TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Estudio y aplicaciones del aprendizaje por refuerzo para control energético

Aplicación y experimentación con técnicas de Deep Reinforcement Learning para control energético en edificios

Autor
Ahmed Brek Prieto

Directores
Miguel Molina Solana
Alejandro Campoy Nieves

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, julio de 2022

Estudio y aplicaciones del aprendizaje por refuerzo para control energético

Ahmed Brek Prieto

Palabras clave: aprendizaje por refuerzo, control energético de edificios, sistemas de climatización, aprendizaje automático.

Resumen

Dada la gravedad de la situación, la preocupación por el cambio climático no ha dejado de crecer durante las últimas décadas. Se trata de un problema que concierne a toda la humanidad, por lo que la búsqueda de medidas para contrarrestarlo se ha vuelto incesante. El consumo energético en edificios toma un papel importante en este aspecto, siendo responsable del 35 % del consumo energético mundial y del 40 % de la emisión de gases de efecto invernadero. Por otro lado, los sistemas de climatización suponen uno de los mayores gastos en estas infraestructuras, por lo que su optimización resulta crucial a la hora de combatir el calentamiento global.

Hasta hace unos años, el control de estos sistemas ha sido mayoritariamente tratado con técnicas reactivas, aunque estas ofrecen un desempeño ciertamente limitado. El reciente éxito del aprendizaje por refuerzo profundo en algunos juegos y problemas de control ha llevado a algunos investigadores a aplicarlo en el control energético de edificios, demostrando ser un paradigma bastante prometedor.

El propósito de este proyecto de fin de grado consiste en el estudio y aplicación de técnicas de aprendizaje por refuerzo profundo para optimizar el control energético en edificios, manteniendo además el confort térmico en su interior. Más concretamente, se diseñan y estudian distintas funciones de recompensa que se adaptan de forma dinámica a la ocupación del edificio. También se analiza el desempeño de diferentes espacios de acciones y tipos de algoritmos. Para ello, es necesario el desarrollo de un entorno de simulación que permita realizar la experimentación propuesta. Asimismo, con esta investigación se trata de contribuir en medida de lo posible a la comunidad científica, aportando nuevas ideas que puedan ser combinadas con el estado del arte para mejorar los resultados.

Study and applications of reinforcement learning for energy control

Ahmed Brek Prieto

Keywords: reinforcement learning, building energy control, HVAC, machine learning.

Abstract

Given the seriousness of the situation, concerns about climate change are steadily increasing during recent decades. It is a problem that affects all mankind, so the search for solutions has become ever-increasing. Energy consumption in buildings plays an important role in this area, being responsible for 35 % of world consumption and 40 % of greenhouse gas emissions. On the other hand, HVAC systems represent one of the largest energy costs in these infrastructures. Therefore, the optimization of its operation is decisive to mitigate global warming.

Until a few years ago, the control task of these systems has been mostly treated with reactive techniques, although showing limited performance. The recent success of deep reinforcement learning in some games and control problems has led some researchers to apply those methods to building energy control, proving to be a promising paradigm.

The main purpose of this project is to study and apply deep reinforcement learning techniques to optimize building energy control, while maintaining thermal comfort inside. More specifically, different reward functions that change dynamically depending on the building occupancy are designed and studied. Also, performance of different action spaces and algorithms is analyzed. Therefore, it will be necessary to develop a simulation environment where experiments can be run. In addition, this research tries to contribute as much as possible to the scientific community, providing new ideas that could be combined with the state of the art in order to improve results.

Yo, **Ahmed Brek Prieto**, alumno de la titulación **Grado en Ingeniería Informática** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77037522B, autorizo la ubicación de la siguiente copia de mi Trabajo de Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Ahmed Brek Prieto

Granada a 4 de julio de 2022.

D. **Miguel José Molina Solana** y D. **Alejandro Campoy Nieves**, investigadores del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada,

Informan:

Que el presente trabajo, titulado ***Estudio y aplicaciones del Aprendizaje por Refuerzo para control energético***, ha sido realizado bajo su supervisión por **Ahmed Brek Prieto**, 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 4 de julio de 2022.

Los directores:

Miguel José Molina Solana Alejandro Campoy Nieves

Agradecimientos

Me gustaría dar las gracias a todas las personas que me han ayudado y apoyado durante esta última etapa de mis estudios de grado, protagonizada por el desarrollo de este proyecto.

Agradecer a mis tutores, Miguel y Alejandro, el enorme interés y atención que han mostrado a lo largo de estos meses, ofreciendo su ayuda siempre que lo he necesitado. Aprecio mucho la oportunidad que me han brindado para descubrir una campo de estudio tan cautivador como es el aprendizaje por refuerzo, en el que pienso seguir formándome de aquí en adelante.

También me gustaría destacar la cálida acogida y la gran familiaridad que han mostrado todos los miembros del equipo que hay detrás de Sinergym desde el primer día. Especialmente Antonio, con quien he tenido el placer de tratar más, y ha estado dispuesto en todo momento a echarme una mano e ilustrarme con su conocimiento en esta materia.

Por último, aunque no por ello menos importante, dar las gracias a mis familiares y amigos, en particular a mi hermano, Yacine, mis padres y mi abuela, por apoyarme de forma incondicional y hacer más fáciles y llevaderos estos meses de trabajo.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En este capítulo introductorio, se expone la motivación del proyecto, se describe brevemente el problema a resolver y se listan los objetivos. Más adelante, se indican tanto la estructura como la planificación económica y temporal de su realización.

1.1. Motivación

El término «cambio climático» da nombre a una de las mayores preocupaciones de la humanidad actualmente. La inquietud por este problema no ha dejado de crecer en las últimas décadas, dado el reciente y progresivo agravamiento de la situación. La cada vez mayor demanda energética junto a la emisión de gases de efecto invernadero generan urgencia en la búsqueda de soluciones. Al analizar las causas del problema, se encuentra que el consumo energético en edificios constituye un 35 % del total, protagonizando el uso del 55 % de la energía eléctrica generada en todo el planeta [1]. Además, los edificios son responsables del 38 % de las emisiones de CO_2. Por otro lado, se ha observado que la mayor parte de la energía empleada en ellos se destina a la climatización [2]. Se prevé que la demanda energética en edificios aumente un 50 % en los próximos 30 años [3, 4].

Estas cifras sugieren que la optimización en el control energético en edificios es una vía para atajar el problema. Otras alternativas pasan por las mejoras constructivas, o la inversión en energías renovables, aspectos ambos que quedan fuera del ámbito de este proyecto, que se centrará en la optimización de la operación. Si se logra cierto ahorro en esta tarea y se aplica a gran escala, puede traducirse en una considerable reducción en la demanda energética a nivel mundial, lo que genera un gran interés en estudios de este tipo.

Asimismo, el problema del control eficiente de energía en edificios, ya sean de tipo comercial como residencial, adquiere cada vez más importancia.

En sus inicios, ha sido tratado con técnicas más clásicas, como controladores basados en reglas (*rule-based controllers*) o control predictivo por modelo (*model predictive control*). Aunque su rendimiento no es necesariamente bajo, presenta la dificultad de su programación y rediseño en caso de que se desee mejorar este desempeño.

Los recientes avances en el desarrollo *hardware*, especialmente en componentes como las GPUs, han permitido explotar parte del potencial de técnicas de aprendizaje profundo (*deep learning*). A su vez, esto ha dado una nueva vida a la disciplina del aprendizaje por refuerzo (*reinforcement learning*), que quedaba parcialmente eclipsada por los aprendizajes supervisado y no supervisado. Surge así el aprendizaje por refuerzo profundo (*deep reinforcement learning*), que plantea técnicas muy interesantes para abordar el problema en cuestión.

Sin embargo, no se trata de una tarea trivial debido a diversos retos [5]. Por una parte, suele ser difícil desarrollar un modelo explícito de la dinámica térmica de un edificio que sea suficientemente preciso y eficiente para el control energético de edificios. Por otro lado, en ocasiones existe cierta incertidumbre sobre algunos parámetros como la temperatura exterior, la ocupación, el flujo de energía renovable generada, etc.). Adicionalmente, aparecen varias restricciones temporales y espaciales que dificultan la resolución del problema [6]. Por último, los problemas de eficiencia energética no pueden ser resueltos en tiempo real utilizando métodos tradicionales cuando el espacio de acciones crece demasiado. Estos métodos tienen además poca versatilidad cuando se usan en distintos entornos, dadas las premisas que requieren para su aplicación. Dicho esto, el empleo de técnicas modernas de aprendizaje por refuerzo profundo resulta indispensable a la hora de afrontar el problema planteado.

Por todo ello, este proyecto de fin de grado consiste en la investigación y aplicación de una serie de propuestas con el objetivo de mejorar el ahorro energético en edificios, orientado al control de sistemas de climatización, tratando de mantener el confort térmico en el interior. Las ideas aquí presentadas no son transversales a la dirección en la que avanzan las principales líneas de investigación, sino que permiten ser combinadas en aras de optimizar los resultados obtenidos. Así, se trata de contribuir a la comunidad científica con el estudio realizado y la extracción de conclusiones de valor útiles para futuros trabajos. Otro propósito de este proyecto consiste en contribuir al proyecto de software libre *Sinergym*, a la par que apoyar su uso y contribuir a su desarrollo.

1.2. Objetivos

1.2.1. Descripción del problema

El problema a resolver adopta el nombre de **control energético inteligente en edificios** (*smart building energy management*, abreviado SBEM). Consiste en la reducción del consumo energético de un edificio, en este caso, regulando el funcionamiento del sistema de climatización con el que se trata de mantener el confort térmico de los ocupantes de la construcción. Para ello, se utilizarán **técnicas de aprendizaje por refuerzo profundo**, y se propondrán funciones de recompensa que adapten la ponderación del confort térmico o ahorro energético de forma dinámica, dependiendo de variables como la ocupación en cada momento, además de cierta experimentación adicional. Todas estas pruebas serán especificadas con mayor detalle en el capítulo 5. Además, la formulación del problema se expone formalmente en 2.5.

1.2.2. Lista de objetivos

A continuación, se especifica una lista de objetivos concretos que permiten la resolución del problema anteriormente descrito, además de contribuir a la investigación científica en el mayor grado posible:

Objetivo 1 : Introducción y estudio de fundamentos de RL y DRL. Familiarización con el problema de control de sistemas HVAC y su resolución con técnicas de aprendizaje por refuerzo.

Objetivo 2 : Revisión literaria e investigación del estado del arte. Análisis de problemas y proposición de soluciones.

Objetivo 3 : Estudio de Sinergym e integración en el equipo de trabajo. Contribuciones al desarrollo.

Objetivo 4 : Desarrollo del entorno *software* necesario para la ejecución de los experimentos, tanto de forma local como remota.

Objetivo 5 : Realización de la experimentación propuesta. Entrenamiento y evaluación de modelos, utilizando herramientas de monitorización como *Tensorboard* y *MLFlow* para su registro.

Objetivo 6 : Propuesta y realización de estudios adicionales.

Objetivo 7 : Análisis desde múltiples enfoques de los resultados obtenidos.

Objetivo 8 : Extracción de conclusiones y aportaciones útiles a la comunidad científica. Planteamiento de posibles trabajos futuros de interés.

1.3. Planificación

En esta sección se detalla la planificación temporal seguida durante la realización del proyecto, junto con un desglose del presupuesto correspondiente.

1.3.1. Cronograma

Acerca de la planificación temporal, cabe indicar que el desarrollo del proyecto ha tenido una **duración de 124 días**, con una dedicación bastante alta por parte del estudiante. Con una media aproximada de 4 horas al día, se estima un **total de 496 horas**.

Por otro lado, se detalla la distribución y duración de tareas y subtareas en la figura 1.1 a lo largo de los meses de trabajo, clasificadas en **tres fases**, solapadas en algunos tramos. Se distinguen así una fase más teórica, de estudio y formación del estudiante sobre la materia; otra fase dedicada tanto al desarrollo *software* como a la experimentación, y una última fase de análisis de resultados y conclusiones del proyecto.

Figura 1.1: Diagrama de Gantt sobre planificación temporal del proyecto.

1.3.2. Presupuesto

A la hora de realizar el presupuesto, se han consultado algunas referencias para tratar de estimar los precios de la forma más objetiva y exacta posible. De esta manera, el valor de los porcentajes destinados a la cotización de la Seguridad Social ha sido extraído del sitio web oficial de la Seguridad Social Española. El sueldo bruto anual promedio de un ingeniero de *machine learning* considerado en España, en 2022, es una estimación de GlassDoor. A su vez, el valor del IRPF se ha estimado utilizando esta herramienta online, donde se ha aplicado el sueldo mencionado para un trabajador de 21 años de edad.

Dicho esto, se obtiene un **presupuesto total de 23.196'30€**. Se muestra el desglose en la tabla 1.1.

Gasto	**Cantidad**
Gastos de personal	**21.846'30€**
Sueldo bruto anual	42.420€
Sueldo bruto mensual	3.535€
Número de pagas	5
IRPF	20'65 %
Cotización Seguridad Social del trabajador	23'6 %
Cotización Seguridad Social de la empresa	4'7 %
Sueldo neto mensual	2.638'88€
Costes varios adicionales	73'00€
Coste mensual empresa	4.369'26€
Gastos *hardware*	**1.350€**
Ordenador portátil HP Omen 15	1.150€
Google Cloud Platform Services	200€

Cuadro 1.1: Desglose del presupuesto total del proyecto.

1.4. Estructura de la memoria

La memoria de este proyecto se estructura de la siguiente forma. En el capítulo introductorio se describen la motivación, objetivos, problema afrontado y planificación adoptada. El segundo capítulo se dedica a explicar el marco teórico y los conceptos necesarios para entender el trabajo, seguido de un capítulo donde se estudia el estado del arte y se hace una revisión literaria de los artículos más relevantes de la actualidad. Posteriormente

se expone el desarrollo *software* llevado a cabo. Finalmente, se presenta la experimentación realizada y el análisis de los resultados obtenidos, para finalizar el proyecto con un capítulo de conclusiones, donde se extraen y condensan las ideas más relevantes del estudio, además de proponer una serie de trabajos futuros de interés.

Capítulo 2

Fundamentos teóricos

2.1. Inteligencia artificial y aprendizaje automático

A pesar de la creciente popularidad que ha adquirido el ámbito de la **inteligencia artificial** durante las últimas décadas, sigue siendo un término bastante difícil de definir. Y es que no se trata de un campo de estudio para nada reciente. De hecho, sus orígenes se remontan al año **1956**, cuando este término fue acuñado por primera vez [7]. A lo largo del tiempo, han surgido distintas aproximaciones que tratan de definir la IA, aunque no se ha consolidado ninguna de ellas como definitiva. En la figura 2.1 encontramos algunas de ellas, suficientes para tener una idea de dicho concepto.

Sistemas que piensan como humanos	**Sistemas que piensan racionalmente**
«El nuevo y excitante esfuerzo de hacer que los computadores piensen… máquinas con mentes, en el más amplio sentido literal». (Haugeland, 1985)	«El estudio de las facultades mentales mediante el uso de modelos computacionales». (Charniak y McDermott, 1985)
«[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje…» (Bellman, 1978)	«El estudio de los cálculos que hacen posible percibir, razonar y actuar». (Winston, 1992)
Sistemas que actúan como humanos	**Sistemas que actúan racionalmente**
«El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren de inteligencia». (Kurzweil, 1990)	«La Inteligencia Computacional es el estudio del diseño de agentes inteligentes». (Poole *et al.*, 1998)
«El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor». (Rich y Knight, 1991)	«IA… está relacionada con conductas inteligentes en artefactos». (Nilsson, 1998)

Figura 2.1: Definiciones de IA desde distintos enfoques.

Aún siendo un campo de estudio que despierta interés entre los científicos, sigue pendiente de investigación en varios ámbitos y cuestiones. Esto se debe al amplio abanico de posibilidades que ofrece la inteligencia artificial en lo que respecta a la resolución de problemas.

Entre las ramas que presenta, destaca el **aprendizaje automático** (*machine learning*), por su capacidad de adaptación y resolución de problemas. Para definirlo, podríamos decir que un programa informático aprende de una experiencia E con respecto a alguna tarea T y una medición del rendimiento P, si su rendimiento en la tarea T, medido como P, mejora con la experiencia E [8]. Otra aproximación algo más informal pero ciertamente ilustrativa presenta el aprendizaje automático como el campo de estudio que otorga a los ordenadores la habilidad de aprender sin haber sido explícitamente programados para resolver una tarea [9].

Se trata también de un área del conocimiento bastante popular, dadas las soluciones que ha permitido desarrollar en problemas antes irresolubles. Dentro del aprendizaje automático, se pueden distinguir principalmente tres tipos, según exista o no la supervisión humana "durante" el aprendizaje del agente:

Aprendizaje supervisado. El objetivo es deducir una función a partir de **datos** de entrenamiento debidamente **etiquetados**. Cada instancia de los datos consiste en un conjunto de propiedades que la caracterizan, acompañados de un dato (etiqueta) que indica el resultado deseado correspondiente a dicha entrada. La función debe ser capaz de devolver una respuesta correcta ante datos nuevos (nunca vistos por el agente). Se distinguen **problemas de regresión y clasificación**, según si el dominio de las etiquetas es continuo o discreto.

Aprendizaje no supervisado. En este caso, el conjunto de entrenamiento no está etiquetado. Al no existir conocimiento a priori, el agente trata de **descubrir patrones** sobre los datos para así poder asignarles una etiqueta automáticamente. Este tipo de aprendizaje es el que se usa para agrupar los datos en subconjuntos (*clustering*).

Aprendizaje semisupervisado. Propuesto por autores como S. Abney o Mitchell; aparece cuando una parte (generalmente una minoría) de los datos están etiquetados, y el resto no [10].

Aprendizaje por refuerzo. En él se trata de determinar qué acciones debe escoger un agente en un entorno dado con el fin de maximizar alguna recompensa. De esta manera, se sigue una estrategia de ensayo y error mediante la que **el propio agente genera la experiencia** para el aprendizaje, durante su interacción con el entorno. Este tipo

de aprendizaje será explicado con mayor profundidad en el siguiente apartado (sección 2.2).

Existen otros criterios en función de los cuales realizar otras clasificaciones, como por ejemplo:

- ***Online learning* y *batch/offline learning*** dependiendo de si pueden aprender de forma incremental mientras son utilizados o si, por el contrario, requieren una fase independiente dedicada al entrenamiento antes de ser utilizados.
- **Aprendizaje basado en modelo y basado en instancia**, dependiendo de si se genera un modelo o simplemente se compara con los datos ya conocidos.

En el marco de este trabajo fin de grado, nos centraremos en el Aprendizaje por Refuerzo que, como veremos, es el que mejor se adapta al problema que vamos a abordar.

2.2. Aprendizaje por refuerzo

Al igual que la inteligencia artificial, el aprendizaje por refuerzo, o *reinforcement learning* (de aquí en adelante RL), es más "antiguo" de lo que se suele pensar. Sus orígenes datan de los años 50, donde coexistían dos líneas de investigación independientes que trataban el problema desde distintos enfoques. Una estudiaba el aprendizaje mediante el **paradigma de ensayo y error**, mientras que la otra, más teórica, centraba su atención en el problema de control óptimo y su solución mediante funciones de valor y programación dinámica, aunque no involucraba aprendizaje. Ambas corrientes se fusionaron en los años 80, dando lugar a un resurgimiento del aprendizaje por refuerzo más similar al enfoque moderno [11]. En los últimos años, el RL ha recobrado aún más protagonismo por las posibilidades computacionales actuales y su integración con técnicas de *deep learning*, además de sus éxitos en juegos de Atari [12] y algunos más clásicos como Go [13].

El término de aprendizaje por refuerzo hace referencia tanto a un tipo de problema, como a la clase de métodos que ofrecen una solución a estos, y la ciencia que los estudia. En cualquier caso, podría ser definido como un tipo de aprendizaje computacional donde el objetivo del agente consiste en **maximizar una señal de recompensa numérica**[1]. No se indica al agente qué acción debe tomar en cada momento, sino que este debe descubrir cuáles son aquellas que le permiten maximizar no solo la recompensa

[1]Tras cada acción ejecutada (o instante de tiempo transcurrido), el agente recibe una señal de recompensa del entorno, que sirve como indicador del beneficio o perjuicio obtenido por realizar dicha acción, teniendo en cuenta el estado en que se halla el entorno.

inmediata sino, sobre todo, la que recibirá a largo plazo [11]. Para ello debe interaccionar con su entorno y aproximar una política óptima mediante una estrategia de ensayo y error. Asimismo, constituye un paradigma especialmente útil para **problemas de decisión**, donde subyace el concepto de procesos de decisión de Markov, explicado con mayor detenimiento más adelante.

Cabe destacar que en el reciente resurgimiento del aprendizaje automático (a excepción quizás del último lustro), tanto el aprendizaje supervisado como el no supervisado han eclipsado en cierto modo al RL, restándole importancia como un tipo de aprendizaje en sí mismo. Sin embargo, expertos como Sutton y Barto defienden la presencia de importantes diferencias respecto a estos dos paradigmas [11].

Por un lado, y como hemos visto, el **aprendizaje supervisado** trabaja con datos previamente etiquetados, tratando de aproximar una función que generalice o extrapole correctamente los datos para responder con éxito ante datos nuevos. En problemas interactivos, es inviable utilizar este paradigma para el aprendizaje, pues no suele ser posible extraer una muestra de datos correcta y representativa de todos los posibles estados.

Por su parte, el **aprendizaje no supervisado** suele tratar de encontrar una estructura "oculta" en los datos. Si bien la búsqueda de esta estructura puede ser útil para un problema de RL, no se cumple el objetivo de maximizar una señal de recompensa. Dicho esto, no deben confundirse estos tipos de aprendizaje por el simple hecho de que ambos trabajen con datos sin etiquetar.

Otro aspecto relevante en RL es la necesidad de buscar un **equilibrio entre explotación y exploración**[2]. El problema reside en que ambos procesos no pueden realizarse de forma simultánea, por lo que es imposible explorar o explotar soluciones en el espacio de búsqueda sin cometer errores. La aproximación más habitual ante este problema propone explorar las distintas acciones en la medida de lo posible y, progresivamente, ir explotando aquellas que parecen más prometedoras.

2.2.1. Elementos del aprendizaje por refuerzo

En cualquier sistema de RL, se pueden distinguir dos elementos esenciales: un agente y el entorno con el que interactúa:

[2]En RL, la explotación trata de elegir aquellas acciones de las que el agente sabe que obtendrá la mayor recompensa. La exploración consiste en tomar acciones subóptimas para intentar encontrar y obtener futuras recompensas mayores que las conocidas.

- El **agente** es el responsable del aprendizaje y la toma de decisiones. Su objetivo consiste en aprender un comportamiento que le permita maximizar la señal de recompensa, a base de interaccionar con el entorno. Así, en cada momento, el agente debe obtener información de dicho entorno para determinar en qué estado se encuentra, una señal de recompensa que indica la "calidad" de ese estado, y determinar qué acción va a ejecutar en función de esta información.

- Una posible forma de definir el **entorno** es describirlo como «todo aquello que el agente no puede modificar de forma arbitraria» [3][11]. Esto también incluiría la función de recompensa, a pesar de que el agente pueda conocerla. El comportamiento del entorno se define por su *dinámica*, la cual no es necesariamente conocida por el agente [4]. Además, si esta dinámica varía en el tiempo, se trataría de un entorno **no estacionario** (en contraposición a los entornos estacionarios).

Aparte de estos, existen **otros componentes** presentes en los problemas de RL. Se describen a continuación.

- Una **política** que define el comportamiento del agente en cada momento. A partir de una observación del estado actual del entorno, devuelve la acción que tomará el agente. Constituye el núcleo del agente de RL, ya que son suficientes para determinar su comportamiento. Son muy habituales las **políticas estocásticas**, que devuelven una distribución de probabilidad sobre las posibles acciones a realizar.

- La **señal de recompensa**, que define el objetivo del problema. En cada instante de tiempo, el agente recibe esta señal, que le permite conocer el beneficio o perjuicio de cada evento. Tiene un carácter "cortoplacista".

