COMPUTATION E INFORMATICA

Recibido 01 Feb. 2023
Aceptado 20 Feb. 2023

ReCIBE, Año 12 No. 1, May. 2023

Una introducción amable pero rigurosa al aprendizaje por refuerzo

A gently but rigorous introduction to reinforcement learning

Mauro Alejandro Montenegro Meza[1*]
m.montenegro.meza@gmail.com

Rolando Menchaca Méndez[1]
rolando.menchaca@gmail.com

Ricardo Menchaca Méndez[1]
ricardo.menchaca@gmail.com

Centro de Investigación en Computación del IPN[1]

RESUMEN

La interacción con el mundo es una de las principales formas en las que se genera el aprendizaje, pues es el medio por el cuál se obtiene información del entorno, y se experimentan relaciones causa-efecto. Esta idea de aprender mediante la interacción es una aspecto fundamental en muchas teorías del aprendizaje y, en este artículo abordaremos un enfoque computacional llamado *aprendizaje por refuerzo* (*Reinforcement Learning*, RL) además de construir de manera progresiva y sencilla sus bases matemáticas, así como los métodos principales de solución. Por último, mostramos aplicaciones y algoritmos que son relevantes en la industria e investigación.

Palabras clave: Aprendizaje por Refuerzo, Aprendizaje de Máquina, Inteligencia Artificial.

ABSTRACT

The interaction within the world constitutes one of the main ways in which learning is generated, as it is the way by which we obtain information from the environment and we experience cause-effect relationships. This idea of learning through interaction is a fundamental issue in many learning theories and, in this paper, we will address a computational approach called Reinforcement Learning (RL) and we will build in a progressive and simple way its mathematical basis, as well as its main solution methods. Lastly, applications and algorithms that are relevant in the industry and research are presented.

Keywords: Reinforcement Learning, Machine Learning, Artificial Intelligence.

1. INTRODUCCIÓN

En la literatura se denomina como "*aprendizaje por refuerzo*" (*Reinforcement Learning*, RL) tanto a una clase de problemas, a una clase de métodos que abordan dichos problemas, así como al campo que los estudia en el contexto del "*aprendizaje de máquina* "(*Machine Learning*, ML). El objetivo del RL no es generalizar situaciones no vistas durante el entrenamiento como en el *aprendizaje no supervisado*, ni encontrar estructuras escondidas entre los datos no etiquetados como en el *aprendizaje no supervisado*; en esencia, el objetivo principal del RL es aprender *políticas de control* que le permitan a un *agente* interactuar de manera exitosa en el *ambiente* que lo contiene. A pesar de las diferencias, las arquitecturas de RL pueden incorporar componentes de ambos paradigmas para mejorar su desempeño.

El estudio del RL tiene como fundamentos el comportamiento basado en estímulos y el desarrollo de la teoría del control. Sus bases principales son los estudios de comportamiento basado en estímulos de Pavlov, las *ecuaciones de Bellman*, y una versión estocástica del problema de control que modela la dinámica de problemas de toma de decisiones secuenciales por medio de *procesos de decisión de Markov* (*Markov Decision Process*, MDP).

2. PROCESO DE DECISIÓN DE MARKOV (MDP)

Como se muestra en la Figura 1, la arquitectura general de un sistema de RL está compuesta por un **agente** que interactúa con un **ambiente** a través de un conjunto de **acciones** que se seleccionan siguiendo una **política** de control π. El ambiente puede estar en cualquiera de los estados s definidos en su espacio de estados S y en cada instante t de tiempo discreto experimenta una transición a un nuevo estado $s' \in S$. La política de control $\pi : S \rightarrow A$ es una función que determina la acción que toma el agente dado un estado s. Cada que el agente realiza una acción a definida en el espacio de acciones A, se obtiene una señal de recompensa r_t.

