ugr | Universidad de Granada

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO OFICIAL EN CIENCIA DE DATOS E INGENIERÍA DE COMPUTADORES

Deep Reinforcement Learning para control energético eficiente de edificios

Autor
Antonio Manjavacas Lucas

Directores
Juan Gómez Romero
Miguel Molina Solana

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 27 de agosto de 2021

energym

Deep Reinforcement Learning para control energético eficiente de edificios

Autor
Antonio Manjavacas Lucas

Directores
Juan Gómez Romero
Miguel Molina Solana

Deep Reinforcement Learning para control energético eficiente de edificios

Antonio Manjavacas Lucas

Palabras clave: Aprendizaje profundo por refuerzo; Control energético de edificios; Agentes inteligentes

Resumen

En las últimas décadas, tanto el calentamiento global como el cambio climático se han visto significativamente afectados por el incremento de la demanda energética de edificios residenciales y comerciales. Estos son responsables de un tercio del consumo mundial de energía y de hasta un 40 % de las emisiones de CO_2, mayormente producidas por los sistemas de calefacción, ventilación y aire acondicionado (HVAC) destinados a garantizar el bienestar de sus ocupantes.

Ante esta problemática, optimizar el control de los sistemas HVAC se plantea como una solución necesaria ante el creciente interés por garantizar la eficiencia energética de los edificios. Dicho control ha sido tradicionalmente llevado a cabo mediante técnicas reactivas, las cuales no siempre garantizan la maximización del confort de los ocupantes y, al mismo tiempo, la minimización del consumo energético a largo plazo.

En contraposición a estos métodos tradicionales, en los últimos años ha aumentado el interés por las técnicas basadas en aprendizaje profundo por refuerzo (DRL) orientadas a control HVAC, a raíz de los resultados exitosos en otras áreas, como la robótica o los juegos. No obstante, se trata de un campo relativamente inmaduro, donde se carece de marcos de referencia y bancos de prueba específicamente destinados a reproducir y comparar los diferentes algoritmos que conforman el estado del arte.

En respuesta a esta necesidad, el objetivo perseguido en este trabajo será el desarrollo de un entorno de ejecución de simulaciones energéticas orientado al uso y evaluación de diferentes algoritmos de DRL en control HVAC. A su vez, se profundizará en la experimentación con estos algoritmos haciendo uso del entorno implementado, evaluando los resultados obtenidos en términos de consumo energético y confort.

Deep Reinforcement Learning for efficient building energy control

Antonio Manjavacas Lucas

Keywords: Deep Reinforcement Learning; Building energy control; Intelligent agents

Abstract

In recent decades, both global warming and climate change have been significantly affected by the increased energy demand of residential and commercial buildings. These are responsible for one third of global energy consumption and up to 40 % of CO_2 emissions, mostly produced by heating, ventilation and air conditioning (HVAC) systems designed to ensure the well-being of their occupants.

Given this problem, optimizing the control of HVAC systems is a necessary solution in view of the growing interest in guaranteeing the energy efficiency of buildings. This control has traditionally been carried out by means of reactive techniques, which do not always guarantee the maximization of occupants' comfort and, at the same time, the long-term minimization of energy consumption.

In contrast to these traditional methods, in recent years there has been a growing interest in deep reinforcement learning (DRL) techniques for HVAC control, following successful results in other areas such as robotics or games. However, this is a relatively immature field, where there is a lack of reference frameworks and benchmarks specifically aimed at reproducing and comparing the different algorithms that comprise the state of the art.

As a response to this need, the objective of this work will be the development of an environment for the execution of energy simulations oriented to the use and evaluation of different DRL algorithms in HVAC control. At the same time, the experimentation with these algorithms using the implemented environment will be deepened, evaluating the results obtained in terms of energy consumption and comfort.

Yo, **Antonio Manjavacas**, alumno de la titulación Máster Universitario Oficial en Ciencia de Datos e Ingeniería de Computadores de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI **06286831B**, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Antonio Manjavacas Lucas

Granada a 27 de agosto de 2021.

D. **Juan Gómez Romero**, profesor titular del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **Miguel Molina Solana**, profesor ayudante doctor del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Deep Reinforcement Learning para control energético eficiente de edificios***, ha sido realizado bajo su supervisión por **Antonio Manjavacas Lucas**, y autorizan la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 27 de agosto de 2021.

Los directores:

Juan Gómez Romero **Miguel Molina Solana**

Agradecimientos

Quisiera dar las gracias y dedicar este trabajo a todas las personas que, desde lo personal como lo profesional, me han acompañado a lo largo de este maravilloso año, repleto de aprendizaje y buenas experiencias.

Gracias a Juan y Miguel, por su cercanía y sus buenos consejos, así como por brindarme la oportunidad de colaborar en este proyecto y profundizar en mi pasión por el aprendizaje por refuerzo, un campo que me apasiona y del que nunca dejaré de aprender.

Este trabajo tampoco habría sido posible sin la ayuda incondicional de Alejandro y Javier, compañeros de trabajo, reuniones, mensajes de Slack y satisfacciones por ver crecer esta iniciativa. En palabras de Javier: "hace unos meses parecía algo muy lejano". ¡Gracias por todo!

También quisiera agradecer a mis padres, Antonio y Criptana, todo su sacrificio y ánimo a lo largo de un año marcado por la distancia. Gracias por entender lo que me hace feliz y prestarme todo vuestro apoyo.

Gracias a Andrea, Ithiel, Patricio, Margarita y Anouk, compañeros a lo largo de un viaje que comenzó en diciembre y terminó en La Gomera. Seguro que volveremos a encontrarnos.

A mis buenos amigos *granaínos* y no tan *granaínos*: Carlos, Mape, Eva, Elena, Jose, Pablo, Juanje, Blanca... ¡sois muchos los que habéis hecho de este un año inigualable! Gracias, de corazón, por todos los buenos momentos y experiencias compartidas.

Por último, pero no menos importante, quisiera dar las gracias a Laura y Virginia. Jamás pensé que en un año marcado por la pandemia y la "distancia social" me acercaría tanto a dos personas tan maravillosas. Gracias por soportarme, ser testigos de mi trabajo y estar ahí en los buenos y malos momentos.

Antonio Manjavacas Lucas
Granada, 27 de junio de 2021

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

En este primer capítulo, se dará a conocer la motivación del proyecto, ofreciendo una perspectiva general del contexto y objetivos que lo atañen.

1.1. Motivación

En las últimas décadas, tanto el calentamiento global como el cambio climático se han visto significativamente alentados por el crecimiento en la demanda energética de edificios residenciales y comerciales. De acuerdo con organizaciones como la Agencia Internacional de Energía[1] (*International Energy Agency*, IEA), o WBCSD[2] (*World Business Council for Sustainable Development*), los edificios y el sector de la construcción son responsables de más de un tercio del consumo mundial de energía, y de hasta un 40 % del total de las emisiones directas e indirectas de CO_2.

Dichas emisiones, tras cierta estabilidad entre 2013 y 2016, alcanzaron su mayor nivel jamás registrado en 2019, produciéndose un crecimiento propiciado por múltiples factores, tales como la creciente demanda de energía destinada a sistemas de calefacción, ventilación y aire acondicionado (HVAC), el uso extendido de combustibles fósiles, la falta de políticas eficaces en eficiencia energética, así como una insuficiente inversión en edificios sostenibles[3].

Ante esta problemática, las propuestas más recientes destinadas a favorecer la eficiencia energética abogan, no sólo por fuentes de energía renovables, sino por la mejora en la gestión energética de las infraestructuras. Así quedó reflejado en el desafío *Secure, Clean and Efficient Energy* del programa Horizonte2020 de la Unión Europea [7], que planteaba entre sus objetivos la

[1] https://www.iea.org/topics/buildings

[2] https://www.wbcsd.org/Programs/Cities-and-Mobility/Resources/Business-realities-and-opportunities-Summary

[3] https://www.iea.org/reports/tracking-buildings-2020

reducción del consumo energético y el desarrollo de áreas urbanas sostenibles dentro del marco de las *Smart Cities*, así como en el clúster 5: *Climate, Energy and Mobility* dentro del nuevo Horizonte Europa[4].

No obstante, la reducción en el consumo energético se ha visto obstaculizada por la demanda energética de los dispositivos HVAC en edificios residenciales y comerciales [16]. Los sistemas HVAC se ocupan de regular la cantidad de energía destinada a la calefacción o refrigeración de los edificios para garantizar el *comfort* de sus ocupantes. La optimización en el control de estos sistemas se plantea, pues, como un reto clave a abordar de cara garantizar la eficiencia energética de los edificios.

Para lograr este cometido, encontramos múltiples propuestas basadas en Ciencia de Datos [31]. Se trata de una disciplina que ha demostrado tener un gran potencial en este ámbito, permitiendo extraer conocimiento implícito relacionado con la demanda energética de los dispositivos HVAC, así como con los factores externos que influyen en dicha demanda. En [31] se muestra una panorámica de las principales aplicaciones de la Ciencia de Datos en control energético de edificios, las cuales incluyen:

- Predicción de la demanda energética de los edificios.
- Optimización de su rendimiento energético.
- Análisis y reacondicionamiento de edificios.
- Prevención de fraudes.
- Análisis económico del consumo energético.

Con respecto a la optimización del rendimiento energético, en los últimos años el control HVAC ha sido abordado desde nuevas perspectivas más flexibles y eficientes que los sistemas de control clásicos. Este es el caso del Aprendizaje por Refuerzo (*Reinforcement Learning*, RL) y su variante más actual: el Aprendizaje Profundo por Refuerzo (*Deep Reinforcement Learning*, DRL), de especial utilidad y repercusión en control HVAC [50]. Actualmente existen numerosas aplicaciones de DRL en este ámbito que han logrado mejorar los resultados ofrecidos por métodos de control convencionales, no sólo reduciendo el consumo energético de los edificios, sino garantizando al mismo tiempo el *comfort* de sus ocupantes [13, 27, 64].

Sin embargo, a pesar de las múltiples ventajas que el DRL puede aportar en el control de dispositivos HVAC, se trata de un campo relativamente inmaduro, donde muy pocas propuestas en la literatura son realmente llevadas

[4] https://ec.europa.eu/info/research-and-innovation/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-europe/cluster-5-climate-energy-and-mobility_en

a la práctica, y donde se carece de *frameworks* específicamente destinados a reproducir y comparar los diferentes algoritmos que conforman el estado del arte. Tal y como Vázquez-Canteli & Nagy [50] evidencian en su reciente revisión de la literatura, la mayor parte de los trabajos en este ámbito son difícilmente reproducibles, lo que complica enormemente la comparación entre diferentes propuestas. Es necesario, por tanto, establecer un marco estandarizado que facilite la evaluación y comparativa de algoritmos de DRL en control HVAC. Este trabajo se presenta como una respuesta ante dicha necesidad.

1.2. Objetivos

Los **objetivos** que serán abordados en este trabajo son los siguientes:

Objetivo 1

Desarrollo de un entorno de ejecución de simulaciones energéticas adaptado para su uso con algoritmos de DRL.

Objetivo 2

Integración, evaluación y comparativa de los resultados obtenidos por diferentes algoritmos de DRL sobre dicho entorno.

Podemos desglosar dichos objetivos en los siguientes **subobjetivos**, los cuales se corresponderán con las actividades principales que se desarrollarán en el proyecto:

Subobjetivo 1

Comprensión de los fundamentos del aprendizaje por refuerzo y su aplicación en control HVAC.

Subobjetivo 2

Extensión de la librería Python Energym haciendo uso de la herramienta Gym de OpenAI.

Subobjetivo 3

Entrenamiento monitorizado de diferentes algoritmos de DRL disponibles en Stable Baselines3 haciendo uso de las herramientas TensorBoard y MLflow.

Subobjetivo 4

Comparativa de los resultados obtenidos para cada uno de los agentes en diferentes contextos de clima y espacios de acciones (discreto y continuo).

Subobjetivo 5

Estudio de la recompensa obtenida por un agente en función de la importancia asignada a confort y consumo energético.

Subobjetivo 5

Ejecución de pruebas de robustez que permitan conocer el rendimiento de los agentes en diferentes climas a los empleados como entrenamiento.

Subobjetivo 5

Aplicación de *curriculum learning* con diferentes agentes y análisis de los resultados obtenidos.

1.3. Planificación

Para llevar a cabo el desarrollo de este proyecto se optó por seguir una metodología **Agile Data Science** [20], basada en una serie de principios clave como el desarrollo iterativo, la continua comunicación con el equipo de trabajo y entrega de prototipos intermedios que permitan al cliente (en este caso, los tutores del proyecto) conocer el estado de su desarrollo. Como se detallará en el Capítulo 3, la realización de este proyecto supuso la integración en un equipo de trabajo profesional y la adaptación a un método de desarrollo basado en la aportación al proyecto mediante *pull requests* e *issues* de GitHub, *integración continua* (CI) y otras prácticas habituales en el marco *DevOps*.

Tabla 1.1: Desglose de gastos del proyecto

Variable	**Valor**
Sueldo bruto anual (€)	25.000
Número de pagas (€)	6
Sueldo bruto mensual (€)	2.083,33
Cotización a la seguridad social (% empresa)	23,60
Cotización a la seguridad social (% trabajador)	4,70
IRPF (%)	14,28
Sueldo neto mensual del empleado (€)	**1.687,91**
Coste mensual para la empresa (€)	**2.574,00**

Con respecto a la **planificación temporal** del proyecto, en la Figura 1.1 se muestra el reparto de días de trabajo y la distribución de las diferentes tareas abordadas. Estas se corresponden con la consecución de los objetivos y subobjetivos anteriormente planteados. Tal y como se muestra en la figura, la duración del proyecto ha sido de un total de **160 días**. Se estima un total de **320 horas** de trabajo, lo cual se ajusta a las 300 horas correspondientes a un Trabajo de Fin de Máster de 12 créditos.

Desde el punto de vista económico, los **gastos del proyecto** son los que se muestran en la Tabla 1.1. Algunos aspectos a considerar en su cálculo:

- La estimación del coste por horas de trabajo se ha hecho de acuerdo a un salario medio aproximado de científico de datos de 25.000 € según la plataforma Glassdoor a fecha de Junio de 2021.
- Los porcentajes de cotización a la seguridad social fueron obtenidos desde la web del Ministerio de Inclusión, Seguridad Social y Migraciones a fecha de Junio de 2021.
- El IRPF ha sido calculado mediante la herramienta online proporcionada por CincoDías para un ingeniero de 23 años, con sueldo bruto anual de 25.000 €.
- Por otro lado, en la estimación de costes indirectos se han tenido en cuenta el consumo eléctrico medio y el *hardware* empleado.

Así, con una duración aproximada de 6 meses, el **coste total** del proyecto se estima en $2.574{,}00 \times 6 = 15.444{,}00$ €.

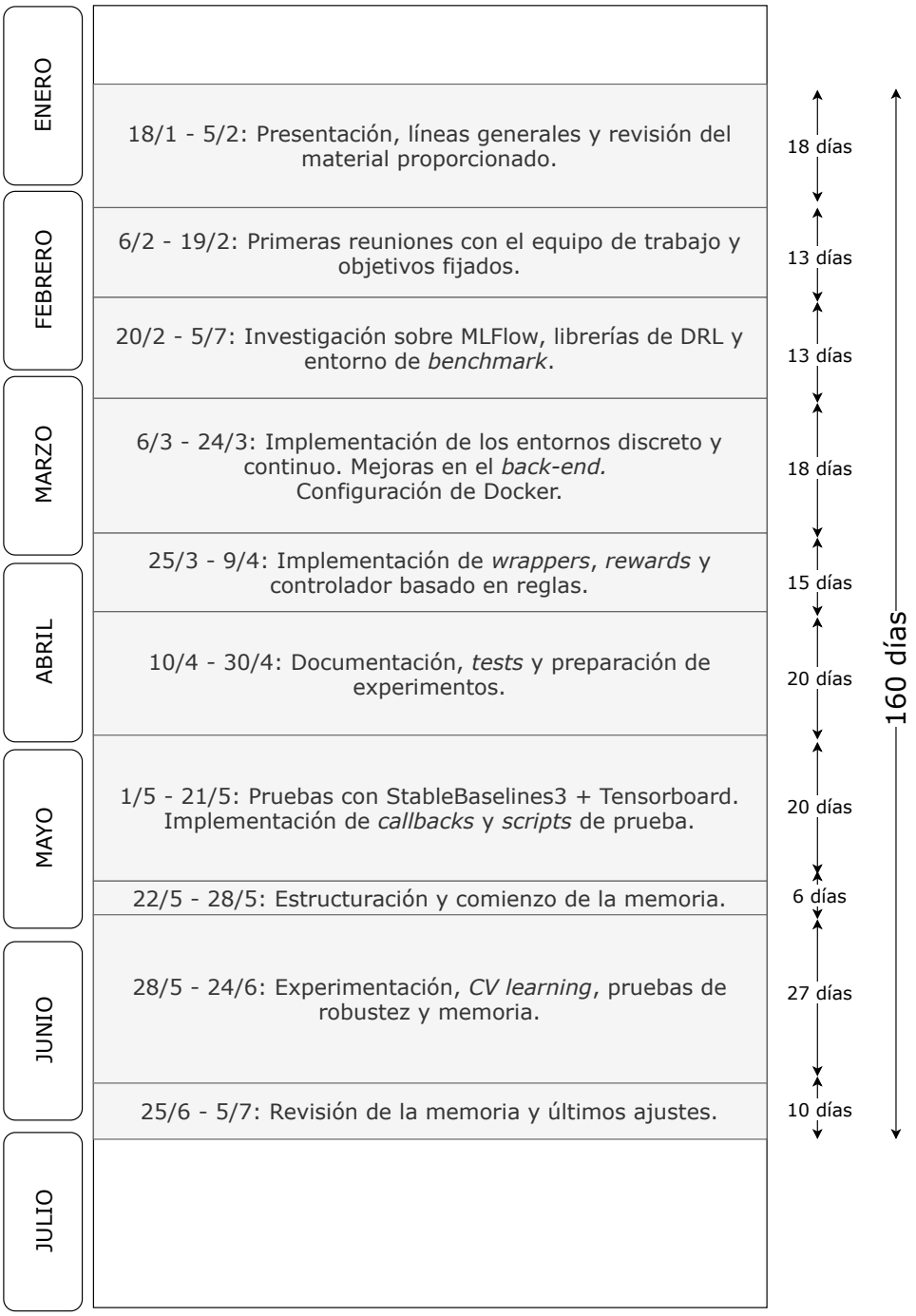

Figura 1.1: Planificación del proyecto

1.4. Estructura

Una vez definidos los objetivos del proyecto, en el Capítulo 2 abordaremos los fundamentos del aprendizaje por refuerzo y sus aplicaciones en control HVAC según el estado del arte. Posteriormente, en el Capítulo 3 se definirá el entorno de simulación *Energym*, así como las aportaciones realizadas al mismo a lo largo de este proyecto. En el Capítulo 4 se describirán los principales resultados obtenidos tras la experimentación con diferentes agentes y entornos. Finalmente, en el Capítulo 5 abordaremos las conclusiones, trabajo futuro y lecciones aprendidas tras el desarrollo de este proyecto.

Todo el código empleado y al que haremos referencia a lo largo de este documento puede encontrarse en el siguiente repositorio de GitHub: `https://github.com/manjavacas/drl-building`. También puede consultarse la documentación de la librería *Energym* elaborada durante el desarrollo de este trabajo: `https://energym.readthedocs.io/`, así como su código fuente: `https://github.com/jajimer/energym`.

Capítulo 2

Marco teórico

En este capítulo se ofrecerá un resumen del estado del arte en el ámbito del aprendizaje por refuerzo y aprendizaje profundo por refuerzo, dando a conocer las nociones básicas de los métodos que los comprenden, así como sus aplicaciones en el control eficiente de dispositivos HVAC.

2.1. Aprendizaje por refuerzo

El **aprendizaje por refuerzo** es un método de aprendizaje computacional centrado en la interacción de un agente con su entorno. Se trata de un aprendizaje iterativo, basado en prueba y error, donde el agente recibe recompensas positivas si sus acciones le conducen a estados deseables. El objetivo del agente será, por tanto, descubrir qué acciones le conducen a la maximización de dicha recompensa [44].

Este modelo de aprendizaje está inspirado en la psicología conductista y el condicionamiento clásico, tratando de modelar el aprendizaje animal desde la perspectiva computacional del aprendizaje automático. Clásicamente, el campo del aprendizaje automático ha sido mayoritariamente definido como la unión de los paradigmas del *aprendizaje supervisado* y *no supervisado*. No obstante, si los comparamos con el *aprendizaje por refuerzo*, encontramos aspectos significativos que permiten diferenciarlos:

- En comparación con el **aprendizaje supervisado**, basado en el aprendizaje a partir de ejemplos previamente etiquetados por un supervisor externo, en el ámbito del *aprendizaje por refuerzo* el agente debe ser capaz de aprender a partir de su propia experiencia. Esto hace que el *aprendizaje supervisado* pierda valor en problemas donde el aprendizaje se produce en base a la interacción.

- Por otro lado, también existen diferencias con respecto al **aprendizaje no supervisado**, centrado en el descubrimiento de patrones en conjuntos de datos no etiquetados. Aunque tanto el *aprendizaje no supervisado* como el *aprendizaje por refuerzo* carecen de ejemplos de comportamiento "correcto", el *aprendizaje por refuerzo* trata de maximizar una señal de recompensa, en lugar de intentar encontrar patrones ocultos en los datos.

Ante esta diferenciación, podemos considerar el *aprendizaje por refuerzo* como un paradigma adicional dentro del aprendizaje automático.

2.1.1. Conceptos básicos

Tal y como se muestra en la Figura 2.1, un problema de *aprendizaje por refuerzo* (en adelante, RL) queda definido por los siguientes componentes:

- Un **agente** que aprende a partir de la interacción con su entorno y que persigue un determinado objetivo. Las interacciones entre el agente y el entorno se producen en una serie de **pasos** o *timesteps*. En cada paso, el agente observa el *entorno*, realiza una *acción* y recibe una nueva *observación* y *recompensa*. Un **episodio** es una secuencia de *pasos*, que parte de un **estado inicial** y termina en un **estado terminal**[1].

- Un **entorno**, definido como todo elemento externo al agente. Se trata de un proceso dinámico que produce datos relevantes para el agente.

- Un conjunto de **estados** u **observaciones** que representan la situación actual del *agente* en el *entorno*. Dependiendo de si la percepción del *entorno* por parte del agente es total o parcial, hablaremos de *estados* u *observaciones*, respectivamente. El espacio de estados puede ser finito o infinito, donde cada estado queda definido por un conjunto finito de variables.

- Una serie de **acciones** que determinan las dinámicas del *entorno*. El *agente* realiza *acciones* sobre el *entorno* que le permiten transitar entre diferentes *estados*.

- Una función de **recompensa**. Se trata de un valor numérico que valora cómo de "buena" es una *acción* o *estado* para el *agente*, por lo que puede corresponderse con una señal positiva (estado/acción deseable)

[1] En caso de encontrarnos ante un *problema continuado*, como es el caso del problema tratado en este trabajo, no existe un estado terminal como tal. En estas situaciones es necesario establecer una limitación temporal o máximo número de *pasos* que determine la finalización de los episodios.

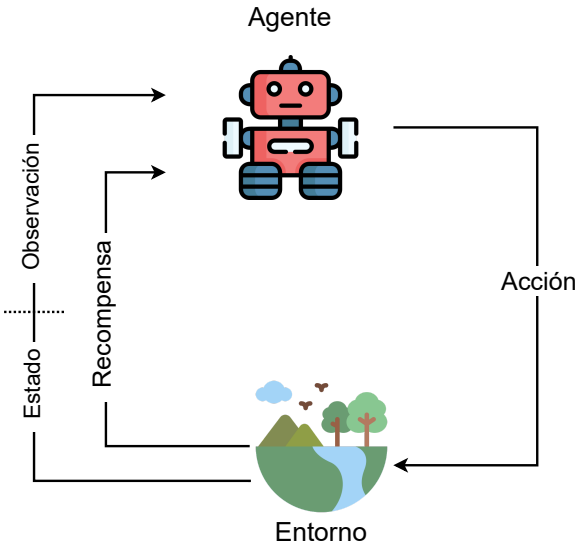

Figura 2.1: Interacción *agente-entorno* en RL

o negativa (estado/acción a evitar). Uno de los principales retos del RL es conocer las consecuencias a largo plazo de las decisiones tomadas, cuya recompensa podría demorarse en el tiempo.

- Una **política** que determina, para cada *estado*, la probabilidad de ejecutar cada una de las *acciones disponibles*. Determina el comportamiento del agente y su preferencia ante diferentes estados o acciones.

A diferencia de otros tipos de aprendizaje, a la hora de abordar un problema desde la perspectiva del RL es necesario establecer un equilibrio entre **exploración** y **explotación**. El agente debe *explotar* las mejores acciones que ha encontrado hasta el momento para maximizar la recompensa obtenida, pero también debe estar abierto a *explorar* alternativas que *a priori* pueden resultar peores, pero que a largo plazo podrían conducir a mejores resultados.

2.1.2. Procesos de decisión de Markov

Formalmente, un problema de RL puede definirse como un **proceso de decisión de Markov** (MDP) compuesto por la quíntupla $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, donde:

- $\mathcal{S}$ es el conjunto de todos los posibles *estados*;
- $\mathcal{A}$ el conjunto o espacio de *acciones* disponibles;

- $\mathcal{R}$ es la función de *recompensa*;
- $\mathcal{P}$ son las *probabilidades de transición* entre estados;
- γ es un factor de descuento $\in [0, 1]$.

Los MDPs parten de la suposición de que las probabilidades de alcanzar un estado s_t dado el estado s_{t-1} y acción a_{t-1} actuales son totalmente independientes de las interacciones previas. Es lo que conocemos como **Propiedad de Markov**, la cual se formula tal que:

$$P(s'|s, a) \doteq P(s'|s, a, S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, ...) \tag{2.1}$$

Dicha propiedad nos permite definir una **función de transición** τ que devuelve el conjunto de probabilidades de transición asociadas a un estado y acción actuales. Esto es:

$$\tau(s, a, s') \doteq p(s'|s, a) = P(S_t = s'|S_{t-1} = s, A_{t-1} = a) \tag{2.2}$$

En problemas totalmente **determinísticos**, la función de transición devolverá un único estado con probabilidad 1 cuando se realice una acción desde un determinado estado. Por el contrario, en problemas **estocásticos** existen diferentes probabilidades de transición a nuevos estados, lo que supone cierta aleatoriedad en las dinámicas del entorno. En ambos casos se asume que la suma de las probabilidades siempre será 1:

$$\sum p(s'|s, a) = 1 \tag{2.3}$$

Como ya hemos adelantado, las transiciones entre estados por parte del agente conducen a determinadas recompensas. Éstas vendrán definidas por una **función de recompensa**: una señal numérica que representa cómo de buena o mala es la transición a cierto estado desde la situación actual del agente.

Existen diferentes formas de representar la función de recompensa dependiendo de las variables involucradas en su cálculo: $\mathcal{R}(s)$, $\mathcal{R}(s, a)$ o $\mathcal{R}(s, a, s')$, siendo esta última la formulación más habitual, al ser más explícita y estar asociada a una menor pérdida de información [32].

