
Aprendizaje por refuerzo

PID_00279459

Samir Kanaan Izquierdo
Marc Maceira Duch
Carles Ventura Royo

Tiempo mínimo de dedicación recomendado: 3 horas



Samir Kanaan Izquierdo**Marc Maceira Duch****Carles Ventura Royo**

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por los profesores: Marc Maceira Duch, Carles Ventura Royo

Primera edición: febrero 2021

© de esta edición, Fundació Universitat Oberta de Catalunya (FUOC)

Av. Tibidabo, 39-43, 08035 Barcelona

Autoría: Samir Kanaan Izquierdo, Marc Maceira Duch, Carles Ventura Royo

Producción: FUOC



Los textos e imágenes publicados en esta obra están sujetos –excepto que se indique lo contrario– a una licencia Creative Commons de tipo Reconocimiento-NoComercial-SinObraDerivada (BY-NC-ND) v.3.0. Se puede copiar, distribuir y transmitir la obra públicamente siempre que se cite el autor y la fuente (Fundació per a la Universitat Oberta de Catalunya), no se haga un uso comercial y ni obra derivada de la misma. La licencia completa se puede consultar en: <http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.es>

Índice

Introducción	5
Objetivos	6
1 Introducción al aprendizaje por refuerzo	7
1.1 Un vistazo al aprendizaje automático	7
1.2 Otra manera de aprender	8
1.3 El aprendizaje por refuerzo	9
1.4 Elementos del aprendizaje por refuerzo	11
1.4.1 Estados	12
1.4.2 Acciones	13
1.4.3 Recompensas	14
1.4.4 Políticas	16
1.5 Aplicaciones	17
2 Tipología de métodos	20
2.1 Modelo del entorno	20
2.1.1 Métodos basados en modelos	20
2.1.2 Métodos sin modelo	21
2.2 Política actual u óptima	21
2.2.1 Métodos de política actual	21
2.2.2 Métodos de política óptima	22
2.3 Política o valor	22
2.3.1 Métodos de gradiente de política	22
2.3.2 Métodos basados en valores-estados	23
2.3.3 Métodos basados en valores-acciones	23
2.3.4 Aproximaciones a Q mediante funciones	25
2.4 Familias de métodos	25
3 Aprendizaje-Q	26
3.1 Antecedentes	26
3.1.1 Trayectorias o episodios	26
3.1.2 Cálculo de V o Q	26
3.1.3 Ecuación de Bellman	27
3.1.4 Aprendizaje por diferencia temporal	27
3.1.5 Método SARSA	28
3.2 Q-learning	28
3.2.1 Ejemplo: laberinto	29
4 Redes Q profundas (DQN)	31
4.1 Redes neuronales profundas y Q-learning	31

4.2	<i>Deep Q-network</i>	33
Resumen		36
Bibliografía		37

Introducción

El aprendizaje por refuerzo permite entrenar a un sistema (llamado agente) sin datos de entrenamiento, simplemente dejándolo interactuar con su entorno y dándole recompensas como única orientación. Este paradigma de aprendizaje permite resolver problemas en los que el aprendizaje supervisado y el no supervisado no son aplicables fácilmente o bien no dan resultados satisfactorios. Veremos los conceptos fundamentales del aprendizaje por refuerzo y las técnicas más avanzadas que utilizan redes neuronales para modelar los agentes.

Objetivos

Este módulo tiene los objetivos siguientes:

- 1.** Conocer los conceptos fundamentales del aprendizaje por refuerzo.
- 2.** Conocer los elementos que componen los métodos del aprendizaje por refuerzo y las posibles variantes.
- 3.** Conocer los algoritmos de aprendizaje por refuerzo básicos y los más avanzados.

1. Introducción al aprendizaje por refuerzo

1.1 Un vistazo al aprendizaje automático

En el área del aprendizaje automático los estilos de aprendizaje vistos hasta ahora se clasifican en dos grandes tipos (aprendizaje supervisado y aprendizaje no supervisado), que dependen de las técnicas utilizadas y los datos disponibles.

En el aprendizaje **supervisado** hay un conjunto de datos etiquetados, es decir, con un valor (numérico o categórico) que es el objetivo de la tarea de aprendizaje: por ejemplo, el objeto que muestra cada fotografía de un conjunto de imágenes, o el precio de la vivienda en función de variables socioeconómicas. Los métodos de aprendizaje deben aprender a relacionar una serie de entradas (imagen, variables socioeconómicas, etc.) con un valor de salida (tipo de objeto, precio).

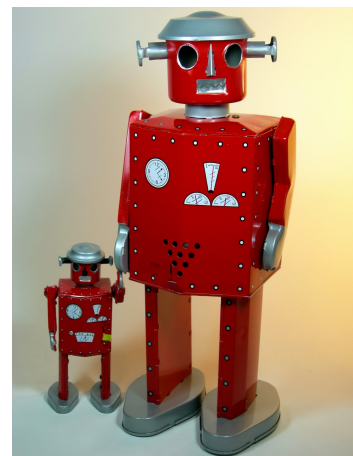
Por otro lado, en el aprendizaje **no supervisado** se dispone de datos pero no hay una «solución», sino que la tarea de los modelos es encontrar relaciones o grupos en los datos, o detectar datos claramente diferentes de los del entrenamiento: por ejemplo, encontrar usuarios con preferencias parecidas a las de un servicio de vídeo para reproducción en continuo.

En cualquiera de las dos situaciones anteriores hay un conjunto de datos de entrenamiento que permite ajustar los parámetros de un **modelo**, que entonces se podrá utilizar para procesar nuevos datos.

Pensemos, no obstante, en una situación diferente: debemos entrenar a un robot para que recoja hojas del suelo y las vierta en un contenedor. ¿Cómo podemos obtener un conjunto de datos para entrenar a este robot y que haga el trabajo? Podríamos diseñar un conjunto de datos con diferentes posiciones del robot y las hojas, e indicar manualmente, para cada pose del robot, si es correcta o no, o si hace falta que gire, coja una hoja o las vierta.

Preparar este conjunto de datos sería evidentemente un trabajo de anotación inmensa, inasumible si tenemos en cuenta que es necesario que un humano prepare los datos, porque habría que anotar todas las posiciones y direcciones posibles del robot y hojas en muchas posiciones diferentes.

Sin embargo, seguramente sería un trabajazo inútil porque se trata de un problema inherentemente dinámico, en el sentido de que pueden aparecer hojas en todas las posiciones y cantidades, y por lo tanto por muchos datos de en-



El robot jardinero
Fuente: <https://bit.ly/399vdvZ>.

trenamiento que hubiera, en la práctica el robot siempre se encontraría con otras situaciones y posiblemente no sabría qué debería hacer.

Por otro lado, si el conjunto de datos no es supervisado (acción correcta o incorrecta), el robot no tendrá ninguna orientación de lo que hay que hacer en cada momento, y en todo caso lo que podrá decirnos finalmente es, por ejemplo, cuáles son las zonas con más concentración de hojas.

En concreto, hay unos cuantos motivos para desaconsejar utilizar métodos de aprendizaje «clásicos» (supervisado o no supervisado) en este tipo de problemas:

- Coste muy elevado o imposibilidad de generar un conjunto de datos supervisado.
- Dificultad para definir la anotación o etiqueta en cada posible situación.
- Entornos dinámicos y no deterministas: en el ejemplo, las hojas están en diferentes posiciones y cantidades cada día. Además, puede haber personas, animales u otros obstáculos imprevistos y en movimiento.
- Incertidumbre: el robot no tiene un conocimiento absoluto de todo el entorno, solo de una parte, que detecta con sus cámaras o sensores.

1.2 Otra manera de aprender

Entonces, ¿cómo podemos plantear este tipo de problemas? Si nos paramos un momento a pensar, situaciones de este tipo son las que nos encontramos los humanos cuando aprendemos a resolver una nueva tarea: un bebé aprende a andar, una niña a jugar a fútbol, un hombre a barrer, etc. ¿Cómo llevamos a cabo este aprendizaje? ¡Merece la pena fijarse en ello, porque después de todo no lo hacemos tan mal!

La primera idea es que, aunque también aprendemos observando, aprendemos principalmente probando, primero equivocándonos y después haciéndolo mejor poco a poco. Es lo que se denomina proceso de **ensayo y error**: aprendemos de las equivocaciones (para no volver a cometerlas) y de los éxitos (para hacerlo del mismo modo).

La segunda idea es que este aprendizaje no es instantáneo: hay que **repetir** una acción muchas veces para asimilarla completamente.

La tercera idea es que la acción más adecuada **depende de la situación** que se dé en cada momento. Si aprendemos a ir en bicicleta, a veces hará falta pedalear y otras frenar, según la velocidad, los obstáculos, la carretera, etc. Es

Imitación

La equivalencia a aprender observando a un experto recibe el nombre de aprendizaje por imitación (*imitation learning*). Es una variante del aprendizaje por refuerzo.

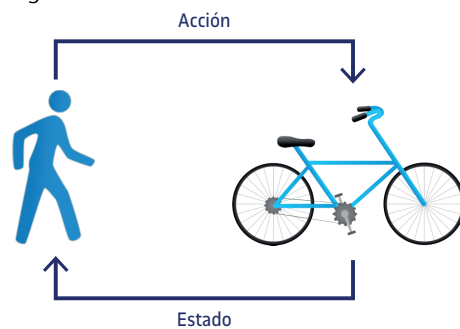
decir, no solamente hay que aprender acciones, sino asociar cada situación a la mejor acción o respuesta para esta situación.

