UNIVERSITAT POLITÈCNICA DE VALÈNCIA | IN-RED 2023 IX Congreso de Innovación Educativa y Docencia en Red

Congreso In-Red 2023
UPV, 13 – 14 de julio de 2023
Doi: https://doi.org/10.4995/INRED2023.2023.16617

Enseñanza del aprendizaje por refuerzo con un sencillo ejemplo de minimización de funciones*

Roger Arnau[1], Luis M. García-Raffi[2], José M. Calabuig[3] y Enrique A. Sánchez Pérez[4]

[1] Universitat Politècnica de València, ararnnot@posgrado.upv.es
[2] Universitat Politècnica de València, lmgarcia@mat.upv.es
[3] Universitat Politècnica de València, jmcalabu@mat.upv.es
[4] Universitat Politècnica de València, easancpe@mat.upv.es

How to cite: R. Arnau, L.M García-Raffi. J. M. Calabuig, E. A. Sánchez Pérez. 2023. Aprendizaje por refuerzo para minimizar funciones. En libro de actas: *IX Congreso de Innovación Educativa y Docencia en Red*. Valencia, 13 – 14 de julio de 2023.
Doi: https://doi.org/10.4995/INRED2023.2023.16617

Abstract

This paper presents a practical session for university students to introduce them on the fundamentals of reinforcement learning (RL). The difficulty of this technique means that it is not studied, so a simplification of RL is presented, which is applied to the solution of an optimization problem. In addition to this technique, we study how to approach the optimization problem as a game, since this is a natural application of RL.

Keywords: *Reinforcement learning, artificial intelligence, neural networks, model.*

Resumen

En este trabajo se presenta una sesión a modo de taller orientada al estudiantado universitarios para que entiendan los fundamentos del aprendizaje por refuerzo (RL). Esta técnica de inteligencia artificial no es comúnmente estudiada por su dificultad, por ello se expone una simplificación del RL, que se aplica a la resolución de un problema de optimización. Además se analizará la manera de abordar el problema de optimización como un juego, puesto que este es una aplicación natural del RL.

Keywords: *Aprendizaje por refuerzo, inteligencia artificial, redes neuronales, modelo.*

*Proyectos financiados por la Universitat Politècnica de València (PAID-01-21), el Ministerio de Ciencia e Innovación (PID2020-112759GB-I00) y Polish National Agency for Strategic Partnership (BPI/PST/2021/1/00031/U/00001).

1 Introducción

El impacto social y mediático del Aprendizaje Automático (ML) es innegable. La ciudadanía no suele estar informada sobre las técnicas de ML y cuando lo está solo lo es al respecto de las consecuencias presentes y futuras que la implantación de las mismas en los diferentes ámbitos (económico, social, legal...) puede tener sobre su vida diaria. Sí que es cierto que cada vez más podemos encontrar informaciones que ligan estas técnicas con las matemáticas, mostrando a las claras que su desarrollo y aplicación pertenece al ámbito de las ciencias (no solo las matemáticas, sino también la estadística, la ciencia de datos...). Este es el marco general social en el cual nuestro alumnado llega a los diferentes grados de las enseñanzas técnicas: podemos hablar de un cierta curiosidad por saber qué son, cómo y dónde se aplican, etc. El hecho de su trascendencia social es una oportunidad para su introducción como materia en el contexto de las asignaturas de Matemáticas. Esto tiene especial interés desde un punto de vista docente en una doble perspectiva. Por un lado, encuadra los problemas relativos al ML dentro de la matemáticas, dando trascendencia y significado al quehacer matemático. Por otro, puede ser una oportunidad para abordar problemas matemáticos clásicos desde un puto de vista nuevo que aporte conocimiento y comprensión del problema.

Dentro de las técnicas de ML que se pueden introducir en los cursos básicos de Matemáticas, destaca con diferencia la del aprendizaje por refuerzo (RL). Este paradigma de aprendizaje permite abordar los problemas mediante la "invención" de un juego en el que el jugador realiza acciones que reciben recompensa o penalización. Este planteamiento ya ha sido abordado por algunos de los autores de esta comunicación en el ámbito de la Educación Secundaria (Calabuig et al., 2021). Sin embargo, y por ello el motivo de esta comunicación, el ámbito de la docencia en Matemáticas en los primeros cursos de los grados de ciencias y técnicos impartidos por las diferentes universidades plantea, a priori, un entorno mucho más rico en conocimiento y conceptos por parte del alumnado. Este concepto, el de la resolución de problemas matemáticos a través de un juego, está presente en la literatura científica (véase por ejemplo (Mnih et al., 2013) donde se entrena un modelo que aprende a jugar a juegos de la consola Atari 2600 o el reciente (Fawzi, 2022), donde se plantea la multiplicación de matrices como un juego en el que un algoritmo de RL ha conseguido encontrar un forma de multiplicar ciertas matrices de forma más eficiente que los conocidos previamente) y también educativa (véase (García et al., 1999)).

En nuestro caso, además del RL, aprovecharemos la ocasión para introducir la redes neuronales. Esta es otra herramienta de ML que resulta muy atractiva para el estudiantado dada su presencia cada vez más importante en nuestra sociedad. Y, desde un punto de vista matemático, puede dar pie a, desde un entorno amigable como es el diseño de un juego, entrar en cuestiones que son realmente de calado como puedan ser qué es una función, qué forma tiene una función y cómo las redes neuronales aproximan funciones.