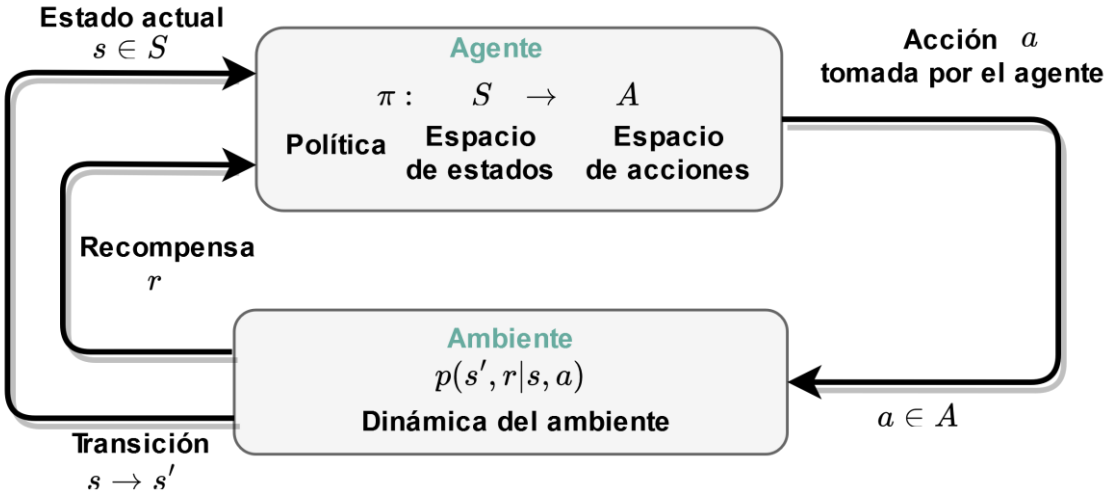

Figura 1. Arquitectura general de un sistema RL.

La dinámica de los sistemas de RL (ambiente + agente) se modela como un proceso de decisión de Markov (MDP) donde se asume que la probabilidad $p(s', r|s, a)$ de transitar del estado s al estado s' y obtener una recompensa r, solo depende del estado actual s y de la acción a tomada por el agente. Esto queda plasmado en la Ec.1 que nos dice que la probabilidad de que el siguiente estado en que estará el ambiente sea s_{t+1} y que se reciba una recompensa igual a r_{t+1} está condicionada al estado s_t en el que está el ambiente en el tiempo t, así como a la acción a_t tomada por el agente también el tiempo t. Esto significa que el sistema *no tiene memoria* y que por lo tanto la historia pasada no tiene injerencia en la transición actual. Es importante mencionar que, como se verá más adelante, los sistemas de RL son capaces de encontrar políticas de control óptimas aún y cuando se desconozca la dinámica del sistema.

$$p(s', r|s, a) = Pr\{r_{t+1} = r, s_{t+1} = s'|s_t, a_t\} \qquad (1)$$

El objetivo de los algoritmos de entrenamiento de RL es encontrar una política de control π que maximice el valor esperado de la recompensa acumulada G_t que se define como la suma de las recompensas individuales r_t recibidas en cada instante de tiempo discreto t. Esto quiere decir que los sistemas de RL buscan seleccionar las acciones de control que maximicen la suma de las recompensas a largo plazo y no necesariamente la recompensa inmediata.

Para cuantificar la efectividad de una política π, se utiliza una *función de valor* de los estados $V^{\pi}(s)$, que precisamente determina el valor esperado de la recompensa que se obtendrá si el ambiente parte del estado s y el agente sigue la política de control π. Esta idea es expresada matemáticamente por medio de la Ec. 1 donde $E[G_t|s_t = s]$ se lee como el valor esperado de G_t dado que el ambiente está en el estado s_t. Las r_{t+k+1} son las recompensas que el agente obtendrá desde el tiempo $t + 1$ hasta el fin de la historia. La constante γ utilizada en la Ec. 2, llamada *factor de descuento*, sirve para atenuar la importancia de las recompensas futuras de las cuáles se tiene menos certeza. La Ec.2 se escribe en términos de un valor esperado debido a que tanto el ambiente como el agente pueden tener comportamientos aleatorios.

$$V^{\pi}(s) = E[G_t|s_t = s] = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s] \qquad (2)$$

De manera similar, la Ec. 3 define la función de valor de los pares estado-acción $Q^{\pi}(a, s)$ (*función estado-acción*) como el valor esperado de la recompensa acumulada al tomar la acción a en el

estado s al tiempo t, y posteriormente seguir la política π. En este caso, la función de estado-acción nos permite saber cuál será la recompensa acumulada esperada si es que tomamos una acción a_t en particular y no necesariamente la definida por π.

$$Q^{\pi}(s,a) = \mathbb{E}[G_t|s_t = s, a_t = a] = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a] \qquad (3)$$

La Ec. 4 muestra la relación entre $Q^{\pi}(s,a)$ y $V^{\pi}(s)$ donde $\pi(a|s)$ es la probabilidad de tomar la acción a dado que el ambiente está en el estado s y se sigue la política π. La Ec. 4 nos indica que $V^{\pi}(s)$ puede verse como el valor esperado de las recompensas a futuro cuando se está en el estado s y se seleccionan acciones siguiendo la política π.