- Por otro lado, la **función de valor** aporta al agente una valoración más a largo plazo. Estima la recompensa acumulada esperable que el agente puede obtener partiendo de un estado. De esta manera, evita el sesgo intrínseco presente en la señal de recompensa, pues permite tener una **visión más general** de cuan prometedor es un estado. Posibilita también la elección de estados con peor recompensa inmediata, pero con mayor recompensa esperable en un futuro. Esto no resta importancia a la señal de recompensa, ya que sin ella sería imposible estimar la función de valor.

[3] La separación entre el agente y el entorno representa el límite del control absoluto del agente, no de su conocimiento.

[4] De hecho, la dinámica suele ser parcial o totalmente desconocida en la inmensa mayoría de los problemas con aplicación en el mundo real.

- Opcionalmente puede existir un **modelo** del entorno. Este determina su dinámica, la cual define su comportamiento indicando cómo afecta cada una de las acciones del agente a su estado. En otras palabras, permite conocer a priori las transiciones de estados según la acción ejecutada en cada uno. Son muy útiles para realizar cualquier tipo de planificación[5]. Dan lugar a los **métodos basados en modelo** y, por contraposición, a los **métodos libres de modelo** (*model-based* y *model-free*, respectivamente). De hecho, en la mayoría de problemas del mundo real, no se dispone de un modelo del entorno, por lo que resultan necesarios los métodos libres de modelo, en consecuencia, la aplicación de técnicas de ensayo y error.

En cualquier caso, estos conceptos serán formalizados y adaptados al problema enfrentado en la sección 2.5.

2.2.2. Aspectos adicionales de RL

Llegados a este punto, se pueden describir algunas características relevantes que permiten clasificar los problemas en RL según distintos criterios. Centrémonos en la distinción de problemas episódicos y continuados, y las tareas de predicción y control.

El proceso de interacción y aprendizaje de un agente con su entorno puede dividirse en distintos intervalos de tiempo (pasos o *timesteps*). En cada uno de ellos, el agente observa el estado y decide qué acción va a ejecutar. Pues bien, dependiendo de si tras un número finito de pasos se alcanza o no un estado «terminal» y, consecuentemente, la tarea del agente se da por finalizada, diremos que se trata de un **problema episódico o continuado**, respectivamente. Se puede adelantar que el problema tratado en este trabajo es continuado, ya que no existen estados finales. En el caso de los problemas continuados, al no existir estos estados terminales, resulta necesaria la inclusión de limitaciones temporales que permitan la finalización de episodios.

Además, podemos encontrar dos enfoques distintos en problemas de aprendizaje por refuerzo. Por un lado, existen las **tareas de predicción**, que consisten en, dada una política π, predecir la recompensa total esperable al partir de un estado y seguir el comportamiento que π dictamina. Por otro lado, el objetivo en las **tareas de control** es aproximar una política óptima que maximice la recompensa total esperable para cualquier estado. De nuevo, se puede adelantar que la tarea resuelta en este proyecto fin de grado es de control.

[5] La planificación permite elegir una acción considerando distintas situaciones futuras sin necesidad de experimentarlas.

2.2.3. Procesos de decisión de Markov

Para formalizar un problema de RL no es suficiente con los términos anteriormente descritos, los cuales son demasiado abstractos, sino que resulta necesario definir todos estos conceptos en un **marco estandarizado**. Así pues, los procesos de decisión de Markov (PDM, o MDP en inglés)[14] cobran vital importancia en este subcampo del aprendizaje automático. Gracias a ellos, se pueden formalizar matemáticamente los problemas de RL [11, 15], permitiendo así garantizar la optimización de algunos algoritmos bajo ciertas condiciones. Un MDP está formado por los siguientes elementos:

- Un **conjunto de estados** $\mathcal{S}$, compuesto por todas las situaciones que el entorno puede presentar durante el proceso de interacción [6].
- Un **espacio de acciones** $\mathcal{A}$, que indica las distintas formas en que el agente puede actuar en cada *timestep*.
- La **función de recompensa** $\mathcal{R}$, que cuantifica la señal de recompensa antes mencionada.
- La distribución de **probabilidades de transición** $\mathcal{P}$ permite conocer de forma exacta qué probabilidades hay de pasar de un estado concreto a otro según la acción que elija el agente. En otras palabras, define la dinámica o comportamiento del entorno de forma numérica.
- Por último, un **factor de descuento** γ que determina la importancia de las recompensas futuras respecto de las inmediatas. Por ejemplo, un valor de $\gamma = 1$ implica que no importa cuan lejana sea la recompensa, pues tendrá el mismo peso que una inmediata.

En un **proceso de decisión de Markov finito**, los conjuntos $\mathcal{S}$, $\mathcal{A}$ y $\mathcal{R}$ son finitos.

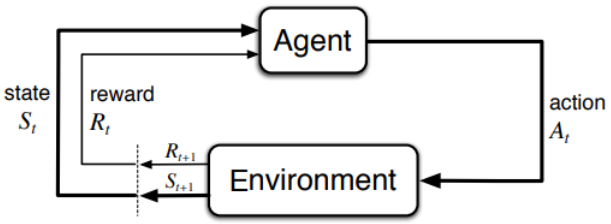


Figura 2.2: Esquema de interacción agente-entorno en un MDP.

[6] Cuando el agente no dispone de una observación total y correcta del estado del entorno, sino que existe cierta incertidumbre, se trataría de un proceso de decisión de Markov parcialmente observable (POMDP), concepto bastante importante en RL.

En base a estos conceptos, surge la propiedad de Markov, enunciada de la siguiente manera:

Propiedad de Markov

Se dice que un estado cumple la propiedad de Markov si la probabilidad de transición a un estado S_t y de obtener una recompensa R_t depende únicamente del estado S_{t-1} y acción A_{t-1} inmediatamente anteriores.

Se puede formular matemáticamente de la siguiente manera:

$$p(s' \mid s_t, a_t) \doteq p(s' \mid s_t,\ a_t,\ s_{t-1},\ a_{t-1},\ s_{t-2},\ a_{t-2},\ ...,\ s_0,\ a_0) \tag{2.1}$$

Otro aspecto a destacar sobre $\mathcal{P}$ es que, por ser las transiciones eventos excluyentes, su suma debe ser igual a 1, esto es:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, for\ all\ s\ in\ \mathcal{S},\ a\ in\ \mathcal{A}(s) \tag{2.2}$$

Según la distribución $\mathcal{P}$, se distinguen dos tipos de entornos. Por un lado existe el **caso determinista**, donde dado un estado s y una acción a ejecutar a hay una única posible transición a un estado s', lógicamente con probabilidad 1 (véase 2.2). Por el contrario, en **entornos estocásticos** aparece cierto azar en el resultado de las acciones. Dicho de otra forma, para cada estado s y acción a, varias transiciones son posibles, teniendo cada una, una probabilidad en el intervalo $(0, 1]$. En consecuencia, no se puede conocer con total certeza el impacto que tendrá el agente sobre el entorno al ejecutar determinadas acciones. Considerando esta dificultad añadida, es lógico pensar que generalmente es más complejo aproximar una solución óptima en entornos estocásticos.

Para continuar con la formalización de los problemas de RL, recuperemos el concepto de recompensa una vez más. Recordemos que el objetivo principal de un agente es maximizar la suma de una señal escalar[7], denominada recompensa (hipótesis de la recompensa [11]). En el caso más simple,

[7] A pesar de que la formulación de objetivos en términos de señales de recompensa pueda parecer limitante, se ha probado empíricamente su gran flexibilidad y posibilidad de aplicación en inmensidad de problemas [11].

la **recompensa acumulada** (*return*) a maximizar, notada como G, es la suma de las recompensas obtenidas durante el proceso:

$$G_t \doteq R_{t+1} \ + \ R_{t+2} \ + \ ... \ + R_T, \text{ siendo T el último } \textit{timestep} \tag{2.3}$$

Desde el punto de vista del aprendizaje, la **recompensa estimada** R en función de un estado y una acción se formula así:

$$r(s,a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, \ A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a) \tag{2.4}$$

Llegados a este punto, es momento de concretar cómo afecta exactamente el ya presentado **factor de descuento** (γ) en la recompensa. Este parámetro pondera la importancia que toma cada recompensa según el momento en que el agente la reciba, dando más importancia a las decisiones inmediatas y devaluando las recompensas a medida que se alejan del instante presente. El objetivo es **modelar la incertidumbre** de las recompensas futuras. La siguiente expresión resulta bastante ilustrativa:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... \tag{2.5}$$

Que si definimos recursivamente, resulta en:

$$G_t \doteq R_{t+1} + \gamma G_{t+1} \tag{2.6}$$

2.2.4. Políticas y funciones de valor

Casi todos los algoritmos de RL implican la estimación de **funciones de valor** (*value functions*). Estas son funciones de estados (o de pares estado-acción) que tratan de estimar el beneficio que aporta al agente el hecho de alcanzar un estado (o tomar una acción en un estado concreto)[8]. Teniendo en cuenta que las recompensas dependen directamente de las acciones elegidas, las funciones de valor se definen en función de una política dada.

Formalmente, una **política** es un mapeo de estados a probabilidades de seleccionar cada posible acción. Si el agente está siguiendo una política π

[8]La noción de "beneficio" se define en términos de recompensa esperable según las acciones que se elijan.

en un instante t, entonces $\pi(a|s)$ es la probabilidad de $A_t = a$ si $S_t = t$ [9][10]. Recordemos que se distinguen políticas deterministas y estocásticas. En el primer caso, para cada estado devuelven una única acción, mientras que en el segundo se devuelve una distribución que indica la probabilidad con la que se tomará cada una de las posibles acciones.

Así, podemos definir la **función estado-valor** de un estado s bajo una política π, notada $v_\pi(s)$, como la recompensa acumulada esperable partiendo de s siguiendo la política π. En el marco de los MDPs, se formula:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right], \; \forall s \in \mathcal{S} \end{aligned} \tag{2.7}$$

Nótese que, en caso de existir un estado terminal, su función de valor siempre será 0.

De forma análoga se define una **función de acción-valor** (para un par estado-acción), notada como $q_\pi(s, a)$. Esta expresa la recompensa acumulada esperable si se toma una acción a en un estado s y se continúa siguiendo la política π:

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s, \; A_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s, \; A_t = a\right] \end{aligned} \tag{2.8}$$

El valor real de ambas funciones se puede estimar mediante la experiencia, promediando la recompensa acumulada que se obtiene partiendo de distintos estados a lo largo de distintas ejecuciones.

Una propiedad fundamental en RL y programación dinámica es que las funciones de valor satisfacen relaciones recursivas similares a la expresión 2.6. Nace así la conocida como ecuación de Bellman [16]:

[9] A_t y S_t son la acción elegida y el estado del entorno en un instante de tiempo t, respectivamente.

[10] No se debe confundir con π la dinámica del entorno definida por $\mathcal{P}$.

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \mid S_t = s\Big] \\
&= \sum_a \pi(a|s) \sum_{s'}\sum_r p(s', r|s, a)\Big[r + \gamma\mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\Big] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big],\ \forall s \in \mathcal{S}
\end{aligned}
\tag{2.9}
$$

De la que se puede derivar otra igualdad aplicable a la función acción-valor:

$$
\begin{aligned}
q_\pi(s, a) &\doteq \mathbb{E}_\pi\Big[G_t \mid S_t = s, A_t = a\Big] \\
&= \sum_{s'}\sum_r p(s', r|s, a)\Big[r + \gamma\mathbb{E}_\pi\big[G_{t+1} \mid S_{t+1} = s'\big]\Big] \\
&= \sum_{s'}\sum_r p(s', r|s, a)\Big[r + \gamma\sum_{a'}\pi(a'|s')q_\pi(s', a')\Big]
\end{aligned}
\tag{2.10}
$$

Estas expresiones serán útiles para el desarrollo de apartados posteriores.

Optimización de políticas y funciones de valor

En base a los conceptos definidos, la resolución de un problema de RL se reduce a encontrar una política que permita maximizar la recompensa obtenida acumulada a largo plazo. En el marco teórico de los MDP, se puede formalizar que una política es mejor que otra si la recompensa acumulada esperable de la primera es mayor que la de la segunda, en todos los estados. Así,

$$
\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s)\ \forall s \in \mathcal{S} \tag{2.11}
$$

De aquí se deduce que siempre existirá al menos una política mejor o igual que el resto, denominada **política óptima** (π_*)[11]. Todas las políticas óptimas comparten una misma función estado-valor, que se nota con $v_*(s)$ y se define:

[11] Nótese que pueden existir varias políticas óptimas en un mismo problema. Con π_* se hace referencia a todas ellas.

$$v_*(s) \doteq \max_\pi v_\pi(s),\ \forall s \in \mathcal{S} \tag{2.12}$$

De la misma manera, las políticas óptimas comparten la misma función acción-valor q_*:

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a),\ \forall s \in \mathcal{S},\ a \in \mathcal{A} \tag{2.13}$$

Por ser v_* una función de valor para una política, debe satisfacer la condición de autoconsistencia dada por la ecuación de Bellman para las funciones estado-valor (véase 2.9). Por tratarse de la función estado-valor óptima, dicha condición de autoconsistencia se puede expresar sin hacer referencia a ninguna política concreta[12]. Es lo que se conoce como **ecuación de optimización de Bellman**:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s,\ A_t = a] \qquad \text{(por 2.6)} \\ &= \max_a \mathbb{E}\big[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s,\ A_t = a\big] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_*(s')\big] \end{aligned} \tag{2.14}$$

De nuevo, se puede definir una ecuación equivalente para q_*:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \end{aligned} \tag{2.15}$$

Sobre el aspecto de optimización presentado en este apartado, es importante añadir que, en la práctica, rara vez se consigue conocer las funciones de valor óptimas, por lo que nos limitamos a su **estimación**. Esto se debe a que los recursos computacionales de una máquina son limitados y generalmente insuficientes para la mayoría de problemas reales. Por tanto, se suelen estimar aproximaciones a la solución óptima. Además de los límites en la potencia computacional, la memoria disponible también conforma una restricción importante. En muchos de los problemas reales existen un número de estados demasiado grande como para poder almacenarse en una

[12] Esto se debe a que es única, y compartida por todas las políticas óptimas.

tabla en memoria. Nuevamente, surge la necesidad de realizar una aproximación, dando lugar así a los **métodos no tabulares** (en contraposición a los métodos tabulares).

2.3. Iteración de la política. Iteración de valor

Una vez formalizados estos conceptos, llega el momento de concretar el proceso de resolución del problema. Como ya se ha explicado, el objetivo es encontrar o, al menos aproximar una política óptima que determine el comportamiento del agente. Para esta tarea, primero se debe definir cómo se va a evaluar una política (para así poder compararlas entre ellas) y cómo se puede mejorar. Aparecen entonces dos conceptos nuevos: evaluación de la política (*policy evaluation*) y mejora de la política (*policy improvement*).

2.3.1. Evaluación de la política

Se puede intuir que para evaluar una política, se debe calcular su función estado-valor[13]. Para ello, el valor inicial de esta función v_0 se puede elegir de forma arbitraria (excepto para el estado terminal, que siempre es 0). A partir de ahí, cada aproximación sucesiva se obtiene aplicando la siguiente regla, basada en la ecuación de Bellman:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}),\ \mid S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big],\ \forall s \in \mathcal{S} \end{aligned} \tag{2.16}$$

Esta expresión es la base del **algoritmo de evaluación iterativa de la política** (*iterative policy evaluation*), descrito a continuación:

[13] En la literatura científica, el problema del cálculo de una función estado-valor v_π para una política arbitraria π también se conoce como **problema de predicción** (*prediction problem*) [11].

Algoritmo 1: Evaluación iterativa de la política

Entrada: - una política π a evaluar
- el conjunto de estados $\mathcal{S}$
- el espacio de acciones $\mathcal{A}$
- el factor de descuento γ
- un umbral de precisión θ

---

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

Asimismo, este algoritmo permite **aproximar la función estado-valor** de una política con la precisión que se desee (se sabe que converge cuando el número de iteraciones tiende a infinito), para poder comparar así distintas políticas. La estrategia que sigue consiste en ir actualizando de forma iterativa la función estado-valor, hasta que dicha actualización sea menor que un umbral dado. La estimación de la función v de cada estado s se basa en la recompensa asociada a pasar a cada estado s' ponderado con su probabilidad de transición, todo ello nuevamente ponderado por la probabilidad de realizar esa transición si se sigue una política π.

2.3.2. Mejora de la política

Una vez se tiene un mecanismo que permite comparar distintas políticas (considerando 2.16, para aproximar una función estado-valor, y la expresión 2.11 para comparar políticas), se puede mejorar una política π dada. Así, partiendo de un estado s, se asigna como mejor opción la acción a que, al ser ejecutada, lleve al agente a un estado s' con la **mayor recompensa acumulada esperable** posible. Así, para estimar la recompensa esperable tras realizar cada acción, se considera la suma de recompensas asociadas a cada posible estado s' y se ponderan por la probabilidad de transición de s a s'. Esta estrategia se formaliza de la siguiente manera:

$$\begin{aligned} \pi'(s) &\doteq \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big] \end{aligned} \tag{2.17}$$

2.3.3. Iteración de la política e iteración de valor

Una vez definidos los métodos para evaluación y mejora de la política, se puede concretar un **algoritmo iterativo** que aproxime la política óptima. Este proceso consiste en aplicar una evaluación de la política y una mejora, de forma sucesiva:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{M} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{M} \pi_2 \xrightarrow{E} \ldots \xrightarrow{M} \pi_* \xrightarrow{E} v_*$$

Se conoce como **iteración de la política** (*policy iteration*), descrito como sigue:

Algoritmo 2: Iteración de la política

Entrada: - una política π a evaluar
- el conjunto de estados $\mathcal{S}$
- el espacio de acciones $\mathcal{A}$
- el factor de descuento γ

---

[1] Inicializar $V(s) \in \mathbb{R}$ y $\pi(s) \in \mathcal{A}(s)$ arbitrariamente $\forall s \in \mathcal{S}$

[2] Evaluación de la política

repetir
  $\Delta \leftarrow 0$
  para cada $s \in \mathcal{S}$ **hacer**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  fin
mientras $\Delta > \theta$ *(determina la precisión de la estimación)*

[3] Mejora de la política

politica_estable $\leftarrow$ *true*
para cada $s \in \mathcal{S}$ **hacer**
  accion_anterior $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$ **si**
   accion_anterior $\neq \pi(s)$ **entonces**
    politica_estable $\leftarrow$ *false*
fin
si *politica_estable* **entonces**
  devolver $V \approx v_*$ y $\pi \approx \pi_*$
en otro caso
  ir a [2]
fin

Se puede observar que la estructura de este algoritmo consiste en alternar evaluación de la política y mejora de la misma. Se itera hasta que la política no varíe, lo cual querrá decir que se ha encontrado la óptima.

Existe otra alternativa a la iteración de política, conocida como **iteración de valor** (*value iteration*). Como su nombre indica, también tiene carácter iterativo. A diferencia de la iteración de política, este método trabaja sobre la **función estado-valor**. Partiendo de una función $v(s)$ arbitraria, se va aplicando la estrategia de mejora (véase 2.17) iterativamente hasta alcanzar una aproximación a la función óptima $v_*(s)$ aceptable. Llegado este punto, se genera una política determinista $\pi \approx \pi_*$ eligiendo, para cada estado s, la acción con mayor valor asociado según la $v(s)$ aproximada. Este es el algoritmo:

Algoritmo 3: Iteración de valor

Entrada: - el conjunto de estados $\mathcal{S}$
- el espacio de acciones $\mathcal{A}$
- el factor de descuento γ
- un umbral de precisión θ

Salida: - La política $\pi \approx \pi_*$ aproximada

---

repetir
  $\Delta \leftarrow 0$
  para cada $s \in \mathcal{S}$ **hacer**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  fin
mientras $\Delta > \theta$

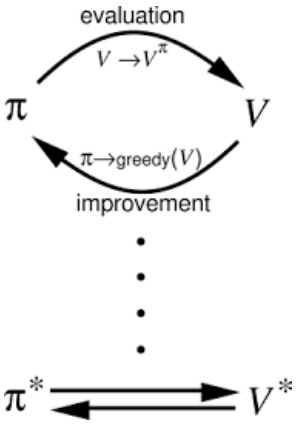
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

devolver $\pi(s)$

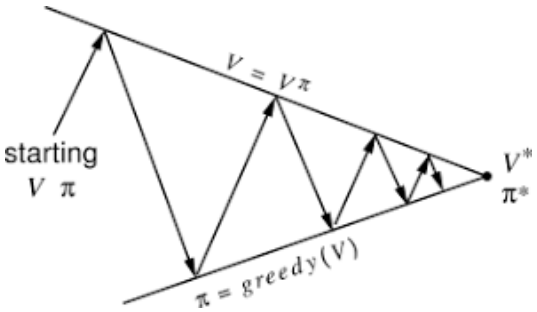
Se ha probado empíricamente que el método de iteración de política **converge antes** que iteración de valor, pues la política se aproxima antes a la óptima que la función estado-valor [11]. Sin embargo, el primer proceso es **más costoso** computacionalmente hablando, ya que ejecuta una fase de evaluación y otra de mejora de la política en cada paso.

Iteración de Política Generalizada

Para cerrar este apartado, es interesante incluir el concepto de **Iteración de Política Generalizada** (GPI). Como se ha explicado, en el algoritmo de iteración de política se va alternando una fase de evaluación completa de la política y otra de mejora. Sin embargo, no se trata de una condición necesaria (algunos métodos de programación dinámica asíncrona intercalan estos procesos incluso antes de que acabe alguna de estas fases, para lograr mayor eficiencia). En general, se denomina iteración de política generalizada a esta dinámica que alterna entre evaluación y mejora de la política, ignorando la granularidad y otros detalles de ambos procesos. Es un concepto muy relevante ya que casi todos los algoritmos de RL están basados en esta idea [11].



(a) Esquema GPI.



(b) Representación bidimensional de evolución de GPI.

Figura 2.3: Imágenes ilustrativas del modelo GPI.

2.4. Algoritmos de aprendizaje por refuerzo

2.4.1. Algoritmos clásicos

Método de Monte Carlo

El método de Monte Carlo es probablemente el más sencillo en RL. Según está planteado, el agente aprende mediante la interacción con el entorno. Durante dicha interacción, el agente genera **distintas trayectorias** en el espacio de estados y acciones, almacenando la recompensa que recibe en cada caso[14]. Basándose en el promedio de la recompensa obtenida, se aproxima

[14] A pesar de que en ML el término Monte Carlo suele hacer referencia a la aleatoriedad, en este caso abarca aquellos métodos basados en estimar recompensas a largo plazo a partir del "sampleado" de trayectorias.

la función estado-valor y/o acción-valor. De esta manera, no se requiere conocimiento completo del entorno, sino que basta con un modelo capaz de generar muestras y simular dicho entorno, esquivando así la necesidad de conocer la distribución de probabilidades de las transiciones (como sucede en métodos como programación dinámica).

Supongamos que se desea estimar la función estado-valor $v_\pi(s)$ dado un conjunto de episodios obtenidos al seguir π y pasando por el estado s. Cada ocurrencia de dicho estado s en un episodio se denomina *visita* a s. Lógicamente, un mismo estado puede ser visitado varias veces durante un mismo episodio. Así pues, surgen dos variantes del método de Monte Carlo según se consideren todas las visitas al estado s durante los episodios para estimar la función $v_\pi(s)$ (***every-visit MC method***) o tan solo la primera visita al estado de cada episodio (***first-visit MC method***). Este último ha sido estudiado con mayor profundidad desde 1940 [11], siendo el más utilizado en RL[15]. Se muestra a continuación la versión *first-visit* del método de Monte Carlo:

Algoritmo 4: *First-visit Monte Carlo method*

Entrada: - π: política a evaluar

Inicializar $V(s) \in \mathbb{R}$ arbitrariamente, para todo $s \in \mathcal{S}$
Inicializar recompensas de $\mathcal{S}$ como una lista vacía $recompensas(s)$
repetir
  Generar un episodio siguiendo π
  $G \leftarrow 0$
  para cada *instante de tiempo t del episodio* **hacer**
    $G \leftarrow \lambda G + R_{t+1}$
    si $S_t \notin [S_0, S_{t-1}]$ **entonces**
      Añadir G a $recompensas(s)$
      $V(S_t) \leftarrow promedio(recompensas(S_t))$
  fin
hasta que *converja*

De esta manera, se estima la función estado-valor de cada estado (o acción-valor de cada par estado-acción)[16], la cual sirve como indicativo de cuan positivo es alcanzar un estado (o ejecutar cierta acción). En pocas palabras, se trata de un **promedio de las recompensas obtenidas** durante las ejecuciones. Se ha probado matemáticamente que este promedio converge al valor real de las funciones según aumenta el número de episodios

[15] Ambas opciones son bastante similares, presentando sólo ligeras diferencias en algunas propiedades teóricas.

