Introducción a las soluciones aproximadas

PID_00283468

Laura Ruiz Dern

Tiempo mínimo de dedicación recomendado: 2 horas

Universitat Oberta de Catalunya

Laura Ruiz Dern

Doctora en Astronomía y Astrofísica por el Observatoire de Paris. Licenciada en Física y MSc en Astrofísica, Cosmología y Física de Partículas por la Universidad de Barcelona, y doble MSc en Gestión de Proyectos y Cooperación Internacional por la ESNECA Business School. Investigadora y científica de datos con experiencia en *data mining* y *machine learning*, y actualmente especializada en *deep learning* y *deep reinforcement learning*. Ha trabajado en varios proyectos de investigación en astrofísica y ciencia de datos del Centre National d'Études Spatiales (CNES); del Centre National de la Recherche Scientifique (CNRS), en colaboración con la European Space Agency (ESA), y del Centre Tecnològic de Catalunya (Eurecat). Desde 2014 compagina sus investigaciones con la divulgación científica y con la docencia universitaria (Observatoire de Paris, Universitat Oberta de Catalunya).

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Jordi Casas Roma

Primera edición: septiembre 2021

Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Laura Ruiz Dern
Producción: FUOC

Índice

Introducción

Anteriormente vimos que los métodos *model-free* en aprendizaje por refuerzo se podían diferenciar en dos: los basados en las funciones de valor (*value-based methods*), y los basados en la política (*policy-based methods*). En los métodos basados en la función de valor, la acción a elegir dado un estado viene determinada por la estimación de su valor (máxima recompensa esperada) a partir de los valores de las acciones precedentes; la única política que existe en estos casos surge implícita de esta estimación de la función acción-valor. Los métodos basados en la política, en cambio, en lugar de aprender una función de valor que nos diga cuál es esa máxima recompensa dados un estado y una acción, aprenden directamente la política óptima que mapea estado y acción; es decir, aprenden una política parametrizada capaz de seleccionar las acciones sin necesidad de saber el valor de estas.

En este módulo iremos un paso más lejos en el aprendizaje por refuerzo que nos permitirá solucionar algunos problemas importantes:

1) situaciones en las que el espacio de estados es demasiado grande para poderlo representar en una tabla,

2) entornos donde se trabaja con espacios continuos.

Para solucionarlo introduciremos el concepto de soluciones aproximadas y cómo estas nos permitirán adaptar las técnicas de RL a espacios mucho más complejos. Asimismo, introduciremos los estimadores de función, tanto de valor como de política, viendo los métodos disponibles para cada caso, y daremos paso así al aprendizaje por refuerzo profundo (*deep reinforcement learning*) y veremos el papel que tienen las redes neuronales en él.

Objetivos

Los principales objetivos que estudiaremos en este módulo son:

1. Entender la necesidad de usar soluciones aproximadas en el aprendizaje por refuerzo.
2. Conocer los estimadores de función más relevantes y cómo se aplican en aprendizaje por refuerzo.
3. Comprender la importancia de las redes neuronales como estimadores de función.

1. Introducción a las soluciones aproximadas en RL

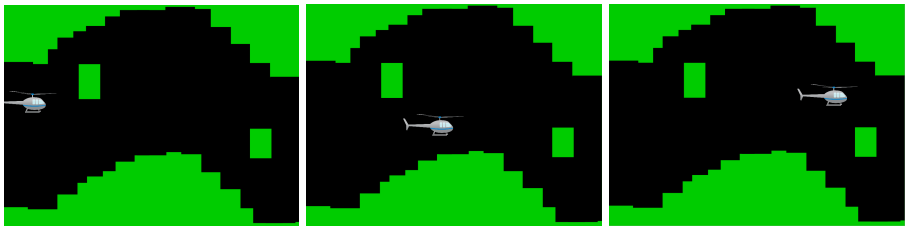
Como ya hemos visto, un estado es el valor que toma un conjunto de características o variables observables. Cada vez que una variable adquiere un nuevo valor obtenemos un nuevo estado.

En los métodos tabulares vistos en los módulos precedentes, como *dynamic programming*, Montecarlo o *temporal-difference learning*, cada estado es memorizado en el sistema. Es decir, que si nuestro agente se puede mover, por ejemplo, en un entorno representado por una cuadrícula de 6×6, en el que cada posición en la cuadrícula representa una nueva característica (es decir, la variable adquiere un nuevo valor), obtenemos 36 posibles estados. Pero si en cada posición el agente puede usar tres herramientas distintas, y puede elegir entre cuatro direcciones distintas (delante, detrás, izquierda, derecha), los posibles estados se multiplican: $36 \times 3 \times 4 = 432$.

Imaginemos por ejemplo un ajedrez. Cada vez que una pieza se mueve, el conjunto de todas las piezas en el tablero representa un nuevo estado s, tenemos una nueva imagen del tablero. Eso implica que podemos llegar a tener unos 10^{120} estados posibles, donde cada estado s se define por la posición de cada una de las piezas en el tablero, y cada cuadrado del tablero tendrá un valor que será igual a la pieza que lo ocupa (una blanca, una negra, o nada). Así, una vez tenemos todo el conjunto de estados posibles, podemos asignar una función de valor de estado $V(s)$ para cada estado. Fijémonos que la cantidad de memoria requerida para guardar cada uno de los estados, así como el tiempo de cálculo para ir construyendo la tabla, van a ser muy elevados.

Otros juegos tienen dimensiones aún mayores, como el juego de Go con 10^{170} estados, o su espacio de estados es continuo, como el juego del Helicóptero (figura 1). Sin olvidar el campo de la robótica donde pretendemos que los robots aprendan como aprendemos los humanos. El problema siempre será encontrar una función de valor o una política que nos permita describir el inmenso espacio de estados.

Figura 1. Juego del Helicóptero



Así pues, en la mayoría de situaciones reales nos encontramos con varios problemas:

- **Limitación de almacenamiento**: el número de estados posibles en los que se puede encontrar el agente es muchísimo más grande que en un juego de ajedrez. Hay demasiados estados y/o acciones y no se pueden almacenar en memoria.
- **Limitación computacional**: dado que los recursos computacionales son limitados, resulta muy lento el proceso de encontrar el valor de cada estado individualmente.
- **Imposibilidad de conocer todos los estados posibles**: es imposible conocer todos los estados posibles con anterioridad, puesto que con frecuencia los estados individuales no son totalmente observables. Imaginemos por ejemplo un robot que tenga una cámara frontal. El agente (robot) verá aquello que enfoca la cámara, lo que tiene delante, pero no puede ver a través de las paredes o lo que tiene detrás: hay muchos estados que no pueden ser observados y muchos que ni siquiera habrán existido antes.