$$V^{\pi}(s) = \sum_{a} \pi(a|s) * Q^{\pi}(s,a) \qquad (4)$$

Una forma de calcular de manera exacta los valores de $Q^{\pi}(s,a)$ y $V^{\pi}(s)$ es por medio de las *ecuaciones de Bellman*, Ec. 5 para la función de valor de los estados y Ec. 6 para la función de estado-acción. En ambas ecuaciones el valor esperado de la recompensa acumulada se calcula en términos del valor esperado de la recompensa r obtenida durante la transición de s a los posibles siguientes estados s', más la formulación recursiva del valor esperado de la recompensa acumulada a partir de ese momento ($V^{\pi}(s')$ o $Q^{\pi}(s',a')$ según sea el caso), siguiendo la política π. En ambas ecuaciones, la sumatoria $\sum_{s',r}$ indica que se están considerando todos los posibles siguientes estados s', así como todos los posibles valores de la recompensa r. Como se ilustra en la Figura 2, la Ec. 5 calcula el valor esperado de las recompensas que se obtendrán siguiendo la política π considerando todas las formas en que el ambiente transitará entre sus diferentes estados como resultado de las acciones de control tomadas por el agente. El caso de la Ec. 6 es análogo, sólo que se asume que se comienza seleccionando una acción a en particular.

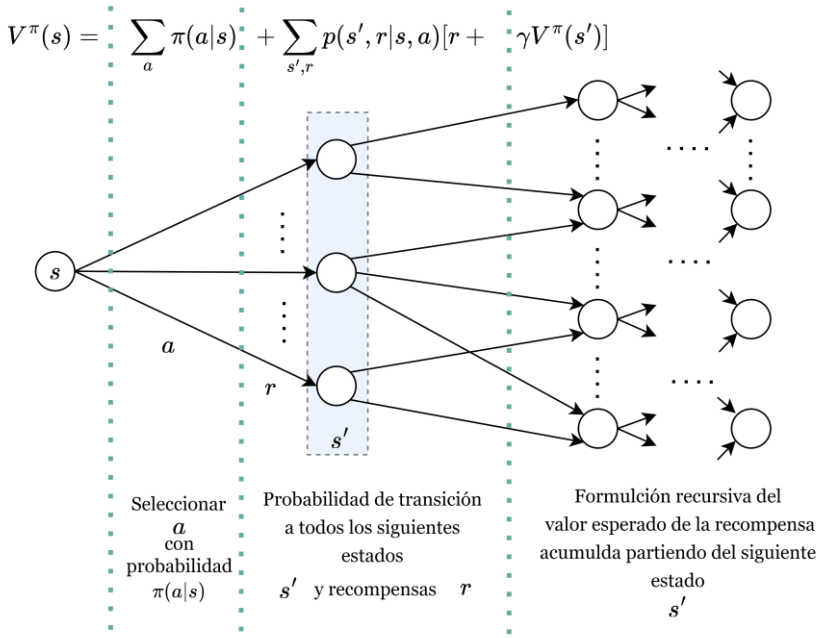

Figura 2. Las ecuaciones de Bellman consideran todas las formas en que puede evolucionar el ambiente dado que el agente sigue la política π.

$$V^{\pi}(s) = \sum_a \pi(a|s) * \sum_{s'} P^a_{s,s'}(r(s,a) + \gamma V^{\pi}(s') \qquad (5)$$

$$Q^{\pi}(s,a) = \sum_{s'} P^a_{s,s'}(r(s,a) + \gamma \sum_{a'} \pi(a'|s') Q^{\pi}(s',a')) \qquad (6)$$

Una variante de las Ecs. de Bellman, conocidas como *Ecuaciones de optimalidad de Bellman*, pueden ser utilizadas para calcular la *política de control óptima* (π^*), es decir, aquella que maximiza la recompensa esperada de todos los estados en los que puede estar el ambiente. Más formalmente, decimos que una política π es mejor que otra π' si, y solamente si, $V^{\pi}(s) \geq V^{\pi'}(s)$ $\forall s \in S$. De esta forma, una política óptima se define como aquella que satisface $\pi^* \geq \pi$ para

todas las políticas π. Las funciones de valor óptimas $V^*(s)$ y $Q^*(s,a)$ se obtienen al seleccionar la acción a que maximice el valor esperado de la recompensa a largo plazo. Esto queda de manifiesto en las Ecs. 7 y 8 que en lugar de seguir una política arbitraría π, simplemente seleccionan la acción a que maximice el valor esperado.

$$V^*(s) = \max_a \sum_{s'} P^a_{s,s'}(r(s,a) + \gamma V^*(s')) \qquad (7)$$

$$Q^*(s,a) = \sum_{s'} P^a_{s,s'}(r(s,a) + \gamma \max_{a'} Q^*(s',a')) \qquad (8)$$

Buscar la solución a las ecuaciones de optimalidad de Bellman constituye uno de los problemas centrales del aprendizaje por refuerzo. Esto se debe a que a partir las funciones de valor óptimas es fácil derivar una política de control óptima. Simplemente se elije la acción que lleve al siguiente estado con mayor valor. En las secciones siguientes se presentan algunos de los métodos más representativos.