¿Y qué problema hemos seleccionado para abordar esto? Tiene que ser un problema lo suficientemente complejo como para justificar en cierta manera el uso de estas herramientas pero, por otro lado, suficientemente simple, claro y habitual en la clase de Matemáticas como para que sea fácilmente reconocible para el alumnado. Ello evitará que este pueda desviar su atención en el objetivo primordial, entender cómo se resuelve el problema desde el punto de vista del ML. Además, si se trata de un problema conocido, podrá comprobar fácilmente la calidad numérica de la solución, el nivel de aproximación, etc. El problema seleccionado es el de minimización de una función de dos variables. Este tipo de problemas se trabajan en el aula de Matemáticas, admiten una representación gráfica y, por lo tanto, una visualización sencilla y, al mismo tiempo, su solución es fácilmente comprobable, si la expresión de la función no es demasiado compleja, de forma analítica.

2 Objetivos

La sesión a modo de taller que vamos a exponer tiene por objetivo principal

- introducir técnicas de Aprendizaje por Refuerzo en los primeros cursos de ingenierías sin necesidad de un alto conocimiento en técnicas de inteligencia artificial.

Destacamos otros objetivos específicos del trabajo como:

- Guiar al alumnado a la hora de plantear un problema real como un juego.
- Enfrentarlo a un problema concreto a modo de trabajo, en el que tenga cierta libertad a la hora de realizar modificaciones en el modelo.
- Motivarlo desde los primeros cursos en un campo en crecimiento como es la Inteligencia Artificial.
- Comparar el método propuesto para optimizar una función con otros métodos (analíticos y numéricos) conocidos por el alumnado por su formación matemática anterior y que le son familiares.
- Trabajar la competencia relativa a la comunicación efectiva, como a la hora de exponer los resultados a modo de presentación.

Con los objetivos anteriormente descritos se pretende enfatizar el carácter matemático de las técnicas de aprendizaje automático, un carácter que en muchas ocasiones no está explícitamente revelado a través del uso de aplicaciones. Además esperamos que a través de un problema clásico de matemáticas nuesto alumnado sea capaz de abordar una cuestión que consideramos fundamental, que es el hecho de que las técnicas de *machine learning* tienen un estatus matemático idéntico al de cualquier otra técnica matemática para la resolución de un problema, y por tanto deben ser utilizadas con el mismo sentido y rigor.

Creemos que las técnicas relacionadas con el RL se suelen evitar por su elevada complejidad. Por ello, lo que presentamos es una simplificación de un modelo de RL concreto, el *Deep Q-learning*. Será necesario el uso de una red neuronal (aunque se podría cambiar por otra técnica de aproximación de funciones), lo que puede ser de utilidad para que los estudiantes vean una aplicación de esta técnica más allá de los habituales ejemplos de predicción y clasificación. Por lo tanto, se recomienda un conocimiento previo de redes neuronales, aunque se podría explicar junto con este taller si se pudiera trabajar en varias sesiones.

3 Desarrollo de la innovación

En esta sección vamos a introducir brevemente qué es el aprendizaje por refuerzo y las redes neuronales, para luego aplicarlo al problema de minimización de una función. Se explicará de una forma sencilla, haciendo hincapié únicamente en las técnicas que sean necesarias utilizar en esta práctica, para que el estudiantado de primeros cursos de ciencias o ingenierías sea capaz de entenderlo. Posteriormente, expondremos el algoritmo, que le servirá como una introducción no solo al RL y la inteligencia artificial, sino como ejemplo de modelización de un problema real en términos de un juego.

3.1 ¿Qué es el aprendizaje por refuerzo?

El aprendizaje por refuerzo (*Reinforcement Learning* en inglés, RL) es una técnica de inteligencia artificial, basado en el aprendizaje que realizan los organismos naturales interactuando con el ambiente. La base de este es la existencia de un entorno, con unas leyes que a priori se desconocen, en el que un agente interactúa sucesivamente con el objetivo de maximizar las recompensas que obtiene. El agente, por lo tanto, aprende continuamente las leyes que rigen el ambiente a partir sus propias acciones y las consecuencias que estas tienen. Su objetivo final es realizar las mejores acciones posibles que le permitan obtener mayores recompensas. En la Figura 1 se representa el típico diagrama que describe el RL.

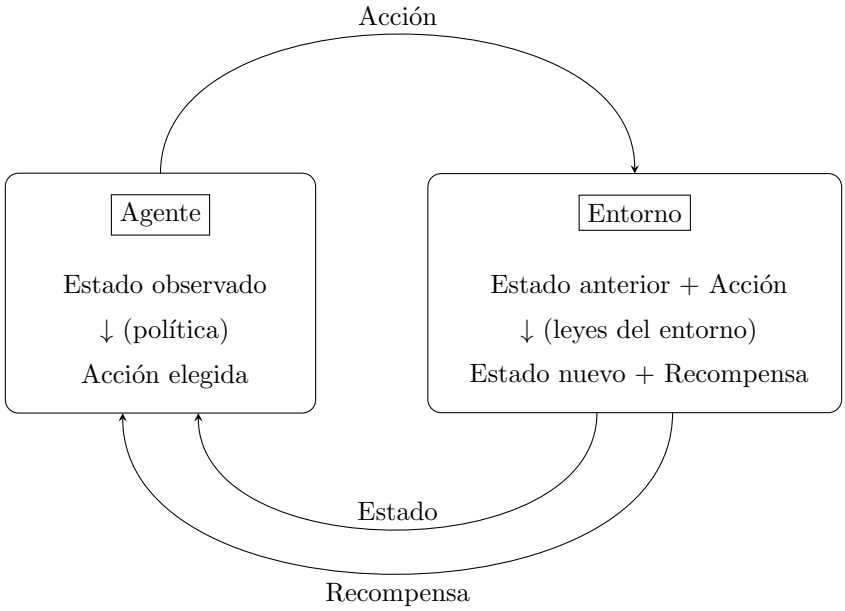

Fig. 1: Diagrama que representa la interacción entre el agente y el entorno en el RL. En la parte izquierda, el agente elige cómo actuar a partir del estado actual. En la derecha, como afecta dicha acción al entorno, que volverá a ser observado por el agente. Este proceso (un paso) se repite continuamente.