Queda claro que en todos estos casos el espacio de estados ya no se puede representar con una tabla, y no podemos esperar encontrar una política o una función de valor óptimas. Los métodos tabulares ya no nos sirven como tales en estos entornos, pero sí podemos intentar encontrar una solución aproximada.

Necesitamos aproximar la función de valor o la política. En la práctica no inventaremos nada nuevo. La idea será combinar el aprendizaje por refuerzo con funciones de aproximación propias del aprendizaje supervisado, en un proceso llamado **generalización**. El objetivo de estas funciones de aproximación es construir una aproximación global de la función deseada (sea una función de valor o de política) generalizando a partir de algunos ejemplos de esta función. Esta combinación es la que conocemos como **aprendizaje por refuerzo con funciones de aproximación**. Cuando la función de aproximación es una red neuronal entonces hablamos de **aprendizaje por refuerzo profundo**, en inglés ***deep reinforcement learning*** (DRL).

No hay que olvidar que el objetivo final sigue siendo el mismo que en los métodos tabulares. Es decir, lo que queremos es determinar, dado un estado y entorno determinados, la acción más óptima, es decir, aquella con la que el agente obtenga las máximas recompensas.

1.1 Funciones de aproximación

Hasta ahora hemos considerado mayoritariamente métodos tabulares que podemos aplicar tanto a casos «estado» (métodos de predicción), donde cada estado s tiene asignado un valor que denominamos $v(s)$, o casos «estado-acción»

(métodos de control) donde cada pareja estado-acción (s,a) tiene asignado un valor que denominamos $q(s,a)$.

Tal y como se ha comentado en la introducción de este módulo, para solventar el problema de los espacios de estados muy grandes adoptaremos una solución aproximada basada en los atributos (características) de cada estado. Dado que constantemente encontraremos estados nuevos (no vistos antes), usaremos el conjunto de atributos de los estados anteriores que sean similares al estado actual, y a partir de estos generalizaremos las decisiones que se tomarán en estados desconocidos. En realidad, se trata de una **estimación del valor** en los estados que tienen características similares. En lugar de representar la función de valor aproximada como una tabla, ahora será una forma de función parametrizada con un vector de pesos $\mathbf{w} \in \mathbb{R}$.

Así, el valor aproximado de un estado s dado un vector de pesos $\mathbf{w}$ vendrá descrito por:

$$\hat{v}(s,\mathbf{w}) \approx v_\pi(s) \tag{1}$$

en casos «estado» (métodos de predicción), o por:

$$\hat{q}(s,a,\mathbf{w}) \approx q_\pi(s,a) \tag{2}$$

en casos «estado-acción» (métodos de control).

Donde $\hat{v}(s,\mathbf{w})$, o $\hat{q}(s,a,\mathbf{w})$, es la función de aproximación.

En el caso de una política, el funcionamiento será distinto porque necesitaremos seguir la misma política durante todo un episodio. En el momento en que cambiemos de episodio, tendremos una política nueva, pero los estados anteriores ya no nos servirán porque seguían otra política. La función de aproximación en este caso nos ayudará a parametrizar la política $\pi(a|s,\theta)$, determinando aquellos parámetros θ que la optimizan.

Como vemos, tanto en el contexto de la función de valor como en el de la política, nunca encontraremos el valor verdadero de un estado, siempre será una aproximación ya sea de su valor o de los parámetros de la política que se siga en ese estado. Pero el cálculo será mucho más rápido y podremos trabajar con espacios de estados mucho más grandes.

1.2 Función objetivo

El aprendizaje por refuerzo juzga las acciones sobre la base de los resultados que estas proporcionan. El objetivo es aprender aquellas secuencias de accio-

nes que permitirán al agente conseguir su propósito, minimizando o maximizando su función objetivo $J(\mathbf{w})$, para el caso de la aproximación de la función de valor o de acción-valor, o $J(\theta)$ para el caso de la aproximación de la política. Esta función objetivo es equivalente a hablar de la función de pérdida de una red neuronal, la cual nos permite medir la efectividad de la función de valor o de la política. En pocas palabras, es una medida del rendimiento de la red.

Así, definimos la función objetivo como una función diferencial, con $\mathbf{w}$ (o θ en la política) representando el vector de parámetros del problema en cuestión. Para actualizar y optimizar la función de valor o la política existen muchos optimizadores distintos en aprendizaje profundo, aunque uno de los métodos más usados es el de descenso o ascenso del gradiente.

Si aplicamos un gradiente a $J(\mathbf{w})$ obtenemos:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \begin{pmatrix} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n} \end{pmatrix} \tag{3}$$

El objetivo será encontrar el mínimo o máximo local de $J(\mathbf{w})$. Para hallar el descenso o ascenso del gradiente desplazaremos los parámetros $\mathbf{w}$ en la dirección de gradiente negativo o positivo, respectivamente:

$$\Delta \mathbf{w} = \pm \frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \tag{4}$$

donde α es el tamaño del paso (*step-size*), es decir, el número de actualizaciones por época. El proceso es equivalente para $J(\theta)$, la función objetivo de la política.

Básicamente, en RL cualquier optimizador que sea capaz de recibir un gradiente y realizar una actualización se puede emplear en este proceso. Aun así, usaremos siempre la versión estocástica del descenso o ascenso del gradiente, puesto que usar un único ejemplo en cada actualización nos permite, en general, converger más rápido y llegar antes a un valor óptimo.

1.3 Tipos de estimadores de función

Existen muchos estimadores de función propios del aprendizaje supervisado como, por ejemplo, las redes neuronales artificiales (ANN), los árboles de decisión, los k-vecinos más próximos, las bases de Fourier y Wavelets, o el *coarse coding* (una manera de generar buenas características para poder ajustar por ejemplo funciones lineales).

¿Pero qué estimador de función es mejor? En principio, cualquiera podría ser usado, pero en el aprendizaje por refuerzo nos encontramos con algunas propiedades distintas de los métodos supervisados y que hay que tener en cuenta (aunque no tienen por qué suponer siempre un problema):

i.i.d.

En inglés, *independent and identically distributed variables*

1) Correlaciones: las experiencias, las actualizaciones que el proceso va realizando, no son variables independientes (i.i.d.), sino que los sucesivos pasos de tiempo están correlacionados.

2) Funciones que varían: la política del agente influye en los datos que recibe, lo cual altera la naturaleza de las funciones que se aprenden.

3) No estacionariedad: las funciones de valor $v_\pi(s)$ pueden no ser estacionarias dependiendo del algoritmo de aprendizaje. Por ejemplo, en *bootstraping*, el valor en el nuevo estado forma parte de la función de actualización, lo que implica que la propia actualización no es estacionaria y puede conllevar problemas con algunas funciones de aproximación (o invalidarlas).

Además, para hacer la actualización de la función, en general usaremos el método de descenso/ascenso del gradiente, por lo que nos interesa que el estimador sea diferenciable. Dadas estas características, los estimadores más comúnmente usados y que mejor se adaptan a la mayoría de los problemas de RL son las funciones lineales y las redes neuronales (funciones no lineales).

Por supuesto, la elección dependerá del objetivo, aunque a veces puede ser difícil encontrar un razonamiento claro. Estamos ante un campo de la inteligencia artificial en pleno desarrollo y evolución. Técnicas como las funciones lineales las conocemos muy bien en la teoría, pero sabemos que en muchos casos su rendimiento no es muy bueno. En cambio, técnicas como las redes neuronales profundas, las desconocemos más a nivel teórico, pero se ha comprobado en muchos campos que suelen dar muy buenos resultados. En este texto abordaremos siempre los métodos con redes neuronales, siguiendo el estado del arte en este campo.

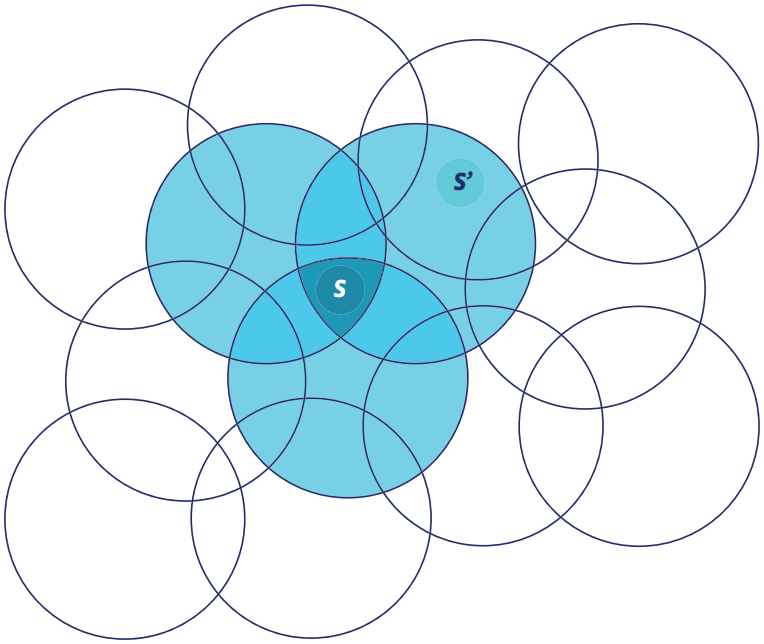
1.4 Estimador de función lineal

Los métodos lineales tienen buena convergencia y son computacionalmente eficientes, hecho que los hace muy útiles en muchos casos. Sin embargo, su principal problema es que son muy dependientes de la naturaleza de las características, que necesitan ser muy bien descritas y elegidas. Si consideramos por ejemplo un robot, necesitamos tener buenas características de los sensores de los que dispone, la localización, el porcentaje de batería que le queda, lecturas, etc. Pero estos datos no siempre son sencillos de conseguir ni elaborar.

Existen diversos métodos que facilitan la determinación de estas características para cada problema en concreto, como algoritmos polinomiales, bases de

Fourier o el ya mencionado *coarse coding*. En este último, las características son representadas en forma de círculos en el espacio de estados (figura 2); la generalización de un estado s a otro s' vendrá dado por los círculos que se solapen.

Figura 2. Representación gráfica del *Coarse Coding*. Observamos que la generalización del estado s al estado s' depende del número de características (círculos) en común



Fuente: Adaptación de Sutton *et al.*, 2019

Lectura complementaria

R. S. Sutton; A. G. Barto (2019). *Reinforcement Learning. An introduction* (capítulo 9.5). Cambridge: The MIT Press.

1.5 Estimador de función no lineal: redes neuronales

Los estimadores de función lineales asumen que la función de valor es una combinación ponderada de un conjunto de características, donde cada característica es una función del estado. Esto hace que, como hemos visto en el subapartado anterior, previamente se requiera una buena descripción y definición de las características, lo cual puede ser una tarea lenta y complicada.

Una alternativa es utilizar estimadores de función no lineales capaces de partir directamente de los estados, sin ninguna necesidad de especificar el conjunto de características ni de conocer bien el problema con antelación.

Y aquí es donde el aprendizaje profundo (*deep learning*, DL) entra en juego. Los avances de los últimos años en este campo han causado una creciente evolución en el aprendizaje por refuerzo, puesto que el DL permite trabajar con datos de entrada de muchas dimensiones. Las redes neuronales profundas (*deep neural networks*, DNN) han pasado a ser un estimador de función universal (no lineal). Gracias a las DNN el agente puede inferir un estado y una política aproximada únicamente a partir de los datos de entrada, puesto que las DNN usan representaciones distribuidas, no locales. Además, el número de nodos/parámetros necesarios para representar una misma función

decrece exponencialmente comparado con otros estimadores. Por último, las DNN son capaces de extraer o aprender los parámetros usando descenso de gradiente estocástico (SGD).

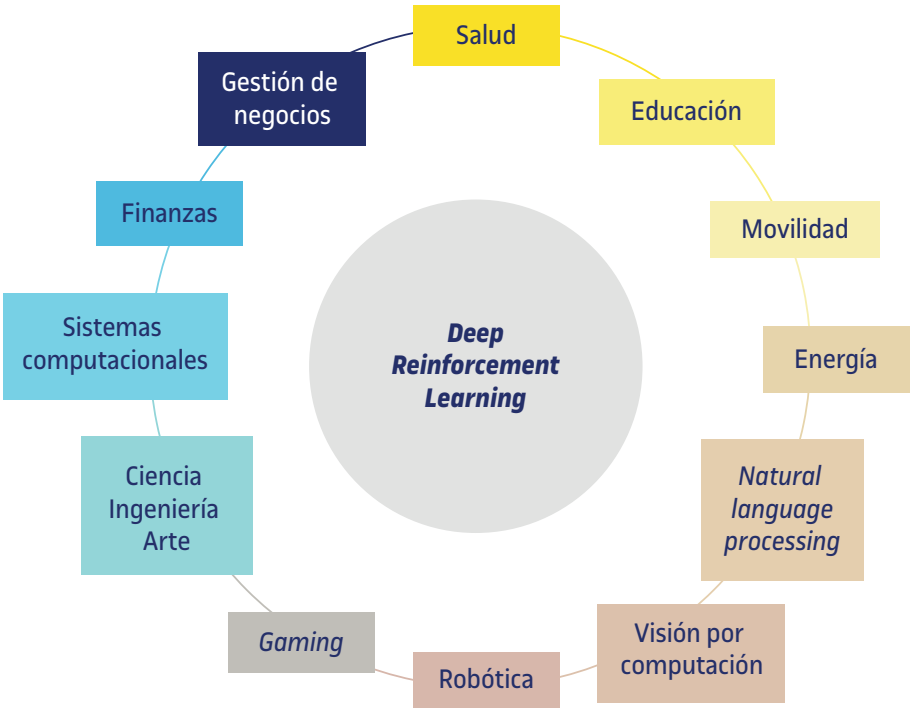
De esta fusión entre el aprendizaje profundo y el aprendizaje por refuerzo nace el **aprendizaje por refuerzo profundo** (en inglés *deep reinforcement learning*, DRL).

Deep learning + reinforcement learning = deep reinforcement learning

Consideraremos métodos de DRL cualquier método RL que use redes neuronales profundas para aproximar: la función de valor, la función Q (en *Q-learning*), la política, o el modelo (función de transición de estado o función de recompensa).

Aunque el DRL se ha dado mucho a conocer gracias a su exitosa aplicación en juegos de ordenador, especialmente los conocidos Atari, está teniendo mucha repercusión en muchos otros ámbitos, como por ejemplo la salud, la robótica, las finanzas, etc. En la figura 3 se muestra gráficamente, y sin entrar en detalles, los campos donde se está introduciendo y aplicando DRL con resultados muy prometedores e interesantes.

Figura 3. Aplicaciones del aprendizaje por refuerzo profundo



1.5.1 Redes neuronales en DRL

Las redes neuronales son algoritmos para aproximar funciones, cuya unidad principal es la neurona. Aunque actualmente no hay un vínculo explícito entre la neurociencia y las redes neuronales (son campos completamente separados), estas últimas nacieron de la idea de intentar explicar matemáticamente el funcionamiento del cerebro humano en la década de los cuarenta (McCulloch y Pitts, 1988, reimpresión de 1943). Años más tarde, empezaron a implementarse diversos modelos de cálculo basados en redes neuronales de una sola capa (perceptrón). En 1982, John Hopfield introdujo el concepto de *back-propagation* o propagación hacia atrás, lo cual permitió solucionar ciertas limitaciones teóricas de los modelos de perceptrón y mejorar el entrenamiento de las redes. Pero no fue hasta el año 2006 que nació el término de *deep learning* y de redes neuronales profundas, gracias un nuevo algoritmo propuesto por Hinton *et al.* (2006) que permitía entrenar redes profundas de forma eficiente. Desde entonces se ha producido una expansión de su uso en muchos ámbitos, llegando a ser el estado del arte en muchos de ellos, superando ampliamente a los demás algoritmos típicos del aprendizaje automático en minería de datos.

Cualquiera que sea el problema, de aprendizaje supervisado o RL, cuando la función de aprximación es una red neuronal, el objetivo siempre es el mismo. Los coeficientes de las redes neuronales permiten aproximar la función que relaciona la entrada con la salida, los *inputs* con los *outputs*. Por lo que la finalidad siempre es buscar los coeficientes o pesos que mejor describen el problema iterando continuamente el proceso para ajustar estos pesos mediante gradientes que permiten reducir el error. Como ya hemos comentado, cuando usamos redes neuronales en aprendizaje supervisado hablamos de *deep learning* (DL) y cuando las usamos en aprendizaje por refuerzo hablamos de *deep reinforcement learning* (DRL).

Lectura complementaria

A. Bosch; J. Casas; T. Lozano (2019). *Deep Learning. Principios y fundamentos*. Universitat Oberta de Catalunya.

Lectura complementaria

A. Zhang; Z. C. Lipton; M. Li.; A. J. Smola. *Dive into Deep Learning*. <https://d2l.ai/>

Ahora bien, si recordamos, en aprendizaje supervisado cada imagen tiene asociada una etiqueta (la clase a la que pertenece) y obtenemos como salida una probabilidad de clasificación a cada una de las posibles clases (por ejemplo, porcentaje de ser perro y porcentaje de ser gato). En DRL, en cambio, la salida es la recompensa que obtendremos al realizar una acción determinada en un estado concreto (por ejemplo, saltar nos puede dar 10 puntos, correr a la derecha 25, coger un objeto determinado 50, o disparar a un objetivo 0 puntos).

Es por esta razón que en RL el tipo de redes neuronales que nos interesan serán aquellas que nos permitan hacer el aprendizaje de forma incremental. Cada vez que el agente interacciona con el entorno obtenemos más datos; unos datos que nos interesa tener en cuenta porque forman parte del proceso de aprendizaje. Es decir, en lugar de utilizar todos los datos en un proceso previo de entrenamiento, cada vez que el agente recibe un retorno esta información necesita ser introducida en la red neuronal.

Tanto las Convolutional Neural Networks (CNN) como las Recurrent Neural Networks (RNN), usando la propagación hacia atrás, permiten este tipo de proceso. Por lo tanto, ambas redes pueden ser utilizadas en aprendizaje profundo. En el equipo de DeepMind fueron los pioneros (en 2013) de la fusión de RL con las CNN mientras buscaban la manera de que una máquina fuera capaz, por sí misma, de observar el entorno, interaccionar con él y aprender de esa interacción, igual que lo hace el ser humano. Empezaron desarrollando algoritmos que aprendieran a jugar a juegos antiguos como Pong o Space Invaders sin ninguna información previa a sus reglas, con el objetivo de conseguir que la máquina fuera capaz de pensar y proceder como lo haría un humano que tampoco conociera las reglas del juego. Dado que el objetivo de su investigación empezó con videojuegos, resulta natural que el tipo de redes utilizadas fueran las convolucionales, puesto que son esenciales cuando tenemos imágenes como entrada.

Curiosidades

DeepMind Technologies era una *start-up* británica que nació en 2010, fundada por el ex-niño prodigio en ajedrez, considerado como el mejor jugador del mundo, y neurocientífico Demis Hassabis, junto con Shane Legg y Mustafa Suleyman. A principios de 2014 Google consiguió adquirir la empresa por un valor de 650 millones de dólares tras, al parecer, algún intento fallido de Facebook en años anteriores.

Aun así, cabe mencionar que también las RNN tienen un papel importante tras la idea de que el agente sea capaz de memorizar aspectos o patrones en datos secuenciales. Muchos son los estudios que muestran que es posible combinar una RNN con RL y entrenar el sistema para que sea capaz, por ejemplo, de jugar a videojuegos como los de Atari (Hauskneckt y Stone, 2017); o de optimizar ciertas reacciones químicas usando la RNN como política (Zhou *et al.*, 2017); o incluso de generar por sí solo imágenes, condicionadas o no por los datos de entrada, para engañar a una red diseñada para distinguir entre datos reales y renderizados (Ganin *et al.*, 2018).

1.5.2 Recordatorio de conceptos básicos de las redes neuronales

En este subapartado repasaremos muy brevemente los conceptos básicos del aprendizaje automático y los principales parámetros de las redes neuronales, necesarios para su implementación en el ámbito del aprendizaje por refuerzo.

Recordemos que el aprendizaje automático o *machine learning* engloba todos los algoritmos y métodos que, basándose en experiencias pasadas, son capaces de encontrar patrones en esos datos y conseguir que una máquina aprenda de manera automática, es decir, sin necesidad de ser programada. Este aprendizaje se divide en supervisado o no supervisado, según si disponemos de información (datos) que nos indican cuál es el resultado esperado o no, respectivamente.

Las redes neuronales permiten realizar tareas en ambos tipos de aprendizaje. En el aprendizaje supervisado encontramos tareas de:

- **Clasificación**: permiten asignar instancias, descritas por atributos (discretos o continuos), a un conjunto de clases. Los datos de entrenamiento deben estar etiquetados con la clase que les corresponde.

- **Regresión**: se utilizan para predecir valores numéricos en juegos de datos continuos, asignando estos valores a las instancias (en lugar de etiquetas).

Mientras que en el lenguaje no supervisado, generalmente, hablamos de tareas de:

- **Agrupamiento** o *clustering*: permiten agrupar los datos en conjuntos por patrones de similitud entre ellos; no se les asigna ninguna etiqueta.

Sea cual sea la tarea que desarrollan, en general, las redes neuronales relacionan el juego de datos inicial con la salida o resultados del algoritmo aplicado sobre estos datos. Por esta razón solemos referirnos a las redes neuronales como **estimadores de función universales**, ya que son capaces de aprender y aproximar funciones que relacionan los datos de entrada x con la salida $f(x)$, $f(x) = y$.

Entrenamiento, validación y test

Para entrenar un algoritmo de aprendizaje automático es importante tener un conjunto de datos para entrenar y otro para verificar que el modelo funciona correctamente y asegurar que generaliza bien. De este modo se intenta que el algoritmo capte los patrones importantes de los datos y evitar que se especialice únicamente en los datos de entrenamiento (lo que haría al algoritmo incapaz de predecir o clasificar correctamente datos nuevos). Cuando ocurre esta especialización de los datos de entrenamiento hablamos de **sobreentrenamiento** o *overfitting*.

Muchas veces se realiza una triple partición de los datos: entrenamiento, validación y test. Esto es así porque si solo se divide en entrenamiento y test, estaremos usando el mismo conjunto de test para validar y para probar el algoritmo *a posteriori*. En consecuencia, las decisiones que se tomen en el ajuste de hiperparámetros puede verse afectada por el conjunto de test. Si este proceso se repite con frecuencia podría conllevar una estimación demasiado optimista de la precisión, es decir, que estuviéramos implícitamente causando sobreentrenamiento. Para evitar esta sobreestimación de la precisión, la solución pasa por dividir el conjunto de datos en tres:

1) Conjunto de entrenamiento para entrenar.

2) Conjunto de validación para validar el modelo durante el entrenamiento y tomar las decisiones sobre el ajuste de hiperparámetros.

3) Conjunto de test para estimar la precisión del modelo.

Cuando no se redefinen los hiperparámetros de forma iterativa, el problema se puede simplificar con solo dos conjuntos: entrenamiento y test.

La técnica más habitual para generar los conjuntos aleatorios de entrenamiento y test es la *k-fold cross validation*.*

* https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Elementos de las redes neuronales

Los elementos básicos de las redes neuronales son las neuronas (su unidad) y las capas. Tenemos tres tipos de capas: la de entrada, las intermedias u ocultas, y la de salida. El número de neuronas en cada capa, así como el número de capas ocultas, dependerá del conjunto de datos y del problema en cuestión. Las redes neuronales se constituyen, además, de otros componentes claves que facilitan el proceso de transmisión de la información inicial entre las distintas capas y que permiten el ajuste preciso de los parámetros, aproximando así la solución óptima.

- **Peso** w^i_j: importancia que se le da a un valor de entrada x_j en la transmisión entre la neurona i y la neurona j. Estos pesos se optimizan a lo largo del proceso de entrenamiento.

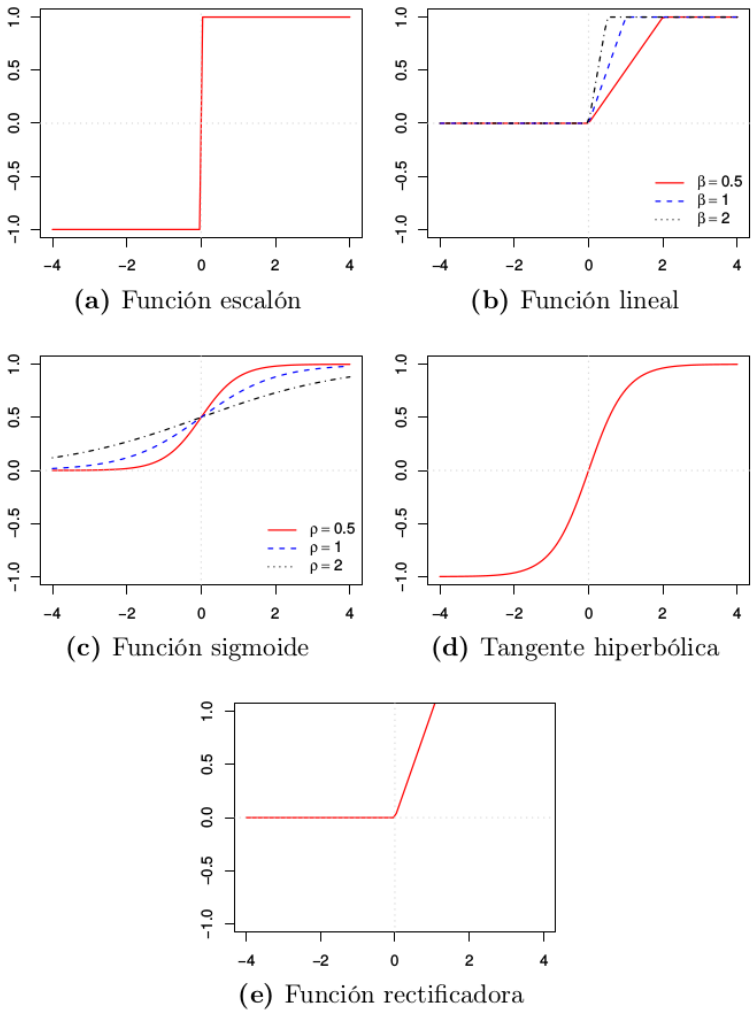
- **Función de entrada**: es la que permite que cada neurona pueda combinar los valores de entrada x_j con su peso (w^i_j) y agregar los valores obtenidos de todas las conexiones de entrada en un único valor resultante. Las más conocidas son: las funciones de suma ponderada, la de máximo, la de mínimo y la lógica.

- **Función de activación o transferencia**: procesa el valor resultante de la función de entrada y lo modula para generar un valor de salida y_i, permitiendo a la red neuronal procesar aproximaciones no lineales. Las más usuales son: la función escalón, la función lineal, la función sigmoide, la tangente hiperbólica y la rectificadora (figura 4).

- ***Bias* o sesgo**: valor o vector constante añadido al productor de las entradas y sus pesos para compensar el resultado de la función de activación hacia el lado positivo o negativo.

- **Función de pérdida**: modela el error o diferencia obtenida entre el resultado obtenido y el esperado, es decir, cuán buena es la red neuronal para resolver el problema en cuestión. Entre las más conocidas encontramos: la función cuadrática, la función de entropía cruzada, o la función exponencial.

Figura 4. Representación de las funciones escalón, lineal, sigmoide, hiperbólica y rectificadora



Fuente: Bosch, Casas y Lozano (2019)

Ajuste de parámetros

Para ajustar los pesos y los sesgos (*biases*) de la red neuronal de forma más eficiente, se usan principalmente dos algoritmos:

1) Descenso del gradiente: este método tiene como objetivo representar el error que se comete en cada instancia del conjunto de entrenamiento (entre su resultado obtenido y el esperado) en función de los pesos y sesgos, y determinar el valor de estos en el mínimo de la función. Es decir, determinar en qué punto el gradiente de la función de coste (i.e., pendiente de la tangente a la curva) es cero. Los nuevos valores de pesos y sesgos dependerán también de la velocidad de aprendizaje, que marca a qué velocidad se ajustan los parámetros durante el entrenamiento.

2) Backpropagation: si la red es multicapa no podemos aplicar el método del descenso del gradiente porque es imposible saber tras cada capa cuál es el valor de salida correcto. Los pesos y sesgos en cada capa oculta deben modificarse únicamente a partir del error obtenido en la capa de salida. Para ello se realiza una propagación hacia atrás de este error por cada neurona en las capas ocultas y de forma proporcional a la importancia que ha tenido cada nodo de las capas ocultas en el error final obtenido.

En cualquier problema, sea de aprendizaje supervisado o de aprendizaje por refuerzo, el objetivo de la red neuronal siempre será buscar los coeficientes o pesos que describen mejor el problema, es decir, que mejor permiten aproximar la función que relaciona la entrada con la salida, los *inputs* con los *outputs*. Y esta búsqueda se realiza en un proceso iterativo en el que se van ajustando estos pesos (y sesgos) por medio de gradientes con tal de reducir el error final entre el resultado y lo esperado.

Tipos de redes neuronales

Existen varios tipos de redes neuronales artificiales, que se distinguen por el tipo de capas que las conforman. Según los datos o el objetivo del problema será más conveniente usar una u otra. Los principales tipos se pueden resumir en:

- **Perceptrón**: la red más sencilla conformada por una única neurona.

- **Perceptrón multicapa**: añade complejidad al perceptrón, incluyendo una capa de entrada, unas capas ocultas, y la capa de salida. La información de la entrada pasará por estas capas ocultas y será tratada y manipulada por los pesos y sesgos establecidos, llegando a un resultado en la salida.

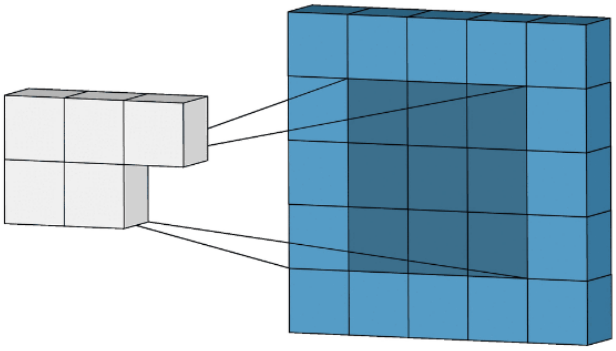
- **Red neuronal convolucional** (CNN): partiendo de la estructura base de las redes multicapa, este tipo de red utiliza capas convolucionales. Estas capas utilizan filtros que se desplazan por una cuadrícula establecida sobre la imagen para intentar extraer, junto con capas de *pooling*, las características y patrones principales de cada imagen (figura 5). Son muy utilizadas para clasificación de imágenes, detección de objetos y visión por computador.

- **Red neuronal recurrente** (RNN): nos permite analizar datos con evolución temporal, que dependen de puntos del pasado para predecir el futuro. Esta representación se hace mediante matrices de estado, las cuales se utilizan para calcular la salida. Los tipos LSTM y GRU son las RNN más usadas en predicciones, NLP, etc.

- **Autoencoders** (AE): son redes destinadas a intentar comprimir los datos sin afectar a la calidad de los mismos. A diferencia de los anteriores tipos,

en los que varias unidades de la entrada compartían la misma etiqueta de salida, en este caso las etiquetas a predecir son las mismas unidades de la entrada. Así, la capa oculta será más pequeña que la de entrada y salida para forzar esta compresión.

Figura 5. Representación del filtro en una capa convolucional



1.5.3 Tipos de algoritmos en DRL

Todo algoritmo que utilice un estimador de función basado en redes neuronales será un método de DRL. Teniendo esto en cuenta, podemos destacar los tres métodos DRL de más importancia y aplicación hoy en día:

1) Redes profundas Q (*deep Q-networks*). Como hemos visto anteriormente, el método *Q-learning* (*off-policy*) es uno de los más potentes entre los métodos de TD. En este caso la tabla que se genera se conoce como tabla Q (*Q-table*).

Por las mismas razones detalladas en la introducción de este módulo, y del mismo modo que en los demás métodos tabulares estudiados, producir y actualizar una tabla Q cuando nos encontramos en entornos de espacios de estados muy grandes puede resultar completamente ineficiente.

La solución está en pasar el estado inicial a una red neuronal para que aproxime la función de valor Q (función de recompensa), y devuelva el valor Q de todas las posibles acciones como salida, minimizando a la vez los errores TD. Finalmente realiza una propagación hacia atrás para encontrar los mejores valores Q. Es lo que llamamos una *deep Q-network* (DQN) o *deep Q-learning* (DQL).

2) Gradientes de política (*policy gradients*). Los métodos de gradiente de política determinan directamente la política. Por lo general son métodos *on-policy*, es decir, que el método aprende de las trayectorias generadas por la política actual. Una manera de obtener una aproximación de la política es maximi-

zando directamente la recompensa esperada usando métodos de gradiente (de aquí el nombre de gradiente de política).

3) Actor-crítico (*actor-critic*). Los métodos actor-crítico combinan los gradientes de política con funciones de valor, sacando partido de las propiedades y ventajas de ambos métodos.

A lo largo de los próximos módulos explicaremos con detalle cada uno de estos algoritmos, empezando por las DQN, luego los gradientes de política y finalmente los métodos actor-crítico.

2. Aproximación de la función de valor

2.1 Actualización y optimización de la función de valor

Para aplicar el gradiente a funciones de valor definimos la función objetivo $J(\mathbf{w})$ como la esperanza $\mathbb{E}$ en una política dada π:

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(s) - \hat{v}_\mathbf{w}(s))^2 \right] \tag{5}$$

donde $v_\pi(s)$ es el valor verdadero del estado s y $\hat{v}_\mathbf{w}(s)$ el valor estimado. Esencialmente es una función de pérdida *on-policy*.

Combinando la anterior ecuación 5 con la ecuación 4 obtenemos el descenso del gradiente:

$$\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_\mathbf{w} J(\mathbf{w}) = \alpha\mathbb{E}_\pi \left[(v_\pi(s) - \hat{v}_\mathbf{w}(s))\nabla_\mathbf{w}\hat{v}_\mathbf{w}(s) \right] \tag{6}$$

y en el caso estocástico:

$$\Delta\mathbf{w} = \alpha \left[v_\pi(s) - \hat{v}_\mathbf{w}(s) \right] \nabla_\mathbf{w}\hat{v}_\mathbf{w}(s) \tag{7}$$

2.2 Estimación lineal de la función de valor

Los métodos lineales aproximan la función de valor de estado $\hat{v}(s,\mathbf{w})$ mediante el producto interno entre el vector de pesos $\mathbf{w}$ y un vector de características $\mathbf{x}(s)$ con el mismo número de componentes que $\mathbf{w}$, que representa el estado s.

$$\mathbf{x}(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix} \tag{8}$$

donde $x_i(s)$ representa cada componente del vector $\mathbf{x}(s)$.

La aproximación lineal de la función de valor vendrá dada por:

$$\hat{v}(s,\mathbf{w}) = \mathbf{x}(s)^T\mathbf{w} = \sum_{i=1}^{n} x_i(s)w_i \tag{9}$$

y la función objetivo será cuadrática en $\mathbf{w}$:

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(v_\pi(s) - \mathbf{x}(s)^T\mathbf{w})^2\right] \tag{10}$$

Si aplicamos descenso del gradiente estocástico, el gradiente de aproximación de la función de valor respecto del vector de pesos $\mathbf{w}$ resultará una regla muy simple que convergerá en un óptimo global:

$$\nabla_{\mathbf{w}} v_{\mathbf{w}}(s) = \mathbf{x}(s) \tag{11}$$

y la actualización:

$$\Delta\mathbf{w} = \alpha(v_\pi(s) - \hat{v}_{\mathbf{w}}(s))\mathbf{x}(s) \tag{12}$$

es decir, el producto del tamaño de paso, la predicción del error y el vector de características. El mismo procedimiento seguiría para una función de acción-valor $q_\pi(s,a)$.

Un caso especial de los métodos lineales es precisamente las tablas de búsqueda de los métodos tabulares considerando un vector de características *one-hot* (es decir, que en cada estado siempre tendremos al menos uno de los valores del vector igual a 1 y los demás valores 0). Esto muestra que el método lineal es solo una generalización del caso tabular:

$$\mathbf{x}^{\text{tabla}}(s) = \begin{pmatrix} 1(s = s^1) \\ \vdots \\ 1(s = s^n) \end{pmatrix} \tag{13}$$

Usando esta representación tabular, podemos realizar el producto escalar entre el vector de características y los pesos, es decir, multiplicar por el peso correspondiente a cada estado en nuestro espacio de estados:

$$\hat{v}(s) = \begin{pmatrix} 1(s = s^1) \\ \vdots \\ 1(s = s^n) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} \tag{14}$$

Vemos claramente aquí el problema de los métodos tabulares, porque en el momento en que haya muchos estados, la matriz/vector será intratable.