2. MÉTODOS TABULARES

En el método de programación dinámica (*Dynamic Programming*, DP) el valor de cada estado $V^{\pi}(s)$ se calcula a partir de la estimación actual del valor de sus estados vecinos. Dos estados s y s' son vecinos, si la probabilidad de que el ambiente transite en un solo paso de s a s' es mayor a cero, es decir si $p(s', r|s, a) > 0$ para cualquier acción a y recompensa r.

Todos los métodos tabulares que estudiaremos aquí operan siguiendo un proceso llamado "*Iteración general de políticas*" (*General Policy Iteration*, GPI) que consiste en iterativamente realizar los procesos de "*Evaluación de política*" (*Policy Evaluation*, PE) y "*Mejorado de política*"(*Policy Improvement*, PI). A grandes rasgos, el objetivo de PE es calcular la función de valor de la política actual y el de PI es obtener una mejor política a partir de dicha información.

El proceso de PE consiste en utilizar la Ec. 9 de manera iterativa para actualizar el valor actual de la función de valor. En la práctica, este proceso se realiza hasta que el cambio en los valores de los estados de dos iteraciones consecutivas sea menor a una cota ψ pre-establecida. Es importante notar que la Ec. 9 es una adecuación de la Ec. 5 utilizada como una regla de actualización. La demostración de convergencia de este procedimiento se basa en el hecho de que la Ec. 9 es un "*mapeo de contracción*" (*Contraction mapping*) y por lo tanto tiene un único punto fijo.

$$v_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a[r + \gamma v^{\pi}(s')]) \qquad (9)$$

Por su parte, el paso de PI consiste en utilizar la ecuación de optimalidad de Bellman para definir una nueva política π' a partir de la función de valor de la política actual π. Como puede verse en la Ec. 10, π' es una versión voraz (*greedy*) de π que selecciona la acción a que maximice la recompensa esperada. Finalmente, observe que el caso en que π' es igual π, la Ec. 10 se convierte en la ecuación de optimalidad de Bellman y por lo tanto $\pi = \pi'$ es una política de control óptima. La Figura 3 ilustra la forma en que los procesos de PE y PI interactúan para moverse tanto en el espacio de las funciones de valor, como en el espacio de las políticas, para dirigirse a la política óptima π^*.

$$\pi'(s) = \arg\max_{a} \sum_{s',r} p(s', r|s, a)[r + \gamma v^{\pi}(s')] \qquad (10)$$

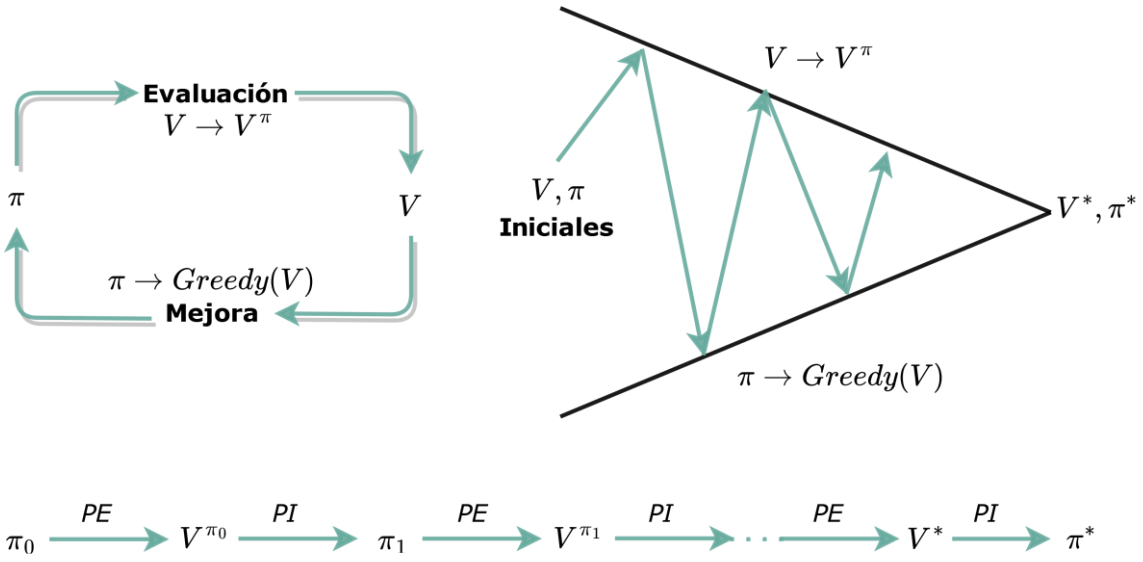

Figura 3. Convergencia a una política óptima en el proceso de Iteración General de Política.