Vamos a explicar el diagrama con más detalle. Cada vez que el agente tiene la posibilidad de realizar alguna acción, este observa el entorno, del que solo reconoce el estado actual, que llamaremos e. Denotaremos por $\mathcal{E}$ el conjunto de los distintos estados que puede haber en el entorno. A partir de esta información, tiene que elegir qué acción a realizar dentro de un conjunto de acciones posibles, $\mathcal{A}$. Para ello, se basa en su memoria (es decir, que acciones ha realizado anteriormente y las recompensas que ha obtenido). El criterio o función que utiliza para elegir se llama *política* y se suele denotar por π. En general la política puede ser estocástica, es decir, fijado un estado, diferentes acciones se pueden elegir, cada una con una probabilidad, $P(\text{acción} = a \mid \text{estado} = e)$. Aunque en nuestro caso consideraremos una política determinista, en la que dado un estado, el agente elige una acción sin intervención del azar (de momento). Por lo tanto, denotamos la acción elegida por la política como

$$a = \pi(e).$$

Una vez escogida la acción, a, esta se realiza sobre el entorno. Esto provoca un cambio de estado, que pasa de ser e a un nuevo estado, e^+ y una recompensa r, que evalúa lo buena o mala que ha sido la elección de esa acción tras ver cómo ha modificado el entorno. En nuestro modelo, supondremos que el azar tampoco interviene en el entorno, es decir, que dado un estado y una acción, estos determinan totalmente el siguiente estado y la recompensa. Utilizando la notación habitual de grupos, lo denotaremos como

$$e^+ = a(e), \quad r = R(e, a, e^+).$$

El proceso explicado en los dos párrafos anteriores, lo denominaremos un *paso*, y lo repetiremos sucesivamente. Por esta razón, solemos escribir las variables anteriores con un subíndice, es decir, e_n, a_n, e_n^+ y r_n para el paso n. De este modo, el agente va obteniendo una experiencia de cómo es el entorno, que servirá para mejorar la política π. La función π, que decide cómo actuar en cada caso, puede estar basada en una red neuronal (ver por ejemplo (Mnih et al., 2013)), un proceso de regresión (como es el caso de extensiones de funciones de Lipschitz, ver (Calabuig et al., 2020)) u otra forma de aproximar funciones, que pueda optimizarse mientras la memoria del agente se amplia.

En cuanto a la recompensa, no solo estaremos interesados en que la acción a elegir maximice la siguiente recompensa, puesto que esto podría provocar que el modelo se atasque en máximos locales, sino que también buscaremos maximizar las recompensas a largo plazo. Con este objetivo, definiremos el valor V_π de un estado como la suma de las siguientes recompensas esperadas, aplicando un factor de descuento a las recompensas futuras, γ:

$$V_\pi(e) = \sum_{n=0}^{\infty} \gamma^n r_n.$$

Donde r_0 representa la recompensa que se obtendrá, si el agente actúa sobre el estado e y r_n representa la recompensa que se obtendrá después de n pasos. Observemos que V_π depende de las acciones elegidas y, por lo tanto, de π. Como esta función es también desconocida, se irá aproximando tras cada paso, para lo que utilizaremos una simplificación de la *ecuación de Bellman*, resultando en

$$V_\pi(e) = r_0 + \gamma \sum_{n=0}^{\infty} \gamma^n r_{n+1} = r_0 + \gamma V(e^+). \tag{1}$$

Notar que, en algunos casos, las leyes del entorno que rigen cómo afecta cada acción a un determinado estado pueden se conocidas previamente por el modelo, parcial o totalmente. En ese caso, el modelo se denomina *model-based*. En caso contrario, modelo tipo *model-free*, las leyes del entorno se asumen desconocidas, y se irán aprendiendo a partir de la experiencia de acciones pasadas. Este último es el modelo que utilizaremos en nuestra propuesta.

Por lo tanto, en el RL, no es necesario tener un conjunto de datos sobre el que "aprender", sino que el modelo aprenderá interactuando con el entorno mediante una técnica de ensayo y error. Esto hace que sea especialmente útil para que robots o máquinas aprendan a realizar ciertas tareas, ordenadores consigan encontrar estrategias ganadoras en juegos u otros muchos ejemplos en el que un modelo tiene que realizar una tarea formada por una sucesión de eventos comportándose y aprendiendo como lo haría un humano. Para una explicación más detallada del RL, de los distintos

tipos de modelos o políticas más utilizadas, recomendamos consultar (Brunton & Kutz, 2022, Capítulo 11).

3.2 Redes neuronales

Las *redes neuronales artificiales* son una técnica de inteligencia artificial que intenta, emulando el funcionamiento de las redes neuronales naturales, enseñar a las máquinas a aprender de una forma automática. Básicamente una red neuronal (artificial) consiste de unidades de proceso muy simples (neuronas) distribuidas paralelamente (en capas) y conectadas entre sí a través de uniones que están moduladas por pesos.

En la Figura 2 se puede ver una representación de rede neuronal como un grafo dirigido y ponderado donde los nodos son las neuronas; las aristas, las conexiones entre ellas; y los pesos representa cómo afecta una neurona a otra (los coeficientes de la combinación lineal). La primera capa está formada por las entradas de la red. El valor de cada neurona (salvo las de entrada) se calcula como una suma de las neuronas de la capa anterior multiplicada por los correspondientes pesos su salida se calcula aplicando una cierta función (función de activación) a dicho valor. Finalmente, la última capa se considera la salida de la red.