Por tanto, la función de recompensa puede ser formulada tal que:

$$r(s, a, s') \doteq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] \tag{2.4}$$

El objetivo del agente será maximizar la **recompensa acumulada** (*return*) desde el inicio de un episodio hasta su final. Siento T el número total de *pasos* de un *episodio*, la recompensa acumulada se define como:

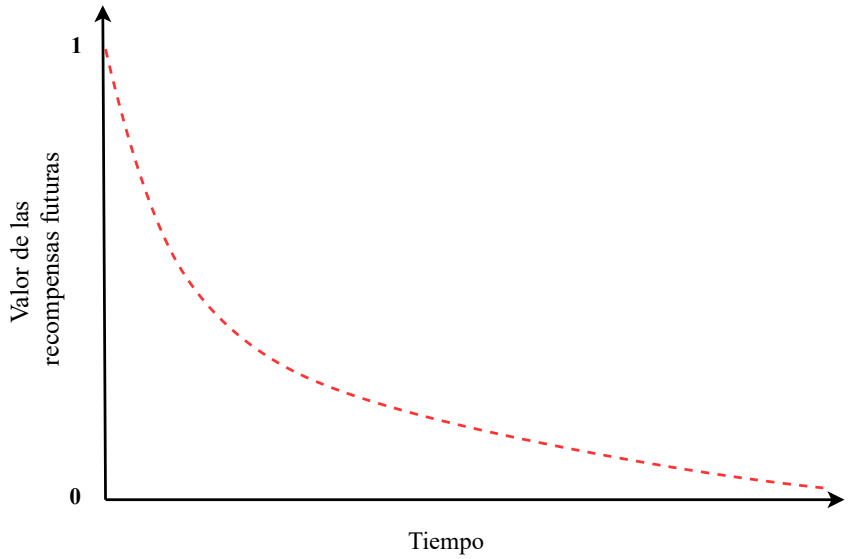

Figura 2.2: Influencia del factor de descuento y el tiempo en el valor de las recompensas futuras

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.5}$$

A la hora de estimar la recompensa acumulada, es común emplear un **factor de descuento**, γ, que ajusta la importancia de las recompensas a lo largo del tiempo. De esta forma, cuanto más tarde se reciba una recompensa, menor peso tendrá en el cálculo de la recompensa total esperada (ver Figura 2.2). Esto nos permite modelar la incertidumbre acerca de recompensas futuras, dando un mayor peso a recompensas inmediatas, o incluso ayudar a modelar problemas no episódicos donde la recompensa del futuro lejano tiende a 0. Así, si añadimos γ al cálculo de la recompensa acumulada, tenemos:

$$\begin{aligned} G_t \doteq & R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-1} R_T \\ = & \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{aligned} \tag{2.6}$$

Otra posible definición de la recompensa acumulada es la siguiente formulación recursiva:

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \tag{2.7}$$

De esta forma, hemos definido los principales elementos que componen

un MPD, contando con un modelo de entorno sobre el que un agente puede desenvolverse.

2.1.3. Políticas y funciones de valor

Una vez contamos con el modelo necesario para formular un problema de RL, es momento de abordar su resolución. Buscamos conocer la secuencia de acciones que maximice la recompensa acumulada al final de un episodio. Es aquí donde entra en juego el concepto de **política**, la cual, como ya adelantábamos, regirá el comportamiento del agente.

Formalmente, una política es una función $\pi(a|s)$ que refleja la probabilidad de emplear una determinada acción $a \in \mathcal{A}(s)$ si el agente se encuentra actualmente en el estado $s \in \mathcal{S}$. En caso de tratarse de una política **determinista**, la acción devuelta por π será siempre única ($probabilidad = 1$), mientras que, en caso de contar con una política **estocástica**, tendremos como salida la distribución de probabilidades de todas las acciones posibles.

Un ejemplo muy simple de política puede ser una *función aleatoria*, la cual asigna la misma probabilidad a todas las acciones posibles partiendo del estado actual. Por lo general, una política aleatoria difícilmente permitirá maximizar la recompensa acumulada, al no verse guiada por los **objetivos** del agente. Un comportamiento más deseable sería, por ejemplo, asignar una mayor probabilidad a aquellas acciones que conduzcan a un estado más favorable y, por tanto, asociado a una mayor recompensa inmediata. Para lograrlo, será necesario definir una serie de funciones que permitan guiar al agente en la maximización de dicha recompensa. Definimos así las funciones *estado-valor* y *acción-valor*:

- **Función estado-valor**, $v_\pi(s)$. Devuelve la recompensa esperada tras aplicar π desde el estado s. Esto es:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] \tag{2.8}$$

  Nótese cómo, de haberlo, el valor del estado terminal será siempre 0.

- **Función acción-valor**, $q_\pi(s, a)$. Devuelve la recompensa esperada tras aplicar la acción a a partir de un estado s siguiendo la política π. Se define tal que:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{2.9}$$

Tanto v_π como q_π pueden estimarse a partir de la experiencia. Por ejemplo, un agente que sigue una política π puede ir manteniendo la media de las

recompensas obtenidas a partir de cada estado, de tal forma que $v_\pi(s)$ acabará convergiendo en el valor real de cada estado. Lo mismo ocurre para q_π si se mantiene la recompensa media obtenida a partir de cada combinación estado-acción.

Visto en qué consiste la función *estado-valor*, podemos desarrollarla recursivamente, dando lugar a la conocida como **ecuación de Bellman** [5]:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in \mathcal{S} \end{aligned} \tag{2.10}$$

La ecuación de Bellman es un principio fundamental de la programación dinámica [5] que expresa la relación entre el valor de un estado y el de sus sucesores. Concretamente, parte del promedio ponderado de todas las probabilidades de transición desde un estado inicial s, y establece que el valor de s debe ser igual al valor descontado del siguiente estado esperado, más la recompensa esperada en el resto de la trayectoria a seguir.

El mismo principio puede aplicarse para la función *acción-valor*, dando lugar a:

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s) \end{aligned} \tag{2.11}$$

Una vez visto en qué consisten las políticas y funciones *estado-valor* y *acción-valor*, estas podrán servirnos para describir, evaluar y mejorar el comportamiento de los agentes. Como ya hemos indicado, el objetivo de un agente de RL es aspirar al **comportamiento óptimo** dentro de su entorno. Esta *optimalidad* nos conduce a la definición de los siguientes términos:

- **Política óptima**, π_*: aquella que, para cada estado, puede obtener recompensas esperadas mayores o iguales a las de cualquier otra política. Podemos comparar políticas en base a sus funciones *estado-valor*:

$$\pi \geq \pi' \leftrightarrow v_\pi(s) \geq v_{\pi'}(s) \tag{2.12}$$

De tal forma que la política óptima será aquella tal que:

$$\pi_* \geq \pi', \forall \pi' \tag{2.13}$$

- **Función *estado-valor* óptima**, v_*: función *estado-valor* que ofrece el máximo valor de todas las políticas para todos los estados.

$$v_*(s) = \max_{\pi} v_\pi(s), \forall s \in S \tag{2.14}$$

- **Función *acción-valor* óptima**, q_*: función *acción-valor* que ofrece el máximo valor de todas las políticas para todo par *estado-acción*.

$$q_*(s,a) = \max_{\pi} q_\pi(s,a), \forall s \in S, \forall a \in \mathcal{A}(s) \tag{2.15}$$

Finalmente, cabe destacar que, mientras múltiples políticas óptimas pueden coexistir para un mismo MDP, únicamente existirá una única función *estado-valor* y *acción-valor* óptimas.

2.1.4. Evaluación y mejora de las políticas

Llegados a este punto, contamos con las herramientas necesarias para poder *evaluar* políticas y funciones de valor, así como para calcular las funciones de valor y políticas óptimas.

El algoritmo de **evaluación iterativa de la política** (*iterative policy evaluation*, o simplemente *policy evaluation*) nos permite conocer cómo de "buena" o "mala" es una política cualquiera en base a su recompensa acumulada esperada. Este algoritmo (ver Algoritmo 1, [44]) permite la aproximación iterativa de la función *estado-valor* de la política evaluada a partir de la fórmula:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{2.16}$$

donde los valores de los estados convergen cuando k tiende a infinito.

Una vez sabemos cómo *evaluar* políticas, podemos compararlas entre sí, tal y como se mostró en 2.12. De hecho, ahora somos capaces de partir de una política cualquiera y tratar de *mejorarla* iterativamente a partir de la función *acción-valor*. Para ello emplearemos el algoritmo de **mejora iterativa de la política** (ver Algoritmo 2, [44]), el cual se define mediante la siguiente fórmula:

Algoritmo 1: Evaluación iterativa de la política

Entrada: π: la política a evaluar
$\mathcal{S}$: el espacio de estados del problema
γ: el factor de descuento
θ: un umbral de convergencia

Salida: V: los valores finales asociados a cada estado

$V(s) = 0, \forall s \in \mathcal{S}+$
$V(terminal) = 0$
repetir
  $\Delta \leftarrow 0$
  para cada $s \in \mathcal{S}$ **hacer**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
mientras que $\Delta < \theta$
devolver V

$$\pi'(s) = \underset{a}{argmax} \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.17}$$

Este algoritmo parte de la estimación de las recompensas para cada acción, a, desde el estado actual, s, y busca aquellas acciones que conducen al agente a estados sucesores, s', con la máxima estimación *estado-valor*.

Si combinamos los procesos de evaluación y mejora de la política de forma cíclica, estaremos mejorando π iterativamente. Es lo que se denomina **iteración de la política** (*policy iteration*), consistente en alternar *evaluación* y *mejora* hasta converger en la política óptima ($\pi \approx \pi_*$).

Una solución alternativa a la iteración de la política es el algoritmo de **iteración de valor** (*value iteration*), consistente en (ver Algoritmo 3):

1. Encontrar la función *estado-valor* óptima, v_*. El algoritmo parte de una función de valor aleatoria que mejora iterativamente hasta alcanzar su valor óptimo.

2. Extraer la política óptima, π_*, asociada a v_*. Una vez contamos con la función de valor óptima, la política óptima π_* será aquella que actúe de forma voraz con respecto a v_*.

En la práctica, tal y como se indica en [44], *policy iteration* converge más rápido que *value iteration* (ya que la política converge más rápido que la función *estado-valor*); no obstante, es computacionalmente más complejo,

Algoritmo 2: Mejora iterativa de la política

Entrada: π: la política a evaluar
$\mathcal{S}$: el espacio de estados del problema
γ: el factor de descuento
Salida: V_*: los valores finales asociados a cada estado
π_*: la política óptima obtenida

```
1  politica_estable ← verdadero
2  para cada s ∈ S hacer
3      a_0 ← π(s)
4      π(s) ← argmax_a ∑_{s',r} p(s', r|s, a)[r + γV(s')]
5      si a_0 ≠ π(s) entonces
6          politica_estable ← falso
7  si politica_estable entonces
8      devolver V ≈ v_* y π ≈ π_*
9  si no
10     aplicar evaluación de la política y repetir
```

Algoritmo 3: Iteración de valor

Entrada: π: la política a evaluar
$\mathcal{S}$: el espacio de estados del problema
γ: el factor de descuento
θ: un umbral de convergencia
Salida: V_*: los valores finales asociados a cada estado
π_*: la política óptima obtenida

```
1  V(s) = 0, ∀s ∈ S+
2  V(terminal) = 0
3  repetir
4      Δ ← 0
5      para cada s ∈ S hacer
6          v ← V(s)
7          V(s) ← max_a ∑_{s',r} p(s', r|s, a)[r + γV(s')]
8          Δ ← max(Δ, |v − V(s)|)
9  mientras que Δ < θ
10 π_*(s) ← argmax_a ∑_{s',r} p(s', r|s, a)[r + γV(s')]
11 devolver V_*, π_*
```

ya que en cada iteración es necesario tanto evaluar como mejorar la política actual.

2.1.5. Exploración y explotación

Muchos de los problemas reales que podemos tratar de resolver mediante RL suponen espacios de estados y acciones muy grandes (o incluso infinitos). Esto hace que los métodos iterativos previamente mencionados sean inviables, ya que no serían capaces de estimar todos los valores y dar con una política óptima en un tiempo razonable.

Como veremos más adelante, los métodos de aprendizaje utilizados para resolver este tipo de problemas deben dar cabida, no sólo a aprovechar el conocimiento del que ya dispone el agente tras haber interactuado con el entorno, sino también a tratar de conocerlo mejor a lo largo del tiempo.

Surge, pues, un nuevo problema hasta ahora no contemplado en la interacción del agente con el entorno: el balance entre **exploración** y **explotación**. Mientras que la *exploración* hace referencia a transiciones entre estados no óptimas destinadas a conocer mejor el entorno, la *explotación* tiene como fin maximizar la recompensa a partir del conocimiento sobre los estados y la política actual del agente.

Cuando un agente explora, "sacrifica" recompensas óptimas inmediatas en pos de descubrir nuevas trayectorias que puedan mejorar sus resultados futuros. No obstante, un agente que únicamente explora el entorno difícilmente convergerá en una buena solución, ya que nunca tomará las acciones que maximicen la recompensa. Es necesario, por tanto, buscar un equilibrio entre *exploración* y *explotación* que permita conocer el entorno de forma progresiva pero sin descuidar la acumulación de recompensas.

Existen diferentes formas de afrontar este equilibrio. Por ejemplo, el método ϵ-*greedy* se basa en actuar siguiendo una estrategia voraz (*greedy*), pero permitiendo tomar acciones aleatorias bajo cierta probabilidad ϵ. Dichas acciones aleatorias suponen *explorar* en lugar de *explotar*.

Finalmente, existen otras estrategias que consideran y/o codifican el factor de incertidumbre de los diferentes estados, de tal forma que, a medida que crece la incertidumbre sobre un estado, mayor probabilidad habrá de transitar hacia él.

2.1.6. Algoritmos clásicos de RL

Hasta el momento, hemos visto cómo emplear *iteración de la política* e *iteración de valor* para extraer la política óptima asociada a un MDP. Decimos, por tanto, que son **métodos dependientes del modelo** del entorno

(*model-based*). Concretamente, las funciones de transición y recompensa son las que definen las dinámicas del entorno, y las empleadas por estos métodos para optimizar la política empleada. Así, la principal ventaja de los métodos basados en modelos es que los estados futuros y las recompensas pueden anticiparse a través del modelo del entorno, lo que permite al agente a lograr una mejor planificación [63].

No obstante, en la mayoría de problemas reales carecemos de un modelo que nos permita conocer de forma anticipada las dinámicas del entorno, ya sea por complejidad o por falta de información[2]. Ante esta problemática, surgen los **métodos libres de modelo**, o *model free*, los cuales estiman la función de valor y la política a partir de la experiencia. Estos modelos son más cercanos a la esencia de aprendizaje por refuerzo, ya que dependen exclusivamente del ensayo-error para aprender el comportamiento óptimo[3]. A continuación, estudiaremos en detalle su funcionamiento.

Monte Carlo

Los algoritmos de tipo **Monte Carlo** (MC) consisten en la ejecución de múltiples episodios que permitan recopilar diferentes trayectorias a lo largo del espacio de estados y acciones. De esta forma, el valor de cada estado será el promedio de las recompensas acumuladas en cada *visita*. La misma filosofía puede aplicarse para estimar el valor de los pares estado-acción, especialmente si no disponemos de un modelo del entorno. Así, una vez conocido el valor de cada estado, o par acción-estado, podremos extraer la política óptima.

Nótese cómo, a medida que se simule un mayor número de episodios, el promedio de las recompensas para cada estado, s, tenderá a converger en $v_\pi(s)$. Lo mismo ocurrirá en el caso de los pares estado-acción, con tendencia a $q_\pi(s, a)$.

Uno de los problemas que pueden darse a la hora de calcular el promedio de las recompensas es que un estado (o par estado-acción) sea visitado múltiples veces en un mismo episodio. Ante esta situación, podemos considerar únicamente el valor obtenido en la primera visita (*first-visit MC*; ver Algoritmo 4) o calcular la media de las recompensas obtenidas (*every-visit MC*).

La exploración sigue siendo un factor importante, ya que adherirnos estrictamente a una política π podría impedirnos conocer trayectorias que

[2]Por ejemplo, podría darse el caso de que no se cumpliese la propiedad de Markov.

[3]Es lo que se denomina un *problema de control*, en contraposición a los *problemas de predicción* ya vistos, donde el objetivo era evaluar políticas o estimar funciones de valor dada una política. La resolución de un *problema de control* consiste en optimizar de forma progresiva las estimaciones de la función *acción-valor* [32].

Algoritmo 4: *First-visit* MC, [63]

Entrada: π: la política a evaluar
Inicializar arbitrariamente $V(s), \forall s \in \mathcal{S}$
$recompensas(s) \leftarrow \{\}, \forall s \in \mathcal{S}$
repetir
    Generar un episodio siguiendo π:
     $S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $t \leftarrow T - 1$
    para cada *paso en el episodio,* $t = T - 1, T - 2, ...0$ **hacer**
        $G \leftarrow \gamma G + R_{t+1}$
        si $S_0, S_1, ..., S_{t-1}$ *no contiene a* S_t **entonces**
            $recompensas(S_t) \leftarrow recompensas(S_t) \cup \{G\}$
            $V(S_t) \leftarrow mean(recompensas(S_t))$
mientras que *no haya convergencia*

afecten a las estimaciones obtenidas. Es aconsejable, por tanto, seguir una política similar a ϵ-*greedy* que dé cabida a dicha exploración. De hecho, el valor de ϵ no tiene por qué ser estático, pudiendo ir reduciéndose a lo largo del tiempo, de tal forma que la exploración disminuya a medida que los valores estimados converjan [32].

Finalmente, aunque los algoritmos de tipo MC son sencillos y eficaces, también presentan ciertos problemas, tales como la necesidad de esperar a la finalización de cada episodio para actualizar los valores estimados, una alta variabilidad en las estimaciones, o tiempos de ejecución muy elevados para poder alcanzar la convergencia en entornos complejos.

Diferencia temporal

El aprendizaje por **diferencia temporal** (*temporal-difference*, TD) es uno de los principales cimientos del aprendizaje por refuerzo, el cual combina ideas de los métodos basados en Monte Carlo y programación dinámica [44]. Además, constituye la base de algunos de los métodos que estudiaremos más adelante.

Tanto TD como Monte Carlo son métodos libres de modelo basados en la experiencia. En el caso de Monte Carlo, es necesario esperar a la finalización del episodio para calcular $V(S_t)$, ya que la estimación del valor de un estado depende de las estimaciones de todos los estados futuros visitados hasta obtener G_t. Veamos como representarlo formalmente:

- Partimos de la idea de que la media $\mu_1, \mu_2, ..., \mu_n$ de una secuencia

$x_1, x_2, ..., x_n$ puede calcularse de forma incremental de la siguiente forma:

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j \tag{2.18}$$

$$= \frac{1}{k}(x_k + \sum_{j=1}^{k-1} x_j) \tag{2.19}$$

$$= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \tag{2.20}$$

$$= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1}) \tag{2.21}$$

- Cuando seguimos un método basado en MC, podemos aplicar esta idea para actualizar de forma incremental el valor de cada estado tras un episodio. De esta forma, para cada estado S_t, dada una recompensa final G_t:

$$N(S_t) \leftarrow N(S_t) + 1 \tag{2.22}$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t)) \tag{2.23}$$

O, de forma general:

$$V(s_{t+1}) \leftarrow V(s_t) + \alpha[objetivo - V(s_t)] \tag{2.24}$$

siendo α un parámetro que representa el "tamaño de paso", o *step-size*, y *objetivo* la n-ésima recompensa obtenida[4].

- En el caso de MC, el objetivo es la recompensa final ($objetivo = G_t$). Esto, finalmente, nos lleva al cálculo incremental de $V(s_t)$:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{2.25}$$

Podemos ver cómo los métodos basados en MC dependen de G_t para actualizar el valor de los estados, lo que supone esperar a la finalización del episodio completo para actualizar dichas estimaciones.

[4] A la expresión $[objetivo - v(s_t)]$ se le denomina *error de la estimación*, y se reduce conforme nos acercamos al *objetivo*.

A diferencia de Monte Carlo, TD no necesita esperar a conocer la recompensa final, G_t, para actualizar $V(s_t)$, sino únicamente esperar al siguiente *paso*, $t+1$. Esto es:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.26}$$

En este caso, *objetivo* $= R_{t+1} + \gamma V(S_{t+1})$, ya que se considera la recompensa R_{t+1} y la estimación $V(S_{t+1})$ del siguiente *timestep*[5,6]. Es lo que denominamos $TD(0)$, o *one-step TD* (ver Algoritmo 5), el cual es un caso especial de *n-step TD*:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha(G_{t:t+n} - V_{t+n-1}(S_t) \tag{2.27}$$

Siendo $G_{t:t+n}$ la recompensa obtenida desde el instante t hasta $t+n$; esto es: $R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$.

Algoritmo 5: $TD(0)$, [44]

Entrada: π: la política a evaluar
α: el *step-size* $\in (0, 1]$

Inicializar arbitrariamente $V(s), \forall s \in \mathcal{S}^+$
$V(terminal) = 0$
para cada *episodio* **hacer**
  Inicializar S
  repetir para cada *paso* en el episodio:
    $A \leftarrow \pi(S)$
    Realizar acción A y observar R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  mientras que S *no sea terminal*

En caso de utilizar *n-step TD*, se nos plantea el interrogante de qué valor de n utilizar. De hecho, podríamos plantear la posibilidad de combinar la información de múltiples *timesteps* (por ejemplo, tomar como valores las medias de las estimaciones entre TD(2) y TD(4)) o incluso de todos ellos, asignando una mayor importancia a aquellos valores estimados más cercanos en el tiempo. Así surge el algoritmo $TD(\lambda)$, donde la recompensa G_t^λ combina todas las recompensas en $G_t^{(n)}$, asignando una ponderación $(1-\lambda)\lambda^{n-1}$ a cada una de ellas. Esto es:

[5] En este caso, $R_{t+1} + \gamma V(S_{t+1})$ es el *objetivo* (*TD target*), mientras que $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ es el *error* (TD error).

[6] Nótese cómo el valor de α pondera la importancia relativa de la nueva estimación de valor frente a la actual. Si $\alpha = 1$, se ignora la estimación previa, mientras que, si $\alpha = 0$, no se produce ninguna actualización.

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)} \tag{2.28}$$

Así, $TD(\lambda)$ generaliza los métodos vistos hasta el momento, ya que equivale a Monte Carlo cuando $\lambda = 1$, y a $TD(0)$ cuando $\lambda = 0$. Existen, a su vez, dos formas de aplicar $TD(\lambda)$: hacia delante (*forward-view*) o hacia atrás (*backward-view*) [32]:

- *Forward-view* $TD(\lambda)$: combina n pasos en una sola actualización. El agente debe esperar hasta el final del episodio antes de poder actualizar las estimaciones.
- *Backward-view* $TD(\lambda)$: divide las actualizaciones de los valores en actualizaciones parciales, que aplica en cada paso. Emplea rastros de elegibilidad (*eligibility traces*) para decidir qué estados actualizar.

A modo de conclusión, TD es capaz de aprender a partir de episodios incompletos, actualizando estimaciones de valor en base a otras estimaciones de valor (es lo que se denomina *bootstrapping*). Esto supone ciertas ventajas con respecto a MC, ya que:

- TD puede aprender de forma *online*, mientras que MC debe esperar al final del episodio hasta conocer la recompensa final obtenida.
- TD presenta una menor variabilidad en las estimaciones con respecto a MC.
- TD puede aprender a partir de secuencias incompletas, mientras que MC solamente aprende a partir de episodios completos.
- TD funciona en entornos continuados donde no existe un estado terminal, mientras que MC únicamente funciona en entornos episódicos.

Finalmente, cabe destacar cómo MC presenta un menor sesgo (*bias*) en las estimaciones, ya que los valores se actualizan directamente en base a la recompensa final, y no con respecto a otras predicciones, como ocurre en el caso de TD.

SARSA

Una vez hemos visto como utilizar TD para resolver el problema de *predicción*, estudiaremos cómo utilizarlo en *control*. Partimos de la idea de que

TD puede emplearse para calcular $q_\pi(s, a)$ de la misma forma que hicimos con $v_\pi(s, a)$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right] \tag{2.29}$$

Esta actualización se produce en cada transición desde estados no terminales S_t. Así, en caso de que S_{t+1} sea terminal, $Q(S_{t+1}, A_{t+1}) = 0$.

Podemos observar cómo el uso de cada uno de los elementos de la quíntupla $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ da lugar al nombre de este algoritmo: **SARSA** (ver Algoritmo 6).

Algoritmo 6: SARSA (*on-policy TD control)*, [44]

Entrada: α: el *step-size* $\in (0, 1]$
$\epsilon > 0$

Inicializar arbitrariamente $Q(s, a), \forall s \in \mathcal{S}^+,\ a \in \mathcal{A}(s)$
$Q(terminal, \cdot) = 0$
para cada *episodio* **hacer**
  Inicializar S
  Elegir A desde S empleando una política derivada de Q (por ejemplo, ϵ-greedy)
  repetir para cada *paso* en el episodio:
    Realizar acción A y observar R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$; $A \leftarrow A'$
  mientras que S *no sea terminal*

Al igual que en el caso de TD, SARSA puede emplearse siguiendo un enfoque *n-step*, dando lugar a *n-step SARSA*, en contraposición a *one-step SARSA* o *SARSA(0)*. En este caso, la regla de actualización empleada es la siguiente:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} + \gamma Q_{t+n-1}(S_t, A_t)\right] \tag{2.30}$$

Llegados a este punto, es necesario diferenciar entre dos tipos de métodos: **on-policy** y **off-policy**:

- Los métodos *on-policy* estiman la recompensa $Q(s, a)$ asociada a cada par acción-estado asumiendo que se continuará siguiendo la política actual.

- Por otro lado, los métodos *off-policy* actualizan el valor de cada par acción-estado asumiendo que posteriormente se seguirá una política voraz. De esta forma, la política actualizada es diferente a la que se asume que será seguida[7].

Por tanto, decimos que SARSA es un método *on-policy* porque la política que se utiliza para actualizar las estimaciones y la que se utiliza para actuar es la misma.

Q-learning

Una alternativa *off-policy* a SARSA es **Q-learning** [55], un algoritmo de control de especial relevancia en el campo del aprendizaje por refuerzo. Decimos que Q-learning es un algoritmo *off-policy* porque la política actualizada es diferente de la política de comportamiento:

- La **política de comportamiento** (*behavior policy*) se emplea para generar interacciones con el entorno.
- Por otro lado, la **política objetivo** (*target policy*) es la política que el agente aprende.

La ecuación empleada por Q-learning para actualizar las estimaciones de cada par acción-estado es la siguiente (ver Algoritmo 7):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (2.31)$$

A diferencia de SARSA, Q-learning aprende una función acción-valor, Q, que se aproxima directamente a la función acción-valor óptima, q_*, independientemente de la política seguida. Así, Q-learning siempre convergerá en la política óptima.

Finalmente, desde un punto de vista computacional, lo habitual es representar los valores de cada par acción-estado a partir de una tabla Q que se actualiza siguiendo el proceso mostrado en la Figura 2.3.

2.1.7. *Deep Reinforcement Learning*

Hasta el momento, hemos estudiado una serie algoritmos de RL "clásicos" capaces de ofrecer muy buenos resultados en entornos de baja complejidad. Simplemente actualizando de forma iterativa una tabla (*lookup table*

[7] Nótese cómo, en caso de seguir una política voraz, la distinción entre *on-policy* y *off-policy* desaparece.