Esta versión de PE tiene dos desventajas. Requiere conocer la función $p(s', r \mid s, a)$ que rige la dinámica del ambiente y puede requerir un número elevado de pasos para converger. Afortunadamente, las propiedades de convergencia de GPI a una política óptima no dependen de calcular de manera exacta el valor de las políticas. Partiendo de este hecho, el algoritmo de "*Iteración de valor*" (*Value Iteration*, VI) trunca la evaluación de la política a un solo paso y utiliza la Ec. 11 para combinar PE y PI en un único paso.

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a)] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \end{aligned} \tag{11}$$

A pesar de sus limitantes, los algoritmos clásicos de DP son importantes teóricamente y otros métodos tratan de emularlos pero de manera más eficiente.

Por su parte, los métodos *Monte Carlo* (MC) no requieren un modelo del ambiente y son capaces de aprender políticas por medio de la experiencia. De manera general, en los métodos MC el agente interactúa con el ambiente siguiendo una política π, almacenando los estados visitados, las acciones tomadas y las recompensas obtenidas en cada transición entre estados. En el caso de procesos episódicos, como una partida de Ajedrez, el proceso se repite varias veces. Al final,

se calculan estimados de la función $V^{\pi}(s)$ de cada estado visitado como un promedio ponderado basado en las recompensas acumuladas G_t experimentadas durante cada episodio.

En el caso de procesos episódicos (como una partida de Ajedrez), el proceso se repite N veces y, en cada iteración se calcula un promedio de las funciones. Existen dos algoritmos parecidos pero con propiedades teóricas particulares usados en los métodos MC: Cuando se promedia solamente la primera visita a cada estado (*algoritmo first visit*) o cuando se promedia cada vez que el estado es visitado en un episodio (*every visit*).

En los métodos MC la actualización de las funciones de valor se realiza mediante la Ec. 12 que calcula el valor de los estados como un promedio ponderado basado en la recompensa durante cada episodio (G_t).

$$V(S_t) = V(S_t) + \frac{1}{N}\alpha[G_t - V(S_t)] \qquad (12)$$

Debido a que en MC no se conoce el modelo del ambiente, es necesario estimar explícitamente el valor de cada acción para poder elegir la mejor de ellas. De forma similar a DP, en MC el paso de mejora de política se realiza por medio de la Ec. 13, sólo que en este caso se trabaja con funciones $Q(s, a)$ que capturan el valor de cada par estado-acción.

$$\pi(s) = \arg\max_a q(s, a) \qquad (13)$$

La principal ventaja de los métodos MC sobre los basados en DP es que son capaces de encontrar políticas de control optimizadas, aún y cuando no se tenga ningún conocimiento del ambiente. Lo anterior mediante un proceso en el cuál el agente experimenta las consecuencias, en términos de la recompensa obtenida, de tomar diferentes acciones en los estados que visita. Es importante mencionar que el cálculo de las políticas contempla la suma acumulada de las recompensas a largo plazo y no sólo las recompensas instantáneas de la acción inmediata. Desafortunadamente, para converger a una política óptima, los métodos MC requieren que el agente experimente un gran número de veces todos los pares estado-acción.

Una idea que combina los métodos MC y DP son los métodos de "*Diferencias temporales*" (*Temporal Difference*, TD). Los métodos TD aprenden directamente de la experiencia sin la necesidad de un modelo y actualizan sus estimados con base en otros estimados aprendidos anteriormente. El método TD más simple es conocido como $TD(0)$ que, como se muestra en la Ec. 14, actualiza la función de valor $V(S_t)$ en cada transición del ambiente a partir de la recompensa inmediata R_{t+1}, más el estimado actual del valor del siguiente estado $V(S_{t+1})$.

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1} - V(S_t))] \qquad (14)$$

Como MC, TD no requiere un modelo del ambiente, y como DP, TD aprovecha los estimados actuales de los valores de otros estados para actualizar inmediatamente el estimado del valor del estado actual.

Dentro de los métodos TD más populares están *SARSA* que utiliza la Ec. 15 para actualizar la función estado-acción, y *Q-Learning* que actualiza la función estado-acción acorde a la Ec. 16.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, a) - Q(S_t, At)] \qquad (15)$$

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, At)] \qquad (16)$$

La diferencia entre SARSA y Q-Learning es sutil pero importante. SARSA es un método del tipo"*en la política*" (*on-policy*) que utiliza la misma política para evaluar estados-acciones y para tomar acciones de control, mientras que Q-Learning es un método del tipo "*fuera de la política*" (*off-policy*) que utiliza una política para evaluar estados-acciones y otra para tomar acciones de control. Normalmente, ambos algoritmos utilizan una política de control ϵ-*voraz* (ϵ-*greedy*) que selecciona una acción al azar con probabilidad ϵ, y que con una probabilidad de $1 - \epsilon$ se comporta de manera voraz seleccionando la mejor acción de acuerdo al estimado actual de la función de estado-acción. Para actualizar la función de estado-acción SARSA utiliza una política ϵ-*greedy*, mientras que Q-Learning utiliza una política voraz ($\epsilon = 0$). El objetivo de las políticas ϵ-*voraz* es permitir que el agente *explote* su conocimiento la mayor parte del tiempo (con probabilidad $1 - \epsilon$) pero que algunas veces (con probabilidad ϵ) *explore* acciones que tal vez no haya usado con anterioridad.

La Figura 4 presenta un resumen de los métodos tabulares mostrando sus pseudo códigos.

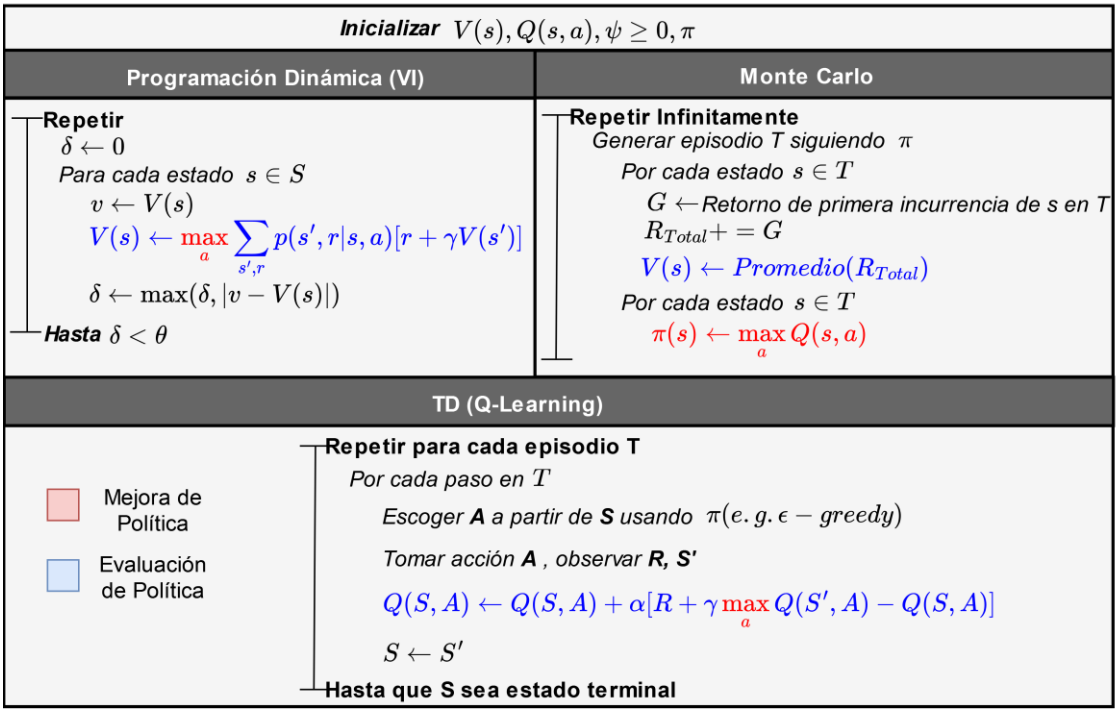

Figura 4. Pseudocódigo de métodos tabulares.

3. MÉTODOS PARAMÉTRICOS

Los métodos paramétricos intentan solventar la falta de escalabilidad de los métodos tabulares por medio de *aproximaciones* y *generalización*. La idea consiste en utilizar modelos cuyos parámetros son optimizados para aproximar las funciones de valor (estado o estado-acción) a partir de las recompensas obtenidas al interactuar con el ambiente. Típicamente, el número de parámetros es mucho menor que el número de pares estado-acción. Una de las principales ventajas de estos modelos, como las redes neuronales artificiales (ANN), es su capacidad de generalización que permite tener estimaciones de valores para pares estado-acción aún y cuando el agente no los haya experimentado en el pasado.

En términos más concretos, las funciones aproximadas cuentan con un vector de parámetros $w \in R^n$, donde $\bar{v}_\pi(s; \vec{w})$ es el valor aproximado del estado s dado un vector de parámetros $\vec{w}$, cuando se sigue la política π, es decir $\bar{v}_\pi(s; \vec{w}) \approx v_\pi(s)$.

El proceso de entrenamiento del modelo paramétrico consiste en optimizar el vector de parámetros $\vec{w}$ para minimizar el error observado en los ejemplos experimentados por el agente durante su interacción con el ambiente.