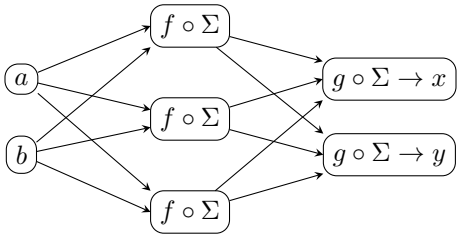

Fig. 2: Ejemplo de red neuronal con dos entradas: a y b, una capa oculta formada por 3 neuronas y dos salidas: x e y. Cada arista representa un peso y las funciones de activación son f para la capa oculta y g para la capa de salida. Cada Σ representa una suma de las entradas de la neurona.

Las redes neuronales permiten:

- Predecir: dada una serie temporal de datos, estimar el valor del dato en un tiempo posterior.
- Aproximar funciones de varias variables.
- Clasificar patrones: asignar una entrada a una clase de entre un conjunto previamente establecido.
- Agrupar los datos en una serie de conjuntos por similitud entre patrones.

Desde un punto de vista de su funcionamiento, las redes neuronales se caracterizan por:

- Arquitectura: es la estructura o patrón de las conexiones entre las neuronas.
- Dinámica de la computación: indica el valor que toman las neuronas que componen la red en base a las funciones de activación.
- Algoritmo de entrenamiento o aprendizaje: es el procedimiento para determinar los pesos entre las diferentes conexiones a partir de los datos de entrenamiento,

Las funciones de activación que utilizaremos en la implementación del modelo son dos de las más sencillas (aunque se pueden modificar por otras):

- *Identidad*: $f(x) = x$;
- *ReLU*: $f(x) = max(x, 0)$, que no modifica los valores positivos, pero asigna 0 a los valores negativos.

Con la funciones de activación distintas a la identidad conseguiremos que la red final sea una función no lineal, de modo que se podrá aproximar mejor a nuestro objetivo.

Un breve conocimiento sobre redes neuronales es conveniente para que el alumnado se centre en comprender la parte importante del RL, que es cómo el agente interactúa con el estado. Si este no es el caso, se puede aprovechar la práctica para introducir las redes neuronales sin entrar en mucho detalle, ya que, a la hora de programar el algoritmo se utilizará el paquete *keras* (Chollet et al., 2015) y este se encargará de realizar los cálculos y entrenamiento de la red.

3.3 El problema de minimizar de función

Dada una función real de varias variables y un dominio, el problema de minimizar (respectivamente maximizar) la función consiste en encontrar el punto del dominio en el que la función alcanza su valor mínimo (resp. máximo). Este problema se estudia desde el bachillerato y sigue presente en la universidad. Cuando la función a estudiar es diferenciable en un conjunto abierto y acotado, la estrategia más utilizada consiste en buscar sus puntos críticos, resolviendo el sistema de ecuaciones que resulta de igualar las derivadas parciales de la función a 0. Posteriormente se clasifican estos en mínimos locales, máximos locales y puntos de silla para finalmente encontrar el mínimo o máximo global. Cuando el dominio es compacto, se utilizan también otras técnicas, como los multiplicadores de Lagrange o las condiciones de Karush-Kuhn-Tucker.

Dado que las funciones a optimizar no siempre son sencillas de derivar, o el sistema de ecuaciones resultante puede no ser posible de resolver a mano, existen métodos heurísticos que permiten encontrar al menos una aproximación de los máximos o mínimos de una función. En otros casos más generales, la función a optimizar, puede no cumplir las condiciones de diferenciabilidad necesarias o no ser una función conocida, como es el caso de algunas funciones provenientes de problemas de la física. Un ejemplo de método numérico es el método del gradiente (descenso del gradiente), que ya se estudia en algunos grados universitarios.

En este trabajo, vamos a estudiar cómo resolver el problema de minimizar (el caso de maximizar sería equivalente) una función utilizando el RL desde un punto de vista académico. Para ello, plantearemos el problema como un juego, en el que un agente deberá moverse por el entorno (el dominio de la función) con el objetivo de resolver el problema. Esto presenta una estrategia sencilla desde el punto de vista matemático para abordar el problema.

3.4 Algoritmo de aprendizaje por refuerzo para optimizar una función

Vamos a considerar, como ejemplo, una función sencilla, definida en $X = [-10, 10] \times [-10, 10] \subset \mathbb{R}^2$ como

$$f(x, y) = \text{mín}\left(2(x-2)^2 + (y+2)^2, 20 + 3(x-7)^2 + (y-7)^2\right),$$

con el objetivo de minimizarla. Esta función tiene un mínimo local en $(2, -2)$ y el mínimo global se alcanza en $(7, 7)$.

Para definir el modelo de RL, empezaremos por exponer el funcionamiento del entorno. El conjunto de estados posibles, $\mathcal{E}$, estará formado por una malla cuadrada de 21 por 21 puntos en X a distancia 1 entre ellos.

El conjunto de posibles acciones estará formado por 5 elementos: $\mathcal{A} = \{0, \rightarrow, \uparrow, \leftarrow, \downarrow\}$, donde cada uno representa un movimiento en la malla. Por lo tanto, dado un estado y una acción, el siguiente estado vendrá dado (determinísticamente) por

Estado actual	Acción	Estado siguiente
(x, y)	0	(x, y)
(x, y)	$\rightarrow$	$(x+1, y)$
(x, y)	$\uparrow$	$(x, y+1)$
(x, y)	$\leftarrow$	$(x-1, y)$
(x, y)	$\downarrow$	$(x, j-y)$

Aunque si, tras realizar una acción, el nuevo punto no pertenece a $\mathcal{E}$, se tomará como nuevo estado el elemento de $\mathcal{E}$ más cercano. Por ejemplo, si en el punto $(1, 10)$, se realiza la acción $\rightarrow$, el nuevo punto debería de ser $(1, 11)$, pero como no pertenece a $\mathcal{E}$, el siguiente estado volvería a ser $(1, 10)$.