[16] La estimación de cada estado queda recogida en la tabla representada por V.

ejecutados o simulados [11]. Por otro lado, es capaz de tratar el conflicto entre explotación y exploración aplicando una **política ε-greedy** a la hora de generar episodios.

Sin embargo, los métodos de Monte Carlo no suelen ofrecer una solución aceptable cuando la dimensión del problema crece mínimamente. Esto se debe a que para alcanzar la convergencia debe completar numerosos episodios. Además, si el número de episodios es bajo, aparece una gran variabilidad en los resultados, pues no consigue generalizar bien los posibles estados y acciones con pocas ejecuciones. Otro inconveniente que presenta es la necesidad de esperar a que el episodio finalice para actualizar la función que trata de aproximar (este problema afecta en especial a tareas continuas, no episódicas).

Teniendo estos aspectos en cuenta, resulta necesaria la búsqueda e investigación de otros algoritmos con una mayor aplicabilidad y eficacia.

Diferencia temporal

Según autores como Richard Sutton y Andrew Barto, el aprendizaje por diferencia temporal (*temporal difference learning*) puede considerarse como una de las ideas centrales del aprendizaje por refuerzo [11]. Abreviado TD (por sus siglas en inglés), el aprendizaje por diferencia temporal es una combinación de los métodos de Monte Carlo y programación dinámica. Se trata de un **método libre de modelo** que constituye la base de algoritmos más sofisticados como algunos de los que se verán más adelante.

Para explicar este algoritmo, retrocedamos por un momento al método de Monte Carlo estudiado en el apartado anterior (véase 2.4.1). Tanto los métodos de TD como MC utilizan la experiencia para resolver el problema de predicción. Sin embargo, la diferencia entre ambos radica precisamente en cómo actualizan su aproximación V de v_π. En el caso más sencillo del método *every-visit* MC, se aplica la siguiente actualización para un entorno no estacionario:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

donde G_t es la recompensa real acumulada a partir del instante de tiempo t, y α es una constante (*step-size*) que pondera el peso de la nueva estimación (si es 0 la actualización será nula y si es 1 se ignorará la estimación previa). Por el hecho de utilizar G_t, los métodos de MC deben esperar a que finalice el episodio para realizar dicha actualización. Por su parte, la forma más simple de TD plantea la siguiente actualización:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big] \tag{2.18}$$

En lugar de utilizar la recompensa acumulada G_t, encontramos que se actualiza con la suma ponderada de la recompensa inmediata R_{t+1} y la aproximación de la función estado-valor del estado alcanzado $V(S_{t+1})$, por lo que se actualiza la estimación V en cada *time-step* y desaparece así la necesidad de esperar hasta el final del episodio (*online learning*).

Concretamente, la ecuación descrita en 2.18 es la que utiliza TD(0) o *one-step TD*, siendo un caso especial de TD(λ) o de *n-step TD*, siendo su pseudocódigo el que sigue:

Algoritmo 5: Método tabular TD(0) para la estimación de v_π

Entrada: - π: la política a evaluar
- $\alpha \in [0, 1]$: *step-size*

Inicializar $V(s) \in \mathbb{R}$ arbitrariamente, para todo $s \in \mathcal{S}^+$ excepto $V(terminal) = 0$
para cada *episodio* **hacer**
  Inicializar S
  para cada *instante t, hasta que S sea terminal* **hacer**
    $A \leftarrow$ acción elegida por π para S
    Ejecutar acción A, observar R, S'
    $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  fin
fin

Se dice que TD(0), al igual que los métodos de programación dinámica, realiza ***bootstrapping***, ya que basa su actualización en una estimación ya existente. Vemos cómo TD combina el muestreo de MC y el *bootstrapping* de DP.

Antes se mencionaba que TD(0) es la forma más simple de los métodos de diferencia temporal. Esto es porque sólo se tiene en cuenta la siguiente acción para la estimación de la función de valor. Sin embargo, se pueden tener en cuenta las n siguientes acciones, dando lugar a ***n-step Temporal Difference***. Ante la elección de qué número de pasos conviene considerar, un planteamiento posible propone combinar la información de las siguientes n acciones. Esta idea es formalizada por el algoritmo **TD(λ)**, que busca maximizar la generalización utilizando una actualización basada en la re-

compensa G_t^λ siguiente:

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n},$$

siendo G_t :

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1}R_{t+n} + \gamma^n \hat{v}(S_{t+n}, w_{t+n-1}),\ 0 \leq t \leq T-n \tag{2.19}$$

Nótese cómo TD(λ) es equivalente a MC cuando $\lambda = 1$ y, lógicamente, a TD(0) cuando $\lambda = 0$, logrando la generalización anteriormente mencionada. De esta misma manera, supera a los métodos de Monte Carlo en distintos aspectos:

- Permite realizar ***online learning***, utilizando los datos que recopila en cada momento.
- Como consecuencia, puede trabajar con **episodios incompletos** y así, resolver tareas continuadas además de episódicas.
- Presenta una **menor variabilidad** que los métodos de MC, muy afectados por la aleatoriedad que implican.
- Sin embargo, **MC incorpora menos sesgo** por basar sus actualizaciones en la recompensa final y no en otras predicciones como hace TD.

SARSA

SARSA es otro algoritmo de vital importancia en aprendizaje por refuerzo. Surge al aplicar TD para resolver el **problema de control**. Asimismo, centra su atención en la función acción-valor (y las transiciones entre pares estado-acción) en lugar de la función estado-valor. Para aproximar dicha función, utiliza la fórmula:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big] \tag{2.20}$$

Como se observa en esta expresión, el cálculo de la función estado-valor se basa en la quíntupla $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, que da nombre a este método.

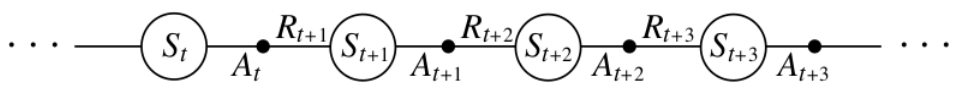


Figura 2.4: Esquema del método SARSA.

Dicho esto, resulta sencillo el diseño de un algoritmo de control ***on-policy***[17], como es SARSA. Como en todos los métodos de este tipo, se estima continuamente q_π para la política de comportamiento π al mismo tiempo que π varía de forma voraz (*greedy*) con respecto a q_π. Resulta el siguiente algoritmo:

Algoritmo 6: Método SARSA para estimación de $Q \approx q_*$

Entrada: - $\alpha \in (0, 1]$: *step-size*
- $\varepsilon > 0$ pequeño

Inicializar $Q(s, a)\ \forall\ s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ arbitrariamente, excepto $Q(terminal, \cdot) = 0$
para cada *episodio* **hacer**
  Inicializar S
  Elegir A de S utilizando una política derivada de Q
  para cada *instante del episodio, mientras que S no sea terminal* **hacer**
    Ejecutar acción A y observar R, S'
    Elegir acción A' de S' utilizando una política derivada de Q
    $Q(S, A) \leftarrow Q(S, A) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$
    $S \leftarrow S'\ ;\ A \leftarrow A'$
  fin
fin

Las propiedades de convergencia de SARSA dependen de la naturaleza de la dependencia de la política sobre Q (se podrían utilizar **políticas ε-*greedy* o ε-suaves** (ε-*soft policies*). SARSA converge a una política óptima y función acción-valor con probabilidad 1 siempre y cuando se visite cada par estado-acción un número infinito de veces y la política converja a una política voraz en el límite [11].

Finalmente, cabe mencionar una de las variantes más relevantes de este algoritmo. Se trata de ***n-step SARSA***. Esta variante considera una recompensa a mayor largo plazo que la inmediata, resultando la siguiente actualización de Q:

[17] Los métodos ***off-policy*** son aquellos que utilizan dos políticas diferentes: una es la política que aproximan y aprenden de forma iterativa y otra la que utilizan para la elección de acciones durante el aprendizaje y generar así interacciones con el entorno. Por el contrario, los métodos ***on-policy*** utilizan una misma para ambas tareas, asumiendo que se continuará siguiendo la política actual.

$$Q_{t+n}(S_t, A_t) \leftarrow Q_{t+n-1}(S_t, A_t) + \alpha\Big[G_{t:t+n} + \gamma Q_{t+n-1}(S_t, A_t)\Big] \quad (2.21)$$

Q-learning

El desarrollo de *Q-learning* por parte de Watkins es uno de los grandes hitos en la historia del RL [17], constituyendo la base además de varios algoritmos de aprendizaje por refuerzo profundo como veremos en 2.4.2. A pesar de su relevancia, no se trata de un algoritmo totalmente innovador, sino que se basa en la estructura de SARSA, con la diferencia de que este es un método ***off-policy***. Al igual que el método anterior, *Q-learning* trata de resolver el **problema de control**. Para ello, aprende una función acción-valor Q que trata de aproximar directamente la función óptima q_*, independientemente de la política que se siga. Esto simplifica drásticamente el análisis del algoritmo y permite probar su **rápida convergencia** [11]. Aunque la política tiene cierto efecto en qué pares estado-acción se visitan y actualizan, las condiciones para afirmar su convergencia son las mismas que para SARSA[18].

La principal característica de este algoritmo es la función que aproxima:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \quad (2.22)$$

En esta expresión observamos cómo en lugar de estimar el valor de la función considerando todas las posibles siguientes acciones, se considera aquella que **mayor valor** Q presenta, suponiendo así que se seguirá una política voraz. Se muestran a continuación esquemas comparativos entre SARSA y *Q-learning*:

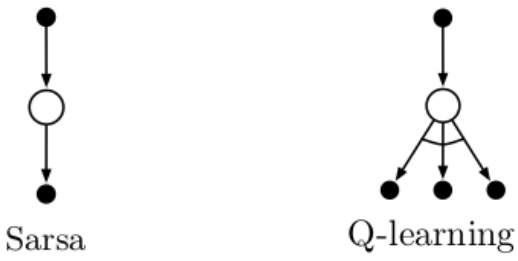


Figura 2.5: Esquemas comparativos de SARSA y *Q-learning*.

[18] En realidad este requisito es básico para probar la convergencia de cualquier método en RL [11].

De esta manera, queda definido *Q-learning*:

Algoritmo 7: Método Q-learning para estimación de $Q \approx q_*$

Entrada: - $\alpha \in (0, 1]$: *step-size*
- $\varepsilon > 0$ pequeño

Inicializar $Q(s, a)\ \forall\ s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ arbitrariamente, excepto $Q(terminal, \cdot) = 0$
para cada *episodio* **hacer**
  Inicializar S
  para cada *instante del episodio, mientras que S no sea terminal* **hacer**
    Elegir A de S utilizando una política derivada de Q
    Ejecutar acción A y observar R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\Big[R + \gamma \max_a Q(S', a) - Q(S, A)\Big]$
    $S \leftarrow S'$
  fin
fin

Expected SARSA

Consideremos ahora un algoritmo exactamente igual que *Q-learning* con la única diferencia de que en lugar del máximo de los siguientes pares estado-acción, se utilizará el **valor esperado**, teniendo en cuenta la probabilidad de elegir cada acción bajo la política actual. Resulta así la siguiente regla de actualización:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \mathbb{E}_\pi\big[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}\big] - Q(S_t, A_t)\Big] \\ &= Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \end{aligned} \tag{2.23}$$

Dado el siguiente estado S_{t+1}, este algoritmo se desplaza de forma determinista en la misma dirección en la que se espera que se desplace SARSA, de ahí su nombre. *Expected SARSA* es más complejo computacionalmente hablando que SARSA pero, a cambio, elimina la varianza debida a la selección aleatoria de A_{t+1}. Ante una misma experiencia, podríamos esperar que tenga un **desempeño ligeramente superior** a SARSA y, efectivamente, es así [11].

Métodos no tabulares. Aproximación de funciones

En la sección anterior se han expuesto varios métodos clásicos del aprendizaje por refuerzo. Todos ellos pueden clasificarse como métodos tabulares (*tabular methods*), ya que basan su funcionamiento en una tabla donde almacenan el resultado de la función estado-valor para cada estado (*lookup table*). Teniendo en cuenta la gran dimensionalidad que adquieren los problemas reales, la aplicación práctica de estos **métodos tabulares** resulta inviable. Aparece de esta manera un gran **problema de generalización**.

Ante dicho obstáculo, surgen los **métodos basados en aproximación de funciones**, que tratan de estimar funciones estado-valor a partir de datos *on-policy*. Esto es, aproximar la función v_π a partir de experiencia generada usando una política π conocida. Ahora, Q no se representa como una tabla sino con una **función parametrizada con un vector de pesos** $w \in \mathbb{R}^d$. Típicamente, el número de pesos (dimensión de w) es mucho menor que el número de estados. En consecuencia, cuando al visitar un estado se modifica un peso, afecta al valor de otros muchos estados. Esta generalización mejora potencialmente el aprendizaje, aunque también hace que sea más difícil de controlar y entender. Por otro lado, permite aplicar RL a problemas parcialmente observables: si la función parametrizada no permite que el valor estimado dependa de ciertos aspectos del estado, a efectos prácticos es como si esos aspectos no fuesen observables.

Cabe añadir que los algoritmos anteriormente presentados tienen adaptaciones al paradigma de aproximación de funciones, implementando técnicas como gradiente descendente estocástico, entre otras.

2.4.2. Aprendizaje por refuerzo profundo

Al tratar los problemas desde una perspectiva de aproximación de funciones, aparece la posibilidad de integrar técnicas de aprendizaje profundo (*deep learning*) en el ámbito del RL, surgiendo así el relativamente novedoso concepto de **aprendizaje por refuerzo profundo** (*deep reinforcement learning*, DRL). La razón por la que se trata de un campo relativamente reciente es porque ha sido hace unos años cuando el desarrollo de arquitecturas *hardware* ha posibilitado el uso de estructuras como las redes neuronales artificiales (*artificial neural networks*, ANN) y su entrenamiento sin necesitar tiempos de ejecución excesivamente altos. Al igual que ha ocurrido con los aprendizajes supervisado y no supervisado, muchos investigadores vuelcan ahora sus esfuerzos en diseñar nuevos algoritmos que aprovechen el potencial de estas técnicas, logrando resultados que no podían esperarse al utilizar los métodos clásicos.

En esta sección se detallan algunos de los algoritmos que constituyen el estado del arte a día de hoy, y que han sido utilizados para llevar a cabo este proyecto.

DQN

Deep Q-Network (abreviado como DQN) da nombre a la primera aproximación del DRL. Surge en 2015, cuando Mnih *et al.* tratan de aplicar las ventajas que ofrece el uso de redes neuronales artificiales al aprendizaje por refuerzo [18]. Para problemas de dimensión muy baja, los métodos tabulares clásicos constituyen una buena solución. Sin embargo, cuando el número de acciones o estados crece, el tamaño de las tablas que utilizan estos métodos aumenta de forma desmedida, por lo que su utilización se vuelve inviable. Es aquí donde intervienen las redes neuronales, capaces de aproximar funciones no lineales sin requerir, en numerosas ocasiones, demasiados recursos de tiempo y espacio. Más formalmente, se utiliza una **red neuronal** para aproximar la función óptima de acción-valor:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\big[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t = s, a_t = a, \pi\big] \tag{2.24}$$

Dado que la función $Q^*(s,a)$ es desconocida, DQN se basa en **Q-learning** para aproximarla. Se hace uso entonces de la ecuación de Bellman y queda:

$$Q^*(s,a) = \mathbb{E}\big[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\big] \tag{2.25}$$

Por otro lado, la función de pérdida que se utiliza es **MSE** (*mean squared error*), aplicado a dicha ecuación de Bellman. Esta permite entrenar la red neuronal reduciendo el error en cada iteración. Además, los valores objetivo de $r+\gamma \max_{a'} Q^*(s',a')$ se sustituyen por una aproximación utilizando pesos de la red de iteraciones anteriores. Podemos definir entonces la **función de pérdida** $L_i(\Sigma_i)$ como:

$$\begin{aligned} L_i(\theta_i) &= \mathbb{E}_{s,a,r}\big[(\mathbb{E}_{s'}[y|s,a] - Q(s,a;\theta_i))^2\big] \\ &= \mathbb{E}_{s,a,r,s'}\big[(y - Q(s,a;\theta_i))^2\big] + \mathbb{E}_{s,a,r}[\mathbb{V}_{s'}[y]] \end{aligned} \tag{2.26}$$

El método utilizado para aproximar los pesos de la red neuronal es el **gradiente descendente estocástico**. Así, derivando la función de pérdida respecto a los pesos, tenemos que el gradiente es:

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right] \tag{2.27}$$

Además, este algoritmo utiliza dos redes neuronales de forma simultánea: una **red de predicción** o red principal (*prediction network*) y otra **red objetivo** (*target network*). La primera se utilizará para calcular la función de valor del estado y acción actuales ($Q(s, a; \theta_i)$), mientras que la segunda servirá para estimar el valor objetivo. Este problema surge porque, si se utiliza una única red, tanto el valor de Q que se predice como el objetivo se desplazan en la misma dirección (indicada por el gradiente). Sin embargo, al utilizar dos redes distintas, se pueden congelar los pesos de la red objetivo y actualizarse cada n iteraciones, haciendo que las estimaciones calculadas por la red objetivo sean más precisas.

Cabe mencionar también que DQN aplica la técnica conocida como ***experience replay*** [19]. Consiste en almacenar información sobre experiencias anteriores y utilizarlas para la predicción de nuevos valores. Por otro lado, se trata de un algoritmo que sólo suele utilizarse en problemas con **espacios de acciones discretos**. Esto se debe a que en espacios continuos no se puede seleccionar una acción óptima (existen infinitas) con la que calcular el error. Hay adaptaciones que aplican técnicas como la discretización, aunque no son usuales.

A continuación se presenta el pseudocódigo del algoritmo:

Algoritmo 8: Deep Q-Network (DQN)

Entrada: - C: frecuencia de sincronización de la red objetivo

---

Inicializar memoria de repetición D
Inicializar la función acción-valor Q con pesos aleatorios θ
Inicializar la función acción-valor objetivo $\hat{Q}$ con pesos aleatorios $\theta^- = \theta$
para cada *episodio* **hacer**
  Inicializar la secuencia $s_1 = x_1$ y la secuencia preprocesada $\phi_1 = \phi(s_1)$
  para cada *timestep* **hacer**
    Seleccionar a_t con criterio ε-greedy bajo $Q(\phi(s_t), a; \theta)$
    Ejecutar acción a_t y observar recompensa r_t
    Establecer $s_{t+1} = s_t, a_t$ y preprocesar $\phi_{t+1} = \phi(s_{t+1})$
    Almacenar transición $(\phi_t, a_t, r_t, \phi_{t+1})$ en D
    Seleccionar un minibatch aleatorio de transiciones de D
    Establecer $y_j = r_j$ si el episodio termina en j+1 o $y_j = r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-)$ en otro caso
    Aplicar el paso de descenso de gradiente en $(y_j - Q(\phi_j, a_j; \theta))^2$
    Actualizar $\hat{Q} = Q$ cada C pasos
  fin
fin

Por último, cabe mencionar que existen algunas de las variantes del algoritmo DQN como pueden ser *Double Q-learning* [20], *Dueling DQN* [21], *Multi-step learning DQN* [22] o *Prioritized replay* [23].

A2C

A2C es la forma abreviada de llamar al algoritmo ***Advantage Actor Critic***. Como su nombre indica, se trata de un método con carácter *actor-critic*, los cuales se basan en el uso de dos redes (*actor* y *critic*) que trabajan de forma simultánea para resolver un problema. La red *actor* elige una acción en cada momento, mientras que la red *critic* evalúa qué estados son mejores y peores. Así, la primera red utiliza esta información para enseñar al agente a explotar los buenos estados y evitar los peores. En otras palabras, la red *actor* se encarga de aprender una política y la red *critic* de evaluarla, con el objetivo de mejorarla.

A2C tiene una **arquitectura *actor-critic*** como la que se muestra en la Fig. 2.6.

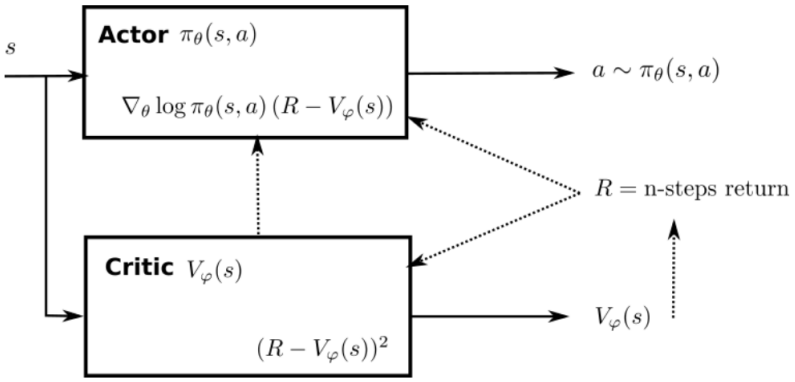


Figura 2.6: Esquema de estructura actor-critic de A2C. Imagen extraída de https://julien-vitay.net/deeprl/ActorCritic.html.

Donde *actor* devuelve una política π_θ para un estado s, y *critic* devuelve el valor de un estado s ($V_\varphi(s)$).

Aparece además la **función ventaja** (*advantage function*), que indica si un estado (o acción) es mejor o peor de lo esperado. En caso de ser mejor, el *actor* tenderá a tomar dicha acción con más frecuencia, y viceversa. En A2C, la función de ventaja es igual al *TD Error*, pudiendo definirse como:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_\varphi(s,a)] \tag{2.28}$$

Donde $A_\varphi(s,a)$ es la estimación de la ventaja (*advantage estimate*) y debe aproximar la ventaja real. Para su cálculo, se pueden usar métodos

como Monte Carlo o diferencia temporal, aunque el más común es *n-step advantage estimate*, que puede entenderse como una combinación de los anteriores:

$$\begin{aligned} A_\varphi(s,a) &= R(s_t,a_t) - V_\varphi(s_t) \\ &= \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\varphi(s_{t+n+1}) - V_\varphi(s_t) \end{aligned} \tag{2.29}$$

Definidos estos conceptos, se puede formular el algoritmo como sigue:

Algoritmo 9: Advantage Actor-Critic (A2C)

Inicializar *actor* π_θ y *critic* V_φ con pesos aleatorios
Observar el estado inicial s_0
para cada *instante de tiempo* t **hacer**
  Inicializar un minibatch vacío
  para cada $k \in [0,n]$ *siendo* k *un episodio* **hacer**
    Seleccionar una acción a_k usando el *actor* π_θ
    Ejecutar la acción a_k y observar el siguiente estado s_{k+1} y recompensa r_{k+1}
    Almacenar $(s_k, a_k, r_{k+1}$ en el minibatch del episodio)
  fin
  si s_n *no es terminal* **entonces**
    $R = V_\varphi(s_n)$
  en otro caso
    $R = 0$
  fin
  Restablecer el gradiente $d\theta$ y $d\varphi$ a 0
  para cada $k \in [n-1, 0]$ *(Se itera hacia atrás en el episodio)* **hacer**
    Se actualiza la suma con descuento de recompensas: $R = r_k + \gamma R$
    Se acumula el gradiente de la política: $d\theta \leftarrow d\theta + \nabla_\theta \log \pi_\theta(s_k, a_k)(R - V_\varphi(s_k))$ Se acumula el gradiente del *critic*: $d\varphi \leftarrow d\varphi + \nabla_\varphi (R - V_\varphi(s_k))^2$
  fin
  Se actualizan *actor* y *critic* aplicando gradiente descendente sobre los acumulados: $\theta \leftarrow \theta + \eta\, d\theta \quad \varphi \leftarrow \varphi + \eta\, d\varphi$
fin

Otro aspecto relevante es que **A2C desciende de A3C** (*asynchronous advantage actor-critic*) [24], constituyendo una versión síncrona de este. Ambos siguen un funcionamiento similar, donde varios nodos o *workers* se entre-

nan de forma paralela e independiente en entornos idénticos. Todos parten de una misma función de valor y política, y se sincronizan con una frecuencia determinada. Es precisamente ahí donde se encuentra la principal diferencia entre las dos versiones que se mencionan[19]:

- En **A2C**, cada nodo calcula un gradiente y, cuando todos han terminado su entrenamiento, la red global se actualiza de forma síncrona con la suma de todos ellos. En la siguiente iteración, cada *worker* parte de la red global actualizada. Así, es un único nodo el que aprende de la experiencia de todos los actores. Fig. 2.7 muestra un esquema que ilustra esta idea.