Resumiendo, en muchas tareas, los humanos aprendemos:

- mediante ensayo y error;
- haciendo muchas repeticiones, y
- de modo que la acción más adecuada depende de la situación concreta.

Un proceso como este (por ejemplo, ir en bicicleta) se puede visualizar como un bucle donde, por un lado, la ciclista actúa (pedalea, frena, se inclina, etc.) y, por otro lado, la bicicleta y otros elementos (carretera, viento, etc.) responden a sus acciones pasando a un estado diferente (más velocidad, curvas, etc.). Entonces la ciclista reacciona con nuevas acciones, posiblemente ajustándose a las situaciones o estados que va encontrando hasta el final de la actividad. La figura 1 muestra esta idea de manera intuitiva.

Figura 1. Bucle de acción-reacción ciclista-bicicleta



Estas ideas son la esencia de lo que se denomina **aprendizaje por refuerzo**.* En el subapartado siguiente veremos la definición del aprendizaje por refuerzo y sus componentes principales.

* En inglés, *reinforcement learning* (RL)

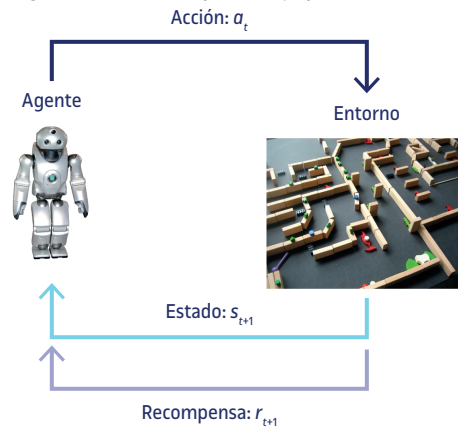
1.3 El aprendizaje por refuerzo

El aprendizaje por refuerzo es una rama del aprendizaje automático, y su característica principal es que no hay unos datos para entrenar a un modelo, sino que hay un entorno (real o virtual) con el cual se puede interactuar y que puede cambiar de estado por causa de estas interacciones, por el paso del tiempo o porque hay otras entidades autónomas que operan en este entorno.

Con cada acción que se ejecuta en el entorno, se produce un cambio de estado y, para guiar el aprendizaje, a menudo también se recibe un valor que indica que valora la acción hecha como buena o no respecto al logro de la tarea, lo que se denomina **recompensa**.

Este proceso, de manera equivalente al aprendizaje humano, requiere muchas interacciones para llegar a un modelo que actúe de la manera deseada. Por lo tanto, se establece un bucle de acción y reacción como sucedía con el sistema ciclista-bicicleta. La figura 2 muestra este bucle y sus elementos constituyentes.

Figura 2. Bucle del aprendizaje por refuerzo



Los elementos principales de un sistema de aprendizaje por refuerzo son:

- **Entorno:** el lugar, físico o virtual, donde se produce el aprendizaje, junto con todos los elementos pasivos o activos que intervienen en él.

Ejemplos

- Circulación en bicicleta: bicicleta, carretera, obstáculos, otros vehículos y peatones.
 - Recogida de hojas del suelo: lugar físico, árboles, hojas, contenedores.
 - Compraventa de acciones: conjunto de valores, otros inversores, otros factores que influyan en ello.
 - Dosificación automática de insulina a diabéticos: datos de glucosa en sangre del paciente.
 - Optimización de la producción: fábrica, máquinas, trabajadores, materiales, proveedores, clientes.
- **Agente:** entidad que interactúa con el entorno y que es el sujeto del aprendizaje. En los ejemplos anteriores serían el ciclista, el robot, el software de inversiones, el aparato dosificador de insulina y la planificación de la producción.
 - **Acción (a):** cada una de las interacciones que el agente realiza con el entorno. Ejemplos: pedalear, girar hacia la izquierda, comprar, inyectar insulina, etc.
 - **Estado (s):** en cada instante de tiempo t el entorno, como por ejemplo la velocidad de la bicicleta, el nivel de azúcar en sangre, la cotización de las acciones, etc., está en un estado o situación determinada. La recogida de todas las variables relevantes para el problema es el estado. A menudo, el estado incluye variables del agente, como por ejemplo el nivel de carga del robot, el peso de la ciclista, la cantidad de insulina disponible en el depósito del aparato, etc.

Observación

A excepción de entornos relativamente sencillos como por ejemplo juegos de mesa, a menudo el agente no puede conocer el estado completo de todo el entorno, sino solamente el que percibe con sus sensores. Entonces le llega solo una parte del estado, lo que se denomina **observación**.

- **Recompensa (r):** para orientar al agente y que aprenda a resolver la tarea que queremos, es necesario que reciba algún tipo de orientación, es decir, que sepa si ha hecho la acción correcta o, por el contrario, se ha equivocado, y en otra ocasión hará otra cosa. La recompensa suele ser un valor escalar en respuesta a cada acción, recompensa que puede ser positiva o negativa, es decir, un premio o una penalización.

A menudo, los problemas no tienen una recompensa explícita y hay que añadirla para que el aprendizaje sea posible.

- **Tiempo (t):** como veis, todo este proceso se desarrolla a lo largo del tiempo. Asumiremos simplemente que el tiempo es discreto, es decir, que las acciones se producen en intervalos de tiempo fijos, designados como t , $t + 1$, $t + 2$, ... La acción del instante t , combinada con el estado en t , produce un nuevo estado en $t + 1$ y también devuelve una recompensa a $t + 1$.
- **Tarea:** aunque no aparece en la figura 2 explícitamente, el objetivo final de entrenar a un agente es que sea capaz de resolver una tarea determinada que tenga interés económico, científico, clínico u otros. Hay que tener esto muy presente, especialmente en cuanto al diseño de las recompensas, porque un diseño inapropiado puede hacer que el agente no aprenda a resolver la tarea que queremos.

En el aprendizaje por refuerzo se quiere entrenar a un agente para que resuelva una tarea en un entorno determinado eligiendo entre un conjunto de acciones y aprendiendo mediante interacciones con el entorno y las recompensas obtenidas.

Los criterios que un agente utiliza para decidir qué acción hacer en cada momento reciben el nombre de **política*** (y se representa con la letra π). Durante el entrenamiento un agente que aprende con RL ha de ir mejorando su política con el fin de que sea tan efectiva como sea posible para la tarea que debe llevar a cabo.

Hay que tener en cuenta que, en general, al agente no se le dan las reglas, estrategias o trucos de la tarea o entorno, sino que durante el proceso de RL ha de aprender solo con las interacciones y recompensas que recibe.

1.4 Elementos del aprendizaje por refuerzo

En el subapartado anterior hemos visto cuáles son los elementos principales del aprendizaje por refuerzo. Hay muchas variantes de todos estos elementos,



Los agentes de RL son especialistas en encontrar «agujeros» en el sistema de recompensas. Por ejemplo, un agente que juega a un juego de plataformas puede pasar un tiempo infinito saltando barriles y no avanzar nunca porque así obtiene puntos infinitos sin riesgo.

Fuente: <https://bit.ly/2J3zO8i>.

* En inglés, *policy*

dependiendo tanto del problema que se quiere resolver como del método utilizado para realizar el aprendizaje. En este subapartado, estudiaremos con más detalle algunos para, primero, caracterizar bien cada problema y, segundo, empezar a entender qué aproximaciones son más adecuadas en cada caso.

1.4.1 Estados

S es el conjunto de todos los estados posibles s_i de un problema. Dependiendo del tipo de problema, el **tamaño** de S , es decir, el número de estados posibles, puede ser muy diferente.

Desde el punto de vista del aprendizaje, solo nos interesa diferenciar como estado aquello que cambia y que hace diferente el problema. Por ejemplo, si tenemos un laberinto donde todas las paredes son fijas, el laberinto no cambia nunca por sí mismo, y lo único que cambia es la posición del agente. Por lo tanto, S será el conjunto de todas las posiciones posibles del agente dentro del laberinto. Eso sí, nuestro agente solo aprenderá a resolver este laberinto concreto.

Por otro lado, si en el laberinto hubiera un monstruo que hay que evitar, S contendría todas las combinaciones de posición del agente y posición del monstruo. Empezamos a ver que, a medida que añadimos elementos al problema, el número de estados crece rápidamente porque a menudo son todas las combinaciones posibles de todos los elementos. La tabla 1 muestra el tamaño del espacio de soluciones de algunos juegos conocidos junto con otros valores para comparar.

Tabla 1. Tamaño de algunos conjuntos conocidos

Juego o entorno	Número de elementos
Tres en raya	255.168 estados
Internet (2018)	$18 * 10^{21}$ bytes
Universo	10^{80} átomos (estimación)
Ajedrez	10^{120} estados posibles
Go	$2.08 * 10^{170}$ estados válidos

Como veremos cuando estudiemos los métodos de RL, el tamaño del espacio de estados de un problema condiciona mucho las estrategias que se pueden seguir para conseguir el aprendizaje. A medida que se añaden variables al problema, el espacio de estados crece exponencialmente.

Otra característica importante de los estados de un problema es el tipo de datos o variables que lo representan. Principalmente, se distingue entre variables **discretas**, por ejemplo, la posición de una ficha en un tablero de ajedrez, y variables **continuas**, como por ejemplo la posición de un peatón que cruza una calle. La representación de un problema puede incluir variables de ambos tipos.



Un laberinto con el agente representado con un círculo.

Fuente: <https://bit.ly/33c3Cq6>.

Maldición de la dimensionalidad

El hecho de que las combinaciones posibles crezcan exponencialmente con el número de variables recibe el nombre de **maldición de la dimensionalidad** (*curse of dimensionality*).