Por lo tanto, el agente deberá moverse por el entorno, utilizando las acciones permitidas, con el objetivo de maximizar las recompensas futuras. Como nuestro problema consiste en encontrar el mínimo de la función, una opción es que la recompensa y el valor de la función tengan signos opuestos, para que minimizar f sea equivalente a maximizar la recompensa. Así, definiremos la recompensa como

$$R(e, a, e^+) = -f(e^+),$$

donde e denota el estado (anterior), a la acción realizada y e^+ el estado siguiente. Otras formas de definir la recompensa podrían ser estudiadas y probadas por el alumnado. Por ejemplo, la recompensa podría ser definida como la disminución en la función tras aplicarle la acción, es decir $R(e, a, e^+) = -(f(e^+) - f(e))$ u otras posibilidades.

El agente empezará de un punto inicial, por ejemplo el $(0, 0)$, y durante una cantidad determinada de pasos (en este caso hemos elegido 50), se moverá por la malla, ver varios ejemplos en la Figura 5. A este camino, formado por una sucesión de 50 estados, acciones y recompensas, lo denominaremos *episodio*. Una vez realizado el camino, se volverá a situar en el punto inicial y se repetirá el proceso, durante una cantidad determinada de episodios (por ejemplo, 20). El objetivo será que el último estado de cada episodio sea el punto del dominio con menor valor posible de la función, es decir, que el agente se dirija en dirección al mínimo de la función.

Para definir la política del agente, es decir, el algoritmo que decidirá qué acción tomar en cada momento, utilizaremos una simplificación del modelo expuesto en (Mnih et al., 2013), donde se utiliza un modelo tipo *deep Q-learning*. El *Q-learning* está basado en la función Q que aproxima a partir de (1) el valor esperado para un par estado-acción, es decir,

$$Q(e,a) = V_\pi(e), \quad \text{si la acción elegida por } \pi \text{ es } a.$$

Siguiendo la ecuación (1), calcularemos Q a partir de

$$Q(e,a) = R(e,a,e^+) + \gamma V(e^+), \quad \text{donde } e^+ = e(a).$$

El prefijo *deep* hace referencia a que se utilizará una red neuronal con capas ocultas para la aproximación de la función Q. La forma de elegir la acción a tomar, se realizará evaluando la función Q en el estado actual y todas las posibles acciones para escoger la que tenga un mayor valor, es decir, la acción que se espera que genere unas mayores recompensas. Por lo tanto,

$$\pi(e) = \arg\max_{a\in\mathcal{A}} Q(e,a) \tag{2}$$

y la función valor en el siguiente estado deberá cumplir que

$$V_\pi(e^+) = Q(e^+, \pi(e^+)) = \max_{a\in\mathcal{A}} Q(e^+, a).$$

Así pues, Q debe cumplir que para cada par (e,a),

$$Q(e,a) = R(e,a,e^+) + \gamma \max_{\hat{a}\in\mathcal{A}} Q(e^+, \hat{a}). \tag{3}$$

La función Q, que será desconocida al iniciar el algoritmo, irá aprendiendo en cada interacción del agente con el entorno, solo a partir de los estados, acciones y recompensas que el agente obtiene de este. En nuestro caso, en lugar de considerarla como una función con entradas e y a y una salida, Q, vamos a considerarla como una función $e \mapsto \big(Q(e,a)\big)_{a\in\mathcal{A}}$. Es decir, dado un estado e, obtendremos como resultado un vector de 5 elementos con el valor de Q para cada una de las acciones a posibles. Esto hará que, una vez fijado el estado, sea más sencillo evaluar las recompensas futuras esperadas para cada acción y escoger la mejor.

Consideraremos, para aproximar Q, una red neuronal con 2 entradas (representando un elemento de $\mathcal{E}$) y 5 salidas (una por cada elemento de $\mathcal{A}$). A parte de las capas de entrada y salida, la red tendrá algunas capas ocultas, que posteriormente analizaremos. Por ello, Q dependerá implícitamente de los pesos de la red θ, que irán cambiando con el tiempo y, como consecuencia, también lo hará la política π.

En cada paso, a la hora de elegir qué acción tomar, se evaluará el estado actual, e, en la red neuronal y la acción elegida a será la que, la correspondiente salida de la red, tenga un valor mayor (Ecuación 2). Una vez realizada la acción, se obtendrá del entorno una recompensa. Una vez conocida esta, puede que $R(e,a,e^+)+\gamma \max_{\hat{a}\in\mathcal{A}} Q(e^+,\hat{a})$ no coincida con el valor que habíamos obtenido por la red para $Q(e,a)$ (ver Ecuación 3), por lo que asumiremos que la red está cometiendo un error de

$$R(e, a, e^+) + \gamma \max_{\hat{a} \in \mathcal{A}} Q(e^+, \hat{a}) - Q(e, a) \tag{4}$$

en la salida asociada a a al obtener como entrada e. Conociendo este error, se reentrenará la red, a partir de los pesos anteriores. El reentrenamiento se realizará con el nuevo dato y también todos los anteriores, para que la red no olvide sus acciones pasadas.

Obsérvese que, si siempre elegimos la mejor acción posible, es posible que el modelo crea, a priori, que unas determinas acciones son malas. Por lo tanto, no las exploraría nunca, aún en el caso de que estas sean las mejores acciones realmente. Es por ello que en los algoritmos de RL se suele añadir un componente aleatorio, por ejemplo, que en el $\varepsilon = 20\,\%$ de las ocasiones la acción elegida sea un sorteo de todas las acciones posibles y no la (a priori) mejor.