Por su simplicidad y efectividad práctica, uno de los métodos más usados para optimizar el vector de parámetros es el *Gradiente Descendente Estocástico* (*Stochastic Gradient Descent*, SGD), el cuál, ajusta los valores de $\vec{w}$ después de cada actualización de la recompensa acumulada G_t en la dirección que reduciría más el error $G_t - \bar{v}(S_t; \vec{w}_t)$ en ese ejemplo en particular.

La Ec. 17 muestra la regla de actualización de los pesos utilizando SGD, donde $\nabla\bar{v}(S_t; \vec{w}_t)$ denota el vector de derivadas parciales de $\bar{v}$ respecto a los parámetros $\vec{w}_t$, llamado gradiente de $\bar{v}$ respecto a $\vec{w}_t$.

$$\vec{w}_{t+1} = \vec{w}_t + \alpha[G_t - \bar{v}(S_t; \vec{w}_t)]\nabla\bar{v}(S_t; \vec{w}_t) \qquad (17)$$

Uno de los métodos paramétricos más populares es una adaptación de Q-Learning, llamada Deep Q-Learning (DQN), que utiliza una red neuronal profunda (*Deep Neural Network*, DNN) como modelo de aproximación a las funciones estado-acción. La actualización de las funciones de estado-acción se realiza mediante la Ec. 16, pero sustituyendo los estimados de las funciones de estado-acción por un modelo con parámetros $\vec{w}$. La actualización de estos parámetros se realiza sustituyendo en la Ec. 17 el valor de G_t por la función objetivo $r + \gamma \max_a \bar{v}(S_{t+1}; \vec{w}_t)$.

En este caso, debido a que se utiliza el mismo vector de parámetros $\vec{w}$ para realizar la predicción $Q(s, a; \vec{w})$ y para evaluar la función *Q objetivo* $r + \gamma \max_a Q(s', a; \vec{w})$ existe el riesgo de crear oscilaciones durante el entrenamiento. Para minimizar el impacto de este problema se utiliza la Ec. 18 en donde se introduce una segunda DNN con parámetros $\vec{w}'$, llamada "*red objetivo*" (*target network*), para evaluar la función *Q objetivo*. Estos parámetros $\vec{w}'$ son actualizados periódicamente mediante copia de los parámetros $\vec{w}$ de la red principal.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[r + \gamma \max_{a'} Q(s', a; \vec{w'}) - Q(s, a; \vec{w})] \qquad (18)$$

Otro aspecto relevante a considerar en el contexto del uso de ANN en RL es el hecho de que la secuencia de recompensas experimentadas por un agente al interactuar con el ambiente están altamente correlacionadas, y por lo tanto, no deben ser usadas directamente para entrenar una ANN. Esto se debe a que los algoritmos de entrenamiento suponen que los ejemplos de entrenamiento son independientes e idénticamente distribuidos (*iid*). DQN aborda este problema por medio de una memoria donde almacena las experiencias pasadas, llamada *memoria de experiencia*, y que es muestreada aleatoriamente para obtener ejemplos de entrenamiento. Esta técnica es llamada *búfer de repetición* o *búfer de experiencia* y almacena gradualmente en un búfer las túplas de experiencias (S, A, R, S') al interactuar con el ambiente. Al acto de muestrear de

manera aleatoria pequeños conjuntos de tuplas del *búfer de experiencia* y utilizarlos en el aprendizaje se le llama *repetición de experiencia*. Además de romper éstas correlaciones entre las tuplas, el uso de ésta técnica permite aprender más acerca de ejemplos individuales al ser posible muestrearlos más de una vez y hacer mejor uso de ésta experiencia.

4. APLICACIONES EXITOSAS DE RL

En el ámbito científico, el primer gran éxito del RL fue *TD-Gammon* (Tesauro, 1995) que en esencia, es un programa computacional que utiliza una forma de TD ($\mathrm{TD}\ -\gamma$) en el entrenamiento de una ANN para aprender a jugar de manera profesional Backgammon. En 2013, se presenta DQN para resolver problemas con un número elevador de estados como los juegos de Atari (Mnih et al., 2013). *AlphaGo* (Silver et al., 2016), desarrollado por DeepMind, aprende a jugar un juego altamente complejo como el Go mediante un *árbol de búsqueda monte carlo* y una red de política entrenada con *gradiente de política*. En 2020 se crea *AlphaZero* (Silver et al., 2018), una mejora de *AlphaGo*, que generaliza a juegos como el ajedrez y no utiliza información de expertos. Más recientemente, en 2022 *AlphaTensor* (Fawzi et al., 2022) fue capaz de descubrir algoritmos más eficientes a los anteriormente conocidos para multiplicación de matrices.