Algoritmo 7: Q-learning (*off-policy TD control)*, [44]

Entrada: α: el *step-size* $\in (0, 1]$
$\epsilon > 0$

Inicializar arbitrariamente $Q(s, a), \forall s \in \mathcal{S}^+,\ a \in \mathcal{A}(s)$
$Q(terminal, \cdot) = 0$
para cada *episodio* **hacer**
  Inicializar S
  Elegir A desde S empleando una política derivada de Q (por ejemplo, ϵ-greedy)
  repetir para cada *paso* en el episodio:
    Realizar acción A y observar R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  mientras que S *no sea terminal*

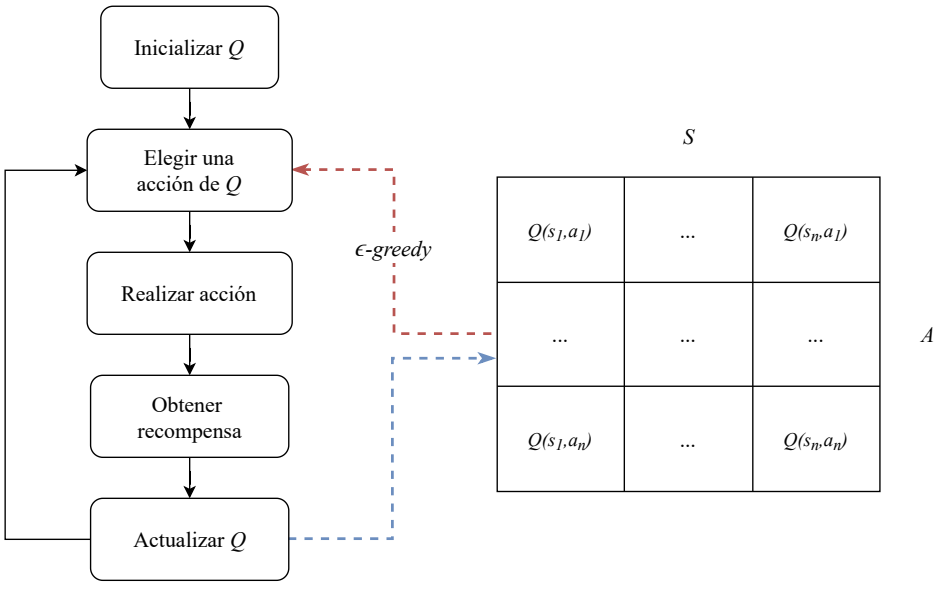

Figura 2.3: Entrenamiento mediante Q-learning

o *memory table*) con diferentes estados, acciones y recompensas ha sido posible optimizar el comportamiento de un agente. No obstante, estos *métodos tabulares* comienzan a ser inviables a medida que la complejidad de los entornos crece [62]. Dicho aumento en la complejidad puede venir dado por un considerable crecimiento en el número de acciones o estados que conforman el entorno, el cual puede tratar de paliarse con métodos *ad hoc* difícilmente generalizables. Por ejemplo, un agente de RL que trabaja con imágenes podría adaptarse para únicamente atender a determinadas porciones de las mismas, en vez de utilizar todos los píxeles que las componen para definir un estado. De esta forma, estaríamos simplificando el espacio de estados, pero la solución propuesta podría no ser válida para otros problemas basados en imágenes donde las regiones a observar sean diferentes.

Ante este tipo de problemas, sería deseable contar con cierta capacidad de abstracción, permitiendo extraer la importancia de las diferentes características que definen un estado. Es aquí donde entra en juego el **aprendizaje profundo** (*Deep Learning*, DL). En los últimos años, la combinación de RL con redes neuronales profundas ha dado lugar a lo que conocemos como **aprendizaje profundo por refuerzo** (*Deep Reinforcement Learning*, DRL), el cual ha abierto la puerta a numerosas aplicaciones del aprendizaje por refuerzo en entornos reales. Así, los algoritmos de DRL combinan lo mejor de ambos mundos, dotando al aprendizaje por refuerzo de poder de representación, eficiencia y flexibilidad del aprendizaje profundo [62].

En las siguientes subsecciones veremos algunos de los algoritmos de DRL más conocidos y empleados en este proyecto, abordando en detalle su funcionamiento y características.

DQN

Los algoritmos basados en ***Deep Q-Networks*** (**DQN**) supusieron un importante hito en el ámbito del aprendizaje por refuerzo, siendo la primera aproximación a lo que hoy conocemos como DRL [30, 29]. Se denominan métodos **basados en valor** (*value-based)*, los cuales se fundamentan en el uso de redes neuronales para aproximar el valor de la función Q, que pasa a ser modelada de forma no lineal. De esta forma, sustituyen la tabla empleada por Q-learning por una red neuronal[8], tal y como se muestra en la Figura 2.4.

Así, la capa de entrada de la red estará compuesta por tantas neuronas como variables conforman un estado u observación, mientras que las neuro-

[8]Esta red puede ser un perceptrón multicapa (MLP) o, en el caso de trabajar con imágenes como datos de entrada, es común emplear redes neuronales convolucionales (CNN) como preprocesamiento previo al MLP. Un ejemplo muy representativo es el de los agentes entrenados para jugar a juegos de Atari [30].

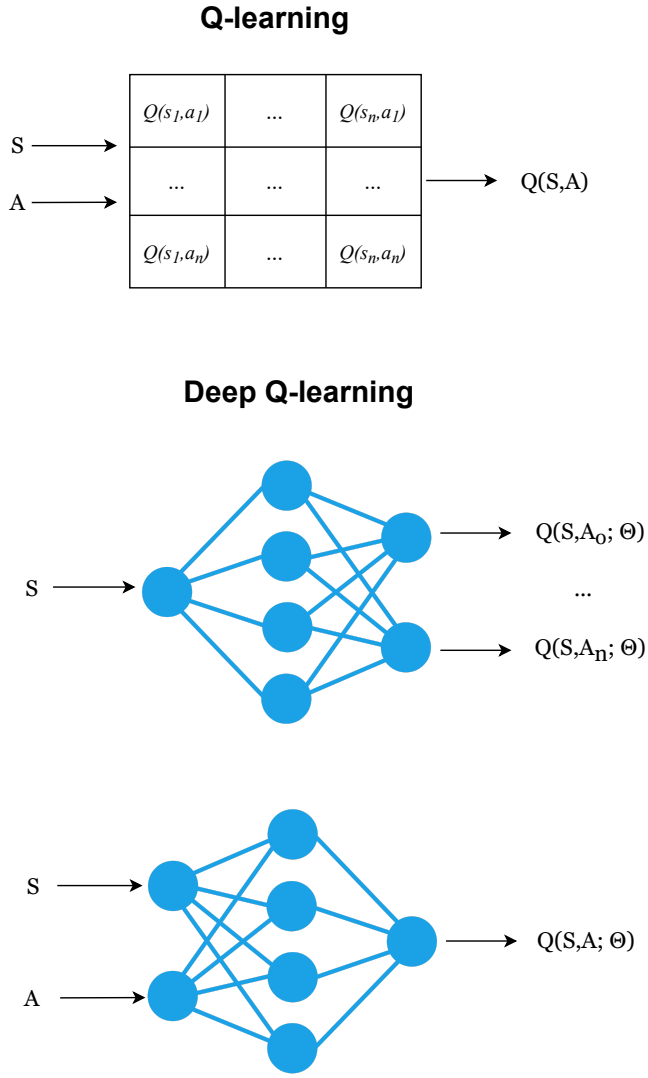

Figura 2.4: Diferencias entre Q-learning y DQN en sus diferentes formulaciones

nas de salida serán tantas como acciones pueda realizar el agente. En base a la entrada s, la siguiente acción a realizar por el agente será la correspondiente a la neurona con mayor valor de salida en la *Q-Network*: $Q(s, a_n; \theta)$, siendo s el estado de entrada, a_n la acción a realizar y θ el conjunto de pesos de la red[9]. Otras formulaciones alternativas permiten tomar pares estado-acción como entrada, para así obtener como salida el valor $Q(s, a; \theta)$ asociado a dicha entrada. Esta última formulación resulta de especial interés en problemas con espacios de acciones continuos, como se verá más adelante.

Si atendemos a cómo estos sistemas aprenden, la función de pérdida (*loss*) empleada por DQN es el error cuadrático medio (MSE) de la diferencia entre el *Q-value* predicho y el valor objetivo, esto es:

$$MSE = \mathbb{E}_\pi[(Q(S, A) - Q(S, A; \theta))^2] \tag{2.32}$$

Una de las principales limitaciones de DQN es el trabajo con espacios de acciones continuos, ya que al existir infinitas posibilidades no podremos obtener la acción óptima que sirva para computar el error. Dicho problema puede tratar de solventarse considerando como entrada el par (s, a), o directamente discretizando el espacio de acciones, lo cual no deja de ser una limitación para este tipo de algoritmos. Esto lleva a que DQN, por lo general, no sea utilizado con espacios de acciones continuos.

El objetivo será, por tanto, medir la diferencia entre el valor real de Q y el obtenido por la red, para finalmente aplicar descenso del gradiente y optimizar la función de error. La actualización de los pesos de la red se lleva a cabo de acuerdo a:

$$\Delta\Theta = \Delta w = -\eta \frac{\partial E}{\partial w} \tag{2.33}$$

$$\frac{\partial E}{\partial w} = 2(Q(S, A) - Q(S, A; \Theta)) \frac{\partial Q(S, A; \Theta)}{\partial w} \tag{2.34}$$

$$\frac{\partial Q(S, A; \Theta)}{\partial w} = \nabla_\Theta Q(S, A; \Theta) \tag{2.35}$$

$$\Delta\theta = -2\eta(Q(S, A) - Q(S, A; \Theta))\nabla_\Theta Q(S, A; \Theta) \tag{2.36}$$

Al no conocer $Q(S, A)$, se aplica la expresión ya vista en Q-learning:

$$Q(S_0, A_0) \leftarrow Q(S_0, A_0) + \alpha[R_1 + \gamma \max_{a \in A} Q(S_1, a) - Q(S_0, A_0)] \tag{2.37}$$

$$\Delta\Theta = \alpha[R + \gamma \max_{a \in A} Q(S', a; \Theta) - Q(S, A; \Theta)]\nabla_\Theta Q(S, A; \Theta) \tag{2.38}$$

[9]También es posible dar cabida a la exploración siguiendo una estrategia ϵ-*greedy*.

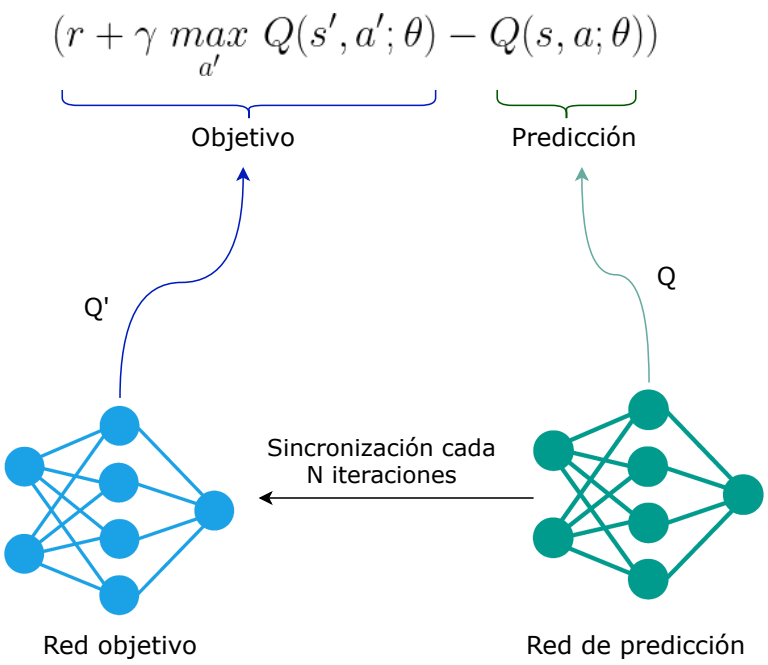

Figura 2.5: Redes empleadas por DQN

Llegados a este punto, un problema que se nos plantea es que la misma red está calculando tanto el valor Q predicho como el valor objetivo. Teniendo en cuenta esto, cuando los pesos se actualicen, los valores Q de salida se actualizarán, pero también lo harán los valores Q objetivo, ya que los objetivos se calculan utilizando los mismos pesos. Así, nuestros valores Q se actualizarán con cada iteración para acercarse a los valores Q objetivo, pero los valores Q objetivo también se moverán en la misma dirección.

Esto nos lleva a introducir una segunda red que haga el entrenamiento más estable, tal y como se propone en el artículo original [29] (ver Figura 2.5):

- Por un lado, la **red de predicción** (*prediction network*) es la encargada de calcular el valor $Q(S, A; \Theta)$.
- Por otro lado, la **red objetivo** (*target network*) se encarga de estimar el valor objetivo, permitiendo el cálculo del error de forma imparcial. Se sincroniza con la red de predicción cada cierto número de iteraciones, lo que hace que sus pesos se actualicen con menor frecuencia.

De forma adicional, es común combinar DQN con una **memoria de experiencias** que permita almacenar y utilizar como entrada diferentes muestras de experiencias pasadas (S_t, A_t, R_t, S_{t+1}) para predecir nuevos valores. Es lo que denominamos **repetición de la experiencia** *experience replay*.

Finalmente, el proceso de aprendizaje mediante DQN queda resumido en el Algoritmo 8. Cabe destacar la existencia de diferentes variantes de DQN orientadas a mejorar el método original, tales como *Double Q-Learning* [48], *Prioritized replay* [39] o *Dueling DQN* [54].

Algoritmo 8: DQN

Entrada: N: número de iteraciones necesarias para sincronizar las redes de predicción y objetivo.

```
Inicializar la memoria de experiencias.
Inicializar la red de predicción aleatoriamente.
Crear la red objetivo como una copia de la red de predicción.
k ← 0
para cada episodio hacer
    Inicializar el estado inicial
    para cada timestep hacer
        a ← seleccionar una acción (vía exploración o explotación) en base al valor de Q predicho.
        Ejecutar la acción a, y observar la recompensa r y el nuevo estado s'.
        Almacenar la tupla (s, a, r, s') en la memoria de experiencias.
        Seleccionar un lote (batch) aleatorio de la memoria de experiencias.
        Introducir batch en la red de predicción
        Obtener la salida de la red objetivo para s'.
        Calcular el error (loss) entre los valores Q de salida y objetivo.
        Aplicar descenso del gradiente para actualizar los pesos de la red de predicción y reducir el valor de loss.
        si k = N entonces
            Actualizar pesos de la red objetivo con los valores de los pesos de la red de predicción.
        k ← k + 1
```

PPO

Los métodos basados en **optimización de la política próxima** (*Proximal Policy Optimization*, o **PPO**) [40, 19] incorporan el gradiente de políticas al aprendizaje por refuerzo. Se trata de un algoritmo *online*, por lo que no emplea una memoria de experiencias. El proceso de aprendizaje es el siguiente:

1. Acumular experiencia hasta completar un lote o *batch*.

2. Utilizar dicha experiencia para optimizar la política.

3. Descartar el *batch* empleado y volver a 1.

Al tratarse de un método *online*, cada *batch* solamente se emplea una vez para actualizar el gradiente, y después se desecha. Esto lleva a que los métodos basados en gradiente generalmente sean menos eficientes en el aprendizaje con respecto a métodos *offline* como DQN.

Veamos en qué consiste este aprendizaje. PPO está basado en la **función de acción-ventaja**, la cual determina cómo de buena es una acción a en base a la recompensa que normalmente se espera obtener siguiendo la política π:

$$a_\pi(s,a) = q_\pi(s,a) - v_\pi(s) \tag{2.39}$$

A partir de esta función acción-ventaja, definimos la función de pérdida como:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{2.40}$$

donde π_θ es la política seguida y $\hat{A}_t$ la función acción-ventaja. La idea será favorecer aquellas acciones que lleven a una función de ventaja positiva y, de esta forma, se incrementará la posibilidad de volver a elegir la acción a a partir del estado s.

Uno de los principales problemas que supone esta forma de actualizar la política es que puede existir una alta divergencia entre la política original y la actualizada. Para evitar esto, PPO hace uso de **TRPO** (*Trust Region Policy Optimization*) y **KL** (coeficiente de divergencia de Kullback-Leibler) para establecer restricciones de divergencia que permitan que las actualizaciones se realicen únicamente dentro de una "región de confianza" [41]. Esto permite que la nueva política obtenida no varíe demasiado con respecto a la anterior. Así, el siguiente ratio:

$$r(\theta) = \frac{\pi_{\theta_{old}}(a|s)}{\pi_\theta(a|s)} \tag{2.41}$$

es empleado en la función objetivo de TRPO tal que:

$$J(\theta)^{TRPO} = E[r(\theta)\hat{A}_{\theta_{old}}(s,a)] \tag{2.42}$$

Si añadimos la restricción que obliga a que este ratio esté entre $1-\epsilon$ y $1+\epsilon$, tenemos la función objetivo empleada por PPO para actualizar la política:

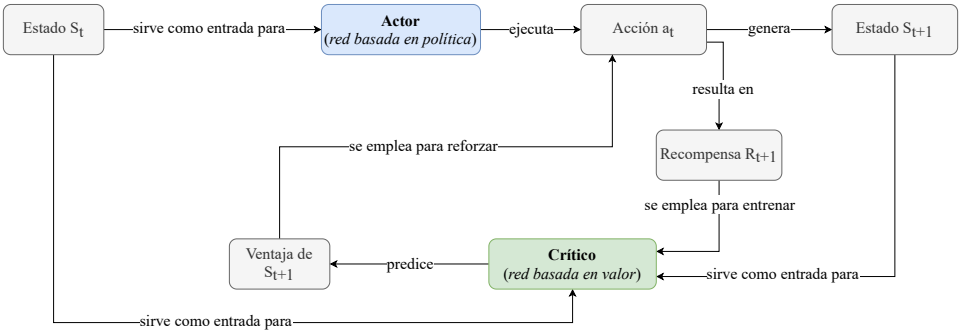

Figura 2.6: Funcionamiento de modelos *actor-critic*. Adaptado de [62]

$$J^{CLIP}(\theta) = \mathbb{E}[min(r(\theta)\hat{A}_{\theta_{old}}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{old}}(s,a)] \quad (2.43)$$

De esta forma, el ratio queda truncado en el rango $[1-\epsilon, 1+\epsilon]$ (mediante la función *clip*), evitando que se produzcan grandes desviaciones en la política, y de tal forma que PPO toma el valor mínimo entre el valor original y el truncado.

A2C

El algoritmo ***Advantage Actor Critic*** (**A2C**) se encuentra dentro de la familia de métodos *actor-critic* (ver Figura 2.6) en los que una red *actor* se encarga de aprender una política, mientras que otra red *crítica* se encarga de evaluarla. De esta forma, la red *critic* aprende el valor de los estados, mapeando cada uno con su valor correspondiente. Esta información es empleada por la red *actor* para mejorar el comportamiento, mapeando cada estado con una distribución de probabilidades que indican la preferencia por unas acciones u otras.

A2C es una versión síncrona de **A3C** (*Asynchronous Advantage Actor Critic*) [28]. Ambos están basados en la interacción entre diferentes nodos (***workers***) encargados de generar experiencias a partir de una copia propia de la función de valor, la política y el entorno, tal y como se muestra en la Figura 2.7. Así, en el caso de A3C, tras la recolección de un lote (*batch*) de experiencias, cada *worker* actualiza un **modelo global** de forma asíncrona, sin existir coordinación alguna con el resto de nodos. Posteriormente, los *workers* actualizan su copia de los modelos y continúan el proceso de aprendizaje.

Por el contrario, A2C es una versión síncrona de A3C que cuenta con un único agente coordinando la interacción con el entorno, tal y como se

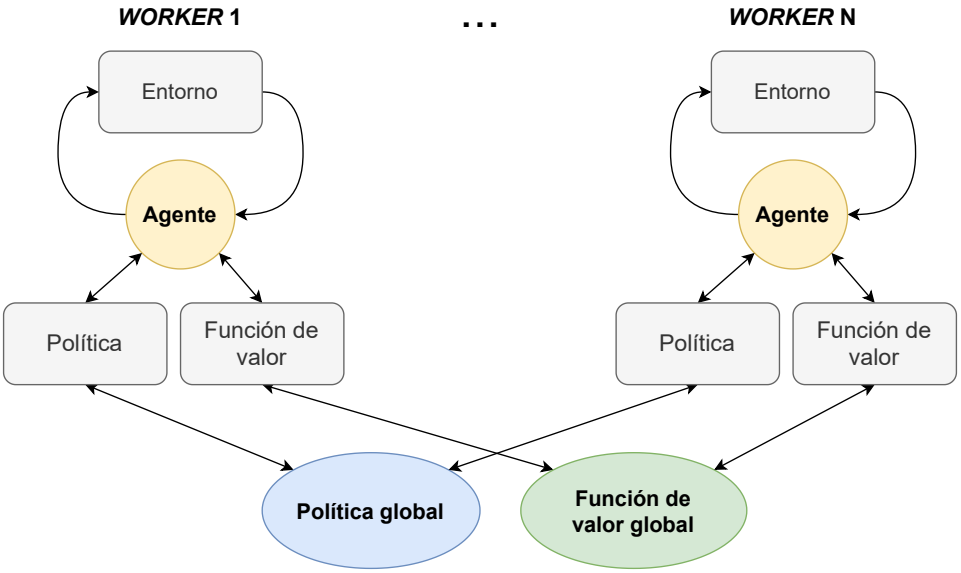

Figura 2.7: Modelo A3C. Adaptado de [32]

muestra en la Figura 2.8. De esta forma, en vez de tener múltiples nodos que actúan y aprenden, pasamos a contar con un único nodo aprendiendo a partir de la experiencia de diferentes actores.

Así, en A2C el nodo **coordinador** espera a que todos los *workers* terminen su trabajo antes de actualizar los parámetros globales. De esta forma, se consigue que en la siguiente iteración todos los *workers* partan de la misma política. La actualización sincronizada del gradiente permite un entrenamiento más cohesionado, evitando que haya agentes empleando diferentes versiones de la política y logrando que la convergencia sea más rápida con respecto a A3C[10].

Finalmente, al tratarse de un método *on-policy*, se sigue un proceso de aprendizaje similar al de PPO. Además, la función de ventaja previamente definida en la ecuación 2.39 vuelve a ser un factor clave para computar el error y actualizar los pesos. Esta puede aproximarse mediante Monte Carlo (2.44), empleando el ya conocido $TD\text{-}error$ $(R_{t+1}+\gamma V(S_{t+1})-V(S_t))$ (2.45) o a partir de una estimación *n-step* (2.46):

[10]Se ha demostrado que A2C es capaz de hacer un uso más eficiente de las GPUs, así como trabajar mejor con *batches* de gran tamaño. De esta forma, A2C consigue un rendimiento igual o superior al de A3C (https://openai.com/blog/baselines-acktr-a2c/).

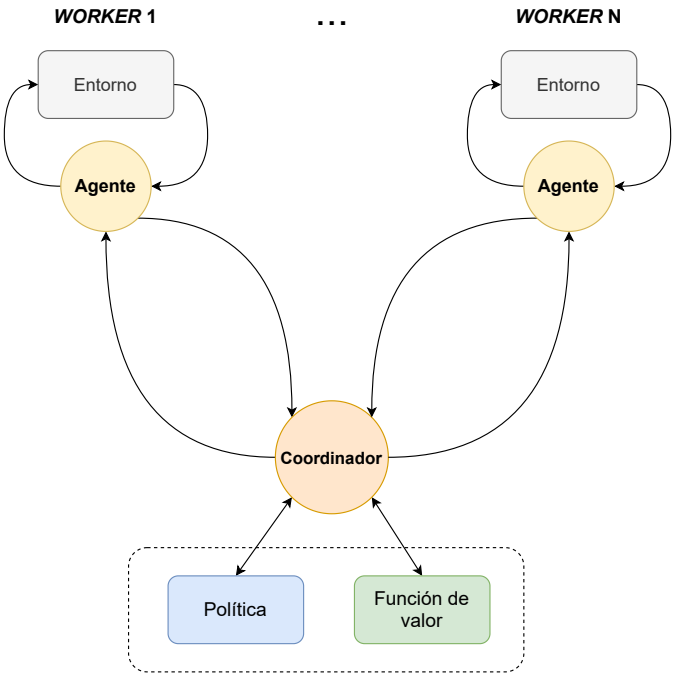

Figura 2.8: Modelo A2C

$$A_\varphi(s_t, a_t) \approx R(s_t, a_t) - V_\varphi(s_t) \tag{2.44}$$
$$\approx r(s_t, a_t, s_{t+1}) + \gamma V_\varphi(s_{t+1}) - V_\varphi(s_t) \tag{2.45}$$
$$\approx \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\varphi(s_{t+n+1}) - V_\varphi(s_t) \tag{2.46}$$

A partir de esta función de ventaja, se lleva cabo la actualización de la red *actor* tal que:

$$\nabla_\theta J(\theta) = \sum_t \nabla_\theta \; log \; \pi_\theta(s_t, a_t) A_\varphi(s_t, a_t) \tag{2.47}$$

Finalmente, el *crítico* se actualiza tratando de minimizar el error TD entre el valor estimado de un estado y su valor real:

$$\mathcal{L}(\varphi) = \sum_t A_\varphi(s_t, a_t)^2 \tag{2.48}$$

DDPG

Deep Deterministic Policy Gradient (**DDPG**) [22] es un algoritmo *off-policy* de tipo *actor-critic* basado en la combinación de **DPG** (*Deterministic Policy Gradient*) [42] con **DQN**. Puede verse como una versión de DQN adaptada a entornos continuos [62].

Hasta el momento hemos visto agentes que, al seguir una política estocástica $\pi_\theta(s)$, asignan cierta probabilidad a cada una de las acciones disponibles desde un estado s. Esto garantiza la exploración en ciertas ocasiones, ya que raramente una acción tendrá una probabilidad nula de ser seleccionada. Así, los métodos basados en valor como DQN tienen como fin converger en una política determinista que actúe de forma voraz tal que: $a_t^* = \underset{a}{argmax}\ Q_\theta(s_t, a)$. La exploración, como ya hemos mencionado, se asegura por medio de una política de comportamiento estocástico, como ϵ-*greedy*, dando lugar a un aprendizaje *off-policy*.

DDPG, por el contrario, tiene como objetivo entrenar una **política determinista** parametrizada $\mu_\theta(s)$. Atendiendo al teorema del gradiente de la política [45] tenemos:

$$J(\theta) = \mathbb{E}_{s\sim\rho_\mu}[R(s, \mu_\theta(s))] \tag{2.49}$$

Así, en vez de tratar de maximizar el valor de la función Q en el siguiente estado para obtener la acción voraz correspondiente (como ocurre en DQN), DDPG trata de aproximar la mejor acción en el siguiente estado utilizando una función de política μ [32]. Por tanto, en comparación con la función de pérdida de DQN:

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}[(r + \gamma \underset{a'}{max}\ Q(s', a'; \theta^-) - Q(s, a; \theta_i))^2] \tag{2.50}$$

$$= \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}[(r + \gamma Q(s', \underset{a'}{argmax}\ Q(s', a'; \theta^-); \theta^-) - Q(s, a; \theta_i))^2] \tag{2.51}$$

la ecuación empleada por DDPG para computar el valor de pérdida es la siguiente:

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}[(r + \gamma Q(s', \mu(s'; \phi^-); \theta^-) - Q(s, a; \theta_i))^2] \tag{2.52}$$

En esta expresión podemos ver cómo μ aprende las acciones voraces determinísticas, a la vez que ϕ actúa como red objetivo. Podemos, por tanto, expresar la función objetivo de DDPG como:

$$J_i(\phi_i) = \mathbb{E}_{s\sim\mathcal{U}(\mathcal{D})}[Q(s, \mu(s;\phi);\theta] \tag{2.53}$$

Una vez contamos con un método para entrenar políticas voraces deterministas, el problema que surge es cómo garantizar la exploración en el entrenamiento, ya que si la política no está entrenada, las acciones estrictamente voraces no son suficientes para obtener un comportamiento óptimo.

Como tantas veces hemos mencionado, es necesario equilibrar exploración con explotación; sin embargo, DDPG aprende una política determinista, por lo que no explorará de forma *on-policy*. Para dar solución a este problema, DDPG añade **ruido** (gaussiano) a las acciones seleccionadas por la política:

$$\mu'(s) = \mu_\theta(s) + \mathcal{N} \tag{2.54}$$

De esta forma, DDPG es capaz de extender DQN a entornos continuos siguiendo un enfoque *actor-critic* y aprendiendo una política determinista óptima.

SAC

El último algoritmo de DRL empleado en este proyecto es ***Soft Actor Critic*** (**SAC**) [17]. Se trata de un método *off-policy* de tipo *actor-critic* que permite optimizar una política estocástica (a diferencia de DDPG) en entornos continuos.

Los métodos *on-policy* como PPO o A3C son ineficientes en su aprendizaje debido a que necesitan información completamente nueva después de cada actualización de la política. Una alternativa más eficiente son los métodos *off-policy* basados en Q-learning, como DDPG. Estos cuentan con un mejor rendimiento en el aprendizaje, ya que pueden aprender de forma eficiente a partir de información pasada almacenada en memoria (*experience replay buffer*). Sin embargo, un inconveniente de este tipo de métodos es que son muy sensibles a los hiperparámetros de los que dependen, lo que dificulta su convergencia. SAC orienta sus esfuerzos en facilitar dicha convergencia.

Una característica fundamental de SAC es la **regularización de la entropía**. La política se entrena para maximizar un equilibrio entre la recompensa esperada y la entropía, la cual actúa como medida de aleatoriedad en el comportamiento del agente. Esto está estrechamente relacionado con el equilibrio entre exploración y explotación: aumentar la entropía da lugar a una mayor exploración, lo que puede acelerar el aprendizaje a largo plazo. También puede evitar que la política converja prematuramente en un óptimo local.

Así, SAC supera el problema de la convergencia alentando a la política a explorar, evitando asignar una probabilidad muy alta a cualquier acción específica de las posibles. Si añadimos la mencionada entropía a la función objetivo del algoritmo, tenemos:

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha\mathcal{H}(\pi_\theta(\cdot|s_t))] \tag{2.55}$$

donde $\mathcal{H}$ es la medida de entropía y α es un coeficiente que controla la importancia que se otorga a dicha medida[11].

SAC hace uso de tres redes para realizar esta tarea de optimización[12]:

1. Una primera red encargada de aproximar la función de valor V_φ (*soft state-value function*) con la siguiente función de pérdida y actualización:

$$\mathcal{L}(\varphi) = \mathbb{E}_{s_t\in\mathcal{D}}[\mathbb{E}_{a_t\in\pi}[(Q_\psi(s_t,a_t) - \log\,\pi_\theta(s_t,a_t)] - V_\varphi(s_t))^2] \tag{2.56}$$

$$\nabla_\varphi\mathcal{L}(\varphi) = \nabla_\varphi V_\varphi(s_t)\,(V_\varphi(s_t) - Q_\psi(s_t,a) + \log\,\pi_\theta(s_t,a)) \tag{2.57}$$

2. Otra red encargada de estimar el valor de la función Q_ψ (*soft Q-value function*). Sus funciones de pérdida y actualización son:

$$\mathcal{L}(\psi) = \mathbb{E}_{s_t,a_t\in\mathcal{D}}[(r_{t+1} + \gamma\,V_\varphi(s_{t+1}) - Q_\psi(s_t,a_t))^2] \tag{2.58}$$

$$\nabla_\psi\mathcal{L}(\psi) = -\nabla_\psi Q_\psi(s_t,a_t)\,(r_{t+1} + \gamma\,V_\varphi(s_{t+1}) - Q_\psi(s_t,a_t)) \tag{2.59}$$

3. Una última red destinada a aproximar la política estocástica π_θ. Esta es entrenada de acuerdo a una función de pérdida que minimiza la divergencia de Kullback-Leibler (KL) [17] entre la política actual π_θ y la función *softmax* aplicada sobre los *soft Q-values*:

$$\mathcal{L}(\theta) = \mathbb{E}_{s_t\in\mathcal{D}}[D_{\mathrm{KL}}(\pi_\theta(s,\cdot)|\frac{\exp Q_\psi(s_t,\cdot)}{Z(s_t)})] \tag{2.60}$$

$$\nabla_\theta\mathcal{L}(\theta) = \nabla_\theta D_{\mathrm{KL}}(\pi_\theta(s,\cdot)|\frac{\exp Q_\psi(s_t,\cdot)}{Z(s_t)}) \tag{2.61}$$

[11] Este coeficiente α se denomina *temperatura*. Su ajuste no es una tarea trivial, lo que ha llevado al desarrollo de nuevas versiones de SAC orientadas ajustar automáticamente su valor [17].

[12] Ecuaciones extraídas de: https://julien-vitay.net/deeprl/.

Finalmente, SAC fue probado y comparado junto a DDPG, PPO y TD3, entre otros algoritmos que constituyen el estado del arte, logrando superarlos en rendimiento y resultados. Por lo general, la ventaja que supone la exploración gracias a la medida de entropía permite que el agente descubra mejores políticas que sus competidores.

2.2. Aplicación de DRL en control HVAC

En los últimos años, el uso de RL y DRL en el control de sistemas de calefacción, ventilación y aire acondicionado (HVAC) ha vivido un crecimiento más que considerable. Con una simple búsqueda en Scopus empleando la cadena de búsqueda "`reinforcement learning`" `AND` "`HVAC`" podemos apreciar el creciente interés en este campo, pasando de 3 publicaciones en el período 1997–2011 a un total de 37 publicaciones en solamente el año 2020 (ver Figura 2.9).

Este crecimiento se ha visto favorecido por los recientes avances en el campo del aprendizaje por refuerzo, y es que en los últimos años se ha demostrado la factibilidad del control HVAC mediante DRL empleando espacios de acciones reducidos y modelos de edificios simplificados [66, 51, 33].

La mayoría de las propuestas en la literatura destinadas a control HVAC mediante DRL hacen uso de diferentes herramientas de simulación energética de edificios, tales como EnergyPlus [57] o Modelica [8], facilitando así el entrenamiento y la experimentación. El uso de este tipo de simuladores se debe a que el entrenamiento de un agente de DRL en un escenario real sería demasiado ineficiente, debido a la necesidad de establecer un mapeo completo entre estados, acciones y recompensas considerando casos extremos [6]. Además, se requiere de un entrenamiento de entre 20 y 50 días para converger en una política de control aceptable [14, 10, 51], lo que dificulta aún más el entrenamiento directo de algoritmos de DRL en entornos reales. Normalmente, los simuladores suelen combinarse con librerías de *deep learning* (ej. TensorFlow/Keras, Pytorch) o DRL directamente (ej. Stable Baselines, RLlib) para pre-entrenar y probar los algoritmos en los entornos simulados antes de su despliegue [52, 47].

2.2.1. Formulación del problema

Uno de los principales motivos que empujan al uso de DRL en control HVAC es la búsqueda de **ahorro energético**. De hecho, los sistemas basados en DRL han logrado conseguir una mayor eficiencia en comparación con los controladores tradicionales basados en reglas [6, 66, 65, 51, 49].

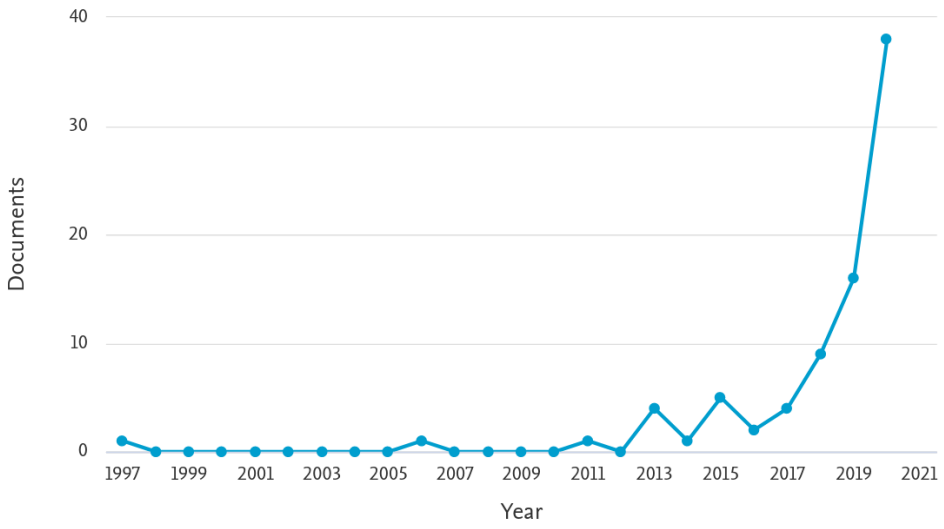

Figura 2.9: Número de publicaciones relacionadas con aprendizaje por refuerzo y control HVAC en período 1997-2021

Por otro lado, los algoritmos de DRL destinados a control HVAC no tienen como único fin reducir el consumo energético, sino también garantizar una serie de requisitos de temperatura o **confort**[13] [56, 14, 11].

Existen diferentes formas de medir el confort, tales como:

- La distancia entre la temperatura actual y la deseada:

$$f(T, \hat{T}) = |T - \hat{T}| \tag{2.62}$$

- La distancia al cuadrado:

$$f(T, \hat{T}) = (T - \hat{T})^2 \tag{2.63}$$

- La distancia a un rango de *confort* objetivo:

$$f(T, T_{lower}, T_{upper}) = |T - T_{inf}|_+ + |T - T_{sup}| \tag{2.64}$$

- La distancia a la temperatura objetivo y al rango de confort según el estándar 55 de ASHRAE [2], el cual define un rango de temperatura aceptable de entre 23–26ºC para verano y 20–23.5ºC para invierno. Esto es:

[13] Es importante destacar que la definición de los estados o recompensas puede contemplar otras variables más allá de la temperatura tales como, por ejemplo, la concentración de CO_2 en el aire o la humedad.

$$f(T, \hat{T}, T_{inf}, T_{sup}) = |T - \hat{T}|_+ + (T - T_{sup})_+^2 + (T - T_{inf})^2 \quad (2.65)$$

- Funciones no lineales como *softplus*, gaussiana truncada, etc.

Mientras que algunos autores consideran que el confort de los ocupantes de un edificio es una restricción a la que siempre hay que dar prioridad, otros se centran en lograr un equilibrio entre dicho confort y el consumo de energético [50].

Por tanto, el objetivo perseguido en el control HVAC de edificios no es otro que el de encontrar la política óptima que maximice el confort y minimice el consumo energético al mismo tiempo. Concretamente, buscamos una política, π, que implique la mayor cantidad de estados deseables posible y la menor cantidad de acciones energéticamente costosas.

Así, partiendo de un conjunto de **estados** u observaciones definidos por las condiciones ambientales del entorno, se plantean los siguientes objetivos:

- Con respecto al consumo energético (medido en kWh), buscamos la política que conduzca a su minimización, esto es:

$$\pi^* = \underset{\pi_\theta}{argmin} \sum_{t=1}^{T} coste(A_t) \quad (2.66)$$

- En el caso del confort, buscamos minimizar la diferencia entre el estado actual del edificio y el objetivo. Podemos definirlo tal que:

$$\pi^* = \underset{\pi_\theta}{argmin} \sum_{t=1}^{T} f(S_t, S_{objetivo}) \quad (2.67)$$

  Un estado será deseable si las variables o condiciones ambientales que lo componen se encuentran dentro de las preferencias del usuario. Decimos que se produce una *violación del confort* si la temperatura en el estado actual se encuentra fuera de los límites definidos por el usuario.

Vistos los objetivos a perseguir, podemos combinar la minimización del consumo energético y la maximización del confort en una sola expresión:

$$\pi^* = \underset{\pi_\theta}{argmin} \sum_{t=1}^{T} w_t \cdot f(S_t, S_{objetivo}) + (1 - w_t) \cdot coste(A_t) \quad (2.68)$$

siendo w_t y $(1 - w_t)$ los pesos asignados a confort y consumo[14], respectivamente. Sobre esta base, podemos definir nuestra función de **recompensa** tal que:

$$r(S_t, A_t) = (1 - w_t) \cdot \lambda_e \cdot coste(A_t) + w_t \cdot \lambda_c \cdot f(S_t, S_{objetivo}) \tag{2.69}$$

donde $coste(A_t)$ es el consumo energético; $f(S_t, S_{objetivo})$ es la función que mide el confort; w y $(1 - w)$ son los pesos asignados a cada parte de la recompensa, y $-\lambda_e$ (medido en $1/W$) y $-\lambda_c$ (medido en $1/^{\text{o}}C$), sendos factores de escala empleados para eliminar las dimensiones del consumo y la temperatura[15].

Finalmente, las **acciones** a realizar sobre el entorno y que darán lugar a una determinada recompensa dependerán del problema al que nos encontremos. En el problema tratado en este trabajo, las acciones consistirán en ajustar los *setpoints* de calefacción y refrigeración, tal y como es habitual en control HVAC [49, 56, 14]. Un *setpoint* marca la temperatura límite a partir de la cual deberá activarse un sistema de calefacción o refrigeración. En caso de fijar un *setpoint* para calor y otro para frío, el objetivo será mantener el edificio dentro del intervalo entre ambos valores. Cabe destacar que, dependiendo del tipo de problema, el espacio de acciones podrá ser:

- **Discreto**: existe un número finito de acciones donde cada acción es una tupla con valores de *setpoints* fijos.
- **Continuo**: cada *setpoint* es un número real a ajustar.

Finalmente, no todos los problemas de control HVAC residen en el ajuste de *setpoints*: otras posibles acciones podrían consistir en regular la potencia de ventilación, o la apertura y cierre de ventanas [9], entre muchas otras.

En conclusión, todo problema de control HVAC destinado a ser resuelto mediante RL cuenta con los siguientes componentes:

- Una serie de **estados** u observaciones compuestos por las condiciones ambientales del entorno: temperatura externa/interna, humedad, luminosidad, concentración de CO_2, etc., en un momento dado.

[14] Como puede observarse, w puede ser variable en el tiempo (w_t) o incluso depender de otras variables como, por ejemplo, el número de ocupantes del edificio.

[15] Es común definir dicha función de recompensa en términos negativos con el fin de que el objetivo sea maximizarla (aproximarla a 0). Esto es: $r(S_t, A_t) = -(1 - w_t) \cdot \lambda_e \cdot coste(A_t) - w_t \cdot \lambda_c \cdot f(S_t, S_{objetivo})$. Se trata de la función de recompensa estándar empleada internamente por Energym.

- Una función de **recompensa** donde se combinan de forma ponderada el consumo energético (a minimizar) y el confort (a maximizar).

- Un espacio de **acciones** consistentes, en este caso, en la regulación de los *setpoints* para calefacción y refrigeración. Dicho espacio de acciones puede ser discreto o continuo.

2.2.2. Estado del arte

Una vez introducido el uso de DRL en control HVAC, pasamos a revisar algunas de las aportaciones más novedosas de la literatura científica en los últimos años.

En el año 1997 tenemos la primera aproximación al uso de RL en control HVAC de la mano de Anderson et al. [1]. En este trabajo, se propuso el uso de RL en combinación con control proporcional-integral (PI) para el control de una bobina de calor (*heating coil*). Un año más tarde, en 1998, tuvo lugar la primera aplicación directa de RL en control HVAC, [34], donde Mozer expuso su *Neural Network House* basada en el control automático de agua caliente, HVAC e iluminación, buscando así reducir el consumo energético y garantizar el bienestar de sus ocupantes.

Desde ese momento, se han ido desarrollando una gran cantidad de trabajos centrados en la reducción del gasto energético de edificios y su gestión inteligente. Algunos ejemplos son: [43], donde se consiguió minimizar el consumo energético en un entorno con múltiples estancias; [24, 23], donde se proponen métodos híbridos orientados a gestionar sistemas de agua fría en grandes centros comerciales, o [10], centrado en la reducción del consumo energético en sistemas de aire acondicionado.

Un factor importante a considerar en la regulación de *setpoints* para calefacción y refrigeración es la ocupación de los edificios. Este factor fue considerado por Barrett & Linder en [4], donde consiguieron reducir en un 10 % los costes energéticos derivados de dispositivos HVAC empleando RL. De forma similar, en el caso de [46, 38], se investigaron métodos basados en *set-back* orientados a relajar los requisitos de temperatura cuando no hay personas en el edificio.

Otros ejemplos de aplicaciones de RL destinadas a incrementar la conservación de energía en edificios son [21, 49, 51, 11, 60, 61, 3] donde tanto el ahorro energético como el confort fueron tenidos en cuenta para optimizar el funcionamiento de dispositivos HVAC en diferentes entornos. En [6], se logró un ahorro de entre un 5 % y un 12 % en el control del sistema de calefacción de un edificio de oficinas haciendo uso de un agente basado en *Double DQN*. Por otro lado, Zhang et al. [66, 65] lograron reducir en un 16.7 % la demanda energética en un edificio de oficinas empleando A3C, mientras que Vázquez-

Canteli et al. [51, 49] lograron un 10 % de ahorro energético aplicando DQN en el control de una bomba de calor. Destaca el caso de Dalamagkidis et al. [11], donde los niveles de CO_2 también se tuvieron en cuenta como métrica de confort.

Un trabajo reciente y de especial interés es el de Azuatalam et al. [3], donde se llevó a cabo una completa revisión del estado del arte en control HVAC mediante RL, proponiendo una arquitectura capaz de alcanzar un 22 % de ahorro energético semanal.

Finalmente, en [12], se propone un modelo *actor-critic* orientado a reducir el error en el ajuste de *setpoints* de sistemas HVAC, mientras que en [18] se llevó a cabo una investigación orientada a probar la eficacia del RL en sistemas de almacenamiento de frío en edificios comerciales.

Como puede verse, contamos con una importante base teórica y práctica sobre la cual se fundamenta el control HVAC por medio de RL y DRL. En esta sección se ha tratado de reflejar parte de la literatura existente, mostrando el amplio abanico de posibilidades que este campo ofrece, y abriendo la puerta al lector a consultar recientes revisiones de la literatura como [50, 26] en caso de buscar una mayor profundización.

Ya mencionamos en el Capítulo 1 que la mayor parte de los sistemas de control HVAC basados en RL y DRL presentes en la literatura son difícilmente reproducibles y comparables [50]. No existe, hasta el momento, ningún *framework* (más allá de los brevemente propuestos en [50], [3]) o *benchmark* destinados a constituir un marco común desde el que comparar modelos de RL y DRL en diferentes climas y entornos. Así, tomando como referencia la arquitectura basada en OpenAI Gym y EnergyPlus propuesta por Zhang et al. en [64, 66], el objetivo de este trabajo será proponer una solución a este problema, tratando de desarrollar un ecosistema de simulación común que pueda ser empleado para validar cualquier nueva aportación al campo.

Capítulo 3

Desarrollo de Energym

En este capítulo abordaremos el proceso de desarrollo de Energym, así como las principales aportaciones realizadas en su implementación y mejora.

3.1. Introducción a Energym

Tal y como se ha adelantado en los anteriores capítulos, el principal objetivo de este proyecto consistió en el desarrollo de un entorno de ejecución de simulaciones energéticas adaptado para su uso con algoritmos de aprendizaje por refuerzo.

Nace así **Energym**, basado en la interfaz de OpenAI Gym[1] y destinado a servir como un marco de pruebas estándar sobre el que ejecutar algoritmos de RL/DRL en diferentes edificios y climas, favoreciendo así su evaluación y comparación. El proyecto tiene su origen en el entorno Gym-Eplus de Zhang & Lam [66, 64], cuyo *back-end* fue tomado como referencia para ofrecer una nueva versión actualizada, escalable y fácilmente reutilizable.

Veamos las principales aportaciones de Energym:

- **Entornos de *benchmarking***. Al igual que los entornos Atari o MuJoCo ampliamente utilizados por la comunidad de RL, Energym incorpora un conjunto de entornos para la evaluación, comparativa y prueba de algoritmos de RL/DRL en control energético de edificios. Estos entornos incluyen diferentes edificios, climas y espacios de acciones.
- **Flexibilidad en la experimentación**. Energym permite la personalización de diferentes aspectos de la simulación, tales como la función

[1] OpenAI Gym es una interfaz de programación destinada al desarrollo de entornos empleados en aprendizaje por refuerzo: `https://gym.openai.com/`.

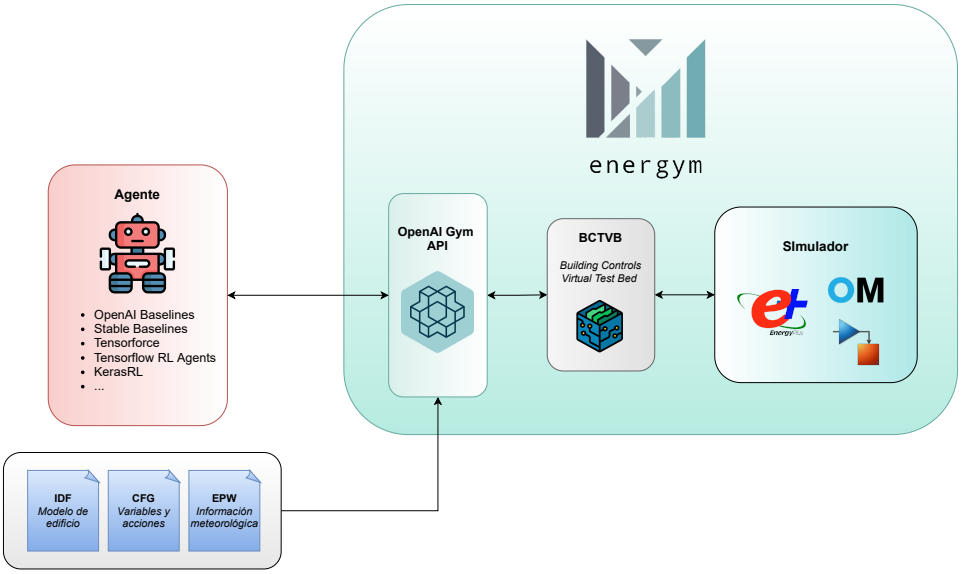

Figura 3.1: Elementos que componen el ecosistema de Energym

de recompensa empleada o la selección de las variables que conforman las observaciones de los agentes.

- **Compatibilidad con diferentes motores de simulación**. Aunque actualmente sólo se ha implementado la conexión con EnergyPlus, el objetivo es que Energym pueda ser empleado con diferentes motores de simulación, como OpenModelica o Simulink.

- **Integración con Stable Baselines3**. Energym cuenta con diferentes funcionalidades adaptadas a esta librería de RL (ej. *callbacks*) con el fin de poder probar fácilmente los entornos implementados.

Una vez definidas las principales características de Energym, pasemos a estudiar su funcionamiento. Como se muestra en la Figura 3.1, el ecosistema de simulación planteado consta de los siguientes elementos:

- Un **agente** que interactúa con el entorno. Su implementación puede llevarse a cabo mediante diferentes librerías de DRL, tales como OpenAI Baselines, Stable Baselines, Tensorflow Agents, KerasRL, etc.

- Una serie de **ficheros de configuración** que serán empleados en la simulación:

  - Un fichero `.idf` (*Input Data File*, IDF) con el modelo del edificio empleado en la simulación.

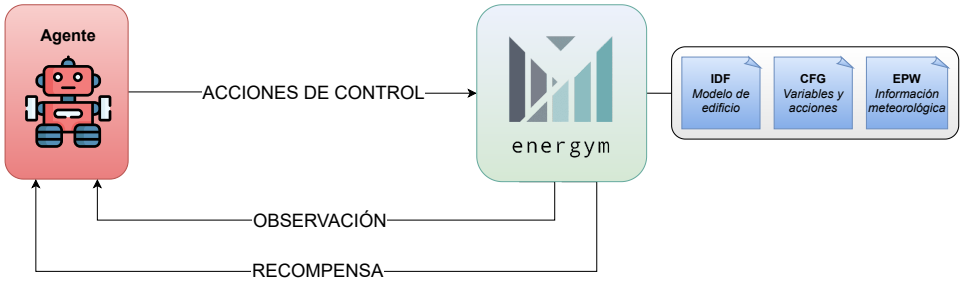

Figura 3.2: Proceso de simulación

- Un fichero `.cfg` con información sobre las acciones, variables de entrada y salida a emplear, sus rangos, etc.
- Un fichero `.epw` (*EnergyPlus Weather*, EPW) con información meteorológica empleada en la simulación.

Todo entorno queda definido por el edificio, variables de entrada y salida, tipos de acciones (discretas o continuas) y clima especificados en estos ficheros.

- El entorno de simulación **Energym**, desarrollado bajo la interfaz de OpenAI Gym y conectado a un simulador (EnergyPlus, OpenModelica, Simulink...) por medio de BCTVB (*Building Controls Virtual Test Bed*). BCVTB es un *middleware open source* que permite el acoplamiento de diferentes sistemas de simulación distribuidos. En nuestro caso, su uso reside en la conexión entre el entorno de simulación y el simulador empleado[2].

Así, la interacción entre estos elementos se realiza tal y como se muestra en la Figura 3.2: el agente interactúa con Energym modificando una serie de variables de entrada y recibe observaciones con el valor de las variables de salida proporcionadas tras la simulación. Por otro lado, en base a las dinámicas del entorno se genera una recompensa que el agente tratará de maximizar.

Cabe destacar que, hasta el momento, no existe ninguna propuesta similar a Energym, lo que supone una importante contribución en este campo. Además, el hecho de distribuirse como código abierto abre la oportunidad a futuras aportaciones por parte de usuarios interesados en colaborar en su desarrollo. El principal objetivo de Energym es ofrecer un entorno de simulación sencillo, eficiente y flexible, de tal forma que cualquier usuario que desee utilizarlo en sus propios proyectos pueda descargar el código y modificar la configuración para adecuar Energym a sus objetivos.

[2]Para más información, consúltese: `https://simulationresearch.lbl.gov/bcvtb`.

Finalmente, Energym abre la puerta a un gran número de posibles experimentaciones: comparativa de algoritmos de RL/DRL sobre unas mismas condiciones; estudio de la influencia del clima en el control energético; análisis de las variables que más influyen en el aprendizaje; comparativa de desempeño para espacios de acciones discretos y continuos, y un largo etcétera.

3.2. Organización y metodología de trabajo

La implementación de Energym llevada a cabo en el marco de este proyecto supuso la colaboración como parte de su equipo de desarrollo. Así, partiendo de un proyecto iniciado el 23 de enero de 2021 y una serie de ideas generales sobre los objetivos a perseguir, se pasó a formar parte de su desarrollo.

El equipo de trabajo del cual se pasó a formar parte durante esta parte del trabajo estuvo compuesto por los siguientes integrantes:

- Javier Jiménez: creador del proyecto, encargado de gestionar el repositorio de Energym, así como de dirigir su desarrollo en líneas generales.
- Alejandro Campoy: incorporado al desarrollo de Energym a partir del 12 de abril.
- Juan Gómez y Miguel Molina: supervisores del proyecto.

La colaboración con el equipo de desarrollo se llevó siguiendo una comunicación continua por medio de herramientas colaborativas como Slack o GitHub. No se trató de un desarrollo al uso, sino que se siguió una forma de trabajo propia de entornos profesionales, siguiendo una metodología de desarrollo basada en *issues/pull requests*, integración continua, versionado y desarrollo en ramas independientes, etc. En las Figuras 3.3, 3.4 y 3.5 se muestra una perspectiva general del repositorio.

En la Figura 3.4 puede observarse cómo la primera parte del desarrollo fue principalmente llevada a cabo por Javier Jiménez, creador del proyecto; posteriormente, el autor de este trabajo tomó el relevo en su implementación (puede verse una mayor frecuencia de contribuciones entre febrero y mayo) para, posteriormente, dar el relevo a Alejandro Campoy como principal desarrollador desde el mes de abril.

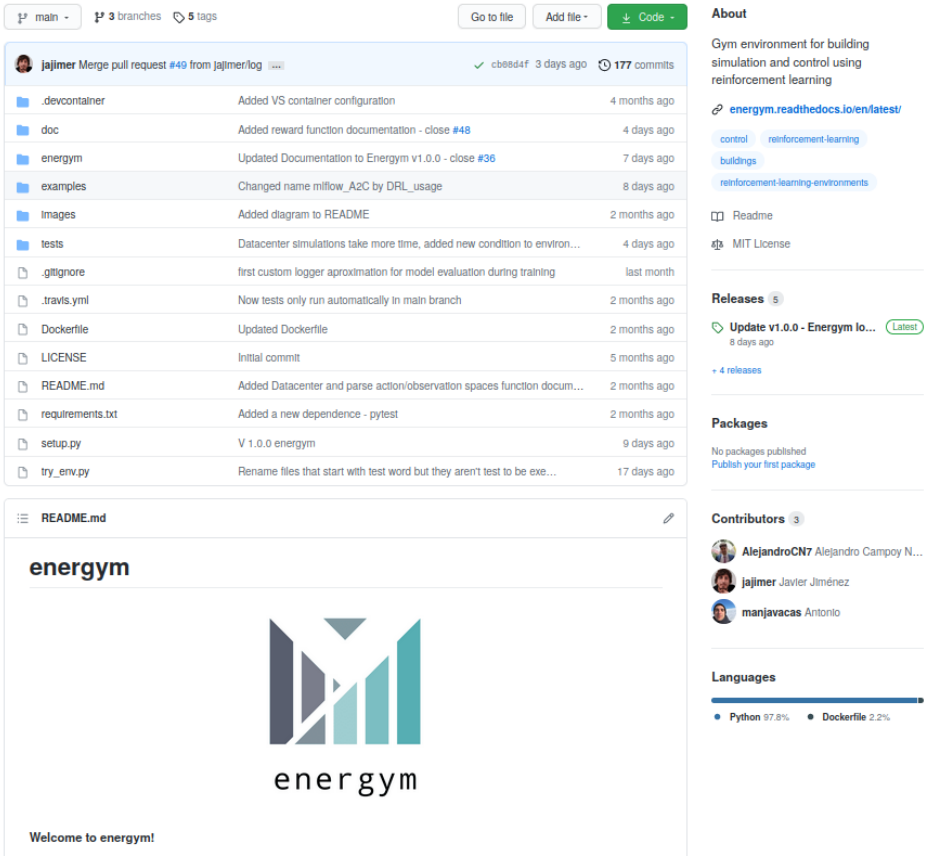

Figura 3.3: Repositorio de Energym en GitHub

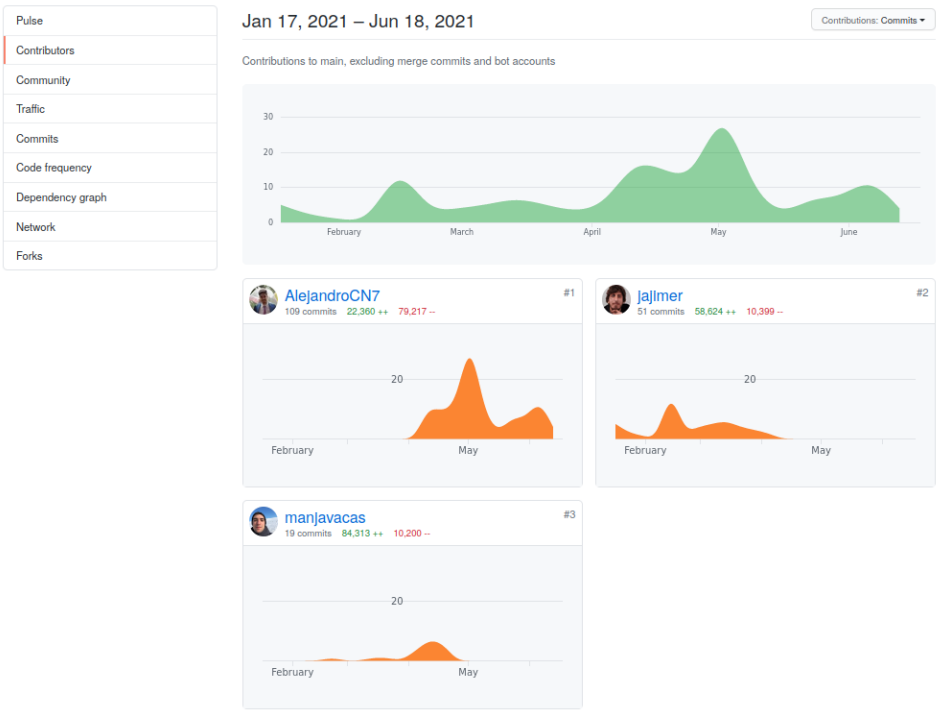

Figura 3.4: Contribuciones al repositorio

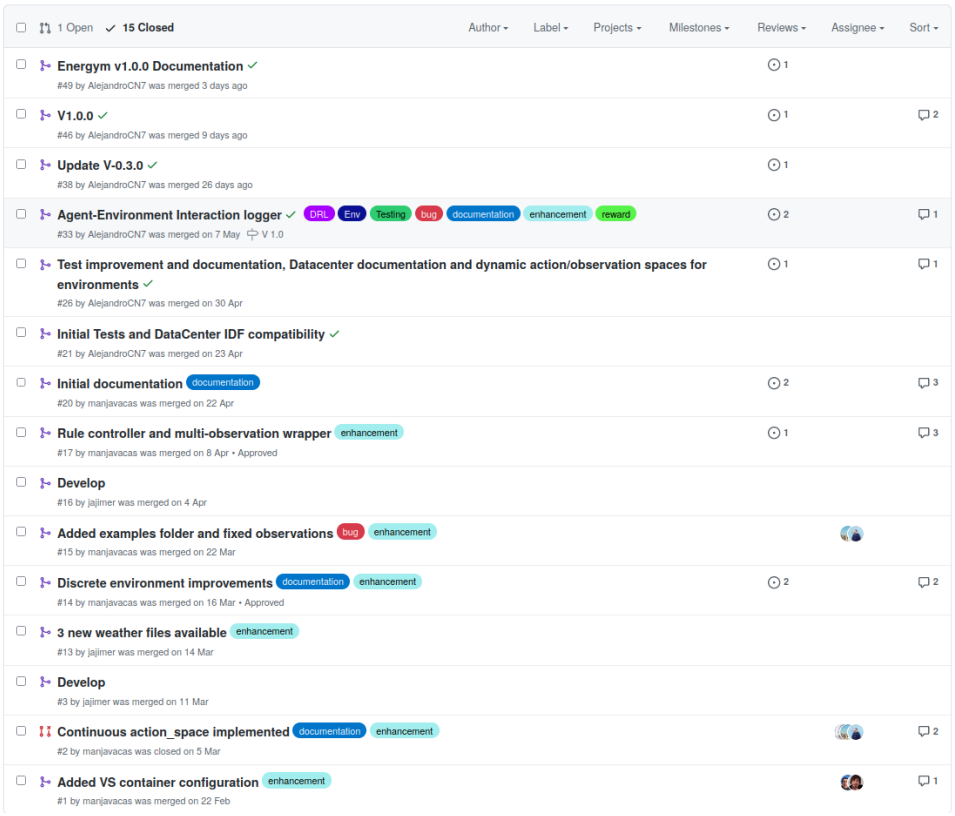

Figura 3.5: Desarrollo basado en *pull requests*

3.3. Contribuciones al desarrollo

Como parte de las aportaciones realizadas en el desarrollo de Energym, destacan las siguientes tareas:

3.3.1. Imagen Docker

Un primer paso fue la implementación de la imagen Docker empleada para desarrollar y utilizar Energym, facilitando la gestión de dependencias y el desarrollo dentro de un entorno aislado.

Docker permite el despliegue de aplicaciones en contenedores, ofreciendo modularidad, una rápida implementación y un desarrollo flexible. Simplemente es necesario definir un fichero Dockerfile donde se especifiquen las operaciones a realizar una vez inicializado el contenedor (instalación de dependencias, declaración de variables de entorno, etc.) para, posteriormente, poder interactuar con el entorno virtualizado que Docker ofrece.

Así, se propuso la implementación de una imagen Docker que permitiese utilizar Energym en un entorno cerrado, gestionándose de forma automática aspectos tales como la instalación de EnergyPlus y BCVTB, o la declaración de las variables de entorno empleadas por Energym.

Finalmente, se incluyó en el repositorio la configuración necesaria para automatizar la construcción y despliegue del contenedor en Visual Studio Code de forma rápida y sencilla.

3.3.2. Desarrollo de entornos de simulación

Posteriormente, se procedió al desarrollo de los entornos basados en espacios de acciones continuos y discretos. Como ya mencionamos en la sección 2.2.1, el problema de regulación de *setpoints* en control HVAC puede formularse considerando un espacio de acciones discreto o continuo. El objetivo perseguido en el desarrollo de Energym fue el de implementar ambos, de cara a evaluar el desempeño de algoritmos de DRL que únicamente operan en un tipo de espacio de acciones (por ejemplo, DQN en entornos discretos, o DDPG en continuos). Otro objetivo considerado fue el de evaluar los resultados obtenidos por un mismo algoritmo en cada tipo de entorno (por ejemplo, A2C).

Para los entornos basados en acciones discretas, los rangos de temperatura se fijaron de acuerdo a los propuestos en [15]. En este caso, un total de 10 configuraciones componen el espacio de acciones[3], donde cada una está

[3]Los espacios de acciones pueden configurarse a voluntad modificando el fichero `.cfg` empleado por Energym.

Tabla 3.1: Espacio de acciones discreto empleado por defecto

N	*Setpoint calor*	*Setpoint frío*
0	15	30
1	16	29
2	17	28
3	18	27
4	19	26
5	20	25
6	21	24
7	22	23
8	22	22
9	21	21

Tabla 3.2: Espacio de acciones continuo empleado por defecto

N	*Setpoint*	Mín.	Max.
0	Calor	15,0	22,5
1	Frío	22,5	30,0

compuesta por una tupla $< setpoint_{calor},\ setpoint_{frio} >$ (ver Tabla 3.1).

Con respecto a los entornos continuos, el ajuste de los *setpoints* quedó acotado a los intervalos que se muestran en la Tabla 3.2. Dentro de dichos intervalos, cada *setpoint* podrá tomar un valor continuo determinando, lo que incrementa más que considerablemente el número de combinaciones acción–estado que podrán darse dentro del entorno.

Una vez visto el tipo de acciones que podemos ejecutar sobre el entorno, otros elementos configurables empleados por Energym son: las **variables** que definen una observación, el **edificio** empleado para la simulación, y el **clima**, los cuales abordaremos en detalle en el Capítulo 4. Además, es posible (y recomendable de cara a favorecer la generalización en el aprendizaje) simular entornos estocásticos donde los datos meteorológicos presentes en el fichero `.epw` son ligeramente modificados en cada episodio a partir de **ruido** gaussiano (con media 0 y desviación estándar $2,5$).

Finalmente, los entornos de simulación considerados en este proyecto son los mostrados en la Tabla 3.3, de acuerdo a las posibles combinaciones de espacio de acciones y clima[4]. Profundizaremos en estos ellos en el Capítulo 4.

[4]Tipos de clima acordes a la clasificación del DOE.

Tabla 3.3: Entornos empleados en este proyecto

Entorno	**Localización**	**IDF**	**Clima**	**Acciones**	**Período**
Eplus-discrete-stochastic-hot-v1	Arizona, USA	5ZoneAutoDXVAV.idf	Hot dry (2B) + ruido	Discrete(10)	01/01 - 31/12
Eplus-discrete-stochastic-mixed-v1	New York, USA	5ZoneAutoDXVAV.idf	Mixed humid (4A) + ruido	Discrete(10)	01/01 - 31/12
Eplus-discrete-stochastic-cool-v1	Washington, USA	5ZoneAutoDXVAV.idf	Cool marine (5C) + ruido	Discrete(10)	01/01 - 31/12
Eplus-continuous-stochastic-hot-v1	Arizona, USA	5ZoneAutoDXVAV.idf	Hot dry (2B) + ruido	Box(2)	01/01 - 31/12
Eplus-continuous-stochastic-mixed-v1	New York, USA	5ZoneAutoDXVAV.idf	Mixed humid (4A) + ruido	Box(2)	01/01 - 31/12
Eplus-continuous-stochastic-cool-v1	Washington, USA	5ZoneAutoDXVAV.idf	Cool marine (5C) + ruido	Box(2)	01/01 - 31/12

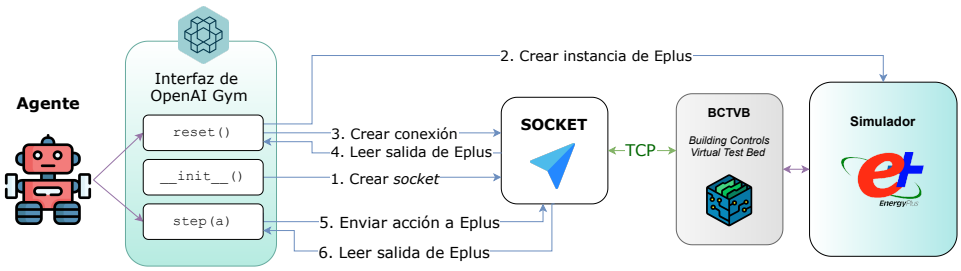

Figura 3.6: Interacción con el *back-end* de Energym, heredado de Gym-Eplus. Imagen modificada de [66]

3.3.3. Gestión de los ficheros de configuración

Esta tarea supuso la lectura y adaptación de los ficheros de configuración (`.idf`, `.cfg` y `.epw`) utilizados por Energym. Por un lado, requirió la búsqueda y uso de librerías destinadas a una lectura eficiente de los ficheros IDF empleados por Energym (por ejemplo, *opyplus*). También se realizaron modificaciones en el modelo de edificio para adaptarlo a la experimentación que posteriormente sería llevada a cabo, como la ampliación del período de simulación a 1 año.

Por otro lado, a las variables devueltas por EnergyPlus como observación en cada *timestep* se añadieron valores adicionales, como la hora o fecha de cada día simulado. Esta información no sólo se incorporó como una variable más a emplear en el aprendizaje, sino también con el fin de añadir a las simulaciones una mayor trazabilidad y facilitar su estudio desde diferentes perspectivas temporales. Por ejemplo: considerar en qué meses el rendimiento obtenido es peor, si existe algún tipo de estacionalidad en la recompensa obtenida, etc.

3.3.4. Limpieza y adaptación del *back-end*

De especial relevancia fue la limpieza y adaptación del código *back-end* heredado del repositorio de Gym-Eplus. El *back-end* de este proyecto fue reutilizado para establecer la conexión entre Energym y EnergyPlus a través de BCVTB, tal y como se muestra en la Figura 3.6.

Al partir de código heredado, fue necesaria una completa interpretación del *back-end* del proyecto original antes de abordar su adaptación y optimización. Posteriormente, se llevó a cabo su reorganización, eliminando aquellas partes de código no empleadas en nuestro proyecto y modificando otras para poder ser utilizadas con Energym. Finalmente, esta parte del proyecto pasó a ser documentada.

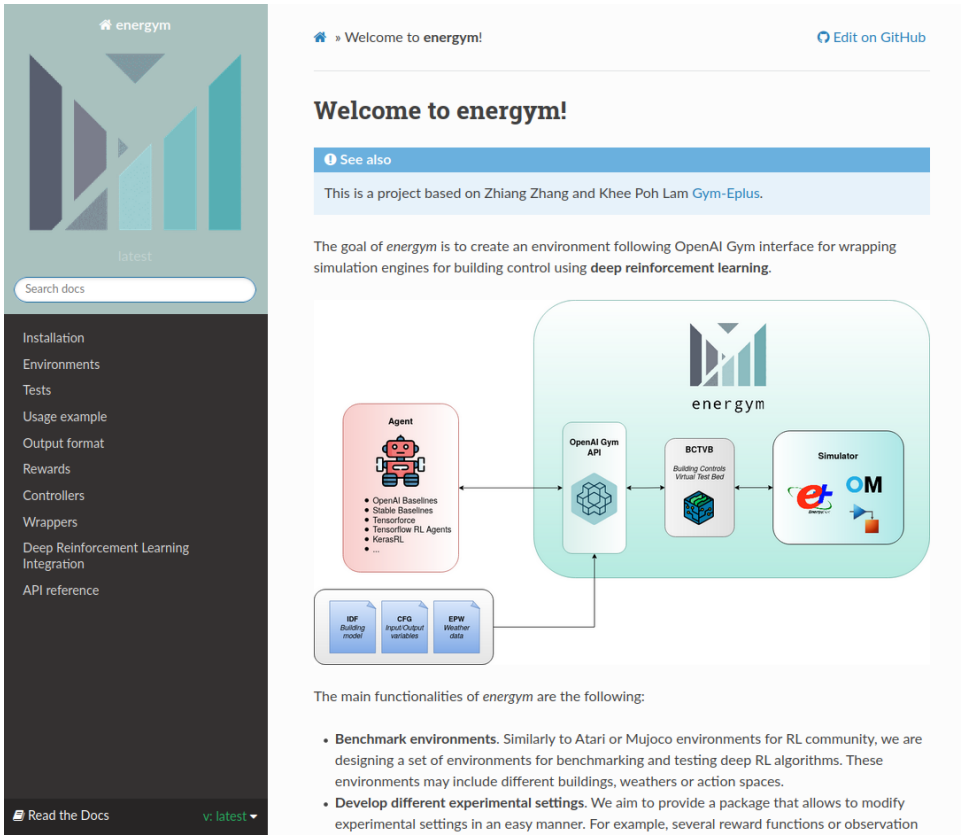

Figura 3.7: Documentación de Energym alojada en *Read the Docs*

3.3.5. Documentación

De cara a contar con un documento de referencia que permitiese detallar el funcionamiento de Energym a la comunidad, se desarrolló la primera versión de la documentación de la librería utilizando *Sphinx*. Así, una vez implementado el núcleo funcional de Energym, se procedió a documentar sus principales componentes. *Sphinx* ofrece numerosas facilidades para elaborar la documentación de código, siempre y cuando se parta de un código comentado en un formato estándar como Numpy o Google.

La documentación generada fue publicada en la plataforma *Read the Docs* (ver Figura 3.7), la cual permite su alojamiento de forma gratuita y la sincronización automática con el repositorio en el que se aloja el proyecto[5].

Finalmente, a la elaboración de la documentación inicial se sumó la implementación de una serie *scripts* de ejemplo destinados a ilustrar el funcio-

[5]La documentación de Energym se encuentra actualizada y puede consultarse en el siguiente enlace: `https://energym.readthedocs.io/`.

namiento de Energym en diferentes contextos. Estos serían empleados para probar la integración de Energym con MLflow y Tensorboard, ejemplificar el uso de *wrappers*, o mostrar la integración de Energym con Stable Baselines3, como veremos más adelante.

3.3.6. Controlador basado en reglas

Un aspecto importante a la hora de poder evaluar la eficiencia de algoritmos de DRL en control HVAC es tener como referencia un controlador basado en reglas con respecto al cual poder comparar. Esta tarea supuso el desarrollo de dicho controlador, cuyo funcionamiento se detalla en el Algoritmo 9. De nuevo, se tomaron como referencia los rangos de temperatura expuestos en [15].

Algoritmo 9: Controlador basado en reglas

Entrada: s: una observación del entorno
Salida: *accion*: la combinación de *setpoints* a ajustar

$t \leftarrow$ obtener temperatura exterior de s
si $t < 15$ **entonces**
    $accion = (19, 21)$
si no si $t < 20$ **entonces**
    $accion = (20, 22)$
si no si $t < 26$ **entonces**
    $accion = (21, 23)$
si no si $t < 30$ **entonces**
    $accion = (26, 30)$
si no
    $accion = (24, 26)$
devolver *accion*

Como puede observarse, un controlador basado en reglas ajusta los *setpoints* en base a la temperatura exterior del entorno. La ventaja de este enfoque basado en reglas es su aplicabilidad en la mayoría de sistemas, la clara interpretación de las reglas, así como su flexibilidad para ajustarse a cualquier tipo de dispositivo HVAC [25].

No obstante, este tipo de controladores suelen ser ineficientes en la práctica, debido a la complejidad de la dinámica térmica de los edificios y a las perturbaciones heterogéneas que puedan darse en el entorno. Tal y como se explica en [56, 58], el rendimiento y fiabilidad de estos enfoques dependen en gran medida de la precisión del modelo térmico del edificio, el cual no sólo debe ser preciso sino también eficiente. Además, la temperatura del edificio

puede verse afectada por muchos factores no contemplados por estos sistemas, como la estructura y materiales del edificio, la humedad, la intensidad de la radiación solar o las ganancias de calor debidas a los ocupantes del edificio. Como resultado, la temperatura del edificio termina por presentar comportamientos aleatorios debido a una modelización incompleta.

Concluimos, por tanto, enfatizando la dificultad para desarrollar un controlador basado en reglas lo suficientemente preciso y eficiente para un control eficaz de la climatización en tiempo real. De aquí parte la idea de desarrollar algoritmos de RL/DRL que traten de mejorar estas propuestas, siempre y cuando se parta de un control basado en reglas a partir del cual poder comparar los resultados obtenidos (tanto en términos de confort como de consumo).

3.3.7. *Wrappers*

OpenAI Gym permite extender los entornos convencionales de forma modular por medio de *wrappers*. Las clases que heredan de `gym.Wrapper` permiten al programador personalizar observaciones (`gym.ObservationWrapper`), acciones (`gym.ActionWrapper`) y recompensas (`gym.RewardWrapper`), sobrescribiendo sus implementaciones por defecto.

En el caso de Energym, se implementaron una serie de *wrappers* de cara a extender las funcionalidades del entorno por defecto[6]:

- `NormalizeObservation`: *wrapper* destinado a normalizar los valores de las variables que componen cada observación. El objetivo perseguido con este *wrapper* es favorecer el entrenamiento de los algoritmos de DRL, reduciendo la variabilidad de los valores empleados.

  Un problema inicial fue decidir qué rangos emplear para normalizar cada valor mediante min–max:

  $$x_{norm} = \frac{x - min}{max - min} \tag{3.1}$$

  Para resolver este problema, se llevó a cabo una aproximación empírica basada la ejecución de múltiples simulaciones con un agente aleatorio. Partiendo de los registros de estas simulaciones, fue posible conocer los valores mínimo y máximo del dominio de cada variable, y emplearlos así para normalizar.

  Finalmente, se comprobó que este *wrapper* mejoraba los resultados obtenidos por los agentes con respecto al uso de observaciones por defecto.

[6]Su implementación puede consultarse en: https://github.com/jajimer/energym/blob/main/energym/utils/wrappers.py

- `MultiObsWrapper`: este *wrapper* permite agrupar múltiples observaciones y utilizar su información de forma combinada para favorecer el entrenamiento. Si bien se trata de un tipo de *wrapper* especialmente útil en entornos como Atari o MuJoCo (ya que sucesivas observaciones agrupadas pueden ayudar a determinar, por ejemplo, el movimiento de un objeto), en el caso de Energym el uso de este tipo de *wrapper* no supuso mejoras significativas sobre el rendimiento de los agentes.

- `LoggerWrapper`: recoge información relacionada con la simulación episodio a episodio y la almacena de forma persistente en formato `.csv`. Resultó especialmente útil para evaluar el rendimiento de los diferentes agentes.

3.3.8. Integración con *Stable Baselines3* y *Tensorboard*

Como se detallará en profundidad en el Capítulo 4, una de las principales tareas realizadas fue la experimentación e integración de Energym con Stable Baselines3. Se trata de una de las librerías más utilizadas en el ámbito del DRL, ya que contiene implementaciones basadas en PyTorch, eficientes y bien documentadas de un importante número de algoritmos de este ámbito.

La mayor parte de los algoritmos disponibles en Stable Baselines3 fueron probados y evaluados empleando Energym en diferentes entornos. Así, a la programación de diferentes *scripts* destinados al entrenamiento y evaluación de los algoritmos, se sumó el desarrollo en paralelo de un *callback* de evaluación destinado a procesar toda la información posible obtenida durante el entrenamiento, tanto desde el punto de vista del entorno como de los agentes[7].

Así, las variables monitorizadas incluyen aspectos de interés como las observaciones percibidas por el agente, las acciones llevadas a cabo en cada *timestep*, los valores de entrenamiento de las redes neuronales, medidas de rendimiento, o las recompensas obtenidas.

Una de las principales ventajas de Stable Baselines3 es que cuenta con integración con TensorBoard, una potente herramienta de visualización orientada, en este caso, a la monitorización del proceso de entrenamiento de los diferentes algoritmos. De esta forma, tal y como se muestra en la Figura 3.8, se pudo llevar a cabo el seguimiento y comparación del entrenamiento de los algoritmos empleados.

Finalmente, veamos la información recogida y mostrada por TensorBoard, la cual se divide en los siguientes grupos:

[7]Para más información, véase: https://energym.readthedocs.io/en/latest/pages/deep-reinforcement-learning.html.

Figura 3.8: Monitorización del entrenamiento mediante TensorBoard

- Registro de las acciones empleadas por el agente durante la simulación.
- Medidas de rendimiento recopiladas episodio a episodio:
  - Porcentaje de tiempo bajo violación de confort, donde la temperatura estuvo fuera de los límites de confort definidos.
  - Valor medio y suma acumulada de las penalizaciones por confort durante un episodio completo.
  - Consumo energético total a lo largo de cada episodio.
  - Valor medio y suma acumulada de las penalizaciones por consumo energético.
  - Recompensa media y acumulada.
  - Duración (en *timesteps*) de cada episodio.
- Valor de las variables observadas durante la simulación (dependientes del entorno).
- Valores normalizados de las variables observadas, en caso de emplearse el *wrapper* de normalización previamente mencionado.
- Métricas ofrecidas por Stable Baselines3 relacionadas con cada algoritmo (por ejemplo, la variación del ratio de exploración en DQN).
- Tiempo de ejecución.
- Información sobre los valores de entrenamiento de las redes neuronales, ofrecida por Stable Baselines3.

- Medidas de rendimiento obtenidas en los episodios de evaluación, en caso de utilizar el *callback* `EvalLoggerCallback` durante el entrenamiento[8]. Este *callback* permite evaluar periódicamente el rendimiento del agente entrenado, empleando un entorno de *test* independiente. A su vez, preserva el mejor modelo obtenido hasta el momento en un directorio indicado por el usuario.

3.3.9. Integración con MLflow

Por último, con el fin de realizar un seguimiento de los experimentos ejecutados y registrar la configuración de los diferentes algoritmos, se decidió emplear MLflow. Se trata de una plataforma *open source* orientada a la gestión del ciclo de vida en proyectos de *machine learning* y ampliamente utilizada en el ámbito del MLOps. Su uso es similar al de otras herramientas como Neptune, Comet, Polyaxon, Valohai, Metaflow o WandB. No obstante, a pesar de las numerosas posibilidades existentes, el hecho de ser *open source*, así como su simplicidad y flexibilidad ante las necesidades de este proyecto fueron las principales razones que llevaron a elegir esta herramienta.

Atendiendo a sus principales funcionalidades, MLflow permite registrar experimentos (*runs*) manteniendo un listado personalizable de las configuraciones empleadas en cada ejecución. También permite comparar su desempeño, elaborar gráficos comparativos entre diversas parametrizaciones, así como almacenar, gestionar y desplegar diferentes versiones de los modelos entrenados.

En el marco de este proyecto, MLflow (y, en especial, su módulo *Model Registry*) resultó de especial interés a la hora de mantener un registro estable de los experimentos realizados y sus configuraciones. Cabe destacar que MLflow no es una herramienta exclusivamente orientada a la gestión de modelos de aprendizaje por refuerzo, pero que contó con la suficiente flexibilidad como para cubrir las necesidades de este proyecto. De esta forma, su principal uso consistió en el registro y estudio de los hiperparámetros empleados en el entrenamiento de los modelos de DRL, así como de las métricas de evaluación empleadas, como la recompensa media o el tiempo de entrenamiento.

Desde un punto de vista práctico, simplemente fue necesario especificar qué parámetros y métricas debían ser empleados durante los experimentos para que MLflow se encargase de su registro y monitorización[9].

[8] Para más información sobre los *callbacks* ofrecidos por Stable Baselines3, véase: https://stable-baselines.readthedocs.io/en/master/guide/callbacks.html.

[9] Véase el ejemplo en código disponible en la documentación de Energym: https://energym.readthedocs.io/en/latest/pages/deep-reinforcement-learning.html#how-use.

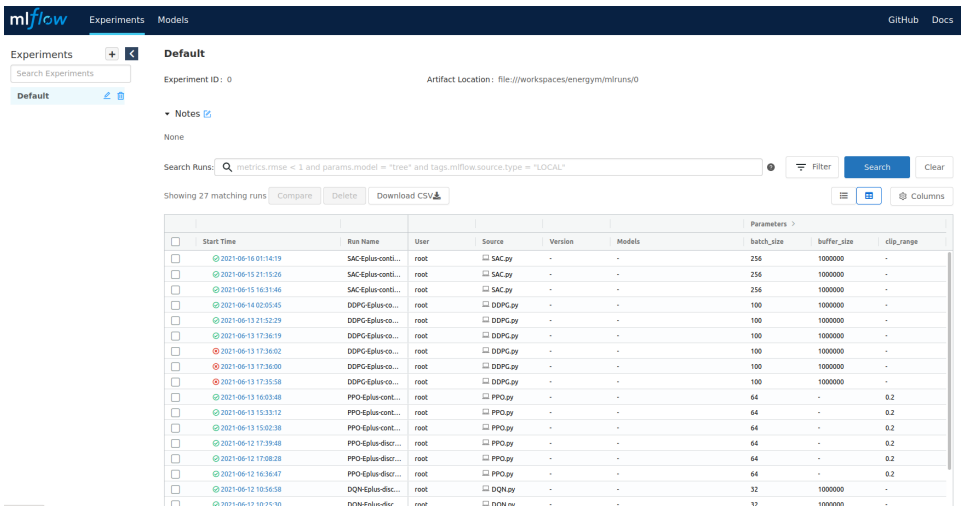

Figura 3.9: Experimentos registrados mediante MLflow

Finalmente, en la Figura 3.9 se muestra el listado de ejecuciones registradas en MLflow, correspondientes a los experimentos que abordaremos en el siguiente capítulo.

Capítulo 4

Experimentación

En este capítulo se detallará la experimentación realizada con Energym, dando a conocer el rendimiento de diferentes tipos de agentes en múltiples entornos y condiciones, mostrando así las posibilidades que ofrece esta herramienta.

4.1. Metodología de experimentación

Los experimentos llevados a cabo en este capítulo serán los siguientes (ver Figura 4.1):

- **Experimentación básica** (sección 4.3): se entrenarán diferentes agentes disponibles en Stable Baselines3 para posteriormente evaluar su desempeño en espacios de acciones discretos y continuos. Los resultados obtenidos tras dicha evaluación se estudiarán desde diferentes perspectivas, enfatizando en las diferencias entre los resultados obtenidos por un agente basado en reglas convencional y los algoritmos de DRL probados.

- **Experimentación avanzada**: estos experimentos tratarán de profundizar en diferentes aspectos avanzados del problema:

  - **Equilibrio confort-consumo** (sección 4.4.1): se estudiará la influencia de las ponderaciones de confort y consumo en el desempeño de los agentes.
  - **Pruebas de robustez** (sección 4.4.2): experimentación destinada a estudiar el desempeño de los agentes en entornos para los cuales no han sido entrenados.

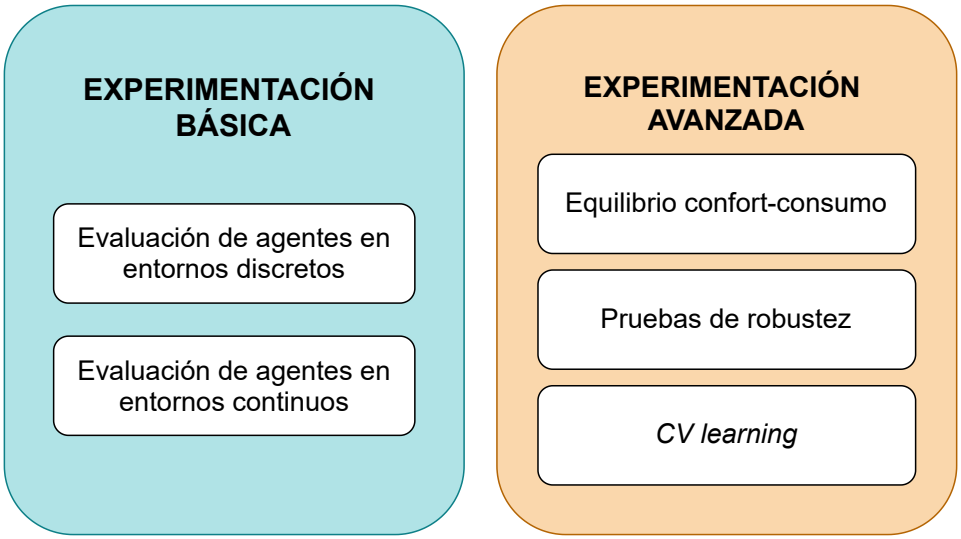

Figura 4.1: Experimentos realizados empleando Energym

- ***Curriculum learning*** (sección 4.4.3): abordaremos un ejemplo de aplicación de *curriculum learning*, estudiando cómo el aprendizaje progresivo puede ser empleado en este ámbito.

4.2. Entorno de simulación

La experimentación se desarrollará sobre una serie de entornos de simulación, compuestos por un modelo de edificio, diferentes tipos de clima y un conjunto de variables de entrada y salida ofrecidas por el simulador.

4.2.1. Edificio

El modelo de edificio empleado fue `5ZoneAutoDXVAV`: un edificio de 5 zonas (4 exteriores y 1 interior) climatizadas por medio de un sistema de aire acondicionado de tipo DX (*Direct Expansion Air Conditioning Unit)*[1].

Se trata de un modelo incluido en el repositorio de modelos de ejemplo de EnergyPlus[2] y previamente empleado en la literatura relacionada [64]. La experimentación realizada se correspondió con el control de temperatura de una de las estancias.

[1]Para más información sobre este tipo de sistemas, véase: https://www.airconditioning-systems.com/direct-expansion-system.html

[2]El listado completo puede consultarse en: https://github.com/NREL/EnergyPlus/tree/v8.6.0/testfiles.

4.2.2. Climas

Para llevar a cabo la simulación energética de un edificio, es necesario conocer las condiciones climáticas que lo atañen. Dichas condiciones pueden resumirse en un conjunto de datos correspondientes a un período de tiempo y lugar determinados, para posteriormente emplearse en la recreación de ese mismo contexto.

La clasificación ofrecida por el DOE (Departamento de Energía de Estados Unidos) diferencia entre 19 tipos de clima[3], desde extremadamente húmedos y cálidos, hasta árticos. Actualmente se encuentran disponibles numerosos repositorios con modelos abiertos de clima destinados a la simulación energética de edificios, como *Climate.OneBuilding*. En este caso, se utilizaron los tres modelos cuyas temperaturas se muestran en la Figura 4.2, estos son:

- **Clima frío** (`cool marine, 5C`): correspondiente al aeropuerto *William R. Fairchild International Airpor* (Washington) entre 1973 y 2005.
- **Clima templado** (`mixed humid, 4A`): correspondiente al aeropuerto *John F. Kennedy International Airport* (Nueva York) entre 1973 y 2005.
- **Clima cálido** (`hot dry, 2B`): correspondiente a la base aérea *Davis-Monthan Air Force* (Arizona) entre 1973 y 2005.

Tal y como se comentó en la sección 3.3.2, la inclusión de ruido en estos conjuntos de datos favorece el entrenamiento y permite a los agentes contar con diferentes variaciones de un mismo clima episodio a episodio. Por tanto, los ficheros de clima empleados para realizar las simulaciones fueron una variante de los ya presentados, a los cuales se añadió cierta estocasticidad.

4.2.3. Variables

Toda simulación cuenta con una serie de variables de entrada (*input variables*) y salida (*output variables*). Mientras que las primeras son modificadas por el agente y tienen una influencia en el entorno, las segundas ofrecen información sobre su situación actual.

En el marco de esta experimentación, un total de 19 variables componen la observación que un agente recibe del entorno[4]. Estas son:

[3]Dicha clasificación puede consultarse en el siguiente enlace: https://www.energycodes.gov/development/commercial/prototype_models

[4]Véase: https://energym.readthedocs.io/en/latest/pages/environments.html#observation-action-spaces

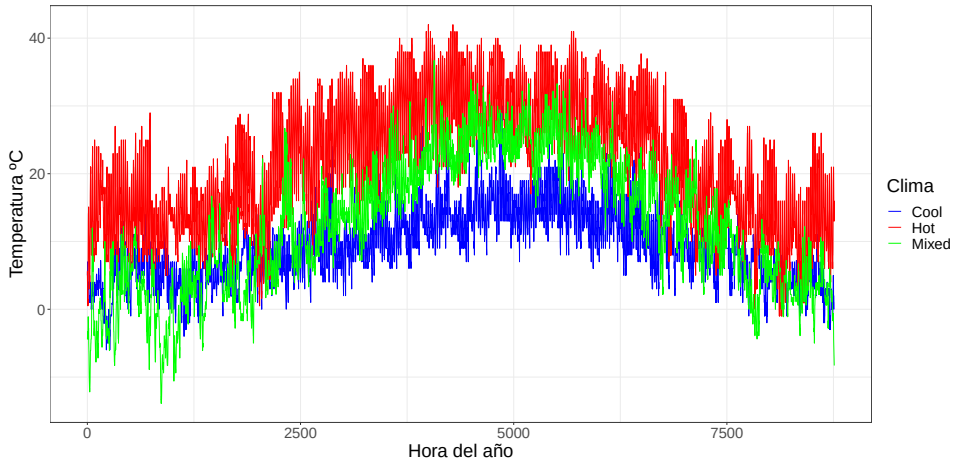

Figura 4.2: Temperaturas (*drybulb*) correspondientes a cada uno de los modelos de clima empleados en la experimentación (se representan los valores medios en cada una de las horas del año)

- **Temperatura seca del aire en el exterior** (*Site Outdoor Air Drybulb Temperature*).
- **Humedad relativa del aire en el exterior** (*Site Outdoor Air Relative Humidity*).
- **Velocidad del viento** (*Site Wind Speed*).
- **Dirección del viento** (*Site Wind Direction*).
- **Radiación solar difusa por área** (*Site Diffuse Solar Radiation Rate per Area*).
- **Radiación solar directa por área** (*Site Diffuse Solar Radiation Rate per Area*).
- **Temperatura de consigna** (*setpoint*) actual para **calefacción** (*Zone Thermostat Heating Setpoint Temperature*).
- **Temperatura de consigna** (*setpoint*) actual para **refrigeración** (*Zone Thermostat Cooling Setpoint Temperature*).
- **Temperatura del aire** (*Zone Air Temperature*).
- **Confort de acuerdo a la temperatura radiante media** (*Zone Thermal Comfort Mean Radiant Temperature*).
- **Humedad relativa del aire** (*Zone Air Relative Humidity*).

- **Confort termal de acuerdo al ropaje** (*Zone Thermal Comfort Clothing Value*).
- **Confort termal de acuerdo al modelo Fanger PPD** (*Zone Thermal Comfort Fanger Model PPD*).
- **Número de ocupantes** (*Zone People Occupant Count*).
- **Temperatura del aire de acuerdo al modelo Fanger** (*People Air Temperature*).
- **Demanda energética de dispositivos HVAC** (*Facility Total HVAC Electric Demand Power*).
- **Día** (*Current Day*).
- **Mes** (*Current Month*).
- **Hora** (*Current Hour*).

Por otro lado, las variables de entrada modificadas por el agente se corresponden con las ya descritas en la sección 3.3.2:

- ***Setpoint* de calefacción** (*Heating Setpoint*).
- ***Setpoint* de refrigeración** (*Cooling Setpoint*).

De esta forma, todo agente recibirá como observación el conjunto de variables inicialmente descrito, y actuará en consecuencia modificando los *setpoints* de acuerdo a sus objetivos de consumo y confort.

4.2.4. Métricas de evaluación

Finalmente, evaluaremos el desempeño de los agentes centrándonos en un conjunto métricas derivadas de las variables de salida previamente descritas. Estas métricas, ejemplificadas en la Figura 4.3, son las siguientes:

- **Recompensa media**: nos indica cómo de bien o mal el agente está garantizando un control eficiente que respete el confort de los ocupantes del edificio y reduzca el consumo.

  Recordemos que, tal y como se introdujo en la sección 2.2.1, la función de recompensa empleada por los agentes es la siguiente:

$$r(S_t, A_t) = -w_t \cdot \lambda_c \cdot Confort - (1 - w_t) \cdot \lambda_e \cdot Consumo \qquad (4.1)$$

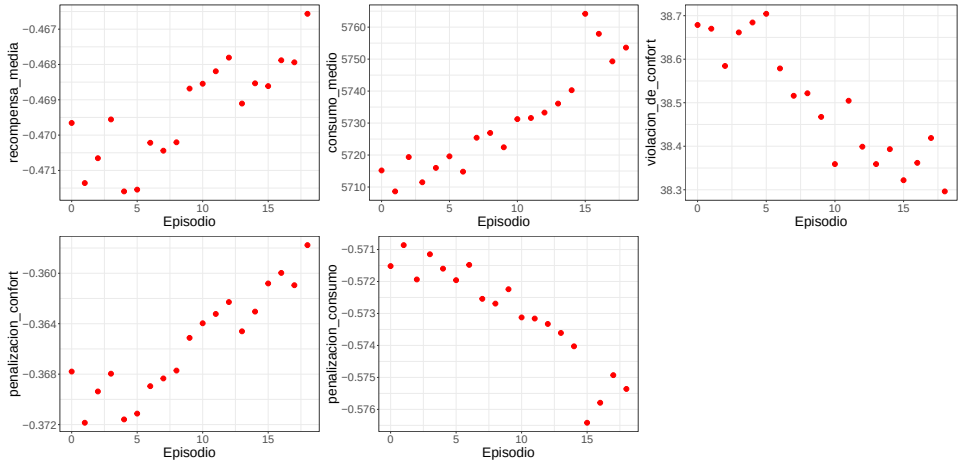

Figura 4.3: Ejemplo de las métricas evaluadas durante la validación de DDPG en el entorno *continuous-stochastic-mixed*

Al estar expresada en términos negativos, buscamos que se mantenga lo más próxima a 0 posible, o lo que es lo mismo: maximizarla. Así, el cálculo de la recompensa media será igual al promedio de las recompensas medias obtenidas por los agentes en cada episodio.

- **Consumo medio**: indica el consumo energético medio a lo largo de la simulación, correspondiente a la variable de salida *Facility Total HVAC Electric Demand Power*.

  Aunque escapa del ámbito de este trabajo profundizar en aspectos referentes al consumo energético, penalización energética, etc., se dota de herramientas a aquellos expertos que deseen hacer un análisis más exhaustivo de estas métricas.

- **Violación de confort**: mide el porcentaje de tiempo de un episodio en el que no se ha respetado el intervalo de temperaturas de confort establecidas por el usuario.

- **Penalizaciones por confort y consumo** indican el valor medio para cada una de las partes de la recompensa episodio a episodio.

  Partiendo de la ecuación 4.1, denominamos "penalización por consumo" a: $(1 - w_t) \cdot \lambda_e \cdot Consumo$, mientras que $w_t \cdot \lambda_c \cdot Confort$ es la "penalización por confort".

Una vez presentadas estas métricas, profundizaremos en su análisis para cada uno de los agentes en la sección 4.3.3.

4.3. Entrenamiento y evaluación de los agentes

Una vez definidas las condiciones de nuestro experimento, en las siguientes subsecciones evaluaremos los resultados obtenidos tras el entrenamiento y evaluación de los agentes en los entornos discretos y continuos.

El *hardware* empleado para entrenar a los agentes consistió en un equipo *HP Pavilion x360 convertible 14-cd0xxx* equipado con:

- Procesador *Intel Core i7-8550U CPU @ 1.80GHz 1.99GHz* de 64 bits.
- 12 GB de RAM.
- Tarjeta gráfica integrada *Intel UHD Graphics 620*.
- Tarjeta gráfica externa *NVIDIA GeForce MX130*.

4.3.1. Espacio de acciones discreto

Los algoritmos empleados en los entornos discretos fueron **DQN**, **A2C** y **PPO**. Su entrenamiento fue monitorizado mediante TensorBoard (Figura 4.4) y se consideró la siguiente configuración:

- Número de episodios de entrenamiento: 20.
- Evaluaciones periódicas durante el entrenamiento cada 5 episodios.
- Episodios empleados para evaluar durante el entrenamiento: 2.
- Hiperparámetros por defecto para todos los algoritmos.
- Misma ponderación para confort y consumo ($w_t = 0.5$).
- Mismo entorno para entrenamiento y evaluación, con variaciones en el clima episodio a episodio en ambos casos.

Atendiendo a la convergencia de los diferentes modelos, A2C logró converger más rápidamente que el resto, seguido por DQN y PPO. Un ejemplo ilustrativo de dicha convergencia es el que se muestra en la Figura 4.5, donde se compara la evolución de la violación de confort para A2C y DQN. En la figura puede verse cómo, a medida que los agentes son entrenados, estos perfeccionan sus políticas, lo que deriva en una reducción de las violaciones de confort y, junto a la reducción del consumo, en una mejora progresiva de las recompensas obtenidas.

Una vez entrenados, se procedió a la validación del mejor modelo obtenido para cada algoritmo de DRL, así como de un agente basado en reglas

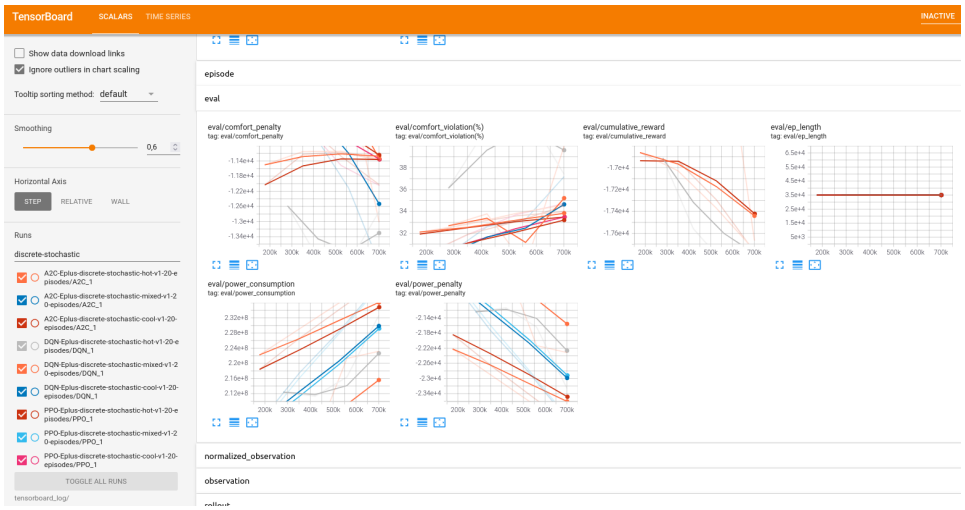

Figura 4.4: Proceso de entrenamiento de los entornos discretos monitorizado mediante TensorBoard

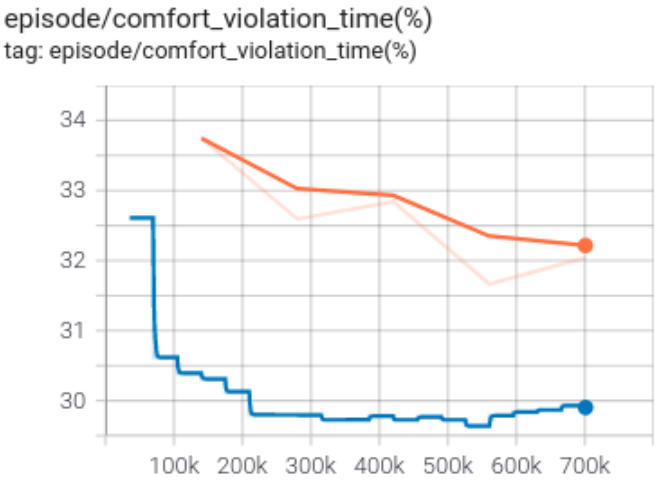

Figura 4.5: Monitorización mediante TensorBoard de la evolución y convergencia de la violación de confort para A2C (azul) y DQN (naranja) en el entorno *discrete-mixed*

(RBC) de cara a poder comparar. Dicha validación consistió en la ejecución de cada modelo durante un total de 20 episodios, lo que sirvió para comprobar la generalización y adaptabilidad de los agentes a medida que el ruido sobre el clima se iba acumulando[5].

Como se detallará en la sección 4.3.3, se observó un rendimiento significativamente bajo (en comparación con el RBC) en prácticamente la totalidad de los casos, pudiendo deberse este bajo rendimiento a la dependencia entre *setpoints* fríos y calientes y al uso un espacio de acciones quizá demasiado limitado. Esto no fue un problema para los agentes en entornos continuos, donde cada *setpoint* pudo tomar valores de forma independiente.

4.3.2. Espacio de acciones continuo

En el caso de los entornos con espacios de acciones continuos, los algoritmos empleados fueron **DDPG**, **PPO**, **A2C** y **SAC**. Por otro lado, las condiciones de entrenamiento (ver Figuras 4.6 y 4.7) fueron las mismas que las de los algoritmos en entornos discretos.

Aunque más adelante profundizaremos en los resultados obtenidos, inicialmente se comprobó que:

- A2C presenta un comportamiento prácticamente similar a RBC en todos los escenarios.
- PPO se comporta de forma similar a A2C y RBC en el entorno templado y ofrece peores resultados para los climas cálido y frío.
- DDPG obtiene peores resultados con respecto al resto de algoritmos en entornos templados, pero destaca junto a SAC en los entornos cálido y frío.
- SAC presenta los mejores resultados, con un rendimiento generalmente superior al resto (especialmente en el entorno templado), aunque similar al de DDPG en los climas cálido y frío.

Un aspecto importante a considerar es cómo la convergencia de los agentes en entornos continuos fue considerablemente más lenta que la de los basados en espacios de acciones discretos. Por lo general, se requiere de un mayor tiempo de entrenamiento para que los algoritmos en entornos continuos alcancen sus mejores resultados.

[5]Evaluaciones de tal longitud no son comunes en la literatura, siendo 1 ó 2 episodios los empleados normalmente para validar un agente ya entrenado. No obstante, como se ha indicado, una validación mucho más prologada permite conocer la capacidad de generalización de los agentes a medida que las condiciones climáticas se vuelven más extremas.

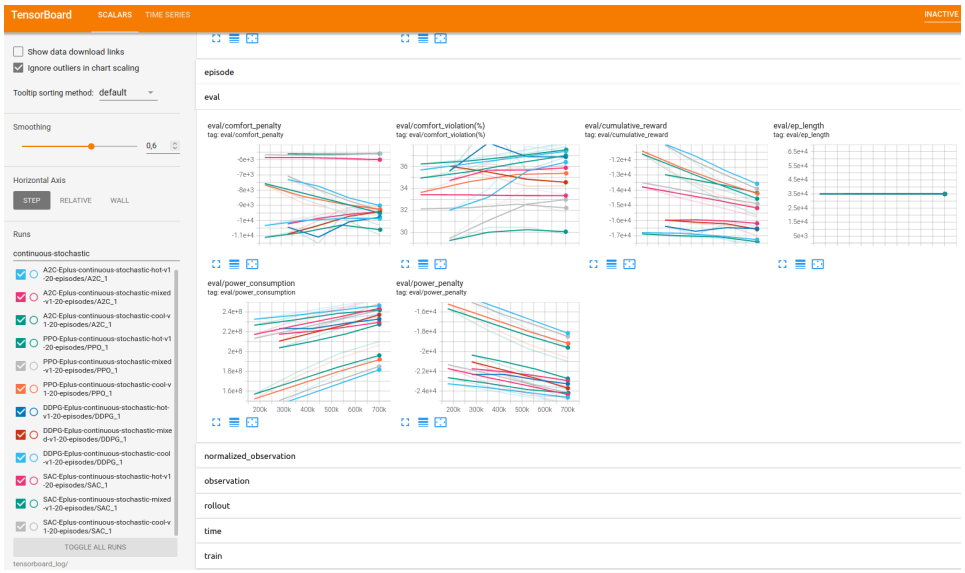

Figura 4.6: Proceso de entrenamiento de los entornos continuos monitorizado mediante TensorBoard

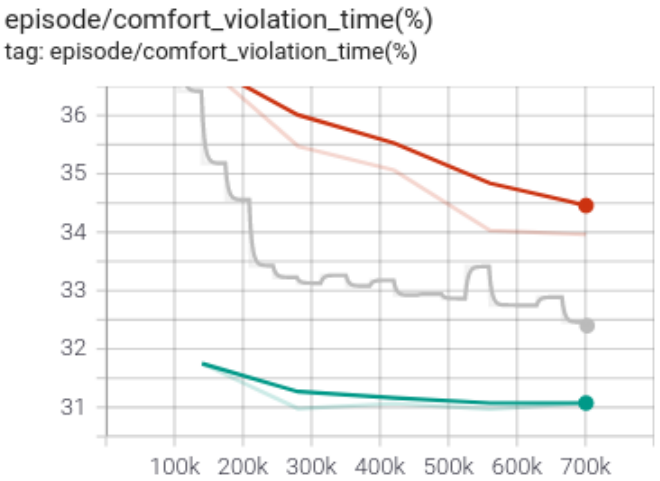

Figura 4.7: Monitorización mediante TensorBoard de la evolución y convergencia de la violación de confort para DDPG (rojo), PPO (gris) y SAC (verde) en el entorno *continous-mixed*

Finalmente, tanto en los entornos discretos como continuos se observó que un mayor tiempo de entrenamiento supone, a largo plazo, un rendimiento mucho mejor. No obstante, dados los altos requisitos computacionales que supone dicho entrenamiento (con tiempos de entrenamiento para 20 episodios de hasta 3 horas en entornos discretos y 9 horas en los continuos), centraremos nuestra atención en ilustrar los resultados obtenidos para esta prueba limitada a 20 episodios.

4.3.3. Resultados

El entrenamiento de los diferentes agentes, tanto en entornos discretos como continuos, dio lugar a una gran cantidad de datos y gráficas correspondientes a las variables monitorizadas por TensorBoard. Como escapa del ámbito de este trabajo realizar un análisis en profundidad de cada métrica y algoritmo, nos centraremos en estudiar el rendimiento de los agentes desde la perspectiva de las métricas de evaluación previamente presentadas en la sección 4.2.4: **recompensa** (recompensa media), **consumo** (consumo medio y penalización por consumo) y **confort** (violación de confort y penalización por confort).

Así, basándonos en los resultados obtenidos en la validación de los agentes, en la Tabla 4.1 se muestra una perspectiva general de los mejores algoritmos para cada entorno, desde el punto de vista de estas métricas de evaluación.

Como se mencionó anteriormente, el rendimiento de los algoritmos en espacios de acciones discretos fue significativamente peor que en los continuos, acercándose al controlador basado en reglas pero sin llegar a superarlo en la mayoría de los casos. Por otro lado, también se enfatizó en el hecho de que los agentes en entornos continuos requieren de un mayor número de episodios para converger en su entrenamiento, al tratarse de espacios de acciones mucho más complejos. Dicha complejidad se debe a que los *setpoints* de ca-

Tabla 4.1: Mejores agentes para diferentes entornos y métricas

Entorno	Recompensa	Consumo	Confort
Disc. hot	RBC/A2C	DQN	RBC/A2C
Disc. mixed	RBC	DQN	RBC/A2C
Disc. cool	RBC/A2C	PPO	RBC/A2C
Cont. hot	DDPG/SAC	SAC	DDPG
Cont. mixed	DDPG/SAC	DDPG	SAC
Cont. cool	SAC	SAC	SAC

Tabla 4.2: Consumo medio de los agentes entrenados en entornos discretos a lo largo de 20 episodios de validación

	HOT		**COOL**		**MIXED**	
	media	*desv*	*media*	*desv*	*media*	*desv*
RBC	6428	171	4421	507	5911	223
A2C	6403	178	4426	524	5960	249
PPO	6278	139	**4294**	490	5800	229
DQN	**6078**	149	4369	505	**5549**	238

lor y frío se ajustan con respecto a un intervalo de valores mucho mayor. De esta forma, estamos ante algoritmos más flexibles, donde no existe ninguna interdependencia entre ambos *setpoints* que pueda afectar al rendimiento, como sí ocurre en los entornos discretos.

La solución al bajo rendimiento de los entornos discretos pasaría por ampliar el número de acciones disponibles, ofreciendo una posibilidad de mapeo mucho mayor, así como por la definición de espacios de acciones multidiscretos. Dichas propuestas están presentes en la sección 5.3, dedicada a aquellos trabajos futuros que puedan derivarse de este proyecto.

Hechas estas aclaraciones, pasemos a analizar cuantitativamente los resultados obtenidos en la **validación** de los agentes entrenados.

Consumo energético

Desde el punto de vista del consumo energético medio en los entornos discretos (Tabla 4.2), A2C ofreció unos resultados muy similares a los del RBC, sin ofrecer una mejora significativa en ninguno de los tres casos. Por otro lado, PPO logró una mejora del 2.87 % en el entorno frío, mientras que DQN redujo el consumo del RBC en los entornos cálido y templado, con disminuciones de consumo del 5.44 % y 6.12 %, respectivamente. No obstante, debe destacarse que estas reducciones en el consumo repercutieron negativamente en el confort, como veremos a la hora de estudiar las recompensas obtenidas.

En el caso de los entornos continuos (Tabla 4.3), SAC y DDPG lograron mejorar los resultados del RBC con resultados bastante positivos: mientras que SAC logró un consumo un 7.67 % y 5.71 % inferior en los entornos *hot* (Figura 4.9) y *mixed*, respectivamente, DDPG obtuvo el menor consumo en el entorno frío con una reducción del 7.45 %.

Finalmente, nótese cómo, tanto en los entornos discretos como continuos, el consumo medio representado en las Figuras 4.8 y 4.9 es creciente debido a la acumulación de ruido en el clima a lo largo del tiempo, lo que dificulta

Tabla 4.3: Consumo medio de los agentes entrenados en entornos continuos a lo largo de 20 episodios de validación

	HOT		**COOL**		**MIXED**	
	media	*desv*	*media*	*desv*	*media*	*desv*
RBC	6674	164	4612	486	6220	249
A2C	6664	160	4630	522	6223	267
PPO	6326	122	4462	486	6146	260
DDPG	6371	132	**4268**	465	6119	234
SAC	**6162**	122	4359	514	**5865**	239

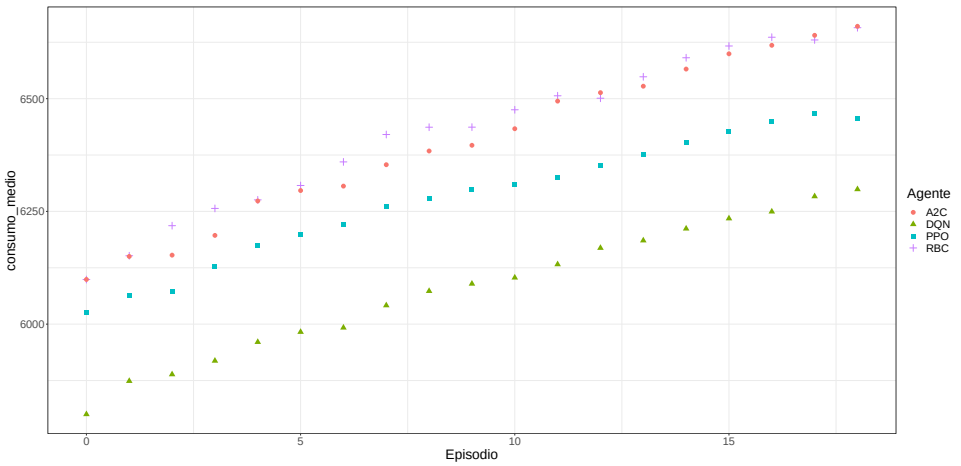

Figura 4.8: Ejemplo del consumo medio obtenido en la validación de los agentes entrenados en el entorno *discrete-stochastic-hot*

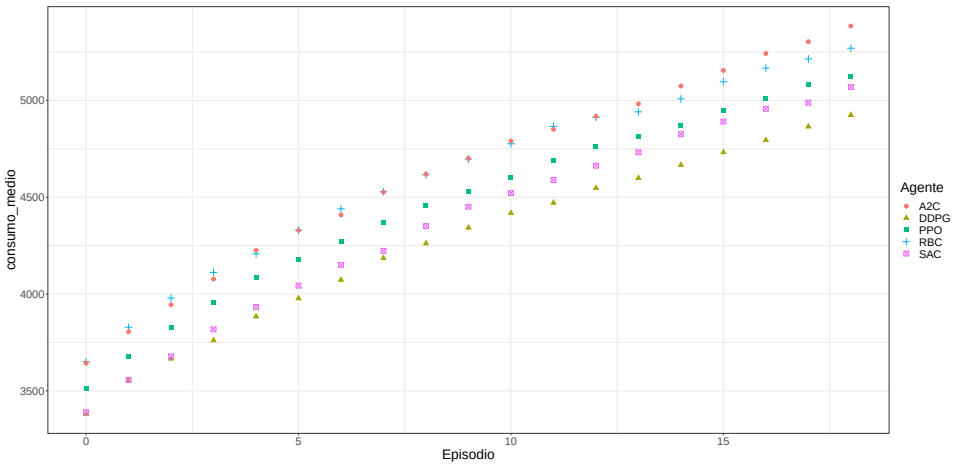

Figura 4.9: Ejemplo del consumo medio obtenido en la validación de los agentes entrenados en el entorno *continuous-stochastic-hot*

Tabla 4.4: Violación de confort media de los agentes entrenados en entornos discretos a lo largo de 20 episodios de validación

	HOT		**COOL**		**MIXED**	
	media	*desv*	*media*	*desv*	*media*	*desv*
RBC	**32.5**	.942	30.4	1.5	**29.9**	.296
A2C	32.6	.882	**30.2**	1.46	30	.225
PPO	33.9	.778	35.4	1.48	32.1	.313
DQN	36.9	1.1	31.5	1.26	33.6	.479

Tabla 4.5: Violación de confort media de los agentes entrenados en entornos continuos a lo largo de 20 episodios de validación

	HOT		**COOL**		**MIXED**	
	media	*desv*	*media*	*desv*	*media*	*desv*
RBC	35.9	.705	35.1	.7	33.5	.064
A2C	36.1	.675	35.1	.701	33.6	.038
PPO	37.5	.557	34.9	.933	33.6	.549
DDPG	**33.3**	.944	32.9	1.55	36	1.06
SAC	35.7	.843	**31**	1.29	**32.6**	.336

la optimización del consumo energético.

Confort

Tomando como referencia el porcentaje de tiempo bajo violación de confort en los entornos discretos, RBC (32.5 %) y A2C (32.6 %) ofrecieron un desempeño similar para el clima cálido. En el clima templado, los porcentajes fueron del 29.9 % y 30 %, respectivamente, mientras que en el frío, RBC logró un 30.4 %, A2C un 30.2 % y DQN un 31.5 %. Así, atendiendo a los resultados de la Tabla 4.4, ningún algoritmo de DRL logró mejorar significativamente los resultados del RBC.

Desde la perspectiva de los entornos continuos, se lograron resultados más positivos (ver Tabla 4.5). Frente al bajo rendimiento de A2C y PPO, DDPG logró reducir en un 2.6 % la violación de confort con respecto al RBC en clima cálido, mientras que SAC consiguió reducciones del 4.1 % y 0.9 % en los entornos frío y templado (Figura 4.11), respectivamente. Aunque no se trata de grandes reducciones en la violación de confort, en la siguiente sección veremos cómo, en combinación con la reducción del consumo, estas se tradujeron en un considerable aumento de las recompensas obtenidas.

Por último, al igual que en el caso del consumo energético, en las Figuras 4.10 y 4.11 se aprecia cómo los porcentajes de violación de confort empeoran

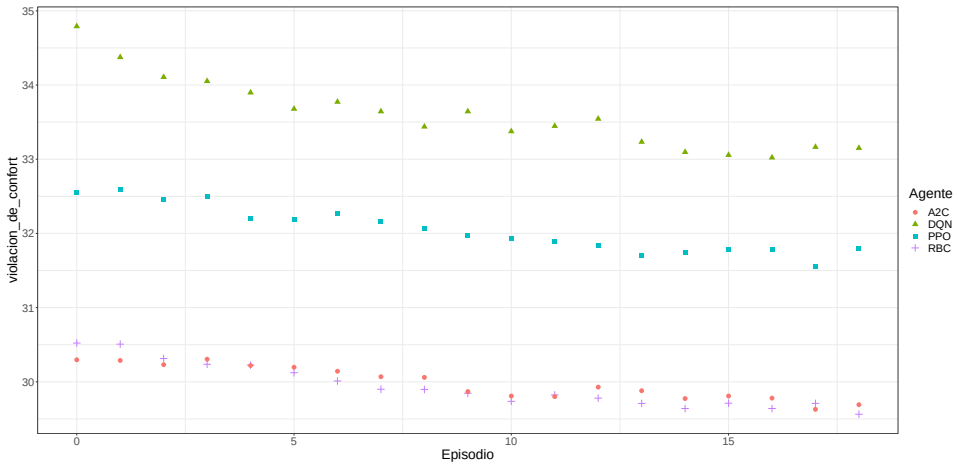

Figura 4.10: Ejemplo de la violación de confort obtenida en la validación de los agentes entrenados en el entorno *discrete-stochastic-mixed*

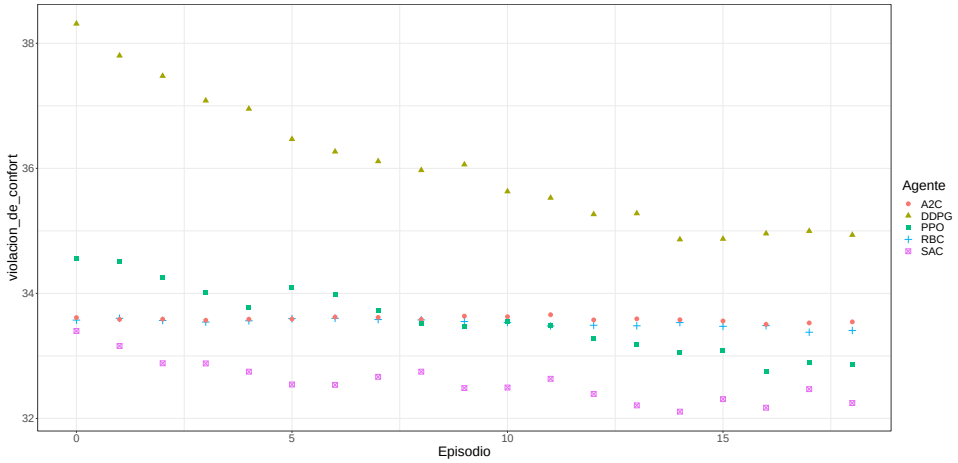

Figura 4.11: Ejemplo de la violación de confort obtenida en la validación de los agentes entrenados en el entorno *continuous-stochastic-mixed*

Tabla 4.6: Recompensas medias obtenidas por los agentes entrenados en entornos discretos a lo largo de 20 episodios de validación

	HOT		**COOL**		**MIXED**	
	media	*desv*	*media*	*desv*	*media*	*desv*
RBC	**-.485**	.004	-.355	.038	**-.422**	.01
A2C	-.486	.005	**-.354**	.04	-.424	.011
PPO	-.502	.004	-.394	.04	-.436	.001
DQN	-.493	.004	-.362	.038	-.44	.008

Tabla 4.7: Recompensas medias obtenidas por los agentes entrenados en entornos continuos a lo largo de 20 episodios de validación

	HOT		**COOL**		**MIXED**	
	media	*desv*	*media*	*desv*	*media*	*desv*
RBC	-.481	.003	-.338	.036	-.403	.013
A2C	-.479	.003	-.341	.04	-.403	.013
PPO	-.496	.002	-.351	.039	-.403	.011
DDPG	**-.455**	.002	-.323	.041	-.461	.004
SAC	**-.455**	.002	**-.322**	.042	**-.384**	.011

a medida que el clima del entorno cuenta con un mayor ruido acumulado.

Recompensa

Finalmente, abordemos el equilibrio entre la minimización del consumo y la maximización del confort, el cual queda reflejado en la función de recompensa empleada.

En los entornos discretos, A2C y RBC lograron resultados similares, sin apenas variación en sus recompensas medias y superando a las de PPO y DQN en todos los climas (ver Tabla 4.6).

Por otro lado, los resultados más llamativos se dieron en los entornos continuos. En el clima cálido (Figura 4.13), las mejoras que SAC y DDPG supusieron frente al RBC fueron del 5.41 %, mientras que SAC logró superar al RBC en los climas frío y templado con mejoras del 4.73 % y 4.71 %, respectivamente. Estos resultados quedan resumidos en la Tabla 4.7.

En base a estos resultados, se observa que los agentes de DRL en entornos discretos no lograron superar el rendimiento de un RBC. Por el contrario, DDPG y SAC demostraron garantizar un mayor confort y un menor consumo energético que un controlador basado en reglas convencional.

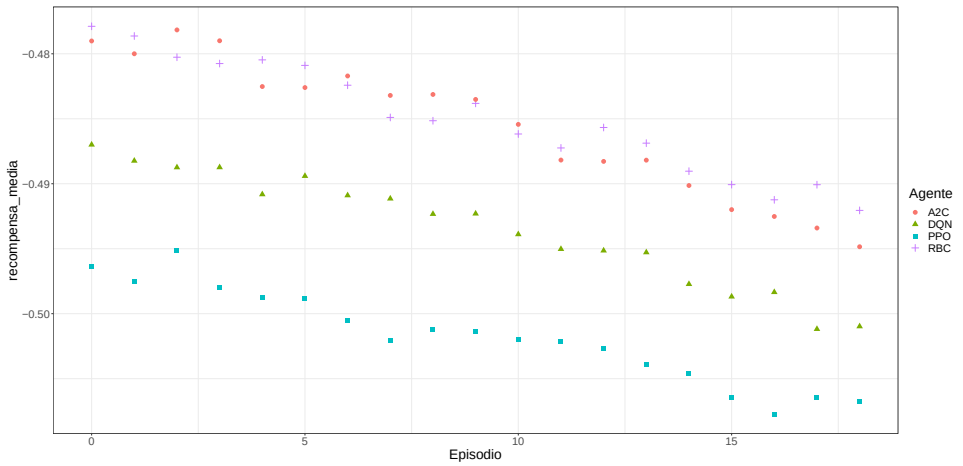

Figura 4.12: Ejemplo de la recompensa media obtenida en la validación de los agentes entrenados en el entorno *discrete-stochastic-hot*

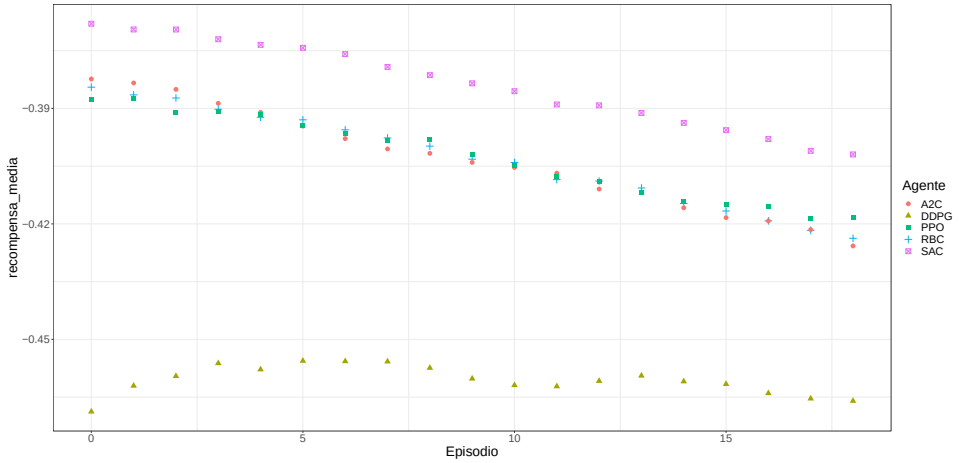

Figura 4.13: Ejemplo de la recompensa media obtenida en la validación de los agentes entrenados en el entorno *continuous-stochastic-hot*

4.4. Experimentación avanzada

Una vez estudiado el desempeño de los agentes en cada tipo de entorno, se llevaron a cabo una serie de experimentos adicionales destinados a comprender en profundidad otros aspectos del problema en cuestión. Dichos experimentos fueron realizados empleando el agente basado en DDPG, al tratarse de uno de los algoritmos que mejores resultados proporcionó en la mayoría de entornos y métricas de evaluación. El hecho de emplear DDPG frente a SAC se debió a que DDPG requirió menores tiempos de entrenamiento.

En las siguientes subsecciones se detallarán dichos experimentos y los resultados obtenidos.

4.4.1. Equilibrio confort-consumo

Como ya se estudió en la sección 2.2.1, el control HVAC por medio de RL supone el empleo de una función de recompensa que contemple tanto la maximización del confort como la reducción del consumo energético. Así, el objetivo a perseguir será la búsqueda del equilibrio entre ambas partes de la recompensa.

En la formulación del problema también abordamos cómo cada una de estas partes cuenta con una ponderación: w_t y $(1 - w_t)$; estos pesos nos permiten definir la importancia asignada al confort y al consumo, respectivamente, o lo que es lo mismo: su influencia en la función de recompensa.

Si atendemos a los experimentos llevados a cabo en la sección 4.3, la importancia asignada a cada parte de la recompensa fue la misma, con $w_t = 0.5$. De forma ilustrativa, pasemos ahora a asignar una mayor importancia al **consumo** y comparar los resultados obtenidos. Para llevar a cabo esta experimentación, emplearemos el agente basado en DDPG, en un clima templado y un espacio de acciones continuo.

Como se muestra en la Figura 4.14, abogar por un 75 % de peso para el consumo (y, por tanto, un 25 % para el confort) frente a una ponderación equilibrada (50 %-50 %) supone:

- Un menor consumo energético medio, ya que este toma una mayor importancia en la función de recompensa. Esto se observa especialmente bien si atendemos a la penalización por consumo, mucho menor para una ponderación 75 %-25 %.

- Un aumento de casi el 50 % de la violación del confort, debido a que el agente centra sus esfuerzos en reducir el consumo a toda costa. De

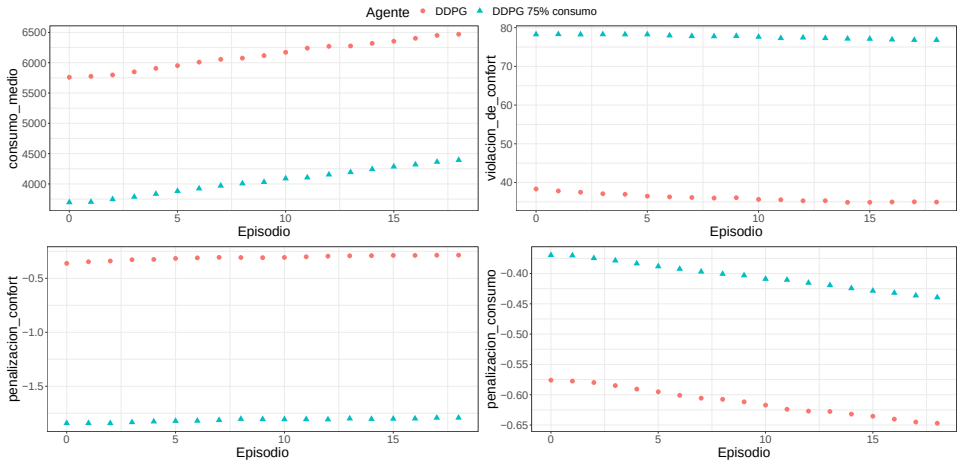

Figura 4.14: Resultados tras la validación de un agente entrenado con DDPG para diferentes ponderaciones de consumo

nuevo, la penalización por confort es considerablemente mayor para el caso 75 %-25 %.

Con este ejemplo se ha tratado de ilustrar la flexibilidad que Energym nos ofrece a la hora de comparar diferentes ponderaciones de confort y consumo. Dichos pesos son un parámetro más a definir por el usuario, permitiendo una mayor personalización de los entornos y dotando a Energym de nuevas capacidades de experimentación.

En conclusión, el equilibrio confort-consumo personalizable en Energym abre un amplio abanico de posibilidades, mayor aún si consideramos ponderaciones dinámicas o funciones de recompensa más complejas, fácilmente definibles a partir de los *wrappers* ofrecidos por OpenAI Gym.

4.4.2. Pruebas de robustez

Pasemos a estudiar la eficiencia de los agentes en entornos para los cuales no han sido entrenados. Estas "pruebas de robustez" nos permiten saber hasta qué punto un agente es capaz de generalizar y obtener un buen rendimiento en condiciones no experimentadas durante su entrenamiento.

Así, el experimento planteado consistirá en la evaluación de un agente entrenado en un clima **templado** en los entornos **cálido** y **frío**. Esto nos permitirá conocer la versatilidad de un agente entrenado en un clima con temperaturas más extremas.

Veamos, pues, los resultados obtenidos para cada prueba de robustez. De nuevo, utilizaremos el agente basado en DDPG en las mismas condiciones

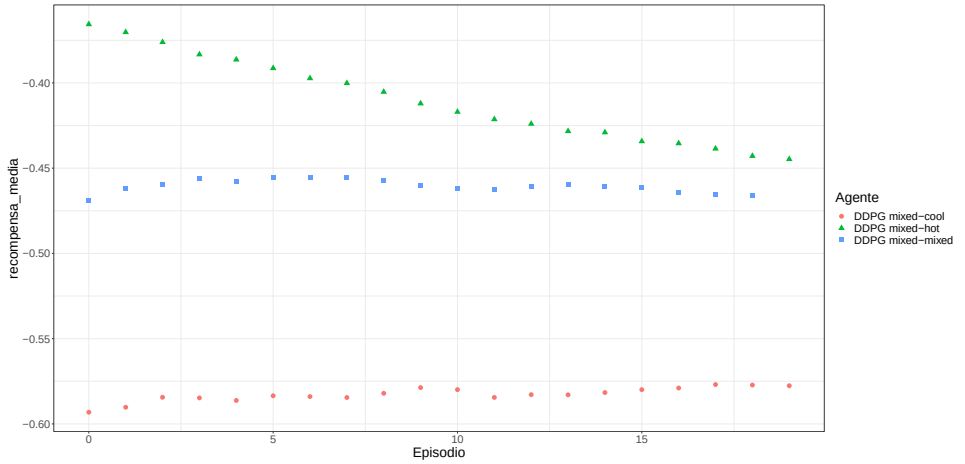

Figura 4.15: Evaluación del agente entrenado en clima *mixed* y ejecutado en *mixed*, *cool* y *hot*

que la experimentación anterior.

Como podemos ver en la Figura 4.15, en un período de 20 episodios un agente entrenado en un clima templado ofrece un mejor control en un clima cálido que en uno mixto. Esto puede deberse a que el clima mixto supone una complejidad superior, al contar con un mayor rango de temperaturas a cubrir. No ocurre lo mismo para el agente entrenado en un entorno mixto y probado en uno frío. En comparación con el caso anterior, el bajo rendimiento podría deberse a que un clima templado es más próximo a las características del cálido que del frío, o bien que el clima frío requiera, por lo general, un mayor consumo energético.

Este tipo de pruebas nos permiten conocer si el entrenamiento de un agente en un determinado tipo de entorno podría ser prescindible. Por ejemplo, en este caso, podríamos emplear el agente *mixed* en el entorno *hot* y obtendríamos mejores resultados, lo que reduce nuestro interés por entrenar un agente para cada entorno, pudiendo disponer de un único agente que opere bien en múltiples contextos.

A modo de reflexión, una discusión más profunda sobre los resultados ofrecidos por este tipo de experimentos sería un interesante debate a abordar de la mano de una persona experta en este campo. Aunque dicha discusión escapa de los objetivos de este proyecto, en esta sección hemos visto cómo Energym podría ser utilizado para un estudio en profundidad ante este tipo de situaciones.

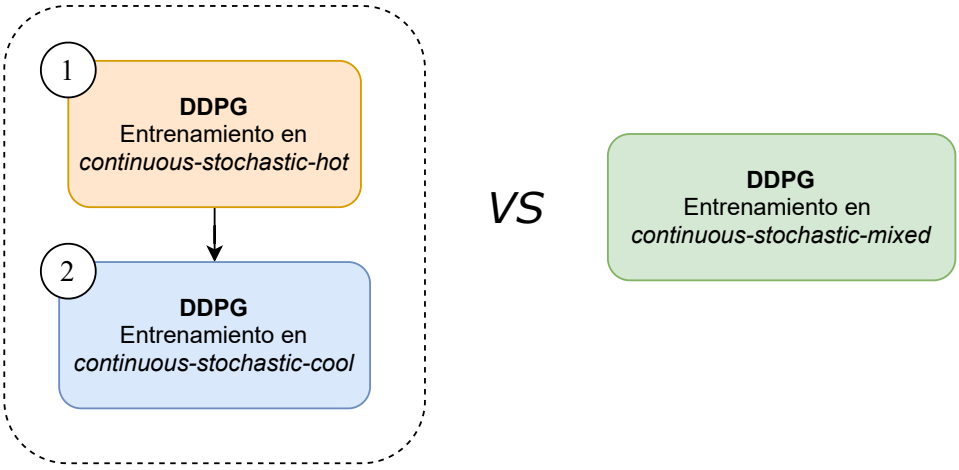

Figura 4.16: Experimentación mediante *curriculum learning*

4.4.3. *Curriculum learning*

En esta última sección abordaremos la aplicación de *curriculum learning* en nuestro problema. Se trata de un tipo de aprendizaje progresivo, consistente en partir del entrenamiento en escenarios sencillos e ir utilizando el conocimiento adquirido hasta el momento para entrenarse en entornos más complejos.

Este método de aprendizaje ha demostrado ser de gran utilidad, logrando importantes avances en el ámbito del aprendizaje por refuerzo [37]. Se trata de un tipo de aprendizaje inspirado profundamente en cómo los humanos aprendemos, ya que nuestra formación se basa en un currículo de situaciones de aprendizaje interdependientes de diversa y progresiva complejidad [36].

En el ámbito de nuestro problema, el aspecto fundamental que marca la complejidad de un entorno es la variabilidad de su clima: es mucho más fácil optimizar el control HVAC en un entorno donde las temperaturas apenas varían a lo largo del año, que en otro donde el clima es más inestable. Así, probaremos a comparar el rendimiento de un agente entrenado directamente sobre un clima *mixed* con otro entrenado en los climas *hot* y *cool*. Utilizaremos DDPG bajo las mismas configuraciones descritas en los experimentos anteriores, siguiendo el proceso de aprendizaje detallado en la Figura 4.16.

Una vez realizado el experimento, si atendemos a la Figura 4.17 y la Tabla 4.8, observamos que el agente entrenado en un entorno cálido y, posteriormente, en un entorno frío, es capaz de ofrecer mejores resultados que un agente entrenado en un entorno templado. Una posible explicación podría ser que el agente entrenado mediante *curriculum learning* cuenta con una mayor especialización en temperaturas altas y bajas, en contraposición con

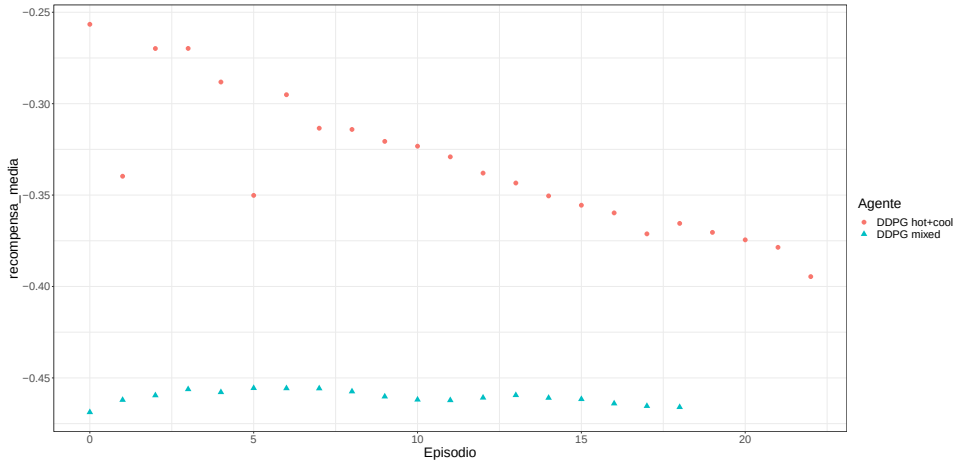

Figura 4.17: Recompensas medias obtenidas en la experimentación con *curriculum learning*

Tabla 4.8: Resultados obtenidos en la validación de DDPG entrenado en *mixed* y DDPG entrenado en *hot* y *cool* sobre el entorno *continuous-mixed*

	Recompensa		**Consumo**		**Viol. confort**	
	media	*desv*	*media*	*desv*	*media*	*desv*
DDPG mixed	-.461	.004	6118.94	234.366	36.044	1.063
DDPG hot+cool	**-.334**	.038	**4396.529**	542.063	**34.683**	2.81

el agente entrenado en *mixed*, el cual no es capaz de gestionar temperaturas moderadamente fuera de su rango habitual.

Finalmente, se reitera en la idea de que la profundización en estos resultados podrían dar lugar a un amplio debate que va más allá de los objetivos de este trabajo. Así, el principal objetivo de esta sección ha sido demostrar cómo Energym puede ajustarse a un tipo de experimentaciones de mayor complejidad que se alejan de las pruebas convencionales, y que, sin duda alguna, cuentan con un enorme potencial en este campo.

Capítulo 5

Conclusiones

En este último capítulo quedan resumidas las principales aportaciones de este trabajo, haciendo especial énfasis en la consecución de los objetivos propuestos y los resultados obtenidos. Finalmente, se detallarán posibles trabajos futuros, así como las principales lecciones aprendidas tras realizar este proyecto.

5.1. Conclusiones

Este trabajo ha tratado de aportar soluciones a un problema aún abierto como es el de la aplicación de RL/DRL en el control óptimo de sistemas HVAC. Se trata de un campo repleto de posibilidades, y que sin duda seguirá creciendo en popularidad en los próximos años.

El desarrollo de Energym es un primer paso hacia la estandarización de este campo, un pilar fundamental de cara a poder estudiar y adoptar las mejores soluciones en control HVAC, garantizando así la reducción del consumo energético y el bienestar de las personas. Se trata de una idea pionera, funcional y que responde a las necesidades expresadas por la comunidad científica.

Por otro lado, la experimentación llevada a cabo en este proyecto pretende, no sólo demostrar la eficacia de Energym como entorno de ejecución de simulaciones energéticas, sino también evidenciar la profundidad y complejidad que supone este problema. Así, en el marco de este proyecto, hemos visto una pequeña parte de la inmensa cantidad de configuraciones, variables y técnicas que pueden afectar al rendimiento de los algoritmos de RL/DRL en control HVAC, abriendo la puerta a futuros trabajos que aprovechen el potencial de Energym en diferentes ámbitos de aplicación.

A su vez, tanto la herramienta desarrollada como el conocimiento adqui-

rido a lo largo de este proyecto permiten plantear cuestiones sobre aspectos hasta el momento asumidos, como la efectividad de las funciones de recompensa expuestas en la literatura, o la capacidad de generalización de modelos entrenados en condiciones muy concretas. Así, Energym pretende arrojar luz sobre algunas de estas cuestiones y servir como un marco de pruebas estándar sobre el que poder corroborar diferentes hipótesis.

Un aspecto a considerar es cómo este tipo de soluciones de control podrían ser aplicadas de forma exitosa en entornos reales, dada su complejidad en comparación con los entornos simulados, donde las variables que influyen en el control se encuentran perfectamente identificadas. Atendiendo a la literatura, existen limitadas aplicaciones reales basadas en métodos de RL [10, 53] y menos aún que hagan uso de algoritmos de DRL [53]. Se trata, pues, de un problema abierto y complejo, para el cual no se cuenta con suficientes aportaciones como para poder ofrecer una respuesta precisa. Así, un objetivo a largo será que el número de soluciones probadas en entornos reales aumente a medida que este campo alcance su madurez.

Otro factor de relevancia son los tiempos de entrenamiento requeridos por este tipo de agentes, crecientes a medida que la complejidad y los escenarios contemplados aumentan. Esto pone en riesgo la escalabilidad de estos sistemas, siendo una posible solución un primer entrenamiento estático que permita ubicar al modelo en el entorno y un posterior entrenamiento *online* que responda a cambios menores [3].

Finalmente, la búsqueda de dicha escalabilidad ha empujado al desarrollo de este campo desde la perspectiva de los sistemas multiagente [35, 59, 21], ofreciendo un control HVAC descentralizado y flexible entre diferentes agentes independientes. Se trata de una idea prometedora y bastante reciente, de la cual cabe esperar importantes avances en los próximos años.

5.2. Cobertura de objetivos

A continuación, abordaremos la consecución de los diferentes objetivos planteados al inicio de este proyecto:

Subobjetivo 1

Comprensión de los fundamentos del aprendizaje por refuerzo y su aplicación en control HVAC.

En el Capítulo 2 se han presentado los fundamentos del aprendizaje por refuerzo y aprendizaje profundo por refuerzo, así como gran parte de los algoritmos que conforman el estado del arte. Posteriormente, se ha abordado su aplicabilidad en el ámbito del control HVAC, presentando el marco formal

empleado para definir este problema.

Finalmente, se han descrito diferentes aplicaciones de RL/DRL en control HVAC presentes en la literatura, evidenciando las motivaciones que impulsaron el desarrollo de este proyecto.

Subobjetivo 2

Extensión de la librería Python Energym haciendo uso de la herramienta Gym de OpenAI.

En el Capítulo 3 se ha descrito el conjunto de actividades llevadas a cabo en el desarrollo de Energym como parte de un equipo de trabajo profesional. Dichas tareas han supuesto la extensión de las funcionalidades de Energym, adaptándolo a diferentes entornos de simulación y facilitando su integración con librerías de DRL como Stable Baselines3.

Todo este proceso de implementación ha estado completamente ligado a la interfaz de programación para entornos de RL que ofrece OpenAI Gym, aprovechando las ventajas que esta herramienta ofrece para desarrollar Energym de una forma estandarizada y escalable.

Subobjetivo 3

Entrenamiento monitorizado de diferentes algoritmos de DRL disponibles en Stable Baselines3 haciendo uso de las herramientas TensorBoard y MLflow.

En el Capítulo 4 se abordó el uso de múltiples algoritmos de DRL disponibles en Stable Baselines3 en combinación con Energym para ilustrar su aplicación en control HVAC. Esto supuso el entrenamiento y ejecución de los agentes en diferentes entornos y condiciones de simulación (espacios de acciones, climas...) de cara a evaluar y comparar los resultados obtenidos.

Por otro lado, este proceso fue continuamente monitorizado haciendo uso de las herramientas TensorBoard y MLflow. La primera sirvió para monitorizar el aprendizaje, mientras que la segunda sirvió para registrar los experimentos realizados y sus configuraciones. Todo el proceso de integración de Energym con estas herramientas quedó detallado en el Capítulo 3.

Subobjetivo 4

Comparativa de los resultados obtenidos para cada uno de los agentes en diferentes contextos de clima y espacios de acciones (discreto y continuo).

La sección 4.3.3 del Capítulo 4 detalla los resultados obtenidos tras la experimentación con diferentes algoritmos de DRL en combinación con Energym. En los entornos discretos, ningún agente logró superar de forma significativa el rendimiento de un agente basado en reglas, siendo A2C el algoritmo que consiguió obtener los resultados más similares.

Por otro lado, en los entornos continuos sí que se lograron importantes mejoras con respecto al agente basado en reglas, siendo SAC y DDPG los algoritmos que ofrecieron mejores resultados. De esta forma, queda demostrada la posibilidad de superar el rendimiento de un agente basado en reglas haciendo uso de un control HVAC basado en DRL.

Subobjetivo 5

Estudio de la recompensa obtenida por un agente en función de la importancia asignada a confort y consumo energético.

Dentro de los experimentos avanzados llevados a cabo en el Capítulo 4 (sección 4.4.1), se estudió la influencia de las ponderaciones de consumo y confort en la función de recompensa empleada por los agentes de DRL.

El ejemplo ilustrativo mostrado en esta sección, planteó cómo un agente con un mayor énfasis en la reducción del consumo energético es capaz de consumir menos a costa de un mayor número penalizaciones por violación de confort.

Subobjetivo 5

Ejecución de pruebas de robustez que permitan conocer el rendimiento de los agentes en diferentes climas a los empleados como entrenamiento.

En la sección 4.4.2 del Capítulo 4 se evaluaron diferentes situaciones en las que un agente era ejecutado en un entorno diferente al empleado en su entrenamiento. Así, las pruebas realizadas revelaron que un agente entrenado en un entorno templado (*mixed*) es capaz de ofrecer muy buenos resultados en un clima cálido, no ocurriendo lo mismo en un clima frío, donde dicho rendimiento disminuye.

Subobjetivo 5

Aplicación de *curriculum learning* con diferentes agentes y análisis de los resultados obtenidos.

Finalmente, en la sección 4.4.3 del Capítulo 4 se ilustró la aplicación de *curriculum learning* en el ámbito de este problema. El ejemplo planteado

mostró cómo un agente entrenado progresivamente a partir de los entornos *hot* y *cool* es capaz de ofrecer un mejor rendimiento que un agente entrenado únicamente en un entorno *mixed*.

5.3. Trabajo futuro

Son muchas las posibilidades y líneas de trabajo que pueden derivarse de este proyecto. Como hemos podido ver a lo largo de su desarrollo, la cantidad de técnicas, configuraciones, entornos o climas a emplear abren la puerta a un sinfín de experimentos de relevancia en el campo. A continuación, se propone un listado de algunas de las principales metas planteadas como trabajo futuro[1]:

- Ampliación de la compatibilidad de Energym con otros entornos de simulación como OpenModelica o Simulink.
- Experimentación con otros entornos, climas y variables. Actualmente se encuentra en proceso la integración de Energym en el entorno Data Center, el cual incluye como variables de entrada tanto los *setpoints* de calefacción y refrigeración como la ventilación del edificio.
- Implementación de entornos con espacios de acciones multidiscretos, así como ampliación del espacio de acciones empleado por los entornos discretos.
- Evaluación de otros algortimos de DRL implementados tanto en Stable Baselines3 (por ejemplo, TD3) como en otras librerías como RLlib, TensorFlow Agents o Keras-RL.
- Comparar los resultados obtenidos para diferentes funciones de recompensa, así como implementar ponderaciones dinámicas (por ejemplo, variables en el tiempo o dependientes del número de personas en el edificio) para consumo y confort.
- Profundización en el uso de *curriculum learning*, estudiando la capacidad de generalización en diferentes entornos donde no sólo cambia el clima sino también las características del edificio.
- Realización de pruebas de robustez adicionales que permitan cubrir todo el espacio de combinaciones climáticas posibles.
- Extensión de las funcionalidades de Energym para el procesamiento y representación automática de los datos generados.

[1] Muchas de estas posibles mejoras se encuentran descritas en el listado de *issues* del repositorio de Energym: `https://github.com/jajimer/energym/issues`.

- Entrenamiento y ejecución de los agentes en la nube, haciendo uso de servicios como Google Cloud, Microsoft Azure o Amazon Web Services.

Tras la conclusión de este proyecto, un objetivo a corto plazo será dar conocer Energym y sus aplicaciones mediante publicaciones científicas en revistas especializadas en este campo, favoreciendo así su difusión y madurez. Esta labor se enmarca dentro del proyecto PROFICIENT, financiado por el programa *EXPLORA* del Ministerio de Ciencia, Innovación y Universidades (TIN2017-91223-EXP) y orientado al desarrollo de soluciones basadas en DRL para el control energético eficiente de edificios.

5.4. Valoración personal

Realizar un trabajo de fin de máster que involucrase la profundización en el campo del aprendizaje por refuerzo fue una de las principales motivaciones a adentrarme de lleno en este proyecto. A su vez, el hecho de poder aunar mi interés por este campo y su aplicación en la resolución de problemas reales fueron motivos de peso para decantarme por un trabajo de tal envergadura.

Considero al aprendizaje por refuerzo como un paradigma de enorme potencial, en ocasiones eclipsado por el aprendizaje supervisado y no supervisado, pero que en los últimos años ha conseguido una gran repercusión gracias a las aportaciones y popularidad de proyectos como AlphaGo o AlphaZero. Actualmente contamos con aplicaciones de RL en inmensidad de ámbitos de lo más diversos, siendo el control energético un dominio llamativo y de especial relevancia para la sociedad.

Desde una perspectiva personal, el uso de RL en el control de sistemas HVAC abre la puerta a una nueva forma mucho más eficiente de garantizar el bienestar de las personas. Si a un control eficiente de estos sistemas sumásemos fuentes de energía renovables, contaríamos con edificios energéticamente autorregulados y sostenibles, algo prioritario ante la amenaza del cambio climático y el calentamiento global.

Finalmente, considero que las aportaciones de este trabajo podrán ser especialmente relevantes para la comunidad científica involucrada en este ámbito. El desarrollo de Energym y la experimentación propuesta en este trabajo abogan por una visión común y unificada de las propuestas que conforman el estado del arte en control HVAC y RL, facilitando el progreso, la reproducibilidad y la estandarización de las soluciones existentes y aún por desarrollar.

Bibliografía

[1] Charles W Anderson y col. «Synthesis of reinforcement learning, neural networks and PI control applied to a simulated heating coil». En: *Artificial Intelligence in Engineering* 11.4 (1997), págs. 421-429.

[2] ASHRAE ANSI y M ASHRAE. «Standard 55—Thermal Environmental Conditions for Human Occupancy». En: *ASHRAE, Atlanta* (2004).

[3] Donald Azuatalam y col. «Reinforcement learning for whole-building HVAC control and demand response». En: *Energy and AI* 2 (2020), pág. 100020.

[4] Enda Barrett y Stephen Linder. «Autonomous HVAC control, a reinforcement learning approach». En: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2015, págs. 3-19.

[5] Richard Bellman. «Dynamic programming». En: *Science* 153.3731 (1966), págs. 34-37.

[6] Silvio Brandi y col. «Deep Reinforcement Learning to optimise indoor temperature control and heating energy consumption in buildings». En: *Energy and Buildings* 224 (2020), pág. 110225.

[7] Nicolaj Tofte Brenneche. «Secure, Clean and Efficient Energy». En: *Visions for Horizon 2020: From Copenhagen Research Forum*. Danmarks Tekniske Universitet. 2012, págs. 39-48.

[8] Lucian Buşoniu y col. «Reinforcement learning for control: Performance, stability, and deep approximators». En: *Annual Reviews in Control* 46 (2018), págs. 8-28.

[9] Yujiao Chen y col. «Optimal control of HVAC and window systems for natural ventilation through reinforcement learning». En: *Energy and Buildings* 169 (2018), págs. 195-205.

[10] Giuseppe Tommaso Costanzo y col. «Experimental analysis of data-driven control for a building heating system». En: *Sustainable Energy, Grids and Networks* 6 (2016), págs. 81-90.

[11] Konstantinos Dalamagkidis y col. «Reinforcement learning for energy conservation and comfort in buildings». En: *Building and environment* 42.7 (2007), págs. 2686-2698.

[12] Dajun Du y Minrui Fei. «A two-layer networked learning control system using actor–critic neural network». En: *Applied mathematics and computation* 205.1 (2008), págs. 26-36.

[13] Yan Du y col. «Intelligent multi-zone residential HVAC control strategy based on deep reinforcement learning». En: *Applied Energy* 281 (2021), pág. 116117.

[14] Pedro Fazenda y col. «Using reinforcement learning to optimize occupant comfort and energy usage in HVAC systems». En: *Journal of Ambient Intelligence and Smart Environments* 6.6 (2014), págs. 675-690.

[15] Florida Solar Energy Center. *Determining Appropriate Heating and Cooling Thermostat Set Points for Building Energy Simulations for Residential Buildings in North America*. 2013.

[16] Nicolas Pardo Garcia y col. *Heat and cooling demand and market perspective*. Publications Office of the European Union, 2012.

[17] Tuomas Haarnoja y col. «Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor». En: *International Conference on Machine Learning*. PMLR. 2018, págs. 1861-1870.

[18] Gregor P Henze y Jobst Schoenmann. «Evaluation of reinforcement learning control for thermal energy storage systems». En: *HVAC&R Research* 9.3 (2003), págs. 259-275.

[19] Chloe Ching-Yun Hsu, Celestine Mendler-Dünner y Moritz Hardt. «Revisiting Design Choices in Proximal Policy Optimization». En: *arXiv preprint arXiv:2009.10897* (2020).

[20] Russell Jurney. *Agile data science 2.0: Building full-stack data analytics applications with spark*. O'Reilly Media Inc, 2017.

[21] Bocheng Li y Li Xia. «A multi-grid reinforcement learning method for energy conservation and comfort of HVAC in buildings». En: *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2015, págs. 444-449.

[22] Timothy P Lillicrap y col. «Continuous control with deep reinforcement learning». En: *arXiv preprint arXiv:1509.02971* (2015).

[23] Simeng Liu y Gregor P Henze. «Evaluation of reinforcement learning for optimal control of building active and passive thermal storage inventory». En: *Journal of Solar Energy Engineering* (2007).

[24] Simeng Liu y Gregor P Henze. «Experimental analysis of simulated reinforcement learning control for active and passive building thermal storage inventory: Part 2: Results and analysis». En: *Energy and buildings* 38.2 (2006), págs. 148-161.

[25] Karel Mařík y col. «Advanced HVAC control: Theory vs. reality». En: *IFAC Proceedings Volumes* 44.1 (2011), págs. 3108-3113.

[26] Karl Mason y Santiago Grijalva. «A review of reinforcement learning for autonomous building energy management». En: *Computers & Electrical Engineering* 78 (2019), págs. 300-312.

[27] Evan McKee y col. «Deep Reinforcement Learning for Residential HVAC Control with Consideration of Human Occupancy». En: *2020 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE. 2020, págs. 1-5.

[28] Volodymyr Mnih y col. «Asynchronous methods for deep reinforcement learning». En: *International conference on machine learning*. PMLR. 2016, págs. 1928-1937.

[29] Volodymyr Mnih y col. «Human-level control through deep reinforcement learning». En: *nature* 518.7540 (2015), págs. 529-533.

[30] Volodymyr Mnih y col. «Playing atari with deep reinforcement learning». En: *arXiv preprint arXiv:1312.5602* (2013).

[31] Miguel Molina-Solana y col. «Data science for building energy management: A review». En: *Renewable and Sustainable Energy Reviews* 70 (2017), págs. 598-609.

[32] M. Morales. *Grokking Deep Reinforcement Learning*. Manning Publications, 2020. ISBN: 9781617295454.

[33] Takao Moriyama y col. «Reinforcement learning testbed for power-consumption optimization». En: *Asian Simulation Conference*. Springer. 2018, págs. 45-59.

[34] Michael C Mozer. «The neural network house: An environment hat adapts to its inhabitants». En: *Proc. AAAI Spring Symp. Intelligent Environments*. Vol. 58. 1998.

[35] Srinarayana Nagarathinam y col. «MARCO-Multi-Agent Reinforcement learning based COntrol of building HVAC systems». En: *Proceedings of the Eleventh ACM International Conference on Future Energy Systems*. 2020, págs. 57-67.

[36] Sanmit Narvekar y col. «Curriculum learning for reinforcement learning domains: A framework and survey». En: *Journal of Machine Learning Research* 21.181 (2020), págs. 1-50.

[37] Rémy Portelas y col. «Automatic curriculum learning for deep rl: A short survey». En: *arXiv preprint arXiv:2003.04664* (2020).

[38] Frederik Ruelens y col. «Learning agent for a heat-pump thermostat with a set-back strategy using model-free reinforcement learning». En: *Energies* 8.8 (2015), págs. 8300-8318.

[39] Tom Schaul y col. «Prioritized experience replay». En: *arXiv preprint arXiv:1511.05952* (2015).

[40] John Schulman y col. «Proximal policy optimization algorithms». En: *arXiv preprint arXiv:1707.06347* (2017).

[41] John Schulman y col. «Trust region policy optimization». En: *International conference on machine learning*. PMLR. 2015, págs. 1889-1897.

[42] David Silver y col. «Deterministic policy gradient algorithms». En: *International conference on machine learning*. PMLR. 2014, págs. 387-395.

[43] Biao Sun y col. «Event-based optimization within the Lagrangian relaxation framework for energy savings in HVAC systems». En: *IEEE Transactions on Automation Science and Engineering* 12.4 (2015), págs. 1396-1406.

[44] Richard S Sutton y Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[45] Richard S Sutton y col. «Policy gradient methods for reinforcement learning with function approximation.» En: *NIPs*. Vol. 99. Citeseer. 1999, págs. 1057-1063.

[46] Daniel Urieli y Peter Stone. «A learning agent for heat-pump thermostat control». En: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. 2013, págs. 1093-1100.

[47] William Valladares y col. «Energy optimization associated with thermal comfort and indoor air control via a deep reinforcement learning algorithm». En: *Building and Environment* 155 (2019), págs. 105-117.

[48] Hado Van Hasselt, Arthur Guez y David Silver. «Deep reinforcement learning with double q-learning». En: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[49] José Vázquez-Canteli, Jérôme Kämpf y Zoltán Nagy. «Balancing comfort and energy consumption of a heat pump using batch reinforcement learning with fitted Q-iteration». En: *Energy Procedia* 122 (2017), págs. 415-420.

[50] José R Vázquez-Canteli y Zoltán Nagy. «Reinforcement learning for demand response: A review of algorithms and modeling techniques». En: *Applied energy* 235 (2019), págs. 1072-1089.

[51] José R Vázquez-Canteli y col. «Fusing TensorFlow with building energy simulation for intelligent energy management in smart cities». En: *Sustainable cities and society* 45 (2019), págs. 243-257.

[52] Yuan Wang, Kirubakaran Velswamy y Biao Huang. «A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems». En: *Processes* 5.3 (2017), pág. 46.

[53] Zhe Wang y Tianzhen Hong. «Reinforcement learning for building controls: The opportunities and challenges». En: *Applied Energy* 269 (2020), pág. 115036.

[54] Ziyu Wang y col. «Dueling network architectures for deep reinforcement learning». En: *International conference on machine learning*. PMLR. 2016, págs. 1995-2003.

[55] Christopher JCH Watkins y Peter Dayan. «Q-learning». En: *Machine learning* 8.3-4 (1992), págs. 279-292.

[56] Tianshu Wei, Yanzhi Wang y Qi Zhu. «Deep reinforcement learning for building HVAC control». En: *Proceedings of the 54th annual design automation conference 2017*. 2017, págs. 1-6.

[57] Michael Wetter y col. «IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation–ongoing developments, lessons learned and challenges». En: *IOP Conference Series: Earth and Environmental Science*. Vol. 323. 1. IOP Publishing. 2019, pág. 012114.

[58] Lei Yang y col. «Reinforcement learning for optimal control of low exergy buildings». En: *Applied Energy* 156 (2015), págs. 577-586.

[59] Liang Yu y col. «Multi-agent deep reinforcement learning for HVAC control in commercial buildings». En: *IEEE Transactions on Smart Grid* 12.1 (2020), págs. 407-419.

[60] Zhen Yu y Arthur Dexter. «Online tuning of a supervisory fuzzy controller for low-energy building system using reinforcement learning». En: *Control Engineering Practice* 18.5 (2010), págs. 532-539.

[61] Xiaolei Yuan y col. «Study on the application of reinforcement learning in the operation optimization of HVAC system». En: *Building Simulation*. Springer. 2020, págs. 1-13.

[62] Alexander Zai y Brandon Brown. *Deep reinforcement learning in action*. Manning Publications, 2020.

[63] Hongming Zhang y Tianyang Yu. «Taxonomy of Reinforcement Learning Algorithms». En: *Deep Reinforcement Learning*. Springer, 2020, págs. 125-133.

[64] Zhiang Zhang y Khee Poh Lam. «Practical implementation and evaluation of deep reinforcement learning control for a radiant heating system». En: *Proceedings of the 5th Conference on Systems for Built Environments*. 2018, págs. 148-157.

[65] Zhiang Zhang y col. «A deep reinforcement learning approach to using whole building energy model for hvac optimal control». En: *2018 Building Performance Analysis Conference and SimBuild*. Vol. 3. 2018, págs. 22-23.

[66] Zhiang Zhang y col. «Whole building energy model for HVAC optimal control: A practical framework based on deep reinforcement learning». En: *Energy and Buildings* 199 (2019), págs. 472-490.