Crear: Entorno
Crear: Agente
Agente: Inicia la red aleatoriamente
$Memoria \leftarrow \emptyset$
para *episode desde* 1 *a 100* **hacer**
  Reiniciar Entorno ($e \leftarrow (0, 0)$)
  para *paso desde* 1 *a 50* **hacer**
    si *Número aleatorio* $<$ *0.2* **entonces**
      $a \leftarrow$ *aleatorio*
    en otro caso
      $Q \leftarrow$ Red del Agente(e)
      $a \leftarrow \arg\max Q$
    fin
    Entorno: Realizar acción a
    $e^+ \leftarrow$ Entorno: obtener estado
    $R \leftarrow$ Entorno: obtener Recompensa
    Añadir a $Memoria \leftarrow (e, a, e^+, R)$
    Agente: Entrenar la Red a partir de $Memoria$ utilizando como error la Ecuación 4
    $e \leftarrow e^+$
  fin
fin

Algoritmo 1: Pseudocódigo simplificado del modelo.

Reiniciar Entorno ($e \leftarrow (0, 0)$)
para *paso desde* 1 *a 50* **hacer**
  $Q \leftarrow$ Red del Agente(e)
  $a \leftarrow \arg\max Q$
  Entorno: Realizar acción a
  $e^+ \leftarrow$ Entorno: obtener estado
  $e \leftarrow e^+$
fin

Algoritmo 2: Pseudocódigo simplificado del episodio en que el modelo ya se considera entrenado.

El esquema general del modelo puede verse escrito como pseudocódigo en el algoritmo 1. Con este algoritmo, el modelo irá aprendiendo a base de ensayo y error. Una vez entrenado para encontrar la mejor solución posible, podemos ejecutar un único episodio, escogiendo siempre la mejor opción,

siguiendo el algoritmo 2. El final del camino obtenido (último estado) será el valor que el modelo aproximará como mínimo de la función.

3.5 Resultados del modelo

Para el ejemplo, se ha tomado los hiperparámetros $\gamma = 0{,}9$, $\varepsilon = 20\,\%$ de probabilidad de escoger una acción aleatoria y, en cuanto a la red, una estructura con 3 capas ocultas de 32, 48 y 32 neuronas respectivamente. *ReLU* como función de activación (salvo en la última capa, que no hemos aplicado función de activación) y una tasa de aprendizaje de 0.002, ver Figura 3. Aunque la arquitectura y aprendizaje de esta red puede ser modificada por el alumnado, observando cómo esta elección puede condicionar la convergencia del algoritmo.

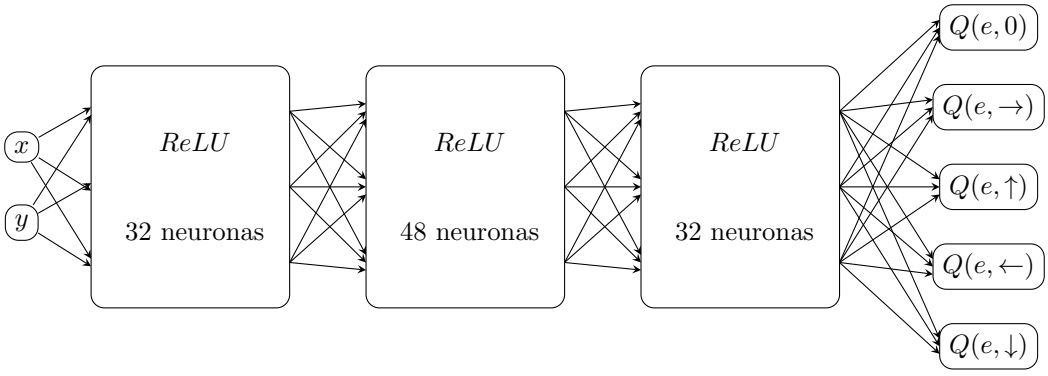

Fig. 3: Diagrama de la arquitectura de la neuronal utilizada. Para un estado de entrada, $e = (x, y)$, las salidas son los valores esperados para $Q(e, a)$, para cualquiera de las acciones posibles $a \in \mathcal{A}$.

En la Figura 4 se muestran un ejemplo de los resultados obtenidos por el modelo. En este caso, cada episodio (camino) está formado por 50 acciones (movimientos en la malla) y se ha entrenado durante 20 episodios. La evolución de las recompensas, que se puede ver en la Figura 5, tiene una tendencia creciente, que se estabiliza en los últimos episodios, cuando el modelo ya está entrenado y reconoce rápidamente el mínimo de la función. Una vez entrenado, se repite un sólo episodio tomando siempre la mejor acción para ver el resultado del mejor camino posible (Figura 5) que está formado por las acciones:

$$\rightarrow, \rightarrow, \rightarrow, \uparrow, \rightarrow, \rightarrow, \uparrow, \rightarrow, \uparrow, \uparrow, \uparrow, \rightarrow, \uparrow, \uparrow, 0, 0, 0, 0, 0, 0, \ldots, 0.$$

Tras los 14 primeros pasos, el camino ya llega al mínimo, el $(7, 7)$ y a partir de ahí, la acción elegida es 0, es decir, quedarse en ese punto. Además, el agente se dirige hacia el mínimo directamente, es decir sin volver hacia la izquierda o abajo en ningún momento, por lo que el camino elegido es también óptimo en el sentido de llegar lo antes posible a dicho punto.

Consideramos que los dibujos expuestos (calculados automáticamente con el fichero Python) son de vital importancia para comprender el funcionamiento del algoritmo.