- Por el contrario, en **A3C** cada *worker* actualiza la red global de forma asíncrona e independiente respecto al resto. Esta estrategia no evita que cada nodo pueda partir de una función valor y política diferentes en cada iteración, dando lugar a un entrenamiento menos robusto. Un posible esquema de A3C se muestra en Fig. 2.8.

Por esta razón, A2C aplica una mejora sobre A3C, acelerando la convergencia y mejorando los resultados como demostró de forma empírica OpenAi.

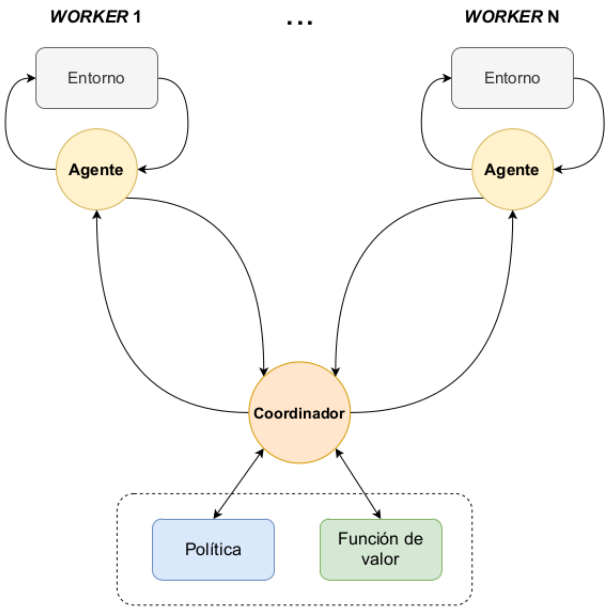


Figura 2.7: Esquema de A2C.

[19]Las figuras 2.7 y 2.8 han sido extraídas de [25].

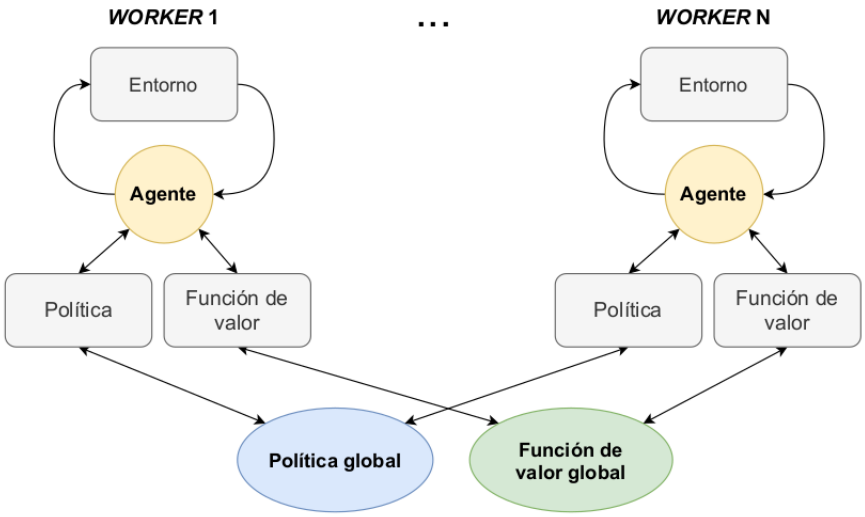


Figura 2.8: Esquema de A3C.

SAC

Soft Actor Critic (SAC) es otro de los algoritmos más destacables en DRL. Como su propio nombre indica, es un método de la familia ***actor-critic***. Se trata de un algoritmo ***off-policy*** basado en *Q-learning* y capaz de optimizar políticas estocásticas [26].

La principal característica de SAC es que utiliza una función objetivo modificada, donde introduce el concepto de **entropía**, que tratará de maximizar junto a la función de recompensa. Aunque la entropía es un término ciertamente difuso, se podría definir como cuan impredecible o aleatoria es una variable: si siempre toma el mismo valor, su entropía será nula, y viceversa. Así, la entropía consiste en una medida de aleatoriedad en el comportamiento del agente. Maximizar dicha entropía es la forma que tiene SAC de lidiar con el dilema del compromiso entre explotación y exploración. La función objetivo resultante es:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot\,|s_t))] \tag{2.30}$$

Por otro lado, SAC hace uso de **tres redes neuronales**: una para aproximar la función estado-valor V parametrizada por ψ, una para la función *soft* Q ($\mathrm{Q}(\theta)$) y otra función $\pi(\phi)$ para la política. Aunque no es estrictamente necesario utilizar aproximadores distintos para V y Q, los autores sostienen que así se consigue una mejora en la práctica [26]. Las redes se entrenan de la siguiente manera:

- Para aproximar V, se minimiza la siguiente función de pérdida:

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t|s_t)])^2\right]$$

  Y la actualización se formula:

$$\hat{\nabla}_\phi J_V(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_\theta(s_t, a_t) + \log \pi_\phi(a_t|s_t))$$

- La función Q se entrena minimizando el error:

$$J_Q(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\frac{1}{2}\Big(Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}[V_\psi(s_{t+1})])\Big)^2\right]$$

  Con la actualización:

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(s_t, a_t)(Q_\theta(s_t, a_t) - r(s_t, a_t) - \gamma V_\psi(s_{t+1}))$$

- Para la red de la política, se considera la siguiente función de error[20]:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[D_{KL}\left(\pi_\phi(\cdot\,|s_t)\middle|\middle|\frac{\exp(Q_\theta(s_t, \cdot))}{Z_\theta(s_t)}\right)\right]$$

  Siendo en este caso la actualización:

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(a_t|s_t) + (\nabla_{a_t} \log \pi_\phi(a_t|s_t) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\phi f_\phi(\epsilon_t; s_t)$$

Por último, cabe destacar que los propios autores del algoritmo realizaron distintos experimentos con varios *benchmarks*, demostrando de forma empírica que este método superaba a los que en ese momento constituían el estado del arte. Se añade para terminar el pseudocódigo de SAC:

[20] Nótese que el término D_{KL} hace referencia a la divergencia de Kullback-Leibler [27].

Algoritmo 10: Soft Actor Critic (SAC)

Entrada: - θ: parámetros iniciales de la política
- ϕ_1, ϕ_2: parámetros de la Q-función
- $\mathcal{D}$: *buffer* de repetición

Asignar a los parámetros objetivo el valor de los parámetros principales: $\theta_{obj} \leftarrow \theta, \phi_{obj,1} \leftarrow \phi_1, \phi_{obj,2} \leftarrow \phi_2$
repetir
  Observar el estado s y seleccionar la acción $a \sim \pi_\theta(\cdot|s)$
  Ejecutar a
  Observar el siguiente estado s', recompensa r, y si es estado terminal (flag d)
  Almacenar (s, a, r, s', d) en el búfer de repetición ⌈
  Si s' es terminal, se resetea el estado del entorno
  si *se debe actualizar* **entonces**
    para cada $j \in [0, \textit{número actualizaciones})$ **hacer**
      Extraer un batch de transiciones B aleatorio
      Calcular los objetivos para las Q-funciones: $y(r, s', d) = r + \gamma(1-d)\Big(min_{i=1,2} Q_{\phi_{obj,i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\Big)$
      Actualizar las Q-funciones con un paso del gradiente descendente: $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s,a) - y(r,s',d))^2$ para $i = 1, 2$
      Actualizar la política con un paso del gradiente ascendente: $\nabla_\phi \frac{1}{|B|} \sum_{s \in B} (\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s)$
      donde $\tilde{a}_\theta(s)$ es una muestra de $\pi_\theta(\cdot|s)$, que es diferenciable con respecto a θ usando el truco de la reparametrización.
      Actualizar las redes objetivo con: $\phi_{obj,i} \leftarrow \rho\phi_{obj,i} + (1-\rho)\phi_i \quad$ para $i = 1, 2$
    fin
mientras *no converja*

TD3

Para hablar de ***Twin Delayed DDPG*** (abreviado TD3), es necesario primero presentar al menos brevemente el algoritmo del que desciende: ***Deep Deterministic Policy Gradient*** (DDPG) [28]. Se trata de un método muy relacionado con *Q-learning*, que trata de aprender una Q-función y una política de forma concurrente. Para ello, se basa principalmente en

el uso de la ecuación de Bellman para función de acción-valor previamente presentada (véase 2.10). Como se puede intuir por tratarse de un algoritmo de DRL, hace uso de redes neuronales para aproximar dicha función. Adicionalmente, aplica dos trucos para mejorar el desempeño, siendo estos el ***buffer* de repetición** (*replay buffer*) y las **redes objetivo** (*target networks*).

Por su parte, uno de las principales inconvenientes de DDPG es su alta sensibilidad a los hiperparámetros. TD3 nace como un algoritmo *off-policy* para espacios de acciones continuos que trata de paliar esta carencia, manteniendo una estructura muy similar a DDPG pero añadiendo tres "trucos":

- Doble Q-learning recortado (*clipped double Q-learning*): consiste en aprender dos Q-funciones en lugar de una sola, y utilizar el Q-valor menor de ambas para formar el objetivo en la función de pérdida de Bellman.
- Actualizaciones retrasadas de la política (*delayed policy updates*). TD3 actualiza la política (y las redes objetivo) con menor frecuencia que la Q-función[21]. Esto mejora el desempeño y da lugar a un comportamiento más estable.
- Suavizado de la política objetivo (*target policy smoothing*), por el cual TD3 añade ruido a la acción objetivo. Suavizando los Q-valores, se pretende que la política no explote tanto los errores de la Q-función.

[21] En el *paper* original se recomienda una actualización de la política por cada dos actualizaciones de la Q-función.

Teniendo todo esto en cuenta, el algoritmo resultante es:

Algoritmo 11: Twin Delayed DDPG (TD3)

Entrada: - θ: parámetros iniciales de la política
- ϕ_1, ϕ_2: parámetros de la Q-función
- $\mathcal{D}$: *buffer* de repetición

Asignar a los parámetros objetivo el valor de los parámetros principales: $\theta_{obj} \leftarrow \theta, \phi_{obj,1} \leftarrow \phi_1, \phi_{obj,2} \leftarrow \phi_2$
repetir
  Observar el estado s y seleccionar la acción $a = clip(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$
  Ejecutar a
  Observar el siguiente estado s', recompensa r, y si es estado terminal (flag d)
  Almacenar (s, a, r, s', d) en el búfer de repetición ⌈
  Si s' es terminal, se resetea el estado del entorno
  si *se debe actualizar* **entonces**
    para cada $j \in [0, número\ repeticiones)$ **hacer**
      Extraer un batch de transiciones B aleatorio
      Calcular las acciones objetivo $a'(s') = clip(\mu_{\theta_{obj}}(s') + clip(\epsilon, -c, c), a_{Low}, a_{High})$
      Calcular los objetivos $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{obj,i}}(s', a'(s'))$
      Actualizar las Q-funciones con un paso del gradiente descendente: $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s, a) - y(r, s', d))^2$ para $i = 1, 2$

      si j *mod policy_delay* $== 0$ **entonces**
        Actualizar l política con un paso del gradiente ascendente usando: $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$
        Actualizar las redes objetivo con:
        $\phi_{obj,i} \leftarrow \rho\phi_{obj,i} + (1 - \rho)\phi_i$ para $i = 1, 2$
        $\theta_{obj} \leftarrow \rho\theta_{obj} + (1 - \rho)\theta$
    fin
mientras *no converja*

PPO

Proximal Policy Optimization, o simplemente PPO, es un método basado en el gradiente de la política que surge con la motivación de crear un algoritmo con la eficiencia y eficacia de TRPO usando sólo optimización de primer orden [29]. Se trata de un algoritmo con aprendizaje ***online*** y

on policy. Recordemos que este último aspecto indica que PPO evalúa y mejora la misma política que sigue durante su entrenamiento, a diferencia de los *off policy*, que mantienen dos políticas distintas para cada tarea. Esta característica simplifica mucho la estructura del algoritmo, además de reducir notablemente los tiempos necesarios para el entrenamiento del agente. Sin embargo, suelen ofrecer peores resultados que los métodos *off policy*, aunque no siempre es así.

Su funcionamiento consiste en acumular experiencia en lotes (*batches*), optimizar la política con dicha experiencia y repetir estos dos pasos. Esta estrategia produce cierta divergencia entre la política original y la actualizada. Se propone utilizar *Trust Region Policy Optimization* (TRPO) y el coeficiente de divergencia de Kullback-Leiber (KL) para solventar esta carencia. En pocas palabras, lo que esto supone es limitar las actualizaciones de la política para que estas se realicen dentro de lo que se conoce como "**región de confianza**"[22][30].

La función de pérdida que trata de minimizar PPO se define como:

$$L_{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{2.31}$$

Donde $\hat{A}_t$ es la función acción-ventaja, que trata de aproximar la bondad de una acción según la recompensa esperable al tomar dicha acción y seguir la política π. Siendo s y a el estado y acción del instante t, respectivamente, se formula la función acción-ventaja como:

$$a_\pi(s,a) = q_\pi(s,a) - v_\pi(s) \tag{2.32}$$

Por otro lado, la función objetivo de TRPO es:

$$J_{TRPO}(\theta) = \mathbb{E}[r(\theta)\hat{A}_{\theta_{old}}(s,a)] \tag{2.33}$$

donde $r(\theta)$ es el ratio:

$$r(\theta) = \frac{\pi_{\theta_{old}}(a|s)}{\pi_\theta(a|s)} \tag{2.34}$$

Tal y como se sugería anteriormente, se trata de limitar las actualizaciones de la política con el objetivo de que esta permanezca dentro de una región de confianza. Para ello se establece una desviación máxima ϵ aplicada al ratio, resultando la función objetivo definitiva que utiliza el algoritmo PPO:

$$J_{CLIP}(\theta) = \mathbb{E}\Big[\min(r(\theta)\hat{A}_{\theta_{old}}(s,a), clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{old}}(s,a)\Big] \tag{2.35}$$

[22] Formulación adaptada de [25].

Cabe aclarar que la función *clip* trunca el valor del primer parámetro al rango indicado por los dos siguientes. Al tomar PPO el valor mínimo entre ambos, reduce el problema de la divergencia.

2.5. Formulación del problema

En este apartado, describiremos los elementos necesarios para formular un problema de Aprendizaje por Refuerzo. De forma paralela, determinaremos cuáles son estos elementos para el problema concreto que enfrenta este trabajo: aprender un controlador automático para control de consumo energético en edificios.

Por un lado, existe un **conjunto de estados** determinado. Cada estado de dicho conjunto estará caracterizado por las distintas variables que se miden en el entorno. En nuestro caso, tendremos varias medidas de temperatura, humedad, velocidad y dirección del viento, y ocupación del edificio, entre otras. Todas ellas se definen con mayor exactitud en el apartado "Variables" de la sección 5.2.2. De esta manera, podemos apreciar que se trata de un espacio de estados continuo, ya que algunas (de hecho, la mayoría) de las variables que lo definen son continuas.

Por otro lado, se debe formalizar el **conjunto de acciones** que el sistema puede ejecutar para tratar de resolver el problema planteado. En el problema que nos concierne, las acciones realizables por el agente consisten en ajustar los denominados ***setpoints***. Se trata de un límite numérico que define la temperatura a partir de la cual debe activarse el sistema de climatización. Así, si la temperatura supera el *setpoint* de calor (cota superior), se activarán los sistemas de refrigeración que corresponda, para reducir la temperatura, y viceversa. Dado que se trata de límites numéricos, se debe especificar si el dominio de tales *setpoints* es continuo o discreto. Pues bien, normalmente los sistemas HVAC permiten ajustar la temperatura con una exactitud de una cifra decimal como mucho, por lo que podría considerarse un dominio continuo (dado su "elevado" número de posibles valores, aunque no sea estrictamente continuo). Sin embargo, existe la posibilidad de restringir el dominio a los números enteros, por ejemplo, convirtiéndolo así en discreto. En el desarrollo de los experimentos, se considerarán **ambos espacios de acciones**, con el objetivo de comparar el desempeño de distintos algoritmos según el espacio utilizado. Se puede intuir que los espacios de acciones discretos requieren un menor entrenamiento por presentar menos opciones. Sin embargo, la optimización de las soluciones que ofrecen es más limitada que en el caso continuo. Asimismo, se pretende estudiar si los espacios continuos efectivamente aportan una mejora en el rendimiento[23]

[23]En *machine learning* aumentar el espacio de búsqueda puede resultar contraprodu-

y si merece la pena a cambio del aumento de coste computacional requerido.

Es necesario también determinar una **función de recompensa** que valore el comportamiento del agente. El sistema debe intentar minimizar el consumo energético a la par que maximizar el confort de los ocupantes del edificio. Al tratarse de objetivos opuestos, en el sentido de que cumplir uno de ellos implica una mayor dificultad en satisfacer el otro (es decir, para maximizar el confort es necesario un mayor consumo energético, y minimizar el consumo requiere sacrificar parcial o totalmente el confort), la función de recompensa debe plantear un balance o *trade-off* entre las dos metas, para alcanzar así un equilibrio. Dicho esto, para definir la función de recompensa es necesario determinar antes cómo se cuantifican el consumo energético y el confort térmico.

El **consumo energético** es relativamente sencillo pues, al tratarse de una magnitud física, se puede cuantificar midiendo la energía consumida por los sistemas de climatización del edificio en un determinado periodo de tiempo.

Un caso distinto es el del **confort**, ya que es un concepto menos tangible. El sistema que se propone sólo interviene en la alteración de la temperatura, sin modificar otros factores como la humedad, radiación térmica o velocidad del aire, que pueden influir en el confort según el estándar [31]. Así, la única variable que consideraremos que influye en esta medida es la temperatura. Adicionalmente, cabe puntualizar que el rango de temperatura idóneo es [23°C - 26°C] en verano y [20°C - 23.5°C] en invierno [32, 31, 33]. Dicho esto, surgen varias formas de definir el confort. Por ejemplo, se puede considerar la distancia entre la temperatura actual y una temperatura objetivo, o la distancia al cuadrado, típicamente utilizada en ML. En cualquier caso, optaremos por la **distancia al rango de confort objetivo** mencionado, tal y como se propone en [34]. Queda así definida la función de recompensa como:

$$r(S_t, A_t) = w_t \cdot \lambda_c \cdot f(S_t, S_{target}) + (1 - w_t) \cdot \lambda_e \cdot coste(A_t) \tag{2.36}$$

Si concretamos la función para cuantificar el confort, nos queda:

$$r(S_t, A_t) = w_t \cdot \lambda_c \cdot \sum_{i=1}^{z} ([T_t^i - \overline{T_t^i}]_+ + [T_t^i - \underline{T_t^i}]_+) + (1 - w_t) \cdot \lambda_e \cdot coste(A_t) \tag{2.37}$$

cente en ocasiones, pues dificulta la convergencia del algoritmo si el entrenamiento no es suficiente.

Siendo w_t un parámetro con valor entre 0 y 1 que pondera la importancia del confort frente al consumo energético en lo que respecta a la función de recompensa. Por su parte, λ_c y λ_e son factores de escala para normalizar las medidas de confort y consumo de energía.

Nótese que el subíndice de w indica que el valor de este parámetro no es fijo y estático, sino que puede variar con el tiempo. Por ejemplo, puede permitir que la función de recompensa se adapte dinámicamente a la ocupación del edificio (primando el ahorro energético sobre el confort cuando no haya nadie en su interior).

Queda entonces definida la función de recompensa básica. A partir de esta se han definido algunas otras funciones adicionales para comprobar empíricamente si permiten mejorar el rendimiento del agente. Serán descritas en la subsección 5.3.4.

Algunos elementos adicionales son los siguientes:

- **Agente**: el agente es en este caso el controlador automático de los sistemas de climatización del edificio.
- **Entorno**: está constituido por todo aquello que el agente no puede cambiar de forma arbitraria. Desde la situación climatológica hasta los materiales del edificio, entre muchos otros factores externos. Para este proyecto, el entorno será simulado por la herramienta Sinergym.
- **Política**: en referencia a la política que se desea aproximar, se puede definir de acuerdo a la función de recompensa anteriormente descrita:

$$\pi^* = \arg\min_{\pi_\theta} \sum_{t=1}^{T} w_t \cdot f(S_t, S_{target}) + (1 - w_t \cdot coste(A_t)) \qquad (2.38)$$

  En otras palabras, se pretende aproximar una política que maximice la función de recompensa (minimizando así la penalización recibida) en cada decisión tomada.

- El **modelo del entorno** es desconocido en este caso. No podemos predecir con exactitud el estado s_{t+1} resultante de aplicar una acción en un estado concreto s_t. La existencia de tantos factores externos que afectan a la temperatura del edificio imposibilitan la predicción correcta y exacta de esta magnitud tras activar o desactivar los dispositivos de calefacción o refrigeración. Esto sugiere utilizar **métodos libres de modelo** (*model-free methods*)[11].

- **Factor de descuento (γ)**: el valor de este parámetro se determinará más adelante, aunque se puede adelantar que será menor que 1 para hacer que el agente aplique acciones con efecto a corto plazo y ofrezca una respuesta relativamente rápida.

Como se ha dejado ver anteriormente, se trata de un **problema continuado** (no episódico), por lo que la duración de cada simulación se limita a un **periodo de un año**.

Por último, cabe indicar que la temperatura de la estancia en el siguiente *timestep* viene determinada únicamente por el estado actual y las perturbaciones del entorno, además del efecto del sistema HVAC en este. A su vez, es independiente de los estados anteriores. Es por ello que el control del sistema de climatización puede tratarse como un proceso de decisión de Markov.

Capítulo 3

Estado del arte

En el momento de realizar la revisión literaria del tema que nos concierne, lo primero que notamos es un aumento en el número de artículos y publicaciones científicas relacionados durante los últimos años, en especial desde 2019 (véase 3.1). Esto demuestra un **creciente interés** por la aplicación de técnicas de RL y DRL en el control energético de edificios y, más concretamente, en los sistemas de climatización. Además, tras observar algunos de los resultados que se muestran en esta sección, este ámbito podría considerarse bastante prometedor.

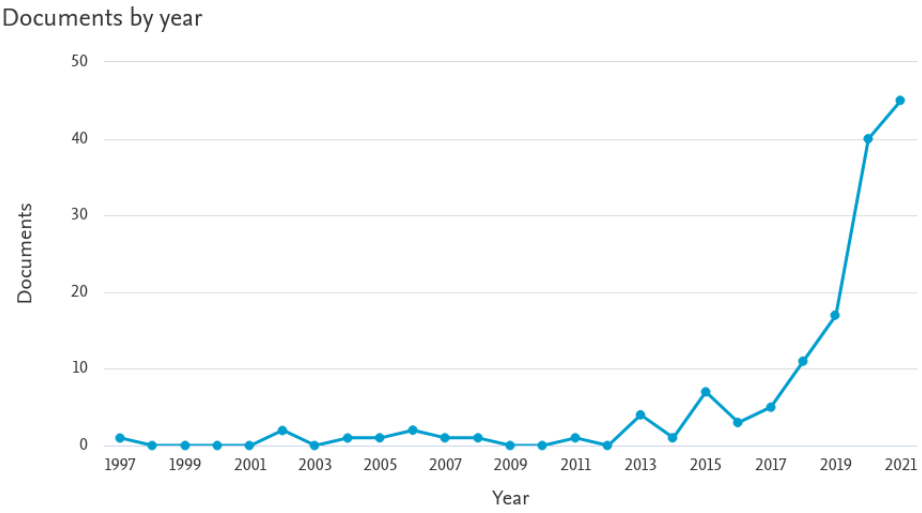


Figura 3.1: Número de artículos sobre RL (ó DRL) y HVAC en Scopus (hasta 2021).

A pesar de todo esto, la utilización del aprendizaje automático para el control energético no es una idea tan reciente. Fue en 1997 cuando W. Anderson desarrolló una **primera síntesis** del RL y redes neuronales orientados al control de sistemas de calefacción. En su estudio demostró que, aunque

el controlador PI ya tuviese un rendimiento aceptable, podía ser mejorado al combinarlo con una red neuronal [35]. Al año siguiente, Mozer publicó el diseño de su *Neural Network House*, un sistema capaz de controlar los sistemas de confort de un edificio residencial: aire acondicionado, ventilación, iluminación y temperatura del agua, constituyendo así la primera aproximación directa de DRL a control de HVAC en un edificio [36].

A partir de ahí, proliferan las publicaciones que tratan el mismo problema desde distintas perspectivas. Muchas de ellas abordan el tema tratando no sólo de minimizar el consumo energético sino también **mantener el confort de los ocupantes** de la infraestructura [37, 38, 39, 40, 41, 42, 43], entre otras.

Se desarrollan también trabajos que tratan de dar una solución en espacios de **mayor dimensionalidad**. Por ejemplo, en [44] se trata de maximizar el ahorro energético en un entorno con múltiples estancias. Autores como Simeng Liu *et al.* plantean métodos híbridos para el control de climatización en centros comerciales [45, 46]. Incluso algunos de ellos van más enfocados a las denominadas **ciudades inteligentes** (*smart cities*), utilizando el simulador *CitySim*, como es el caso de [42, 47]. Estos modelos desarrollados por J. Vázquez-Canteli *et al.* hacen uso de técnicas como *batch Q-learning*, *Q-iteration* para el control de sistemas HVAC desde la perspectiva de una ciudad.

Destacan las publicaciones de autores como Barret [48], Ruelens [43] e Urieli [40] por centrar su atención en el **ajuste de *setpoints***, enfoque que también será adoptado en este trabajo.

Paralelamente, varios investigadores centran sus esfuerzos en explorar nuevas alternativas y estudiar si alguna de ellas resulta ventajosa. De esta manera, en [49] se hace uso del método Bayesiano y **algoritmos genéticos** para la calibración del modelo. En [50, 51] se incorpora **lógica difusa**. Además, en este último Dalamagkidis supera a *Fuzzy-PD* y un controlador *on/off* tradicional tras tan solo un par de años de entrenamiento simulados. Destaca la señal de recompensa que utiliza, no solo basada en la temperatura sino también en el **nivel de CO_2**. Por otro lado, en [40, 43] se sugiere el uso de métodos basados en ***set-back*** que relajan la demanda energética cuando no hay personas en el edificio. Por su parte, [52] plantea la opción de coordinar **sistemas multiagente** que participen en la respuesta de demanda energética teniendo en cuenta los programas que definen los precios de la electricidad. Ante todas estas soluciones que en cierto modo aumentan la complejidad, [53] demuestra que con un sistema basado en una red neuronal con un bajo número de capas también se pueden obtener resultados aceptables.

Un aspecto a destacar de la mayoría de estos trabajos es que, para ser más completos, deberían realizar **pruebas de robustez**. Por ejemplo, en [37] se prueban los modelos entrenados en cuatro escenarios distintos con distintas variables meteorológicas y niveles de ocupación.

Presentemos ahora algunos de los **resultados numéricos** obtenidos en la literatura:

- En [37], Brandi consigue entre un 5 % y un 12 % de ahorro energético con un modelo bastante robusto, tal y como se ha mencionado antes.
- Al adaptarse a los horarios de ocupación del edificio, [40] logra un ahorro de entre 7 % hasta un 14.5 %.
- [54] destaca por conseguir un 22 % de ahorro semanal tras entrenar su *DR-aware RL controller* en entornos con demanda energética muy variable (alcanzando hasta un 50 % de variabilidad).
- Yuan logra reducir la demanda energética desde un 4.7 % hasta un 7.7 % al compararlo con un sistema basado en reglas (*rule based controller*, RBC) y un sistema PID [55].
- El modelo entrenado en [56] está basado en EnergyPlus, y alcanza una disminución del 16.6 % al 18.2 % respecto a un RBC.
- En [57], cuando se consideran 30 zonas distintas, el algoritmo propuesto puede reducir el coste energético entre un 56.5 % y un 75.25 %, en comparación con otros *baselines*, y manteniendo el confort de los ocupantes.
- En [58], un modelo basado en DDPG es capaz de reducir un 15 % el coste energético en comparación con el método DQN, protagonista del estado del arte en el momento de su publicación, además de reducir un 79 % las violaciones de confort en el edificio.
- Gupta *et al.* mejoran el confort termal entre un 15 % y un 30 %, y reducen la demanda energética entre un 5 % y un 12 % en el entorno simulado [59]. Demuestran además que el control descentralizado tiene un mejor desempeño que un controlador centralizado cuando crece el número de edificios y aumentan las diferencias entre los *setpoints*.

La principal pregunta que sugieren estos resultados es si son comparables entre sí, es decir, si la experimentación ha sido realizada en igualdad de condiciones, pudiendo así distinguir qué resultados son mejores o peores. La respuesta es no. Cada uno de estos modelos ha sido entrenado, evaluado e incluso comparado en las condiciones que ha decidido cada autor. Existen

diferencias entre los edificios utilizados, los entornos de simulación, las medidas de ahorro energético y violación de confort... e incluso se comparan con modelos de referencia (*baselines*) distintos, por lo que es imposible distinguir si un modelo es mejor o peor que otro. Este problema es detectado en [52], donde, al igual que en [54], se propone un *framework* para poder desarrollar y comparar distintos modelos de DRL aplicados al control de HVAC. Sin embargo, sigue siendo un aspecto sin resolver, ya que aún **no existe un entorno común** suficientemente completo, preciso y adecuado para abordar todas las pruebas que los expertos proponen. Cabe puntualizar que *Sinergym*, basándose en la arquitectura de *OpenAi Gym* y *EnergyPlus*, como Zhang *et al.* propusieron en [56, 49], tiene como objetivo crear un *framework* universal para poder desarrollar y comparar distintos modelos, tratando de ofrecer un amplio abanico de posibilidades (desde algoritmos y edificios hasta variables del entorno, entre otros aspectos, descritos con más detalle en 5.2.1). Sin embargo, se trata de un proyecto en desarrollo que, probablemente, requiera cierto tiempo hasta ser lo suficientemente completo como para ofrecer una solución que se adapte a los investigadores de este campo.

Capítulo 4

Desarrollo software

Este capítulo será dedicado a la exposición del desarrollo *software* realizado durante el periodo de elaboración de este proyecto. Primero se explica la estrategia seguida durante el desarrollo. Se continúa hablando del equipo detrás de Sinergym [60], la herramienta *software* protagonista, para después especificar las contribuciones realizadas y código implementado.

4.1. Metodología de desarrollo

4.1.1. Metodología ágil

Quizás el aspecto más destacable acerca de la metodología es precisamente la estrategia seguida durante el desarrollo del *software* de este proyecto. No obstante, primero debemos tener en cuenta que este desarrollo *software* ha sido principalmente orientado a la implementación del entorno que permita la realización del estudio planteado y la inherente ejecución de experimentos. Por tanto, no se trata del diseño y desarrollo de un producto o aplicación final como tal, sino más bien de un paso intermedio que posibilitase la investigación que este proyecto tiene como objetivo. Así pues, en este caso se ha tratado de seguir la filosofía que plantea un **desarrollo ágil**, como se propone en [61], en el sentido de minimizar el tiempo de reacción ante nuevas necesidades y optimizar la corrección de errores.

4.1.2. Integración en el equipo

Otro punto importante ha sido la integración en el equipo encargado de desarrollar Sinergym. Los miembros que conforman este equipo se listan a continuación:

- **Javier Jiménez**: creador y gestor del proyecto. También se ocupa de dirigir y contribuir a su desarrollo general.

- **Antonio Manjavacas**, **Alejandro Campoy** y **Pablo Torres**: programadores encargados de la implementación y documentación de la funcionalidad de Sinergym.
- **Miguel Molina** y **Juan Gómez**: supervisores del proyecto.

La incorporación al equipo fue relativamente rápida, dada la frecuente y constante comunicación. Las dos herramientas más relevantes en este aspecto han sido Github (donde se almacena el repositorio que contiene el código desarrollado) y Slack, que ha permitido la comunicación de forma fluida y organizada. Además, se han realizado **reuniones presenciales de forma periódica** donde cada miembro exponía el trabajo realizado.

A pesar de que, como se detalla más adelante, la implementación de *software* relativa a este proyecto ha sido realizada de forma paralela y, en parte, independiente a la línea de desarrollo principal, la integración en el equipo de trabajo ha sido fundamental para que el autor de este proyecto estuviese al tanto de los cambios, actualizaciones y corrección de errores que se han ido realizado de forma progresiva.

4.2. Desarrollo software

Llega el momento de detallar el código implementado durante este proyecto.

4.2.1. Contribuciones al desarrollo

Antes de entrar en el detalle de las contribuciones al desarrollo, resulta indispensable destacar un aspecto. Toda la funcionalidad que se ha implementado para este trabajo no ha sido incluida de forma definitiva en Sinergym, al menos en el momento en que se redacta esta memoria. Al no saber con antelación si las ideas propuestas en este trabajo ofrecerían resultados satisfactorios, se decidió desde un principio realizar un **desarrollo paralelo** que no interfiriese en la línea principal de Sinergym. Así, se consideró adecuado realizar una bifurcación (*fork*) del repositorio en la que implementar el código necesario. Más adelante, los cambios se aplicaron en una rama *experimental* del repositorio original, por razones que se explican en los apartados subsiguientes.

Antes de seguir, es necesario indicar que tanto el código implementado como los resultados obtenidos se encuentran en el repositorio de Github Ahmed2BP/TFG-GII. Se trata de un repositorio público, para que el lector pueda explorarlos si lo desea. Nótese que en este sólo se han incluido los modelos y resúmenes de resultados, ya que subir todos los archivos generados

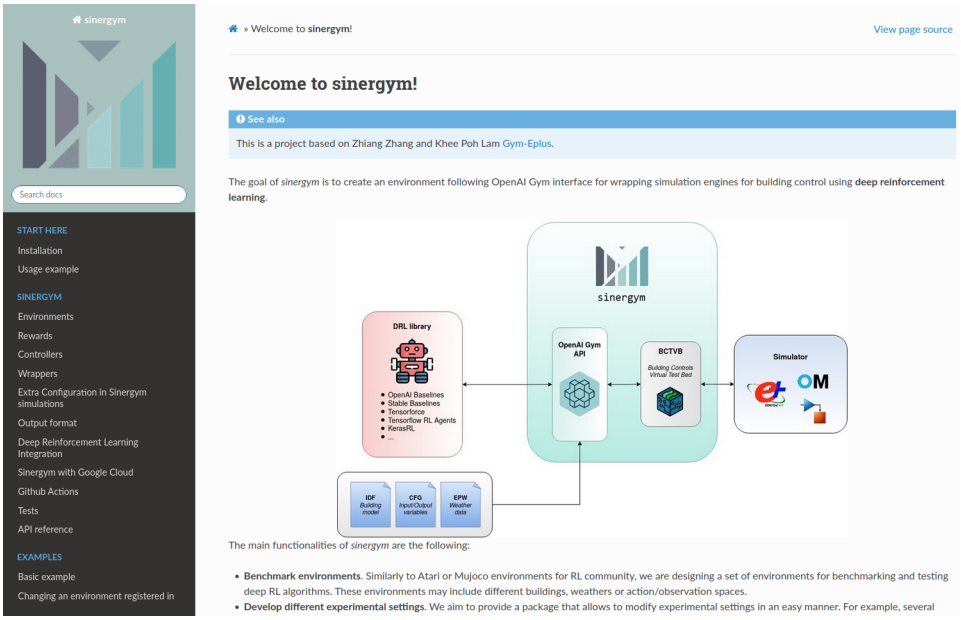


Figura 4.1: Página principal de la documentación de Sinerygm.

no es posible por limitaciones de espacio, además de no ser de gran interés.

Podríamos distinguir dos fases principales en el desarrollo, que serán descritas a continuación.

4.2.2. Desarrollo de entorno local

Antes de comenzar con la implementación, fue necesario un **análisis profundo del código** de Sinergym desarrollado hasta el momento. Esto supuso un primer contacto y permitió conocer tanto la estructura como las funcionalidades del código. A continuación, se ha desarrollado el código que permitía entrenar y evaluar los agentes previamente propuestos. Esto incluye los distintos ***scripts*** que modelan cada agente con unos parámetros determinados. Una vez implementados, se realizaron **pruebas unitarias** de forma local, lo que permitió la **detección de errores** y familiarización con el entorno y su funcionamiento. También se programó el código para automatizar la ejecución de la batería de experimentos con *scripts* sencillos. Finalmente, se ejecutaron algunos experimentos y se advirtió de la inviabilidad de realizar las pruebas de forma local. La respuesta frente a este obstáculo se describe en el siguiente capítulo.

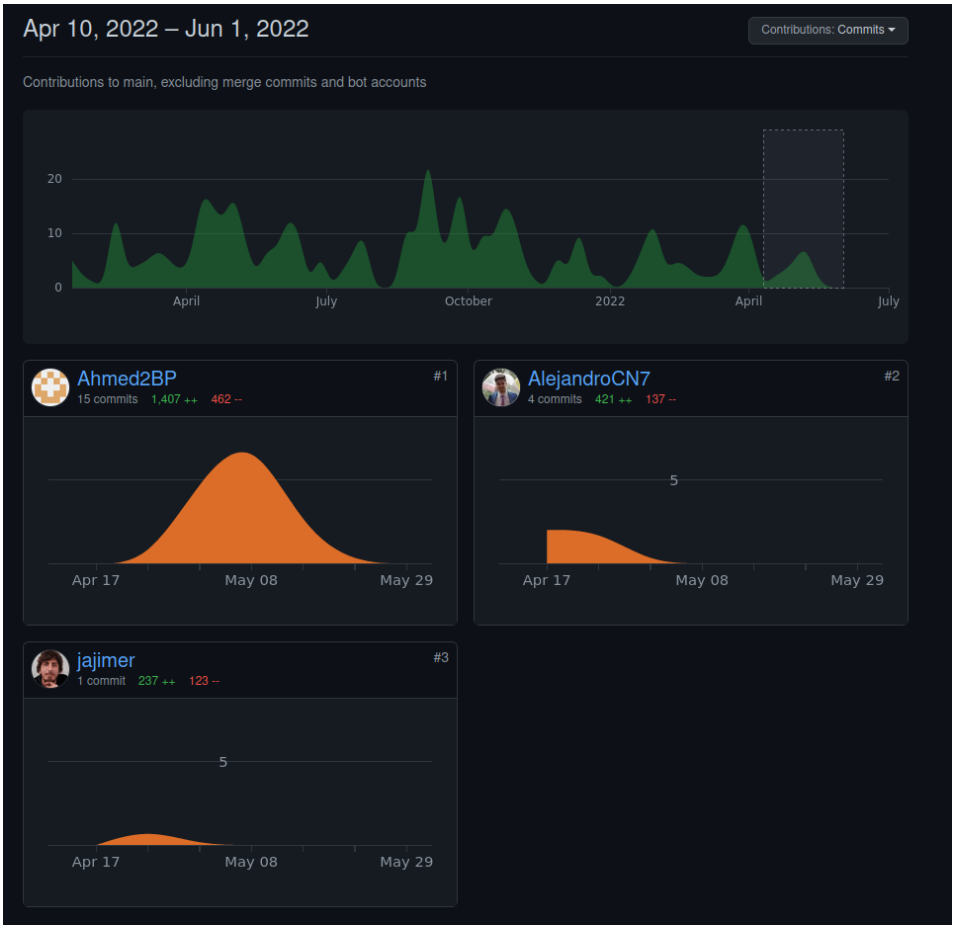


Figura 4.2: Contribuciones registradas del repositorio bifurcado. Extraído de https://github.com/Ahmed2BP/sinergym/graphs/contributors?from=2022-04-10&to=2022-06-01&type=c.

El código desarrollado referente a este apartado se encuentra en la carpeta "sinergym-local" del repositorio anterior. En ella, podemos encontrar la implementación de una función de recompensa personalizada, el código para entrenar distintos agentes, junto a un *script* para su automatización, y los resultados de los entrenamientos de algunos de ellos, además del resto del entorno necesario para su ejecución.

4.2.3. Migración para ejecución remota

En el momento en que se encontró que no era factible la ejecución de experimentos de forma local, se decidió que se realizarían en un **entorno remoto de cómputo masivo**. Así, siguiendo la metodología de desarrollo ágil antes mencionada, se abandonó la línea de desarrollo principal hasta ese momento para reaccionar ante este contratiempo.

En definitiva, hubo que **adaptar el código** que modelaba los agentes para que funcionase en un subproyecto privado que permite utilizar Sinergym con recursos remotos de Google Cloud Platform: **SinergymCloud**. Sin embargo, ya existía la posibilidad de confeccionar los modelos en este entorno, por lo que sólo fue necesaria la **implementación de *scripts*** que permitiesen ejecutar la batería de experimentos.

Adicionalmente, fue necesario incluir en el repositorio de Github el código para definir las distintas funciones de recompensa, así como los espacios de acciones utilizados.

Por último, se **automatizó el preprocesado** de los resultados obtenidos para facilitar su posterior análisis. En este, se seleccionan las métricas de evaluación que se desean considerar, y se calculan su media y desviación típica.

En este caso, las carpetas del repositorio correspondientes son "sinergym-remote" y "sinergym-cloud". En la primera se encuentra el entorno en el estado en que se realizó la experimentación, mientras que la segunda almacena los *scripts* utilizados para lanzar los experimentos con SinergymCloud, cuyo contenido no será publicado por tratarse de un repositorio privado.

Capítulo 5

Experimentación

Este quinto capítulo se dedica a describir el desarrollo de la experimentación realizada. Primero, se indica qué recursos *hardware* han sido utilizados. A continuación, se presenta el entorno empleado para la realización de las pruebas, indicando sus aspectos más relevantes, para coronar el capítulo con una descripción de los experimentos propuestos.

5.1. Recursos y equipo utilizado

Antes de entrar en mayor detalle acerca de la experimentación realizada en el proyecto, es importante concretar la **componente *hardware*** que ha permitido la ejecución de dichas pruebas. El *hardware* utilizado para la realización de pruebas "unitarias", así como para la familiarización con el entorno de Sinergym, ha sido un ordenador *HP OMEN 15-DC0xxx*. Este dispositivo consta de un procesador *Intel Core i7-8750 CPU @ 2.20GHz*, 12 GB de RAM, una tarjeta gráfica *Intel UHD Graphics 630* y otra GPU adicional *Nvidia GeForce GTX 1050*.

En vista de la alta carga computacional que requiere la ejecución de los experimentos (en especial el entrenamiento de los agentes), se ha optado por alternativas de ***cloud computing*** para la realizar la experimentación completa, eligiendo los servicios que presenta *Google Cloud Platform*. Concretamente, se han empleado las máquinas de Google Cloud *n2-highcpu-8*, equipadas con un procesador *Intel Cascade Lake* de 2.2 GHz, 8 vCPU y 8192 MiB de RAM[1]. De este modo, se han podido realizar las distintas ejecuciones de forma paralela en máquinas remotas (con un máximo de 7 instancias ejecutándose concurrentemente). La decisión de tomar recursos remotos se debe a la inviabilidad de realizar las pruebas en local, por sus extendidos tiempos de ejecución requeridos.

[1]Más información en https://cloud.google.com/compute/docs/machine-types.

5.2. Sinergym. Descripción del entorno

A la hora de abordar la realización de los experimentos, el *software* de Sinergym [60] ha resultado ser una herramienta fundamental. Sinergym (anteriormente conocido como Energym) da nombre a un proyecto de código abierto en continuo desarrollo, cuyo objetivo principal es ofrecer al usuario un entorno virtual sobre el que realizar simulaciones enfocadas al ámbito del control energético en edificios mediante RL y *Deep Reinforcement Learning* (DRL). Este *software* está basado en Gym Eplus, de Zhiang Zhang y Khee Poh Lam. Además, se ha hecho uso de BCVTB para la interacción entre Sinergym y EnergyPlus.

Como se ha indicado, el propósito de este proyecto es construir un **entorno virtual** suficientemente fiel a la realidad, sobre el que realizar experimentos (tanto entrenamiento como testeo de agentes) de RL y evitar así la necesidad de utilizar entornos reales, dado el coste económico y temporal que esto supone. Para ello, durante su desarrollo se ha tratado de cumplir con la guía sobre el diseño de entornos de *benchmarking* para optimización energética de edificios que presentan Wölfle et al. [62]. Dos de los principales problemas que se plantean en dicho artículo son la falta de generalización de las soluciones publicadas en la literatura, que únicamente consideran un edificio o entorno concreto en su entrenamiento, y la poca adaptabilidad que tienen de ser empleados en otros entornos. Así, Sinergym trata de solventar estas cuestiones incluyendo varios (y cada vez más) edificios donde pueden entrenarse los agentes. Además, una vez entrenado, podría ser evaluado en distintos escenarios y climas, independientemente de las condiciones de su entrenamiento.

Por último, cabe recordar la necesidad de utilizar un entorno virtual (donde simular los entrenamientos y evaluaciones) en lugar de uno real, dados las restricciones temporales y económicas que esto implicaría. Se necesitarían entrenamientos de varios años para la convergencia de los algoritmos como se verá más adelante, además de la cuestión económica, que también excede el alcance de un trabajo de fin de grado.

5.2.1. Configuración del entorno

Para cumplir dicho objetivo, Sinergym presenta un amplio abanico de posibilidades en lo que a configuración se refiere. Entre ellas figuran las siguientes:

- **Observaciones**: el usuario puede determinar qué factores y variables del entorno detecta el agente, entre todas las posibles. Es útil para mejorar la eficiencia en caso de no considerar relevantes ciertas características del medio.

- **Espacio de acciones**: se permite al programador modificar el espacio de acciones (*setpoints* disponibles), pudiendo así elegir entre un espacio de acciones continuo o discreto, por ejemplo.

- **Recompensas**. La recompensa del problema también es un parámetro ajustable. El entorno ya ofrece tres ejemplos de funciones de recompensa básicas (*LinReward*, *ExpReward* y *HourlyLinReward*). La posibilidad de modificar la función de recompensa será explotada en el apartado referente a los experimentos, por lo que ha sido fundamental para su desarrollo.

- **Agentes o controladores**. Adicionalmente, se puede modificar el agente de RL y, con este, el algoritmo de aprendizaje utilizado. Asimismo, permite el uso de sistemas basados en reglas (RBCs), que pueden ser usados como *baseline* para comparar el desempeño de distintos algoritmos.

- **Configuración adicional en la simulación** de detalles menores, como pueden ser el número de *timesteps* por hora o la duración de cada periodo.

5.2.2. Escenarios y variables

Por su parte, existe una gran variabilidad en los distintos escenarios posibles para las simulaciones. Principalmente, hay tres componentes que no se deben pasar por alto.

Climas

La variabilidad meteorológica es otro elemento que Sinergym permite. Por defecto, se incluyen tres tipos de clima para cada edificio:

- **Clima cálido**. Un clima cálido y seco, correspondiente a Arizona, donde hay una temperatura media anual de 21.7°C y una humedad relativa media anual del 34.9 %.

- **Clima templado**. Simula una zona de Nueva York, con una temperatura media de 12.6°C y una humedad relativa del 68.5 %.

- **Clima fresco**. Se trata de un clima marítimo con una temperatura media de 9.3°C y un 81.1 % de humedad relativa. Se basa en el historial meteorológico de un aeropuerto de Washington[2].

[2] Al igual que en los dos casos anteriores, estos datos han sido recabados de bases de datos reales, con la información que han registrado de distintas zonas durante varios años [60].

Además, para cada clima, existen los casos determinista y estocástico. En este último, se inserta un **ruido** en la dinámica del entorno[3] de media 0 y desviación típica 2.5.

Como es lógico, la componente térmica del clima tiene un alto impacto en las estrategias de control de sistemas de climatización [63]. En este sentido, Sinergym resulta ser una herramienta ciertamente completa, pues integra distintos climas para el entrenamiento de los agentes, evitando así el sesgo que estos podrían adquirir si son entrenados siempre en un mismo clima.

Por último, durante la realización de los experimentos se han utilizado los **tres climas** para cada modelo generado, en aras de realizar un estudio lo más completo posible.

Edificios

Por un lado, existen 3 edificios diferentes (aunque se pueden añadir más siempre y cuando sean descritos correctamente).

- **Edificio de 5 zonas**. Se trata de un edificio de una planta de 463.6m^2 con 4 zonas exteriores y una zona interior. Sólo en esta última se controla la temperatura mediante un sistema de volumen de aire variable (VAV), del que deben ajustarse los *setpoints*. El periodo de simulación es de un año completo.
- **Centro de procesamiento de datos** (o *data center)*. Un área de 491.3m^2 dividida en dos zonas asimétricas, cuyo calor interno es generado principalmente por los servidores. Cada zona dispone de economizadores de aire, enfriadores por evaporación, ventiloconvectores, serpentines de enfriamiento y sistemas de volumen de aire variable, que funcionan como un sistema de climatización. Así, la tarea consiste en el ajuste de *setpoints* para ambas zonas durante un periodo de un año.
- Una **zona de trabajo** de 600m^2 con aulas y oficinas, con un sistema de radiadores de agua. En este caso, se debe ajustar la temperatura del agua del sistema de calefacción, durante 3 meses.

El edificio utilizado ha sido el primero (**5 zonas**). Se ha considerado el más adecuado ya que, por un lado, hay mayor flujo de personas que en el *data center*, donde las restricciones de temperatura no están enfocadas al confort de los ocupantes sino al mantenimiento y correcto funcionamiento de los servidores. Por otro lado, se ha descartado la zona de trabajo dado

[3]Se incluye una semilla para permitir la reproducibilidad de las ejecuciones.

que el periodo de simulación es demasiado breve (3 meses). Además, el edificio de 5 zonas es idóneo para evaluar el comportamiento de las funciones de recompensa que se plantean en este trabajo, pues la ocupación varía suficientemente con el paso del tiempo, y no presenta inconvenientes de otro tipo.

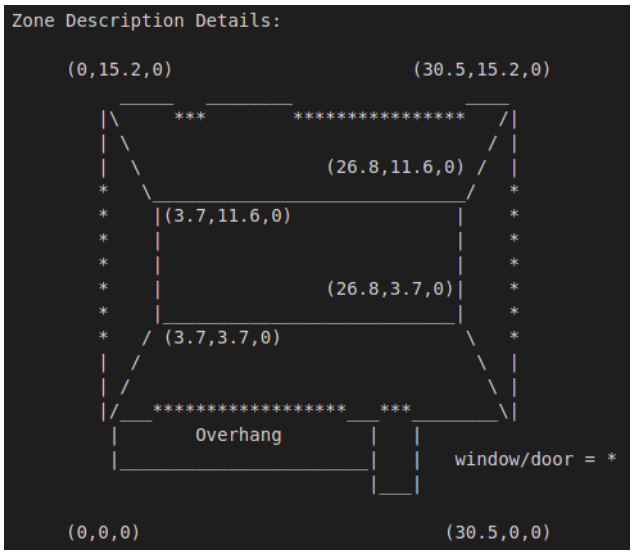


Figura 5.1: Esquema del edificio de 5 zonas. Extraído del fichero *.idf*, que lo modela.

Al igual que ocurría con la temperatura, las características del edificio (tamaño, materiales, etc) también afectan al control de sistemas de climatización más de lo que uno podría pensar [63]. De nuevo, Sinergym se convierte en una herramienta de *benchmarking* bastante adecuada por ofrecer distintos edificios para la simulación.

Variables

Llega el momento de describir las variables que intervienen en el proceso de simulación. Cabe distinguir dos tipos: variables de entrada, o *inputs* (las que el agente puede modificar), y de salida, o *outputs* (fijadas por el entorno de simulación en este caso, y observadas por el agente para obtener información del estado).

Las **variables de entrada** que presenta Sinergym son los *setpoints* de calefacción y de refrigeración de los sistemas de climatización de cada edificio[4]. El agente debe aprender a ajustarlos de manera óptima para cumplir

[4]Sin embargo, está en desarrollo la integración de nuevos edificios con más parámetros de entrada ajustables.

con su objetivo. Para ello, en cada observación deberá almacenar información de las **variables de salida** que mantiene el entorno y tomar una decisión en función de dicho *output*. Estas son:

- **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*).
- **Tasa de radiación solar difusa por área** (*Site Diffuse Solar Radiation Rate per Area*).
- **Tasa de radiación solar directa por área** (*Site Direct Solar Radiation Rate per Area*).
- **Temperatura de activación (*setpint*) para calefacción** (*Zone Thermostat Heating Setpoint Temperature*).
- **Temperatura de activación (*setpint*) para refrigeración** (*Zone Thermostat Cooling Setpoint Temperature*).
- **Temperatura del aire** (*Zone Air Temperature*).
- **Confort térmico según la temperatura radiante media** (*Zone Thermal Comfort Mean Radiant Temperature*).
- **Humedad relativa del aire** (*Zone Air Relative Humidity*).
- **Valor del confort térmico de acuerdo al ropaje** (*Zone Thermal Comfort Clothing Value*).
- **Confort térmico de acuerdo al modelo Fanger PDD** (*Zone Thermal Comfort Fanger Model PPD*).
- **Número de ocupantes de la zona** (*Zone People Occupant Count*).
- **Confort térmico de acuerdo al modelo Fanger** (*People Air Temperature*).
- **Demanda energética total del sistema de climatización** (*Facility Total HVAC Electricity Demand Rate*).
- **Año actual** (*Current year*).
- **Mes actual** (*Current month*).

- **Día actual** (*Current day*).
- **Hora actual** (*Current day*).

Cabe puntualizar que las anteriores son las variables con las que Sinergym permite trabajar. Sin embargo, el usuario puede no considerar todas ellas, ignorando las que no crea relevantes o necesarias. En cualquier caso, para las pruebas realizadas **se han incluido todas**, para dotar al modelo de la máxima información posible y no omitir así ninguna variable que pudiese resultar importante para el control.

5.3. Experimentos propuestos

En este apartado se describen los experimentos que han sido llevados a cabo como parte del proyecto. Todo ello sin perder de vista el principal objetivo de este trabajo, que radica en la modelización y optimización de un agente capaz de controlar el sistema de climatización de un edificio de forma eficiente. Aunque el principal interés de la experimentación realizada en este trabajo gira en torno al planteamiento y estudio de nuevas funciones de recompensa, es necesario definir también otros aspectos relevantes como son los algoritmos, espacios de acciones y métricas de evaluación a utilizar, entre otros.

5.3.1. Condiciones de entrenamiento y validación

Comencemos con algunos detalles de las pruebas realizadas, detallando las condiciones bajo las que se han realizado los entrenamientos de los agentes, así como su validación:

- La **duración del entrenamiento** es de 20 episodios, ya que se ha demostrado de forma empírica que esta cantidad es más que suficiente para la convergencia del aprendizaje, tal y como se muestra en 5.2[5][6].
- La selección de **hiperparámetros** se ha realizado en base a [25, 60], considerando también los que se proponen en la documentación de Sinergym. No se ha incluido una fase de *fine tuning* ya que se sale del ámbito de este trabajo, además de ser inviable por cuestiones temporales y económicas.
- El **número de episodios de validación** es 5 pues, de nuevo, son suficientes para mostrar un promedio representativo del resultado, ya que como se puede observar en las tablas mostradas más adelante,

[5]Esta imagen está recortada para permitir una buena visualización.

[6]Cada episodio tiene una duración de 35040 *timesteps*. Así, se observa que todos ellos convergen antes incluso de los 5 episodios (175200 *timesteps*).

la desviación típica es relativamente baja y se debe principalmente al ruido introducido en la dinámica del entorno. Este **ruido** se añade para hacer los modelos más robustos y minimizar el posible sesgo.

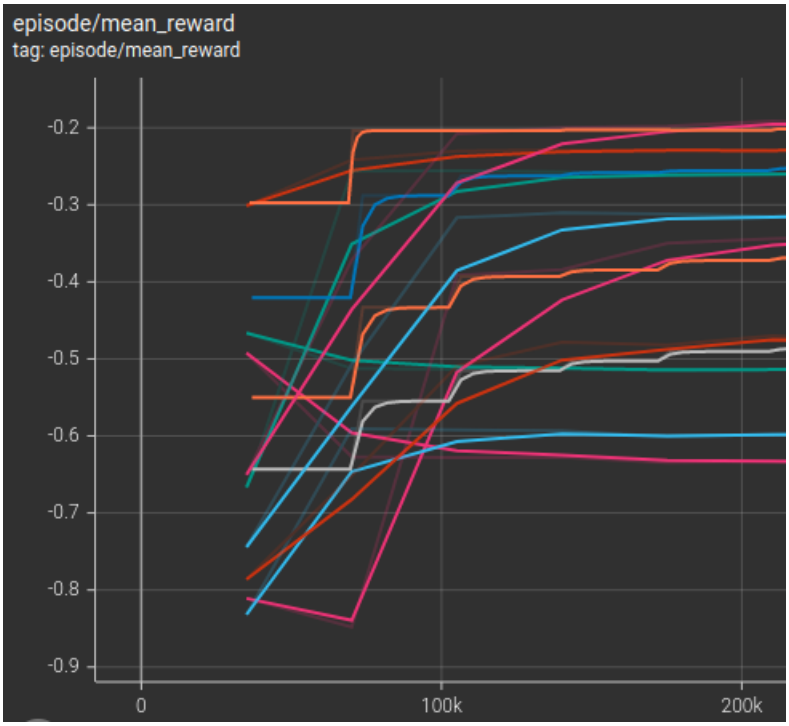


Figura 5.2: Prueba de convergencia temprana de algunos modelos.

5.3.2. Espacios de acciones

Como se ha explicado anteriormente, existen dos tipos de espacios de acciones según su naturaleza: discretos y continuos. En aras de lograr una mayor completitud en el estudio, se han utilizado ambos para las pruebas realizadas.

Espacio de acciones discreto

El espacio de acciones discreto que Sinergym presenta por defecto consiste en 10 pares de *setpoints* mostrados a continuación:

Setpoint calefacción	15	16	17	18	19	20	21	22	22	21
Setpoint refrigeración	30	29	28	27	26	25	24	23	22	21

Cuadro 5.1: Conjunto de *setpoints* del espacio de acciones discreto. Los valores se indican en grados centígrados de temperatura.

Figura 5.3: Seguimiento y organización de experimentos mediante MLFlow.

Los *setpoints* de calefacción indican qué temperatura se debe alcanzar para que se activen los sistemas de calefacción. Los *setpoints* de refrigeración se definen de forma análoga.

Espacio de acciones continuo

En el caso del espacio de acciones continuo, las tuplas de setpoints pueden tomar cualquier valor real que permitan los dispositivos dentro de un rango. El recorrido de este espacio de acciones se define en la siguiente tabla:

Setpoint	**Mínimo**	**Máximo**
Frío	22.5	30.0
Calor	15.0	22.5

Cuadro 5.2: Rangos de *setpoints* del espacio de acciones continuo.

Durante su elección, se ha tratado de reducir el espacio de acciones en medida de lo posible con intención de optimizar el entrenamiento[7], sin perder de vista las recomendaciones de confort y ahorro que se plantean en [64].

5.3.3. Algoritmos

Los algoritmos de aprendizaje por refuerzo profundo constituyen una parte fundamental en la resolución del problema. Tras la integración de Sinergym con *Stable Baselines 3*[8], se posibilita el uso de algoritmos de DRL ya implementados en esta librería. Así, la elección de los algoritmos se ha limitado a aquellos ya presentes en este software, por ser los compatibles con la herramienta de Sinergym. Por otro lado, se ha tratado de elegir candidatos que permitan trabajar con los diferentes espacios de acciones ya descritos, pues no hay una compatibilidad total en este aspecto. De esta manera, se han elegido los siguientes algoritmos:

- Para espacio de acciones discreto: **DQN**.
- Para espacio de acciones continuo: **SAC** y **TD3** (este último por ser una mejora de DDPG).

La razón por la que sólo se ha utilizado un algoritmo para el primer caso es que, en general, los espacios de acciones discretos resultan de menor interés que el caso continuo, ya que suelen suponer una limitación por su propia naturaleza. De todos modos, como se verá en 5.3.5, se utilizarán algunos

[7] Extender en exceso el espacio de acciones tiende a perjudicar el entrenamiento y rendimiento del modelo.

[8] Stable Baselines 3 (o SB3) da nombre a un conjunto de implementaciones de algoritmos de aprendizaje por refuerzo basado en Pytorch.

métodos adicionales durante la experimentación, por lo que no resultará incompleta.

5.3.4. Métricas de evaluación

En este apartado, se describen las distintas métricas de evaluación utilizadas para valorar la calidad de los modelos desarrollados. Antes de comenzar, es preciso recalcar que se tendrán en cuenta los **valores medios** en lugar de los acumulados ya que estos son más "manejables" e intuitivos a la hora de realizar las comparaciones, además de ser igualmente representativos.

Función de recompensa

Como ya se ha indicado, los experimentos se centran especialmente en investigar cómo la función de recompensa puede influir en el desempeño del modelo. Así, la idea principal consiste en plantear varias funciones de recompensa que varíen de forma dinámica según la ocupación del edificio. De esta manera, surgen dos aproximaciones principales:

- La primera propuesta, quizás más intuitiva, aplicaría un **cambio total** en la función de recompensa diferenciando dos posibles escenarios: el edificio o estancia está completamente vacío, o bien hay al menos una persona en su interior. En caso de que esté vacío, se primaría únicamente el ahorro energético, sin considerar el confort térmico[9]. Por el contrario, si hay alguien en la zona (basta con una sola persona), la función de recompensa a aplicar sería la considerada por defecto, que pondera el confort y el consumo energético con un mismo peso (50 % cada uno). Nos referiremos a esta función de recompensa con *OccupRew*.

- La segunda opción planteada propone un **cambio *proporcional***. En otras palabras, el confort térmico toma una importancia directamente proporcional al porcentaje de ocupación de la zona. Por tanto, el peso varía en un rango continuo de $[0, 0.5]$, tomando valor 0 cuando no hay nadie, y 0.5 (peso estándar) cuando la ocupación es máxima. Consecuentemente, el peso del ahorro energético (w_e) tomaría un valor, también dinámico, de $1 - w_c$, siendo w_c el factor que pondera el confort térmico. El nombre que toma esta función es *OccupPropRew*.

Además de una recompensa "lineal" (que pondera con un mismo peso fijo tanto el confort como el consumo energético) utilizada como *baseline*, estas son las dos funciones de recompensa propuestas para este estudio. Sin embargo, se podría utilizar cualquier otro criterio, como modificar los pesos

[9]Esta intuición surge de la idea de que no tiene sentido considerar el confort en un lugar donde no hay nadie.

según las franjas horarias, o dar más prioridad al confort (por ejemplo, darle un peso de 0.5 cuando la ocupación sea superior a la mitad), entre muchos otros. En cualquier caso, se han planteado las opciones consideradas más prometedoras, pues utilizar más funciones de recompensa escapa de los objetivos de este trabajo.

Cabe recordar que todas estas recompensas vienen definidas en base a la expresión antes formulada:

$$r(S_t, A_t) = w_t \cdot \lambda_c \cdot f(S_t, S_{target}) + (1 - w_t) \cdot \lambda_e \cdot coste(A_t) \tag{5.1}$$

En el caso de la **recompensa lineal** (*LinRew*), el peso w_t toma un valor fijo de 0.5, haciendo que tanto el confort como el consumo energético adquieran una misma importancia.

Para conseguir que las dos recompensas diseñadas se adapten a la ocupación de la estancia o edificio, se debe considerar el número de personas que hay en cada momento, n_{occup}. Así, en el caso de ***OccupRew***, el peso w_t se define como:

$$w_t = \begin{cases} 0, & \text{si } n_{occup} = 0 \\ 0.5, & \text{si } n_{occup} > 0 \end{cases}$$

Por su parte, en ***OccupPropRew*** w_t toma un valor proporcional a la ocupación, considerando la máxima posible. Quedaría que:

$$w_t = 0.5 * \frac{n_{occup}}{Occup_{max}}$$

Más concretamente, la métrica utilizada será la **recompensa media** obtenida por los agentes en cada entorno, que se calcula promediando la recompensa recibida del episodio completo. En este caso, permite saber en qué grado el agente controla el consumo de energía de forma eficiente tratando de garantizar el máximo confort térmico posible.

Consumo energético

Aunque la función de recompensa es quizás la medida más relevante por considerar todo lo que se desea optimizar, no se debe perder de vista el consumo energético que, recordemos, es parte de la motivación principal de este trabajo. Por esta razón, resulta de gran interés incluir entre las métricas de evaluación alguna referente a esta magnitud. Se considerará el **consumo**

energético medio de cada episodio para poder estudiar qué ahorro es capaz de lograr cada uno de los agentes entrenados. Esta métrica es calculada por el *logger*, basándose en el valor de la variable *Facility Total HVAC Electric Demand Power*.

Adicionalmente, se puede observar la penalización por consumo (*power penalty*), que equivale a la componente de la función de recompensa referente al consumo: $(1 - w_t) \cdot \lambda_e \cdot Cons$, siendo $Cons$ el consumo energético.

Violación de confort

Además del consumo energético, el otro aspecto que trata de optimizarse en este problema es el confort térmico de los ocupantes de la infraestructura. Para ello, se considerará la **violación de confort** medida por Sinergym, que refleja el porcentaje de tiempo del episodio que se pasa con una temperatura fuera de los rangos recomendados para cada estación.

Es importante destacar que la violación de confort no es igual en todas las funciones de recompensa. En *LinRew* se considerará la misma penalización durante todo el episodio. Sin embargo, tanto en *OccupRew* como en *OccupPropRew* la penalización será nula cuando no haya ocupación en el edificio, siendo en realidad así más precisa, ya que cuando no hay nadie, no debe importar la temperatura interna de la estancia[10].

Al igual que ocurría con el consumo, también se considera la penalización por confort (*comfort penalty*) que, de forma análoga, hace referencia a la componente correspondiente al confort en la función de recompensa, pudiendo definirla como: $w_t \cdot \lambda_c \cdot Conf$, siendo $Conf$ el confort térmico.

5.3.5. Experimentación adicional

Además de los experimentos principales propuestos, según se profundizaba tanto en el ámbito del RL como en la herramienta y posibilidades de Sinergym, surgieron algunas ideas adicionales interesantes para extender la experimentación propuesta y poder estudiar así ciertos aspectos descritos en esta sección.

Extensión del espacio de acciones discreto

La primera cuestión que llama la atención acerca de los espacios de acciones que propone Sinergym es la dimensionalidad del caso discreto. Existen

[10]Este es un aspecto particular de cada edificio, ya que en el *data center*, la temperatura es importante durante las 24 horas del día, para el buen funcionamiento de los equipos electrónicos.

tan sólo 10 pares de *setpoints*. Aparentemente, parecen muy pocas posibilidades para tratar el problema en cuestión, por lo que se propone aumentar este espacio de acciones y comparar el rendimiento de ambos, con el objetivo de comprobar si dicha extensión en el conjunto de acciones puede lograr una mejora en el desempeño de los agentes.

A la hora de decidir qué tuplas añadir, se han considerado las recomendaciones en [64].

Experimentación con algoritmos *on-policy*

Otra idea sugerente es el uso de algoritmos *on-policy*[11], ya que permiten realizar un entrenamiento bastante más rápido, lo que puede resultar muy útil cuando hay restricciones temporales y/o el entorno es tan complejo que el entrenamiento requiere una duración inviable. Sin embargo, este ahorro de tiempo tiene un coste que suele reflejarse en la calidad de las soluciones. En otras palabras, los algoritmos *on-policy* resultan generalmente en un modelo con peor desempeño que los entrenados con algoritmos *off-policy*. En cualquier caso, se realizará un estudio para ver en qué grado afecta este aspecto al resultado obtenido.

Para ello, se propone el uso de dos algoritmos de esta naturaleza: **A2C** para el espacio de acciones discreto y **PPO** para el continuo. De nuevo, la elección de estos métodos se debe parcialmente a que son opciones integradas en *Stable Baselines 3*.

[11] Recordemos que los algoritmos *on-policy* son aquellos que hacen uso de una sola política, que será utilizara para tomar decisiones durante el aprendizaje mientras trata de aproximar la óptima.

Capítulo 6

Análisis de resultados

Antes de comenzar a estudiar los resultados, resulta conveniente mencionar la "dificultad" de su análisis, y es que este puede ser planteado desde varios enfoques, pues existen distintas variables en las pruebas como son los climas, algoritmos, espacios de acciones y funciones de recompensa utilizadas. En este capítulo se presentarán en distintas tablas todos los resultados obtenidos en los experimentos realizados, con el objetivo de que el lector disponga de toda la información y pueda realizar las comparaciones adicionales que desee.

En cualquier caso, la estructura del análisis es la que sigue:

1. Primero se centrará la atención en el valor de **recompensa media** obtenido. Esto permitirá, entre otras cosas, estudiar si la extensión aplicada al espacio de acciones discreto ha sido útil y permite mejorar el rendimiento. Además, permite observar qué algoritmos funcionan mejor para cada recompensa en los diferentes climas. Recordemos que las funciones de recompensa a maximizar toman valores negativos, por lo que su valor será mejor cuanto más próximo a cero esté.

2. A continuación, se observará el **consumo energético**, que permitirá observar si se ha cumplido uno de los principales objetivos de este trabajo, mostrando si las recompensas propuestas logran algún ahorro energético y en qué medida, además de indicar los agentes óptimos en los distintos escenarios.

3. Finalmente, se estudiará el **confort térmico** (más concretamente, la violación de este) de cada agente con las distintas recompensas en cada clima. Además, con la segunda recompensa (*OccupPropRew*) asumimos un aumento en el porcentaje de violación de confort, así que se observará en qué medida se da tal incremento, permitiendo valorar la calidad de esta función de recompensa.

Además, se mostrarán los resultados de un **controlador basado en reglas** (RBC), que será utilizado como referencia (*baseline*). El comportamiento de este controlador es bastante sencillo: establece un par de *setpoints* fijos u otro según la estación del año vigente en cada momento: 26°C y 29°C para invierno, y 20°C y 23.5°C para verano [31].

6.1. Recompensa media

Siguiendo el orden propuesto, observemos primero las distintas recompensas medias de cada algoritmo. Para empezar, se muestra una tabla con los resultados de medir la recompensa lineal media obtenidas por agentes entrenados con DQN y A2C con los espacios de acciones reducido y extendido, en tres climas distintos.

	HOT		**MIXED**		**COOL**	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C (red)	**-.402**	.001	**-.270**	.001	-.476	.001
DQN (red)	-.412	.002	-.291	.002	-.489	.002
A2C (ext)	-.578	.002	-.420	.001	-.203	.001
DQN (ext)	-.461	.001	-.305	.001	**-.194**	.001

Cuadro 6.1: Recompensa lineal media con espacios de acciones discretos (reducido y extendido). Se resalta el mejor resultado en cada clima.

Contrariamente a lo que se podría intuir, observamos que **no existe una clara superioridad** en los resultados de los agentes entrenados con un espacio de acciones extendido sobre el reducido. En los climas cálido y mixto, los espacios de acciones reducidos muestran un rendimiento ciertamente superior (con una mejora de 0.059 y 0.035, respectivamente[1]), aunque bastante peor en el clima frío (un empeoramiento en la recompensa de 0.282).

Como se ha indicado, la intuición nos dice que los agentes entrenados en el espacio de acciones extendido deberían ofrecer mejores resultados, ya que este espacio contiene al reducido. Sin embargo, observamos que no siempre es así. La razón más probable que explica este comportamiento es que los agentes entrenados con el espacio extendido **no realicen una exploración y explotación equilibradas** durante su entrenamiento, quizás porque necesiten más iteraciones para descubrir qué acciones son mejores en ciertos estados. Es el problema ya mencionado en capítulos anteriores de encontrar un equilibrio entre explotación y exploración durante la búsqueda, cuya dificultad aumenta de forma proporcional con la dimensionalidad del espacio

[1]Nótese que estas diferencias se han calculado considerando los dos mejores agentes de cada espacio de acciones.

de acciones. En otras palabras, no siempre es beneficioso extender las acciones permitidas para el agente, ya que este necesitará más tiempo para explorarlas y evaluarlas todas.

Con intención de realizar un análisis comparativo lo más justo y consistente posible, de aquí en adelante se considerarán los agentes entrenados en un mismo espacio de acciones, en lugar de elegir el mejor para cada caso (esto rompería en cierto modo la igualdad de condiciones). Concretamente, **se considerará el espacio de acciones extendido** principalmente por tres razones. Por un lado, si se suman las diferencias anteriormente calculadas, se observa que los agentes entrenados en el espacio de acciones extendido presentan mejor rendimiento (0.094 frente a 0.282). Dicho de otra forma, la suma de las distancias al mejor agente en cada caso es menor. Por otro lado, existe un interés especial en el clima frío (*cool*), donde la segunda versión de A2C y DQN son mejores, ya que es el clima del edificio real en que se basa el entorno utilizado. Adicionalmente, el hecho de elegir un espacio de acciones más grande supone la resolución de un problema más completo a la par que complejo. Dicho esto, de ahora en adelante se considerarán únicamente las versiones entrenadas en el espacio de acciones extendido para los algoritmos DQN y A2C.

Veamos ahora la recompensa lineal media que se logra con cada agente, considerando espacios de acciones continuo y discreto[2].

	HOT		**MIXED**		**COOL**	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	-.578	.002	-.420	.001	-.203	.001
DQN	-.461	.001	-.305	.002	**-.194**	.001
PPO	-.467	.001	-.360	.002	-.250	.001
SAC	-.451	.001	-.346	.001	-.226	.001
TD3	-.593	.003	-.342	.001	-.258	.001
RBC	**-.415**	.001	**-.284**	.001	-.197	.002

Cuadro 6.2: Recompensa lineal media obtenida por cada agente. Se resalta el mejor resultado en cada clima.

Lo primero que llama la atención es la superioridad que presenta el controlador basado en reglas (RBC), obteniendo la mejor recompensa en todos los climas excepto en el frío, donde presenta una recompensa bastante similar al mejor agente (DQN). Si consideramos sólo los agentes de DRL,

[2]Los espacios de acciones discretos considerados de aquí en adelante son los extendidos.

encontramos que aquellos que mejores resultados ofrecen son DQN (con espacio de acciones discreto) y SAC (espacio de acciones continuo), por lo que no se observa una ventaja clara al usar un espacio de acciones u otro. Esto **niega la intuición** de que un espacio continuo es generalmente mejor que uno discreto por presentar más posibilidades pues, en este caso, parece que el espacio de acciones discreto permite las opciones suficientes y su uso no supone una limitación[3].

Observemos ahora las recompensas obtenidas con las funciones de recompensa propuestas, que consideran la ocupación de las estancias para ponderar la importancia del confort: ***OccupRew*** y ***OccupPropRew***.

	HOT		**MIXED**		**COOL**	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	-.414	.001	-.376	.002	-.211	.001
DQN	-.420	.002	-.374	.003	-.208	.001
PPO	-.405	.001	-.348	.002	-.222	.001
SAC	**-.382**	.000	**-.341**	.002	-.200	.001
TD3	-.629	.003	-.513	.003	**-.187**	.001
RBC	-.463	.001	-.474	.002	-.317	.002

Cuadro 6.3: Recompensa *OccupRew* media obtenida por cada agente. Se resalta el mejor resultado en cada clima.

	HOT		**MIXED**		**COOL**	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	-.409	.001	-.346	.002	-.240	.001
DQN	-.410	.002	-.354	.003	-.203	.002
PPO	-.397	.001	-.341	.002	-.203	.001
SAC	**-.394**	.001	-.342	.002	-.194	.001
TD3	-.551	.001	**-.330**	.002	**-.186**	.001
RBC	-.488	.001	-.492	.002	-.322	.002

Cuadro 6.4: Recompensa *OccupPropRew* media obtenida por cada agente. Se resalta el mejor resultado en cada clima.

Al tratarse de funciones de recompensa diferentes, no tiene mucho senti-

[3]No se debe confundir esta comparación entre espacios de acciones discreto y continuo con la realizada anteriormente, que sólo trataba el caso discreto.

do compararlas entre sí. Sin embargo, nos permite saber qué agentes obtienen mejores resultados considerando consumo energético y confort, siendo en ambos casos SAC y TD3 los mejores algoritmos. Aparece así cierta predominancia de los espacios de acciones continuos cuando se utilizan estas recompensas. Por otro lado, el RBC deja de ser la mejor opción en ambos casos, en especial con *OccupPropRew*, donde adopta el peor comportamiento respecto al resto de agentes. Se explicará la razón en el capítulo 7.

6.2. Consumo energético

Llega el momento de estudiar el consumo energético de los distintos agentes. De nuevo, empecemos con aquellos entrenados con **recompensa lineal** (*LinRew*):

	HOT		**MIXED**		**COOL**	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	**5821**	9.5	5346	31.8	3512	17.1
DQN	5886	24.6	**5202**	34.5	3262	17.5
PPO	6104	9.2	5626	32.9	3366	15.6
SAC	5864	10.5	5231	29.8	**3167**	15.9
TD3	6006	10.1	5442	30.5	3441	16.4
RBC	5888	10.6	5645	22.2	3301	14.6

Cuadro 6.5: Consumo energético medio con *LinRew* obtenido por cada agente. Se resalta el mejor resultado en cada clima.

No encontramos un algoritmo mejor que el resto en general, sino que el mejor puesto se reparte entre A2C, DQN y SAC según sea clima cálido, mixto o frío, respectivamente, aunque sin presentar mejoras muy considerables con respecto al resto de agentes. Todos ellos superan al *baseline* y, aunque la diferencia no es extremadamente alta, sí que existe **cierto ahorro energético** (entre el 1.14 % y el 7,85 %).

	HOT		MIXED		COOL	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	**4443**	8.8	**3575**	20.2	**1944**	8.8
DQN	4933	20.5	4108	31.0	2226	13.7
PPO	4557	6.5	3930	22.5	2078	11.8
SAC	4791	9.2	3825	20.0	2155	12.4
TD3	6006	10.1	5677	34.1	2092	10.6
RBC	5895	11.7	5662	28.4	3319	16.5

Cuadro 6.6: Consumo energético medio con *OccupRew* obtenido por cada agente. Se resalta el mejor resultado en cada clima.

Con la recompensa ***OccupRew***, se puede notar un **ahorro bastante notable** en referencia a la recompensa lineal: entre el 23.67 % y el 38.62 %. Tal y como podría esperarse, se alcanza cierta reducción en el consumo energético, ya que al no considerar el confort si no hay ocupantes, se relaja bastante la demanda energética. Destaca el agente entrenado con A2C, por ser el que menor consumo requiere en los tres escenarios posibles. Por otro lado, al igual que en la siguiente recompensa, el RBC requiere una cantidad de energía similar al caso anterior. Esto se debe a que es independiente de la función de recompensa utilizada. En consecuencia, se acentúan las diferencias en el consumo (el ahorro oscila entre el 24,63 % y el 41,43 %), resultando el RBC en una opción cada vez menos atractiva.

	HOT		MIXED		COOL	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	**4473**	8.5	**3565**	20.1	2014	9.9
DQN	4544	20.4	3680	25.1	2114	15.0
PPO	4551	6.9	3696	20.9	2020	10.0
SAC	4673	10.5	3660	20.4	2060	8.8
TD3	6264	11.1	3626	19.2	**1988**	9.0
RBC	5909	15.0	5627	31.1	3307	31.7

Cuadro 6.7: Consumo energético medio con *OccupPropRew* obtenido por cada agente. Se resalta el mejor resultado en cada clima.

Por último, tenemos la **recompensa con peso proporcional a la ocupación**. En el cuadro 6.7 encontramos que A2C es el mejor candidato para entornos cálidos y mixtos, mientras que TD3 supera a todos en el clima frío. Si observamos sólo los mejores resultados (aquellos resaltados en la

tabla) y los comparamos con los mejores de la recompensa anterior, nos percatamos de que apenas hay diferencia. Sin embargo, si comparamos el resto de algoritmos encontramos que se logra un ahorro de hasta 36.12 % entre ambas, por lo que sí existe una **diferencia sustancial entre estas dos funciones de recompensa**. De todos modos, parece existir **cierta cota en el ahorro** cuando los valores disminuyen, donde las dos recompensas consiguen un consumo energético muy similar, tal y como se ha visto. En relación al RBC, se alcanza un ahorro muy similar al caso anterior.

6.3. Confort térmico

Finalmente, es conveniente estudiar el confort térmico basándonos en el **porcentaje de tiempo con violación de confort** que producen los distintos modelos. Cabe indicar que se considerará violación de confort en aquellos *timesteps* donde la temperatura de la estancia esté fuera de los intervalos anteriormente indicados, independientemente de la distancia a los extremos del intervalo (es decir, no se considera la magnitud de la violación de confort). Primero, se presenta el caso de la **recompensa lineal**:

	HOT		**MIXED**		**COOL**	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	41.2	.131	37.9	.201	**20.3**	.171
DQN	33.9	.059	28.8	.335	20.5	.311
PPO	33.8	.171	33.0	.088	32.6	.094
SAC	32.2	.071	33.6	.070	27.3	.117
TD3	41.4	.121	29.6	.079	34.0	.042
RBC	**30.4**	.121	**27.8**	.328	26.3	.298

Cuadro 6.8: Violación de confort media (%) con *LinRew* obtenido por cada agente. Se resalta el mejor resultado en cada clima.

Se usarán estos resultados para tener una referencia de los valores que se obtienen, que van desde el 20.3 % al 41.9 % según el clima y algoritmo utilizado. El controlador basado en reglas recupera el protagonismo en los climas cálido y mixto, mientras que A2C se convierte en el mejor candidato para el clima frío.

	HOT		MIXED		COOL	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	18.8	.101	17.1	.080	17.4	.106
DQN	12.6	.109	15.0	.175	8.3	.049
PPO	13.3	.117	13.8	.072	14.5	.085
SAC	9.6	.071	12.1	.093	8.2	.106
TD3	18.2	.095	12.8	.001	**6.15**	.069
RBC	**2.5**	.037	**8.5**	.134	10.2	.229

Cuadro 6.9: Violación de confort media (%) con *OccupRew* obtenido por cada agente. Se resalta el mejor resultado en cada clima.

Al emplear la función de recompensa ***OccupRew***, llama la atención el gran decremento que experimentan los valores del porcentaje de violación de confort obtenidos. Este fenómeno tiene una explicación lógica, y es que con la nueva recompensa, las violaciones de confort sólo serán posibles cuando haya alguien en el edificio. De esta manera, esta métrica debe ser igual o menor que en el caso lineal. Es más, una reducción de esta magnitud es bastante normal, ya que son bastantes horas en las que el edificio está vacío y no tiene sentido considerar violaciones de confort.

En cualquier caso, encontramos que los agentes que más respetan el confort térmico son TD3 para entornos fríos y RBC para el resto, logrando este valores tan bajos como 2.5 % y 8.5 %.

	HOT		MIXED		COOL	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	18.8	.099	17.2	.079	17.3	.194
DQN	13.0	.112	16.6	.083	13.3	.072
PPO	12.9	.146	15.0	.071	15.2	.112
SAC	10.0	.036	14.1	.053	10.7	.127
TD3	12.8	.003	13.0	.086	**8.4**	.111
RBC	**2.5**	.034	**8.6**	.062	10.2	.075

Cuadro 6.10: Violación de confort media (%) con *OccupPropRew* obtenido por cada agente. Se resalta el mejor resultado en cada clima.

Por último, observemos la función de recompensa ***OccupPropRew***. Es importante recordar que la penalización para *OccupPropRew* es la misma que para *OccupRew*. De esta se puede intuir que provocará una mayor viola-

ción de confort, ya que tiende a otorgar mayor importancia al ahorro energético (sólo considera el confort tan importante como el consumo cuando la ocupación es máxima). La tabla 6.10 confirma esta hipótesis, donde podemos encontrar un aumento de hasta un 5 % en la violación de confort. Sin embargo, si centramos la atención en los mejores agentes (los de mejor confort, sin considerar el RBC ya que es independiente de la función de recompensa), la penalización que añade esta recompensa con respecto a *OccupRew* va desde un 0.4 % a un 2.25 %.

De todos modos, como se indica, para esta recompensa se asume un empeoramiento a priori del confort térmico. Dicho esto, resulta más interesante compararla con la recompensa *OccupRew* y observar qué ahorro energético ofrece a cambio de cuánto empeoramiento en la violación de confort. Para ello, se han realizado los cálculos necesarios para cada algoritmo y se muestran los resultados en la tabla 6.11 [4]. A la hora de valorar la bondad de la recompensa *OccupPropRew*, para cada agente se desea maximizar el valor de la columna *cons* y minimizar el valor de *v. conf*[5].

	HOT		**MIXED**		**COOL**	
	cons (↓)	*v. conf* (↑)	*cons* (↓)	*v. conf* (↑)	*cons* (↓)	*v. conf* (↑)
A2C	-30	0.0	10	0.1	-70	-0.1
DQN	**389**	**0.4**	428	1.6	112	5.0
PPO	**6**	**-0.4**	234	1.2	**58**	**0.7**
SAC	118	0.4	165	2.0	95	2.5
TD3	-258	-5.4	**2051**	**0.2**	104	2.25

Cuadro 6.11: Comparación *OccupPropRew* y *OccupRew* para todos los algoritmos. Se resalta el mejor resultado en cada clima.

Se resaltan en negrita los casos que muestran más mejora con esta nueva recompensa, es decir, permiten una mayor reducción de la demanda energética sacrificando el mínimo confort posible.

Aparece un comportamiento relativamente irregular, ya que esta función de recompensa debería reducir el consumo energético aumentando generalmente el porcentaje de violación de confort, pero no siempre es así[6]. De hecho, hay algunos casos como A2C donde la diferencia es mínima, mientras que el agente entrenado con DQN en el clima cálido o, sobre todo, TD3 en

[4] No se ha incluido la variación de RBC ya que este es ajeno a la función de recompensa, y su variación se debe exclusivamente a la componente estocástica de las simulaciones.

[5] La columna *cons* hace referencia a la diferencia de consumo energético. La columna *v. conf* indica cuánto aumenta el porcentaje de violación de confort.

[6] Hay casos donde el porcentaje de violación de confort disminuye (valores negativos), lo cual podría resultar incoherente. Sin embargo, no son valores muy altos y esto podría deberse a la aleatoriedad de las simulaciones.

el clima mixto devuelven resultados bastante deseables. En conclusión, esta función de recompensa resulta menos útil de lo que podría parecer, aunque su utilización siempre dependerá del interés del usuario.

Otro estudio interesante es la evaluación de los modelos entrenados usando *LinRew* con la función de recompensa *OccupRew*, que no penaliza el confort cuando no hay ocupación en la estancia. Esta evaluación es necesaria si se quiere analizar cómo afecta la función *OccupRew* al confort utilizando la función base *LinRew* como referencia, ya que no se pueden comparar los resultados de las tablas 6.8 y 6.9, pues sus valores son resultados de aplicar distintas penalizaciones. Se obtiene la sigiente tabla:

	HOT		**MIXED**		**COOL**	
	media	*desv*	*media*	*desv*	*media*	*desv*
A2C	18.0	.103	15.9	.185	10.1	.150
DQN	9.3	.098	10.1	.124	7.4	.085
PPO	11.7	.129	12.2	.079	11.6	.067
SAC	8.6	.054	10.5	.053	6.3	.131
TD3	18.2	.095	8.9	.075	12.8	.002

Cuadro 6.12: Violación de confort media (%) de modelos entrenados con *LinRew* medido con *OccupRew*. Se resalta el mejor resultado en cada clima.

Antes de analizar los resultados, se van a calcular las diferencias de consumo y confort entre los modelos entrenados con *LinRew* y *OccupRew*, para que la comparación resulte más intuitiva y amigable al lector:

	HOT		**MIXED**		**COOL**	
	cons (↓)	*v. conf* (↑)	*cons* (↓)	*v. conf* (↑)	*cons* (↓)	*v. conf* (↑)
A2C	**-1378**	**0.8**	**-1771**	**1.2**	-1568	7.3
DQN	-953	3.3	-1094	4.9	-1036	0.9
PPO	-1547	1.6	-1696	1.6	-1288	2.9
SAC	-1073	1.0	-1406	1.6	-1012	1.9
TD3	0	0.0	235	3.9	**-1349**	**-6.65**

Cuadro 6.13: Comparación *LinRew* y *OccupRew* para todos los algoritmos. Se resalta el mejor resultado en cada clima.

En la tabla 6.13 se muestra el resultado de restar los valores de consumo y confort de los agentes entrenados con *OccupRew* a aquellos entrenados con *LinRew*. Los valores negativos de la columna de consumo reflejan un ahorro energético por parte de los primeros a cambio de un aumento en la violación de confort, que se muestra en las columnas adyacentes. Se resaltan

los agentes que consiguen mayor ahorro energético con el mínimo aumento en la violación de confort (en otras palabras, los que más mejoran al utilizar esta recompensa).

La gran mayoría de valores en la columna de consumo son negativos, lo cual indica que en esos casos se consigue una **reducción de la demanda energética** al utilizar esta señal de recompensa. Casi todos ellos superan las 1000 unidades y, teniendo en cuenta que la escala del consumo energético va desde 2000 a 6000, esta cantidad representa un ahorro bastante significativo. Por otro lado, está la violación de confort, expresada en porcentaje. En la mayoría de los casos no supera el 2 %. Esto quiere decir que **el ahorro energético del que se habla no supone apenas un aumento en la penalización por confort**, resultando ser así la función de recompensa *OccupRew* una alternativa bastante sugestiva, cumpliendo el objetivo para el que ha sido diseñada. No obstante, considerando la propia naturaleza de la recompensa, la pérdida en la métrica de confort sugiere que el entrenamiento y, en consecuencia el rendimiento del agente, **no es óptimo**. Probablemente, gran parte de la violación de confort se produzca cuando el edificio comienza a tener ocupación (por ejemplo, tras su apertura por la mañana). Así, si durante el aprendizaje no se capta correctamente la **inercia térmica** del edificio, el sistema de climatización no se activará suficientemente pronto como para conseguir una temperatura idónea cuando aumente la ocupación, traduciéndose esto en un decremento del confort. El concepto de inercia térmica está relacionado con el principio físico de conservación de energía. En este caso, se refiere a la tendencia de un edificio de mantener una misma temperatura a lo largo del tiempo, teniendo en cuenta también que los materiales con los que se construyen son en cierto grado aislantes térmicos, además de ser diseñados para conservar la temperatura interior independientemente de la exterior. Por otro lado, de forma ajena a la temperatura del exterior, esta inercia térmica también afecta al interior: se requiere un esfuerzo (y tiempo) para cambiar la temperatura interior. Idealmente, el agente debería "aprender" esta dinámica y actuar en consecuencia.

Una vez realizado este análisis, es momento de recapitular y formular todas las ideas y conclusiones que pueden extraerse de este estudio. Esta tarea queda reflejada en el siguiente y último capítulo.

Capítulo 7

Conclusiones

A modo de compendio, en este capítulo final se resume el trabajo realizado, además de extraer las conclusiones más relevantes a las que este ha dado pie. Para ello, se comenzará indicando qué tareas se han llevado a cabo y si se han cubierto los objetivos planteados al principio. Después, se condensan las conclusiones que se pueden obtener a partir del estudio y sus resultados, para ultimar con una serie de posibles trabajos futuros que derivan de este proyecto.

7.1. Tareas realizadas

A lo largo de este proyecto, se puede distinguir una lista de subtareas que han resultado imprescindibles para su realización. Así, ha sido necesario un estudio intensivo de los fundamentos del aprendizaje por refuerzo, además de una introducción en la vertiente del aprendizaje por refuerzo profundo para poder afrontar el problema que se plantea. También se ha llevado a cabo un estudio del *software* utilizado, en el que figuran *Sinergym*, *SinergymCloud*, *EnergyPlus*, herramientas de análisis de datos y monitorización de modelos de ML como *Tensorboard* y *MLFlow*, y librerías de RL como *Stable Baselines*. Ha sido necesario además el desarrollo de un entorno para la ejecución de los experimentos tanto en local como en remoto. Adicionalmente, se ha llevado a cabo dicha experimentación, para su posterior análisis y la extracción de conclusiones.

7.1.1. Cobertura de objetivos

En este apartado, se retoman los objetivos planteados al principio de la memoria y se analiza su consecución.

Objetivo 1 : **Introducción y estudio de fundamentos de RL y DRL. Familiarización con el problema de control de sistemas HVAC y su resolución con técnicas de aprendizaje por refuerzo.**

Antes de afrontar la resolución del problema, ha sido necesario un estudio previo de las bases del RL y DRL. Estos conocimientos han quedado reflejados en el capítulo 2, abordando desde los conceptos más básicos hasta algoritmos de *deep learning* relativamente avanzados. Por otro lado, se ha estudiado y formulado el planteamiento del problema de control energético de edificios desde la perspectiva del aprendizaje por refuerzo.

Objetivo 2 : **Revisión literaria e investigación del estado del arte. Análisis de problemas y proposición de soluciones.**
En el capítulo 3 se realiza la revisión literaria y se analiza el estado del arte, destacando las publicaciones más relevantes hasta la fecha en que se elabora este proyecto. Durante dicho análisis, se han tratado de identificar los principales problemas vigentes en la mayoría de estudios realizados, proponiendo soluciones cuando era posible. En concreto, el principal problema hallado es la falta de comparabilidad entre los modelos desarrollados por cada autor, causada por la inexistencia de un entorno común donde entrenar, validar y comparar los agentes bajo igualdad de condiciones. Esto impide saber cuáles son los mejores modelos desarrollados hasta ahora, y añade bastante incertidumbre al analizar su calidad, pues el ahorro energético que consiguen es en referencia a un *baseline* que cada autor desarrolla. Uno de los objetivos de Sinergym es proponer un *framework* universal para centralizar todos estos trabajos, y poder así comparar los modelos desarrollados.

Objetivo 3 : **Estudio de Sinergym e integración en el equipo de trabajo. Contribuciones al desarrollo.**
Como ya se ha indicado varias veces a lo largo de la memoria, Sinergym ha resultado una herramienta indispensable. Así, ha sido necesario un estudio previo de este *software*, además de conocer e integrarse al equipo que hay detrás, formado principalmente por miembros de la UGR. Conocer bien el código y la estructura del programa ha sido crucial para poder implementar más adelante los cambios necesarios para posibilitar la realización de los experimentos propuestos.

Objetivo 4 : **Desarrollo del entorno *software* necesario para la ejecución de los experimentos, tanto de forma local como remota.**
Descrito en el capítulo 4 con más profundidad, ha sido preciso también el desarrollo del código que permitiese la realización de la experimentación tanto en local (pruebas unitarias) como en remoto (batería de experimentos con alta carga computacional). Además, el desarrollo *software* también incluye la detección y corrección de errores menores detectados.

Objetivo 5 : **Realización de la experimentación propuesta. Entrenamiento y evaluación de modelos, utilizando herramientas de monitorización como *Tensorboard* y *MLFlow* para su registro.**

Una vez completadas todas las tareas precedentes, se ha continuado con la ejecución de los experimentos previamente propuestos. Durante el entrenamiento de los agentes, ha sido muy útil el uso de *MLFlow*, para registrar los modelos, sus hiperparámetros y resultados, además de *Tensorboard*, que ha permitido detectar errores, comprobar la corrección de los resultados y facilitar su análisis. Cabe destacar que los experimentos realizados han sido diseñados buscando la mayor completitud posible, considerando aún así las limitaciones existentes. De esta manera, se han entrenado agentes en espacios de acciones discretos y continuos, utilizando distintos algoritmos y en tres entornos distintos cada uno, con sus características climatológicas propias.

Objetivo 6 : **Propuesta y realización de estudios adicionales.**

Durante el planteamiento de los experimentos y el estudio previo a este, surgieron ideas sobre pruebas adicionales que se podían realizar, en aras de obtener nuevas conclusiones e investigar cómo afectan ciertos aspectos en el desempeño de los agentes que se diseñarían. Entre estas pruebas, figuran el estudio de mejora que puede ofrecer un espacio de acciones discreto extendido sobre uno reducido (el original de Sinergym), o si el uso de algoritmos con una estrategia *on-policy* puede mejorar en algún caso los resultados obtenidos por técnicas *off-policy*. Se describen en 5.3.5.

Objetivo 7 : **Análisis desde múltiples enfoques de los resultados obtenidos.**

Una vez recopilados los datos resultantes, se ha procedido a realizar su análisis desde distintos puntos de vista, tratando de cubrir todas las dimensiones y factores variables entre los distintos modelos. Asimismo, de forma cuidadosa, se han recogido los resultados para más tarde procesarlos y mostrarlos de forma resumida a lo largo del capítulo 6. Como se indica, se ha comparado el desempeño de los modelos generados considerando distintos criterios, con el objetivo de realizar un análisis más profundo y completo, además de poder extraer diversas conclusiones.

Objetivo 8 : **Extracción de conclusiones y aportaciones útiles a la comunidad científica. Planteamiento de posibles trabajos futuros de interés.**

Por último, en este capítulo y especialmente en la sección 7.2 se han aglomerado y detallado todas las conclusiones que el desarrollo de este proyecto ha permitido extraer. Además de intentar resumir los

conceptos más importantes de este trabajo, se ha buscado recopilar todas aquellos aspectos que se pueden deducir de la experimentación, con el fin de contribuir en medida de lo posible a futuras investigaciones por parte de la comunidad científica.

Tras este repaso, queda confirmada la realización de todos y cada uno de los objetivos propuestos en el planteamiento del proyecto. Incluso ha sido posible lograr la consecución de algunos objetivos adicionales cuyo interés surgió durante el desarrollo de este trabajo.

7.2. Conclusiones

Una vez realizado el análisis de resultados desde los distintos enfoques propuestos, se procede a condensar en esta sección las conclusiones que se pueden extraer de este proyecto.

Antes de comenzar, conviene recordar el interés que tiene la resolución de este problema. La optimización en el control energético de edificios puede permitir una reducción significativa del consumo energético y consigo, de las emisiones de CO_2 y otros gases y residuos contaminantes. Sin embargo, esto no quiere decir que se trate de un problema ya resuelto. Especialmente cuando se aborda desde la disciplina del aprendizaje por refuerzo. Asimismo, aún sigue siendo un reto superar sistemas clásicos como RBCs mediante el uso de técnicas más sofisticadas, como DRL. En cualquier caso, la aplicación del aprendizaje por refuerzo en el control de sistemas de climatización parece bastante prometedora, pues aunque no siempre consiga superar a los métodos clásicos, sí que lo logra en muchos casos, y cada vez más, como se verá más adelante en la tabla 7.1. Dicho esto, podemos comenzar a enumerar las principales ideas que pueden concluirse de este trabajo.

Por un lado, se ha visto cómo el espacio de acciones discreto óptimo no debe ser necesariamente amplio. De hecho, se ha encontrado que **el espacio de acciones discreto extendido no siempre mejora al reducido**. Esto sugiere que la calidad de un espacio de acciones no es directamente proporcional a su tamaño. Nace así la necesidad de un estudio más profundo acerca de cuál es el conjunto de pares de *setpoints* óptimo a la hora de entrenar un modelo, considerando distintos entornos como edificios residenciales, centros comerciales, infraestructuras industriales, etcétera.

Continuando con el espacio de acciones, también se ha demostrado empíricamente que **el caso continuo no es estrictamente mejor que el discreto**. Si consideramos que, en general, el entrenamiento del caso discreto suele ser menos costoso que el continuo, resultaría bastante útil el diseño de uno o varios espacios de acciones discretos "universales" para incrementar

la eficiencia de los modelos.

Respecto al entrenamiento, se ha observado que tampoco se requiere que su duración sea necesariamente alta (más de 20 episodios, por ejemplo). En 5.2 se mostraba la **pronta convergencia de los modelos** durante los primeros episodios. Sabiendo esto, puede resultar más interesante volcar los esfuerzos y recursos computacionales en tareas como el ajuste de hiperparámetros (*hyperparameter tuning*) en lugar de realizar entrenamientos excesivamente prolongados. Se propone así un nuevo enfoque para experimentaciones futuras.

Centremos ahora la atención en los resultados numéricos. A lo largo de este trabajo, se ha tratado de maximizar el ahorro en el consumo energético, manteniendo el confort térmico de los ocupantes del edificio. En el cuadro 7.1 se muestra un resumen de los mejores agentes según distintos criterios, donde destaca la columna " %", que refleja el porcentaje en que se ha superado al RBC (Rule-based controller, controlador basado en reglas) en cada métrica:

	HOT			**MIXED**			**COOL**		
	Agente	*Val*	%	*Agente*	*Val*	%	*Agente*	*Val*	%
Consumo	A2C	4443	24.3	A2C	3565	36.6	A2C	1944	41.4
Confort	RBC	2.5	-	RBC	8.5	-	TD3	6.15	4.05
LinRew	RBC	-.415	-	RBC	-.284	-	DQN	-.194	1.1
OccupRew	SAC	-.382	17.5	SAC	-.341	28.1	TD3	-.187	42.9
OccupPropRew	SAC	-.394	19.3	TD3	-.330	32.9	TD3	-.186	42.2

Cuadro 7.1: Resumen de mejores agentes según distintos criterios.

En esta tabla encontramos que la experimentación planteada ha sido exitosa, logrando un **ahorro energético** del **24.3 %** en climas cálidos, **36.6 %** en mixtos y hasta un **41.4 %** de ahorro en ambientes fríos con respecto al *baseline*. De esta forma, se cumple el objetivo que motiva principalmente este trabajo: la reducción de la demanda energética de los sistemas de climatización. Como se estudió en el capítulo 6, son los agentes basados en las funciones de recompensa diseñadas los que mayor ahorro suponen, lo cual valida el **buen diseño de las funciones propuestas**. Sin embargo, esto no quiere decir que dichas funciones sean óptimas, pues se encuentra que, en lo que a confort se refiere, el RBC es el mejor controlador en climas cálidos y mixtos.

Cabe añadir que el **RBC** obtiene buenos resultados con la recompensa lineal pero no tanto con el resto. Esto se debe a que dicho controlador **prima bastante el confort** (al mantener fijos los *setpoints*, se consigue reducir en gran medida la violación de confort), y este tiene menos importancia en las

otras recompensas[1]. Por su parte, vemos que las funciones de recompensa *OccupRew* y *PropOccupRew* logran dicho ahorro energético, aunque esto supone "sacrificar" parcialmente el confort. Además, encontramos que el porcentaje de violación de confort de estas funciones de recompensa supera al caso de *LinRew*, lo que puede ir en contra de la intuición. La principal hipótesis que explicaría este comportamiento es que los modelos no hayan sido capaces de captar correctamente la ya mencionada inercia térmica y de aprender cómo tratarla (por ejemplo, activando el sistema de climatización antes de que crezca la ocupación)[2]. Todo esto sugiere que **las funciones de recompensa diseñadas pueden ser aún más refinadas** hasta alcanzar o superar totalmente al controlador basado en reglas, que sigue siendo una tarea pendiente.

Paralelamente, queda demostrado de forma empírica que el diseño de funciones de recompensa que se adapten dinámicamente a la ocupación del edificio puede influir de forma muy positiva al desempeño del agente. No obstante, un controlador basado en reglas semejante al utilizado en este proyecto, dada su naturaleza, priorizará el confort térmico y será más difícil de superar con respecto a esta métrica, por lo que aún queda pendiente cierta investigación en ese aspecto.

Por otro lado, la experimentación adicional que se propuso ha permitido extraer algunas conclusiones de interés. Además de las ideas ya mencionadas, referentes a los espacios de acciones, se ha evidenciado que **los métodos *on-policy* pueden superar en ciertas ocasiones[3] a los métodos *off-policy***, presentando la ventaja de necesitar un tiempo de entrenamiento significativamente menor. Este aspecto es importante ya que contradice la intuición de que un algoritmo *on-policy* suele tener peor rendimiento, por lo que los investigadores y programadores pueden tender a descartarlos.

Una aportación importante de este trabajo ha sido el análisis del estado del arte, que ha permitido identificar uno de los principales problemas existentes en la literatura. Se trata de la **falta de uniformidad y estandarización** presente en los modelos desarrollados por los distintos autores. Y es que, al no utilizar estándares comunes, la adecuada comparación entre distintas soluciones resulta imposible. Ante dicho problema, se propone el uso de **Sinergym** que, como ya se ha visto, consiste en una herramienta en desarrollo que tiene entre sus objetivos servir como *framework* universal

[1] Realmente, no es que el confort pierda importancia en las funciones de recompensa dinámicas sino que, al no ser tenido en cuenta cuando no hay ocupación o tener una dependencia proporcional, hay tramos horarios donde afecta de forma menos significativa.

[2] Nótese que para contrastar esta hipótesis sería necesario un estudio de mayor profundidad, que escapa del alcance de este trabajo.

[3] Especialmente en el caso de espacios de acciones discretos.

para los proyectos de DRL aplicado al control energético de edificios.

Por último, otra contribución a la comunidad de este proyecto es el desglose detallado de resultados que se presenta en el capítulo 6. Se ha tratado de facilitar al resto de investigadores los resultados obtenidos en cada configuración con distintas métricas para describir qué modelos funcionan mejor y peor en las distintas circunstancias. Además, los modelos confeccionados son públicos con el fin de poder ser aprovechados y optimizados en un futuro.

7.3. Posibles trabajos futuros

A lo largo de este proyecto, se han indicado algunos posibles estudios adicionales. Aunque se hayan trabajado y analizado distintos aspectos, también se han destapado algunas líneas de investigación secundarias pero igualmente interesantes. En este apartado se enumeran las propuestas de posibles estudios adicionales que se pueden plantear a partir de este proyecto. Se listan a continuación:

- **Diseño, implementación y evaluación de nuevas funciones de recompensa** que varíen dinámicamente en función de la ocupación, así como el refinamiento de las aquí propuestas, con el objetivo de que estas capten y traten correctamente la inercia térmica del edificio. Se pueden considerar aspectos alternativos a la ocupación.

- **Aplicación de *curriculum learning*** en la confección de los modelos. Se aplicaría esta técnica durante el entrenamiento de los agentes y se estudiaría si la robustez de estos aumenta al evaluarlos en distintos entornos.

- Considerando la propuesta anterior, surge cierto interés en la **realización de un *benchmark*** basado en Sinergym, o algún entorno alternativo, que permita la evaluación y comparación de distintos agentes.

- **Diseño e inclusión de más edificios de distintos tipos**, como infraestructuras industriales o centros comerciales, cuyo gasto energético más elevado que en edificios residenciales. La **experimentación en entornos con restricciones duras** (como el *data center* recientemente integrado en Sinergym) sería de especial interés.

- **Ampliación de los climas** permitidos en el *framework* de Sinergym. **Estudio de las variables más determinantes** en el ajuste de *setpoints* y su agregación en caso de no ser consideradas.

- **Integración y experimentación con algoritmos no implementados en *Stable Baselines 3***. Ahora mismo, el entorno está limitado

a los métodos que ofrece esta librería. Se propone la integración con otras librerías como *KerasRL*, *Tensorforce* o *MushroomRL*, entre otras; incluso la implementación directa de algoritmos adicionales por parte del programador.

- **Extensión de las variables de salida**. El problema planteado ahora mismo se limita al ajuste de *setpoints*. Dada la creciente popularidad del ámbito de la domótica, se propone la agregación de entornos que permitan modificar otros aspectos (apertura y cierre de ventanas, modificaciones en el sistema de ventilación...) y la experimentación correspondiente.

- **Implementación de una interfaz gráfica para ejecución de baterías de experimentos en Sinergym**, en aras de facilitar al usuario esta tarea, permitiendo utilizar una interfaz más amigable con una mayor abstracción del *software* subyacente.

7.4. Valoración personal

Este proyecto de fin de grado ha supuesto una excelente oportunidad para conocer una disciplina tan interesante como es el aprendizaje por refuerzo al que, por desgracia, no se dedica ninguna asignatura en la titulación. Por su parte, la posibilidad de contribuir a la lucha contra el cambio climático motivó aún más mi interés por el desarrollo de un TFG de esta índole.

Personalmente, diría que la realización de este trabajo ha sido una experiencia muy constructiva y enriquecedora. Me ha permitido poner en práctica y afianzar muchos conocimientos adquiridos durante el grado que, por falta de tiempo, no se desarrollan lo suficiente. Hablo del uso de herramientas muy utilizadas en el mundo laboral, como GitHub, Google Cloud, Docker, el uso de entornos y contenedores remotos, librerías y herramientas de *machine learning* como SB3, TensorFlow, MLFlow, y un largo etcétera. Además, he podido experimentar la integración a un equipo de trabajo profesional encargado del desarrollo de un proyecto real, pudiendo conocer así la organización y forma de trabajar desde un punto de vista práctico. Por otro lado, este primer acercamiento al mundo de la investigación ha despertado bastante curiosidad en mí, permitiéndome conocer más de cerca una posibilidad muy interesante sobre mi proyección profesional.

Por último, considero que el trabajo realizado puede resultar de gran interés y utilidad para la comunidad científica, demostrando que el aprendizaje por refuerzo conforma una buena herramienta para la optimización del control energético. El hecho de ser un campo relativamente inmaduro desata el interés por explotar el potencial que sugiere tener. Trabajos de este tipo

y proyectos como Sinergym tienen como objetivo la unificación y estandarización de los diferentes estudios en esta materia, tratando de posibilitar la combinación de distintas investigaciones e impulsar así el progreso y desarrollo de futuras soluciones. Asimismo, el control energético inteligente en edificios es un problema pendiente de resolver, en el que probablemente cada vez más investigadores vuelquen sus esfuerzos. No obstante, las soluciones desarrolladas avanzan a grandes pasos, y su aplicación en el mundo real será cada vez mayor.

Bibliografía

[1] The Global Alliance for Buildings and Construction (GABC). The global status report. Technical report, 2020.

[2] U.S. Department of Energy. Quadrennial technology review, an assessment of energy technologies and research opportunities. 2015.

[3] Sumedha Sharma, Yan Xu, Ashu Verma, and Bijaya Panigrahi. Time-coordinated multi-energy management of smart buildings under uncertainties. *IEEE Transactions on Industrial Informatics*, 2019.

[4] Nan Zhou, Nina Khanna, Wei Feng, Jing Ke, and M. Levine. Scenarios of energy efficiency and co2 emissions reduction potential in the buildings sector in china to year 2050. *Nature Energy*, 2018.

[5] Liang Yu, Shuqi Qin, Meng Zhang, Chao Shen, Tao Jiang, and Xiaohong Guan. A review of deep reinforcement learning for smart building energy management. *IEEE Internet of Things Journal*, 8(15):12046–12063, 2021.

[6] Liang Yu, Shuqi Qin, Meng Zhang, Chao Shen, Tao Jiang, and Xiaohong Guan. A review of deep reinforcement learning for smart building energy management. *IEEE Internet of Things Journal*, 8(15):12046–12063, 2021.

[7] Peter Norvig and Stuart Russell. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2021.

[8] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[9] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3), 1959.

[10] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, 1998.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2nd edition, 2018.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[14] M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. 2008.

[15] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[16] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[17] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[19] Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, 1992.

[20] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[21] Marc Lanctot Ziyu Wang, Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2015.

[22] J. Fernando Hernandez-Garcia and Richard S. Sutton. Understanding multi-step deep reinforcement learning: A systematic study of the dqn target, 2019.

[23] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2015.

[24] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Procs. 33rd International Conference on International Conference on Machine Learning*, volume 48 of *ICML'16*, pages 1928—-1937. JMLR.org.

[25] Antonio Manjavacas Lucas. Deep reinforcement learning para control energético eficiente de edificios. Master's thesis, Universidad de Granada, 2021.

[26] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Procs. 35th International Conference on Machine Learning*, Stockholm, Sweden, 2018.

[27] S. Kullback and R.A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[28] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. 2015.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017.

[30] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. 2015.

[31] AHSRAE. *2004 Revision of ASHRAE standard thermal environmental conditions for human occupancy.* 2004.

[32] AHSRAE. *ASHRAE standard thermal environmental conditions for human occupancy.* 1992.

[33] H.E. Burroughs and Shirley J. Hansen. *Managing Indoor Air Quality.* River Publishers, 5th edition, 2011.

[34] Tianshu Wei, Yanzhi Wang, and Qi Zhu. Deep reinforcement learning for building hvac control. In *Proceedings of the 54th Annual Design Automation Conference 2017.* Association for Computing Machinery, 2017.

[35] Charles W. Anderson, Douglas C. Hittle, Alon D. Katz, and R.Matt Kretchmar. Synthesis of reinforcement learning, neural networks and pi control applied to a simulated heating coil. *Artificial Intelligence in Engineering*, 11(4):421–429, 1997.

[36] Michael C. Mozer. The neural network house: An environment that adapts to its inhabitants. In *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pages 110–114, 1998.

[37] Silvio Brandi, Marco Savino Piscitelli, Marco Martellacci, and Alfonso Capozzoli. Deep reinforcement learning to optimise indoor temperature control and heating energy consumption in buildings. *Energy and Buildings*, 224, 2020.

[38] Giuseppe Tommaso Costanzo, Sandro Iacovella, Frederik Ruelens, T. Leurs, and Bert Claessens. Experimental analysis of data-driven control for a building heating system. *Sustainable Energy, Grids and Networks*, 6:81–90, 2016.

[39] P. Fazenda, K. Veeramachaneni, P. Lima, and Una-May O'Reilly. Using reinforcement learning to optimize occupant comfort and energy usage in hvac systems. *Journal of Ambient Intelligence and Smart Environments*, 6(6):675–690, 2014.

[40] D. Urieli and P. Stone. A learning agent for heat-pump thermostat control. In *Procs. 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, volume 2, pages 1093–1100, 2013.

[41] Bocheng Li and Li Xia. A multi-grid reinforcement learning method for energy conservation and comfort of hvac in buildings. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 444–449, 2015.

[42] José Vázquez-Canteli, Jérôme Kämpf, and Zoltán Nagy. Balancing comfort and energy consumption of a heat pump using batch reinforcement learning with fitted q-iteration. *Energy Procedia*, 122:415–420, 2017. CISBAT 2017 International ConferenceFuture Buildings & Districts – Energy Efficiency from Nano to Urban Scale.

[43] Frederik Ruelens, Sandro Iacovella, Bert J. Claessens, and Ronnie Belmans. Learning agent for a heat-pump thermostat with a set-back strategy using model-free reinforcement learning. *Energies*, 8(8):8300–8318, 2015.

[44] Biao Sun, Peter B. Luh, Qing-Shan Jia, and Bing Yan. Event-based optimization within the lagrangian relaxation framework for energy savings in hvac systems. *IEEE Transactions on Automation Science and Engineering*, 12(4):1396–1406, 2015.

[45] Simeng Liu and Gregor P. Henze. Evaluation of Reinforcement Learning for Optimal Control of Building Active and Passive Thermal Storage Inventory. *Journal of Solar Energy Engineering*, 129(2):215–225, 10 2006.

[46] Simeng Liu and Gregor P. Henze. Experimental analysis of simulated reinforcement learning control for active and passive building thermal storage inventory: Part 2: Results and analysis. *Energy and Buildings*, 38(2):148–161, 2006.

[47] José R. Vázquez-Canteli, Stepan Ulyanin, Jérôme Kämpf, and Zoltán Nagy. Fusing tensorflow with building energy simulation for intelligent energy management in smart cities. *Sustainable Cities and Society*, 45:243–257, 2019.

[48] Enda Barrett and Stephen Paul Linder. Autonomous hvac control, a reinforcement learning approach. volume 9286, 09 2015.

[49] Zhiang Zhang, Adrian Chong, Yuqi Pan, Chenlu Zhang, and Khee Poh Lam. Whole building energy model for hvac optimal control: A practical framework based on deep reinforcement learning. *Energy and Buildings*, 199:472–490, 2019.

[50] Zhen Yu and Arthur Dexter. Online tuning of a supervisory fuzzy controller for low-energy building system using reinforcement learning. *Control Engineering Practice*, 18(5):532–539, 2010.

[51] K. Dalamagkidis, Denia Kolokotsa, Kostas Kalaitzakis, and G. Stavrakakis. Reinforcement learning for energy conservation and comfort in buildings. *Building and Environment*, 42:2686–2698, 07 2006.

[52] José R. Vázquez-Canteli and Zoltán Nagy. Reinforcement learning for demand response: A review of algorithms and modeling techniques. *Applied Energy*, 235:1072–1089, 2019.

[53] Dajun Du and Minrui Fei. A two-layer networked learning control system using actor–critic neural network. *Applied Mathematics and Computation*, 205:26–36, 11 2008.

[54] Donald Azuatalam, Wee-Lih Lee, Frits de Nijs, and Ariel Liebman. Reinforcement learning for whole-building hvac control and demand response. *Energy and AI*, 2, 2020.

[55] Xiaolei Yuan, Yiqun Pan, Jianrong Yang, Weitong Wang, and Zhizhong Huang. Study on the application of reinforcement learning in the operation optimization of hvac system. *Building Simulation*, 14, 12 2019.

[56] Zhiang Zhang and Khee Poh Lam. Practical implementation and evaluation of deep reinforcement learning control for a radiant heating system. In *Proceedings of the 5th Conference on Systems for Built Environments*, page 148–157. Association for Computing Machinery, 2018.

[57] Liang Yu, Yi Sun, Zhanbo Xu, Chao Shen, Dong Yue, Tao Jiang, and Xiaohong Guan. Multi-agent deep reinforcement learning for hvac control in commercial buildings. *IEEE Transactions on Smart Grid*, PP:1–1, 2020.

[58] Yan Du, Helia Zandi, Olivera Kotevska, Kuldeep Kurte, Jeffery Munk, Kadir Amasyali, Evan Mckee, and Fangxing Li. Intelligent multi-zone residential hvac control strategy based on deep reinforcement learning. *Applied Energy*, 281:116117, 2021.

[59] Anchal Gupta, Youakim Badr, Ashkan Negahban, and Robin G. Qiu. Energy-efficient heating control for smart buildings with deep reinforcement learning. *Journal of Building Engineering*, 34:101739, 2021.

[60] Javier Jiménez-Raboso, Alejandro Campoy-Nieves, Antonio Manjavacas-Lucas, Juan Gómez-Romero, and Miguel Molina-Solana. Sinergym: A building simulation and control framework for training reinforcement learning agents. In *Proceedings of the 8th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pages 319—-323, New York, NY, USA, 2021. Association for Computing Machinery.

[61] Alistair Cockburn. *Agile Software Development*. Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[62] David Wölfle, Arun Vishwanath, and Hartmut Schmeck. A guide for the design of benchmark environments for building energy optimization. In *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, BuildSys '20, pages 220—-229. Association for Computing Machinery, 2020.

[63] Ali Ghahramani, Kenan Zhang, Kanu Dutta, Zheng Yang, and Burcin Becerik-Gerber. Energy savings from temperature setpoints and deadband: Quantifying the influence of building and system properties on savings. *Applied Energy*, 165:930–942, 2016.

[64] Florida Solar Energy Center. Determining Appropriate Heating and Cooling Thermostat Set Points for Building Energy Simulations for Residential Buildings in North America. Technical Report FSEC-CR-2010-13, FSEC Energy Research Center, 2013.