Además del estado real del entorno con el que trabajamos, hay que tener en cuenta qué información llega al agente. Imaginemos una cadena de montaje donde dos robots se tienen que coordinar para llevar a cabo una tarea: un robot toma dos piezas metálicas y el otro las tiene que soldar. El robot soldador tiene que esperar a que el otro coja las piezas para empezar la soldadura. ¿Cómo lo sabe? Una manera es añadiendo una cámara al robot soldador para que empiece a soldar cuando vea que están las piezas preparadas; la otra es que el robot que toma las piezas avise al robot soldador de que las piezas están listas y le envíe las coordenadas. Aunque la tarea es la misma y la información que recibe el robot tiene la misma finalidad, son muy diferentes en cuanto al formato y la complejidad de procesamiento. Estas dos maneras de recibir información del entorno se denominan **sensorial** (se percibe con los sentidos) y **simbólica** (se reciben valores que representan posiciones, estados, etc.). También puede haber sistemas híbridos, con información sensorial y simbólica a la vez, como sucede con los vehículos autónomos, que reciben información de cámaras y radares pero también valores simbólicos extraídos de mapas, otros vehículos, etc.

Un último aspecto muy importante es si los agentes perciben el **estado completo** del entorno o solo una parte. Un agente que juega al ajedrez puede ver todo el tablero y las piezas, y entonces tiene un conocimiento completo del estado; en cambio, un robot con una cámara solo ve lo que tiene delante, dentro de su cono de visión, y por lo tanto no percibe el estado completo sino solo una parte. Cuando esto sucede, se dice que el agente recibe una **observación** en vez del estado completo. Una buena parte de los sistemas reales (es decir, no juegos o sistemas simulados) reciben observaciones parciales del entorno.

Características del espacio de estados: tamaño, tipo de variables (continuas o discretas), manera de recibirlas el agente (sensorial, simbólico o híbrida), estado completo o incompleto (observaciones).

1.4.2 Acciones

Otro elemento fundamental de un sistema RL son las acciones que puede realizar el agente para interactuar con el entorno y completar su tarea. Un agente que juegue automáticamente a un juego de plataformas en 2D, por ejemplo, simplemente se tendrá que mover o saltar, o las dos cosas a la vez; un agente que juega al ajedrez tiene que mover todas las fichas (de acuerdo con las reglas); un piloto automático para un dron debe regular la potencia de cada uno de los motores.

Cuantas más acciones pueda realizar un agente, más complejo se vuelve el aprendizaje, como veremos cuando estudiemos los métodos de aprendizaje. El

conjunto de todas las acciones que puede llevar a cabo un agente se denomina A , y a_i es la acción que ejecuta en la iteración i .

Las acciones se clasifican en dos tipos independientemente del número: acciones **discretas**, que se ejecutan o no (saltar o no saltar), y acciones **continuas**, que se pueden ejecutar con diferentes intensidades (potencia de los motores del dron).

La mayoría de los métodos de RL están diseñados para ejecutar acciones discretas.

1.4.3 Recompensas

Las recompensas son los valores que reciben los agentes como respuesta a cada acción que hacen, junto con el nuevo estado del sistema, y son las únicas orientaciones que los agentes tienen para aprender. El conjunto de las recompensas que un agente puede recibir se designa con la letra R , y la recompensa recibida en la iteración i se escribe r_i .

Son un elemento central del RL; de hecho, cualquier problema de RL se puede formular como el objetivo de crear un agente que maximice la recompensa acumulada G hasta completar la tarea (o bien agotar tiempo, energía u otros recursos que hagan que la tarea falle). Esta recompensa acumulada es dada por la expresión:

$$G(k) = \sum_{i=0}^k r_i \quad (1)$$

donde k es el número de pasos o iteraciones ejecutados. El agente ideal debe seleccionar la secuencia de acciones $a_0, a_1, a_2, \dots, a_k$ que maximicen $G(k)$.

Nos podemos encontrar diferentes estilos de recompensa según los aspectos siguientes:

- **Origen:** las recompensas son **intrínsecas** o **implícitas** si el entorno ya las proporciona, como suele pasar en los juegos. En caso contrario, se trata de recompensas **extrínsecas**, que hay que añadir cuando se diseña el sistema de aprendizaje. A veces hay recompensas implícitas, pero hay que añadir otras auxiliares para facilitar la resolución de la tarea.

Ejemplo 1: en el juego Pac-Man el agente recibe recompensas por comer objetos, y entonces ya hay una recompensa implícita.

Ejemplo 2: un sistema de conducción autónoma no tiene un sistema de recompensas, sino que hay que añadirlo.

- **Frecuencia:** las recompensas son **densas** cuando el agente recibe una recompensa (diferente de cero) en cada iteración acción-reacción o en la mayoría de las iteraciones. Contrariamente, hablamos de recompensas **dispersas*** cuando solo se reciben recompensas (diferentes de cero) al lograr objetivos principales o al completar la tarea, que sería el caso más extremo.

* En inglés, *sparse*

Ejemplo 1: un robot recibe una recompensa por cada paso que da sin caer, recompensa que es densa.

Ejemplo 2: otro robot solo recibe una recompensa cuando acaba de soldar una pieza, recompensa que es dispersa.

- **Valor:** las recompensas pueden ser **positivas** (lo que se entiende habitualmente como recompensa) o **negativas** (lo que se denominaría castigo). En un mismo sistema puede haber recompensas de los dos tipos. Ejemplo: jugando a ajedrez podéis recibir una recompensa positiva por comer una pieza del enemigo y una negativa si el enemigo os come una pieza.
- **Momento:** hay recompensas que son **inmediatas**, es decir, el agente las recibe inmediatamente después de hacer una acción que merece esta recompensa. Pero a menudo muchas recompensas son **diferidas**; es decir, hay que realizar muchas acciones correctas antes de recibir la recompensa.

Ejemplo 1: Pac-Man recibe una recompensa de un punto de manera inmediata cuando se come una bolita pequeña.

Ejemplo 2: si hay que encontrar la ruta óptima teniendo en cuenta el tráfico en tiempo real, hasta que no se completa el recorrido no se confirma cómo era de buena la elección hecha, y entonces la recompensa es diferida.



Los agentes de RL son especialistas en encontrar «agujeros» en el sistema de recompensas. Por ejemplo, un agente que juega a un juego de plataformas puede pasar un tiempo infinito saltando barriles y no avanzar nunca, porque así obtiene una infinidad de puntos sin riesgo. Fuente: <https://bit.ly/2KtiPFS>.

El diseño del sistema de recompensas es uno de los aspectos más delicados en el RL, puesto que hay que pensarlo bien para no caer en algunos problemas comunes:

- No recompensar lo que realmente queremos que haga el agente: si en una carrera la recompensa solo es pasar por la línea de meta, el coche, en lugar de recorrer todo el circuito, dará marcha atrás para cruzarla.
- Ponérselo muy difícil al agente: en general, interesa añadir recompensas intermedias para que sepa que va por el buen camino (por el camino que queremos).
- No valorar bien las recompensas, de modo que el agente encuentre más provechoso quedarse en bucle completando un objetivo secundario y acumulando pequeñas recompensas indefinidamente.

Diseñar un buen sistema de recompensas requiere a menudo un cierto grado de ensayo y error hasta que vamos encontrando los comportamientos deseados.

A veces, es muy difícil diseñar un sistema de recompensas para un problema dado, especialmente cuando se trata de problemas reales muy complejos como por ejemplo la conducción autónoma. En estos casos, hay dos alternativas de aprendizaje que pueden ser útiles. El **aprendizaje por imitación** (*imitation learning*) imita a un experto que realiza la tarea para intentar hacerlo de la manera más parecida que se pueda; por ejemplo, tomando como referencia lo que hacen los conductores humanos. Por otro lado, el **aprendizaje por refuerzo inverso** (*inverse RL*) deduce las recompensas de las acciones que realizan los expertos, y genera así un sistema de recompensas de manera experimental y más o menos automática.

Tesla

El fabricante de automóviles Tesla usa una aproximación parecida al aprendizaje por imitación para sus sistemas de conducción autónoma: registra qué hacen los conductores humanos cuando conducen los coches manualmente y entrena sus sistemas de conducción con estos datos.

1.4.4 Políticas

El conjunto de criterios que un agente utiliza para decidir qué acción debe ejecutar, dado el estado actual del sistema, se denomina **política**, y se designa con la letra π . El objetivo del RL es encontrar la política que maximice la recompensa acumulada $G(k)$, tal como se ha definido en la ecuación 1.

Este objetivo, no obstante, es complicado por diferentes motivos. Primero, como hemos visto, la recompensa para una acción puede ser diferida en el tiempo, y entonces el agente debe elegir una acción que puede que no tenga una recompensa inmediata tan buena como otras pero que a largo plazo lo lleve a un resultado (recompensa acumulada) mayor. También, a menudo no se puede saber *a priori* qué acción lleva a qué estado, y solo se puede encontrar esta asociación por ensayo y error. Finalmente, en la mayoría de los problemas mínimamente interesantes, el espacio de estados S es tan grande que aprender estado por estado qué acciones son mejores resulta inviable.

Los mejores métodos de RL intentan superar estas dificultades con diferentes estrategias que iremos estudiando.