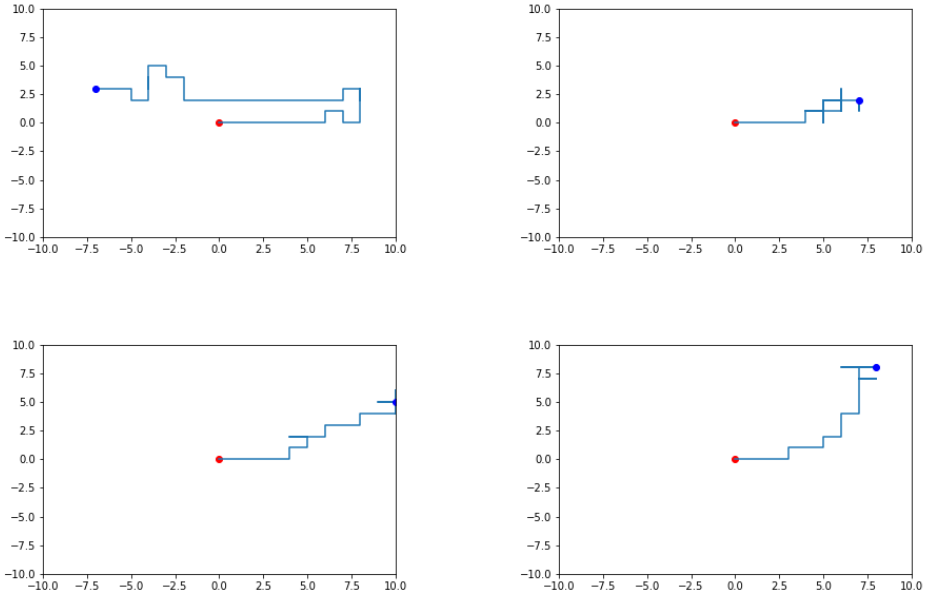

Fig. 4: Episodios 1, 5 (arriba), 10 y 20 (abajo) del entrenamiento del modelo. En rojo, el punto inicial, $(0, 0)$*, y en azul, el punto final de cada camino.*

3.6 Propuestas para el alumnado

Aunque las mejores propuestas o ejercicios los puede proponer el profesorado según el nivel y el tiempo del que disponga, dejamos aquí sólo algunas de las múltiples modificaciones que se pueden implementar:

- Optimizar distintas funciones y variar el modelo para encontrar el máximo en lugar del mínimo.
- Cambiar los hiperparámetros y arquitectura de la red.
- Ampliar el dominio y permitir que las aproximaciones sean más precisas (por ejemplo, que cada paso represente un movimiento menor que 1). Si se amplia dominio y el espacio de estados, será necesario ampliar el número de pasos por episodio y tal vez también el número de episodios para que el algoritmo converja.
- Si se amplían estos hiperparámetros, el entrenamiento empieza a ser muy lento. Algunas opciones de mejora son cambios en cómo la red neuronal se entrena, como por ejemplo que la red no se entrene todos los pasos o que utilice solo una parte de la memoria para entrenar.
- El valor ε se puede tomar no constante. Por ejemplo, que el modelo explore más al principio (ε casi 1) y, tal y como avanza el aprendizaje, considerar ε menores.
- Realizar cambios en la función recompensa, la política o el hiperparámetro γ. Por ejemplo, la acción elegida puese ser aleatoria con probabilidad dependiente de la recompensa esperada.

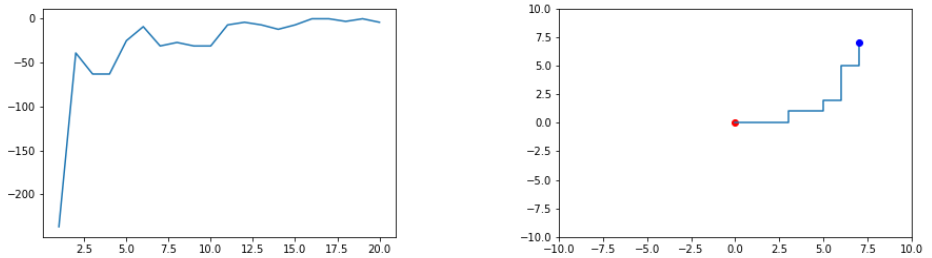

Fig. 5: A la izquierda, la evolución de las recompensas durante los 20 episodios de entrenamiento y a la derecha, el resultado final del algoritmo.

Estudiar las consecuencias (por ejemplo, un γ pequeño puede provocar que el algoritmo se atasque en mínimos locales).

- Comparar los resultados (en términos de precisión, tiempo de computación, etc.) con otros métodos numéricos para resolver el mismo problema que el alumnado haya aprendido en otras asignaturas.

4 Resultados

Vamos a exponer ahora la forma en la que creemos que se debería implementar esta propuesta en el aula. Será muy importante determinar a quién va dirigido, el contexto, los materiales, etc. En nuestro caso pretendemos la introducción de la nueva cultura matemática relativa al aprendizaje automático y la ciencia de datos en los primeros cursos de las carreras científico-técnicas para poner las matemáticas en contexto en el momento tecnológico y social que viven nuestro alumnos, lo cual es fundamental para atraer su atención y su interés por al disciplina.

Al elegir como motivo la optimización de funciones de varias variables, estamos hablando de un contenido que existe en casi todos los cursos de cálculo las carreras técnicas (la asignatura de Cálculo II del Grado de Ingeniería Física o la asignatura de Matemáticas II del Grado de Ingeniería Civil por poner tan solo un par de ejemplos dentro de la UPV). Además, se da la circunstancia (y las dos asignaturas mencionadas son también un ejemplo) que se están incorporando actividades de trabajo colectivo (póster, pequeñas exposiciones) en prácticamente todas la asignaturas con el objetivo de tener elementos de evaluación de algunas de las competencias transversales. Para acabar de cerrar el círculo, la mayoría de las escuelas están adoptando como lenguaje de programación para todas sus asignaturas el Python y su *Notebook* Jupiter.