El siguiente pseudocódigo muestra la implementación de la versión con descenso del gradiente del algoritmo de Montecarlo con una función lineal de aproximación:

Algorithm 1 Pseudocódigo del algoritmo de descenso del gradiente de Montecarlo para estimar $\hat{v} \approx v_\pi$

Entrada: política π a evaluar
Entrada: una función diferenciable $\hat{v} : \mathbb{S}x\mathbb{R}^d \rightarrow \mathbb{R}$
Parámetro del algoritmo: paso $\alpha > 0$
Inicializar los pesos $\mathbf{w} \in \mathbb{R}$ de la función de valor arbitrariamente (p.ej., $\mathbf{w} = 0$)
for cada episodio **do**
  Generar un episodio $S_0, A_0, R_1, S_1, A_1, ...R_T, S_T$ usando π
  for cada paso del episodio, $t = 0,1,...T-1$: **do**
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha[G_t - \hat{v}(S_t,\mathbf{w})]\nabla\hat{v}(S_t,\mathbf{w})$
  end for
end for

2.3 Estimación no lineal de la función de valor

Como se comentaba en el apartado anterior, una alternativa a la dificultad de no conocer todas las características de los estados y los detalles del problema, es utilizar estimadores de función no lineales capaces de partir directamente de los estados, como son las redes neuronales. Si en el caso lineal las funciones de aproximación $\hat{v}$ y $\hat{q}$ venían definidas por un vector de parámetros o pesos $\mathbf{w}$, en el caso no lineal este mismo vector $\mathbf{w}$ representará los pesos de conexión en todas las capas de la red neuronal artificial.

En los próximos módulos nos centraremos únicamente en la fusión de las CNN en DRL, en concreto en el ámbito del Q-*learning*. La evolución de esta unión del DL con el RL en los últimos años ha tenido un gran impacto en el aprendizaje por refuerzo profundo, ofreciendo un amplio abanico de aplicaciones en muchos campos.

Resumen

En este módulo hemos introducido el concepto de «solución aproximada» para poder resolver problemas de aprendizaje por refuerzo en situaciones en las que el espacio de estados es demasiado grande para calcular el valor de cada estado y almacenarlo todo en una tabla, como hacíamos en *Q-learning*. Al problema del almacenaje se le suman también las limitaciones computacionales y el hecho de que llega un momento en que es imposible conocer todos los estados posibles.

Las funciones de aproximación nos permiten abordar la situación generalizando a partir de unos pocos ejemplos. Construimos una aproximación de la función que se busca (ya sea la de valor o la de política) según unas primeras experiencias, y luego se va actualizando y perfeccionando conforme se añaden experiencias nuevas y recompensas de las acciones tomadas en cada estado. Cuando esta función de aproximación es una red neuronal hablamos del aprendizaje por refuerzo profundo, en inglés, *deep reinforcement learning* (DRL), en el que se unen los métodos del *deep learning* del aprendizaje supervisado y los del aprendizaje por refuerzo.

Distinguimos tres métodos en DRL: las redes profundas Q (DQN), los gradientes de política (PG) y los actor-crítico (AC), según si aproximan la función de valor, la política, o ambas, respectivamente.

Glosario

ANN *m* Véase **red neuronal artificial**.

aprendizaje automático *m* Disciplina de las ciencias de computación que da a los ordenadores la capacidad de aprender sin ser explícitamente programados.
sigla **ML**
en machine learning

aprendizaje profundo *m* Conjunto de algoritmos de aprendizaje automático que intenta modelar abstracciones de alto nivel utilizando redes neuronales de múltiples capas o niveles.
sigla **DL**
en deep learning

artificial neural network *m* Véase **red neuronal artificial**.

CNN *m* Véase **red neuronal convolucional**.

convolutional neural network *m* Véase **red neuronal convolucional**.

DL *m* Véase **aprendizaje profundo**.

deep learning *m* Véase **aprendizaje profundo**.

i.i.d. *m* Véase **independientes e idénticamente distribuidos**.

independent and identically distributed *m* Véase **independientes e idénticamente distribuidos**.

independientes e idénticamente distribuidos *m* Conjunto de datos o variables aleatorias que son mutuamente independientes entre sí y además cada variable tiene la misma distribución de probabilidad.
sigla **i.i.d.**
en independent and identically distributed

ML *m* Véase **aprendizaje automático**.

machine learning *m* Véase **aprendizaje automático**.

recurrent neural network *m* Véase **red neuronal recurrente**.

red neuronal artificial *m* Modelo computacional compuesto de un conjunto de unidades (neuronas) conectadas entre sí, y que permite procesar una información proporcionando unos valores de salida.
sigla **ANN**
en artificial neural network

red neuronal convolucional *m* Tipo de red neuronal aplicada a matrices bidimensionales, normalmente usada en tareas de visión artificial.
sigla **CNN**
en convolutional neural network

red neuronal recurrente *m* Tipo de red neuronal consistente en ciclos de realimentación que permiten retener la información durante algunos pasos o épocas de entrenamiento, usualmente aplicada a series temporales, listas y textos.
sigla **RNN**
en recurrent neural network

RNN *m* Véase **red neuronal recurrente**.

Bibliografía

Bosch, A.; Casas, J., Lozano, T. (2019). *Deep Learning. Principios y fundamentos*. Universitat Oberta de Catalunya.

Ganin, Y.; Kulkarni, T.; Babuschkin, I.; Eslami, S.; Vinyals, O. (2018). *Synthesizing Programs For Images Using Reinforced Adversarial Learning*.

McCulloch, W. S.; Pitts, W. (1988). *Neurocomputing: Foundations of research*. Chapter «A Logical Calculus of the Ideas Immanent in Nervous Activity» (págs. 15-27). Cambridge: MIT Press. Reedición de 1943 de *Bulletin of Mathematical Biophysics*, vol. 5, págs. 115-133.

Hausknecht, M. J.; Stone, P. (2017). *Deep Recurrent Q-Learning For Partially Observable MDPs*. Austin: University of Texas.

Hinton, G. E.; Osindero, S.; Teh, Y. W. (2006). «A fast learning algorithm for deep belief nets». *Neural Comput.*, vol. 18, n.º 7, págs. 1527-1554.

Sutton, R. S.; Barto, A. G. (2019). *Reinforcement Learning. An introduction*. Cambridge, MA: MIT Press.

Zhang, A.; Lipton, Z. C.; Li, M.; Smola, A. J. *Dive into Deep Learning*. `<https://d2l.ai/>`

Zhou, Z.; Li, X.; Zare, R. N. (2017). «Optimizing Chemical Reactions with Deep Reinforcement Learning». *ACSCentral Science3*.