Los diferentes aspectos que caracterizan las políticas de un agente son:

- **Inmediatez:** (a) la política elige siempre recompensas inmediatas, y entonces se denomina política **avariciosa**,* o (b) la política trata de conseguir recompensas mejores a largo plazo, y sería una política **óptima**.
- **Exploración frente a explotación:** la política trata de, o bien **explorar** todas las posibilidades a su alcance, o bien de **explotarlas**, y entonces, cuando encuentra una manera de resolver el problema, la utiliza siempre y no prueba otras vías, alguna de las cuales podría ser mejor que la actual.
- **Azar:** una política es **determinista** cuando, dadas las mismas condiciones, ejecuta la misma acción siempre. En cambio, una política **estocástica** tiene

* En inglés, *greedy*

un factor aleatorio más o menos importante que hace que a veces elija acciones diferentes para el mismo estado.

Todos estos aspectos están muy relacionados entre ellos; por ejemplo, una política puramente estocástica favorecerá totalmente la exploración pero no explotará (aprovechará) lo que aprenda.

En la práctica, se utiliza mucho un tipo de política denominado ϵ -**avariciosa** (ϵ -*greedy*). Dado un valor $0 \leq \epsilon \leq 1$, la política elegirá una acción aleatoria con probabilidad ϵ o, si no, elegirá la que considere mejor acción (comportamiento avaricioso). De este modo, se busca encontrar un equilibrio entre exploración o aleatoriedad (ϵ) y explotación o determinismo ($1 - \epsilon$).

Cuando un agente empieza su proceso de aprendizaje, interesa que explore diferentes vías, en primer lugar porque todavía no sabe qué consecuencias tendrán sus acciones y, por lo tanto, no tiene mucho sentido intentar explotar un camino concreto. En cambio, cuando el agente ya ha explorado y aprendido más, es mejor que busque utilizar las acciones correctas en vez de continuar explorando al azar. Para modelar este comportamiento, se suele modificar la ϵ con una **atenuación**,* que hace que la ϵ empiece con un valor relativamente alto (próximo a 1) y vaya decreciendo poco a poco hasta un valor final muy bajo (cero o próximo a cero). Esta atenuación puede ser exponencial o lineal; por ejemplo, la atenuación lineal viene dada por la expresión

* En inglés, *decay*

$$\epsilon_{t+1} = \max \left(\epsilon_{\text{final}}, \epsilon_t - \frac{\epsilon_{\text{inicial}} - \epsilon_{\text{final}}}{k} \right) \quad (2)$$

donde k es una constante que regula el número de pasos para llegar de $\epsilon_{\text{inicial}}$ a ϵ_{final} , es decir, si la atenuación es más lenta o más rápida. Elegir la k depende de la complejidad del problema: cuanto más complejo sea, más iteraciones serán necesarias y, por lo tanto, más tiempos de exploración harán falta; entonces interesará una k mayor.

1.5 Aplicaciones

El aprendizaje por refuerzo requiere miles o millones de acciones para entrenar a los agentes. Por este motivo, la mayoría de los experimentos se aplican a entornos simulados, ya que el coste en tiempo y recursos de hacer estos experimentos sería inasumible físicamente. Concretamente, los juegos suelen ser los entornos donde se prueban la mayoría de los métodos nuevos, porque suelen tener un espacio de estados muy grande y es relativamente fácil establecer un sistema de recompensas. No obstante, el aprendizaje por refuerzo ya se utiliza en varias aplicaciones prácticas de todo tipo. A continuación veremos algunos de los ejemplos más destacables actualmente:

- **Economía y finanzas.** A pesar de que la predicción de las cotizaciones de valores es una tarea de aprendizaje supervisado con series temporales, estos sistemas solo pueden predecir las cotizaciones pero no decidir cómo actuar. Entonces se añaden agentes entrenados con RL que deciden, en función de diferentes variables económicas y políticas y de las predicciones del mercado, qué valores comprar o vender teniendo como recompensa los beneficios obtenidos. Los principales bancos de inversión en todo el mundo ya utilizan este tipo de sistemas.
- **Medicina.** Encontrar las dosis adecuadas de tratamientos para dolencias graves o crónicas es un problema complejo porque a menudo se requiere una dosis mínima para lograr el efecto deseado, pero a la vez una dosis excesiva puede producir efectos secundarios al paciente (y, además, comporta un gasto innecesario). Se utilizan sistemas de RL para encontrar la dosis más adecuada según diferentes parámetros clínicos, genéticos, etc., del paciente y, además, la dosificación se puede ajustar constantemente en función de la evolución de la dolencia y el estado general del paciente.
- **Organización de la producción.** La optimización de cadenas de producción y suministro es un problema muy complejo y dinámico. Los planificadores basados en RL permiten una gestión en tiempo real de los recursos, proveedores, materiales, etc., para optimizar la producción y distribución de productos en grandes empresas.
- **Química.** Hay diferentes aplicaciones de RL en la industria química. Por un lado, se utiliza en el diseño de fármacos, en el que las moléculas que encajan mejor con proteínas objetivo son seleccionadas y mejoradas, dado que su recompensa viene dada por la capacidad de encajar con estos objetivos.

En el ámbito de la ingeniería química, los sistemas de RL permiten ajustar diferentes parámetros en los reactores para lograr unas condiciones determinadas y optimizar los procesos de producción. Entonces, ajustando temperaturas, añadiendo ingredientes o catalizadores, etc., se puede mejorar mucho el rendimiento final y la calidad del producto.

- **Conducción autónoma.** La conducción autónoma es un tema de investigación que actualmente recibe muchísima atención por parte de investigadores y fabricantes. Aunque el aprendizaje por refuerzo no se utiliza como sistema global de conducción, sí que se emplea con tareas específicas como aparcar, mantenerse en el carril, hacer cambios de carril, mantener o ajustar la velocidad, optimizar el uso de combustible o baterías, etc.
- **Robótica.** La robótica es una de las áreas naturales de investigación y aplicación del RL. Un problema de los sistemas de control robótico «tradicionales» es que su comportamiento es bastante rígido. En cambio, los sis-

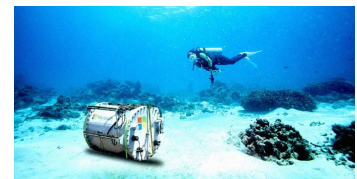
temas de control basados en RL muestran una flexibilidad mucho mayor, como por ejemplo la lograda con el sistema QT-Opt de entrenamiento con robots reales, que les permite agarrar correctamente (con el 98 % de éxito) piezas que no han visto durante el entrenamiento.



Banco de entrenamiento del sistema QT-Opt, con siete robots que trabajan en paralelo para acelerar el aprendizaje.

- **Contenidos y marketing en línea.** La elección de qué contenidos, productos o anuncios mostrar a un usuario en una página web o aplicación equivalente es crucial para mejorar el impacto que puedan tener, motivo por el cual detrás de cada anuncio en línea hay muchos algoritmos que intentan hacer el trabajo todo lo bien que pueden. El aprendizaje por refuerzo es una buena manera de tratar esta tarea porque permite ajustarse rápidamente a los intereses de los usuarios, y esto está demostrado que puede despertar el mayor interés y, por lo tanto, conseguir un impacto mejor. Hay grandes inversiones en RL en este ámbito; por ejemplo, Facebook ha desarrollado una plataforma llamada *Horizon* con esta finalidad.
- **Ingeniería.** Actualmente, en muchos problemas de ingeniería se aplica RL. Veremos un caso de esta aplicación, que nos toca muy de cerca a los científicos de datos: la gestión de la energía y la temperatura de los grandes centros de datos. Estos centros tienen decenas de miles de ordenadores que trabajan 24/365, lo que implica un consumo energético y unas necesidades de refrigeración enormes. Por estos motivos, una gestión eficiente de los equipos y consumos puede ahorrar millones de euros. Google usa un sistema de RL para gestionar los consumos y la refrigeración en sus centros de datos.

Del mismo modo, la distribución de trabajos entre todos estos ordenadores también se hace, en muchos casos, con métodos de aprendizaje por refuerzo.



Microsoft ha probado con éxito centros de datos submarinos, que tienen una temperatura estable y una refrigeración mucho más sencilla.
Fuente: <https://bit.ly/3o46qxj>.

2. Tipología de métodos

Como en cualquier otra tarea de aprendizaje automático, en el aprendizaje por refuerzo se han propuesto muchos métodos diferentes, diseñados contando con distintos elementos, enfocados a diferentes tipos de problemas y teniendo en cuenta distintas complejidades.

A pesar de que hay muchos elementos que caracterizan los diferentes métodos de RL, aquí veremos los rasgos más importantes que nos permitirán identificar rápidamente cuál es el planteamiento utilizado por cada uno y clasificarlos por familias. Estos rasgos son:

- **Modelo del entorno:** el método usa un modelo que simula el entorno para encontrar la política óptima o aprende simplemente interactuando con el entorno.
- **Política actual u óptima:** se usa una única política para aprender e interactuar o, por el contrario, son dos políticas diferentes.
- **Política o valor:** el método representa la política mediante variables específicas del problema que hay que ajustar o aprende estimando la recompensa potencial de cada estado o acción.

Finalmente, veremos un mapa con las diferentes familias de métodos; los más importantes los estudiaremos con detalle en apartados posteriores.

2.1 Modelo del entorno

¿Qué sabe un agente del entorno? Tal como hemos definido el RL, en principio el agente no sabe nada del entorno, sino que lo va descubriendo a medida que interactúa con él, viendo qué cambios de estado se producen y qué recompensas se reciben.

2.1.1 Métodos basados en modelos

De algunos entornos sencillos, sin embargo, se conoce perfectamente el funcionamiento, que es de un estilo determinado: si estamos en el estado s_i y hacemos la acción a_n , entonces pasaremos al estado s_j . Cuando hay una descripción completa del entorno de este estilo, se dice que hay un **modelo** del

entorno. Este modelo se puede aprovechar para explorar todas las posibles ramas de acción-estado y calcular cuál es la política óptima. Los métodos que usan un modelo del entorno de esta manera se denominan **basados en modelos**.

Hay dos problemas con esta aproximación:

- 1) Puede ser muy difícil construir un modelo de un problema o entorno.
- 2) Explorar todas las posibilidades, incluso si se hace de manera eficiente, es extremadamente costoso, salvo en entornos donde hay muy pocos estados.

2.1.2 Métodos sin modelo

Por estos motivos, en la práctica, la mayoría de los métodos que se utilizan no están basados en modelos del entorno, sino que, como se planteaba al principio, aprenden mediante las interacciones con el entorno; son lo que se denomina métodos **sin modelo** (*model-free*).

2.2 Política actual u óptima

Como hemos visto, la política de un agente es el conjunto de criterios con el que se decide qué acción ejecutar dado el estado actual del sistema. Por otro lado, también hemos visto que el agente aprende de las interacciones con el entorno, es decir, modifica su política en función de lo que se va haciendo y de las recompensas que va obteniendo.

2.2.1 Métodos de política actual

Según este planteamiento, la misma política con la que va decidiendo qué hacer va cambiando en función de lo que ha hecho. Esta situación es un poco compleja porque se ajusta a un modelo y a la vez se usa para decidir qué hacer. Posiblemente, las decisiones tomadas irán cambiando de una iteración a otra, lo que puede provocar inestabilidades en el aprendizaje. Los métodos que trabajan de este modo, actualizando inmediatamente la misma política con la que interactúan con el entorno, se denominan métodos de **política actual**.* A modo de analogía, imaginemos un piloto de avión que debe aprender con la práctica. En este caso va pilotando el avión y corrige sus acciones (su política) a medida que adquiere práctica y ve cómo funciona el avión.

* En inglés, *on-policy*

2.2.2 Métodos de política óptima

Una manera alternativa de entrenar un agente es tener dos políticas: una **política de comportamiento**,* que es la que el agente utiliza para decidir qué acción ejecutar y que solo se actualiza de vez en cuando, y una **política objetivo**,** que se actualiza constantemente y es en la que se produce realmente el aprendizaje. Esta configuración provoca que las acciones del agente sean más estables porque la política de comportamiento no cambia con tanta frecuencia, pero a la vez el agente aprende de todas las acciones y recompensas mediante la política objetivo. Este tipo de métodos se denominan métodos de **política óptima*****. Siguiendo con la analogía de los pilotos, en este caso hay dos pilotos, uno que pilota y el otro que lo observa todo y aprende qué es lo que hay que hacer en cada momento. El que pilota, en cambio, no aprende durante todo el vuelo, sino que cuando acaba el piloto observador le explica lo que ha aprendido.

* En inglés, *behaviour policy*

** En inglés, *target policy*

*** En inglés, *off-policy*

En general, los métodos de política óptima, a pesar de ser más complejos de programar, dan mejores resultados y son más eficientes, es decir, aprenden más con las mismas iteraciones. Como veremos, los métodos de RL más actuales y potentes son *off-policy*.

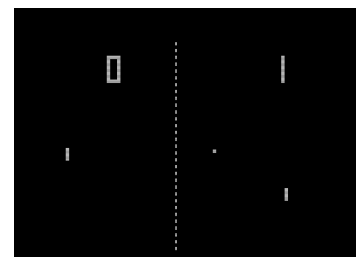
2.3 Política o valor

Hasta ahora hemos hablado de políticas y de las propiedades que pueden tener, pero todavía no hemos visto cómo un agente de RL representa una política ni cómo la entrena. A grandes rasgos, se distinguen dos aproximaciones para representar y entrenar políticas.

2.3.1 Métodos de gradiente de política

Las políticas de los métodos denominados de **gradiente de política** almacenan un conjunto de parámetros Θ que representan los elementos del entorno que puedan influir en la política. Por ejemplo, para el juego Pong clásico, las políticas tienen que registrar las posiciones y velocidades (vectoriales) de los jugadores y de la pelota, además de otras variables internas para modelar el comportamiento. Entonces, el agente empieza a interactuar con el entorno (a jugar, en este ejemplo) y, con la técnica de optimización de descenso de gradientes, los valores de Θ se van ajustando de manera que se maximice la recompensa obtenida por la política.

Los métodos de gradiente de política son muy estables y, además, permiten trabajar con acciones continuas (acelerador del coche). En cambio, son menos eficientes, en el sentido de que les cuesta más aprender.



Juego Pong de las consolas Atari, parecido al ping-pong. Fuente: <https://bit.ly/2UTc7ld>.

2.3.2 Métodos basados en valores-estados

La otra gran familia de métodos son los de **políticas basadas en valores**, en los que la política aprende cuál es el **valor** de un estado s , es decir, la recompensa acumulada G estimada por este estado teniendo en cuenta la política actual:

$$V_{\pi}(s) = E_{\pi}[G|s_0 = s] = E_{\pi}\left[\sum_{t=0}^k r_t | s_0 = s\right] \quad (3)$$

es decir, la suma de recompensas estimada empezando en el estado s y hasta el final de la tarea, suponiendo que se aplica la política π .

El problema con la expresión 3 es que todas las recompensas tienen el mismo peso, tanto las inmediatas como las esperadas en un futuro lejano, lo que hace difícil el aprendizaje en la práctica. Por este motivo, se suele introducir lo que se denomina **factor de descuento** $0 \leq \lambda \leq 1$, que resta peso a las recompensas más lejanas en el tiempo, según la expresión:

$$V_{\pi}(s) = E_{\pi}[G|s_0 = s] = E_{\pi}\left[\sum_{t=0}^k \lambda^t r_t | s_0 = s\right] \quad (4)$$

donde λ suele ser, en la práctica, un valor entre 0.95 y 0.99. Cuanto mayor sea t , es decir, a medida que nos movemos hacia el futuro, λ^t disminuye y, por lo tanto, las recompensas más lejanas influyen menos en el cálculo del valor V del estado s .

Entonces, para cada estado $s \in S$ se va calculando su valor V (ya veremos de qué manera), y esto da a la política una indicación de qué estados son los mejores. Con todo esto se construye una tabla con una entrada para cada estado $s \in S$ y que contiene $V(s)$. Esta tabla no se conoce *a priori*, sino que se va calculando durante el entrenamiento del agente.

2.3.3 Métodos basados en valores-acciones

Un problema práctico de los métodos basados en valores-estados es el siguiente: supongamos que en el instante t el agente ve que el estado del sistema es s_t . Entonces puede saber que el mejor estado (valor más grande) sería s_* , pero lo que no puede saber es qué acción $a \in A$ lo puede llevar a s_* . Por lo tanto, la información de $V(s)$ no da ninguna ventaja al agente.

La solución a este problema es crear una tabla Q con una fila para cada estado de S y una columna para cada acción posible $a \in A$, es decir, una tabla de $|S| \times |A|$ elementos. Esta tabla se denomina **tabla de valores-acciones**, o $Q(s,a)$, y ofrece una información muy útil al agente porque le permite saber qué acciones tienen un valor estimado más alto en un estado s_t .

Recompensa acumulada G

La recompensa acumulada G , definida en la ecuación 1, es la suma de las recompensas obtenidas hasta completar la tarea.

Nota

A modo de orientación, λ deberá ser más próxima a 1 para tareas en las que el número de acciones previsto sea más grande, y menor para tareas con un sistema de recompensas más inmediato.

Nota

La tabla V tiene tantos valores como estados hay en S y, como ya hemos visto, el número de estados puede ser muy grande; esto supone una limitación práctica en el uso de tablas de valores.

Tabla Q

La tabla se denomina tabla Q porque da una idea de la calidad (*quality*) de la acción para este estado.

Figura 3. Ejemplo de tabla Q

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	327	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Fuente: <https://bit.ly/3nSLTVS>.

La expresión para calcular Q es parecida a la expresión 4, pero teniendo en cuenta tanto el estado como la acción:

$$Q_{\pi}(s,a) = E_{\pi}[G|s_0 = s, a_0 = a] = E_{\pi}\left[\sum_{t=0}^k \lambda^t r_t | s_0 = s, a_0 = a\right] \quad (5)$$

es decir, la estimación de la recompensa acumulada si en el estado s se elige la acción a aplicando un descuento a las recompensas más lejanas en el tiempo mediante el factor λ .

Entonces, cuando un agente está en el estado s , tiene que mirar la tabla Q y elegir la acción con una entrada más alta. Suponiendo que conocemos la tabla de valores-acciones óptima, designada como Q^* , la acción óptima a^* para el estado s será:

$$a^*(s) = \operatorname{argmax}_a Q^*(s,a) \quad (6)$$

En general, los métodos de RL que aprenden la función Q reciben el nombre de métodos de aprendizaje-Q.*

Figura 3

Tabla Q: inicialmente, cuando todos los valores son cero, y después del entrenamiento, cuando hay valores de recompensa acumulada estimada para cada par (s,a) .

Nota

Los métodos de aprendizaje automático basados en tablas de valor-acción como por ejemplo Q-learning usan este criterio para elegir la acción óptima.

* En inglés, Q-learning

2.3.4 Aproximaciones a Q mediante funciones

A pesar de que los métodos basados en tablas Q dan muy buenos resultados a problemas de aprendizaje por refuerzo, tienen una limitación impuesta por su naturaleza: dado que requieren una tabla de estados $\times$ acciones, y en cualquier problema mínimamente complejo el número de estados es enorme, la tabla Q resulta demasiado grande; esto no solamente importa por la cantidad de memoria requerida para almacenarla, sino también porque el tiempo de entrenamiento necesario para llenar tantas posiciones se dispara.

La solución a este problema es eliminar la tabla Q y, a cambio, aprender una función que aprenda, de manera aproximada, lo que habría en la tabla, es decir, una función $f : S \times A \rightarrow \mathbb{R}$ que, dado un estado s y una acción a , devuelva la recompensa acumulada estimada.

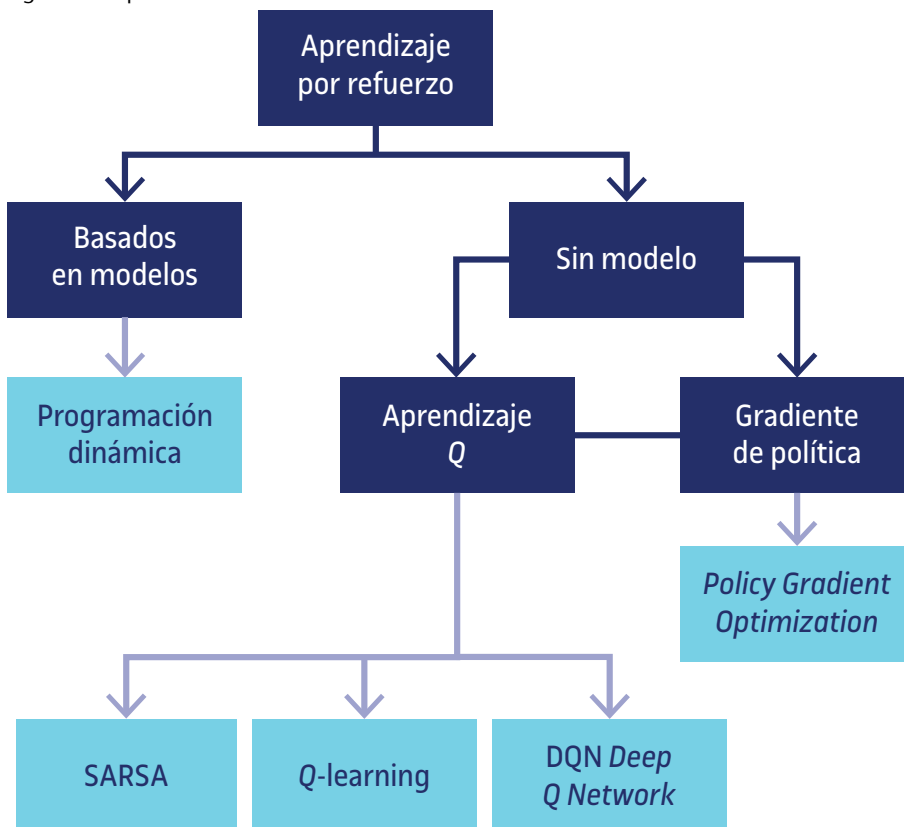
DQN

Los métodos de Q -learning basados en redes neuronales, como por ejemplo DQN, son el ejemplo más exitoso de este tipo de solución.

2.4 Familias de métodos

En la figura 4 se pueden ver las principales familias de métodos de aprendizaje por refuerzo, de acuerdo con las características descritas antes, y algunos de los métodos más representativos.

Figura 4. Mapa de métodos de RL



En los próximos apartados estudiaremos con detalle los métodos más relevantes actualmente, que pertenecen a la rama de métodos de aprendizaje-Q.

3. Aprendizaje-Q

En este apartado veremos de qué manera se llegó a los métodos de aprendizaje-Q, qué ventajas aportan y cuáles son los métodos más representativos, como por ejemplo el método Q-learning. Este método ha dado lugar, a la vez, a las innovaciones más exitosas en el área del aprendizaje por refuerzo.

3.1 Antecedentes

La historia de los métodos de aprendizaje-Q se inicia en los años cincuenta del siglo XX, al comienzo de la informática, con diferentes avances y mejoras que llegan hasta nuestros días. Haremos un repaso de la historia de estos métodos que nos permitirá entenderlos y captar las diferentes mejoras que se han introducido a lo largo de los años.

3.1.1 Trayectorias o episodios

La **trayectoria** τ de un agente siguiendo una política π se define como la secuencia completa de acciones y cambios de estado que se suceden hasta que se completa la tarea. Las trayectorias también reciben el nombre de **episodios**, más frecuente en las aplicaciones prácticas del RL.

La forma natural (y exitosa) de finalizar un episodio es que el agente complete la tarea asignada. Pero también es posible definir otras condiciones de terminación: agotamiento de la batería de un robot, máximo de turnos o tiempo utilizado, etc. Entonces el episodio finaliza pero sin éxito, lo cual hará simplemente que la recompensa sea menor. En la práctica, es conveniente limitar la duración de los episodios para mejorar el aprendizaje.

3.1.2 Cálculo de V o Q

La manera más sencilla de rellenar las tablas V o Q para entrenar a un agente de aprendizaje-Q es ejecutarlo durante muchas trayectorias (episodios) y anotar las recompensas obtenidas en cada estado o par (estado, acción) según las ecuaciones 4 o 5 respectivamente. El valor de cada estado o par (estado, acción) será finalmente la media aritmética de las recompensas acumuladas obtenidas.

Esta estrategia de ejecutar muchas trayectorias y medir las recompensas obtenidas para estimar los valores se denomina método de **Montecarlo**.

En la práctica, sin embargo, el método de Montecarlo es costoso computacionalmente porque implica recorrer muchas trayectorias enteras, anotar los estados visitados y volver atrás para calcular la recompensa acumulada y las medias correspondientes, además de que una trayectoria puede que no tenga final o que sea demasiado larga, lo que empeora todavía más el aprendizaje.

3.1.3 Ecuación de Bellman

En el año 1954, Richard E. Bellman propuso una manera de calcular V y Q que no requiere recorrer las trayectorias enteras ni volver atrás para recalcular valores, ya que el valor de un estado se define de manera recursiva en función del valor de los estados sucesores. La ecuación de Bellman define el valor de un estado s como la suma de la recompensa inmediata r_t más el valor estimado del siguiente estado s_{t+1} , multiplicado por un factor de descuento γ :

$$V_{\pi}(s) = E_{\pi}[r_t + \gamma V_{\pi}(s_{t+1})] \quad (7)$$

La estrategia de estimar el valor de un estado a partir de los valores estimados de los estados sucesores se denomina **bootstrapping**, y su ventaja es que permite aprender con cada una de las acciones que el agente ejecuta: no hay que esperar a finalizar la trayectoria.

Aunque Bellman propuso su ecuación para resolver problemas de programación dinámica, lo que corresponde a métodos basados en modelos, en esta estrategia veremos que se puede aprovechar para crear métodos sin modelo.

3.1.4 Aprendizaje por diferencia temporal

El aprendizaje por diferencia temporal (*temporal difference learning*, TD) combina lo mejor de los dos ámbitos: se aprende mediante interacciones repetidas con el entorno, como hace el método de Montecarlo, pero se aprende mediante **bootstrapping**, es decir, se aplica la ecuación de Bellman para que en cada acción ejecutada se actualice el valor correspondiente al estado actual.

Dado que el agente se entrena durante muchos episodios para poder lograr una política lo más próxima posible a la política óptima, cada vez que pasa por un estado actualizará el valor correspondiente con la recompensa obtenida pero a la vez teniendo en cuenta el valor estimado hasta el momento, es decir, sin olvidar lo que había aprendido en episodios anteriores. Esta actualización viene dada por la expresión:



Richard E. Bellman, matemático estadounidense y creador de la programación dinámica.
Fuente: <https://bit.ly/2j1wiee>.

Bootstrapping

El **bootstrapping** en el aprendizaje supervisado es una estrategia que consiste en utilizar muestras de entrenamiento con repetición; en el aprendizaje por refuerzo se aplica de manera diferente, aunque la idea de fondo es parecida.

Episodio

Recordemos que *trayectoria* y *episodio* son sinónimos, pero en los ámbitos del aprendizaje-Q y TD se suele hablar de episodios.

$$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)] \quad (8)$$

Entonces, esta actualización del valor existente $V(s_t)$ se controla con un parámetro α que se denomina **tasa de aprendizaje**, de modo que si es 0 no aprenderá nada, si es 1 olvidará todo lo que había aprendido antes, y con valores intermedios se hace un balance entre aprender y recordar. Habitualmente, $0.001 \leq \alpha \leq 0.5$.

3.1.5 Método SARSA

Este es un método sin modelo, de diferencia temporal, que utiliza una tabla Q de valores-acciones para representar su política y que aprende utilizando la política actual (*on-policy*); es decir, solo hay una política que se utiliza tanto para interactuar con el entorno como para ir aprendiendo.

Para trabajar con valores-acciones (tabla Q), se utiliza la misma expresión 8 pero aplicada a Q en vez de a V :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (9)$$

La política que se utiliza es ϵ -avariciosa para favorecer la exploración del espacio de estados y permitir que encuentre soluciones mejores que el actual.

Aunque SARSA garantiza que encontrará la política óptima, para ello es necesario que cada par (estado, acción) se visite una infinidad de veces y que la política sea determinista. Evidentemente, estas condiciones no se pueden trasladar a la práctica, aunque en general se ha aplicado con éxito a algunos problemas no muy complejos. Además, el hecho de que sea un método *on-policy* provoca que el aprendizaje sea poco estable porque la política se utiliza y se va cambiando a la vez.