Por lo tanto la actividad que nos proponemos es resolver un problema de minimización de una función de varias variables, sencilla, como alguna de las que hayan visto en clase de problemas (prácticas de aula) o analizado en las prácticas regulares de cálculo (ya que en general todas estas asignaturas tienen asociadas prácticas en el laboratorio de informática; las asignaturas mencionadas arriba las tienen). Se trata de realizar un pequeño trabajo o póster donde se explique en que consiste el Aprendizaje Automático y las Redes Neuronales, ejemplos de aplicación en la vida cotidiana que encuentren en la prensa y finalmente la aplicación al problema planteado. No es difícil dar a

cada alumno una función diferente. Dependiendo de cada titulación y el nivel de conocimientos de programación se les proporcionará, bien el código en Python, bien una guía de las funciones necesarias y un esquema de como programarlo. Con él podrán experimentar la búsqueda del mínimo de una función. Se les propondrá como actividad complementaria que, además de la función dada por el profesor, exploren otras funciones de su elección y traten de describir el comportamiento del algoritmo (velocidad de convergencia, precisión alcanzada...). Para cursos más avanzados, como es el caso de máster, se les puede dar la posibilidad de que modifiquen algunas partes del modelo y lo implementen en el código, como las propuestas en la Subsección 3.6 u otras que piense el alumnado. De este modo, pueden estudiar las consecuencias que tiene en la convergencia del modelo y tratar de razonar porqué ocurre. Finalmente, bien por escrito, bien en presentación oral, explicarían todo el proceso realizado y los resultados alcanzados.

Esta práctica a modo de taller aún no ha sido implementada en el aula. Aún así, uno de los autores sí que ha experimentado como alumno una sesión práctica con una estructura y forma de evaluación similar en las asignaturas Espacios de Funciones y Aproximación (33208) y Redes Neuronales y Algoritmos Genéticos (33219) del Máster Universitario en Investigación Matemática (Universitat Politècnica de València y Universitat de València). El autor valora positivamente la estructura de la práctica, que considera adecuada y de gran utilidad, puesto que le permite acercase a técnicas complicadas y, sobre todo, poder experimentar con el código y ajustar los hiperparámetros. Como el alumnado tiene acceso al código del modelo y no depende de gran cantidad de librerías (al menos en el modelo principal), puede ejecutar el código a trozos para ver cómo funciona en cada momento, poder realizar cambios y, finalmente, adaptar el modelo a otro problema.

5 Conclusiones

En este proyecto se presenta una práctica a modo de taller que pretende ser una primera toma de contacto de los estudiantes con el aprendizaje por refuerzo (RL), una técnica de la inteligencia artificial que comúnmente no se estudia por su dificultad. El modelo elegido es una simplificación del *Deep Q-learning,* y el problema a abordar es el de encontrar el mínimo de una función, que ya es conocido por el alumnado, para conseguir que toda dificultad resida en el aprendizaje de la técnica, y no en la complejidad del problema.

Aunque la parte más importante es que los estudiantes comprendan cómo abordar un problema utilizando el RL, se les pedirá que programen el algoritmo en Python (utilizando partes del código escritas por el profesor si se requiere). Como el modelo estará basado en una red neuronal, se introducirán brevemente en el caso que el alumnado no conozca su uso. Se evitará el uso de paquetes (salvo en el caso de la red neuronal en la que recomendamos le uso de *keras* (Chollet et al., 2015) por su sencillez), con el objetivo de que el alumnado comprenda cómo funciona el algoritmo y no lo perciba como una "caja negra". Además, esto le permitirá realizar modificaciones en el código o adaptarlo para abordar otros problemas. Finalmente, se propone que los estudiantes presenten el trabajo a modo de exposición, donde comparen los resultados con otros métodos conocidos para resolver el mismo problema, y explicar las modificaciones realizadas.

Aunque este taller aún no ha sido implementado en el aula tal y como se describe, se ha pensado para que los contenidos se adapten al conocimiento e interés del alumnado de grados en ingeniería. En un futuro, se pretende implementar como piloto en la práctica 3 de la asignatura Cálculo II para el curso 2024/2025.

Disponibilidad el código

El código descrito está disponible para su consulta y reproducción. Para obtener acceso póngase en contacto a través de correo electrónico a la dirección ararnnot@posgrado.upv.es o con cualquiera de los autores.

Referencias bibliográficas

Brunton, S. L., & Kutz, J. N. (2022). *Data-driven science and engineering: Machine learning, dynamical systems, and control.* Cambridge University Press.

Calabuig, J., Falciani, H., & Sánchez-Pérez, E. (2020). Dreaming machine learning: Lipschitz extensions for reinforcement learning on financial markets. *Neurocomputing*, *398*, 172-184.

Calabuig, J., García, L., & Sánchez, E. (2021). Aprender como una máquina: Introduciendo la inteligencia artificial en la ense nanza secundaria. *Modelling in Science Education and Learning*, (14(1)), 5-14.

Chollet, F., et al. (2015). Keras.

Fawzi, A. e. a. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, (610(7930)), 47-53.

García, L., Sánchez, E., & Giménez, F. (1999). Dos ejemplos de introducción de los métodos de simulación Monte Carlo en los primeros cursos de las carreras técnicas. *Lecturas matemáticas*, *20(1)*, 39-60.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning.