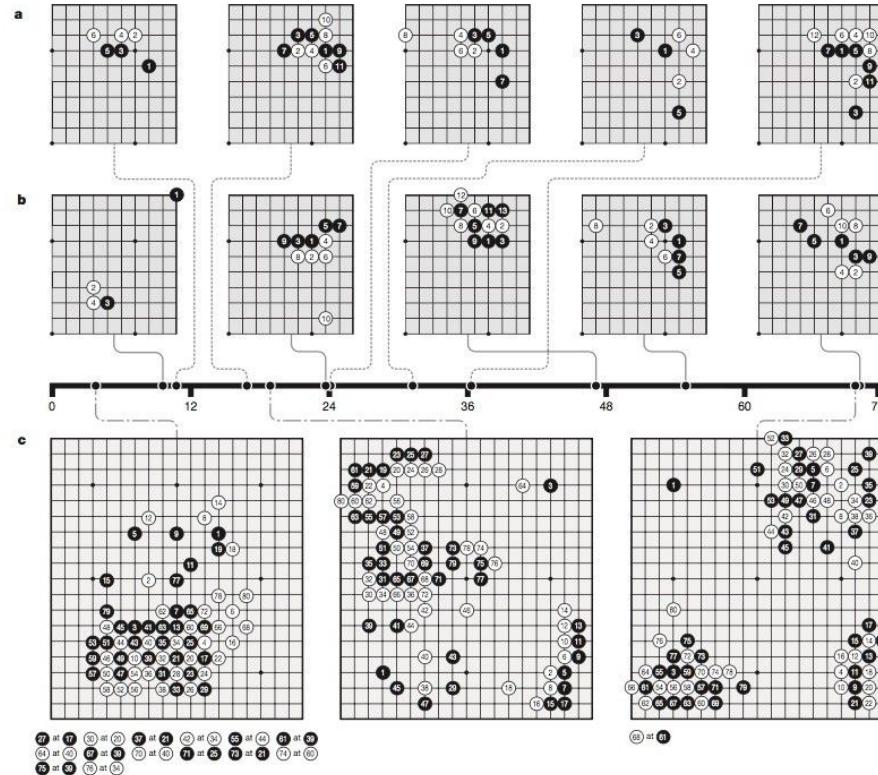
https://es.m.wikipedia.org/wiki/Archivo:Alphago_logo_Reversed.svg
<http://incompleteideas.net/book/ebook/node108.html>
<https://arxiv.org/pdf/1803.10122.pdf>

Ejemplos de soluciones Model Based



https://es.m.wikipedia.org/wiki/Archivo:Alphago_logo_Reversed.svg
<https://deepmind.com/research/case-studies/alphago-the-story-so-far>

Ejemplos de soluciones Model Based



En su siguiente versión, Alphago Zero, el entrenamiento de la solución se produce mediante *self-play*, sin incluir la primera fase de aprendizaje supervisado.

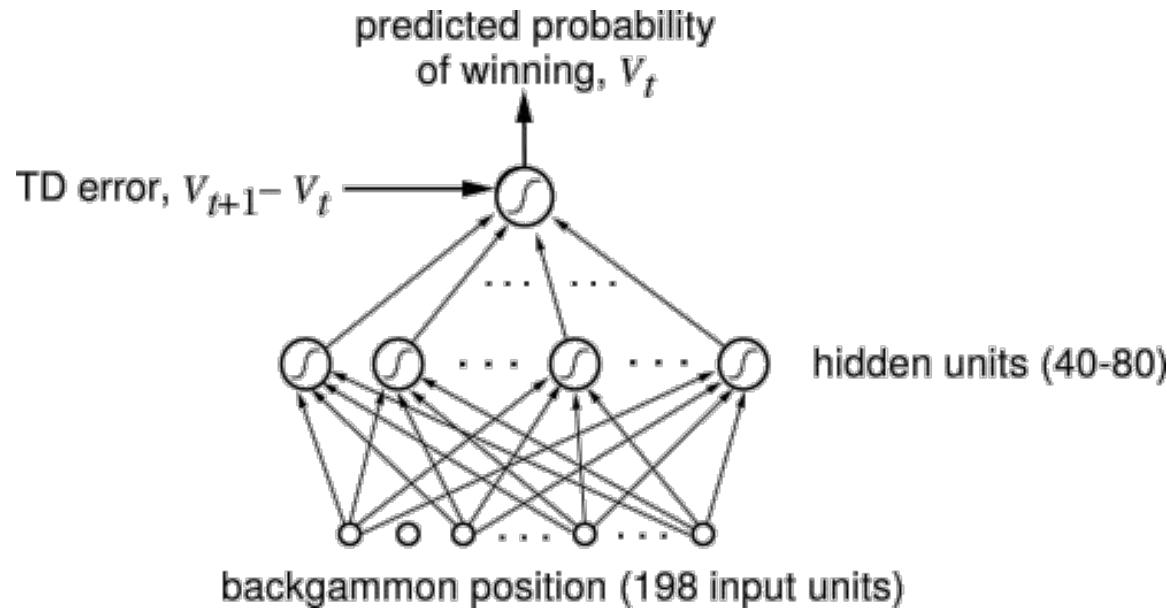
Como versión siguiente, Deepmind desarrolla AlphaZero, para generalizar a los juegos del ajedrez y Shogi además del Go.

Como curiosidad, se estima que Alphago Zero costó unos 35M\$ en recursos computacionales*

<https://www.xataka.com/robotica-e-ia/la-nueva-alphago-esta-un-paso-mas-cerca-de-la-singularidad-aprende-de-si-misma-y-deja-en-ridiculo-a-la-anterior>

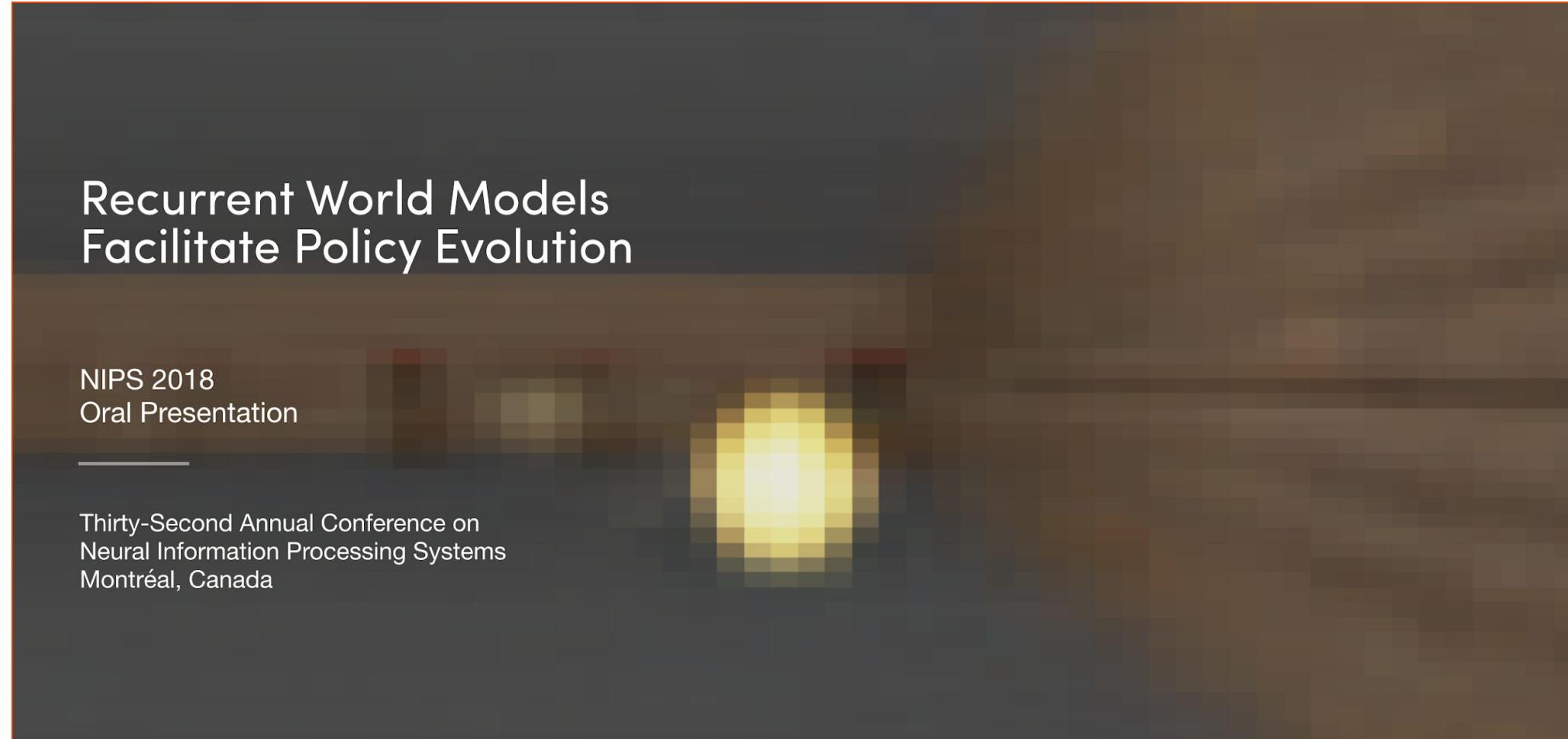
*<https://www.yuzeh.com/data/agz-cost.html>

Ejemplos de soluciones Model Based



- TD-Gammon, desarrollado por Gerry Tesauro (Tesauro, 1992, 1994, 1995).
- A diferencia de otras soluciones de la época, utiliza como datos de entrada los datos en crudo del tablero.
- Otro elemento diferencial es la función de evaluación utilizada, basada en diferencias temporales para optimizar la red neuronal con la que se aproxima la probabilidad de ganar.

Ejemplos de soluciones Model Based



<https://worldmodels.github.io/>

Sesión 7 – Algoritmos basados en modelo

Conclusiones

- 1) La principal diferencia entre model-free y model-based es que con model-based conocemos las dinámicas del entorno, por ejemplo en cuanto a transiciones y funciones de recompensa.
- 2) Dentro de model-based, podemos encontrarnos con dos situaciones diferentes: que el modelo del entorno sea conocido o que sea desconocido y, por tanto, tengamos que aproximarla
- 3) Dentro de las metaheurísticas usadas en estas soluciones, las implementaciones más comunes combinan técnicas de planificación junto con simulaciones de Montecarlo.

Bibliografía recomendada

- “*Model-Based Reinforcement Learning: A survey*”, Moerland, T. et al
<https://arxiv.org/pdf/2006.16712.pdf>

- “*AlphaGo*”, Google Deepmind
<https://deepmind.com/research/case-studies/alphago-the-story-so-far>

- “*World Models*”, David Ha, Jurgen Schmidhuber
<https://arxiv.org/abs/1803.10122>
<https://worldmodels.github.io/>



viu

Universidad
Internacional
de Valencia

universidadviu.com

De:
 Planeta Formación y Universidades