3.2 Q-learning

Como hemos visto, SARSA era un buen punto de partida para el aprendizaje basado en tablas Q , pero tiene algunas limitaciones prácticas, en especial por el hecho de que es un método *on-policy*. El método Q-learning resuelve esta limitación trabajando con dos políticas:

- Una política de **comportamiento**, que se utiliza para interactuar con el entorno.

Terminología

El nombre SARSA viene del hecho de que se calcula con $s_t, a_t, r_t, s_{t+1}, a_{t+1}, \dots$

Ved también

Una política ϵ -avariciosa elige una acción al azar con probabilidad ϵ o, si no, la mejor acción disponible, tal como se explica en el subapartado 1.4.4.

Nota

Q-learning también es un método sin modelo, de diferencia temporal y de aprendizaje-Q mediante tabla, como SARSA.

- Una política **objetivo**, que es la que aprende de las interacciones y la que se utilizará finalmente cuando acabe el entrenamiento.

La política de comportamiento se actualiza periódicamente copiando los valores de la política objetivo para conseguir un comportamiento más optimizado.

La expresión de actualización de Q-learning es muy parecida a la de SARSA; la diferencia principal es que se elige la acción con valor máximo:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (10)$$

Hay dos diferencias principales entre SARSA y Q-learning: SARSA utiliza solo una política, mientras que Q-learning usa dos, y la función de optimización de Q-learning toma la acción con valor máximo.

Q-learning es un método efectivo de aprendizaje por refuerzo mucho más eficiente que SARSA, por ejemplo. La versión original, descrita aquí, utiliza una tabla Q, lo cual limita el número de estados que puede gestionar. Pero se puede adaptar para utilizar una función para aproximar Q en vez de una tabla, con lo cual mostrará todo su potencial con problemas mucho más complejos, tal como veremos a continuación.

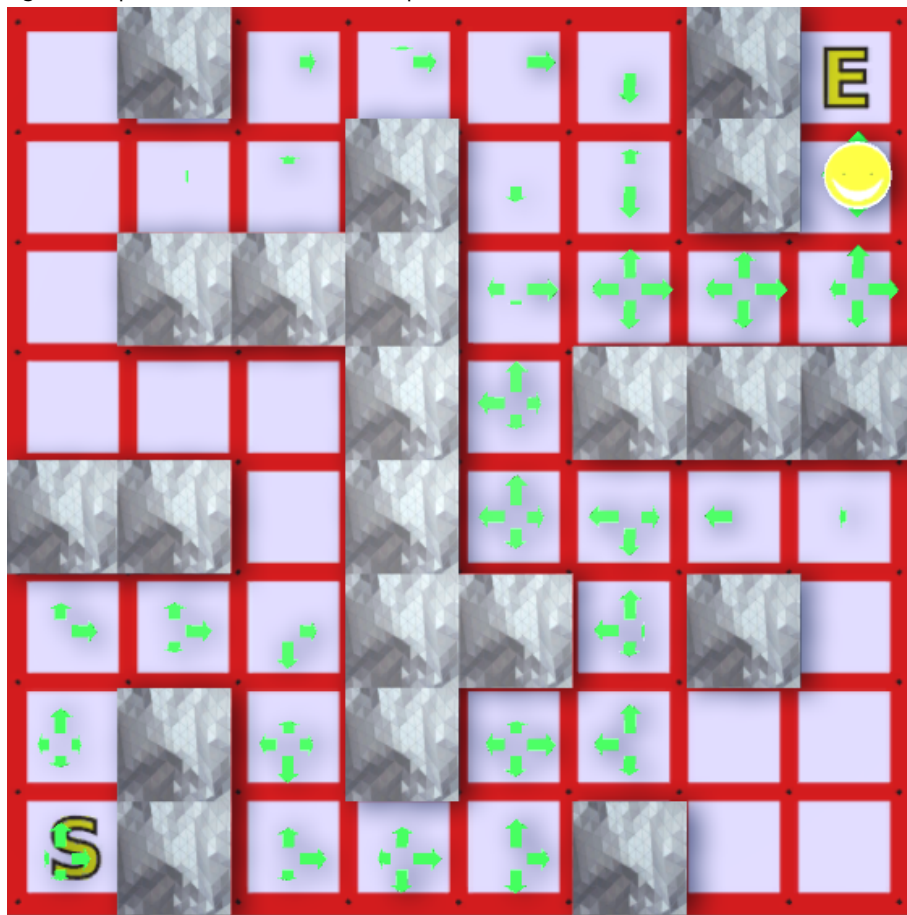
3.2.1 Ejemplo: laberinto

El problema que hemos de resolver es muy sencillo: en un laberinto hay una posición inicial (start) S, una salida (exit) E y algunas casillas bloqueadas con una roca. Queremos utilizar Q-learning para que el agente aprenda a encontrar la salida del laberinto.

Los posibles estados en este ejemplo coinciden con las posibles posiciones del agente porque no hay ningún otro elemento que cambie en este problema. Entonces la tabla Q será una tabla con tantas filas como casillas, y con tantas columnas como acciones posibles, es decir, 4 (arriba, abajo, a la izquierda, a la derecha).

En la figura 5 se han representado gráficamente los valores que tendría la tabla Q después de entrenar al agente mediante flechas de tamaño proporcional al valor $Q(s, a)$ correspondiente; como se ve, siguiendo la flecha más grande se llega directamente a la salida.

Figura 5. Representación visual de Q en el problema del laberinto



4. Redes Q profundas (DQN)

En el año 2014, la empresa Deep Mind publicó un artículo que situó el aprendizaje por refuerzo bajo los focos, revolucionó la manera de trabajar con RL y permitió aplicarlo en numerosos problemas que hasta el momento habían sido inabarcables.

El artículo en cuestión presentaba un nuevo método de RL denominado **deep Q-network** (DQN), que combina redes neuronales profundas y aprendizaje por refuerzo de una manera muy exitosa para aprender a jugar a videojuegos de la consola Atari 2600. En concreto, se logró por primera vez un sistema de RL que:

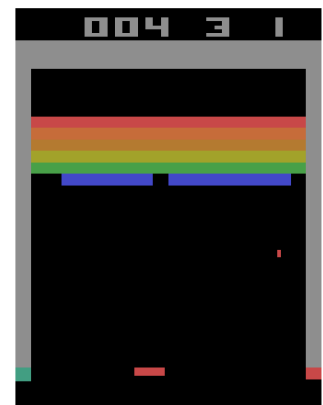
- aprende a jugar a videojuegos sin conocer las reglas o estrategias;
- solo recibe las imágenes del juego como entrada;
- solo recibe como señal de aprendizaje la recompensa obtenida en el juego;
- puede aprender a jugar a muchos juegos diferentes con la misma arquitectura;
- logra un nivel excelente en casi todos los juegos, en muchos de los cuales con un nivel superior al de los jugadores humanos profesionales;
- es capaz de encontrar nuevas estrategias por su cuenta.

Los avances más importantes de los últimos años en el área del RL tienen su origen en este artículo y el método propuesto, DQN.

En este apartado veremos una primera aproximación al RL con redes neuronales, el interés que tiene y a la vez los problemas que presenta, y entonces presentaremos cómo DQN consiguió superar estos problemas y lograr un rendimiento que no se había visto hasta el momento.

4.1 Redes neuronales profundas y Q-learning

Como hemos visto anteriormente, el método Q-learning, a pesar de tener un buen comportamiento con problemas sencillos, no se puede utilizar con problemas mínimamente complejos porque crea una tabla Q de estados $\times$ acciones y, por lo tanto, cuando el número de estados es grande, la tabla se hace demasiado grande en términos de memoria y de entrenamiento necesario para rellenar todas las posiciones. ¡Imaginemos una tabla de 10^{120} filas por 1024 columnas para jugar al ajedrez!*



En el juego Breakout el jugador tiene que hacer rebotar la pelota con la pala para destruir los bloques. Fuente: <https://bit.ly/39e5d2q>.

Lectura complementaria

V. Mnih y otros (2015). «Human-level control through deep reinforcement learning». *Nature*.

* 1024 es una cota superior del número de acciones posibles.

La solución es reemplazar la tabla Q por una aproximación mediante lo que se denomina **función aproximadora**. Esta función es $f : S \times A \rightarrow \mathbb{R}$, es decir, recibe un estado y una acción y devuelve el valor estimado de esta combinación; de este modo, resulta una aproximación de la tabla Q .

Las redes neuronales son un buen candidato a hacer de función aproximadora porque se pueden entrenar para aprender funciones a partir de unos datos de entrenamiento, y además pueden generar muchas salidas a la vez, en este caso una por cada posible acción $a \in A$.

Más formalmente, sea θ el conjunto de parámetros (pesos y sesgos) de una red neuronal, la función Q está parametrizada por θ , es decir, que tendremos una función $Q_\theta(s, a)$. En este caso el objetivo del aprendizaje por refuerzo es aprender una θ tal que Q_θ sea lo más parecido posible al valor óptimo de Q .

Para encontrar θ , hay que entrenar la red neuronal, y para ello se requiere una función de pérdida (*loss*). En principio, la elección más evidente parece la función de error de Bellman:

$$\theta \leftarrow \theta - \alpha [r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a)] \nabla_\theta Q_\theta(s, a) \quad (11)$$

donde $\nabla_\theta Q_\theta(s, a)$ es el gradiente de Q respecto a θ . En la práctica, no obstante, se pueden encontrar mejores funciones de pérdida para encontrar la matriz Q .