En la industria se encuentran aplicaciones que utilizan un esquema de GPI con restricciones y una arquitectura DQN en la optimización de sistemas de refrigeración para centros de datos (Luo et al., 2022). Por otra parte, Netflix utiliza SARSA y Q-Learning en sus nuevos sistemas de recomendación (Elahi, 2022), mientras que IBM los emplea en sus sistemas de predicción de mercado. En el área médica se utilizan políticas entrenadas mediante Q-Learning en la optimización de *regímenes de tratamiento dinámico* (DTR). En la Figura 5 se presenta una línea de tiempo con algunas de las aplicaciones exitosas de RL.

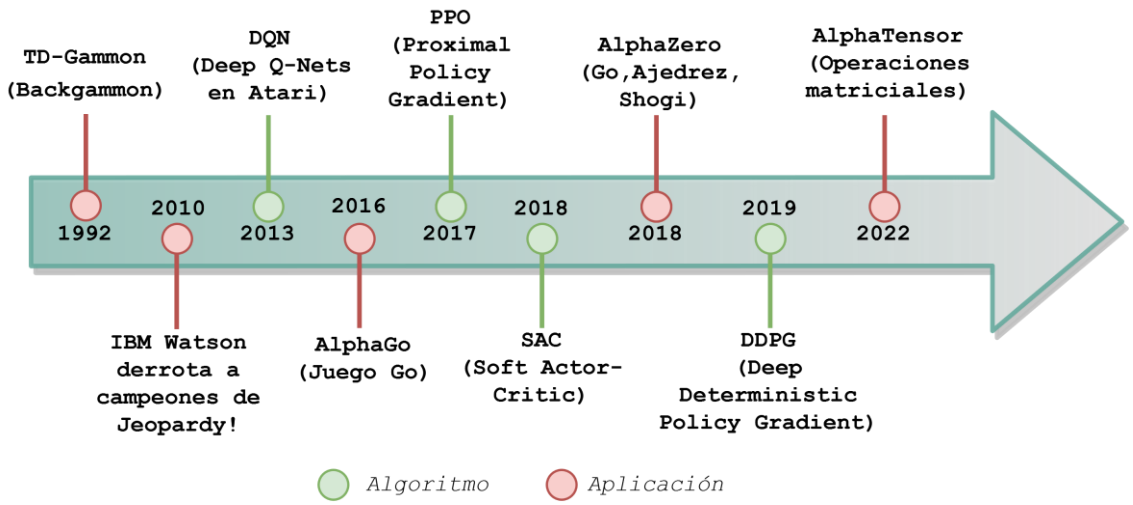

Figura 5. Linea de tiempo de trabajos sobresalientes en RL.

5. CONCLUSIONES

De forma similar a cómo los animales aprendemos, el proceso de aprendizaje en RL consiste en registrar las consecuencias que un agente experimenta al realizar acciones de control que afectan el ambiente que lo contiene. Conforme el agente acumula experiencias, los algoritmos de RL calculan políticas de control que maximizan el valor esperado de la cantidad de recompensas que el agente recibirá a lo largo del tiempo. En RL, los agentes se encuentran en una constante disyuntiva entre *explotar* el conocimiento adquirido que se encuentra codificado en la política de control actual para maximizar la recompensas y *explorar* acciones de control no utilizadas en el pasado con la esperanza de encontrar una mejor política de control.

Los algoritmos para calcular políticas de control se pueden clasificar en *métodos tabulares* (e.g., DP,MC,TD) en donde se almacenan las funciones de valor en tablas y en *métodos paramétricos* (e.g., DQN) que utilizan modelos parametrizados para aproximar dichas funciones. El progreso del RL en los últimos años ha sido notable y se espera que esta tendencia continúe tanto en el ámbito científico como el industrial. Al lector interesado en profundizar este tema se le recomiendan los libros de Sutton & Barto (Sutton & Barto, 2018) y de Bertsekas (Bertsekas, 2012).

6. REFERENCIAS BIBLIOGRÁFICAS

Bertsekas, D. (2012). *Dynamic programming and optimal control: Volume i* (Vol. 1). Athena scientific.

Elahi, E. (2022). *Reinforcement learning for budget constrained recommendations.* Retrieved 14 January 2023, from https://netflixtechblog.com/reinforcement-learning-for-budget-constrained-recommendations-6cbc5263a32a

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., … others (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, *610*(7930), 47–53.

Luo, J., Paduraru, C., Voicu, O., Chervonyi, Y., Munns, S., Li, J., … others (2022). Controlling commercial cooling systems using reinforcement learning. *arXiv preprint arXiv:2211.07357*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., … others (2016). Mastering the game of go with deep neural networks and tree search. *nature*, *529*(7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., … others (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tesauro, G. (1995). Td-gammon: A self-teaching backgammon program. *Applications of neural networks*, 267–285.