Por este motivo, se sustituye por una función de pérdida que se sabe que funciona bien con redes neuronales: el error cuadrático medio.* Entonces, la función de pérdida es:

$$L(\theta) = E(s, a, r, s') [(y_i - Q_\theta(s, a))^2] \quad (12)$$

donde y_i es el valor ideal de Q , es decir, el que produciría una política óptima, dado por:

$$y_i = r_i + \gamma \max_{a'_i} Q_\theta(s'_i, a'_i) \quad (13)$$

Entonces, los parámetros de la red neuronal se actualizan por descenso de gradientes según la expresión:

$$\theta \leftarrow \theta - \alpha \nabla L(\theta) \quad (14)$$

Esta actualización de θ no se hace en cada paso, sino que se aplica una actualización por minilotes (*mini-batches*), es decir, se toman N ejemplos cada vez para calcular gradientes y actualizar parámetros. Esta manera de trabajar es la habitual con redes neuronales y es la que da mejores resultados en general.

Gradiente

El gradiente de una función f respecto a un conjunto de variables V es la derivada parcial de f respecto a las variables V .

* En inglés, *mean squared error*, MSE

Hasta aquí parece que todo tendría que ir bien, pero el planteamiento tiene problemas estructurales que lo hacen fracasar. De hecho, está demostrado que Q-learning tabular con suficiente entrenamiento encuentra la Q óptima, pero en cambio esta primera aproximación con redes neuronales no. ¿Por qué?

Primer problema. A pesar de que esta red neuronal se plantea de manera bastante parecida a un problema de aprendizaje supervisado, para que el entrenamiento de una red neuronal sea válido se necesita que los ejemplos de entrenamiento en cada minilote sean **independientes y idénticamente distribuidos** (IID). En cambio, si se toman las últimas N iteraciones para ajustar pesos, estas acciones forman parte de una secuencia (la trayectoria actual) y están bastante relacionadas porque pertenecen a estados próximos. Como consecuencia, el ajuste de los pesos mediante descenso de gradientes será muy inestable.

Segundo problema. Tal y como hemos visto en la ecuación 13, cuando entrenamos una red neuronal con Q-learning aprendemos realmente dos cosas a la vez: el valor ideal de Q , designado por y_i , y el conjunto de parámetros θ de la red neuronal para producir y_i como salida. Este hecho de ajustar θ para que produzca y_i , que a la vez también está cambiando, hace que tengamos un «objetivo móvil», y cuando θ se ajusta bien y_i ha cambiado. Y así siempre. Entonces, el aprendizaje es inestable y no se detiene nunca.

Estos problemas hacen inviable una aproximación directa a Q-learning con redes neuronales; hay que resolverlos para poder hacer uso de todo el potencial de las redes neuronales.

4.2 Deep Q-network

El algoritmo DQN, que como hemos dicho en la introducción revolucionó el campo del aprendizaje por refuerzo, añade básicamente dos mejoras al algoritmo anterior para superar los problemas de aprendizaje.

Primera mejora. Para conseguir que los ejemplos de aprendizaje cumplan las propiedades IID* y, por lo tanto, el descenso de gradientes sea estable, en vez de tomar las N últimas iteraciones, se utiliza una **memoria de experiencia** (*replay buffer*) que almacena un gran número de iteraciones pasadas. El ajuste de θ por descenso de gradientes toma N ejemplos extraídos al azar de toda la memoria de experiencia, con lo cual se consigue:

* IID: ejemplos independientes y idénticamente distribuidos.

- 1) El objetivo principal, que los ejemplos de entrenamiento sean IID.
- 2) Reutilizar ejemplos y mejorar la eficiencia del entrenamiento (con las mismas iteraciones se ajustan mejor los parámetros).

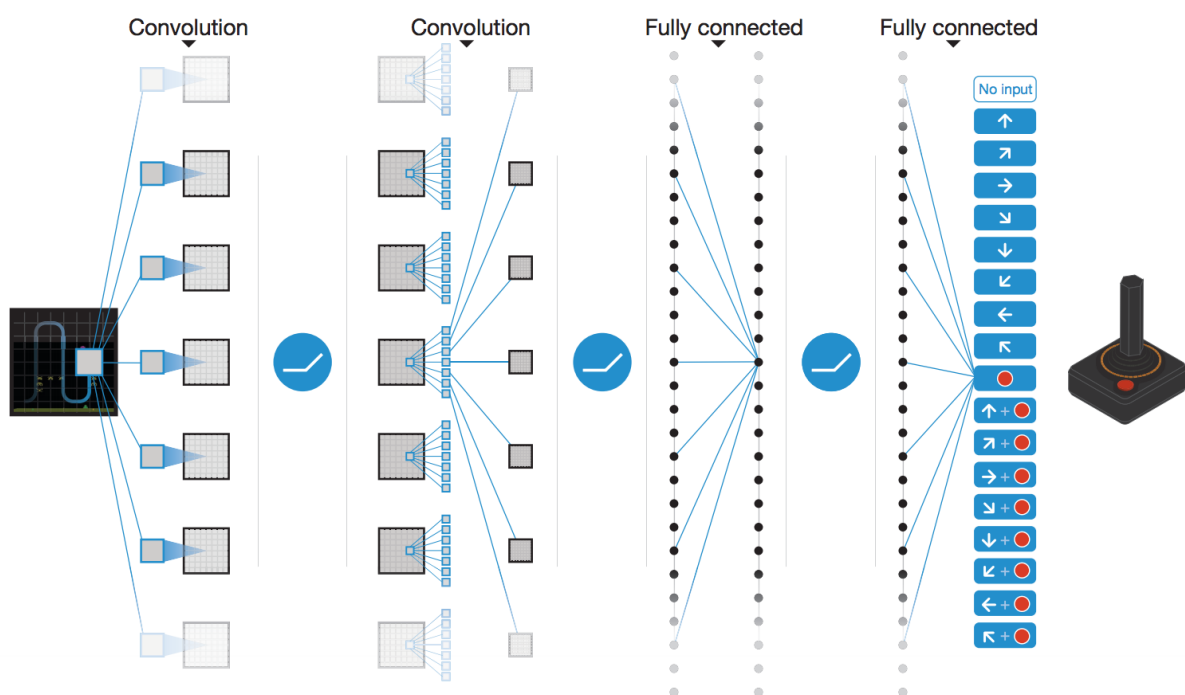
Segunda mejora. Para evitar el problema del objetivo móvil, en el que los parámetros de la red neuronal y la función y_i que se quiere aproximar cambian simultáneamente, lo que se hace es utilizar **dos** redes neuronales con una estructura idéntica:

- **Red en línea.** Se actualiza constantemente mediante descenso de gradiente (minilotes). Es la que interactúa con el entorno.
- **Red objetivo.** Se actualiza solo cada $1.000 \leq N \leq 10.000$ iteraciones, y su objetivo es modelar lo mejor posible los valores Q ideales (y_i).

Dado que durante N iteraciones los objetivos son fijos, el aprendizaje es muy estable, por lo que se supera el problema del objetivo móvil.

Volviendo al ejemplo de los juegos de Atari 2600, la red DQN propuesta recibe como entrada las imágenes del juego. La red tiene dos capas convolucionales seguidas de dos capas conectadas completamente (*fully connected*). La última capa tiene dieciocho salidas que corresponden a las dieciocho acciones posibles que permiten los juegos de Atari, tal como se puede ver en la figura 6.

Figura 6. Estructura de la red DQN



Fuente: <https://bit.ly/33eVfdn>.

Dado que el estado del juego no se puede deducir solo con la última imagen (no sabemos si la pelota va hacia la izquierda o hacia la derecha, por ejemplo), DQN recibe como entrada las cuatro últimas imágenes del juego, y así ya puede deducir velocidades, direcciones, etc.

Con este método y una red neuronal relativamente sencilla, DQN consiguió superar a jugadores humanos en muchos videojuegos clásicos, y más adelante los sucesores de DQN han logrado hitos como superar a los maestros mundiales de Go. Durante los próximos años, veremos seguramente numerosas aplicaciones de DQN y sus sucesores que expandirán los límites de lo que hoy en día puede hacer la inteligencia artificial.

Resumen

En este módulo hemos visto cómo, en entornos donde no hay un conjunto de datos para entrenar, el aprendizaje por refuerzo puede aprender mediante la interacción con el entorno y la recogida de recompensas. El elemento central del aprendizaje por refuerzo es el bucle **acción → estado, recompensa**. La política es el conjunto de criterios que permite decidir qué acción se lleva a cabo en cada momento. El método más extendido para modelar la política es una tabla o función Q que estima la recompensa acumulada que se obtendrá siguiendo un curso de acción u otro. Las redes neuronales, en concreto el modelo DQN, permiten entrenar sistemas de aprendizaje por refuerzo en los que el número de estados es muy grande y sin ningún conocimiento previo del problema o las estrategias para resolverlo.

Bibliografía

Géron, A. (2019). *Hands-on machine learning with Scikit-learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems* (2.^a ed.). Farnham, Reino Unido: O'Reilly UK.

Ravichandiran, S. y otros (2019). *Python reinforcement learning*. Birmingham, Reino Unido: Packt Publishing.

Saito, S.; Wenzhuo, Y.; Shanmugamani, R. (2018). *Python reinforcement learning projects*. Birmingham, Reino Unido: Packt Publishing.

Sutton, Richard S.; Barto, Andrew G. (2018). *Reinforcement learning. An introduction* (2.^a ed.). Cambridge, MA: The MIT Press.

